D3.1v0.1 WSMO Primer

WSMO Final Draft 01 April 2005

This version:
http://www.wsmo.org/TR/d3/d3.1/v0.1/20050401/
Latest version:
http://www.wsmo.org/TR/d3/d3.1/v0.1/
Previous version:
http://www.wsmo.org/TR/d3/d3.1/v0.1/20050323/
Editors:
Cristina Feier
Authors:
Cristina Feier
John Domingue
Reviewer:
Jos de Bruijn
John Domingue
Dieter Fensel
Holger Lausen
Axel Polleres

Abstract

The Web Service Modeling Ontology (WSMO) is an ontology for describing various aspects related to Semantic Web services. This primer is designed to provide the reader with the fundamental knowledge required to use WSMO effectively. It presents the basic concepts of WSMO with respect to the Version 1.1 of the conceptual specification. It describes how to model ontologies, Web services, goals and mediators using the language devised for formalizing WSMO, the Web Services Modeling Language (WSML). Finally, it describes the content and purpose of the other documents in the specification.

Table of contents

---

1. Introduction

The Web Service Modeling Ontology (WSMO) is a conceptual model for describing various aspects related to Semantic Web services. The objective of WSMO and its accompanying efforts is to solve the application integration problem for Web services by defining a coherent technology for Semantic Web services. WSMO takes the Web Service Modeling Framework (WSMF) [Fensel & Bussler, 2002] as starting point and further refines and extends its concepts. It is developed by the WSMO working group.

In order to achieve automated discovery, composition, and execution of Web services a conceptual model alone is insufficient. In addition, a formal language is required to write down annotations of Web services according to the conceptual model. Logical inference-mechanisms can be applied to the descriptions in this language. In order to allow appropriate, satisfactorily logic-based reasoning on Semantic Web services, the description language has to provide reasonable expressivity, together with well-defined formal semantics. The Web Services Modeling Language (WSML) is a family of languages which formalizes WSMO. A related goal of WSML is to provide a rule-based language for the Semantic Web. It is developed by the WSML working group, which is a subgroup of the WSMO working group. For a specification of the different WSML variants you can consult [de Bruijn et al., 2005].

Another working group related to WSMO is dedicated to the Web Service Execution Environment (the WSMX working group). The goal of this working group is the development of an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of Semantic Web services based on the WSMO specification. WSMX shall provide a reference architecture and

implementation for this purpose.

The aim of this primer is to provide an introduction to WSMO and to describe it in sufficient detail to enable the reader to model goals, mediators and Web services. Further, it exemplifies the use and modeling of domain ontologies, which comprise the fourth pillar of WSMO. The target audience are researchers, software engineers and developers, and all other persons interested in and dealing with Semantic Web services. It is the primer's intention to answer questions such as:

- What is WSMO in a nutshell?
- How are Semantic Web services modeled using WSMO?
- How is a WSMO component described?

The primer is based on the work carried on [Roman et al., 2005], which presents the core elements of the Web Service Modeling Ontology and their interrelations. All the examples provided throughout the primer use WSML. It is not the intention of the primer to provide a complete specification of WSMO: further details can be found by in the documents which are listed in Section 6.

The WSMO Primer is organized as follows: Section 2 introduces the basic concepts and model behind WSMO, together with its principles; Section 3 describes the use case that guides the primer; Section 4 provides an overview of the main elements of WSMO. Some conclusions regarding WSMO usability are drawn in Section 5. Finally, Section 6 presents the WSMO specification. The WSML definitions of the entities used for describing the chosen use case, which are spread throughout the whole document in order to exemplify each entity in turn, are gathered as a unified example in a separate Appendix.

2. Overview

WSMO provides means to describe all relevant aspects of Semantic Web services in a unified manner. Here, we particularly consider all aspects concerning the problems arising in Enterprise Application Integration as relevant. This section describes the approach and basic ideas behind WSMO in order to tackle these problems.

2.1 WSMO Design Principles

WSMO provides ontological specifications for the core elements of Semantic Web services. In fact, Semantic Web services aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services, thereby turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing. Therefore, appropriate frameworks for Semantic Web services need to integrate the basic Web design principles, those defined for the Semantic Web, as well as design principles for distributed, service-orientated computing of the Web. WSMO is therefore based on the following design principles:

- **Web Compliance** - WSMO inherits the concept of URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the Word Wide Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, and supports XMLand other W3C Web technology recommendations, as well as the

decentralization of resources.

- **Ontology-Based** - Ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the central enabling technology for the Semantic Web. The extensive usage of ontologies allows semantically enhanced information processing as well as support for interoperability; WSMO also supports the ontology languages defined for the Semantic Web.
- **Strict Decoupling** - Decoupling denotes that WSMO resources are defined in isolation, meaning that each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
- **Centrality of Mediation** - As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data or process. WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.
- **Ontological Role Separation** - User requests are formulated independently of (in a different context than) the available Web services. The underlying epistemology of WSMO differentiates between the desires of users or clients and available Web services.
- **Description versus Implementation** - WSMO differentiates between the descriptions of Semantic Web services elements (description) and executable technologies (implementation). While the former requires a concise and sound description framework based on appropriate formalisms in order to provide a concise expression for semantic descriptions, the latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services. WSMO aims at providing an appropriate ontological description model, and to be compliant with existing and emerging technologies.
- **Execution Semantics** - In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.
- **Service versus Web service** - A Web service is a computational entity which is able (by invocation) to achieve a goal. A service in contrast is the actual value provided by this invocation [Baida et al., 2004], [Preist, 2004]. Thus, WSMO does not specify services, but Web services which are actually means to buy and search services.

2.2 Basic Concepts and Model

WSMO defines the modeling elements for describing several aspects of Semantic Web services based on the conceptual grounding set up in the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002], wherein four main components are defined:

- **Ontologies**: provide the formal semantics to the information used by all other components.
- **Goals:** specify objectives that a client might have when consulting a Web service.
- **Web services:** represent the functional (and behavioral) aspects which must be semantically described in order to allow semi-automated use.

- **Mediators**: used as connectors, they provide interoperability facilities among the other elements.

WSMO inherits these four top elements, further refining and extending them. Below there is a more detailed description of each of these top-level components of WSMO.

Ontologies represent a key element in WSMO because they provide (domain specific) terminologies for describing the other elements. They serve a twofold purpose: first, defining the formal semantics of the information, and second, linking machine and human terminologies. WSMO specifies the following constituents as part of the description of an ontology: *concepts*, *relations*, *functions*, *axioms*, and *instances* of concepts and relations, as well as *non-functional properties*, *imported ontologies*, and *used mediators*. The latter allows the interconnection of different ontologies by using mediators that solve terminology mismatches. A more detailed description of the structure of an ontology together with an example of such a description is given in Section 4.1.

Web services connect computers and devices with each other using the standard Web-based protocols to exchange data and combine data in new ways. Their distribution over the Web confers on them the advantage of platform independence. Each Web service represents an atomic piece of functionality that can be reused to build more complex ones. In order to allow their discovery, invocation, composition, execution, monitoring, mediation, and compensation, Web services are described in WSMO from three different points of view: non-functional properties, functionality and behavior. Thus, a Web service is defined by *its non-functional properties*, *its imported ontologies*, *its used mediators*, *its capability* and *its interfaces*. A Web service can be described by multiple interfaces, but has one and only one capability. The capability of a Web service encapsulates its functionality and an interface of a Web Service describes the behavior of the Web Service from two perspectives: communication and collaboration. For more detailed information about how to model Web services within WSMO please refer to Section 4.2.

Goals describe aspects related to user desires with respect to the requested functionality as opposed to the provided functionality described in the Web service capability. In WSMO a goal is characterized by a set of *non-functional properties*, *imported ontologies*, *used mediators*, the *requested capability* and the *requested interface*. Goals are described in Section 4.3.

Mediators describe elements that aim to overcome structural, semantic or conceptual mismatches that appear between the different components that build up a WSMO description. Currently the specification covers four different types of mediators:

- *OOMediators* - import the target ontology into the source ontology by resolving all the representation mismatches between the source and the target;
- *GGMediators* - connect goals that are in a relation of refinement and resolve mismatches between those;
- *WGMediators* - link Web services to goals and resolve mismatches;
- *WWMediators* - connect several Web services for collaboration.

More details about the particular use of mediators and the role they play in WSMO are provided in Section 4.4.

3. Use case description

In order to introduce the basic concepts and model, a use case from the domain of e-tourism is used throughout the Primer. In this use case, an agent wants to buy a ticket to travel from Innsbruck (Austria) to Venice (Italy) on a certain date. The goal that specifies the intent of buying a ticket for a trip from Innsbruck to Venice is abstracted from this agent desire. A hypothetical Book Ticket Web service, offered by a "Virtual Travel Agency "(VTA), is considered for achieving the goal. This Web service allows the service requester to search and buy tickets for itineraries starting in Austria. The only accepted payment method is credit card. For the execution of the transaction, the credit card must be a valid PlasticBuy or GoldCard (two fictitious credit card brands).

4. WSMO Modeling Elements

This section explains the WSMO modeling elements, describing their purpose and illustrating how they can be specified. This section is divided into four subsections: Section 4.1 describes ontologies and all the essential components used to define them, Section 4.2 presents Web services, Section 4.3 presents goals and how they are used to specify the functionality required from Web services, and finally Section 4.4 exemplifies the use of mediators which provide the means to bypass interoperability problems among the rest of the elements. The examples presented throughout this section are based on the previously presented use case.

