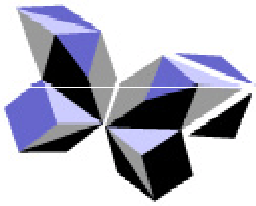

Digital Business Ecosystem

Contract n° 507953

WP 16: Service Description Language

D16.4: SSL Compiler

Project funded by the European Community under the "Information Society Technology" Programme.

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D16.4

Due date: July 2006

Delivery Date: July 2006

Short Description: This Deliverable describes the SSL Compiler introduced in the project to make possible the automatic generation of part of the SDL model and to guide users toward the most convenient technical definition of the service.

Author: Soluta.net

Partners contributed: Soluta.net

Made available to: Public

Versioning		
Version	**Date**	**Author, Organisation**
00.01	2006/06/27	Irina Dumitrascu, Giulio Montanari, Umberto Pernice – Soluta.net

Quality check

1st Internal Reviewer: Fotis G. Kazasis - TUC

2nd Internal Reviewer: Victor Bayron -UCE

Table of Contents

1. EXECUTIVE SUMMARY **6**

2.INTRODUCTION **7**

3.DESCRIPTION **8**
3.1 SSL Overview 8
3.2 SSL in comparison with SDL 9
3.3 Transformation process 9
3.4 Transformation description 11
3.4.1 SSL – type information 11
3.4.2 ODM – type information 13
3.4.3 SDL – structure 14
3.4.4 Transformation logic 16
3.4.5 Implementation details 18
3.4.5.1 Rules 18
3.4.5.2 Helpers 21

4.TECHNICAL DETAILS **24**

5.USAGE **25**

6. EXAMPLES **27**
6.1 Echo name service 27
6.1.1 SSL description 27
6.1.2 SDL description 28
6.2 PC sale 29
6.2.1 SSL description 29
6.2.2 SDL description 32
6.3 Hotel reservation 35
6.3.1 SSL description 35
6.3.2 ODM description 37
6.3.3 SDL description 41

7. APPENDIXES **47**
7.1 Transformation code (ATL) 47

8.GLOSSARY **56**

9.REFERENCES **58**

Table of Figures

FIGURE 1: THE MDA APPROACH OF DBE..... 10
FIGURE 2: SSL COMPILER..... 10
FIGURE 3: SSL METAMODEL – SERVICE BEHAVIOR MODELING..... 12
FIGURE 4: SSL METAMODEL – TYPE REFERENCING SCHEME..... 13
FIGURE 5: ODM METAMODEL – PROPERTIES IN ODM..... 14
FIGURE 6: SDL METAMODEL..... 15
FIGURE 7: SDL TYPES..... 16
FIGURE 8: INVOKE THE SSL COMPILER..... 25
FIGURE 9: HOTEL RESERVATION EXAMPLE..... 46

1. Executive Summary

This document describes the SSL[1] Compiler which aims at generating SDL[2] models starting from SSL models (part of the BML[3]) and the semantic service ontology (ODM[4]).

The main purpose of the SSL Compiler is the automatic generation of the SDL model, significantly reducing the time spent to create it, this will also reduce the effort needed to develop services.

The SSL is annotated with the Ontology Definition Metamodel (ODM) references in order to allow SSL models to make use of concepts defined in domain specific ontologies increasing the semantic interoperability among SMEs.

The SSL to SDL transformation is an example of the CIM[5] to PIM[6] automation chain that OMG is envisioning in the context of MDA[7]. What is described here is currently the first known working example of CIM to PIM transformation which uses two input metamodels (SSL, ODM).

The SSL Compiler is offered as a plug-in for DBE Studio, the Integrated Development Environment (IDE) based on Eclipse plug-ins to give SMEs the possibility to model, implement, publish, deploy, compose, search and execute DBE services.

1 Semantic Service Language (SSL). See [DBECoreArch] and [Deliverable D14.1]
2 Service Description Language (SDL). See [DBECoreArch] and [Deliverable D16.1]
3 Business Modelling Language (BML). See [DBECoreArch] and [Deliverable D15.3]
4 ODM is a metamodel developed in the DBE project that can be used for defining business and service ontologies. See [Deliverable D14.1]
5 Computation Independent Model (CIM). See [MDA]
6 Platform Independent Model (PIM). See [MDA]
7 Model Driven Architecture. See [MDA]

2.Introduction

This document describes the SSL Compiler as a result of the task C56 of the project which aims at generating the SDL model starting from the SSL model (part of the BML) and possibly using information from the Semantic Service Information System Ontology. In this direction this deliverable takes into consideration results achieved by the two tasks it was depending on, as follows:

- Task C11 - Knowledge Base Model of the DBE, from which it uses the definition of ODM and SSL.
- Task C13 - SDL Definition, since the SDL is needed in order to realize the transformation.
- Task B4 – SME needs and functional requirements, from which the SSL receives some inputs so as to be refined according to the business needs.

3.Description

The SSL Compiler has been introduced in the project to achieve the following objectives:

- To generate SDL models from SSL models (the SSL being part of the BML) and the Semantic Service Information System Ontology;
- To map SSL to SDL concepts;
- To reduce the effort needed to define services.

3.1 SSL Overview

It is not a goal of this document to describe the Semantic Service Language (SSL) as it focuses on the SSL Compiler and the process of generating SDL models from SSL models. Detailed information on the SSL can be found in [Deliverable D14.1]. However a brief overview of the SSL is here below provided for a broader comprehension of the tool and the generation process.

The SSL is used to define semantic service description models for capturing the semantics of specific types (e.g. hotel booking, flight reservation, etc.) of business offerings (services). These semantics (i.e. instances of SSL models) will be used to advertise a specific service of a particular type to the external world (the candidate service consumers).

In a wider sense, the SSL is used to semantically model classes of services from the external business point of view. That is, to describe what the candidate consumers of the services will see. Typically, the semantics captured with SSL will be one of the main input in the semantic service discovery process.

The SSL describes three kinds of semantics:

- **Special features of a service** that it is believed will be valuable to the candidate service consumer (i.e. supported payment methods, applied charging schemes, product or resource features, as well as information that may be influencing to the decision such as the weather information in a resort booking service).
- **Business Level Interaction needed to Consume a Service,** that relates to the business interactions between the business parties and include: what (information and resources) the consumer has to give to the provider, and what (information and resources) the service provider will give back to the consumer and under which conditions.
- **Contacts Responsible for the Provision of a Service**. An SSL description of a service may describe one or more information structures for referring to agents or individuals responsible for the service (or some aspect of the service).

The SSL makes use of ODM (Ontology Definition Metamodel [Deliverable D14.1]) in order to allow SSL models to make use of concepts defined in domain specific ontologies, hence easing the interoperability among SMEs services. Ontology elements can be used, for example, to indicate that a service operation has a “Customer” as input, whose semantics and relationships with other concepts belonging to the same domain are defined in the ontology. This mechanism provides SSL modellers with a

means to annotate the appropriate part of the service description with concepts from a richer semantic model.

3.2 SSL in comparison with SDL

SDL is a language that can be used to describe the technical interface of services in a platform independent way.

The SSL and SDL standards are mutually related since they are both used to describe services; the difference is that while the SSL specifies the semantics of the service from a business perspective (what the service actually does) the SDL describes the technical interfaces of the service (the protocol for the service invocation). Nevertheless a technical interface for a certain service can be partially obtained from its semantic description; therefore, the SSL to SDL transformation converts an SSL model into an SDL model, also using the ODM model referred in the SSL model (if any) to include the necessary data from an ontology into the technical specification of the service.

As an example, if in a SSL model a reference to an ontology class is used to indicate that a service operation has a “Customer” as input, the transformation can lookup the Customer definition in the ontology and therefore the generated SDL model defines that a Customer is a ComplexType that has a field Name of SDLString type, a field DateOfBirth of SDLDate, etc. Information is very similar, but it is observed from two different viewpoints: CIM for SSL and ODM, PIM for SDL. In addition the ontology link can clarify the semantic and hence the meaning of the term used.

3.3 Transformation process

The SSL to SDL transformation process is a CIM to PIM transformation. As envisioned by the MDA approach transformations can be of type CIM to PIM, PIM to PIM, PIM to PSM[8] and PSM to code. A computational independent model can be refined, i.e. a CIM can be mapped to another CIM, in the same way PIMs can be refined. Transformations from PIM to CIM, PSM to CIM, and PSM to PIM are not possible. Benefits are the ability to postpone in the development process the creation of models that take into account technological aspects of a platform as much as possible as well as the ability to react efficiently and with low costs to technology changes promoting portability, interoperability and reusability trough architectural separation of concerns.

Figure 1: The MDA approach of DBE[9] below provides an overview of how the DBE Knowledge can be exploited in the (semi-) automatic software generation objective of the DBE.

8 Platform Specific Model (PSM). See [MDA]
9 The image is taken from [Deliverable D14.1]

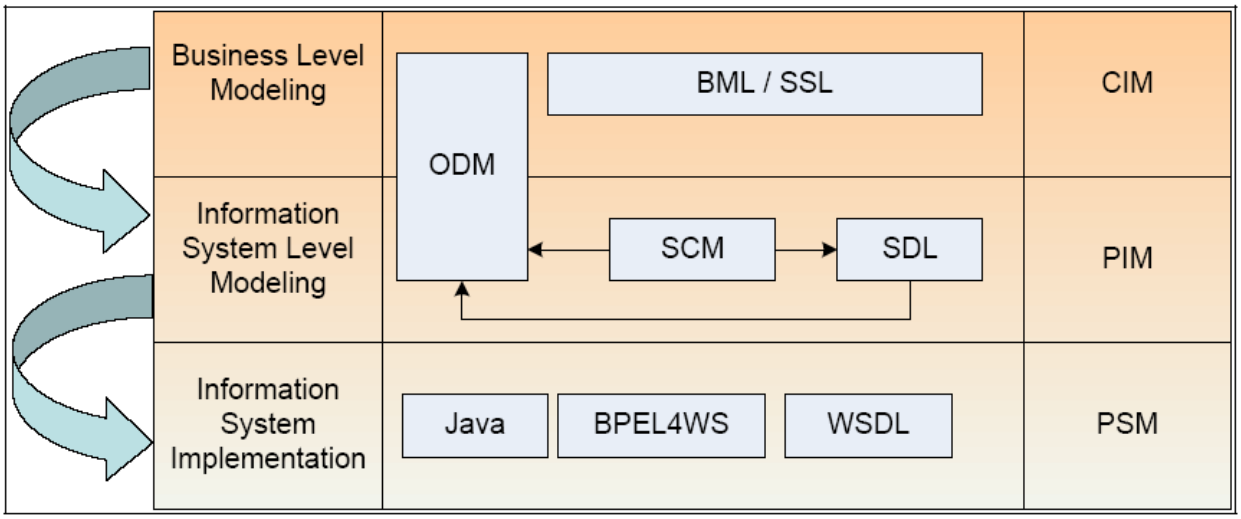

Figure 1: The MDA approach of DBE

The process of automatically obtaining the SDL model is illustrated in Figure 2: SSL Compiler below. In particular, starting from an SSL model representing the computation independent business viewpoint about the SME provider and its services, the SSL Compiler automatically generates (partially or entirely) an SDL platform independent model that can be used as is or modified using the SDL Editor (delivered with the DBE Studio[10]) in order to fulfil some specific purposes and needs. Such transformation avoids the SDL modeller doing much of the work to map SSL to SDL manually through the SDL Editor, that would result in a more time-consuming and error-prone process.

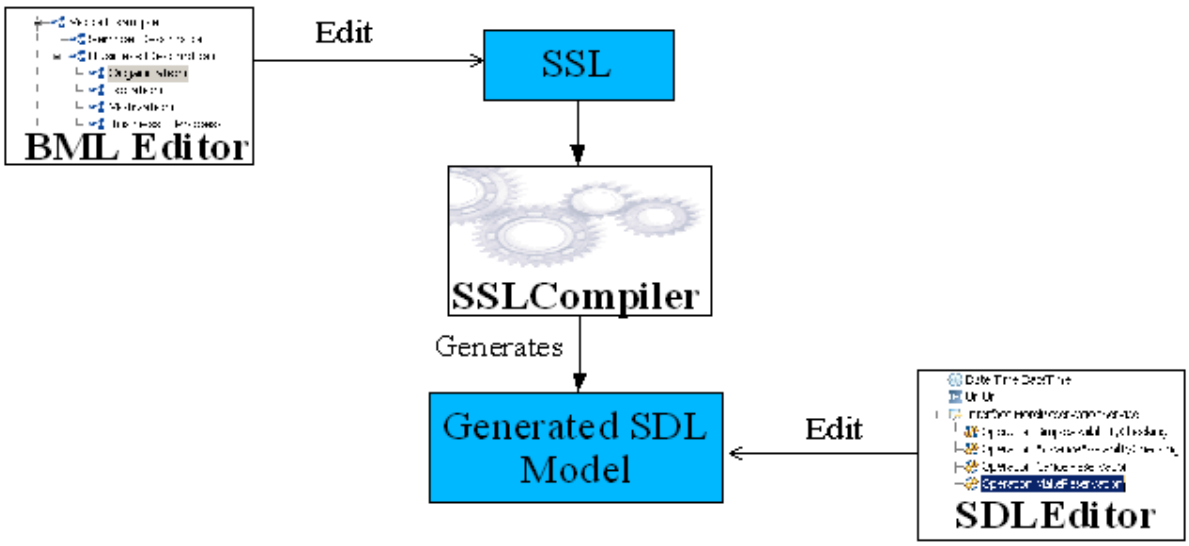

Figure 2: SSL Compiler

10 The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE) based on the Eclipse platform. See http://dbestudio.sourceforge.net/.

The SSL compiler is fed with SSL models, which can be:

- new models, created using the dedicated BML editor that is delivered with DBE Studio;
- existing models retrieved from the DBE Knowledge Base using the Model Navigator view[11].

