Abstract  This chapter presents the Web Service Modeling Ontology, an ontology for describing various aspects related to Semantic Web Services. WSMO is the major initiative in the area of Semantic Web Services in Europe. The starting point of WSMO was the Web Service Modeling Framework (WSMF) (Fensel and Bussler in Electronic Commerce Research and Applications 1(2):113–137, 2002), which was refined and extended, resulting in (i) a formal ontology, the Web Service Modeling Ontology (WSMO), (ii) a formal family of languages, the Web Service Modeling Language (WSML), and (iii) an execution environment, the Web Service Modeling Environment (WSMX). The four main elements of WSMF have been adopted and refined in WSMO, resulting in four main elements for describing Semantic Web Services: (i) ontologies that provide the terminology used by other elements, (ii) goals that define the problems that should be solved by Web services, (iii) Web services descriptions that define various aspects of a Web service, and (iv) mediators which facilitate the resolution of interoperability problems. WSMO is a key technology needed for the successful realization of Semantic Web Services and Semantically Enabled Service-oriented Architectures. It provides a conceptual model for Semantic Web Services that combines the design principles of the Web, the Semantic Web and distributed service oriented computing in order to provide a clean modeling solution for various aspects related to Semantic Web Services.

7.1 Motivation

Current Web service technologies around SOAP [11], WSDL [3], and UDDI [1] operate at the syntactic level. Therefore, although they support interoperability between the many diverse application development platforms that exist today through common standards, they still require human interaction to a large extent: the human programmer has to manually search for appropriate Web services in order to combine them in a useful manner. Having a human programmer in the loop that needs to take care of various service related tasks limits the scalability of service-oriented solutions and greatly curtails the added economic value of envisioned with the advent of Web services [4]. In order to automate tasks such as Web service discovery, composition and execution, semantic descriptions of Web services are required. Semantic Web Services are building on top of Web services technology by describing
various aspects of services using explicit, machine-understandable semantics that enables a certain degree of automation for various service related tasks. In a nutshell, work in the area of Semantic Web [2] is being applied to Web services in order to keep the intervention of the human user to a minimum. WSMO provides a fully-fledged framework for Semantic Web Services. It includes a conceptual model, a formal language with a formal syntax and semantics based on different logics in order to support different levels of logical expressiveness, and an execution environment that is able to discover, select, compose and invoke Semantic Web Services modeled and formalized in WSMO. This chapter introduces one of the most important approaches to Semantic Web Services, namely the Web Service Modeling Ontology.

7.2 Technical Solution

The WSMO approach to Semantic Web Services includes: (i) the Web Service Modeling Ontology (WSMO), a conceptual model for Semantic Web Services, (ii) the Web Service Modeling Language (WSML), a language providing a formal syntax and semantics for WSMO, and (iii) the Web Service Execution Environment (WSMX), an execution environment for WSMO descriptions formalized in WSML. The rest of this section describes in detail the WSMO conceptual model. The other two building blocks of the WSMO approach, namely the Web Service Modeling Language (WSML) and the Web Service Execution Environment (WSMX), will be described in Chaps. 8 and 9, respectively.

WSMO [7] provides ontological specifications for the core elements of Semantic Web Services being based on the following design principles:

- **Web Compliance**
  WSMO inherits the concept of Uniform Resource Identifier (URI) for unique identification of resources as the essential design principle for the Word Wide Web. Moreover, it adopts the concept of namespaces for denoting consistent information spaces, supports XML and 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. 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 regarding possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
7.2 Technical Solution

- **Centrality of Mediation**
  As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneities can occur in terms of data, underlying ontology, protocol, and 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**
  Users, or more generally clients, operate in specific contexts which will not be the same as for available Web services. For example, a user may wish to book a holiday according to preferences for weather, culture, and childcare, whereas Web services will typically cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and available services.

- **Description versus Implementation**
  WSMO distinguishes between the descriptions of Semantic Web Services elements (description) and executable technologies (implementation). The former requires a concise and sound description framework based on appropriate formalisms in order to provide concise semantic descriptions. The latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services.

- **Execution Semantics**
  In order to verify the WSMO specification, the formal execution semantics of reference implementations such as WSMX as well as other WSMO-enabled platforms provide the technical realization of WSMO. This principle serves as a means to precisely define the functionality and behavior of the systems that are WSMO-compliant.