4.1 Ontologies

A widespread definition of ontology is given in [Gruber, 1993] where an ontology is defined as "a formal explicit specification of a shared conceptualization" . In WSMO, ontologies provide machine-readable semantics for the information used by all actors implied in the process of Web services usage, either providers or requesters, allowing interoperability and information interchange among components. As an example, we present the definition of an ontology used for specifying the "Book Ticket Web Service" described in the use case and for defining a goal that can be solved by this Web service. This ontology is called "Trip Reservation Ontology", since it defines the necessary terminology for describing trip and reservation related information.

A namespace declaration can appear at the beginning of each WSML file. Such a declaration may comprise the *default namespace* and abbreviations for *other used namespaces*. Listing 1 starts with the namespace declaration at the top of the WSML file where the "Trip Reservation Ontology" is specified.

Any ontology declaration starts with the keyword `ontology` optionally followed by the ontology identifier. A set of non-functional properties may follow this statement. Non-functional properties can be specified for almost any WSMO element and they describe information that does not affect the functionality of the element like title, authorship, copyrights, etc. For each different WSMO element, there is a recommended set of non-functional properties, but this set is extensible. The "Trip Reservation Ontology" is characterized by the non-functional properties that appear in Listing 1.

Note that, the majority of the recommended non-functional properties are properties defined in the Dublin Core Metadata Element Set. However, their values have types

according to the WSMO recommendation:

- the property `dc#title` has as value the name of the element, in this case the "Trip Reservation Ontology" .
- the property `dc#identifier` has been defined following the WSMO recommendation to use an IRI for identifying the element. In the case of the "Trip Reservation Ontology", the identifier was already declared, but here is repeated in order to allow Dublin Core metadata aware applications to process this information.
- the property `dc#creator` has been defined following the WSMO recommendation to use an instance of the class `foaf#agent` defined in the FOAF ontology [FOAF] to indicate unambigously the entity responsible for creating the content of the element.
- the property `dc#description` has been defined following the WSMO recommendation, as a free-text description of the content of the element.
- the property `dc#publisher` has been defined following the WSMO recommendation to use an instance of the class `foaf#agent` defined in the FOAF ontology [FOAF] to indicate unambigously the entity responsible for publishing the content of the element.
- the property `dc#contributor` has as value an instance of the class `foaf#agent` defined in the FOAF ontology [FOAF] in order to indicate unambigously the entity responsible for making contributions to the content of the element, as WSMO recommends.
- the property `dc#date` has been defined following the WSMO recommendation to encode date values as defined in the ISO standard 8601:2000 [ISO8601].
- the property `dc#format` has been defined following the WSMO recommendation to use types defined in the list of Internet Media Types by the Internet Assigned Numbers Authority.
- the property `dc#language` has as value a language tag defined in the ISO standard 639 [ISO639].
- the property `dc#rights` reference a document that contains a rights management statement for the element.

The only non-functional property declared for this ontology that is not included in the Dublin Core Metadata Element Set is `wsml#version`, its value being a CVS revision tag, as recommended by WSMO.

Listing 1. Trip Reservation Ontology Header

```
namespace {_"http://example.org/tripReservationOntology#",
  dc    _"http://purl.org/dc/elements/1.1#",
  loc   _"http://example.org/locationOntology#",
  po    _"http://example.org/purchaseOntology#",
  foaf  _"http://xmlns.com/foaf/0.1/",
  wsml  _"http://www.wsmo.org/wsml/wsml-syntax#",
  prs   _"http://example.org/owlPersonMediator#"}

ontology  _"http://example.org/tripReservationOntology"
  nonFunctionalProperties
    dc#title hasValue "Trip Reservation Ontology"
    dc#identifier hasValue _"http://example.org/tripReservationOntology"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#description hasValue
      "an ontology for describing trip reservations related knowledge"
    dc#publisher hasValue _"http://example.org/foaf#deri"
    dc#contributor hasValue  _"http://example.org/foaf#cristina"
    dc#date hasValue _date(2004,12,16)
```

```
    dc#format hasValue "text/x-wsml"
    dc#language hasValue "en-us"
    dc#rights hasValue _"http://deri.at/privacy.html"
    wsml#version hasValue "$Revision 1.17 $"
  endNonFunctionalProperties

  importsOntology  { _"http://example.org/locationOntology",
              _"http://example.org/purchaseOntology"}

  usesMediator _"http://example.org/owlPersonMediator"
```

Ontologies are applied in every other WSMO element as domain specific vocabulary definitions. WSMO allows the importing of other ontologies into an ontology either directly, when no conflicts need to be resolved, or indirectly, by using mediation, which is the process of resolving any conflicts that may appear through aligning, merging or transforming imported ontologies. In the first case the imported ontologies are specified by using the keyword `importsOntology`, while in the second case the `usesMediator` statement points to the mediators, the components that realize the mediation. Mediation may be done not only between ontologies, but also between other components involved in the Web service descriptions, the mediators that realize the mediation between ontologies being named *ooMediators*. More about the possible types of mediators can be found in Section 4.4. In the "Trip Reservation Ontology", two ontologies are imported without any mediation and one mediator is used for importing one OWL [OWLSemantics] ontology.

The basic blocks of an ontology are **concepts**, **relations**, **functions, instances**, and **axioms.** The examples provided in the next subsections for each of these elements are taken mainly from the "Trip Reservation Ontology".

4.1.1 Concepts

Concepts are defined by their subsumption hierarchy and their attributes, including range specification. The range of the attributes can be a datatype or another concept. There are two kinds of attribute definitions: constraining definitions, using the keyword `ofType` and inferring definitions using the keyword `impliesType`. In the first case, the values of the attribute are constrained to having the mentioned type, while in the latter, the values of the attribute are inferred to have the mentioned type. For specifying that an attribute can have any type we can declare its type as being `wsml#true`. Like for other WSMO entities, a set of non-functional properties can be used for specifying additional information about concepts. Below the definitions of some concepts from the "Trip Reservation Ontology" are presented, that will be further used for defining other elements of WSMO.

Listing 2. Concept definitions

```
concept trip
  origin impliesType loc#location
  destination impliesType loc#location
  departure ofType _date
  arrival ofType _date

concept tripFromAustria subConceptOf trip
  nonFunctionalProperties
    dc#relation hasValue tripFromAustriaDef
  endNonFunctionalProperties
```

```
axiom tripFromAustriaDef
  definedBy
      forall {?x ,?origin}
            (?x memberOf tripFromAustria
             implies
             ?x[
                origin hasValue ?origin] and
             ?origin[
                loc#locatedIn hasValue loc#austria]
            ).

concept ticket
  provider ofType _string
  trip ofType trip
  recordLocatorNumber ofType _integer

concept reservationRequest
  nonFunctionalProperties
    dc#description hasValue "This concept represents a
    reservation request for some trip for a particular person"
  endNonFunctionalProperties
  reservationItem impliesType wsml#true
  reservationHolder impliesType prs#person

concept reservation
  nonFunctionalProperties
     dc#description hasValue "concept of a confirmation for some item"
  endNonFunctionalProperties
  reservationItem impliesType wsml#true
  reservationHolder impliesType prs#person
```

For expressing subsumption relations between concepts the keyword `subConceptOf` is used. In the example above, the concept `tripFromAustria` is subsumed by the concept `trip`.

The extension of a concept can be defined or restricted by one or more logical expressions that specify a necessary and/or a sufficient condition for an instance to be an element of the extension of the concept. Such logical expressions are embedded in axioms. It is recommended that the concept refers to its related axioms by declaring their identifiers as values for the non-functional property `dc#relation`. The definition of the concept `tripFromAustria` is completed with an axiom, `tripFromAustriaDef`, that specifies that a necessary and sufficient condition for an individual to be an instance of this concept is to have as the values of its `origin` attribute the location Austria.

4.1.2 Relations

Relations describe interdependencies between a set of parameters. Optionally, the domain of parameters which can be a datatype or a certain concept can be specified using the `impliesType` or `ofType` keywords. Thus, a relation declaration comprises the keyword `relation`, the identifier of the relation and optionally: its arity, its superrelations, the domain of its parameters, and a set of non-functional properties. Axioms can be used for defining/constraining the relation extension. If there are such axioms, it is recommended to refer to them in the `dc#relation` non-functional property.

Below is a relation definition taken from a different ontology, the "Purchase Ontology", that has a single argument, a credit card, and holds when the credit card

is valid. Accompanying this relation is an axiom that compares the credit card expiry date with the current date for establishing its validity. For completeness, the definition of the concept `creditCard` is also provided, since it is used in the function definition. Note that this listing makes part from a different WSML file (the file in which the "Purchase Ontology" is defined) than the file to which the other examples presented in this section belong (the file in which the "Trip Reservation Ontology" is defined).

Listing 3. The definition of the relation `validCreditCard`

```
namespace { _"http://example.org/purchaseOntology#",
  dc          _"http://purl.org/dc/elements/1.1",
  wsml        _"http://www.wsmo.org/wsml/wsml-syntax#",
  prs         _"http://example.org/owlPersonMediator.wsml#"}

ontology  _"http://example.org/purchaseOntology"

concept creditCard
   owner impliesType prs#person
   number ofType _integer
   type ofType _string
   expiryDate ofType _date
   balance ofType _integer


relation validCreditCard(ofType creditCard)
 nonFunctionalProperties
    dc#description hasValue "Relation that holds for a valid credit card"
    dc#relation hasValue ValidCreditCardDef
 endNonFunctionalProperties


axiom ValidCreditCardDef
definedBy
  forall {?x, ?y}  (
      validCreditCard(?x)  impliedBy
      ?x[expiryDate hasValue ?y] memberOf creditCard
      and
      neg (wsml#dateLessThan(?y, wsml#currentDate()))).
```

4.1.3 Functions

Functions are a special type of relations that have a unary range beside the set of parameters. Although in the conceptual model, they are regarded as entities distinct from relations, in WSML they are modeled as a special type of relations that have one functional parameter. Besides the typical declaration for a relation, when declaring a function one must include an axiom that states the functional dependency. The function `ticketPrice` is modeled as a relation with three parameters: the first one is a ticket, the second one is a currency, while the third one is the result returned by the function: the price of the ticket in the given currency.