3.4 Transformation description

The transformation selects from the SSL model those parts which are of interest for obtaining technical descriptions for the SDL model. These parts can be contained directly in the SSL model or they can be defined in an independent ODM model and referenced in the SSL model. Therefore, for the rest of this section these elements are highlighted, together with some brief information about the SDL structure. For the full description of the metamodels involved in the transformation the reader should refer to [DBECoreArch], [Deliverable D14.1] and [Deliverable D16.1].

3.4.1 SSL – type information

In SSL models, services can be described using SSL ServiceProfile elements that contain, among others, multiple SSL ServiceFunctionality elements. Each SSL ServiceFunctionality describes a logical function of the modelled service and it can have multiple input elements (SSL ServiceInput) and output elements (SSL ServiceOutput). This structure can be also observed in Figure 3: SSL Metamodel – Service Behavior Modeling below[12].

The kind of data that is expected by each operation, for its input and output is specified as each SSL ServiceInput and ServiceOutput has an attribute named Type that belongs to the class SSL TypeURI. This element is interesting for the purpose of this transformation, as it can be used to infer the corresponding type in the generated SDL model. In order to understand the possible kind of values for the type of a service input or service output, the hierarchy of the class SSL TypeURI is shown in Figure 4: SSL Metamodel – Type referencing scheme below. Since for the moment the enumerated types cannot be defined outside ontologies, the types considered for this transformation are SSL OntologyClassURI and SSL DataTypeURI.

11 Even if model retrieval can be also done using the DBE searching mechanisms (QF/SDT), in order to reduce dependencies between the DBE tools the SSL Compiler only works with local xmi files .

12 Image based on [Deliverable D14.1]

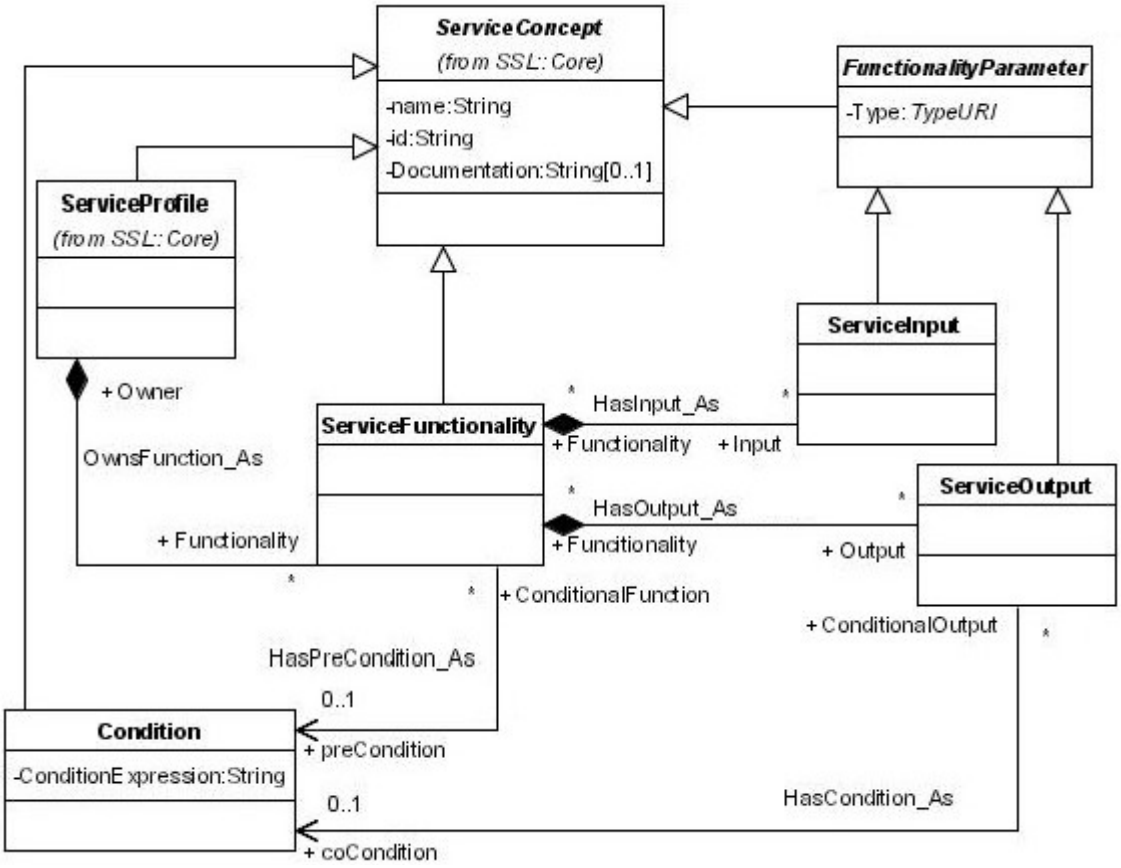

Figure 3: SSL Metamodel – Service Behavior Modeling

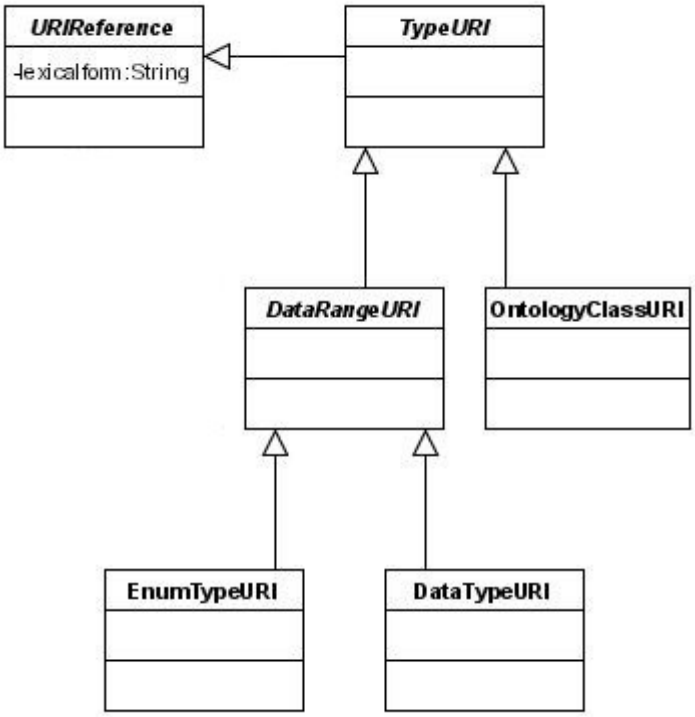

Figure 4: SSL Metamodel – Type referencing scheme

3.4.2 ODM – type information

In SSL, the type of a certain element can be a class from a specified ontology, if it belongs to the class OntologyClassURI. Inside an ontology, the properties of classes are specified using ODM Property elements; each ODM Property has a domain, that indicates the classes that have the current property, and an ODM DataRange element that specifies its type. This structure is presented in Figure 5: ODM Metamodel – Properties in ODM below[13], where the following can be observed:

- each ODM Property has an association to an ODM Class, that has the role (name of the association end) theDomain
- the class ODM DatatypeProperty extends ODM Property
- each ODM DatatypeProperty has an association to an ODM DataRange, and the association end corresponding to ODM DataRange has the name theDataRange.

13 Image taken from [Deliverable D14.1].

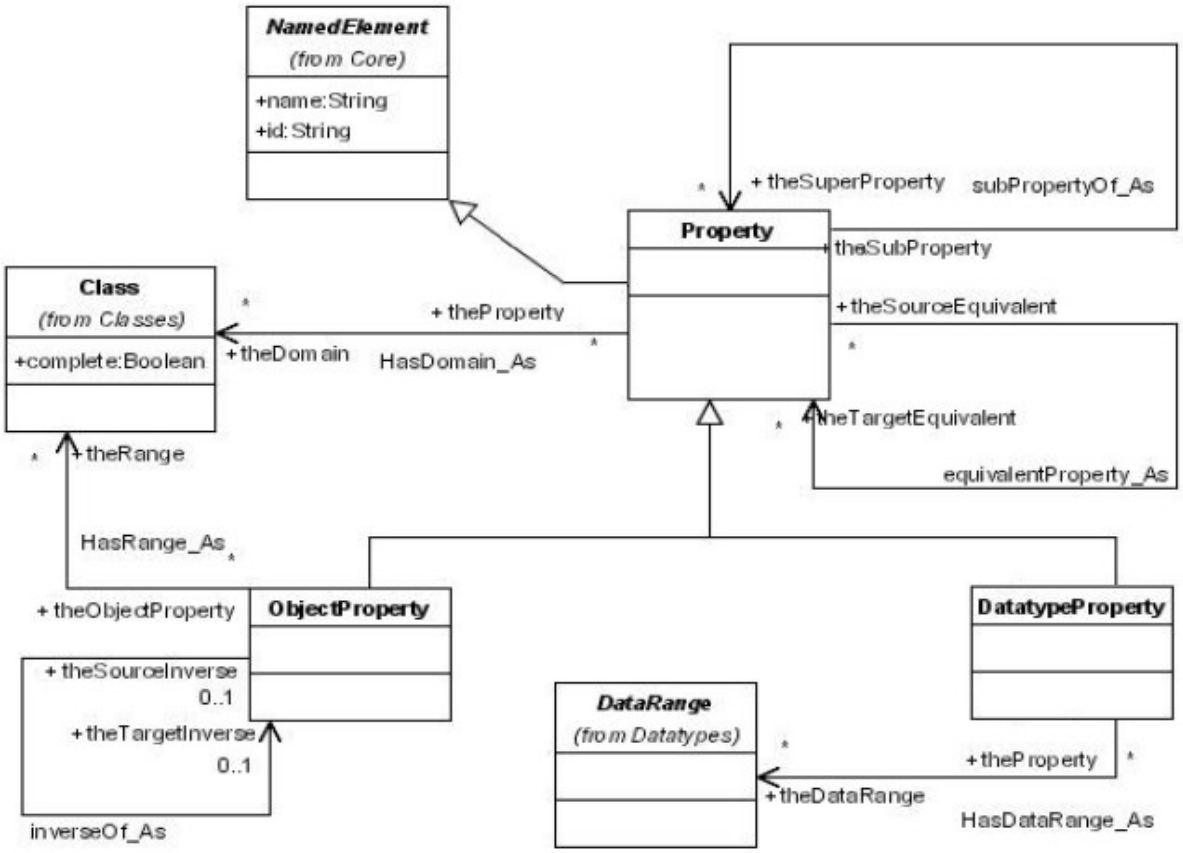

Figure 5: ODM Metamodel – Properties in ODM

3.4.3 SDL – structure

The elements of the SDL that are of interest for understanding this transformation are briefly introduced below, more information be found in [Deliverable D16.1].

The root of any SDL model is a Definitions element, that contains the defined SDL types, interfaces, messages and message lists. Each interface has at least one operation, that in turn contains an input and an output message (or none) and possibly some exceptions. A visual description of some relevant elements from the SDL model is presented in Figure 6: SDL Metamodel[14], below.

14 Image based on [Deliverable D16.1].

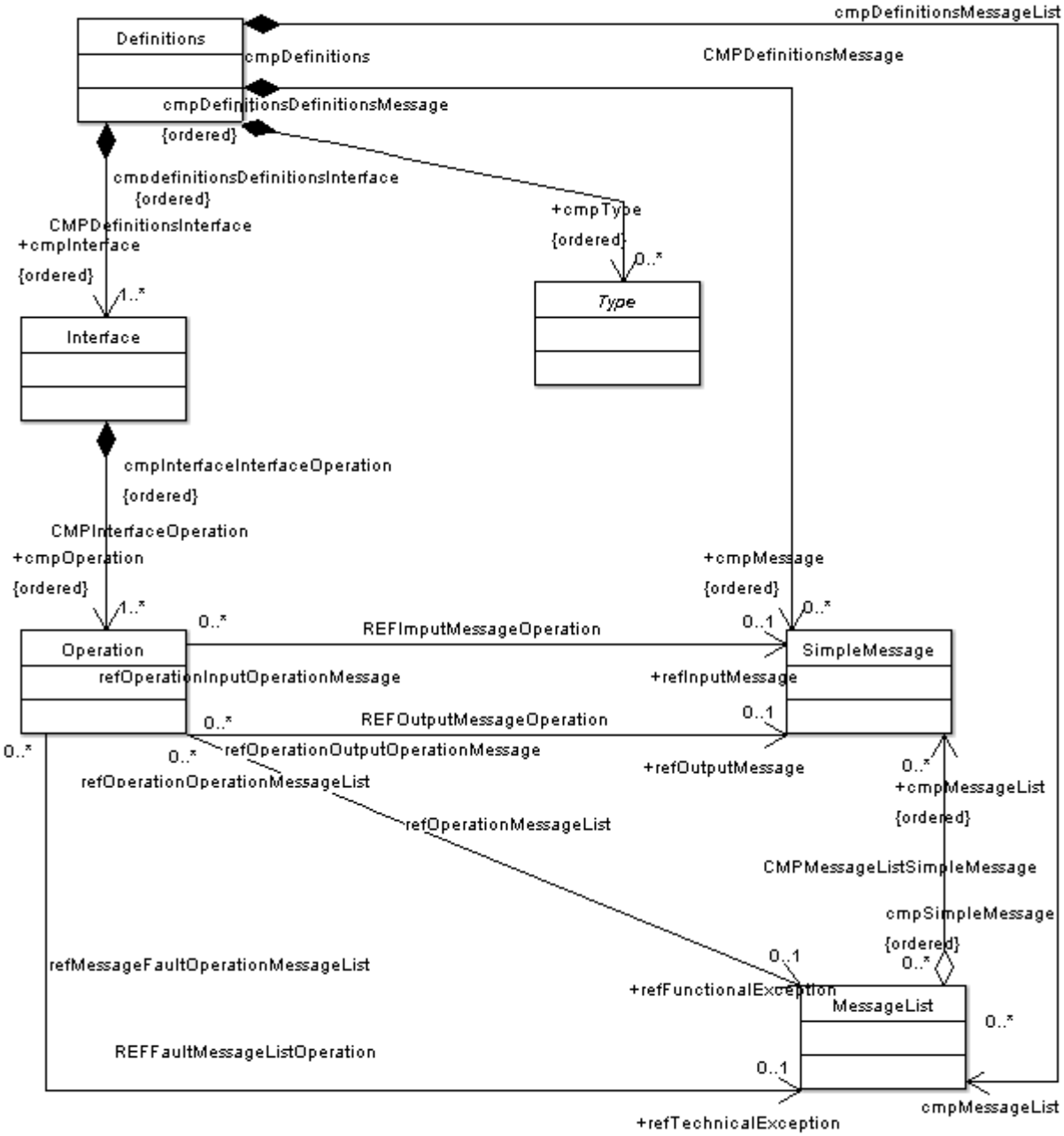

Figure 6: SDL Metamodel

The SDL types can be:

- simple types: SDLBoolean, SDLInteger, SDLReal, SDLDateTime, SDLUri and SDLString
- complex types, that are composed of multiple Parts, each type having an associated type.