- **Service versus Web service**
  A Web service is a computational entity that is able to achieve a user goal by invocation. A service, in contrast, is the actual value provided by this invocation. WSMO provides the means to describe Web services that provide access (searching, buying, etc.) to services. WSMO is designed as a means to describe the former, and not to replace the functionality of the latter.

Based on these design principles, the WSMO meta-model has four top level elements which can be seen in Fig. 7.1, namely Ontologies, Web Services, Goals, and Mediators. To effectively describe Semantic Web Services, we need to understand each of these four elements. WSMO makes use of the Meta-Object Facility (MOF) [6] specification to define its meta-model. The MOF provides a language and framework for specifying technology neutral meta-models. The benefit of using MOF is that the model and the languages that are ultimately used to describe Semantic Web Services are separated from one another. This separation gives significantly more freedom than with competing approaches such as OWL-S (see Chap. 11). In WSMO, every element can be annotated using the WSMO annotations. One of the most used standards for annotations is the Dublin Core (DC) Metadata Set [12]. In the following sections, we use the MOF specification to describe the four top level elements of the WSMO metamodel and their child elements.
7.2.1 Ontologies

Ontologies in WSMO provide the terminology used across the descriptions of all other WSMO elements. Ontologies are crucial to the success of Semantic Web Services, as they provide the means by which enhanced information processing becomes possible and complex interoperability problems can be solved. WSMO is an epistemological model in that it is general enough to intuitively capture existing languages used for describing ontologies. In Listing 7.1 a WSMO ontology is described using the MOF notation.

7.2.1.1 Annotations

It is possible to add annotations onto all WSMO elements. They include aspects such as the creator, the creation date, the version, etc. The elements defined by the Dublin Core Metadata Initiative [12] are taken as a starting point. Dublin Core is a set of attributes that define a standard for cross-domain information resource description. WSMO uses IRIs for identification of elements. This is also true of annotations which are specified as key value pairs, where the key is an IRI identifying the property, and the value is another element.

7.2.1.2 Imported Ontologies

All top levels elements within the WSMO meta-model may use the importsOntology statement to import ontologies that contain the relevant concepts needed to
build a description, for example, a WSMO Web service description will import those WSMO Ontologies that contain concepts needed to describe the service in question.

### 7.2.1.3 Mediators for Importing Ontologies

Of course, it may not always be possible to directly import a given ontology, as mismatches between statements in the importing and imported ontology or between any two given imported ontologies may exist. In this case, a mediator is required, and the ontology is imported via this mediator. This mediator deals with the alignment, merging and transformation of the imported ontology so that existing heterogeneity issues that may arise are fixed. Like the `importsOntology` statement, the `usesMediator` statement may be used on all top level elements within the WSMO meta-model.

### 7.2.1.4 Concepts

Concepts represent the basic agreed upon terminology for a given domain. A concept is made up of a set of attributes, where each attribute has a name and a type. Using the MOF notation, the definition of a concept is depicted in Listing 7.2.

A concept can also be a sub-concept of one or more direct super-concepts using the `hasSuperConcept` statement. Sub-concepts inherit the signatures of all its super-concepts, for example, all attributes defined for a given concept will also hold for any of its sub-concepts.

Using the `hasAttribute` statement a set of attributes on the concept can be specified. The MOF definition of an attribute is also specified in Listing 7.2. The range of an attribute can be constrained to a given concept, meaning that all instances of the concept that specify a value for this attribute must conform to this range restriction.

Furthermore, a concept can be further refined by adding a logical expression through the `hasDefinition` statement. This logical expression can be used to express additional constraints on the concept than cannot be captured through the `hasAttribute` or `hasSuperConcept` statement.
7.2.1.5 Relations and Functions

Relations and functions define connection between several concepts. The arity of a relation is not restricted, thus a relation is able to model dependencies between two or more concepts. The MOF definition of a relation is presented in Listing 7.3.

Each relation can have zero or more super-relations. A relation inherits the signature of the super relation along with any associated constraints. Similarly to attributes for concepts, a relation can define a set of parameters, which may be a named or an ordered unnamed set. It is possible to define the domain of each of the parameters, where the domain specifies the allowed values that can be placed in this slot of the relation when it is instantiated. WSMO Ontologies can also have functions, specified with the `hasFunction` statement. Functions, as described in Listing 7.4, are special relations with a unary range, specified along with the parameters (the domain). Functions can be used to represent built in predicates of common data types. The semantics of a relation or function can be captured within a logical expression specified with the `hasDefinition` statement.

