



D4.2v01 Formal Comparison WSMO/OWL-S

DERI Working Draft 15 March 2004

This version:

<http://www.wsmo.org/2004/d4/d4.2/v01/20040315/>

Latest version:

<http://www.wsmo.org/2004/d4/d4.2v01/>

Previous version:

<http://www.wsmo.org/2004/d4/d4.2/v01/20040308/>

Editors:

Axel Polleres
Ruben Lara

Authors:

Axel Polleres
Ruben Lara
Dumitru Roman

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

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

Table of contents

[1. Introduction](#)

[2. Use Cases in OWL-S](#)

[2.1 The *BravoAir* Web Service](#)

[2.2 The *Congo* Web Services](#)

[3. From OWL-S to WSMO](#)

[3.1 The *BravoAir* Web Service in WSMO-Standard](#)

[3.2 The *Congo* Web Services in WSMO-Standard](#)

[4. Use Cases in WSMO](#)

[5. From WSMO to OWL-S](#)

[6. What is Missing?](#)

[7. Discussion and Open Points](#)

[8. Conclusions and Further Work](#)

[Acknowledgement](#)

1. Introduction

In this document it is our aim to formally compare the Web Service Modeling Ontology (WSMO) [[Roman et al., 2004](#)] and the Semantic Web Services ontology OWL-S [[Martin et al., 2004](#)] by hand of some well-known examples. For this purpose we consider examples

provided by the OWL-S Coalition [[Martin et al., 2004b](#)] and some preliminary WSMO use cases defined in [[Stollberg & Arroyo, 2004](#)]. WSMO and OWL-S are the most salient initiatives to describe Semantic Web Services, aiming at describing the various aspects related to Semantic Web Services in order to enable the automation of Web Service discovery, composition, interoperation and invocation.

By trying to match the chosen examples in either formalization manually we want to achieve a better understanding how far one can get using either approach, which are the disjoint features, and which compatibilities exist. In this sense, the present document shall refine and substantiate the conceptual comparison we provided in [[Lara et al., 2004](#)]

One crucial point for use case specifications is that the authors normally tend to take only into account what is expressible in their own particular formalism. In trying to deploy WSMO and OWL-S, respectively, in Use case scenarios originally designed for the other formalism, this document should be a helpful starting point in detecting mismatches and flaws in both approaches. Eventually this shall lead to a fruitful combination of both or in the conclusion that one simply subsumes the other, respectively. We plan to extend the results of this comparison in the following in the direction of an automatic mapping tool for (syntactic fragments of) web service descriptions in WSMO or OWL-S in either direction.

The comparison presented in this document exposes the relation between the latest stable versions of WSMO and OWL-S, i.e WSMO-Standard v0.2 [[Roman et al., 2004](#)] and OWL-S v1.0 [[OWL-S, 2004](#)]. The purpose of the comparison is to highlight the conceptual overlaps and conceptual differences between the two specifications. This document is based on previous results in [[Lara et al., 2004](#)]. A previous comparison between DAML-S and WSMF is presented. [[Flett, 2002](#)].

[Section 2](#) presents some use case examples specified in OWL-S taken from [[Martin et al., 2004b](#)] and we analyze which particular features of OWL-S are addressed in the particular examples and which are not. In the following [Section 3](#) we investigate how far we can get in formulating these examples in WSMO highlighting differences between the two formalisms, difficulties, and general patterns which can be recognized from these manual translations. [Section 4](#) and [Section 5](#) will cover the other way from WSMO to OWL based on the WSMO use case examples from [[Stollberg & Arroyo, 2004](#)]. In [Section 6](#) we elaborate a more on the lack of a logical formalism for representing conditions, rules and variables in OWL which hampers the specification of complex services in OWL-S. Since the concrete formalism to be chosen for this purpose is still left open in the current version of OWL as, we will discuss the most likely candidates DRS and SWRL briefly and compare their expressive and syntactic features with F-Logic, the proposed logical formalism underlying WSMO. In order to exemplify the difference we again chose some examples from ???(Rules examples from: ORL (=SWRL) copy paste by Ian Horrocks, reference?) and [[Stollberg & Arroyo, 2004](#)] We will summarize and discuss our findings in [Section 7](#) pinpointing to open questions. Finally, we conclude with an outlook to further work in [Section 8](#).

2. Use Cases in OWL-S

In this section we describe the examples that accompany the version 1.0 of OWL-S [[Martin et al., 2004b](#)]. Two examples are given by the OWL-S Coalition in order to illustrate the use of the service ontology, namely:

- **Bravo Air** Web Service, an airline ticketing service.
- **Congo** Web Service, a book selling service.

In the next subsections, we summarize the most important aspects of the examples. For more details (including the OWL files for the examples), we refer the reader to [[Martin et al., 2004b](#)].