This is also shown in Figure 7: SDL Types[15] below.

15 Image based on [Deliverable D16.1].

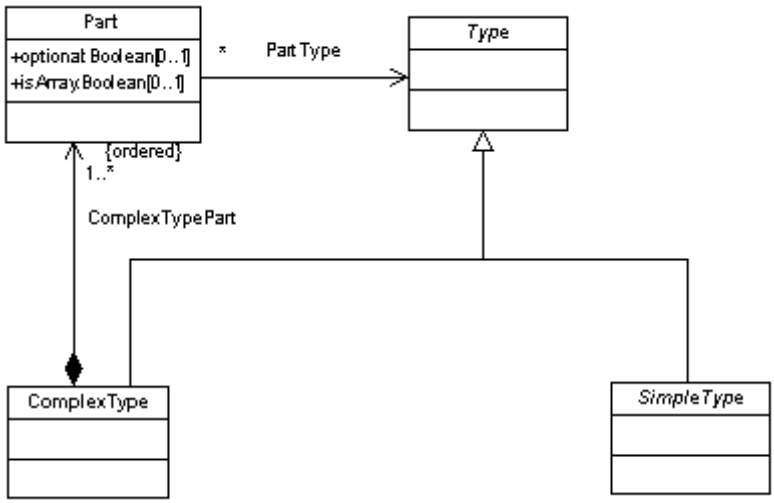

Figure 7: SDL Types

3.4.4 Transformation logic

The transformation gathers all the information regarding the service operations and the involved types from the SSL model and from the associated ODM model (if any) in order to include it in the generated SDL model.

From the SSL root element, ServiceProfile, the SDL root element, SDL Definitions, is created. It contains the following elements:

- the SDL types (that are generated);
- one SDL Interface, that includes all the generated SDL Operation elements;
- the list of generated SDL SimpleMessage elements.

Each SSL Service profile contains SSL ServiceFunctionality elements that describe the functionalities that the service offers. In the SDL metamodel, there is an element that groups information related to functionalities: SDL Operation. Therefore, from each SSL ServiceFunctionality element the corresponding SDL Operation is created.

As an SSL ServiceFunctionality can have multiple Input(s)/Output(s) while in SDL each Operation must have at maximum one input and one output, a “compaction” of multiple inputs/outputs into a single message is needed and it is realized as follows:

- if a given SSL ServiceFunctionality has more than one Input (it has multiple inputs) then an SDL SimpleMessage is created, having as SDL Part-s the elements obtained from the transformation of its inputs
- if a given SSL ServiceFunctionality has more than one Output (it has multiple outputs) then an SDL SimpleMessage is created, having as SDL Part-s the elements obtained from the transformation of its outputs.

If an SSL ServiceFunctionality has only one input, then an SDL SimpleMessage is constructed for that input. Similarly, if an SSL ServiceFunctionality has only one

output, an SDL SimpleMessage is constructed to hold the information from that output.

From each SSL DataTypeURI (that is pointing to an XML Schema[16] type) that is the Type attribute of an SSL ServiceInput or an SSL ServiceOutput, an SDL Part is created, referring to the SDL type that corresponds to the indicated XML Schema type. The correspondence between the XML Schema types and the SDL types is shown in Table 1: XML Schema and SDL types below.

XML Schema Type	***SDL Type***
integer, positiveInteger, negativeInteger, nonPositiveInteger, nonNegativeInteger, long, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte	SdlInteger
boolean	SdlBoolean
decimal, float, double	SdlReal
datetime, date, time, gYearMonth, gMonthDay, gYear, gMonth, gDay	SdlDateTime
string, normalizedString, token, language, NMTOKEN, Name, NCName, hexBinary, base64Binary	SdlString

Table 1: XML Schema and SDL types

From each SSL OntologyClassURI (containing the id of an ODM Class defined in the ontology) that is the Type attribute of an SSL ServiceInput or an SSL ServiceOutput the following elements are created:

- an SDL ComplexType containing an SDL Part for each ODM DatatypeProperty element that refers to that ODM Class in its theDomain relation. Each generated SDL Part has a reference to a type, that is the translation of the type specified in the attribute theDataRange of the corresponding ODM DatatypeProperty
- if the SSL OntologyClassURI is the Type of an SSL ServiceInput included in an SSL ServiceFunctionality with multiple inputs or if it is the Type of an SSL ServiceOutput included in an SSL ServiceFunctionality with multiple outputs, then an additional SDL Part element is created, and it has as its type the SDL ComplexType that it generated.

The ODM specifies that the types used in ODM models should belong to a subset of the XMLSchema types, and this subset contains exactly the elements presented in Table 1: XML Schema and SDL types above - so the same transformation is applied to obtain the corresponding SDL type from a type specified in an ontology model.

16 XML Schema provides a way to define structure and content for XML documents. See http://www.w3.org/XML/Schema.

3.4.5 Implementation details

The transformation is realized using the Atlas Transformation Language (ATL)[17], a metamodel transformation language that responses to the OMG's QVT request for proposals[18]. This is possible because all the metamodels created in the DBE project are defined using an MDA approach and, therefore, created using MOF as meta-metamodel.

ATL allows defining the transformation by specifying:

- rules – that state how some specified input elements are transformed into specified output elements;
- helpers – through which global variables and functions can be defined.

The rest of this section explains the rules and helpers that comprise the SSL and ODM to SDL transformation that is realized by the SSL compiler. The full code of the transformation is presented in appendix 7.1 - Transformation code (ATL).

For details about the ATL language and the rules that govern the execution of ATL transformations, please see [ATL User Manual].

3.4.5.1 Rules

The following rule is the main rule of the transformation, that constructs the skeleton of the SDL model and uses the results of the other rules in order to fill it.

Definitions	It creates the SDL types definitions, generates the definitions of the interfaces (containing the generated operations) and groups the generated messages.

The following rules drive the transformation between SSL ServiceFunctionality elements and the corresponding constructs in SDL:

ServiceFunctionality11	For every SSL ServiceFunctionality that has maximum one input and maximum one output, this rule constructs an SDL Operation that contains the results of the input's and output's transformation (see rules SingleInput_Ontology, SingleInput_NonOntology and their correspondents for output messages).
ServiceFunctionalityN1	For every SSL ServiceFunctionality that has multiple inputs and maximum one output, this rule constructs an SDL Operation that contains the results of the output's transformation (see rules SingleOutput_Ontology, SingleOutput_NonOntology) and that constructs an SDL Message in which it "compacts" the inputs by placing, as parts, the results of the transformation of each input's type (see rules

17 See http://www.sciences.univ-nantes.fr/lina/atl/.
18 Query / Views / Transformations Request for proposals - http://www.omg.org/docs/ad/02-04-10.pdf

	TypeFromOntology_Input: part_c and TypeFromSchema: part_c).
ServiceFunctionality1N	For every SSL ServiceFunctionality that has maximum one input and multiple outputs, this rule constructs an SDL Operation that contains the results of the input's transformation (see rules SingleInput_Ontology, SingleInput_NonOntology) and that constructs an SDL Message in which it "compacts" the outputs by placing, as parts, the results of the transformation of each output's type (see rules TypeFromOntology_Output: part_c and TypeFromSchema: part_c).
ServiceFunctionalityNN	For every SSL ServiceFunctionality that has multiple inputs and multiple outputs, this rule constructs an SDL Operation that contains: - an SDL Message in which it "compacts" the inputs by placing, as parts, the results of the transformation of each input's type (see rules TypeFromOntology_Input: part_c and TypeFromSchema: part_c) - an SDL Message in which it "compacts" the outputs by placing, as parts, the results of the transformation of each output's type (see rules TypeFromOntology_Output: part_c and TypeFromSchema: part_c).

The following rules describe the transformation from SSL ServiceInput and ServiceOutput elements to the corresponding messages in SDL:

SingleInput_Ontology	For every SSL ServiceInput that is the only input of an SSL ServiceFunctionality and whose type is SSL OntologyClassURI (it is described in the ontology), this rules construct an SDL SimpleMessage that describes it (the message contains a single SDL Part that refers to the complex type generated from the input's type - see rule TypeFromOntology_Input: part_c).
SingleOutput_Ontology	This rule realizes the same transformation as SingleInput_Ontology, but for SSL ServiceOutput.
SingleInput_NonOntology	For every SSL ServiceInput that is the only input of an SSL ServiceFunctionality and whose type is not SSL OntologyClassURI, this rule constructs an SDL SimpleMessage

	that describes it (the type of the input is transformed according to the rule TypeFromSchema).
SingleOutput_NonOntology	This rule realizes the same transformation as SingleInput_NonOntology, but for SSL ServiceOutput.

The following rules dictate how the types that are specified in the SSL model – either directly or as types of properties in an ODM model – are transformed into SDL types:

TypeFromOntology_Input	An SSL OntologyClassURI element contains the id of an element in the associated ontology. For every SSL OntologyClassURI element that is the Type attribute of an SSL ServiceInput, this rule creates: 1) an SDL ComplexType that has an SDL Part for each ODM DatatypeProperty in which this class is referred to (by id). If there is no such ODM DatatypeProperty, by default an SDL Part with the type SDLString is created 2) if this element is the Type of an SSL ServiceInput that belongs to an SSL ServiceFunctionality with multiple inputs, then an SDL Part that has a reference to the generated SDL ComplexType is created(and it will be included in the element obtained from the transformation of that SSL ServiceFunctionality).
TypeFromOntology_Output	This rule realizes the same transformation as TypeFromOntology_Input, but for types that belong to a ServiceOutput.
TypeFromSchema	An SSL DataTypeURI element refers to a type defined in a specified XMLSchema. For every SSL DataTypeURI element that is the Type attribute of an SSL ServiceInput or an SSL ServiceOutput, this rule creates an SDL Part that refers to the corresponding SDL type (see the helper getSDLTypeForSchemaType).

3.4.5.2 Helpers

The following helpers concern the transformation of SSL or ODM types into SDL types, as explained in subsection 3.4.4 - Transformation logic:

getSDLTypeForODM (p : ODM!DataRange) returns String	Returns the name of the target pattern (in rule Definitions) that generates the SDL type that corresponds to the specified ODM DataRange. For the types that do not have direct mapping to an SDL type, the returned target pattern name is one returned by the helper getSDLTypeForODM_Default.
getSDLTypeForODM_Default () returns String	Returns the name of the target pattern (created in the rule Definitions) that is assigned by default to elements from the ontology whose type is not mapped directly to an SDL type (the pattern that generates SDL SdlString).
getSDLTypeForSchemaType (e : String) returns String	Returns the name of the target pattern (created in the rule Definitions) that generates the SDL type that corresponds to the schema type with the specified name. (The mapping realized for the schema: http://www.w3.org/2001/XMLSchema.) For the types that do not have direct mapping to an SDL type, the returned target pattern name is the one corresponding to SDL String.
getTypeGenerator () returns SSL!ServiceProfile	Returns the root element of the input SSL model, as it is the one that generates the SDL types.
getOntologyIDName (e : String) returns String	An SSL OntologyClassURI element has a lexicalForm attribute with the following structure: <ontologyID>#<ontologyClassID>. This helper takes this value as input and returns the <ontologyClassID> part.
getSchemaTypeName (e : String) returns String	An SSL DataTypeURI element has a lexicalForm attribute that contains the URI that points to an XMLSchema and to the desired type, in the form: <baseURI>#<typeName>. This helper takes this value as its input and returns the <typeName> part.

The next helpers provide utility functionalities regarding service operations, inputs and outputs:

getMultipleInputs () returns Sequence of SSL!ServiceInput	This helper returns a sequence containing all the SSL ServiceInput elements that belong to an SSL ServiceFunctionality that has multiple inputs.
getMultipleOutputs () returns Sequence of SSL!ServiceOutput	Similar with getMultipleInputs(), but referring to SSL ServiceOutput.
inputMessageName (c : SSL!OntologyClassURI) returns String	This helper returns the name of the SSL ServiceInput that has as Type the specified SSL OntologyClassURI.
outputMessageName (c: SSL!OntologyClassURI) returns String	Similar with inputMessageName(), but referring to SSL ServiceOutput.
getInputOutputHavingURIType (e: SSL!DataTypeURI) returns Sequence(SSL!FunctionalityParameter)	Returns a sequence of all the SSL ServiceInput and SSL ServiceOutput that have the Type attribute equal to the specified SSL DataTypeURI.
getServiceFunctionalityHavingInput(i : SSL!ServiceInput) returns SSL!ServiceFunctionality	Returns the SSL ServiceFunctionality that has, as one of its inputs, the SSL ServiceInput provided as parameter.
getServiceFunctionalityHavingOutput(o : SSL!ServiceOutput) returns SSL!ServiceFunctionality	Returns the SSL ServiceFunctionality that contains, as one of its outputs, the SSL ServiceOutput provided as parameter.