7.2.1.6 Instances of Concepts and Relations

As can be seen in Listing 7.5, it is possible to instantiate both concepts and relations defined within a WSMO Ontology. When instantiating a concept or relation, values are assigned to the attributes or parameters of the concept or relation being instantiated, where the type of the value being assigned conforms to the range of the attribute or the domain of the parameter. Instances may be defined explicitly within the ontology, however, in general a link to an external store of instances will be given when a large number of instances exist.
7.2 Technical Solution

Class instance
  hasAnnotations type annotations
  hasType type concept
  hasAttributeValues type attributeValue

Class relationInstance
  hasAnnotations type annotations
  hasType type relation
  hasParameterValue type parameterValue

Listing 7.5  Instance definitions

Class axiom sub−Class wsmoElement
  hasDefinition type logicalExpression

Listing 7.6  Axiom definition

Class service
  hasAnnotations type annotations
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type {ooMediator, wwMediator}
  hasCapability type capability multiplicity = single−valued
  hasInterface type interface

Listing 7.7  Web service definition

7.2.1.7 Axioms

An axiom, as showed in Listing 7.6, is a logical expression together with its non-functional properties. Axioms provide a mechanism for adding arbitrary logical expressions to an ontology, and can be used to refine concepts, relation or function definitions, to add arbitrary axiomatic domain knowledge, or to express constraints.

7.2.2 Web Services

Web services are computational entities that provide some functionality that has an associate value in a given domain. A WSMO Web service is a formal description of the Web services functionality, in terms of a capability, and the method to interact with it, in terms of an interface. A formal description of a WSMO Web service using the MOF notation given in Listing 7.7.

The ontologies that are required in order to define the service can be imported via the importsOntology and usesMediator statements. The usesMediator statement may also be used with a wwMediator (see Sect. 7.2.4.4), in cases where process or protocol heterogeneity issues need to be resolved.
7.2.2.1 Non-Functional Properties for Web Services

A service can specify one or more non-functional properties. Listing 7.8 contains the MOF definition of a non-functional property in WSMO.

In a nutshell, the non-functional properties captured constraints over the functional and behavioral properties of the service. Typical examples of non-functional properties include temporal and locative availability of the service, security, rights, obligations, reliability, performance, etc. Non-functional properties are not specific to Web services only, but other WSMO elements descriptions (i.e., goal, mediators, capabilities and interfaces) can contain non-functional properties descriptions.

7.2.2.2 Capabilities

The capability of the service describes the real value of the service and is described in MOF in Listing 7.9.

The capability of a service contains a set of axioms that describe the state of the world before and after the execution of the service. Using the hasPrecondition and hasPostcondition statements, it is possible to make axiomatic statements about the expected inputs and outputs of the service, i.e., what information must be available for the service to be executed and what information will be available after the service has been executed. The hasAssumption and hasEffect statements can be used to state the assumed state of the world prior to execution and the guaranteed state of the world afterwards. The capability of a service can be used by a requester for discovery purposes, i.e., to determine if the functionality of the service meets the requesters functional needs.
7.2 Technical Solution

7.2.2.3 Interfaces

The interface of a service describes how the functionality can be achieved. It is described in MOF as depicted in Listing 7.10.

The interface of a service provides a dual view on the operational competence of the service. Using the `hasChoreography` statement, a decomposition of the capability in terms of interaction with the service is provided, while using the `hasOrchestration` statement the capability can be decomposed in terms of the functionality required from other services in order to realize this service. This corresponds to the distinction between communication and cooperation. The choreography describes how to interact with a service, while the orchestration describes how the overall function of the service is realized through cooperation with other services (see Listing 7.11). The interface of a service is presented in a machine processable manner, allowing for software to automatically determine the behavior of the service and to reason on it.

Both choreography and orchestration are defined using the same formalism, based on Abstract State Machines [5]. The choreography of a service is defined in MOF as follows.

The most important parts of the definition are the state signature and the transition rules. The state signature defines the state ontology used by the service together with the definition of the types of modes the concepts and relations may have which describes the service’s and the requestor’s rights over the instances. The transition rules express changes of states by changing the set of instances [8].

