

# **ATL Transformation Examples**

## **The SSL to SDL**

### **ATL transformation**

**- version 0.1 -**


*February 2006*

*by*

*Soluta.Net*


*ITALY*

*[www.soluta.net](http://www.soluta.net)*

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p>Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p>SSL to SDL</p>	<p>Date: 20/02/2006</p>

## Table of Contents

1 Abstract .....	3
2 Introduction.....	3
3 The SSL to SDL ATL transformation.....	3
3.1 Transformation overview.....	3
3.2 Metamodels.....	3
3.2.1 SSL Metamodel.....	4
3.2.2 SDL Metamodel.....	6
3.2.3 Ontology Metamodel.....	8
3.3 Rules Specification.....	9
3.4 ATL code.....	10
3.4.1 Helpers.....	10
3.4.2 Rules.....	11
Appendix I - The SSL metamodel in KM3 format.....	14
Appendix II - The SDL metamodel in KM3 format.....	19
Appendix III - The ODM metamodel in KM3 format.....	22
Appendix IV - The SSL to SDL ATL code.....	27

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

## 1 Abstract

This document describes a transformation which enables to create a SDL model from a SSL model, using an auxiliary ODM model that contains the ontology elements referred in the SSL model. SDL is a language for describing concrete service interfaces, from a technical point of view, while SSL can be used to create semantic descriptions of services. SSL models can refer to terms defined in an external ontology, therefore another metamodel, ODM, is used for querying ontologies.

## 2 Introduction

SSL (Semantic Service Language) is used to consistently define service description models for semantically describing services.

ODM (Ontology Definition Metamodel) is used for defining business and service ontologies.

SDL (Service Description Language) is a language used to describe services in a platform independent way. It allows to describe the technical interface of a service.

The SSL and SDL standards are mutually related since a technical interface for a certain service can be obtained from its semantic description.

The three above mentioned standards have been developed for the Digital Business Ecosystem (DBE) project. Supported by the European Commission's 6th FP for research and development in IST, DBE<sup>1</sup> is a 3-year pan-European project involving 120 researchers and specialists from 20 organizations, including some of the important names in the international computing and business context. The goal of the DBE Project is to build an open source environment, an Internet-based ecosystem in which business applications can be developed and used by small to medium enterprises (SMEs). It will enable end users, even those SMEs with low ICT access, to easily interact among each others and use Internet-based business applications as services for expanding their business and market, getting the benefits of intelligence, interaction and adaptation as the software evolves in response to its usage.

## 3 The SSL to SDL ATL transformation


### 3.1 Transformation overview

The SSL to SDL transformation converts a SSL model into a SDL model using an ODM model to include the necessary data from an ontology into the technical specification of the service.

### 3.2 Metamodels

DBE has chosen to adopt a MDA based approach using MOF as the meta-meta-model for defining

<sup>1</sup> More details can be found at the following website: <http://www.digitalecosystem.org/>.

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

all the languages that will be used to represent knowledge in DBE. The SSL to SDL transformation is based on some subsets of the SSL, SDL and ODM metamodels; these metamodels are provided, in km3 format, in Appendix I - The SSL metamodel in KM3 format<sup>2</sup>, Appendix II - The SDL metamodel in KM3 format<sup>3</sup> and Appendix III - The ODM metamodel in KM3 format<sup>4</sup> and they are described in the following sections.

### 3.2.1 SSL Metamodel

The main concepts defined in SSL are shown in a graphical form in Illustration 1: SSL Metamodel - Service Behavior Modeling<sup>5</sup>, below.

A ServiceProfile element contains, among others, multiple SSL ServiceFunctionality elements, each of them having multiple inputs and outputs. Each SSL ServiceInput and ServiceOutput has a Type attribute, that is interesting for the purpose of this transformation; the hierarchy of the SSL TypeURI class is shown in Illustration 2: SSL Metamodel – Type referencing scheme<sup>6</sup>, below. As for the moment enumerated types cannot be defined outside ontologies, the types considered for this transformation are SSL OntologyClassURI and SSL DataTypeURI.