2.1 The *BravoAir* Web Service

This example models a Web Service for gathering information about flights and booking the selected flight.

Main classes:

- BravoAir_ReservationAgent (Service),
- Profile_BravoAir_ReservationAgent (ServiceProfile),
- BravoAir_ReservationAgent_ProcessModel (ProcessModel), and
- Grounding_BravoAir_ReservationAgent (Grounding).

Profile:

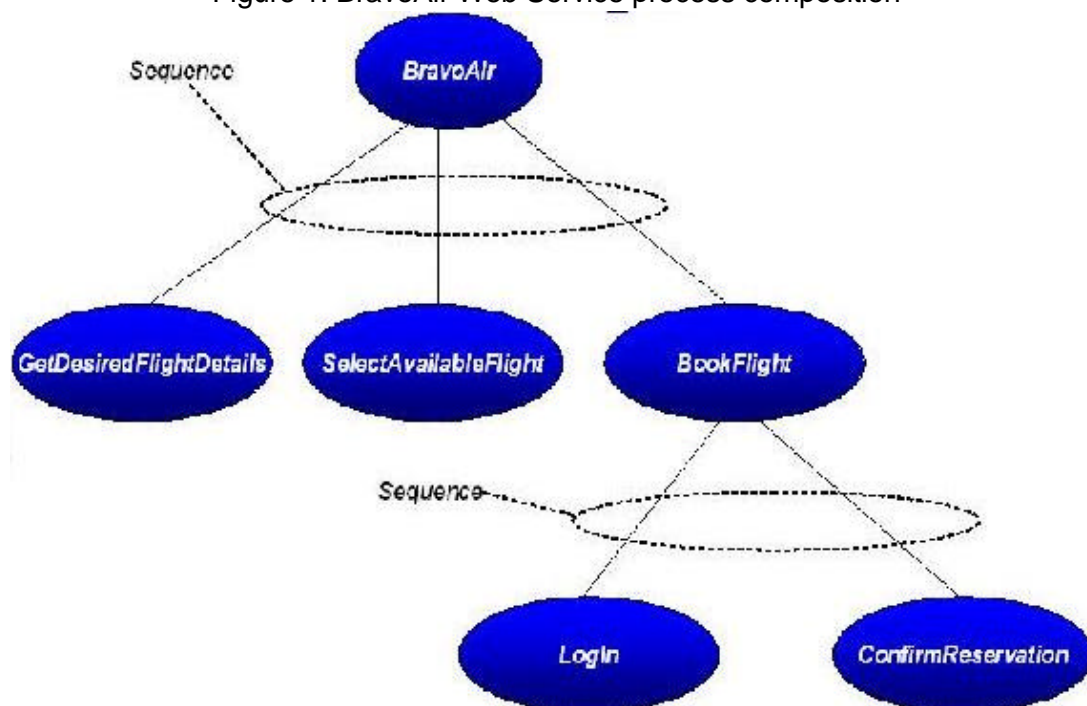
- Non-functional properties
 - Textual description: the service provides flight reservations based on the specification of a flight request. This typically involves a departure airport, an arrival airport, a departure date, and if a return trip is required, a return date. If the desired flight is available, an itinerary and reservation number will be returned.
 - Contact information: two contacts specified, a reservation department and the sale representative John Doe.
 - Geographical radius: Limited to the USA.
 - Quality rating: GoodRating in the SomeRating rating system.
 - Service category: value "Airline reservation services" and code 561599 in the NAICS categorization, and value "Travel agent" and code 90121500 in the UNSPC categorization.
- Hierarchy: The service is defined as an AirlineTicketing service in the profile hierarchy, a subprofile of E-Commerce. This hierarchy is defined in [ProfileHierarchy.owl](#).
- IOPEs (all of them defined in the ProcessModel and referenced in the Profile):
 - Inputs: DepartureAirport_In, ArrivalAirport_In, OutboundDate_In, InboundDate_In, RoundTrip_In, PreferredFlightItinerary_In, AvailableFlightItineraryList_Out, AcctName_In, Password_In, ReservationID_In, Confirm_In.
 - Outputs: PreferredFlightItinerary_Out, AcctName_Out, ReservationID_Out.
 - Effects: HaveSeat.

Process model:

- Composition: The BravoAir Process is a composite process. It is composed of a sequence whose components are 2 atomic processes, GetDesiredFlightDetails and SelectAvailableFlight, and a composite process, BookFlight. The process BookFlight is again a composite process. It is composed of a sequence whose components are 2 atomic processes, LogIn and ConfirmReservation, as shown in Figure 1 below.
- Atomic processes:
 - GetDesiredFlightDetails gets details such as airports, preferred time and round trip.
 - Inputs: DepartureAirport_In (Airport), ArrivalAirport_In (Airport), OutboundDate_In (FlightDate), InboundDate_In (FlightDate), RoundTrip_In (RoundTrip).
 - SelectAvailableFlight gets user's preferred flight choice from available itineraries.
 - Inputs: PreferredFlightItinerary_In (FlightItinerary).
 - Outputs: AvailableFlightItineraryList_Out (FlightItineraryList, unconditional output).
 - LogIn gets user details.
 - Inputs: AcctName_In (AcctName), Password_In (Password).