Applying different transition rules, the choreography evolves from the initial state which technically is the same as the state signature, if not otherwise specified, to the final state, going through several intermediate states. In the final state, no further updates based on transition rules can be applied.

The orchestration has a similar definition as the choreography, with the main difference being that the choreography considers two participants in a conversation, while the orchestration is a description of the cooperation of multiple participants.
7.2.3 Goals

A goal specifies the objectives that a client may have when consulting a Web service, describing aspects related to user desires with respect to the requested functionality and behavior. Ontologies are used as the semantically defined terminology for goal specification. Goals model the user’s view in the Web service usage process, and therefore are a separate top-level entity in WSMO. The goal class is described in Listing 7.12.

7.2.4 Mediators

Mediators describe elements that resolve interoperability problems between different elements, e.g., between two ontologies or two services. Mediators are a core element of WSMO and aim to resolve heterogeneity problems at the data, process and protocol levels. The definition of a mediator using the MOF notation is depicted in Listing 7.13.

Like all WSMO elements, a mediator can define a set of annotations using the hasAnnotations statement. Furthermore, the terminology needed from other ontologies to define this mediator can be imported using the importsOntology statement. The source component of a mediator defines the resources for which the heterogeneities are resolved, while the target component defines the resources that receives these mediated source components. A mediation service can be used to define the facility applied for performing the mediation using the hasMediationService statement. This service may explicitly link to a Web service description or may link to a Goal describing the functionality needed, which can then be resolved to a Web service at runtime using service discovery. There are four types of mediators within the WSMO metamodel.
7.2 Technical Solution

Listing 7.14  ooMediator definition

Listing 7.15  ggMediator definition

7.2.4.1 Ontology-to-Ontology Mediators

Ontology-to-Ontology Mediators (ooMediators), as defined in Listing 7.14, provide a mechanism by which mismatches between two or more ontologies can be resolved. The source component of an ooMediator is an ontology or another ooMediator, and the source defines the resources for which mismatches will be resolved by the mediator. The target of an ooMediator can be an ontology, goal, Web service or a mediator, and the target defines the target component which will receive the results of the mediation.

As described in Sect. 7.2.1, ooMediators are used across all WSMO top-level elements within the usesMediator statement in order to import the terminology required by a resource description whenever there are mismatches between the ontologies to be used.

7.2.4.2 Goal-to-Goal Mediators

Goal-to-Goal Mediators (ggMediators), as defined in Listing 7.15, connect goals to another, allowing for relationships between different goals to be specified. For instance, a ggMediator can be used to specify that one goal is equivalent to another goal, and that the source goal is a refinement of the target.

The source of a ggMediator can be a goal or another ggMediator, which is also true for the target of a ggMediator. By specifying that a ggMediator is the source or target, one can build chains of mediators.

7.2.4.3 Web Service-to-Goal Mediators

Web Service-to-Goal Mediators (wgMediators), defined in Listing 7.16, or Goal-to-Web Service Mediators, provide a mechanism for expressing relationships between Web services and goals. Primarily, wgMediators are used to pre-link services to goals, or to cache the results of previously performed discovery actions. For example, a wgMediator may be used to express that a given Goal can be totally or partially fulfilled by a given Web service.
7.2.4.4 Web Service-to-Web Service Mediators

Web Service-to-Web Service Mediators (wwMediators), defined in Listing 7.17, provide a mechanism for expressing relationships between Web services. These relationships could include stating that two Web services provide equivalent functionality, a group of Web services provides equivalent functionality to another Web service, or that one Web service is a refinement of another.

wwMediators can also be used to establish interoperability between Web services in cases where they would otherwise be not interoperable, i.e., a wwMediator could be used to specify a mediation between the choreography of two Web services, where mediation can involve the data, protocol or process levels.

7.3 Extensions

In this section, we briefly introduce a number of extensions for modeling services that were inspired by the WSMO approach. One characteristic that is common to all classical approaches for Semantic Web Services, including WSMO, is the top-down modeling. However, top-down modeling decouples the Semantic Web Service model from the actual syntactic description which contains where the service is located and how exactly it can be invoked. Another approach that gained more momentum is the bottom-up approach for modeling that is adopted by a number of extensions for modeling services.