The helpers that are presented in the following paragraphs are utility functions for working with ODM DatatypeProperty elements:

getDataTypePropertiesForId(input Id : String) returns Sequence(ODM!DatatypeProperty)	This helper selects the ODM Class that has the id equal to the provided String input and returns a sequence containing all the ODM DatatypeProperty elements that apply to that class. The returned sequence includes also the properties that are inherited from its superclasses (the ODM DatatypeProperty-es that apply to any of its superclasses). It uses the helper getAllDataTypeProperties to realize this.
getAllDataTypeProperties(c : ODM!Class) returns Sequence(ODM!DatatypeProperty)	Taking an ODM Class, returns a sequence containing all the ODM DatatypeProperty elements that have in their theDomain property a reference to that class or to one of its superclasses (it does a recursion on all the defined super classes). It uses the helper getOwnDataTypeProperties in order to obtain the properties of a specified class.
getOwnDataTypeProperties(c : ODM!Class) returns Sequence(ODM!DatatypeProperty)	Taking an ODM Class, returns a sequence containing all the ODM DatatypeProperty elements that have in their theDomain property a reference to that class (not taken into account its superclasses).
getFakeDatatypeSequence (e : Sequence(ODM!DatatypeProperty)) returns Sequence(String)	Returns a sequence containing a 'fake' element if the size of the provided sequence is 0, otherwise it returns an empty sequence. It is used to drive the transformation in the rules TypeFromOntology_Input and TypeFromOntology_Output.

4.Technical details

The SSL Compiler is a plug-in for DBE Studio (the Integrated Development Environment that contains various tools that enable effective usage of the DBE platform [DBE Studio]).

As the transformation uses the Atlas Transformation Language (ATL), the plug-in libraries include the one for ATL and its dependencies.

ATL enables the definition and execution of transformations at the level of metamodels. When the transformation is applied, from the concrete input model(s) the output model(s) are generated applying the rules described for the corresponding metamodels.

5.Usage

The SSL Compiler is available from the Navigator view as a context menu entry for suitable elements (files having the extension .bml.xmi), as shown in Figure 8: Invoke the SSL Compiler, below.

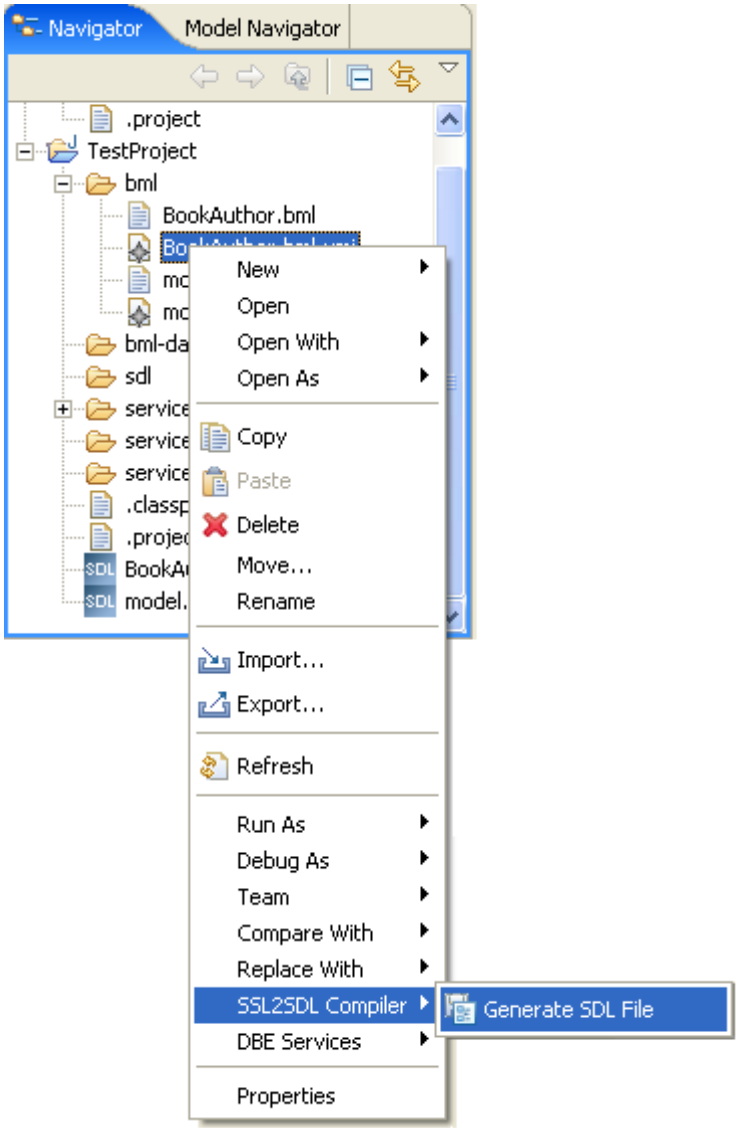

Figure 8: Invoke the SSL Compiler

The result of the transformation is a file containing the generated SDL model; it can be stored in the desired location.

Using the Pareto principle[19], it can be said that around 80% of the SDL model can be generated automatically by applying the transformation. It is expected that the total time spent

19 The Pareto principle (also known as the 80-20 rule, the law of the vital few and the principle of factor sparsity) states that for many phenomena, 80% of the consequences stem from 20% of the causes. http://en.wikipedia.org/wiki/Pareto_principle

for this 80% of the model, thanks to the generation approach, takes the 20% of the entire time of the service generation process. For the custom elements that are not comprised in the general logic of the transformation, after generating the SDL model it can be edited, if necessary, in order to modify it or to add other needed elements. These modifications can be realized using the dedicated SDL Editor.

6. Examples

6.1 Echo name service

6.1.1 SSL description

The SSL model for the echo name service that is presented below describes a very simple service with one functionality that has one input and one output, both being strings. The output should be, when the service is implemented, exactly the same as the provided input – therefore, as its name implies, it is an “echo” service.

The types of the messages are specified directly as SSL DataTypeURI elements, therefore there is no reference to any ontology in this example.

```
<?xml version="1.0" encoding="UTF-8"?>
   <XMI xmi.version="1.2" timestamp="Thu Apr 27 12:44:18 BST 2006">
     <XMI.header>
       <XMI.documentation>
         <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
         <XMI.exporterVersion>1.0</XMI.exporterVersion>
       </XMI.documentation>
     </XMI.header>
     <XMI.content>
       <SSL.Core.SemanticPackage xmi.id="a1" name="EchoNameService"
id="http://www.mySME.com/EchoNameService" Documentation="">
         <SSL.Core.NameSpace.OwnedElement>
           <SSL.Core.ServiceProfile xmi.id="a2" name="EchoNameService"
id="1146138099314">
             <SSL.Core.ServiceConcept.Position>
               <SSL.Core.XYCoordinates xmi.id="a3" X="166" Y="125"/>
             </SSL.Core.ServiceConcept.Position>
             <SSL.Core.ServiceProfile.Functionality>
               <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a4"
name="getMessage" id="1146138099315" Documentation="">
                 <SSL.ServiceBehavior.ServiceFunctionality.Output>
                   <SSL.ServiceBehavior.ServiceOutput xmi.id="a5"
name="returnMessage" id="1146138099317" Documentation="">
                     <SSL.ServiceBehavior.FunctionalityParameter.Type>
                       <SSL.Types.DataTypeURI xmi.id="a6"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                     </SSL.ServiceBehavior.FunctionalityParameter.Type>
                   </SSL.ServiceBehavior.ServiceOutput>
                 </SSL.ServiceBehavior.ServiceFunctionality.Output>
                 <SSL.ServiceBehavior.ServiceFunctionality.Input>
                   <SSL.ServiceBehavior.ServiceInput xmi.id="a7" name="inputName"
id="1146138099316" Documentation="">
                     <SSL.ServiceBehavior.FunctionalityParameter.Type>
                       <SSL.Types.DataTypeURI xmi.id="a8"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                     </SSL.ServiceBehavior.FunctionalityParameter.Type>
                   </SSL.ServiceBehavior.ServiceInput>
                 </SSL.ServiceBehavior.ServiceFunctionality.Input>
               </SSL.ServiceBehavior.ServiceFunctionality>
             </SSL.Core.ServiceProfile.Functionality>
           </SSL.Core.ServiceProfile>
         </SSL.Core.NameSpace.OwnedElement>
```

```
      </SSL.Core.SemanticPackage>
    </XMI.content>
  </XMI>
```

6.1.2 SDL description

The SDL model generated by the transformation is also very simple, as it can be seen below. It contains two definitions for SDL SimpleMessage elements, one for the input and one for the output, each of them having a single part of type SDL SdlString. In the SDL Interface element there is one SDL Operation element that has one input and one output message. Note that, in this example, the SSL Compiler generated 100% of the SDL model avoiding the SDL modeller browsing the SSL model and mapping it to SDL manually. Moreover the model generated would be the same if manually created with the SDL Editor.

```
<?xml version = '1.0' encoding = 'windows-1252' ?>
<XMI xmi.version = '1.2' xmlns:sdl = 'org.omg.xmi.namespace.sdl' timestamp = 'Thu
Jul 06 12:34:18 EEST 2006'>
 <XMI.header>
   <XMI.documentation>
     <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
   <sdl:Definitions xmi.id = 'a1' ElName = 'EchoNameService'>
     <sdl:Definitions.cmpMessage>
       <sdl:SimpleMessage xmi.id = 'a2' ElName = 'getMessageRequest'>
         <sdl:SimpleMessage.cmpPart>
           <sdl:Part xmi.id = 'a3' ElName = 'inputName'>
             <sdl:Part.refPart>
               <sdl:SdlString xmi.idref = 'a4'/>
             </sdl:Part.refPart>
           </sdl:Part>
         </sdl:SimpleMessage.cmpPart>
       </sdl:SimpleMessage>
       <sdl:SimpleMessage xmi.id = 'a5' ElName = 'getMessageResponse'>
         <sdl:SimpleMessage.cmpPart>
           <sdl:Part xmi.id = 'a6' ElName = 'returnMessage'>
             <sdl:Part.refPart>
               <sdl:SdlString xmi.idref = 'a4'/>
             </sdl:Part.refPart>
           </sdl:Part>
         </sdl:SimpleMessage.cmpPart>
       </sdl:SimpleMessage>
     </sdl:Definitions.cmpMessage>
     <sdl:Definitions.cmpInterface>
       <sdl:Interface xmi.id = 'a7' ElName = 'EchoNameServiceService'>
         <sdl:Interface.cmpOperation>
           <sdl:Operation xmi.id = 'a8' ElName = 'getMessage'>
             <sdl:Operation.refInputMessage>
               <sdl:SimpleMessage xmi.idref = 'a2'/>
             </sdl:Operation.refInputMessage>
             <sdl:Operation.refOutputMessage>
               <sdl:SimpleMessage xmi.idref = 'a5'/>
             </sdl:Operation.refOutputMessage>
           </sdl:Operation>
         </sdl:Interface.cmpOperation>
```

```
      </sdl:Interface>
    </sdl:Definitions.cmpInterface>
    <sdl:Definitions.cmpType>
      <sdl:SdlInteger xmi.id = 'a9' ElName = 'Integer'/>
      <sdl:SdlBoolean xmi.id = 'a10' ElName = 'Boolean'/>
      <sdl:SdlReal xmi.id = 'a11' ElName = 'Real'/>
      <sdl:SdlString xmi.id = 'a4' ElName = 'String'/>
      <sdl:SdlDateTime xmi.id = 'a12' ElName = 'DateTime'/>
      <sdl:SdlUri xmi.id = 'a13' ElName = 'Uri'/>
    </sdl:Definitions.cmpType>
  </sdl:Definitions>
 </XMI.content>
</XMI>
```

6.2 PC sale

6.2.1 SSL description

The SSL model presented below contains the description for the service PCWholesaler, that contains the following functionalities:

- getProductCategories
  - inputs: -
  - outputs: categoryList (string)
- getProductList
  - inputs: categoryName (string)
  - outputs: productList (string)
- getProductInfo
  - inputs: productName (string)
  - outputs: productID (string), description (string), costPerUnit (string), imageURL (string)
- purchase
  - inputs: productID (string), quantity (integer number), customerName (string), customerEmail (string), resellerId (string)
  - outputs: confirmation (string).