Listing 4. Modeling the function `tripPrice` as a relation

```
relation ticketPrice(ofType ticket, ofType po#currency, ofType _integer)
 nonFunctionalProperties
   dc#description hasValue
      "Function that returns the price of a ticket in a given currency"
   dc#relation hasValue {FunctionalDependencyTripPrice}
 endNonFunctionalProperties
```

```
axiom FunctionalDependencyTicketPrice
    definedBy
      !- ticketPrice(?x,?y,?z1) and ticketPrice(?x,?y,?z2) and ?z1 != ?z2 .
```

4.1.4 Instances

Instances are embodiments of concepts or relations, being defined either explicitly by specifying concrete values for attributes or parameters or by a link to an instance store. In the listing below three instances are presented, the first two being instances of the concepts `trip`, respectively `ticket`, presented in Listing 2, and the third one being an instance of the function (relation) `ticketPrice` introduced in Listing 4. Instances explicitly specified in an ontology are those that are shared together with the ontology. This may be the case for the first instance, `tripInnVen`. However, most instance data exists outside the ontology in private data stores. These private data stores are called instance stores and they are linked to the ontology. We expect the second and the third instance which represent the coordinates of a ticket for a trip from Innsbruck to Venice, respectively the price for that given ticket instance, to reside in such an instance store, since they are specific to the Web service provider. For simplicity, here we consider them to be declared in the same file as the "Trip Reservation Ontology". We will come back to this issue in a later section.

Listing 5. Instance definitions

```
instance tripInnVen memberOf trip
  origin hasValue loc#innsbruck
  destination hasValue loc#venice
  departure hasValue  _date(2005,11,22)
  arrival hasValue  _date(2005,11,22)

instance ticketInnVen memberOf ticket
  provider hasValue "Book Ticket Service"
  trip hasValue tripInnVen
  recordLocatorNumber hasValue 93

relationInstance ticketPrice(ticketInnVen, po#euro, 120)
```

4.1.5 Axioms

Axioms are specified as logic expressions and help to formalize domain specific knowledge. Different WSML variants allow different expressivities for the logical expressions. For more detail about that, see the WSML specification. As already described in the previous sections, they are declared by using the keyword `axiom` optionally followed by the axiom identifier, by a set of non-functional properties and by the logical expression introduced by the keyword `definedBy`.

4.2 Web Services

A Web service description in WSMO consists of five sub-components: *non-functional properties*, *imported ontologies*, *used mediators*, *a capability* and *interfaces*.

In this section we illustrate the declaration of a Web service, by defining the "Book Ticket Web Service" offered by the "Virtual Travel Agency" described in Section 3.

The declaration of a Web service starts with the keyword `webService` followed by the identifier of the Web service. After this, comes the declaration of the non-functional properties of the Web service. A Web service can import ontologies either directly, by using the `importsOntology` statement, or via *oomediators* declared in the `usesMediator` statement. The "Book Ticket Web Service" imports the "Trip Reservation Ontology". It must be noted that the Web service has its own instance store linked to this ontology in which there are instances specific to the Web service like the available tickets and the prices of those tickets. Below is the definition of this Web service.

Listing 6. Definition of the "Book Ticket Web Service "

```
namespace {_"http://example.org/bookTicket#",
  dc      _"http://purl.org/dc/elements/1.1#",
  tr      _"http://example.org/tripReservationOntology#",
  foaf    _"http://xmlns.com/foaf/0.1/",
  wsml    _"http://www.wsmo.org/wsml/wsml-syntax#",
  bti     _"http://www.example.org/BookTicketInterfaceOntology#"}

webService _"http://example.org/bookTicketWebService"

importsOntology _"http://example.org/tripReservationOntology"

capability BookTicketCapability
interface BookTicketInterface
```

4.2.1 Web Service Capability

The *capability* of a Web service defines its functionality in terms of pre and postconditions, assumptions and effects. A Web service defines one and only one capability. A Web service capability is defined by specifying the next elements: *non-functional properties*, *imported ontologies*, *used mediators*, *shared variables, precondition*, *postcondition*, *assumption*, and *effect*.

Ontologies may be imported either directly using the keyword `importsOntology` or via *ooMediators* declared in the `usesMediator` section.

A capability, therefore a Web service, may be linked to certain goals that are solved by the Web service via special types of mediators, named *wgMediators*. These links are useful in the Web Service Discovery phase. *WGMediators* resolve also terminology mismatches between Web services and goals. Such mediators are specified in the `usesMediator` part of a Web service capability definition.

For declaring the basic blocks of the "Book Ticket Web Service" capability, we will take a closer look at what the Web service offers to a client (*postcondition*), when some conditions are met in the information space (*precondition*), and how the execution of the Web service changes the world (*effect*), given that some conditions over the world state are met before execution (*assumption*). Preconditions, assumptions, postconditions and effects are expressed through a set of axioms. A set of shared variables can be declared within the capability. These variables are implicitly all-quantified and their scope is the whole Web service capability. Thus, in informal language, the logical interpretation of a Web service capability is: for any values taken by the shared variables, the conjunction of the precondition and of the assumption implies the conjunction of the postcondition and of the effect.

When its execution is successful, the "Book Ticket Web Service" has as result a

reservation that includes the reservation holder and a ticket for the desired trip (*postcondition*) if there is a reservation request for a trip with its starting point in Austria for a certain person (*precondition*) and if the credit card intended to be used for paying is a valid one and its type is either PlasticBuy or GoldenCard (*assumption*). As a consequence of the execution of the Web service, the price of the ticket will be deducted from the credit card (*effect*). The shared variables for the "Book Ticket Web Service" are those that designate the desired trip (the variable `?trip`), the reservation holder (the variable `?reservationHolder`), the credit card information (the variable `?creditCard`), the initial balance of the credit card (the variable `?initialBalance`) and the provided ticket (the variable `?ticket`). The precondition checks whether the user request is about a trip whose origin is in Austria. This is done by checking the membership of the trip value from the user request to the concept `tripFromAustria` defined in the "Trip Reservation Ontology". The assumption uses the relation `validCreditCard` defined in the "Purchase Ontology" to see whether the credit card is valid. The postcondition states that the result of the trip is a reservation that includes a ticket for the desired trip using the special symbol `result`. The effect updates the value of the credit card balance by subtracting the price of the ticket. The price of the ticket is kept in the Web service instance store as an instance of the function (relation) `ticketPrice`.

Listing 7. Definition of the "Book Ticket Web Service" capability

```
capability BookTicketCapability

 sharedVariables {?creditCard, ?initialBalance, ?trip, ?reservationHolder, ?ticket}

 precondition
    nonFunctionalProperties
        dc#description hasValue "The information of a trip which starts in Austria
         together with the information about the person who wants to have a reservation
         must be given as a part of a reservation request. A credit card is also required."
    endNonFunctionalProperties
    definedBy
       ?reservationRequest[
          reservationItem hasValue ?trip,
          reservationHolder hasValue ?reservationHolder
       ] memberOf tr#reservationRequest
       and
       ?trip memberOf tr#tripFromAustria and
       ?creditCard[
          balance hasValue ?initialBalance
       ] memberOf po#creditCard.

 assumption
    nonFunctionalProperties
       dc#description hasValue "The credit card information provided by the
        requester must designate a valid credit card that should be either
        PlasticBuy or GoldenCard."
    endNonFunctionalProperties
    definedBy
       po#validCreditCard(?creditCard)
       and ( ?creditCard[
                 type hasValue "PlasticBuy"]
             or
             ?creditCard[
                 type hasValue "GoldenCard"]
           ).

 postcondition
```

```
    nonFunctionalProperties
       dc#description hasValue "A reservation containing the details of a ticket for
        the desired trip and the reservation holder is the result of the successful
        execution of the Web service."
    endNonFunctionalProperties
    definedBy
      ?reservation memberOf tr#reservation[
        reservationItem hasValue ?ticket,
        reservationHolder hasValue ?reservationHolder
      ]
      and
      ?ticket[
        trip hasValue ?trip
      ]
      memberOf tr#ticket.

  effect
    nonFunctionalProperties
       dc#description hasValue "The credit card  will be charged with the cost of
        the ticket."
    endNonFunctionalProperties
    definedBy
       ticketPrice(?ticket, "euro", ?ticketPrice)
       and
      ?finalBalance= (?initialBalance - ?ticketPrice)
       and
       ?creditCard[
           po#balance  hasValue  ?finalBalance
       ] .
```

4.2.2 Web Service Interface

The *interface* of a Web service provides further information on how the functionality of the Web service is achieved. It contains:

- the description of the communication pattern that allows to one to consume the functionality of the Web service, named *choreography*
- the description of how the overall functionality of the Web service is achieved by means of cooperation of different Web service providers, named *orchestration*.

Thus, a Web service interface is defined by specifying the next elements: *non-functional properties*, *imported ontologies***,** *used mediators*, *choreography*, and *orchestration*.