Current extensions to WSMO conceptual model include those that are following the bottom-up path such as WSMO-Lite, a lightweight ontology for service semantics, Semantic Annotations for WSDL (SAWSDL) that provides an semantic annotation mechanisms allowing the specification of pointers to semantic concepts from within a syntactic description of a WSDL service and MicroWSMO that basically has the same purpose as SAWSDL but for RESTful services. SAWSDL and MicroWSMO build on top of syntactic representations formalisms, namely WSDL and hRESTS. The approaches mentioned above are known as lightweight for services given the fact that they “add a bit of semantics” on top of existing non-semantic
standards. More details about each of the mentioned lightweight approaches for Semantic Web Services are provided in Chap. 12. In the rest of this section, we briefly summarize the main features of the lightweight approaches.

WSMO-Lite is a simple, yet powerful ontology that can be used to model all important aspects of a service. It distinguishes between four type of semantics, i.e., functional, non-functional, behavioral, and information model. Furthermore, WSMO-Lite provides an RDF Schema ontology to express the four types of semantics mentioned previously. The WSMO-Lite ontology can be used to semantically annotate both WSDL-based and RESTful-based services. In the former case, the approach used to provide hooks for attaching semantics is SAWSDL, while the later is MicroWSMO.

SAWSDL is a set of extensions for WSDL. The two major constructs provided by SAWSDL are **model references** that can be used to point to semantic concepts and **schema mappings** that can be used to specify data transformations between the XML data structure of messages and the associated semantic model. SAWSDL by itself, however, does not specify any actual types of semantics.

MicroWSMO can be seen as a direct realization of SAWSDL annotations over hRESTS. As SAWSDL, MicroWSMO provides a set of elements that can be used to point to semantic descriptions. These include **model**, **lowering** and **lifting** which can be associated with the underlying hREST description.

### 7.4 Illustration by a Larger Example

In this section, we use the “Virtual Traveling Agency” scenario introduced in Chap. 4 to exemplify how services can be semantically annotated using the WSMO approach. In this scenario, a customer uses the Virtual Traveling Agency (VTA) application to access eTourism services. The reminder of this section illustrates how a concrete service offers online train booking tickets, as well as how related ontologies, goals, and mediators can be modeled using WSMO. We use the Web Service Modeling Language (WSML) to represent all of these elements. The modeling example presented here is partially based on content available in [9].

#### 7.4.1 Ontologies

We use the following domain ontologies: **International Train Ticket, Temporal, Locative, Price, Quality, Provider, Measures**.

Six of the ontologies mentioned above, namely **Temporal, Locative, Price, Quality, Provider, Measures**, are generic ontologies containing non-functional properties models. These ontologies have been defined in [10].
The “International Train Ticket” ontology defines a train trip and the surrounding concepts shown in Listing 7.18. The definition of the ontology is based on the travel itinerary ontology from the DAML ontology library. This ontology captures information about travel itineraries for trips by plane. Our ontology reuses the itinerary and flight concepts and adapts them to define train trips, also introducing new concepts such as the train station. The International Train Ticket ontology also makes use of the person ontology which defines a subset of vCard. The person concept is used to define the passenger information for an itinerary.

### 7.4.2 Goals

Goals denote what a user wants as the result of a Web service. To exemplify how goals are modeled in WSMO, we define in Listing 7.19 a generic goal for buying a ticket for any kind of trip. Please note that in WSMO, goals can be defined on different levels of granularity. Through ggMediators, new, more specific goals can be created out of generic existing goals. Generic goals can also be thought of as being pre-defined in a specific application context, from which concrete Goals can be generated.

### 7.4.3 Web Services

For exemplification, let us consider an end-user service (a service with which the user interacts) for purchasing international train tickets offered by the Austrian national train operator OEBB; this Web service can be composed of other Web services, each for the search and purchasing facility of international train tickets. This setting allows for the correct modeling of 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.