All the types are specified as XML Schema types using SSL DataTypeURI elements.

```
<?xml version="1.0" encoding="UTF-8"?>
  <XMI xmi.version="1.2" timestamp="Wed Jan 18 08:57:42 GMT 2006">
    <XMI.header>
      <XMI.documentation>
        <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
        <XMI.exporterVersion>1.0</XMI.exporterVersion>
      </XMI.documentation>
    </XMI.header>
    <XMI.content>
      <SSL.Core.SemanticPackage xmi.id="a1" name="LakeComponentPCWholesaler"
id="http://www.lakecomponents.com/LakeComponentPCWholesaler" Documentation="">
        <SSL.Core.NameSpace.OwnedElement>
          <SSL.Core.ServiceProfile xmi.id="a2" name="PCWholesaler"
id="1137574625164">
```

```
            <SSL.Core.ServiceConcept.Position>
              <SSL.Core.XYCoordinates xmi.id="a3" X="143" Y="54"/>
            </SSL.Core.ServiceConcept.Position>
            <SSL.Core.ServiceProfile.Attribute>
              <SSL.Core.ServiceAttribute xmi.id="a4" name="name"
id="1137574625165" Documentation="">
                <SSL.Core.ServiceAttribute.AttributeType>
                  <SSL.Types.DataTypeURI xmi.id="a5"
lexicalform="http://www.w3c.org/2001/XMLSchema#int"/>
                </SSL.Core.ServiceAttribute.AttributeType>
              </SSL.Core.ServiceAttribute>
            </SSL.Core.ServiceProfile.Attribute>
            <SSL.Core.ServiceProfile.Functionality>
              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a6"
name="getProductCategories" id="1137574625166" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a7"
name="categoryList" id="1137574625167" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a8"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                </SSL.ServiceBehavior.ServiceFunctionality.Output>
              </SSL.ServiceBehavior.ServiceFunctionality>
              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a9"
name="getProductList" id="1137574625168" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a10"
name="productList" id="1137574625170" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a11"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                </SSL.ServiceBehavior.ServiceFunctionality.Output>
                <SSL.ServiceBehavior.ServiceFunctionality.Input>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a12"
name="categoryName" id="1137574625169" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a13"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                </SSL.ServiceBehavior.ServiceFunctionality.Input>
              </SSL.ServiceBehavior.ServiceFunctionality>
              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a14"
name="getProductInfo" id="1137574625171" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a15"
name="productID" id="1137574625173" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a16"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a17"
name="description" id="1137574625174" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
```

```
                        <SSL.Types.DataTypeURI xmi.id="a18"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceOutput>
                    <SSL.ServiceBehavior.ServiceOutput xmi.id="a19"
name="costPerUnit" id="1137574625175" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a20"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceOutput>
                    <SSL.ServiceBehavior.ServiceOutput xmi.id="a21" name="imageURL"
id="1137574625176" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a22"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceOutput>
                  </SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceFunctionality.Input>
                    <SSL.ServiceBehavior.ServiceInput xmi.id="a23"
name="productName" id="1137574625172" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a24"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceInput>
                  </SSL.ServiceBehavior.ServiceFunctionality.Input>
                </SSL.ServiceBehavior.ServiceFunctionality>
                <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a25"
name="purchase" id="1137574625177" Documentation="">
                  <SSL.ServiceBehavior.ServiceFunctionality.Output>
                    <SSL.ServiceBehavior.ServiceOutput xmi.id="a26"
name="confirmation" id="1137574625183" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a27"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceOutput>
                  </SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceFunctionality.Input>
                    <SSL.ServiceBehavior.ServiceInput xmi.id="a28" name="productID"
id="1137574625178" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a29"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceInput>
                    <SSL.ServiceBehavior.ServiceInput xmi.id="a30" name="quantity"
id="1137574625179" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a31"
lexicalform="http://www.w3c.org/2001/XMLSchema#int"/>
                      </SSL.ServiceBehavior.FunctionalityParameter.Type>
                    </SSL.ServiceBehavior.ServiceInput>
                    <SSL.ServiceBehavior.ServiceInput xmi.id="a32"
name="customerName" id="1137574625180" Documentation="">
                      <SSL.ServiceBehavior.FunctionalityParameter.Type>
                        <SSL.Types.DataTypeURI xmi.id="a33"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
```

```
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a34"
name="customerEmail" id="1137574625181" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a35"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a36"
name="resellerID" id="1137574625182" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a37"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                </SSL.ServiceBehavior.ServiceFunctionality.Input>
              </SSL.ServiceBehavior.ServiceFunctionality>
            </SSL.Core.ServiceProfile.Functionality>
          </SSL.Core.ServiceProfile>
        </SSL.Core.NameSpace.OwnedElement>
      </SSL.Core.SemanticPackage>
    </XMI.content>
  </XMI>
```

6.2.2 SDL description

The SDL model that is created by the SSL compiler for the PCWholesaler service above is presented below. Note the way in which the multiple outputs of the functionality getProductInfo are grouped together as parts of the same simple message (named getProductInfoResponse), and how the multiple inputs of the purchase functionality are grouped in a simple message (named purchaseRequest).

```
<?xml version = '1.0' encoding = 'windows-1252' ?>
<XMI xmi.version = '1.2' xmlns:sdl = 'org.omg.xmi.namespace.sdl' timestamp = 'Thu
Jul 06 12:53:42 EEST 2006'>
 <XMI.header>
   <XMI.documentation>
     <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
   <sdl:Definitions xmi.id = 'a1' ElName = 'PCWholesaler'>
     <sdl:Definitions.cmpMessage>
       <sdl:SimpleMessage xmi.id = 'a2' ElName = 'getProductInfoRequest'>
         <sdl:SimpleMessage.cmpPart>
           <sdl:Part xmi.id = 'a3' ElName = 'productName'>
             <sdl:Part.refPart>
               <sdl:SdlString xmi.idref = 'a4'/>
             </sdl:Part.refPart>
           </sdl:Part>
         </sdl:SimpleMessage.cmpPart>
       </sdl:SimpleMessage>
       <sdl:SimpleMessage xmi.id = 'a5' ElName = 'getProductListRequest'>
         <sdl:SimpleMessage.cmpPart>
           <sdl:Part xmi.id = 'a6' ElName = 'categoryName'>
             <sdl:Part.refPart>
```

```
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
      </sdl:SimpleMessage.cmpPart>
    </sdl:SimpleMessage>
    <sdl:SimpleMessage xmi.id = 'a7' ElName = 'getProductListResponse'>
      <sdl:SimpleMessage.cmpPart>
        <sdl:Part xmi.id = 'a8' ElName = 'productList'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
      </sdl:SimpleMessage.cmpPart>
    </sdl:SimpleMessage>
    <sdl:SimpleMessage xmi.id = 'a9' ElName = 'purchaseResponse'>
      <sdl:SimpleMessage.cmpPart>
        <sdl:Part xmi.id = 'a10' ElName = 'confirmation'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
           </sdl:Part.refPart>
        </sdl:Part>
      </sdl:SimpleMessage.cmpPart>
    </sdl:SimpleMessage>
    <sdl:SimpleMessage xmi.id = 'a11' ElName = 'getProductCategoriesResponse'>
      <sdl:SimpleMessage.cmpPart>
        <sdl:Part xmi.id = 'a12' ElName = 'categoryList'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
      </sdl:SimpleMessage.cmpPart>
    </sdl:SimpleMessage>
    <sdl:SimpleMessage xmi.id = 'a13' ElName = 'purchaseRequest'>
      <sdl:SimpleMessage.cmpPart>
        <sdl:Part xmi.id = 'a14' ElName = 'resellerID'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
        <sdl:Part xmi.id = 'a15' ElName = 'customerEmail'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
        <sdl:Part xmi.id = 'a16' ElName = 'quantity'>
          <sdl:Part.refPart>
            <sdl:SdlInteger xmi.idref = 'a17'/>
          </sdl:Part.refPart>
        </sdl:Part>
        <sdl:Part xmi.id = 'a18' ElName = 'customerName'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
        <sdl:Part xmi.id = 'a19' ElName = 'productID'>
          <sdl:Part.refPart>
            <sdl:SdlString xmi.idref = 'a4'/>
          </sdl:Part.refPart>
        </sdl:Part>
      </sdl:SimpleMessage.cmpPart>
```

```
      </sdl:SimpleMessage>
      <sdl:SimpleMessage xmi.id = 'a20' ElName = 'getProductInfoResponse'>
        <sdl:SimpleMessage.cmpPart>
          <sdl:Part xmi.id = 'a21' ElName = 'productID'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a4'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a22' ElName = 'imageURL'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a4'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a23' ElName = 'description'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a4'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a24' ElName = 'costPerUnit'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a4'/>
            </sdl:Part.refPart>
          </sdl:Part>
        </sdl:SimpleMessage.cmpPart>
      </sdl:SimpleMessage>
    </sdl:Definitions.cmpMessage>
    <sdl:Definitions.cmpInterface>
      <sdl:Interface xmi.id = 'a25' ElName = 'PCWholesalerService'>
        <sdl:Interface.cmpOperation>
          <sdl:Operation xmi.id = 'a26' ElName = 'getProductList'>
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a5'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a7'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
          <sdl:Operation xmi.id = 'a27' ElName = 'getProductCategories'>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a11'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
          <sdl:Operation xmi.id = 'a28' ElName = 'purchase'>
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a13'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a9'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
          <sdl:Operation xmi.id = 'a29' ElName = 'getProductInfo'>
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a2'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a20'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
        </sdl:Interface.cmpOperation>
      </sdl:Interface>
```

```
      </sdl:Definitions.cmpInterface>
      <sdl:Definitions.cmpType>
        <sdl:SdlInteger xmi.id = 'a17' ElName = 'Integer'/>
        <sdl:SdlBoolean xmi.id = 'a30' ElName = 'Boolean'/>
        <sdl:SdlReal xmi.id = 'a31' ElName = 'Real'/>
        <sdl:SdlString xmi.id = 'a4' ElName = 'String'/>
        <sdl:SdlDateTime xmi.id = 'a32' ElName = 'DateTime'/>
        <sdl:SdlUri xmi.id = 'a33' ElName = 'Uri'/>
      </sdl:Definitions.cmpType>
    </sdl:Definitions>
  </XMI.content>
</XMI>
```

6.3 Hotel reservation

6.3.1 SSL description

The SSL model presented below, taken from [Deliverable D14.1], describes the semantics of a hotel reservation service. It defines the following functionalities:

- SimpleAvailabilityChecking
  - inputs: CheckinDate (datetime), CheckoutDate (datetime), RoomType (RoomID from ontology)
  - outputs:AvailabilityResponse (boolean)
- MakeReservation
  - inputs: ReservationRequest (ReservationRQID from ontology)
  - outputs:ReservationResponse (ReservationResponseID from ontology)
- CancelReservation
  - input: CancelReservationRequest (CancellationRQID from ontology)
  - output: CancelationResponse (boolean).As it can be seen, the model refers to classes defined in an external ontology that has the id tourismOntologyID, that is presented below in chapter 6.3.2 - ODM description. When the transformation is run, the ontology is retrieved from the Knowledge Base and the information that it contains can be used. For each SSL ServiceInput or SSL ServiceOutput that contains a reference to an ontology class, the ontology can be used to infer the structure of the messages and their types in the generated SDL model.

```
<?xml version="1.0" encoding="UTF-8"?>
<XMI xmi.version="1.2" timestamp="Mon Oct 31 16:49:11 EET 2005">
      <XMI.header>
        <XMI.documentation>
          <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
          <XMI.exporterVersion>1.0</XMI.exporterVersion>
        </XMI.documentation>
      </XMI.header>
      <XMI.content>
        <SSL.Core.SemanticPackage xmi.id="a1" name="ZaragozaHotelModel"
id="http://www.mySME.com/ZaragozaHotelModel" Documentation="">
          <SSL.Core.NameSpace.OwnedElement>

            <!-- removed elements not of interest for the transformation -->
```

```
          <SSL.Core.ServiceProfile xmi.id="a6" name="HotelReservation"
id="1130769993287">
            <SSL.Core.ServiceConcept.Position>
              <SSL.Core.XYCoordinates xmi.id="a57" X="66" Y="79"/>
            </SSL.Core.ServiceConcept.Position>
            <SSL.Core.ServiceProfile.Domain>
              <SSL.Core.DBEServiceDomain xmi.idref="a53"/>
              <SSL.Core.DBEServiceDomain xmi.idref="a55"/>
            </SSL.Core.ServiceProfile.Domain>
            <SSL.Core.ServiceProfile.Attribute>
              <SSL.Core.ServiceAttribute xmi.id="a58" name="ServiceName"
id="1130769993290" Documentation="">
                <SSL.Core.ServiceAttribute.AttributeType>
                  <SSL.Types.DataTypeURI xmi.id="a59"
lexicalform="http://www.w3c.org/2001/XMLSchema#string"/>
                </SSL.Core.ServiceAttribute.AttributeType>
              </SSL.Core.ServiceAttribute>
              <SSL.Core.ServiceAttribute xmi.id="a60" name="Provider"
id="1130769993291" Documentation="">
                <SSL.Core.ServiceAttribute.AttributeType>
                  <SSL.Types.OntologyClassURI xmi.id="a61"
lexicalform="tourismOntologyID#HotelID"/>
                </SSL.Core.ServiceAttribute.AttributeType>
              </SSL.Core.ServiceAttribute>
            </SSL.Core.ServiceProfile.Attribute>
            <SSL.Core.ServiceProfile.Functionality>
              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a62"
name="SimpleAvailabilityChecking" id="1130769993292" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a63"
name="AvailabilityResponse" id="1130769993296" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a64"
lexicalform="http://www.w3c.org/2001/XMLSchema#boolean"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                </SSL.ServiceBehavior.ServiceFunctionality.Output>
                <SSL.ServiceBehavior.ServiceFunctionality.Input>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a65"
name="CheckinDate" id="1130769993293" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a66"
lexicalform="http://www.w3c.org/2001/XMLSchema#dateTime"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a67"
name="CheckoutDate" id="1130769993294" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a68"
lexicalform="http://www.w3c.org/2001/XMLSchema#dateTime"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a69" name="RoomType"
id="1130769993295" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.OntologyClassURI xmi.id="a70"
lexicalform="tourismOntologyID#RoomID"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
```

```
                </SSL.ServiceBehavior.ServiceFunctionality.Input>
              </SSL.ServiceBehavior.ServiceFunctionality>

              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a76"
name="MakeReservation" id="1130769993300" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a77"
name="ReservationResponse" id="1130769993302" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.OntologyClassURI xmi.id="a78"
lexicalform="tourismOntologyID#ReservationResponseID"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                </SSL.ServiceBehavior.ServiceFunctionality.Output>
                <SSL.ServiceBehavior.ServiceFunctionality.Input>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a79"
name="ReservationRequest" id="1130769993301" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.OntologyClassURI xmi.id="a80"
lexicalform="tourismOntologyID#ReservationRQID"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                </SSL.ServiceBehavior.ServiceFunctionality.Input>
              </SSL.ServiceBehavior.ServiceFunctionality>
              <SSL.ServiceBehavior.ServiceFunctionality xmi.id="a81"
name="CancelReservation" id="1130769993303" Documentation="">
                <SSL.ServiceBehavior.ServiceFunctionality.Output>
                  <SSL.ServiceBehavior.ServiceOutput xmi.id="a82"
name="CancelationResponse" id="1130769993305" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.DataTypeURI xmi.id="a83"
lexicalform="http://www.w3c.org/2001/XMLSchema#boolean"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceOutput>
                </SSL.ServiceBehavior.ServiceFunctionality.Output>
                <SSL.ServiceBehavior.ServiceFunctionality.Input>
                  <SSL.ServiceBehavior.ServiceInput xmi.id="a84"
name="CancelReservationRequest" id="1130769993304" Documentation="">
                    <SSL.ServiceBehavior.FunctionalityParameter.Type>
                      <SSL.Types.OntologyClassURI xmi.id="a85"
lexicalform="tourismOntologyID#CancellationRQID"/>
                    </SSL.ServiceBehavior.FunctionalityParameter.Type>
                  </SSL.ServiceBehavior.ServiceInput>
                </SSL.ServiceBehavior.ServiceFunctionality.Input>
              </SSL.ServiceBehavior.ServiceFunctionality>
            </SSL.Core.ServiceProfile.Functionality>
          </SSL.Core.ServiceProfile>
        </SSL.Core.NameSpace.OwnedElement>
      </SSL.Core.SemanticPackage>
    </XMI.content>
  </XMI>
```

6.3.2 ODM description

In this example the transformation makes use of the tourism domain ontology as it contains some classes that are referred by the Hotel

reservation SSL model; these classes are shown in Table 2: Ontology classes referred in the SSL model, below.