Listing 8. Definition of a "Book Ticket Web Service" interface

```
interface BookTicketInterface
  importsOntology _"http://www.example.org/BookTicketInterfaceOntology"
  choreography BookTicketChoreography
  orchestration BookTicketOrchestration
```

The interface of the Web service, more specifically its components, the choreography and the orchestration, are depicted in Figure 1. The next subsections will describe in greater detail each of these two components.

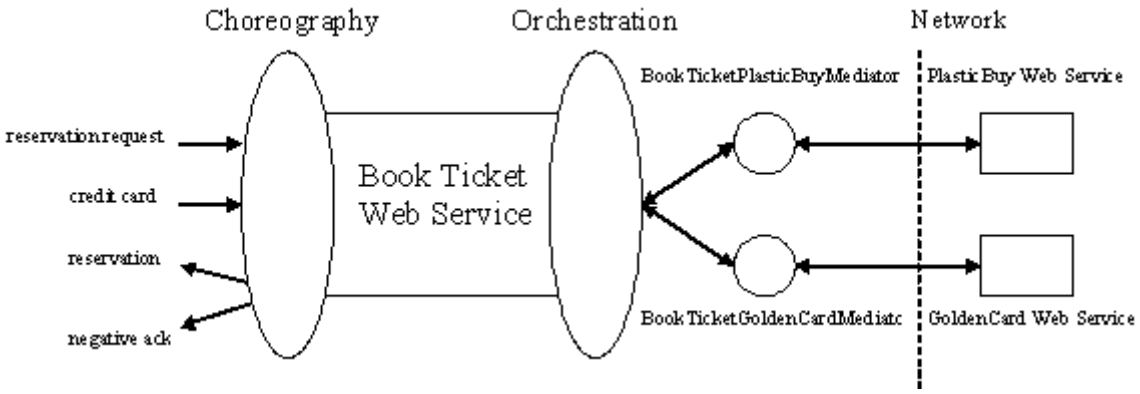

Figure 1. The "Book Ticket Web Service" interface

4.2.2.1 Choreography

The choreography specifies the behavior interface for Web service consumption. A user of the Web service has to support this for consuming a Web service. A choreography description has two parts: *the state* and *the guarded transitions*. A state is represented by an ontology, while guarded transitions are if-then rules that specify transitions between states.

Listing 9. Definition of a Book Ticket Web service Choreography

```
choreography BookTicketChoreography
  state _"http://example.org/BookTicketInterfaceOntology"
  guardedTransitions BookTicketChoreographyTransitionRules
```

The ontology that represents the state has a twofold role: first, it provides the vocabulary of the transition rules, and second, it contains the set of instances that change their values from one state to the other. Thus, its content is dynamic. Informally, a state is considered to be the dynamic set of intances at a certain point in time. The concepts, relations and functions of an ontology used for representing a state may have specified one or two additional non-functional properties. These non-functional properties are *mode* and *grounding*. The *mode* of an element gives an indication about who (the Web service and/or the environment) is allowed to modify the extension of the element and in which way (read/write). It can have one of the following values: *static* (the default value), *controlled*, *in*, *out*, *shared*. In the case of elements which have the mode *in*, *out* or *shared,* a grounding mechanism should be provided by using the non-functional property *grounding.* The ontology used for defining the choreography may reuse external ontologies by importing them and redefining the elements for which the specification of the *mode* is needed, and possibly that of *grounding*. This is done by declaring similar elements that inherit the definition of the original ones and adding the desired non-functional properties.

In our use case, the ontology used for describing the choreography, named "Book Ticket Interface Ontology" imports the ontologies "Trip Reservation Ontology" and "Purchase Ontology", and redefines the concepts `reservationRequest`, `creditCard and reservation` as having the modes *in*, *in*, and *out* respectively. Two new concepts are defined in this ontology, `temporaryReservation` and `negativeAcknowledgement` that have the modes *controlled* and *out* respectively. An instance of the first concept will be created in the choreography after receiving a reservation request and before receiving the credit card information, and an instance

of the second concept will be created as a result of receiving the information of a credit card that is invalid. The name of this ontology is "Book Ticket Interface Ontology" and not "Book Ticket Choreography Ontology", as it would be expected, because both the choreography and the orchestration of this Web service defined in the interface "BookTicketInterface" use this ontology.

Listing 10. Definition of the "Book Ticket Interface Ontology"

```
namespace {_"http://example.org/BookTicketInterfaceOntology#",
   dc     _"http://purl.org/dc/elements/1.1#",
   tr     _"http://example.org/tripReservationOntology#",
   wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
   po     _"http:/example.org/purchaseOntology#"
  }

ontology _"http://example.org/BookTicketInterfaceOntology#"
  nonFunctionalProperties
    dc#title hasValue "Book Ticket Interface Ontology"
    dc#creator hasValue "DERI Innsbruck"
    dc#description hasValue "an ontology  that redefines concepts and
      relations from other ontologies in order to reuse them in the
      choreography and orchestration; two additional non-functional
      properties are defined for the targeted concepts and relations:
      mode and grounding"
    dc#publisher hasValue "DERI International"
  endNonFunctionalProperties

  importsOntology { _"http://www.example.org/tripReservationOntology",
     _"http://www.wsmo.org/ontologies/purchaseOntology"
   }

  concept reservationRequest subConceptOf tr#reservationRequest
    nonFunctionalProperties
      wsml#mode hasValue wsml#in
      wsml#grounding hasValue reservationWSDL#reservationRequest
    endNonFunctionalProperties

  concept reservation subConceptOf tr#reservation
    nonFunctionalProperties
      wsml#mode hasValue wsml#out
      wsml#grounding hasValue reservationWSDL#reservation
    endNonFunctionalProperties

  concept temporaryReservation subConceptOf tr#reservation
    nonFunctionalProperties
      wsml#mode hasValue wsml#controlled
    endNonFunctionalProperties

  concept creditCard subConceptOf po#creditCard
    nonFunctionalProperties
      wsml#mode hasValue wsml#in
      wsml#grounding hasValue reservationWSDL#creditCard
    endNonFunctionalProperties

 concept negativeAcknowledgement
    nonFunctionalProperties
      wsml#mode hasValue wsml#out
      wsml#grounding hasValue reservationWSDL#negAck
    endNonFunctionalProperties
```

The guarded transitions are rules that are triggered when the current state fulfils certain conditions. This is done by checking the values of the attributes of the instances from the state ontology or the existence of certain instances in the state ontology in the current state. When this is the case a transition to a new state is performed, by changing the values of some of the attributes of the instances from the state ontology or by adding new instances to this ontology. The choreography defined in this example has three transition rules.

Listing 11. Definition of the guarded transition rules for the "Book Ticket Web Service" choreography

```
guardedTransitions BookTicketChoreographyTransitionRules

if  (reservationRequestInstance [
        reservationItem hasValue ?trip,
        reservationHolder hasValue ?reservationHolder
     ] memberOf bti#reservationRequest
    and
    ?trip memberOf tr#tripFromAustria
    and
    ticketInstance[
       trip hasValue ?trip,
       recordLocatorNumber hasValue ?rln
    ] memberOf tr#ticket
    then
     temporaryReservationInstance[
        reservationItem hasValue ticketInstance,
        reservationHolder hasValue ?reservationHolder
     ] memberOf bti#temporaryReservation

if (temporaryReservationInstance[
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
    ] memberOf bti#temporaryReservation
    and
    creditCardInstance memberOf bti#creditCard
    and
    po#validCreditCard(creditCardInstance))
then
     reservationInstance[
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
     ]memberOf bti#reservation

if (temporaryReservationInstance [
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
    ] memberOf bti#temporaryReservation
    and
    creditCardInstance memberOf bti#creditCard
    and
    neg (po#validCreditCard(creditCardInstance)))
then
     negativeAcknowledgementInstance memberOf bti#negativeAcknowledgement
```

The first rule checks whether a reservation request instance exists (i.e. it has been already received, since the corresponding concept has the mode *in*) with the request for a trip that starts in Austria and whether there a ticket instance for the desired trip in the Web service instance store. If this is the case, it creates a temporary reservation for that ticket. The next step is to wait for a credit card information, to check whether it points to a valid credit card and if this is the case to create a

reservation instance. This is depicted by the second guarded transition rule. In case the received credit card information points to an invalid credit card, a negative acknowledgement instance is created. This is depicted by the third transition rule.

4.2.2.2 Orchestration

The orchestration of a WSMO Web service defines how a Web service makes use of other WSMO Web services or goals in order to achieve its capability. Like for the choreography, an orchestration description has two parts: *the state* and *the guarded transitions*. A state is represented by an ontology, while guarded transitions are if-then rules that specify transitions between states.

Listing 12. Definition of the "Book Ticket Web Service" orchestration

```
orchestration BookTicketOrchestration
  state _"http://example.org/BookTicketInterfaceOntology"
  guardedTransitions BookTicketOrchestrationTransitionRules
```

In extention to the choreography, in an orchestration can also appear transition rules that have as postcondition the invocation of a mediator. Such a mediator is used for linking the orchestration with the choreography of a required Web service. This is done either directly, when the required Web service is known (the type of the mediator being *wwMediator*) or indirectly, via a goal, when the required Web service is not known (the type of the mediator being *wgMediator*) .