A Web service capability in WSMO is described by pre- and postconditions, assumptions and effects, as defined in [7]. Listing 7.20 shows the OEBB Web service. The capability of the OEBB Web service includes:

- **Precondition**
  This input has to include information about the buyer, a train trip that a ticket is to be purchased for, and information on the passenger for whom the ticket shall be valid. To be a valid input, the following restrictions are defined for the trip: the start and end locations are restricted to stations in Austria or Germany;

---

wsmlVariant _"http://www.wsmo.org/wsml/wsml−syntax/wsml−rule"

namespace _"http://www.wsmo.org/ontologies/trainConnection",
dc _"http://purl.org/dc/elements/1.1#",
prs _"http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/resources/owlPersonMediator.wsml"
xsd _"http://www.w3.org/2001/XMLSchema#",
wsml _"http://www.wsmo.org/wsml/wsml−syntax#",
loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
temp _"http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
qua _"http://www.wsmo.org/ontologies/nfp/qualityNFPOntology#",
po _"http://www.wsmo.org/ontologies/nfp/providerNFPOntology#",
prc _"http://www.wsmo.org/ontologies/nfp/priceNFPOntology#",
mes _"http://www.wsmo.org/ontologies/nfp/measuresNFPOntology#" }

ontology _"http://www.wsmo.org/ontologies/trainConnection"
annotations
dc#title hasValue "International Train Ticket Ontology"
dc#creator hasValue "STI Innsbruck"
dc#subject hasValues {"Train", "Itinerary", "Train Connection", "Ticket"}
dc#description hasValue "Domain Ontology International Train Ticket"
dc#format hasValue "text/html"
dc#identifier hasValue _"http://www.wsmo.org/ontologies/trainConnection"
dc:contributor hasValues {"Michael Stollberg", "Ruben Lara",
"Holger Lausen", "Ioan Toma"}
dc#language hasValue "en−US"
wsml#version hasValue "Revision : 2.0"
endAnnotations

importsOntology _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
_ "http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#"

usedMediators
_ "http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/resources/owlPersonMediator.wsml"

concept station subConceptOf loc#GeoLocation
annotations
dc#description hasValue "Train station"
endAnnotations
code ofType _string
annotations
dc#description hasValue "Code of the station"
endAnnotations
borderToCountry ofType loc#Border
annotations
dc#description hasValue "For stations located at the border"
endAnnotations

cursor ticket
annotations
dc#description hasValue "a ticket for an itinerary"
endAnnotations
itinerary ofType itinerary
provider ofType po#Provider
price ofType prc#AbsoulutePrice

concept itinerary
annotations
dc#description hasValue "An itinerary between two locations"
endAnnotations
passenger ofType prs#person

Listing 7.18  Domain ontology “International Train Ticket”
the departure date for the trip has to be after the current date; and the payment method of the buyer has to be a credit card.

- **Assumption**
  The credit card submitted as input has to be valid (not expired).

- **Postcondition**
  The service returns a purchase for train tickets valid in Austria and Germany by the OEBB as the provider, with payment by credit card only.

- **Effect**
wsmlVariant "http://www.wsmo.org/wsml/wsml−syntax/wsml−rule"

namespace {
  __"http://www.wsmo.org/example/VTA/goals/GeneralTrainTrip#",
  _dc __"http://purl.org/dc/elements/1.1#",
  _tc __"http://www.wsmo.org/ontologies/trainConnection#",
  _temp __"http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
  _loc __"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
  _pay __"http://www.wsmo.org/ontologies/nfp/paymentNFPOntology#",
  _xsd __"http://www.w3.org/2001/XMLSchema#",
  _wsml __"http://www.wsmo.org/wsml/wsml−syntax#",
  _po __"http://www.wsmo.org/ontologies/purchase#"
}

goal "http://www.wsmo.org/example/VTA/goals/GeneralTrainTrip.wsml"

  annotations
    _dc#title hasValue "Buying a ticket online"
    _dc#subject hasValues ("Tickets", "Online Ticket Booking", "trip")
    _dc#creator hasValue "STI Innsbruck"
    _dc#description hasValue "Express the goal of buying a ticket for a trip"
    _dc#contributor hasValues ("Michael Stollberg", "Ruben Lara", "Holger Lausen", "Ioan Toma")
    _dc#format hasValue "text/html"
    _dc#language hasValue "en−US"
    _dc#coverage hasValues (_tc#austria, _tc#germany)
    _dc#version hasValue "Revision : 2.0"
  endAnnotations

  importsOntology { __"http://www.wsmo.org/ontologies/trainConnection#",
    __"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
    __"http://www.wsmo.org/ontologies/purchase#"
}

  postcondition

    axiom purchasingTicketForTrip
      annotations
        _dc#description hasValue "This goal expresses the general desire of buying a ticket for any kind of itinerary."
      endAnnotations