- ConfirmReservation confirms selected reservation.
 - Inputs: ReservationID_In (ReservationNumber), Confirm_In (Confirmation).
 - Outputs: PreferredFlightItinerary_Out (FlightItinerary, unconditional output), AcctName_Out (AcctName, unconditional output), ReservationID_Out (ReservationNumber, unconditional output).
 - Effects: HaveSeat (HaveFlightSeat, unconditional effect).
- It is important to notice that in this example no complex control constructs (involving conditions) are used, and all the outputs and effects are unconditional. Therefore, the BravoAir Web Service does not illustrate the use of conditions in OWL-S. It can also be seen that no data flow is defined between the different processes, so it is assumed that the user provides the input information for each process. Furthermore, concepts such as FlightDate are not defined in the example, and only a stub is defined. We can conclude that the process modelling in the example is incomplete and under-defined, and that the description given so far makes it difficult to figure out what is the sequence of information the user has to send to the service.

Figure 1. BravoAir Web Service_ process composition



Grounding (WSDL grounding provided for the example):

- Grounding for atomic processes: WsdGrounding_GetDesiredFlightDetails, WsdGrounding_SelectAvailableFlight, WsdGrounding_LogIn, WsdGrounding_ConfirmReservation.

Each atomic process is mapped to a WSDL operation (defined in a WSDL file), each OWL-S input message to the correspondent message parts in the WSDL definition, and each OWL-S output message to the correspondent message parts in the WSDL definition. A WSDL file is also given, defining the mappings from the WSDL interface to the OWL-S description (operations to atomic processes, and message parts to the corresponding OWL-S inputs and outputs).

2.2 The Congo Web Services

This example models two Web Services that offer the electronic purchase of books: the ExpressCongoBuy service and the FullCongoBuy service. We will first present the express

version of the service, a one-step form, CongoBuy, with the service treated as atomic; i.e., no interactions between buying and selling agents are required, apart from invocation of the service and receipt of its outputs by the buyer. Given certain inputs and preconditions, the service provides certain outputs and has specific effects [\[Martin et al., 2004b\]](#).

Main classes:

- ExpressCongoBuyService (Service),
- Profile_Congo_BookBuying_Service (ServiceProfile),
- ExpressCongoBuyProcessModel (ProcessModel), and
- CongoBuyGrounding (Grounding).

Profile:

- Non-functional properties
 - Service name: Congo_BookBuying_Agent.
 - Textual description: This agentified service provides the opportunity to browse a book selling site and buy books there.
 - Contact information: the contact information gives the name, phone, fax, e-mail, physical address and webURL of the service representative.
 - Quality rating: GoodRating in the SomeRating rating system.
- Hierarchy: The service is defined as a BookSelling service in the profile hierarchy, a subprofile of E-Commerce.
- IOPEs (all of them defined in the ProcessModel and referenced in the Profile):
 - Inputs: ExpressCongoBuyBookISBN, ExpressCongoBuySignInInfo.
 - Preconditions: AcctExists, CreditExists.
 - Outputs: ExpressCongoOrderShippedOutput, ExpressCongoOutOfStockOutput.
 - Effects: ExpressCongoOrderShippedEffect.

Process model:

- Composition: The ExpressCongoBuy process is an atomic process.
- Atomic processes:
 - ExpressCongoBuy.
 - Inputs: ExpressCongoBuyBookISBN (string), CongoBuySignInInfo (SignInData).
 - Preconditions: AcctExists, CreditExists.
 - Outputs: CongoOrderShippedOutput (OrderShippedOutput, condition: BookInStock), CongoOutOfStockOutput (BookOutOfStockOutput, condition: BookOutOfStock).
 - Effects: CongoOrderShippedEffect (OrderShippedEffect, condition: BookInStock).
- In this example, conditions are needed to express preconditions, conditional outputs, and conditional effects. However, these conditions are only stubs. As stated in the documentation of the example, there is no official specification as to how conditions are to be expressed in OWL-S.

Grounding (WSDL grounding provided for the example):

- Grounding for atomic processes: ExpressCongoBuyGrounding.

As in the BravoAir example, each atomic process is mapped to a WSDL operation, each OWL-S input message to the correspondent message parts in the WSDL definition, and each OWL-S output message to the correspondent message parts in the WSDL definition. A

WSDL file is also given, defining the mappings from the WSDL interface to the OWL-S description.