Listing 13. Definition of the guarded transition rules for the "Book Ticket Web Service" orchestration

```
guardedTransitions BookTicketOrchestrationTransitionRules

 if (creditCardInstance[
      type hasValue "PlasticBuy"
   ] memberOf bti#creditCard
   and
   po#validCreditCard(creditCardInstance))
 then
    _"http://example.org/BookTicketPlasticBuyMediator"

 if (creditCardInstance[
      type hasValue "GoldenCard"
   ] memberOf bti#creditCard
   and
   po#validCreditCard(creditCardInstance))
 then
    _"http://example.org/BookTicketGoldenCardMediator"
```

Thus, the orchestration of this Web service is described by two transition rules. These transition rules specify that when a valid credit card is received, depending on its type, the appropriate Web service for payment is invoked. As already described in its capability, this Web service only accepts PlasticBuy or GoldenCard credit cards. The invocation of one of the payment Web services is done using one of the two *wwMediators*, `BookTicketPlasticBuyMediator` or `BookTicketGoldenCardMediator`.

4.3 Goals

A goal specifies objectives that a client might have when consulting a Web service, i.e. functionalities that a Web service should provide from the user perspective. The coexistence of goals and Web services as non-overlapping entities ensures the decoupling between request and Web service. The requester, either human or machine, defines a goal to be resolved and the Web Service Discovery detects suitable Web services for solving the goal automatically. This kind of stating problems, in which the requester formulates objectives independent/ without regard to Web services for resolution is known as the *goal-driven approach*, derived from the AI rational agent approach.

A goal in WSMO is described by *non-functional properties*, *imported ontologies*, *used mediators*, *requested capability* and *requested interface*.

As already described in Section 3, our scenario is that a user wants to buy a ticket from Innsbruck (Austria) to Venice (Italy) on the date of 15 June 2005. A goal is abstracted from this user desire that does not include the desired date for the trip, because this information is not considered relevant in the Web service Discovery phase. Usually, a Web service will not advertise that it sells tickets only for certain dates and since the goal will have to be matched with one or more Web service capabilities, there is no point in making it more complex than any of these Web services capabilities.

A goal declaration starts with the keyword `goal` followed by the goal identifier. Optionally a set of *non-functional properties* follows, those being defined in a similar fashion with the non-functional properties for the components already described.

Furthermore, a goal can import existing ontologies to make use of ontological knowledge defined elsewhere, either directly, or via *ontology-to-ontology mediators*. The goal described above makes use of three ontologies, no mediation being needed. Another type of resources that can be reused within the definition of a goal via mediators are another goals. The mediation between two goals is realized by a type of mediators named *goal-to-goal mediators*. The relation between a goal and another goal imported via such a mediator is one of refinement. More generic goals are reused by specifying additional constraints. For example, the goal described above, abstracted from the user's desires, can be based on another goal, more generic, that says only that the intent is to have a reservation for a ticket for any trip. For the moment it is not defined how the *goal-to-goal mediator* achieves its functionality and how the functionality of the generic goal is refined by the concrete goal. When this will be defined, the generic goal will be included in the example and the concrete goal will only reuse this goal.

Below is the definition of the concrete goal previously described:

Listing 14. Goal declaration

```
namespace {_"http://example.org/goals#",
  dc     _"http://purl.org/dc/elements/1.1#",
  tr      _"http://example.org/tripReservationOntology",
  wsml  _"http://www.wsmo.org/wsml/wsml-syntax#",
  loc    _"http://www.wsmo.org/ontologies/locationOntology#"}

goal  _"http://example.org/havingATicketReservationInnsbruckVenice"
  importsOntology {_"http://example.org/tripReservationOntology",
          _"http://www.wsmo.org/ontologies/locationOntology"}
  capability
     postcondition
```

```
definedBy
  ?reservation[
     reservationHolder hasValue ?reservationHolder,
     item hasValue ?ticket
   ] memberOf tr#reservation
   and
   ?ticket[
     trip hasValue ?trip
  ] memberOf tr#ticket
  and
  ?trip [
     origin hasValue loc#innsbruck,
     destination hasValue loc#venice
  ] memberOf tr#trip.
```

The main element that appears in the declaration of a goal is the *requested capability*, which specifies the functionality required from a Web service. This is declared in the same way as a Web service capability, by using the keyword `capability` followed by the actual declaration. As described in the scenario above, the user expects to find a Web service, which as a result of its execution offers a reservation for a ticket for the desired trip. Thus, the only element of the capability the user is interested in, is its postcondition.

The user has also the possibility to specify the desired way of interacting with the Web service by declaring the *requested interface*. Such a declaration is done using the keyword `interface` followed by a declaration of some interface. It is not the case in this example.

4.4 Mediators

The existence of mediators allows one to link possibly heterogeneous resources. Mediators resolve incompatibilities that arise at different levels:

- *data level* - mediating between different used terminologies, more specifically solving the problem of ontology integration.
- *process level* - mediating between heterogeneous communication patterns. This kind of heterogeneity appears during the communication between different Web services. The process mediator provides the functionality for a runtime analysis of two given patterns, and compensates for the possible mismatches that may appear, by, for instance, generating dummy acknowledgement messages, grouping several messages into a single one, changing their order or even removing some of the messages in order to facilitate the communication between the two parties.

In the general case WSMO defines mediators by means of *non-functional properties*, *imported ontologies*, *source component*, *target component* and *mediation service*, where source and target component can be a mediator, a Web service, an ontology or a goal. As it was anticipated in Section 2.2 the current version of the WSMO specification distinguishes among four types of mediators: *ooMediators*, *ggMediators*, *wgMediators* and *wwMediators*.

4.4.1 OOMediators

OOMediators allow any WSMO component to import an ontology by solving all the

terminological mismatches that can appear during the process. Ontology mediation may be done in more than one step, hence the possibility for an *ooMediator* to mediate between an ontology and another *ooMediator*. The listing below contains the declaration of the *ooMediator* used by the "Trip Reservation Ontology" (*target*) for importing the "OWL Person Ontology" (*source*).

Listing 15. OOMediator for importing the "OWL Person Ontology" into the "Trip Reservation Ontology "

```
namespace{_"http://example.org/mediators#",
  dc    _"http://purl.org/dc/elements/1.1" ,
  wsml  _"http://www.wsmo.org/wsml/wsml-syntax#"
}
ooMediator _"http://example.org/owlPersonMediator"
  nonFunctionalProperties
    dc#title hasValue "OO Mediator importing the OWL Person ontology to WSML"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#description hasValue "Mediator to import an OWL person ontology into a WSML
        trip reservation ontology"
    dc#publisher hasValue _"http://example.org/foaf#deri"
    dc#contributor hasValue _"http://example.org/foaf#lausen"
    dc#language hasValue "en-us"
    dc#relation hasValue {_"http://daml.umbc.edu/ontologies/ittalks/person/",
                  _"http://example.org/tripReservationOntology"}
    dc#rights hasValue _"http://www.deri.org/privacy.html"
    wsml#version hasValue "$Revision: 1.17 $"
  endNonFunctionalProperties
  source _"http://daml.umbc.edu/ontologies/ittalks/person/"
  target _"http://example.org/tripReservationOntology"
  usesService  _"http://example.org/OWL2WSML"
```

4.4.2 GGMediators

GGmediators can link a goal to another goal. Like in the case of ontologies, mediation between goals can be done in more than one step, hence the possibility to have *ggMediators* that have as source or target other *ggMediators*. The link represents the refinement of the source goal into the target goal, allowing the definition of sub-goal hierarchies. Assuming that for our use case, besides the concrete goal, a generic goal was also defined, as described in Section 4.3, a *ggMediator* will be used for linking the two goals. This mediator is described in Listing 16. *GGMediators* can also use *ooMediators* for resolving terminology mismatches between the vocabularies used for description of goals, but this is not the case for the VTA example.

Listing 16. GGMediator that allows the concrete goal to reuse the generic goal

```
namespace { _"http://www.example.org/mediators",
   dc#  _"http://purl.org/dc/elements/1.1#",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}
ggMediator _"http://www.example.org/ggMed"
  nonFunctionalProperties
    dc#title hasValue "GG Mediator that links the generic goal for
        reserving tickets with a concrete goal for reserving a ticket
        for a trip from Innsbruck to Venice"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#publisher hasValue _"http://example.org/foaf#deri"
    dc#contributor hasValue _"http://example.org/foaf#stollberg"
```

```
    dc#format hasValue "text/html"
    dc#identifier hasValue _"http://example.org/ggmed"
    dc#language hasValue "en-us"
    dc#relation hasValue {
      _"http://example.org/havingATicketReservationInnsbruckVenice",
      _"http://example.org/havingATicketReservation"}
    dc#rights hasValue _"http://deri.at/privacy.html"
   wsml#version hasValue "$Revision: 1.17 $"
  endNonFunctionalProperties
  source _"http://example.org/havingATicketReservationInnsbruckVenice"
  target  _"http://example.org/havingATicketReservation"
```

4.4.3 WG Mediators

WGMediators are mediators that link Web services to goals. A *wgMediator* has- depending on which is the source and target of the mediator - one of the following functions: (1) it links a goal to a Web service via its choreography interface meaning that the Web service (totally or partially) fulfills the goal to which it is linked (2) it links a Web service to a goal via its orchestration interface meaning that the Web service needs this goal to be resolved in order to fulfil the functionality described in its capability. A *wgMediator* can be defined for the VTA use case that links the concrete goal "reserve a ticket for a trip from Innsbruck to Venice"(*source*) with the Book Ticket Web service(*target*).

Listing 17. WGMediator that links the "Book Ticket Web Service" with a related goal

```
namespace{ _"http://example.org/mediators",
   wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}
wgMediator _"http://example.org/wgMed"
  nonFunctionalProperties
    dc#title hasValue "WG Mediator that links the Book Ticket Web
    service with the goal that specifies the requirement of having a
    ticket reservation for a trip from Innsbruck to Venice"
  endNonFunctionalProperties
  source  _"http://example.org/BookTicketWebService"
  target _"http://example.org/havingATicketReservationInnsbruckVenice"
```

4.4.4 WWMediators

WWMediators link two Web services in order to enable interoperability of heterogeneous Web services. In the orchestration of the "Book Ticket Web Service" two such mediators are used for realizing the connection with the two Web services for payment, belonging to PlasticBuy, respectively GoldenCard. Listing 18 presents the definition of the `BookTicketPlasticBuyMediator`.