ClassID	Referred by	Referrer type
RoomID	RoomType	SSL ServiceInput
ReservationRQID	ReservationRequest	SSL ServiceInput
ReservationResponseID	ReservationResponse	SSL ServiceOutput
CancellationRQID	CancelReservationRequest	SSL ServiceInput

Table 2: Ontology classes referred in the SSL model

The datatype properties for each of these classes are described in Table 3: Datatype properties for referred classes below.

ClassID	Property Name	Property Type
RoomID	RoomAdditionalInfo	XMLSchema#string
	RoomType	RoomTypeEnumeration
ReservationRQID	ArrivalIflight	XMLSchema#integer
	CheckinDate	XMLSchema#string
	CheckoutDate	XMLSchema#string
	NumofAdults	XMLSchema#integer
	NumofNights	XMLSchema#integer
	NumOfRooms	XMLSchema#integer
	RoomType	RoomTypeEnumeration
ReservationResponseID		
CancellationRQID	ReservationNumber	XMLSchema#string

Table 3: Datatype properties for referred classes

The most relevant parts of the tourism domain ontology is presented below.

```
<?xml version="1.0" encoding="UTF-8"?>
<XMI xmi.version="1.2" timestamp="Mon Dec 06 14:43:00 GMT 2004">
.......
 <XMI.content>
   <ODM.Ontology.Ontology xmi.id="a1" name="tourismOntology"
id="tourismOntologyID">
     <ODM.Core.NameSpace.OwnedElement>

       <ODM.Classes.Class xmi.id="a32" name="Destination" id="DestinationID"
complete="false"/>
       <ODM.Classes.Class xmi.id="a52" name="Address" id="AddressID"
complete="false"/>
       <ODM.Classes.Class xmi.id="a53" name="Country" id="CountryID"
complete="false"/>
       <ODM.Classes.Class xmi.id="a68" name="Room" id="RoomID" complete="false"/>
       <ODM.Classes.Class xmi.id="a76" name="Response" id="ResponseID"
complete="false"/>
```

```
       <ODM.Classes.Class xmi.id="a77" name="CancellationResponse"
id="CancellationResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a76"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a78" name="CancellationFailureResponse"
id="CancellationFailureResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a77"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a79" name="CancellationSuccessResponse"
id="CancellationSuccessResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a77"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a80" name="ReservationResponse"
id="ReservationResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a76"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a81" name="ReservationSuccessResponse"
id="ReservationSuccessResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a80"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a82" name="ReservationFailureResponse"
id="ReservationFailureResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a80"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a83" name="AvailabilityResponse"
id="AvailabilityResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a76"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a84" name="SimpleAvailabilityResponse"
id="SimpleAvailabilityResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a83"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a85" name="SpecialAvailabilityResponse"
id="SpecialAvailabilityResponseID" complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a83"/>
         </ODM.Classes.Class.theSuperClass>
       </ODM.Classes.Class>
       <ODM.Classes.Class xmi.id="a86" name="Request" id="RequestID"
complete="false"/>
       <ODM.Classes.Class xmi.id="a87" name="ReservationRQ" id="ReservationRQID"
complete="false">
         <ODM.Classes.Class.theSuperClass>
           <ODM.Classes.Class xmi.idref="a86"/>
```

```
          </ODM.Classes.Class.theSuperClass>
        </ODM.Classes.Class>
        <ODM.Classes.Class xmi.id="a88" name="ReservationRecord"
id="ReservationRecordID" complete="false">
          <ODM.Classes.Class.theSuperClass>
            <ODM.Classes.Class xmi.idref="a87"/>
          </ODM.Classes.Class.theSuperClass>
          <ODM.Classes.Class.theSuperClass>
            <ODM.Classes.Class xmi.idref="a70"/>
          </ODM.Classes.Class.theSuperClass>
        </ODM.Classes.Class>
        <ODM.Classes.Class xmi.id="a89" name="AvailabilityRequest"
id="AvailabilityRequestID" complete="false">
          <ODM.Classes.Class.theSuperClass>
            <ODM.Classes.Class xmi.idref="a86"/>
          </ODM.Classes.Class.theSuperClass>
        </ODM.Classes.Class>
        <ODM.Classes.Class xmi.id="a90" name="CancellationRQ" id="CancellationRQID"
complete="false">
          <ODM.Classes.Class.theSuperClass>
            <ODM.Classes.Class xmi.idref="a86"/>
          </ODM.Classes.Class.theSuperClass>
        </ODM.Classes.Class>
        <ODM.Properties.DatatypeProperty xmi.id="a109" name="Availability"
id="AvailabilityID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a84"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a17"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a113" name="CheckinDate"
id="CheckinDateID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a87"/>
            <ODM.Classes.Class xmi.idref="a89"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a15"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a114" name="CheckoutDate"
id="CheckoutDateID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a87"/>
            <ODM.Classes.Class xmi.idref="a89"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a15"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a131" name="NumberOfRooms"
id="NumberOfRoomsID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a20"/>
            <ODM.Classes.Class xmi.idref="a89"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a16"/>
```

```
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a132" name="NumofAdults"
id="NumofAdultsID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a87"/>
            <ODM.Classes.Class xmi.idref="a89"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a16"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a133" name="NumofNights"
id="NumofNightsID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a87"/>
            <ODM.Classes.Class xmi.idref="a89"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a16"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a134" name="NomOfRooms"
id="NomOfRoomsID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a87"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a16"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
        <ODM.Properties.DatatypeProperty xmi.id="a137" name="ReservationNumber"
id="ReservationNumberID">
          <ODM.Properties.Property.theDomain>
            <ODM.Classes.Class xmi.idref="a81"/>
            <ODM.Classes.Class xmi.idref="a88"/>
            <ODM.Classes.Class xmi.idref="a90"/>
          </ODM.Properties.Property.theDomain>
          <ODM.Properties.DatatypeProperty.theDataRange>
            <ODM.Datatypes.PrimitiveType xmi.idref="a15"/>
          </ODM.Properties.DatatypeProperty.theDataRange>
        </ODM.Properties.DatatypeProperty>
      </ODM.Core.NameSpace.OwnedElement>
    </ODM.Ontology.Ontology>
  </XMI.content>
</XMI>
```

6.3.3 SDL description

This SDL model illustrates the SSL to SDL transformation applied to an SSL model that contains references to ontology classes. As it can be seen in the generated model and also in Table 4: Generated complex types for referred ontology classes, below, the transformation creates SDL ComplexType elements for the referred ontology classes. From each datatype property for a certain class, an SDL Part is created in the corresponding complex type.

Therefore, the type information contained in the ontology is retrieved for the referred classes and it is captured in the generated SDL model.

ClassID	*Property Name*	*Property Type*	*Created complex type*
RoomID	RoomAdditionalInfo	XMLSchema#string	RoomTypeParameters
	RoomType	RoomTypeEnumeration	
ReservationRQID	Arrivallflight	XMLSchema#integer	ReservationRequestParameters
	CheckinDate	XMLSchema#string	
	CheckoutDate	XMLSchema#string	
	NumofAdults	XMLSchema#integer	
	NumofNights	XMLSchema#integer	
	NumOfRooms	XMLSchema#integer	
	RoomType	RoomTypeEnumeration	
ReservationResponseID			ReservationResponseParameters
CancellationRQID	ReservationNumber	XMLSchema#string	CancelReservationRequestParameters

Table 4: Generated complex types for referred ontology classes

```
<?xml version = '1.0' encoding = 'windows-1252' ?>
<XMI xmi.version = '1.2' xmlns:sdl = 'org.omg.xmi.namespace.sdl' timestamp = 'Tue
Jul 11 13:27:48 EEST 2006'>
 <XMI.header>
   <XMI.documentation>
     <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
   <sdl:Definitions xmi.id = 'a1' ElName = 'HotelReservation'>
     <sdl:Definitions.cmpMessage>
       <sdl:SimpleMessage xmi.id = 'a2' ElName = 'CancelReservationRequest'>
         <sdl:SimpleMessage.cmpPart>
           <sdl:Part xmi.id = 'a3' ElName = 'CancelReservationRequestComplexType'>
             <sdl:Part.refPart>
               <sdl:ComplexType xmi.idref = 'a4'/>
             </sdl:Part.refPart>
           </sdl:Part>
         </sdl:SimpleMessage.cmpPart>
       </sdl:SimpleMessage>
       <sdl:SimpleMessage xmi.id = 'a5' ElName = 'MakeReservationRequest'>
```

```
          <sdl:SimpleMessage.cmpPart>
            <sdl:Part xmi.id = 'a6' ElName = 'ReservationRequestComplexType'>
              <sdl:Part.refPart>
                <sdl:ComplexType xmi.idref = 'a7'/>
              </sdl:Part.refPart>
            </sdl:Part>
          </sdl:SimpleMessage.cmpPart>
        </sdl:SimpleMessage>
        <sdl:SimpleMessage xmi.id = 'a8' ElName = 'MakeReservationResponse'>
          <sdl:SimpleMessage.cmpPart>
            <sdl:Part xmi.id = 'a9' ElName = 'ReservationResponseComplexType'>
              <sdl:Part.refPart>
                <sdl:ComplexType xmi.idref = 'a10'/>
              </sdl:Part.refPart>
            </sdl:Part>
          </sdl:SimpleMessage.cmpPart>
        </sdl:SimpleMessage>
        <sdl:SimpleMessage xmi.id = 'a11' ElName = 'SimplebilityCheckingResponse'>
          <sdl:SimpleMessage.cmpPart>
            <sdl:Part xmi.id = 'a12' ElName = 'AvailabilityResponse'>
              <sdl:Part.refPart>
                <sdl:SdlBoolean xmi.idref = 'a16'/>
               </sdl:Part.refPart>
            </sdl:Part>
          </sdl:SimpleMessage.cmpPart>
        </sdl:SimpleMessage>
        <sdl:SimpleMessage xmi.id = 'a14' ElName = 'CancelReservationResponse'>
          <sdl:SimpleMessage.cmpPart>
            <sdl:Part xmi.id = 'a15' ElName = 'CancelationResponse'>
              <sdl:Part.refPart>
                <sdl:SdlBoolean xmi.idref = 'a16'/>
              </sdl:Part.refPart>
            </sdl:Part>
          </sdl:SimpleMessage.cmpPart>
        </sdl:SimpleMessage>
        <sdl:SimpleMessage xmi.id = 'a17' ElName =
'SimpleAvailabilityCheckingRequest'>
          <sdl:SimpleMessage.cmpPart>
            <sdl:Part xmi.id = 'a18' ElName = 'CheckoutDate'>
              <sdl:Part.refPart>
                <sdl:SdlDateTime xmi.idref = 'a13'/>
              </sdl:Part.refPart>
            </sdl:Part>
            <sdl:Part xmi.id = 'a19' ElName = 'RoomType'>
              <sdl:Part.refPart>
                <sdl:ComplexType xmi.idref = 'a20'/>
              </sdl:Part.refPart>
            </sdl:Part>
            <sdl:Part xmi.id = 'a21' ElName = 'CheckinDate'>
              <sdl:Part.refPart>
                <sdl:SdlDateTime xmi.idref = 'a13'/>
              </sdl:Part.refPart>
            </sdl:Part>
          </sdl:SimpleMessage.cmpPart>
        </sdl:SimpleMessage>
      </sdl:Definitions.cmpMessage>
      <sdl:Definitions.cmpInterface>
        <sdl:Interface xmi.id = 'a22' ElName = 'HotelReservationService'>
          <sdl:Interface.cmpOperation>
            <sdl:Operation xmi.id = 'a23' ElName = 'SimpleAvailabilityChecking'>
```

```
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a17'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a11'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
          <sdl:Operation xmi.id = 'a24' ElName = 'CancelReservation'>
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a2'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a14'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
          <sdl:Operation xmi.id = 'a25' ElName = 'MakeReservation'>
            <sdl:Operation.refInputMessage>
              <sdl:SimpleMessage xmi.idref = 'a5'/>
            </sdl:Operation.refInputMessage>
            <sdl:Operation.refOutputMessage>
              <sdl:SimpleMessage xmi.idref = 'a8'/>
            </sdl:Operation.refOutputMessage>
          </sdl:Operation>
        </sdl:Interface.cmpOperation>
      </sdl:Interface>
    </sdl:Definitions.cmpInterface>
    <sdl:Definitions.cmpType>
      <sdl:SdlInteger xmi.id = 'a26' ElName = 'Integer'/>
      <sdl:SdlBoolean xmi.id = 'a16' ElName = 'Boolean'/>
      <sdl:SdlReal xmi.id = 'a27' ElName = 'Real'/>
      <sdl:SdlString xmi.id = 'a28' ElName = 'String'/>
      <sdl:SdlDateTime xmi.id = 'a13' ElName = 'DateTime'/>
      <sdl:SdlUri xmi.id = 'a29' ElName = 'Uri'/>
      <sdl:ComplexType xmi.id = 'a10' ElName = 'ReservationResponseParameters'>
        <sdl:ComplexType.cmpPart>
          <sdl:Part xmi.id = 'a30' ElName = 'ReservationResponseID'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
        </sdl:ComplexType.cmpPart>
      </sdl:ComplexType>
      <sdl:ComplexType xmi.id = 'a4' ElName =
'CancelReservationRequestParameters'>
        <sdl:ComplexType.cmpPart>
          <sdl:Part xmi.id = 'a31' ElName = 'ReservationNumber'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
        </sdl:ComplexType.cmpPart>
      </sdl:ComplexType>
      <sdl:ComplexType xmi.id = 'a7' ElName = 'ReservationRequestParameters'>
        <sdl:ComplexType.cmpPart>
          <sdl:Part xmi.id = 'a32' ElName = 'NumOfRooms'>
            <sdl:Part.refPart>
              <sdl:SdlInteger xmi.idref = 'a26'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a33' ElName = 'RoomType'>
```

```
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a34' ElName = 'CheckinDate'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a35' ElName = 'CheckoutDate'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a36' ElName = 'Arrivallflight'>
            <sdl:Part.refPart>
              <sdl:SdlInteger xmi.idref = 'a26'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a37' ElName = 'NumofNights'>
            <sdl:Part.refPart>
              <sdl:SdlInteger xmi.idref = 'a26'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a38' ElName = 'NumofAdults'>
            <sdl:Part.refPart>
              <sdl:SdlInteger xmi.idref = 'a26'/>
            </sdl:Part.refPart>
          </sdl:Part>
        </sdl:ComplexType.cmpPart>
      </sdl:ComplexType>
      <sdl:ComplexType xmi.id = 'a20' ElName = 'RoomTypeParameters'>
        <sdl:ComplexType.cmpPart>
          <sdl:Part xmi.id = 'a39' ElName = 'RoomType'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
          <sdl:Part xmi.id = 'a40' ElName = 'RoomAdditionalInfo'>
            <sdl:Part.refPart>
              <sdl:SdlString xmi.idref = 'a28'/>
            </sdl:Part.refPart>
          </sdl:Part>
        </sdl:ComplexType.cmpPart>
      </sdl:ComplexType>
    </sdl:Definitions.cmpType>
  </sdl:Definitions>
 </XMI.content>
</XMI>
```

Figure 9: Hotel reservation example below illustrates the model as represented in the SDL Editor.