The **Full version** of the Congo example shows its composition from its component services. The full-fledged version of the service, FullCongoBuy, includes an arrangement of subprocesses LocateBook, PutInCart, SignIn, CreateAcct, CreateProfile, LoadProfile, SpecifyDeliveryDetails, and FinalizeBuy each, with its own specification of inputs and outputs [Martin et al., 2004b].

Main classes:

- FullCongoBuyService (Service),
- Profile_Full_Congo_BookBuying_Service [1] (ServiceProfile),
- FullCongoBuyProcessModel (ProcessModel), and
- FullCongoBuyGrounding (Grounding).

Profile:

- Non-functional properties
 - Service name: Congo_BookBuying_Agent.
 - Textual description: This agentified service provides the opportunity to browse a book selling site and buy books there.
 - Contact information: the contact information gives the name, phone, fax, e-mail, physical address and webURL of the service representative.
 - Quality rating: GoodRating in the SomeRating rating system.
 - Delivery region (from the Profile Hierarchy): United States.
- Hierarchy: The service is defined as an BookSelling service in the profile hierarchy, a subprofile of E-Commerce.
- IOPEs (all of them defined in the ProcessModel and referenced in the Profile):
 - Inputs: BookName, PutInCart, SignInInfo, DeliveryAddress, PackagingSelection, DeliveryTypeSelection, CreditCardNumber, SpecifyPaymentMethodCreditCardType, CreditCardExpirationDate, CreateAcctInfo.
 - Preconditions: ProfileExists.
 - Outputs: LocateBookOutput, CreateAcctOutput.

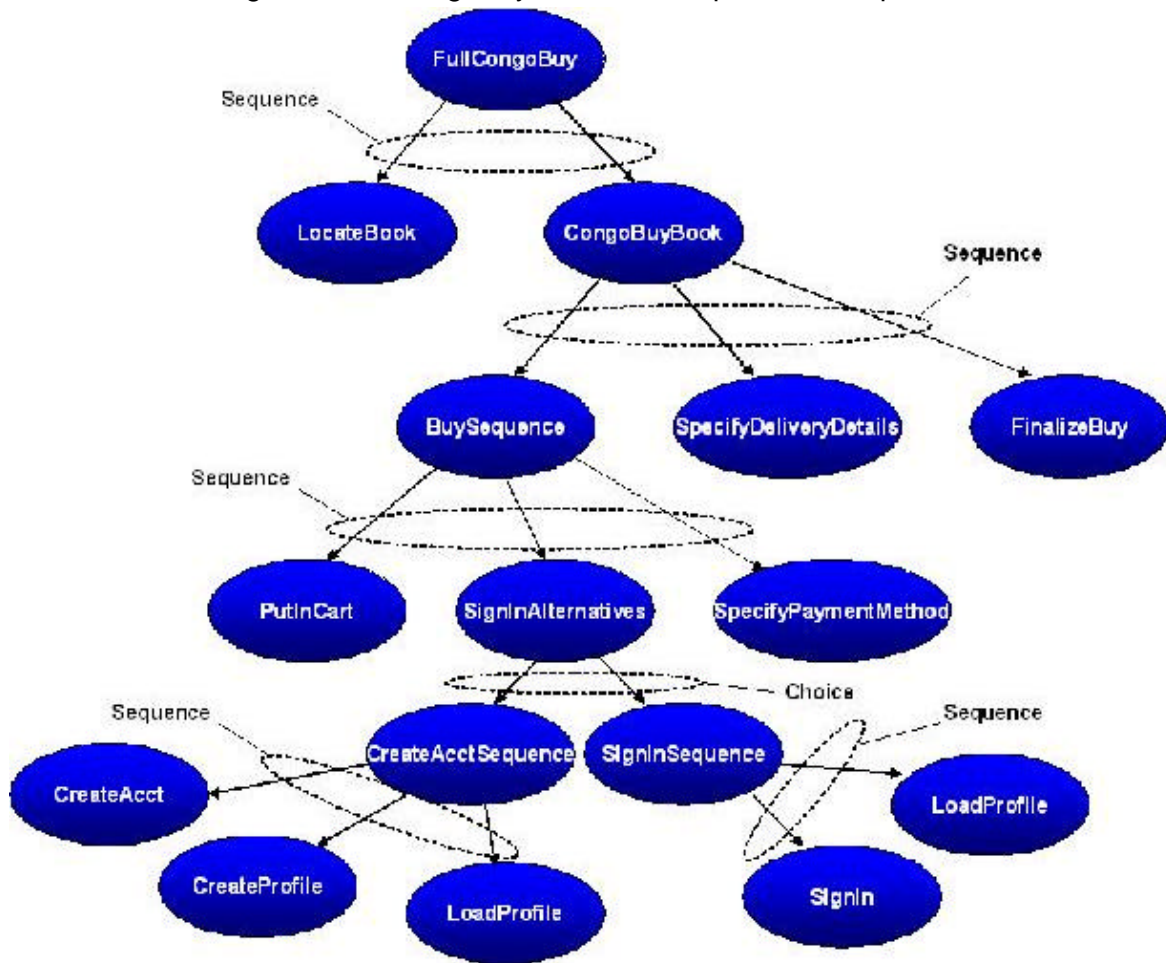
Process model:

- Composition: The FullCongoBuy process is a composite process, composed of the sequence of locating a book (atomic process) and buying a book (composite process). The decomposition of the FullCongoBuy can be seen in Figure 2.
- Processes:
 - FullCongoBuy.
 - Composition: Sequence of LocateBook and CongoBuyBook.
 - Inputs (derived from the atomic processes): FullCongoBuyBookName (string), FullCongoBuySignInInfo (SingInData), FullCongoBuyCreateAcctInfo (AcctInfo), FullCongoBuyCreditCardNumber (number, xsd:decimal), FullCongoBuyCreditCardType (CreditCardType), FullCongoBuyCreditCardExpirationDate (string), FullCongoBuyCreditCardDeliveryAddress (string), FullCongoBuyPackagingSelection (string), FullCongoBuyDeliveryTypeSelection (DeliveryType).
 - Outputs (derived from the atomic processes): FullCongoBuyBookISBNOutput (string, condition: InCatalogueBookInstance), FullCongoBuyCreateAcctOutput

- (CreateAcctOutputType, unconditional).
- LocateBook.
 - Composition: Atomic.
 - Inputs: BookName (string).
 - Outputs: LocateBookOutput (string, condition: InCatalogueBookInstance).
- CongoBuyBook.
 - Composition: Sequence of BuySequence, SpecifyDeliveryDetails and FinalizeBuy.
 - Inputs (derived from the atomic processes): CongoBuyBookISBN (string), CongoBuyBookCreateAcctInfo (AcctInfo), CongoBuyBookSignInInfo (SignInData), CongoBuyBookCreditCardNumber (string), CongoBuyBookCreditCardType (CreditCardType), CongoBuyBookCreditCardExpirationDate (Instant), CongoBuyBookDeliveryAddress (string), CongoBuyBookPackagingSelection (PackagingType), CongoBuyBookDeliveryTypeSelection (DeliveryType).
 - Outputs (derived from the atomic processes): CongoBuyBookCreateAcctOutput (CreateAcctOutputType, unconditional).
- BuySequence.
 - Composition: Sequence of PutInCart, SignInAlternatives and SpecifyPaymentMethod.
 - Inputs (derived from the atomic processes): BuySequenceBookISBN (string), BuySequenceCreateAcctInfo (AcctInfo), BuySequenceSignInInfo (SignInData), BuySequenceCreditCardNumber (number, xsd:decimal), BuySequenceCreditCardType (CreditCardType), BuySequenceCreditCardExpirationDate (instant).
 - Outputs (derived from the atomic processes): BuySequenceCreateAcctOutput (CreateAcctOutputType, unconditional).
- SpecifyDeliveryDetails.
 - Composition: Atomic.
 - Inputs: DeliveryAddress (string), PackagingSelection (PackagingType), DeliveryTypeSelection (DeliveryType).
- FinalizeBuy.
 - Composition: Atomic.
- PutInCart.
 - Composition: Atomic.
 - Inputs: PutInCartBookISBN (string).
- SignInAlternatives.
 - Composition: Choice of CreateAcctSequence and SignInSequence [\[2\]](#).
 - Inputs (derived from the atomic processes): SignInAlternativesCreateAcctInfo (AcctInfo), SignInAlternativesSignInInfo (SignInData).
 - Outputs (derived from the atomic processes): SignInAlternativesCreateAcctOutput (CreateAcctOutputType, unconditional).
- SpecifyPaymentMethod.
 - Composition: Atomic.
 - Inputs: CreditCardNumber (number, xsd:decimal), SpecifyPaymentMethodCreditCardType (CreditCardType), CreditCardExpirationDate (instant).
- CreateAcctSequence.
 - Composition: Sequence of CreateAcct, CreateProfile and LoadProfile.
 - Inputs (derived from the atomic processes): CreateAcctSequenceCreateAcctInfo (AcctInfo).
 - Outputs (derived from the atomic processes): CreateAcctSequenceCreateAcctOutput (CreateAcctOutputType, unconditional).