Listing 18. WWMediator that links the orchestration of the "Book Ticket Web Service" with the choreography of the "PlasticBuy Web Service"

```
namespace{ _"http://example.org/mediators",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}

wwMediator _"http://example.org/BookTicketPlasticBuyMediator"
  nonFunctionalProperties
```

```
    dc#title hasValue "WWMediator that links the orchestration of the
    Book Ticket Web service with the choreography of the PlasticBuy Web service"
endNonFunctionalProperties
source _"http://example.org/BookTicketWebService"
target _"http://example.org/PlasticBuyWebService"
```

5. Conclusions

This document presented how the main elements defined by the WSMO specification, ontologies, goals, Web services and mediators, and their subcomponents can be modeled in WSMO by providing examples for each of them. Due to the core design principles that underpin WSMO, decoupling and mediation, describing the various aspects of Semantic Web services can be done in an efficient and simple way.

For some parts of the specification, like the mediators, the work is in progress. In the future versions, the primer will include more detailed examples for modeling these parts.

6. WSMO Specification Structure

This section presents an overview of the documents the WSMO specification consists in. Every deliverable is listed below, together with a brief description of its content. Further, it provides links to the latest final draft, and the latest working draft of each currently available document. At the moment of writing the primer the WSMO specification included the following deliverables:

- **D2 Web Service Modeling Ontology**
  Presents the Web Service Modeling Ontology (WSMO), an ontology for describing Semantic Web services. The work takes as starting point the Web Service Modeling Framework (WSMF).
  - latest final draft: D2v1.1. Web Web service Modeling Ontology
  - latest working draft: D2v1.2. Web Web service Modeling Ontology

- **D3.1 Primer**
  Provides the reader with the fundamental knowledge required to effectively use WSMO, presenting its basic concepts, and finally describing the content and purpose of other documents within the specification.
  - latest working draft: D3.1v0.1. WSMO Primer

- **D3.2 WSMO Use Case Modeling and Testing**
  Outlines general usage of Semantic Web services in different application fields, identifying the usage scenarios and the arising technical requirements
  - latest final draft: D3.2v01. WSMO Use Case Modeling and Testing
  - latest working draft: D3.2v02. WSMO Use Case Modeling and Testing

- **D3.3 WSMO Use Case "Virtual Travel Agency"**

Describes a use case of Semantic Web services in the B2C area: how to model in WSMO a Web service for purchasing train tickets provided by a Virtual Travel Agency .
  - latest working draft: D3.3v0.1 WSMO Use Case "Virtual Travel Agency"

- **D3.4 WSMO B2B Use Case**
  Describes a use case of Semantic Web services in the B2B area and showcases how this can be modeled within WSMO.
  - latest working draft: D3.4v0.1 WSMO B2B Use Case

- **D3.5 Semantic Web Fred Use Case**
  Describes a use case of Semantic Web Web services defined for Semantic Web Fred - an agent system for automated, cooperative goal resolution that realizes WSMO.
  - latest working draft: D3.5v1.2 SWF Use Case

- **D5.1 WSMO Discovery**
  Provides a conceptual model for Web service discovery and different approaches to one of the steps of such model, web Web service discovery, which can be realized by WSMO and its related efforts.
  - latest working draft: D5.1v0.1. WSMO Discovery

- **D5.2 WSMO Discovery Engine** Investigates the implementation of a discovery engine that can provide dynamic web Web service discovery.
  - latest working draft: D5.2v0.1.WSMO Discovery Engine

- **D13.0 Overview and Scope of WSMX**
  The Web Service Execution Environment (WSMX) is an execution environment for dynamic discovery, selection, mediation and invocation of Web services. WSMX is based on the Web Service Modeling Ontology (WSMO) which describes all aspects related to discovery, mediation, selection and invocation.
  - latest working draft: D13.0v0.3. WSMX Conceptual Model

- **D13.1 WSMX Conceptual Model**
  Investigates and analyzes several different formal and logical frameworks for describing knowledge about a static world as well as formalisms that allow to model dynamic aspects of a changing world.
  - latest final draft: D13.1v0.2. WSMX Conceptual Model
  - latest working draft: D13.1v0.3. WSMX Conceptual Model

- **D13.2 WSMX Execution semantics**
  Defines the execution semantics of the Web Service Execution Environment.
  - latest working draft: D13.2v0.2. WSMX Execution Semantics

- **D13.3 WSMX Data Mediation**
  Provides an overview of the main problems that may occur during the mediation process together with the solutions that can be adopted for solving them.
  - latest final draft: D13.3v0.1. WSMX Mediation
  - latest working draft: D13.3v0.2. WSMX Mediation

- **D13.4 WSMX Architecture**
  Describes the WSMX architecture components.

  - latest final draft: D13.4v0.1. WSMX Architecture
  - latest working draft: D13.4v0.2. WSMX Architecture

- **D13.5 WSMX Implementation**
  Brings together the work described in the other deliverables of the specification. The implementation of WSMX will act as a WSMO test bed. Issues and solutions encountered in the course of the software design and development will provide a valuable source of feedback to the WSMO working group.
  - latest working draft: D13.5v0.1. WSMX Implementation

  D13.6 WSMX Use Cases
  Exemplifies several usage scenarios of WSMX.
  - latest working draft: D13.6v0.1. WSMX Use Cases

- **D14 Ontology-based Choreography and Orchestration of WSMO Services**
  Provides a core conceptual model for describing choreographies and orchestrations in WSMO. The state-based mechanism for describing WSMO choreographies and orchestrations is based on the Abstract State Machines methodology .
  - latest working draft: D14v0.1. Choreography in WSMO

-
- **D16.1 The WSML Family of Representation Languages**
  Presents the languages used to describe the concepts defined in WSMO.
  - latest working draft: D16.1v0.2. The WSML Family of Representation Languages

- **D16.2 WSML Reasoner Implementation**
  Lists the requirements posed by the different WSML variants on the WSML reasoner implementation and provides a survey of existing reasoners.
  - latest working draft: D16.2v0.2. WSML Reasoner Implementation

- **D17 WSMO Tutorial**
  Provides the resources to effectively disseminate the work carried within the WSMO specification.
  - latest working draft: D17v0.2. WSMO Tutorial

- **D19.1 WSMO in DIP**
  Presents the relation between the Semantic Web services working group of the SDK cluster which is WSMO and DIP, emphasizing on DIP contribution to the working group and how the project benefits from the close cooperation with SDK working group.
  - latest final draft: D19.1v0.2. WSMO in DIP

- **D19.2 WSMO in Knowledge Web**
  Presents the relation between the Semantic Web services working group of the SDK cluster which is WSMO and Knowledge Web, making clear how Knowledge Web contributes to the working group and how the project benefits from the close cooperation with SEKT and DIP in the context of the cluster
  - latest final draft: D19.2v0.1. WSMO in Knowledge Web

- **D19.3 WSMO in SEKT**
  Sets out the relationship between the Semantic Web services working group of

the SDK cluster and SEKT, making clear how SEKT contributes to the working group and how the project benefits from the close cooperation with SEKT and DIP in the context of the cluster.
    - latest working draft: D19.3v0.1. WSMO in SEKT

- **D20.1 OWL$^-$**
  Presents restricted variants of the OWL Lite, DL and Full species of the OWL ontology language, called OWL Lite$^-$, OWL DL$^-$ and OWL Full$^-$.
    - latest working draft: D20.1v0.2. OWL$^-$

- **D20.2 OWL Lite$^-$ Reasoning with Rules**
  Describes reasoning with ontologies using the TRIPLE language and reasoning system and elaborates on requirements for a reasoner for WSML.
    - latest working draft: D20.2v0.2. OWL Lite$^-$ Reasoning with Rules

- **D20.3 OWL Flight**
  Introduces OWL Flight, an extension of OWL$^-$ (a subset of OWL, which can be translated to Datalog) with different kinds of constraints and a form of the local closed-world assumption, as well as support for datatypes based on the OWL-E datatypes extension for OWL.
    - latest working draft: D20.3v0.1. OWL Flight

- **D21 WSMX Triple Space Computing**
  Describes how the Tuple and Triple Space Computing communication paradigms can be used within WSMX.
    - latest working draft: D21v0.1. Integrating WSMX with Tuple and Triple Space Computing

- **D22 WSMX Documentation**
  Provides the description of a step by step WSMX installation and examples of its usage.
    - latest working draft: D22v0.1. WSMX Documentation

- **D23 WSMX System Functionality Scope**
  Provides an overview of the expected functionality WSMX is going to provide with each subsequent release of the software.
    - latest working draft: D23v0.1. WSMX System Functionality Scope

- **D24.1 Aligning WSMO and WSMX with existing Web services specifications**
  Provides a listing of those Web services specifications that may use WSMO or may be used by WSMO, together with a description of the mechanisms linking WSMO and the particular Web services specifications.
    - latest working draft: D24.1v0.1. Aligning WSMO and WSMX with existing Web Web services specifications

- **D24.2 WSMO Grounding**
  Investigates the two independent areas of relationship between WSMO and the syntactical descriptions of Web services, collectively referred as WSMO grounding: data in WSMO ontologies that has to be mapped to XML data, and functional and behavioral Web service descriptions in WSMO that have to be related to the description construct present in WSDL.
    - latest working draft: D24.2v0.1.WSMO Grounding

- **D25.1 WSMO and Grid**
  Investigates how the conceptual model provided by WSMO can be used for Grid computing applications and what variant of WSML is more suitable for Grid requirements.
  - latest working draft: D25.1v0.1. WSMO and Grid

- **D26 WSMX Grounding**
  Describes how WSMX handles the translation to and from WSML to other data representations at the boundary of the environment
  - latest working draft: D26v0.1. WSMX Grounding

- **D28.1v0.1. Functional Specification in common Specification Frameworks for Software Components**
  Overviews some of the most well-known frameworks for the functional specification of software components and evaluates their use for Web service capability specification in WSMO.
  - latest working draft: D28.1v0.1. Functional Specification in common Specification Frameworks for Software Components

- **D29v0.1. WSMO Mediators**
  Provides a general overview of what mediation is in the context of Semantic Web services, and presents what solutions WSMO offers for dealing with different types of mediation
  - latest working draft: D29v0.1. WSMO Mediators

References

[Baida et al., 2004] Z. Baida, J. Gordijn, B. Omelayenko and H. Akkermans: *A Shared Service Terminology for Online Service Provisioning*, ICEC '04: Proceedings of the 6th International Conference on Electronic Commerce, p.1-10, 2004.