        definedBy
          (?Purchase memberOf _po#purchase[
            _po#purchaseorder hasValue ?Purchaseorder,
            _po#buyer hasValue ?Buyer
          ] and
          ?Buyer memberOf _po#buyer and
          ?Purchaseorder memberOf _po#purchaseOrder[
            _po#product hasValues (?Product),
            _po#payment hasValue ?PaymentMethod
          ] and
          ?PaymentMethod memberOf _po#paymentMethod and
          ?Product memberOf _po#product[
            _po#item hasValues (?Ticket)
          ] and
          ?Ticket memberOf _tc#ticket[
            _tc#itinerary hasValue ?Itinerary
          ] and
          ?Itinerary memberOf _tc#itinerary[
            _tc#passenger hasValue ?Passenger,
            _tc#trip hasValue ?Trip
          ]
          )
effect
axiom havingTradeForTrip
annotations
  dc#description hasValue "The goal effect is to get the purchased ticket delivered to the buyer."
endAnnotations
definedBy
  (?Delivery memberOf po#delivery[
    po#deliveryItem hasValues {?Product},
    po#receiver hasValue ?Buyer
  ] and
  ?Product memberOf po#product[
    po#item hasValues {?Ticket}
  ] and
  ?Buyer memberOf po#buyer and
  ?Ticket memberOf tc#ticket
  ).

Listing 7.19 (Continued)

The sold ticket is delivered to the buyer’s shipping address, either by a drop ship carrier or via online delivery.

7.4.4 Mediators

As mentioned in Sect. 7.2.4, mediators play a significant role in the WSMO modeling as they are responsible for solving the heterogeneity problem. WSMO defines four types of mediators, namely ooMediators, ggMediators, wgMediators and ww-Mediators. Listing 7.21 contains a description of an ooMediator that imports other ontologies or ooMediators into WSMO entities and resolves possible terminology mismatches. If no mismatch has to be resolved, the syntactical simplification “importOntologies” can be used.

7.5 Summary

Semantic Web Services are one of the most promising research directions to improve the integration of applications within and across enterprise boundaries. In this context, this chapter provided an overview of one of the most important approaches to Semantic Web Services, namely the Web Service Modeling Ontology. We argue that, in order for Semantic Web Services to succeed, a fully fledged framework needs to be provided: starting with a conceptual model, continuing with a formal
wsmlVariant "http://www.wsmo.org/wsml/wsml−syntax/wsml−rule"

namespace {
  _"http://www.wsmo.org/example/VTA/services/OEBBService.wsml",
  dc "http://purl.org/dc/elements/1.1#",
  tc _"http://www.wsmo.org/ontologies/trainConnection#",
  temp _"http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
  loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
  pay _"http://www.wsmo.org/ontologies/nfp/paymentNFPOntology#",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  wsml _"http://www.wsmo.org/wsml/wsml−syntax#",
  po _"http://www.wsmo.org/ontologies/purchase#"}

webservice
"http://www.wsmo.org/example/VTA/services/OEBBService.wsml"

annotations
dc#title hasValue "OEBB Online Ticket Booking Web Service"
dc#creator hasValue "STI Innsbruck"
dc#description hasValue "web service for booking online train tickets for Austria and Germany"
dc#contributor hasValues {"Michael Stollberg", "Ruben Lara", "Holger Lausen", "Ioan Toma"}
dc#format hasValue "text/html"
dc#language hasValue "en−US"
dc#coverage hasValues {tc#austria, tc#germany}
dc#version hasValue "Revision: 2.0"
endAnnotations

importsOntology {"http://www.wsmo.org/ontologies/trainConnection#",
  _"http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
  _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
  _"http://www.wsmo.org/ontologies/nfp/paymentNFPOntology#",
  _"http://www.wsmo.org/ontologies/purchase#"}

capability oebbWSCapability

precondition

axiom oebbWSprecondition
  annotations
dc#description hasValue "The oebbWSprecondition puts the following conditions on the input: it has to include a buyer with a billTo and a shipTo address, and credit card as a paymentMethod, and trip with the start− and end− location have to be in Austria or in Germany, and the departure date has to be later than the current date."
endAnnotations

definedBy
  Buyer memberOf po:buyer
    po#billToAddress hasValue ?BuyerBilltoAddress,
    po#shipToAddress hasValue ?BuyerShiptoAddress,
    po#hasPaymentMethod hasValues {?BuyerPaymentMethod} )
  and
  ?BuyerBilltoAddress memberOf loc#Address and
  ?BuyerShiptoAddress memberOf loc#Address and
  ?BuyerPaymentMethod memberOf pay#CreditCard and
  ?Trip memberOf tc#trainTrip[
    tc#start hasValue ?Start,
    tc#end hasValue ?End,
    tc#departure hasValue ?Departure
  ]
)