- SignInSequence.
 - Composition: Sequence of SignIn and LoadProfile.
 - Inputs (derived from the atomic processes): SignInSequenceSignInInfo (SignInData).
 - CreateAcct.
 - Composition: Atomic.
 - Inputs: CreateAcctInfo (AcctInfo).
 - Outputs: CreateAcctOutput (CreateAcctOutputType, unconditional).
 - CreateProfile [\[3\]](#).
 - LoadProfile.
 - Composition: Atomic.
 - Preconditions: ProfileExists.
 - SignIn.
 - Composition: Atomic.
 - Inputs: SignInInfo (SignInData).
- Abstract processes: An abstract process is defined in the example in case the provider wants to hide the composition details. This AbstractCongoBuy process expands to the FullCongoBuy process. Inputs and outputs are supposed to be defined by the provider in order to suit his specific needs, so the authors have decided not to define them in the example.
- Data flow: The data flow between the different process involved in the FullCongoBuy example is provided. We give a couple of examples of the data flow between the FullCongoBuy processes. For a complete specification of the data flow, we refer the reader to [\[Martin et al., 2004b\]](#).
 - FullCongoBuy.
 - FullCongoBuyBookName from FullCongoBuy and BookName from LocateBook have the same values i.e. there is a data flow between the FullCongoBuyBookName output of FullCongoBuy and the BookName input of LocateBook.
 - LocateBookOutput from LocateBook, CongoBuyBookISBN from CongoBuyBook and FullCongoBuyBookISBNOutput from FullCongoBuy have the same values.
 - FullCongoBuySignInInfo from FullCongoBuy and CongoBuyBookSignInInfo from CongoBuyBook have the same values.
 - ...
 - ...
- As for the express version of the service, conditions are needed to express preconditions, conditional outputs, and conditional effects, although only stubs are defined in the example.

Figure 2. FullCongoBuy Web Service process composition



Grounding (WSDL grounding provided for the example):

- Grounding for atomic processes: LocateBookGrounding, PutInCartGrounding, SignInGrounding, CreateAcctGrounding, SpecifyPaymentMethodGrounding, SpecifyDeliveryDetailsGrounding.

As for the other examples, atomic processes are mapped to a WSDL operation, each OWL-S input message to the correspondent message parts in the WSDL definition, and each OWL-S output message to the correspondent message parts in the WSDL definition. A WSDL file is also given, defining the mappings from the WSDL interface to the OWL-S description. Nevertheless, some of the atomic processes used in the service decomposition have no grounding defined.

3. From OWL-S to WSMO

In this section, we use the Web Service Modeling Ontology WSMO-Standard v0.2 [Roman et al., 2004] to model the examples described in the previous section.

The OWL-S examples assume the homogeneity of the terminologies (domain ontologies) used to describe the Web Services. As a consequence, the ooMediators in WSMO-Standard are not necessary to model the examples given.

Regarding wwMediators, they would be used to define the decomposition of the service, i.e. the WSMO-Standard orchestration. Unfortunately, the current version of WSMO-Standard does not sufficiently define how the orchestration is described, which makes the use of wwMediators and the definition of the decomposition of the service not possible. Future

versions of WSMO-Standard will define this aspect in detail, and the modelling of the service orchestration will be included in the following versions of this document.

As a consequence, we focus in the following sections on the modelling of WSMO-Standard goals, ggMediators, wgMediators and Web Services for the OWL-S examples presented. We assume that the OWL domain ontologies defined in the examples are valid, and we will not redefine them here.

3.1 The *BravoAir* Web Service in WSMO-Standard

The OWL-S profile is interpreted as advertising both the capability of the service and the requirements of the requester. As discussed in [\[Lara et al., 2004\]](#), this unified view is split in WSMO-Standard into two different views: the requester view (modelled as a goal) and the provider view (modelled as a capability). Therefore, we start the modelling of the BravoAir service by defining the goal for the requester.

In the OWL-S modelling, the profile of the service is classified using a profile hierarchy. This can be modelled in WSMO-Standard via ggMediators, that can define the reuse and refinement of other goals. Therefore, we will model the profile hierarchy given in the example as a successive refinement of goals.

Nevertheless, it can be seen that the OWL-S profile hierarchy does not define real profiles i.e. does not define the IOPEs of the service, but a taxonomy of profiles. In WSMO-Standard, this taxonomy is defined by REAL goals i.e. the categorization is given by the use and refinement of existing goals.

In order to give a more accurate idea of the use of goals and ggMediators in WSMO-Standard, we will slightly adapt the OWL-S example. We will assume that the properties defined for the profile hierarchy, such as *merchandise* or *deliveryMode*, mean that the profile describes an effect of performing a purchase and having the goods delivered, and we will model them as effects in the goal. The OWL-S taxonomy could be defined using subclassing of goals, but this is not the intended use of goals and ggMediators in WSMO-Standard.

In the following, we model the taxonomy defined in the OWL-S profile hierarchy using goals and ggMediators.