2 The SSL metamodel has been developed at the Technical University of Crete (TUC) and it is available as a part of the DBE project source files and also, in XMI format, on the website <http://www.music.tuc.gr/DBE/soft/metamodels/>.

3 The SDL metamodel has been developed by Soluta.net as part of the Deliverable D16.1: Service description Language of the DBE project (see <http://www.digital-ecosystem.org/>).

4 The ODM metamodel has been developed at the Technical University of Crete (TUC) and it is available as a part of the DBE project source files and also, in XMI format, on the website <http://www.music.tuc.gr/DBE/soft/metamodels/>.

5 Image taken from Deliverable D14: DBE Knowledge Representation Models of the DBE project (see <http://www.digital-ecosystem.org/>).

6 Image based on Deliverable D14: DBE Knowledge Representation Models of the DBE project (see <http://www.digital-ecosystem.org/>).



SOLUTA.NET  
[www.soluta.net](http://www.soluta.net)

## ATL TRANSFORMATION EXAMPLE

Irina Dumitrascu  
[idumitrascu@soluta.net](mailto:idumitrascu@soluta.net)  
Natalia Rebeja  
[nrebeja@soluta.net](mailto:nrebeja@soluta.net)

SSL to SDL

Date: 20/02/2006

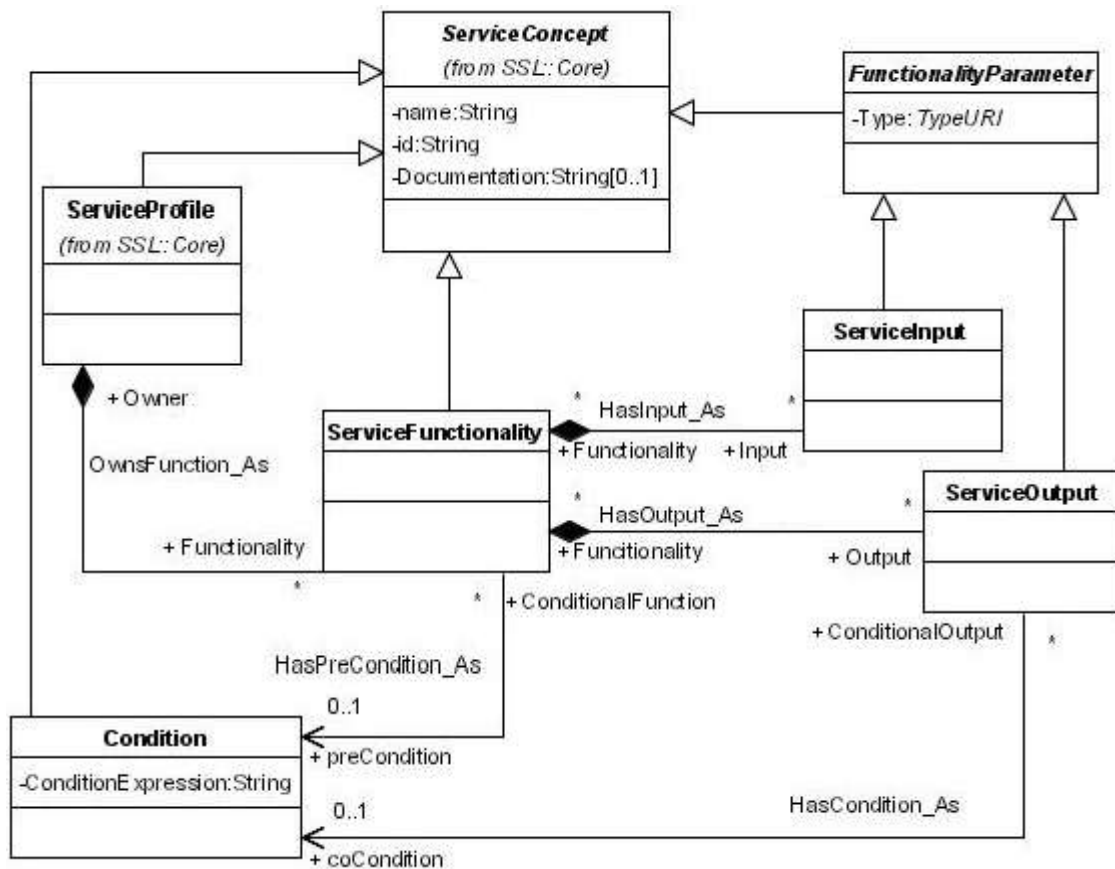



Illustration 1: SSL Metamodel - Service Behavior Modeling

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

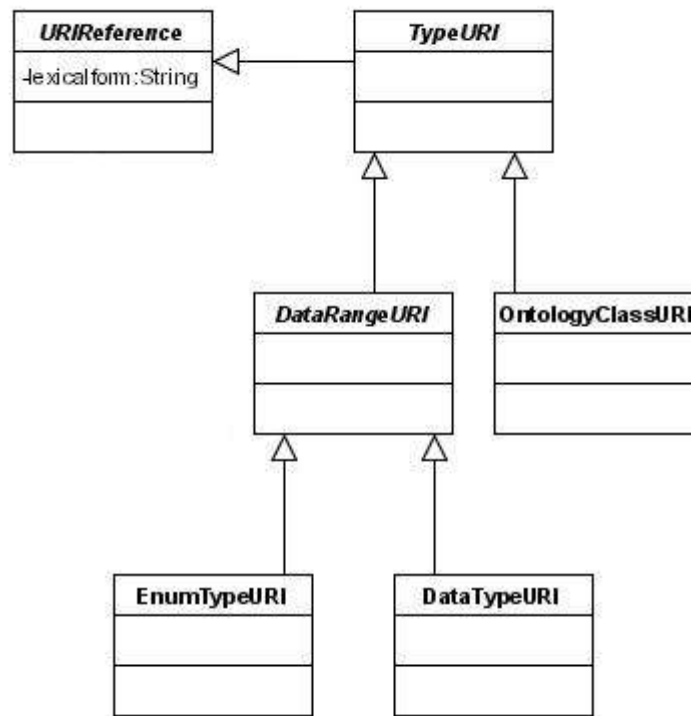


Illustration 2: SSL Metamodel – Type referencing scheme

### 3.2.2 SDL Metamodel

A visual description of some relevant elements from the SDL model is presented in Illustration 3: SDL Metamodel<sup>7</sup>, below.

The root of a SDL model is a Definitions element, that contains the defined SDL types, interfaces, messages and message lists. Each interface has at least one operation, each operation containing its input and output messages and possibly exceptions.

<sup>7</sup> Image based on Deliverable D16.1: Service description Language of the DBE project (see <http://www.digital-ecosystem.org/>).



SOLUTA.NET  
www.soluta.net

## ATL TRANSFORMATION EXAMPLE

Irina Dumitrascu  
[idumitrascu@soluta.net](mailto:idumitrascu@soluta.net)  
Natalia Rebeja  
[nrebeja@soluta.net](mailto:nrebeja@soluta.net)