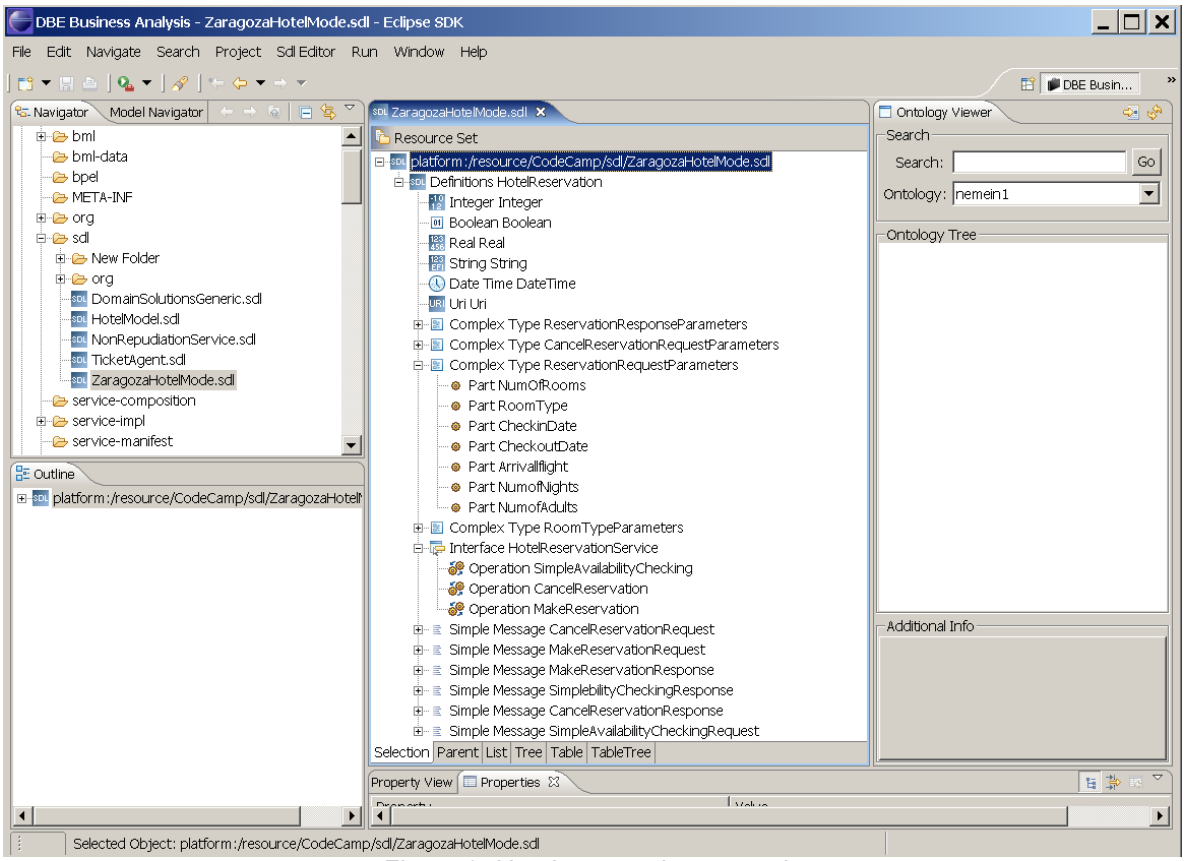

Figure 9: Hotel reservation example

7. Appendixes

7.1 Transformation code (ATL)

```
module SSL2SDL;
create OUT : SDL from IN : SSL, IN2 : ODM;

-- A SSL!OntologyClassURI element has a lexicalForm attribute with
-- the following structure: <ontologyID>#<ontologyClassID>.
-- This helper takes this value and returns the <ontologyClassID> part.
helper def : getOntologyIDName(e : String) : String =
      e.substring(e.indexOf('#') + 2, e.size())
;

-- A SSL!DataTypeURI element has a lexicalForm attribute that
-- contains the URI that points to a XMLSchema and to the desired type,
-- in the form: <baseURI>#<typeName>
-- This helper takes this value and returns the <typeName> part.
helper def : getSchemaTypeName(e : String) : String =
      e.substring(e.indexOf('#') + 2, e.size())
;

-- Taking a String value, this helper returns a sequence containing all the
-- ODM!DatatypeProperty elements that apply to the ODM!Class that has the id
-- attribute equal with that value. The returned sequence includes the
-- ODM!DatatypeProperty-es that apply to any of its superclasses (the inherited
-- ODM!DatatypeProperty-es).
helper def: getDataTypePropertiesForId(inputId : String) :
Sequence(ODM!DatatypeProperty) =
      ODM!Class.allInstances() -> select (d | d.id = inputId) ->
            collect (d | thisModule.getAllDataTypeProperties(d)) -> flatten()
;

-- Taking a ODM!Class, returns a sequence containing all the ODM!DatatypeProperty
-- elements that have in their theDomain property a reference to that class or to
-- one of its superclasses (recursion on all the defined super classes)
helper def: getAllDataTypeProperties(c : ODM!Class) :
Sequence(ODM!DatatypeProperty) =
      Sequence{}
      -> append (thisModule.getOwnDataTypeProperties(c))
      -> append ( c.theSuperClass ->
                        iterate(e; result : Sequence(ODM!DatatypeProperty)
                        = Sequence{} |
                        result -> append(thisModule.getAllDataTypeProperties(e)) ->
                        flatten())
         ) -> flatten()
;

-- Taking a ODM!Class, returns a sequence containing all the ODM!DatatypeProperty
-- elements that have in their theDomain property a reference to that class.
helper def: getOwnDataTypeProperties(c : ODM!Class) :
Sequence(ODM!DatatypeProperty) =
      ODM!DatatypeProperty.allInstances() -> select(e | e.theDomain
                  -> includes (c))
;
```

```
-- This helper returns a sequence containing all the SSL!ServiceInput
-- elements that belong to a SSL!ServiceFunctionality that has
-- multiple inputs
helper def: getMultipleInputs() : Sequence(SSL!ServiceInput) =
      SSL!ServiceFunctionality.allInstances() -> select(e | e.Input.size() > 1)
      -> collect(e | e.Input) -> flatten()
;

-- Similar with getMultipleInputs(), but referring to SSL!ServiceOutput.
helper def: getMultipleOutputs() : Sequence(SSL!ServiceOutput) =
      SSL!ServiceFunctionality.allInstances() -> select(e | e.Output.size() > 1)
      -> collect(e | e.Output) -> flatten()
;

-- Returns the SSL!ServiceFunctionality that has, as one of its inputs, the
SSL!ServiceInput provided as parameter
helper def: getServiceFunctionalityHavingInput(i : SSL!ServiceInput) :
SSL!ServiceFunctionality =
      SSL!ServiceFunctionality.allInstances() -> select(sf | sf.Input ->
            includes (i)) -> asSequence() -> first()
;

-- Returns the SSL!ServiceFunctionality that contains, as one of its outputs, the
SSL!ServiceOutput provided as parameter
helper def: getServiceFunctionalityHavingOutput(o : SSL!ServiceOutput) :
SSL!ServiceFunctionality =
      SSL!ServiceFunctionality.allInstances() -> select(sf | sf.Output ->
            includes (o)) -> asSequence() -> first()
;

-- This helper returns the name of the SSL!ServiceInput that has as
-- Type the specified SSL!OntologyClassURI.
helper def: inputMessageName(searchedType: SSL!OntologyClassURI) : String =
     SSL!ServiceInput.allInstances() -> select(e | e.Type = searchedType) ->
            collect(e | e.name) -> asSequence() -> first()
;

-- Similar with inputMessageName(), but referring to SSL!ServiceOutput.
helper def: outputMessageName(searchedType: SSL!OntologyClassURI) : String =
     SSL!ServiceOutput.allInstances() -> select(e | e.Type = searchedType) ->
            collect(e | e.name) -> asSequence() -> first()
;

-- Returns the root element of the input SSL model, as it is the one
-- that generates the SDL types.
helper def: getTypeGenerator() : SSL!ServiceProfile =
            SSL!ServiceProfile.allInstances()->asSequence()->first()
;

-- Returns the name of the target pattern (in rule Definitions) that
-- generates the SDL type that corresponds to the specified ODM!DataRange.
-- For the types that do not have direct mapping to an SDL type, the
-- returned target pattern name is the one returned by the helper
-- getSDLTypeForODM_Default.
helper def: getSDLTypeForODM(p : ODM!DataRange) : String =
      if p.oclIsKindOf(ODM!PrimitiveType) then
            thisModule.getSDLTypeForSchemaType(
                  thisModule.getSchemaTypeName(p.TypeDefinitionURI.lexicalForm))
      else thisModule.getSDLTypeForODM_Default()
```

```
      endif
;

-- Returns the name of the target pattern (in rule Definitions) that is
-- assigned by default to elements from the ontology whose type is not mapped
-- directly to a SDL type (the pattern that generates SDL!SdlString).
helper def: getSDLTypeForODM_Default() : String =
      'type4'
;

-- Returns the name of the target pattern (in rule Definitions) that
-- generates the SDL type that corresponds to the schema type with the
-- specified name. (The mapping realized for the schema:
-- http://www.w3.org/2001/XMLSchema.)
-- For the types that do not have direct mapping to an SDL type, the
-- retured target pattern name is the one corresponding to SDL!String.
helper def: getSDLTypeForSchemaType(name : String) : String =
      if  name = 'integer' or name = 'positiveInteger' or
            name = 'nonPositiveInteger' or name = 'negativeInteger' or
            name = 'nonNegativeInteger' or name = 'long' or
            name = 'int' or name = 'short' or name = 'byte' or
            name = 'unsignedLong' or name = 'unsignedInt' or
            name = 'unsignedShort' or name = 'unsignedByte'
      then 'type1'      -- SDL!SdlInteger
      else if name = 'boolean'
             then 'type2'     -- SDL!SdlBoolean
             else if name = 'decimal' or name = 'float' or name = 'double'
                        then 'type3'      -- SDL!SdlReal
                        else if name = 'dateTime' or name = 'date' or
                                    name = 'time' or
                                    name = 'gYearMonth' or name = 'gMonthDay' or
                                    name = 'gYear' or name = 'gMonth' or
                                    name = 'gDay'
                               then 'type5'     -- SDL!SdlDateTime
                               else if name = 'string' or
                                    name = 'normalizedString' or
                                    name = 'token' or name = 'language' or
                                    name = 'NMTOKEN' or
                                    name = 'Name' or name = 'NCName' or
                                    name = 'hexBinary' or name = 'base64Binary'
                                    then 'type4'      --  SDL!SdlString
                                    else 'type4'      -- default for unsupported
                                                      -- types: SDL!SdlString
                                    endif
                               endif
                        endif
             endif
      endif
;

-- Returns a sequence with a 'fake' element if the size of the
-- provided sequence is 0, otherwise it returns an empty sequence.
helper def: getFakeDatatypeSequence(m : Sequence(ODM!DatatypeProperty)) :
Sequence(String) =
      if m.size() = 0
            then Sequence{''}
            else Sequence{}
      endif
```

```
;

-- Returns a sequence of all the SSL!ServiceInput and SSL!ServiceOutput
-- that have the Type attribute equal to the specified SSL!DataTypeURI.
helper def: getInputOutputHavingURIType(t : SSL!DataTypeURI) :
Sequence(SSL!FunctionalityParameter) =
      Sequence{SSL!ServiceInput.allInstances() -> asSequence(),
            SSL!ServiceOutput.allInstances() -> asSequence()} -> flatten() ->
            select(e | e.Type = t)
;

-- An SSL!OntologyClassURI element contains the id of an element in the
-- associated ontology.
-- For every SSL!OntologyClassURI element that is the Type attribute of a
-- SSL!ServiceInput, this rule creates:
-- 1) a SDL!ComplexType that has a SDL!Part for each ODM!DatatypeProperty
-- in which this class is reffered (by id). If there is no such
-- ODM!DatatypeProperty, by default a SDL!Part with the type SDL!String
-- is created
-- 2) if this element is the Type of a SSL!ServiceInput that belongs
-- to a SSL!ServiceFunctionality with multiple inputs, then a SDL!Part
-- that has a reference to the generated SDL!ComplexType is created
-- (and it will be included in the element obtained from the transformation
-- of that SSL!ServiceFunctionality)
rule TypeFromOntology_Input {
      from inp : SSL!OntologyClassURI (
          SSL!ServiceInput.allInstances() -> select(e | e.Type = inp) -> size() > 0
      )
      using{
            properties : Sequence(ODM!DatatypeProperty) =
                  thisModule.getDataTypePropertiesForId(
                        thisModule.getOntologyIDName( inp.lexicalform));
            service_name : SSL!ServiceInput = thisModule.inputMessageName(inp);
      }
      to complex : SDL!ComplexType(
            ElName <- service_name + 'Parameters',
            cmpPart <- Sequence{part_a, part_b}
      ),
      part_a : distinct SDL!Part foreach(a in properties)(
            ElName <- a.name,
            refPart <- thisModule.resolveTemp(thisModule.getTypeGenerator(),
                  thisModule.getSDLTypeForODM(a.theDataRange)
            )
      ),
      part_b : distinct SDL!Part foreach(c in
                  thisModule.getFakeDatatypeSequence(properties) ) (
            ElName <- thisModule.getOntologyIDName(inp.lexicalform),
            refPart <- thisModule.resolveTemp(thisModule.getTypeGenerator(),
                  thisModule.getSDLTypeForODM_Default()
            )
      ),

      part_c : distinct  SDL!Part foreach(b in
                  SSL!ServiceFunctionality.allInstances() ->
                  select(c | c.Input.size() > 1) -> select(v | v.Input ->
                        select(u | u.Type = inp).size() > 0))(
            ElName <- service_name,
            refPart <- complex
      )
}
```

```
-- The same transformation as TypeFromOntology_Input, but for types
-- that belong to a ServiceOutput.
rule TypeFromOntology_Output {
      from inp : SSL!OntologyClassURI (
         SSL!ServiceOutput.allInstances() -> select(e | e.Type = inp) -> size() > 0
      )
      using{
            properties : Sequence(ODM!DatatypeProperty) =
                  thisModule.getDataTypePropertiesForId(
                        thisModule.getOntologyIDName(inp.lexicalform));
            service_name : SSL!ServiceOutput = thisModule.outputMessageName(inp);
      }
      to complex : SDL!ComplexType(
            ElName <- service_name + 'Parameters',
            cmpPart <- Sequence{part_a, part_b}
      ),
      part_a : distinct SDL!Part foreach(a in properties)(
            ElName <- a.name,
            refPart <- thisModule.resolveTemp(thisModule.getTypeGenerator(),
                  thisModule.getSDLTypeForODM(a.theDataRange)
                  )
      ),
      part_b : distinct SDL!Part foreach (c in