Listing 1. Goal definition for E-commerce

```
eCommerceGoal:goal[
effects ->> eCommerceEffect
]
```

Listing 2. Effect for the E-commerce goal

```
eCommerceEffect:axiomDefinition[
definedBy -> X:purchase[goodType -> product], Y:delivery[deliveryMode ->
transportation]
]
```

Listing 3. Goal definition for AirlineTicketing

```
airlineTicketingGoal:goal[
usedMediators ->> eCommerceAirlineTicketingMediator
]
```

Listing 4. Mediator between E-commerce goal and AirlineTicketing goal

```
eCommerceAirlineTicketingMediator:ggmediator[
sourceComponent -> eCommerceGoal
targetComponent -> airlineTicketingGoal
reduction -> airlineTicketingReduction
]
```

Listing 5. AirlineTicketing reduction

```
airlineTicketingReduction:axiomDefinition[
definedBy -> X:purchase[goodType -> AirlineTicket]
]
```

Once the goals and their relationship are defined, we will model the actual Web Service and link it to the appropriate goal using a wgMediator.

Listing 6. Web Service definition for BravoAir

```
bravoAirWebService:webService[
nonFunctionalProperties -> nonFunctionalPropertiesBravoAir
capability -> capabilityBravoAir
interfaces ->> interfaceBravoAir
]
```

Listing 7. Non functional properties definition for the BravoAir Web Service

```
nonFunctionalPropertiesBravoAir:nonFunctionalPropertiesWS[
title -> "BravoAir Web Service"
subject -> naics561599
description -> "The service provides flight reservations based on the specification
of a flight request. This typically involves a departure airport, an arrival
airport, a departure date, and if a return trip is required, a return date. If the
desired flight is available, an itinerary and reservation number will be returned."
publisher -> reservationDepartment
identifier -> "http://www.deri.at/FormalComparison/Examples/BravoAir"
source -> "http://www.daml.org/services/"
performance -> goodRating
reliability -> goodRating
security -> goodRating
scalability -> goodRating
robustness -> goodRating
accuracy -> goodRating
networkRelatedQoS -> goodRating
]
```

Notice that some non-functional properties that are available in WSMO-Standard but that are not defined in the OWL-S example, have not been used.

Some values, such as naics561599, reservationDepartment or goodRating are defined as instances of the appropriate domain ontologies. The exact definition of these instances and the correspondent ontologies is out of the scope of this document.

In the OWL-S modelling of the BravoAir non-functional properties, two contacts and two categorizations are described. However, the cardinality constraints in WSMO-Standard do not allow the definition of more than one value for the non-functional properties of the service.

The contact information given in the OWL-S example has been included as the publisher of the service. It can be argued that it could also be the creator or either the contributor, but with the information available about the OWL-S example, defining the contact information as the publisher seems to be the original intention of the authors.

The OWL-S modelling includes information about the geographical radius of the service. There is not such a non-functional property in WSMO-Standard.

The rating property in OWL-S could be interpreted in different ways. As WSMO-Standard provides a more detailed non-functional characterization of the service, we chose to interpret the rating given in the example as representing several non-functional aspects of the service, such as performance, reliability, security, etc. In general, a service modelled using WSMO-Standard will describe specific values for each of these properties.

Listing 8. Capability definition for the BravoAir Web Service

```
capabilityBravoAir:capability[
usedMediators ->> airlineTicketingBravoAirMediator
preconditions ->> bravoAirPrecondition
postconditions ->> bravoAirPostcondition
effects ->> bravoAirEffect
]
```

We do not define non-functional properties for the capability because this information is not defined in the OWL-S example. However, for other use cases, they could be used.

As discussed before, we do not define ooMediators because in the example homogeneity of terminologies is assumed.

Listing 9. Mediator between AirlineTicketing goal and BravoAir capability

```
airlineTicketingBravoAirMediator:wgMediator[
sourceComponent -> airlineTicketingGoal
targetComponent -> capabilityBravoAir
reduction -> bravoAirReduction
]
```

The reduction includes the postconditions that are defined by the BravoAir capability but not defined in the goal, stating the difference of functionality between the goal and the service.

Listing 10. BravoAir reduction

```
bravoAirReduction:axiomDefinition[
definedBy -> O1:output, O1:preferredFlightItinerary_Out, O2:output, O2:acctName_Out,
O3:output, O3:reservationID_Out
]
```

Listing 11. BravoAir precondition

```
bravoAirPrecondition:axiomDefinition[
definedBy -> I1:input, I1:departureAirport_In, I2:input, I2:arrivalAirport_In,
I3:input, I3:outboundDate_In, I4:input, I4:inboundDate_In, I5:input,
I5:roundTrip_In, I6:input, I6:preferredFlightItinerary_In, I7:input,
I7:availableFlightItineraryList_Out, I8:input, I8:acctName_In, I9:input,
I9:password_In, I10:input, I10:reservationID_In, I11:input, I11:confirm_In
]
```

Listing 12. BravoAir postcondition

```
bravoAirPostcondition:axiomDefinition[
definedBy -> O1:output, O1:preferredFlightItinerary_Out, O2:output, O2:acctName_Out,
O3:output, O3:reservationID_Out ]
```

In WSMO-Standard, the postcondition defines the relationship between the input and the output. However, as this information is not capture in OWL-S, it cannot be extracted from the BravoAir example. For that reason, we cannot include that information here, although it is necessary to completely describe the functionality of the service.