[Brickley & Miller, 2004] D. Brickley and L. Miller: *FOAF Vocabulary Specification*, available at: http://xmlns.com/foaf/0.1/

[de Bruijn et al., 2005] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel: *The Web Service Modeling Language (WSML)*. Deliverable D16.1v02, WSML, 2005. Available from http://www.wsmo.org/TR/d16/d16.1/v0.2/

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

[Gruber, 1993] Thomas R. Gruber: *A Translation Approach to Portable Ontology Specifications*, Knowledge Acquisition, 5:199-220, 1993.

[IANA] Internet Assigned Numbers Authority. <http://www.isi.edu/in-notes/iana/assignments/media-types/media-types>

[ISO639, 1988] International Organization for Standardization (ISO): *ISO 639:1988 (E/F). Code for the Representation of Names of Languages.* First edition, 1988-04-01. Reference number: ISO 639:1988 (E/F). Geneva: International Organization for

Standardization, 1988.

[ISO8601, 2004] International Organization for Standardization (ISO): *ISO 8601:2000. Representation of dates and times*. Second edition, 2004-06-08. Reference number. Geneva: International Organization for Standardization, 2004. Available from http://www.iso.org/.

[Patel-Schneider et al., 2004] P. F. Patel-Schneider, P. Hayes, and I. Horrocks: *OWL Web Ontology Language Semantics and Abstract Syntax*. Recommendation 10 February 2004, W3C, 2004. Available from http://www.w3.org/TR/owl-semantics/.

[Preist, 2004] Chris Preist. *A Conceptual Architecture for Semantic Web Services.* In Proceedings of the International Semantic Web Conference 2004 (ISWC 2004), November 2004.

[Roman et al., 2005] D. Roman, H. Lausen, U. Keller (eds.): *Web Service Modeling Ontology (WSMO)*. Deliverable D2v1.1, WSMO, 2005. Available from http://www.wsmo.org/TR/d2/v1.1/.

[TGN] Getty Thesaurus of Geographic Names. <http://www.getty.edu/research/tools/vocabulary/tgn/index.html>