                  thisModule.getFakeDatatypeSequence(properties)  )(
            ElName <- thisModule.getOntologyIDName(inp.lexicalform),
            refPart <- thisModule.resolveTemp(thisModule.getTypeGenerator(),
                  thisModule.getSDLTypeForODM_Default()
            )
      ),

      part_c : distinct  SDL!Part foreach(b in
                  SSL!ServiceFunctionality.allInstances() ->
                    select(c | c.Output.size() > 1) -> select(v | v.Output ->
                        select(u | u.Type = inp).size() > 0))(
            ElName <- service_name,
            refPart <- complex
      )
}

-- A SSL!DataTypeURI element refers to a type defined in a specified XMLSchema.
-- For every SSL!DataTypeURI element that is the Type attribute of a
-- SSL!ServiceInput or a SSL!ServiceOutput, this rule creates a
-- SDL!Part that refers to the corresponding SDL type
-- (see the helper getSDLTypeForSchemaType).
rule TypeFromSchema {
      from inp : SSL!DataTypeURI (
            thisModule.getInputOutputHavingURIType(inp) -> size() > 0
      )
      to part_c : SDL!Part (
            ElName <- thisModule.getInputOutputHavingURIType(inp) -> first().name,
            refPart <- thisModule.resolveTemp(thisModule.getTypeGenerator(),
                  thisModule.getSDLTypeForSchemaType(
                        thisModule.getSchemaTypeName(inp.lexicalform)) )
      )
}

-- For every SSL!ServiceInput that is the only input of a
-- SSL!ServiceFunctionality and whose type is not SSL!OntologyClassURI,
```

```
-- this rules construct a SDL!SimpleMessage that describes it (the type
-- of the input is transformed according to the rule TypeFromSchema)
rule SingleInput_NonOntology {
      from e : SSL!ServiceInput (not thisModule.getMultipleInputs()->includes(e)
            and not e.Type.oclIsKindOf(SSL!OntologyClassURI ))
      to inm : SDL!SimpleMessage(
              ElName <- thisModule.getServiceFunctionalityHavingInput(e).name +
                        'Request',
              cmpPart <- e.Type
    )
}

-- The same transformation as SingleInput_NonOntology, but for
-- SSL!ServiceOutput.
rule SingleOutput_NonOntology {
      from e : SSL!ServiceOutput (not thisModule.getMultipleOutputs()->includes(e)
            and not e.Type.oclIsKindOf(SSL!OntologyClassURI ))
      to outm : SDL!SimpleMessage (
        ElName <- thisModule.getServiceFunctionalityHavingOutput(e).name +
                        'Response',
      cmpPart <- e.Type
    )
}

-- For every SSL!ServiceInput that is the only input of a
-- SSL!ServiceFunctionality and whose type is SSL!OntologyClassURI (it
-- is described in the ontology), this rules construct a SDL!SimpleMessage
-- that describes it (the message contains a single SDL!Part that refers
-- to the complex type generated from the input's type - see rule
-- TypeFromOntology_Input: part_c)
rule SingleInput_Ontology {
      from e : SSL!ServiceInput ( not thisModule.getMultipleInputs()->includes(e)
and
            e.Type.oclIsKindOf(SSL!OntologyClassURI ))
      to inm : SDL!SimpleMessage(
            ElName <- thisModule.getServiceFunctionalityHavingInput(e).name +
                        'Request',
            cmpPart <- part
      ),
      part : SDL!Part(
            ElName <- e.name + 'ComplexType',
            refPart <- e.Type
      )
}

-- The same transformation as SingleInput_Ontology, but for
-- SSL!ServiceOutput.
rule SingleOutput_Ontology {
      from e : SSL!ServiceOutput ( not thisModule.getMultipleOutputs()->includes(e)
and
            e.Type.oclIsKindOf(SSL!OntologyClassURI ))
      to outm : SDL!SimpleMessage(
          ElName <- thisModule.getServiceFunctionalityHavingOutput(e).name +
                        'Response',
            cmpPart <- part
      ),
      part : SDL!Part(
            ElName <- e.name + 'ComplexType',
            refPart <- e.Type
```

```
      )
}

-- For every SSL!ServiceFunctionality that has maximum one input and
-- maximum one output, this rule constructs a SDL!Operation that contains
-- the results of the input's and output's transformation (see rules
-- SingleInput_Ontology, SingleInput_NonOntology and their
-- correspondents for output messages)
rule ServiceFunctionality11 {
      from e : SSL!ServiceFunctionality (e.Input.size() <= 1 and
                                         e.Output.size() <= 1 )
      to outOper : SDL!Operation(
            ElName <- e.name,
            refInputMessage <- e.Input,
            refOutputMessage <- e.Output
      )
}

-- For every SSL!ServiceFunctionality that has multiple inputs and
-- maximum one output, this rule constructs a SDL!Operation that
-- contains the results of the output's transformation (see rules
-- SingleOutput_Ontology, SingleOutput_NonOntology) and that constructs
-- a SDL!Message in which it "compacts" the inputs by placing, as parts,
-- the results of the transformation of each input's type (see rules
-- TypeFromOntology_Input: part_c and TypeFromSchema: part_c)
rule ServiceFunctionalityN1 {
      from e : SSL!ServiceFunctionality (e.Input.size() > 1 and
                                         e.Output.size() <= 1)
      to inp : SDL!SimpleMessage(
            ElName <- e.name + 'Request',
            cmpPart <- Sequence{e.Input -> collect(k |
                        thisModule.resolveTemp(k.Type, 'part_c'))}
      ),
      outOper : SDL!Operation(
            ElName <- e.name,
            refInputMessage <- inp,
            refOutputMessage <- e.Output
      )
}


-- For every SSL!ServiceFunctionality that has maximum one input and
-- multiple outputs, this rule constructs a SDL!Operation that contains
-- the results of the input's transformation (see rules
-- SingleInput_Ontology, SingleInput_NonOntology) and that constructs
-- a SDL!Message in which it "compacts" the outputs by placing, as parts,
-- the results of the transformation of each output's type (see rules
-- TypeFromOntology_Output: part_c and TypeFromSchema: part_c)
rule ServiceFunctionality1N {
      from e : SSL!ServiceFunctionality (e.Input.size() <= 1 and e.Output.size() >
1)
      to out : SDL!SimpleMessage(
            ElName <- e.name + 'Response',
            cmpPart <- Sequence{
                  e.Output -> collect(k | thisModule.resolveTemp(k.Type, 'part_c'))
            }
      ),
      outOper : SDL!Operation(
            ElName <- e.name,
            refInputMessage <- e.Input,
```

```
            refOutputMessage <- out
      )
}

-- For every SSL!ServiceFunctionality that has multiple inputs and
-- multiple outputs, this rule constructs a SDL!Operation that contains:
-- - a SDL!Message in which it "compacts" the inputs by placing, as parts,
-- the results of the transformation of each input's type (see rules
-- TypeFromOntology_Input: part_c and TypeFromSchema: part_c)
-- - a SDL!Message in which it "compacts" the outputs by placing, as parts,
-- the results of the transformation of each output's type (see rules
-- TypeFromOntology_Output: part_c and TypeFromSchema: part_c)
rule ServiceFunctionalityNN {
      from e : SSL!ServiceFunctionality (e.Input.size() > 1 and e.Output.size() >
1)
      to inp : SDL!SimpleMessage(
            ElName <- e.name + 'Request',
            cmpPart <- Sequence{
                  e.Input -> collect(k | thisModule.resolveTemp(k.Type, 'part_c'))
            }
      ),
      out : SDL!SimpleMessage(
            ElName <- e.name + 'Response',
            cmpPart <- Sequence{
                  e.Output -> collect(k | thisModule.resolveTemp(k.Type, 'part_c'))
            }
      ),
      outOper : SDL!Operation(
            ElName <- e.name,
            refInputMessage <- inp,
            refOutputMessage <- out
      )
}

-- The main rule of the transformation. It constructs the skeleton of the SDL
model, and uses the
-- results of the other rules in order to fill it.
-- It creates the SDL types definitions, generates the definitions of the
interfaces (containing the
-- generated operations) and groups the generated messages.
rule Definitions {
      from e : SSL!ServiceProfile
      to out : SDL!Definitions (
            ElName <- e.name,
            cmpType <- Sequence{type1, type2, type3, type4, type5, type6,
                  SSL!OntologyClassURI.allInstances()},
            cmpInterface <- interfaces,
            cmpMessage <- Sequence{
            SSL!ServiceInput.allInstances() -> select(e |
                  not thisModule.getMultipleInputs() -> includes(e)) ->
                  collect (e | thisModule.resolveTemp(e, 'inm') ),
            SSL!ServiceOutput.allInstances() -> select(e |
                  not thisModule.getMultipleOutputs() -> includes(e)) ->
                  collect (e | thisModule.resolveTemp(e, 'outm') ),
            SSL!ServiceFunctionality.allInstances() ->
                  select (e | e.Input.size() > 1) ->
                        collect (e | thisModule.resolveTemp(e, 'inp') ),
            SSL!ServiceFunctionality.allInstances() ->
                  select (e | e.Output.size() > 1) ->
                        collect (e | thisModule.resolveTemp(e, 'out') )
```

```
            }
      ),
      interfaces : SDL!Interface (
            ElName <- e.name + 'Service',
            cmpOperation <- e.Functionality ->
                  collect(e | thisModule.resolveTemp(e, 'outOper') )
      ),
      type1 : SDL!SdlInteger(
         ElName <- 'Integer'
      ),
      type2 : SDL!SdlBoolean(
         ElName <- 'Boolean'
      ),
      type3 : SDL!SdlReal(
         ElName <- 'Real'
      ),
      type4 : SDL!SdlString(
         ElName <- 'String'
      ),
      type5 : SDL!SdlDateTime(
         ElName <- 'DateTime'
      ),
      type6 : SDL!SdlUri(
         ElName <- 'Uri'
      )
}
```

8.Glossary

Term	*Acronym*	*Synonymous*	*Obsolete*	*Description*
Atlas Transformation Language	ATL			ATL is a QVT implementation, a way to query, view and transform a metamodel into another metamodel.
Business Modelling Language	BML			BML is a language through which SMEs can describe their business model in order to enhance mechanism of discovery and selection of e-business partner within the DBE.
Digital Business Ecosystem	DBE			A Digital Business Ecosystem is defined as "evolutionary self-organising system aimed at creating a digital software environment for small organisations" that support regional and local development by empowering open, distributed and adaptive technologies and evolutionary business models for the growth of small organisations [DBE Technical Annex]
DBE Studio				The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE). It is represented by a collection of open-source Eclipse plug-ins that allow business analysts and software developers to model, design, implement, publish, deploy, compose, search and execute DBE service
Ontology Definition Metamodel	ODM			A metamodel developed in the DBE project that can be used for defining business and service ontologies [Deliverable D14.1]
Small and Medium-sized Enterprises	SMEs			Small and Medium-sized Enterprises are companies whose headcount or turnover falls below certain limits.

Term	*Acronym*	*Synonymous*	*Obsolete*	*Description*

9.References

ATL User Manual: ATLAS group LINA & INRIA - Nantes, ATL: Atlas Transformation Language User Manual, 2005
DBE Studio: David McKitterick, DBE Studio Integration, 2006
DBE Technical Annex: , DBE Technical Annex, October 2003
DBECoreArch: Pierfranco Ferronato, DBE Core Architecture Ver 01.3, 2004/06/04
Deliverable D14.1: TUC, DBE Knowledge Representation Models,
Deliverable D15.3: Maurizio De Tommasi, ISUFI, BML framework 2nd release, 2005
Deliverable D16.1: Soluta, Service Description Model and Language Definition, April, 2005
MDA: David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, January 10, 2003

- end of document -