Listing 7.20  OEBB web service for booking online train tickets for Austria and Germany
postcondition

axiom oebbWSpostcondition

annotations
dc#description hasValue "the output of the service is a purchase for a
    ticket for train trips wherefore
    the start — and end — location have to be in Austria or in Germany and
    the departure date has to be later than the current Date."

endAnnotations

definedBy

forAll ?Purchase, ?Seller, ?OEBBContactInformation, ?PurchaseOrder,
    ?PaymentMethod, ?Product, ?Ticket,

{ (?Purchase memberOf po#purchase[  
    purchaseOrder hasValue ?PurchaseOrder,  
    seller hasValue ?Seller

] and
?Seller memberOf po#seller[  
    contactInformation hasValue ?OEBBContactInformation,  
    acceptsPaymentMethod hasValues (?PaymentMethod)

] and
?OEBBContactInformation memberOf po#contactInformation[  
    name hasValue "Oesterreichische Bundesbahn",  
    emailAddress hasValue "office@oebb.at",  
    physicalAddress hasValue ?OEBBAddress

] and
?OEBBAddress memberOf loc#Address[  
    streetAddress hasValue "Hauptfrachtenbahnhof 4",  
    city hasValue innsbruck,  
    country hasValue austria

] and
?PurchaseOrder memberOf po#purchaseOrder[  
    product hasValues (?Product),  
    payment hasValue ?PaymentMethod

] and
?PaymentMethod memberOf pay#CreditCard and
?Product memberOf po#product[  
    item hasValues (?Ticket)

] and
?Ticket memberOf tc#ticket[  
    itinerary hasValue ?Itinerary

] and
?Itinerary memberOf tc#Itinerary[  
    trip 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
temp#after(?Departure, currentDate)

).

Listing 7.20 (Continued)
Listing 7.20  (Continued)

language to provides formal syntax and semantics (based on different logics in order to provide different levels of logical expressiveness) for the conceptual model, and ending with an execution environment that glues together all components that use the language for performing various tasks that would eventually enable automation of the service. The WSMO approach tackles, in a unifying manner, all the aspects of such a framework, and potentially provides the conceptual basis and the technical means for realizing Semantic Web Services: it defines a conceptual model
Listing 7.21  OO-Mediator “importing the OWL address ontology to the location ontology”

(WSMO) for defining the basic concepts of Semantic Web Services, a formal family of languages (WSML) (see Chap. 8) which provides a formal syntax and semantics for WSMO by offering different variants based on different logics in order to provide different levels of logical expressiveness (thus allowing different trade-offs between expressivity and computability), and an execution environment (WSMX) (see Chap. 9) which provides a reference implementation for WSMO and interoperability of Semantic Web Services. This chapter has focused on the WSMO conceptual model. The following two chapters discuss the other constituents of the approach, namely the formal family of languages, WSML, and the execution environment, WSMX. To conclude, the clean conceptual model and the design principles of the Web that underly the approach, make WSMO a key contribution towards realizing the vision of Semantic Web Services and Semantically Enabled Service-oriented Architectures.

7.6 Exercises

Exercise 1  Extend the WSMO ontology available in Listing 7.18 with other elements that capture knowledge about bus transportation. You should add at least
one example of concept, relation, function, concept instance, relation instance, and axiom.

**Exercise 2**  Consider a bus transportation company as one company that provides tourism services as part of the VTA scenario. Create a WSMO description for the bus transportation service. The resulting description should include description of the capability, including pre-conditions, post-conditions, assumptions, and effects. Explain in your own words what the difference between these elements is using your example.

**Exercise 3**  Discuss differences between a service and a Web service in the WMSO approach. Give an example of service and one of Web service using in the hotel booking sub-domain of the VTA scenario.

**References**