SSL to SDL

Date: 20/02/2006

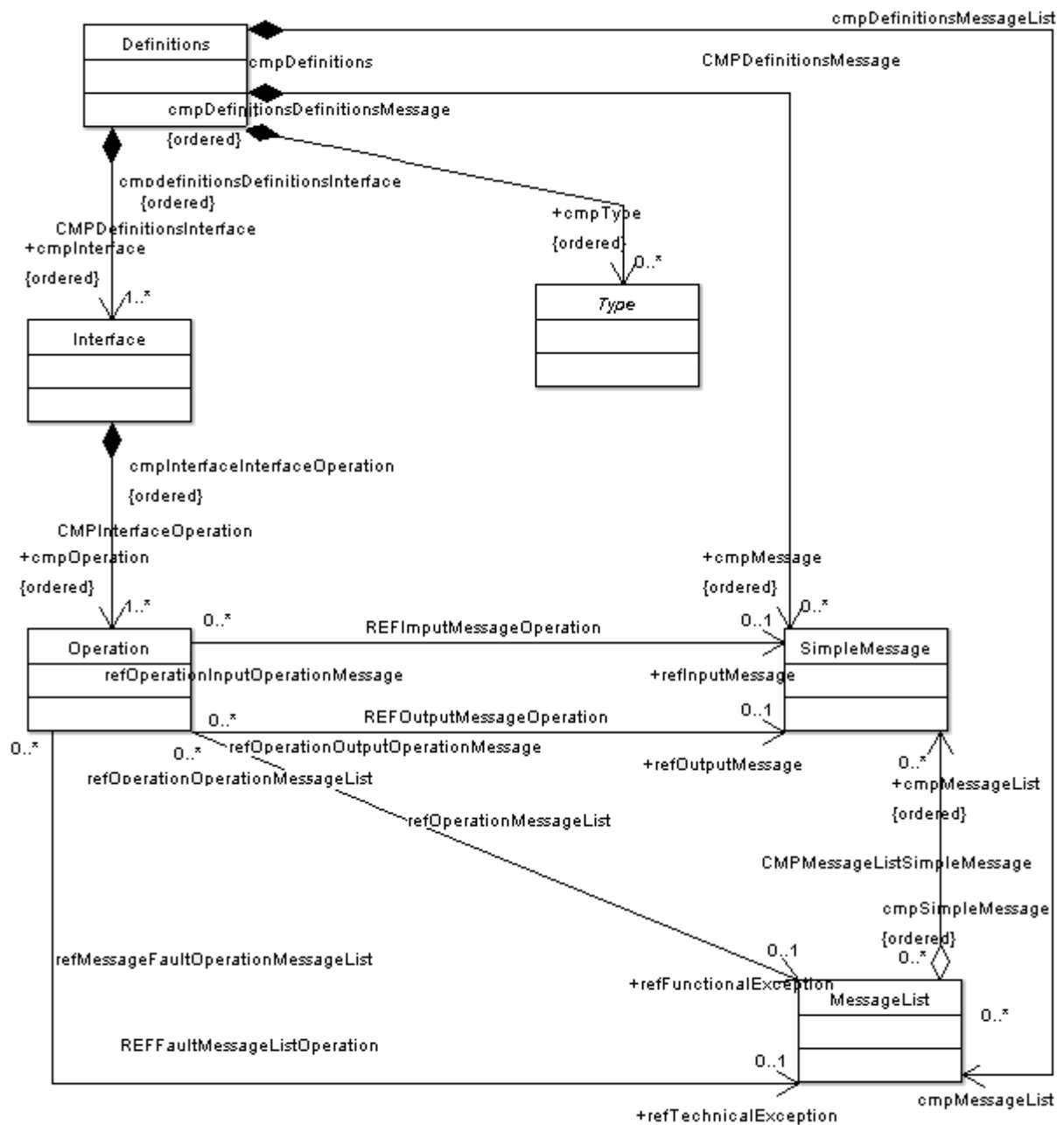



Illustration 3: SDL Metamodel

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

### 3.2.3 Ontology Metamodel

Illustration 4: ODM Metamodel - Properties in ODM<sup>8</sup>, below shows the ontology metamodel elements of interest for the presented transformation.

A ODM Property has a domain and a range that, for the ODM DatatypeProperty subclass is an ODM DataRange element.