[W3CDTF] Date and Time Formats, W3C Note. <http://www.w3.org/TR/NOTE-datetime>

Appendix

Here you can find the WSML definitions for the entities used to describe the use case that guides the primer. While all these definitions were presented throughout the document, the definitions of certain entities (e.g. the Trip Reservation Ontology) were spread in different listings or interleaved with the definitions of other entities. Hence, the need for this integrated listing.

```
/***********************************
The Trip Reservation Ontology
***********************************/

namespace {_"http://example.org/tripReservationOntology#",
  dc     _"http://purl.org/dc/elements/1.1#",
  loc    _"http://example.org/locationOntology#",
  po     _"http://example.org/purchaseOntology#",
  foaf   _"http://xmlns.com/foaf/0.1/",
  wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
  prs    _"http://example.org/owlPersonMediator#"}

ontology  _"http://example.org/tripReservationOntology"
  nonFunctionalProperties
    dc#title hasValue "Trip Reservation Ontology"
    dc#identifier hasValue _"http://example.org/tripReservationOntology"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#description hasValue
      "an ontology for describing trip reservations related knowledge"
    dc#publisher hasValue _"http://example.org/foaf#deri"
    dc#contributor hasValue  _"http://example.org/foaf#cristina"
    dc#date hasValue _date(2004,12,16)
```

```
  dc#format hasValue "text/x-wsml"
  dc#language hasValue "en-us"
  dc#rights hasValue _"http://deri.at/privacy.html"
  wsml#version hasValue "$Revision 1.17 $"
endNonFunctionalProperties

importsOntology{ _"http://example.org/locationOntology",
                 _"http://example.org/purchaseOntology"}

usesMediator _"http://example.org/owlPersonMediator"

concept trip
  origin impliesType loc#location
  destination impliesType loc#location
  departure ofType _date
  arrival ofType _date

concept tripFromAustria subConceptOf trip
  nonFunctionalProperties
    dc#relation hasValue tripFromAustriaDef
  endNonFunctionalProperties

axiom tripFromAustriaDef
  definedBy
     forall {?x ,?origin}
          (?x memberOf tripFromAustria
            implies
            ?x[
               origin hasValue ?origin] and
            ?origin[
               loc#locatedIn hasValue loc#austria]
          ).

concept ticket
  provider ofType _string
  trip ofType trip
  recordLocatorNumber ofType _integer

concept reservationRequest
  nonFunctionalProperties
    dc#description hasValue "This concept represents a
    reservation request for some trip for a particular person"
  endNonFunctionalProperties
  reservationItem impliesType wsml#true
  reservationHolder impliesType prs#person

concept reservation
  nonFunctionalProperties
     dc#description hasValue "concept of a confirmation for some item"
  endNonFunctionalProperties
  reservationItem impliesType wsml#true
  reservationHolder impliesType prs#person

relation ticketPrice(ofType ticket, ofType po#currency, ofType _integer)
nonFunctionalProperties
   dc#description hasValue
      "Function that returns the price of a ticket in a given currency"
   dc#relation hasValue {FunctionalDependencyTripPrice}
 endNonFunctionalProperties

axiom FunctionalDependencyTicketPrice
   definedBy
      !- ticketPrice(?x,?y,?z1) and ticketPrice(?x,?y,?z2) and ?z1 != ?z2.

instance tripInnVen memberOf trip
```

```
    origin hasValue loc#innsbruck
    destination hasValue loc#venice
    departure hasValue  _date(2005,11,22)
    arrival hasValue  _date(2005,11,22)

  instance ticketInnVen memberOf ticket
    provider hasValue "Book Ticket Service"
    trip hasValue tripInnVen
    recordLocatorNumber hasValue 27

  relationInstance ticketPrice(ticketInnVen, po#euro, 120)


/************************************
The Purchase Ontology
************************************/


namespace { _"http://example.org/purchaseOntology#",
  dc     _"http://purl.org/dc/elements/1.1",
  wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
  prs    _"http://example.org/owlPersonMediator#"}

ontology  _"http://example.org/purchaseOntology"

usesMediator _"http://example.org/owlPersonMediator"

concept creditCard
  owner impliesType prs#person
  number ofType _integer
  type ofType _string
  expiryDate ofType _date
  balance ofType _integer

relation validCreditCard(ofType creditCard)
  nonFunctionalProperties
    dc#description hasValue "Relation that holds for a valid credit card"
    dc#relation hasValue ValidCreditCardDef
  endNonFunctionalProperties

axiom ValidCreditCardDef
definedBy
   forall {?x, ?y}  (
        validCreditCard(?x)  impliedBy
        ?x[expiryDate hasValue ?y] memberOf creditCard
        and
        neg (wsml#dateLessThan(?y, wsml#currentDate()))).

relationInstance TicketPrice(ticketInnVen, po#euro, 120)


/*********************************************
The Book Ticket Interface Ontology
*********************************************/

namespace {_"http://example.org/BookTicketInterfaceOntology#",
    dc     _"http://purl.org/dc/elements/1.1#",
    tr     _"http://example.org/tripReservationOntology#",
    wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
    po     _"http:/example.org/purchaseOntology#"
   }

ontology _"http://example.org/BookTicketInterfaceOntology#"
   nonFunctionalProperties
      dc#title hasValue "Book Ticket Interface Ontology"
```

```
    dc#creator hasValue "DERI Innsbruck"
    dc#description hasValue "an ontology  that redefines concepts and
      relations from other ontologies in order to reuse them in the
      choreography and orchestration; two additional non-functional
      properties are defined for the targeted concepts and relations:
      mode and grounding"
    dc#publisher hasValue "DERI International"
  endNonFunctionalProperties

  importsOntology { _"http://www.example.org/tripReservationOntology",
     _"http://www.wsmo.org/ontologies/purchaseOntology"
   }

  concept reservationRequest subConceptOf tr#reservationRequest
    nonFunctionalProperties
      wsml#mode hasValue wsml#in
      wsml#grounding hasValue reservationWSDL#reservationRequest
    endNonFunctionalProperties

  concept reservation subConceptOf tr#reservation
    nonFunctionalProperties
      wsml#mode hasValue wsml#out
      wsml#grounding hasValue reservationWSDL#reservation
    endNonFunctionalProperties

  concept temporaryReservation subConceptOf tr#reservation
    nonFunctionalProperties
      wsml#mode hasValue wsml#controlled
    endNonFunctionalProperties

  concept creditCard subConceptOf po#creditCard
    nonFunctionalProperties
      wsml#mode hasValue wsml#in
      wsml#grounding hasValue reservationWSDL#creditCard
    endNonFunctionalProperties

 concept negativeAcknowledgement
    nonFunctionalProperties
      wsml#mode hasValue wsml#out
      wsml#grounding hasValue reservationWSDL#negAck
    endNonFunctionalProperties


/************************************
The Book Ticket Web service
************************************/

namespace {_"http://example.org/bookTicket#",
   dc      _"http://purl.org/dc/elements/1.1#",
   tr      _"http://example.org/tripReservationOntology#",
   foaf    _"http://xmlns.com/foaf/0.1/",
   wsml    _"http://www.wsmo.org/wsml/wsml-syntax#",
   bti     _"http://www.example.org/BookTicketInterfaceOntology#"}

webService _"http://example.org/bookTicketWebService"

importsOntology _"http://example.org/tripReservationOntology"
capability BookTicketCapability

  sharedVariables {?creditCard, ?initialBalance, ?trip, ?reservationHolder, ?ticket}

  precondition
    nonFunctionalProperties
        dc#description hasValue "The information of a trip which starts in Austria
         together with the information about the person who wants to have a reservation
```

```
        must be given as a part of a reservation request. A credit card is also required."
    endNonFunctionalProperties
    definedBy
       ?reservationRequest[
          reservationItem hasValue ?trip,
          reservationHolder hasValue ?reservationHolder
       ] memberOf tr#reservationRequest
       and
       ?trip memberOf tr#tripFromAustria and
       ?creditCard[
          balance hasValue ?initialBalance
       ] memberOf po#creditCard.

  assumption
    nonFunctionalProperties
       dc#description hasValue "The credit information provided by the requester
        must designate a valid credit card that should be either PlasticBuy
        or GoldenCard."
    endNonFunctionalProperties
    definedBy
       po#validCreditCard(?creditCard)
       and ( ?creditCard[
                type hasValue "PlasticBuy"]
             or
            ?creditCard[
                type hasValue "GoldenCard"]
           ).

  postcondition
    nonFunctionalProperties
       dc#description hasValue "A reservation containing the details of a ticket for
        the desired trip and the reservation holder is the result of the successful
        execution of the Web service."
    endNonFunctionalProperties
    definedBy
      ?reservation memberOf tr#reservation[
        reservationItem hasValue ?ticket,
        reservationHolder hasValue ?reservationHolder
      ]
      and
      ?ticket[
        trip hasValue ?trip
      ]
      memberOf tr#ticket.

  effect
    nonFunctionalProperties
       dc#description hasValue "The credit card  will be charged with the cost of
        the ticket."
    endNonFunctionalProperties
    definedBy
       ticketPrice(?ticket, "euro", ?ticketPrice)
       and
       ?finalBalance= (?initialBalance - ?ticketPrice)
       and
       ?creditCard[
          po#balance  hasValue  ?finalBalance
       ] .

interface BookTicketInterface
  choreography BookTicketChoreography
    state _"http://example.org/BookTicketInterfaceOntology"
    guardedTransitions BookTicketTransitionRules

  guardedTransitions BookTicketChoreographyTransitionRules
```

```
if  (reservationRequestInstance[
        reservationItem hasValue ?trip,
        reservationHolder hasValue ?reservationHolder
     ] memberOf bti#reservationRequest
    and
    ?trip memberOf tr#tripFromAustria
    and
    ticketInstance[
       trip hasValue ?trip,
       recordLocatorNumber hasValue ?rln
    ]memberOf tr#ticket
    then
     temporaryReservationInstance[
        reservationItem hasValue ticketInstance,
        reservationHolder hasValue ?reservationHolder
     ] memberOf bti#temporaryReservation

if (temporaryReservationInstance[
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
    ] memberOf bti#temporaryReservation
    and
    creditCardInstance memberOf bti#creditCard
    and
    po#validCreditCard(creditCardInstance))
then
     reservationInstance[
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
     ]memberOf bti#reservation

if (temporaryReservationInstance [
       reservationItem hasValue ticketInstance,
       reservationHolder hasValue ?reservationHolder
    ]memberOf bti#temporaryReservation
    and
    creditCardInstance memberOf bti#creditCard
    and
    neg (po#validCreditCard(creditCardInstance)))
then
     negativeAcknowledgementInstance memberOf bti#negativeAcknowledgement

orchestration BookTicketOrchestration
  state _"http://example.org/BookTicketInterfaceOntology"
  guardedTransitions BookTicketOrchestrationTransitionRules

 if (creditCardInstance[
       type hasValue "PlasticBuy"
    ] memberOf bti#creditCard
    and
    po#validCreditCard(creditCardInstance))
then
     _"http://example.org/BookTicketPlasticBuyMediator"

if (creditCardInstance[
       type hasValue "GoldenCard"
    ] memberOf bti#creditCard
    and
    po#validCreditCard(creditCardInstance))
then
     _"http://example.org/BookTicketGoldenCardMediator"


/*************************************************************
```

```
The goal of booking a ticket for a trip from Innsbruck to Venice
*************************************************************/

namespace {_"http://example.org/goals#",
  tr     _"http://example.org/tripReservationOntology",
  loc    _"http://www.wsmo.org/ontologies/locationOntology#"}

goal  _"http://example.org/havingATicketReservationInnsbruckVenice"
 importsOntology {_"http://example.org/tripReservationOntology",
          _"http://www.wsmo.org/ontologies/locationOntology"}
 capability
    postcondition
        definedBy
          ?reservation[
             reservationHolder hasValue ?reservationHolder,
             item hasValue ?ticket
           ] memberOf tr#reservation
           and
           ?ticket[
             trip hasValue ?trip
          ] memberOf tr#ticket
          and
          ?trip [
            origin hasValue loc#innsbruck,
            destination hasValue loc#venice
          ] memberOf tr#trip.

/*************************************************************
An example of oomediator
*************************************************************/

namespace{_"http://example.org/mediators#",
 dc     _"http://purl.org/dc/elements/1.1" ,
 wsml   _"http://www.wsmo.org/wsml/wsml-syntax#"
}
ooMediator _"http://example.org/owlPersonMediator"
 nonFunctionalProperties
   dc#title hasValue "OO Mediator importing the OWL Person ontology to WSML"
   dc#creator hasValue _"http://example.org/foaf#deri"
   dc#description hasValue "Mediator to import an OWL person ontology into a WSML
        trip reservation ontology"
   dc#publisher hasValue _"http://example.org/foaf#deri"
   dc#contributor hasValue _"http://example.org/foaf#lausen"
   dc#language hasValue "en-us"
   dc#relation hasValue {_"http://daml.umbc.edu/ontologies/ittalks/person/",
                  _"http://example.org/tripReservationOntology"}
   dc#rights hasValue _"http://www.deri.org/privacy.html"
   wsml#version hasValue "$Revision: 1.17 $"
 endNonFunctionalProperties
 source _"http://daml.umbc.edu/ontologies/ittalks/person/"
 target _"http://example.org/tripReservationOntology"
 usesService  _"http://example.org/OWL2WSML"


/*************************************************************
An example of ggmediator
*************************************************************/


namespace { _"http://www.example.org/mediators",
   dc#  _"http://purl.org/dc/elements/1.1#",
   wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}
ggMediator _"http://www.example.org/ggMed"
  nonFunctionalProperties
```

```
    dc#title hasValue "GG Mediator that links the generic goal for
        reserving tickets with a concrete goal for reserving a ticket
        for a trip from Innsbruck to Venice"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#publisher hasValue _"http://example.org/foaf#deri"
    dc#contributor hasValue _"http://example.org/foaf#stollberg"
    dc#format hasValue "text/html"
    dc#identifier hasValue _"http://example.org/ggmed"
    dc#language hasValue "en-us"
    dc#relation hasValue {
      _"http://example.org/havingATicketReservationInnsbruckVenice",
      _"http://example.org/havingATicketReservation"}
    dc#rights hasValue _"http://deri.at/privacy.html"
    wsml#version hasValue "$Revision: 1.17 $"
  endNonFunctionalProperties
  source _"http://example.org/havingATicketReservationInnsbruckVenice"
  target  _"http://example.org/havingATicketReservation"


/*****************************************************************
An example of wgmediator
*****************************************************************/

namespace{ _"http://example.org/mediators",
   wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}
wgMediator _"http://example.org/wgMed"
  nonFunctionalProperties
    dc#title hasValue "WG Mediator that links the Book Ticket Web
    service with the goal that specifies the requirement of having a
    ticket reservation for a trip from Innsbruck to Venice"
  endNonFunctionalProperties
  source  _"http://example.org/BookTicketWebService"
  target _"http://example.org/havingATicketReservationInnsbruckVenice"


/*****************************************************************
An example of wwmediator
*****************************************************************/

namespace{ _"http://example.org/mediators",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}

wwMediator _"http://example.org/BookTicketPlasticBuyMediator"
  nonFunctionalProperties
    dc#title hasValue "WWMediator that links the orchestration of the
    Book Ticket Web service with the choreography of the PlasticBuy Web service"
  endNonFunctionalProperties
  source  _"http://example.org/bookTicketWebService"
  target _"http://example.org/PlasticBuyWebService"
```

Acknowledgments

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.

The editor would like to thank to all the members of the WSMO working group for

their advices and inputs to this document.

---

webmaster