Listing 13. BravoAir effect

```
bravoAirEffect:axiomDefinition[
  definedBy -> X:purchase[goodType -> airlineTicket], Y:delivery[deliveryMode ->
  transportation] ]
```

It can be seen that the BravoAir effect has been slightly adapted from the OWL-S example, in order to make it consistent with the interpretation of the service effect given for the goal.

Listing 14. Interface definition for the BravoAir Web Service

```
interfaceBravoAir:interface[
  choreography => choreographyBravoAir
  orchestration => orchestrationBravoAir
]
```

Non-functional properties and used mediators are not defined for the interface for the same reasons they were not defined for the capability. Regarding the choreography and orchestration, the current version of WSMO-Standard does not provide the required modeling elements to describe them yet. Future versions of WSMO-Standard will define the required modeling elements and the details of the choreography and orchestration of the BravoAir Web Service will be added to this document.

3.2 The Congo Web Service in WSMO-Standard

4. Use Cases in WSMO

At the current state, the use cases in WSMO are still underdefined to allow for a concrete analysis of the feasibility of translations to OWL-S. [We prefer to wait for the final version of these specifications in favor of just some adhoc encodings in OWL-S at the moment.]

5. From WSMO to OWL-S

6. What is Missing in OWL-S?

At the current state, OWL-S still lacks of a formal specification of a logical formalism to define rules and logical expressions, in particular for describing relations of inputs and outputs or preconditions and postconditions respectively. The OWL-S use cases above to not require the definition of such relations, but as we shall see by more complex examples, the capability to model such relations is essence for the functional description of Semantic Web Services. However, it shall be mentioned that in the latest OWL-S version several extensions for incorporating rules in OWL-S are mentioned. In particular, the authors mention DRS and SWRL as potential candidates for a formal language for conditions in the context of OWL-S. Short comparison between - OWL - OWL+DRS - OWL+SWRL - F-Logic Rules examples from: ORL (=SWRL) copy paste.

7. Discussion and Open Points

- Use cases are too simply yet, not taking into account necessity of rules and logic in OWL-S.
- All the OWL-S use cases have exactly one profile, service, process model and grounding. It would be interesting to have more complex examples with different profiles. Same applies to WSMO use cases with different interfaces for one capability.
- Their is a necessity for orchestration and choreography use cases of WSMO as soon as these points are clearly defined.

- We are missing use cases with complex conditions in the goals and capabilities.

Summarizing, for a more formal comparison, we would need examples covering all features of the respective approaches (WSMO and OWL-S) as well as more elaborate versions of both.

8. Conclusions and Further Work

Based on the manual translations here, we try to come up with a first mapping from OWL-S to WSMO (refer D.4.3). Note that this should also be aligned with the proof obligations defined in D5.

References

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

[Flett, 2002] A. Flett: *A comparison of DAML-S and WSMF*, Technical report, 2002.

[Keller et al., to appear] U. Keller, A. Polleres, R. Lara, and Y. Ding: *Language Evaluation and Comparison*, to appear.

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

[Lara et al., 2004] R. Lara, D. Roman, and A. Polleres: *Conceptual Comparison WSMO/OWL-S.*, version 0.1 available at <http://wsmo.org/2004/d4/d4.1/v01/index.html>

[McIlraith et al., 2001] S. A. McIlraith, T. C. Son, H. Zeng: *Semantic Web Services*, IEEE Intelligent Systems, 16(2), March/April, 2001.

[Martin et al., 2004] D. Martin, et al.: *OWL-S: Semantic Markup for Web Services*, version 1.0 available at <http://www.daml.org/services/owl-s/1.0/owl-s.pdf>.

[Martin et al., 2004b] D. Martin, et al.: *OWL-S 1.0 Release: Examples*, available at <http://www.daml.org/services/owl-s/1.0/examples.html>.

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

[Stollberg & Arroyo, 2004] M. Stollberg, S. Arroyo (eds.): *WSMO Use Case Modeling and Testing*, version 0.1 available at <http://wsmo.org/2004/d3/d3.2/index.html>

Acknowledgement

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

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

[1] There is a mistake in the example, as the Service definition refers to Profile_Congo_BookBuying_Service via the *presents* property, but the real profile that references the service via the *presentedBy* property is Profile_Full_Congo_BookBuying_Service.

[2] There is a mistake in the example. SignInSequence is defined as atomic in the choice definition, although this process is defined as a composite process later on. We will consider it as a composite process, as this seems to be the intention of the authors.

[3] The CreateProfile process is not defined in the example.

[webmaster](#)