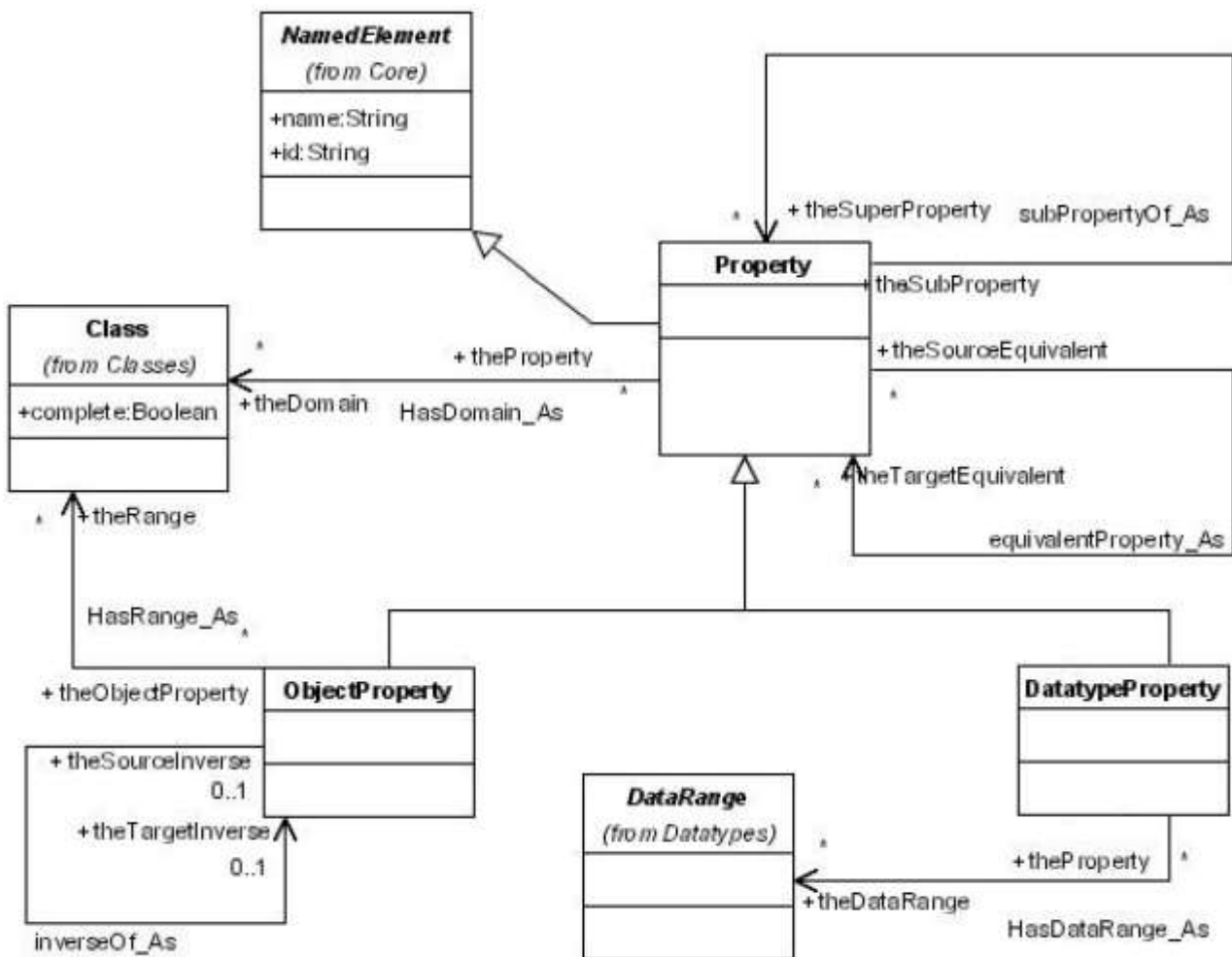



Illustration 4: ODM Metamodel - Properties in ODM

<sup>8</sup> Image taken from Deliverable D14: DBE Knowledge Representation Models of the DBE project (see <http://www.digital-ecosystem.org/>).




 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

### 3.3 Rules Specification

The rules to transform a SSL model to a SDL model are the following:

- ◆ 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)
  - ◆ the list of SDL Interface-s (a SDL Interface consists of all the generated SDL Operations, each of them having references to its input, output and exception conditions)
  - ◆ the list of generated SDL SimpleMessage-s.
- ◆ From each SSL ServiceFunctionality a SDL Operation is created.  
 As a SSL ServiceFunctionality can have multiple Input(s)/Output(s) and, by contrast, in SDL each Operation must have exactly 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 a SDL SimpleMessage is created, having as SDL Part-s the elements obtained from the transformation of each Input's Type attribute
  - ◆ if a given SSL ServiceFunctionality has more than one Output (it has multiple outputs) then a SDL SimpleMessage is created, having as SDL Part-s the elements obtained from the transformation of each Output's Type attribute.
- ◆ For each SSL ServiceInput that is the only input of a SSL ServiceFunctionality, a SDL SimpleMessage is constructed.
- ◆ From each SSL ServiceOutput that is the only output of a SSL ServiceFunctionality, a SDL SimpleMessage is constructed.
- ◆ From each SSL DataTypeURI (that is pointing to a XML Schema type) that is the Type attribute of a SSL ServiceInput or a SSL ServiceOutput, a SDL Part is created, referring to the SDL type that corresponds to the indicated XML Schema type.  
 From each SSL OntologyClassURI (containing the id of a ODM Class defined in the ontology) that is the Type attribute of a SSL ServiceInput or a SSL ServiceOutput the following elements are created:
  - ◆ a SDL ComplexType containing a SDL Part for each ODM DatatypeProperty element that contains that OntologyClassURI in its theDomain attribute. 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 a SSL ServiceInput included in a SSL ServiceFunctionality with multiple inputs or if the SSL OntologyClassURI is the Type of a SSL ServiceOutput included in a SSL ServiceFunctionality with multiple outputs, then a SDL Part is created, referring to the SDL ComplexType that it generated.

As in the current specification of the SSL metamodel the only usable TypeURI subclasses are SSL DataTypeURI and SSL OntologyClassURI, only these are considered in this

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006


transformation.

### 3.4 ATL code

The ATL code for the UML to MOF transformation is provided in Appendix IV - The SSL to SDL ATL code and it is presented in the following sections.

#### 3.4.1 Helpers

- ◆ **getOntologyIDName (e : String)** returns **String**  
A SSL!OntologyClassURI element has a lexicalForm attribute with the following structure: <ontologyID>#<ontologyClassID>.  
This helper takes this value and returns the <ontologyClassID> part.
- ◆ **getSchemaTypeName (e : String)** returns **String**  
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.
- ◆ **getDataTypeProperties (e : String)** returns **Sequence of ODM!DatatypeProperty**  
Taking a String value, this helper returns a sequence of ODM!DatatypeProperty elements that have in their theDomain property a reference to a class with the id attribute equal to that String value.
- ◆ **getMultipleInputs ()** returns **Sequence of SSL!ServiceInput**  
This helper returns a sequence containing all the SSL!ServiceInput elements that belong to a 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.
- ◆ **getTypeGenerator ()** returns **SSL!ServiceProfile**  
Returns the root element of the input SSL model, as it is the one that generates the SDL types.
- ◆ **getSDLTypeForODM (p : ODM!DataRange)** returns **String**  
Returns the name of the target pattern (in rule Definitions) that generates the SDL type that


 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

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 name is the one returned by the helper `getSDLTypeForODM_Default`.

- ◆ **getSDLTypeForODM\_Default ()** returns **String**  
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).
- ◆ **getSDLTypeForSchemaType (e : String)** returns **String**  
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 returned target pattern name is the one corresponding to SDL!String.
- ◆ **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.
- ◆ **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.


### 3.4.2 Rules

- ◆ **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 a SSL!ServiceInput, this rule creates:
  - 1) a SDL!ComplexType that has a SDL!Part for each ODM!DatatypeProperty in which this class is referred (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)
- ◆ **TypeFromOntology\_Output**  
The same transformation as TypeFromOntology\_Input, but for types that belong to a

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

ServiceOutput.

- ◆ **TypeFromSchema**  
 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).
- ◆ **SingleInput\_NonOntology**  
 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).
- ◆ **SingleOutput\_NonOntology**  
 The same transformation as SingleInput\_NonOntology, but for SSL!ServiceOutput.
- ◆ **SingleInput\_Ontology**  
 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).
- ◆ **SingleOutput\_Ontology**  
 The same transformation as SingleInput\_Ontology, but for SSL!ServiceOutput.
- ◆ **ServiceFunctionality11**  
 For every SSL!ServiceFunctionality that has a single input and a single 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).
- ◆ **ServiceFunctionalityN1**  
 For every SSL!ServiceFunctionality that has multiple inputs and a single 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).
- ◆ **ServiceFunctionality1N**  
 For every SSL!ServiceFunctionality that has a single 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

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006


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 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).

- ◆ **Definitions**

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

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p>Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p>SSL to SDL</p>	<p>Date: 20/02/2006</p>

### *Appendix I - The SSL metamodel in KM3 format*

```

package PrimitiveTypes {

    datatype Integer;

    datatype Long;

    datatype Float;

    datatype Double;

    datatype Boolean;

    datatype String;

}

package SSL {

    package Instantiation {

        class PropertyInstance extends SSLInstance {
            reference Value : Value;
            attribute InstanceOf : OntologyPropertyURI;
            attribute name : String;
        }

        class ClassInstance extends Value, SSLInstance {
            reference Property container : PropertyInstance;
            attribute InstanceOf : OntologyClassURI;
            attribute name : String;
        }


        class ContactInstance extends SSLInstance {
            reference contType : ContactInformation;
            reference Value : Value;
            attribute InAssociation[0-1] : String;
        }

        class ParamInstance extends SSLInstance {
            reference Value : Value;
            reference paramType : ServiceParameter;
            attribute InAssociation : String;
            attribute paramTypeID : String;
        }

    }

}

```

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p style="text-align: center;"><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p style="text-align: right;">Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p style="text-align: center;">SSL to SDL</p>	<p style="text-align: right;">Date: 20/02/2006</p>

```

abstract class Value {
}

class AttrInstance extends SSLInstance {
    reference Value : Value;
    reference AttrType : ServiceAttribute;
    attribute AttrTypeID : String;
}

class ProfileInstance extends SSLInstance {
    reference contInstance[*] container : ContactInstance;
    reference paramInstance[*] container : ParamInstance;
    reference profType : ServiceProfile;
    reference AttrInstance[*] container : AttrInstance;
    attribute ProfTypeID : String;
}

abstract class SSLInstance {
    attribute InstanceID : String;
}

}

package Types {

    class OntologyPropertyURI extends URIReference {
    }

    class Literal extends Value {
        attribute LexicalForm : String;
    }


    class OntologyClassURI extends TypeURI {
    }

    class OntologyThingURI extends URIReference {
    }

    abstract class URIReference {
        attribute lexicalform : String;
    }

    abstract class TypeURI extends URIReference {
    }
}

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class EnumTypeURI extends DataRangeURI {
}

class DataTypeURI extends DataRangeURI {
}

abstract class DataRangeURI extends TypeURI {
}

class Multiplicity {
    attribute LowerBound : Integer;
    attribute UpperBound[0-1] : Integer;
}

}

package Associations {

    class Target extends AssociationEnd {
    }

    class Source extends AssociationEnd {
    }

    abstract class AssociationEnd extends ServiceConcept {
        reference EndType : ServiceConcept;
        attribute Multiplicity : Multiplicity;
    }

    class Association extends ServiceConcept {
        reference TargetEnd container : Target;
        reference SourceEnd container : Source;
    }

}


package ServiceBehavior {

    class ServiceOutput extends FunctionalityParameter {
        reference coCondition[0-1] : Condition;
    }

    class ServiceInput extends FunctionalityParameter {
    }
}

```



 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class ServiceFunctionality extends ServiceConcept {
    reference Output[*] container : ServiceOutput;
    reference preCondition[0-1] : Condition;
    reference Input[*] container : ServiceInput;
}

abstract class FunctionalityParameter extends ServiceConcept {
    attribute Type : TypeURI;
}

class Condition extends ServiceConcept {
    attribute ConditionExpression : String;
}

}

package Core {

    class XYCoordinates {
        attribute X : Integer;
        attribute Y : Integer;
    }


    class ContactInformation extends ServiceConcept {
        attribute Type : TypeURI;
    }

    class ServiceProfile extends NameSpace {
        reference Functionality[*] container : ServiceFunctionality;
        reference Domain[1-*] : DBEServiceDomain;
        reference Attribute[*] container : ServiceAttribute;
        reference Category[*] : IndustryDomain;
        reference Identifier container : ServiceIdentifier;
    }

    class ServiceParameter extends ServiceConcept {
        attribute Type : TypeURI;
    }

    class ServiceIdentifier extends ServiceConcept {
        attribute Type : DataTypeURI;
    }
}

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class DBEServiceDomain extends ServiceConcept {
    attribute DomainRef : OntologyClassURI;
}

class IndustryDomain extends ServiceConcept {
    attribute TaxonomyName : String;
    attribute TaxonomyRef[0-1] : URIReference;
}


class ServiceAttribute extends ServiceConcept {
    attribute AttributeType : TypeURI;
    attribute Multiplicity : Multiplicity;
}

abstract class NameSpace extends ServiceConcept {
    reference OwnedElement[*] container : ServiceConcept;
}

class SemanticPackage extends NameSpace {
    reference profInstance[*] container : ProfileInstance;
}

abstract class ServiceConcept {
    reference Position[0-1] container : XYCoordinates;
    attribute name : String;
    attribute id : String;
    attribute Documentation[0-1] : String;
}
}
}

```

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p>Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p>SSL to SDL</p>	<p>Date: 20/02/2006</p>

## Appendix II - The SDL metamodel in KM3 format

```

package sdl {

    abstract class Element {
        attribute ElName : String;
    }

    abstract class DocumentedElement extends Element {
    }

    abstract class SemanticElement extends DocumentedElement {
        attribute ontologyReference[0-1] : String;
    }

    class Definitions extends SemanticElement {
        reference cmpMessageList[*] container : MessageList;
        reference cmpMessage[*] ordered container : SimpleMessage;
        reference cmpInterface[1-*] ordered container : Interface;
        reference cmpType[*] ordered container : Type;
    }

    class Interface extends SemanticElement {
        reference cmpOperation[1-*] ordered container : Operation;
    }


    class SimpleMessage extends Message {
        reference cmpPart[1-*] ordered container : Part;
    }

    class Operation extends SemanticElement {
        reference refOutputMessage : SimpleMessage;
        reference refInputMessage : SimpleMessage;
        reference refFunctionalException[0-1] : MessageList;
        reference refTechnicalException[0-1] : MessageList;
    }

    class Part extends SemanticElement {
        reference refPart : Type;
        attribute optional[0-1] : Boolean;
        attribute isArray[0-1] : Boolean;
    }

    class Type extends SemanticElement {
    }
}

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class ComplexType extends Type {
    reference cmpPart[1-*] ordered container : Part;
}

abstract class SimpleType extends Type {
}

class SdlString extends SimpleType {
}

class SdlReal extends SimpleType {
}

class SdlInteger extends SimpleType {
}

class SdlBoolean extends SimpleType {
}

class SdlDateTime extends SimpleType {
}

class SdlUri extends SimpleType {
}

class MessageList extends Message {
    reference cmpMessageList[*] ordered : SimpleMessage;
}

abstract class Message extends SemanticElement {
}
}

package PrimitiveTypes {

    datatype Integer;


    datatype Long;

    datatype Float;


    datatype Double;

    datatype Boolean;
}

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```
datatype String;  
}
```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

### Appendix III - The ODM metamodel in KM3 format

```

package ODM {

    package Classes {

        abstract class ValueRange {
        }

        abstract class Value {
        }

        class DeprecatedClass extends Class {
        }

        class Class extends ValueRange, NamedElement {
            reference theTargetDisjointClass[*] : Class;
            reference theAnnotationProperty[*] : AnnotationProperty;
            reference theComplemented[0-1] : Class;
            reference theInstance[*] : ClassThing;
            reference theMember[*] : Class;
            reference theTargetEquivalent[*] : Class;
            reference theSuperClass[*] : Class;
            reference theIntersected[*] : Class;
            attribute complete : Boolean;
        }

        class Restriction extends Class {
            reference theProperty : Property;
            reference theCardinality[0-1] : NonNegativeInteger;
            reference theValue[0-1] : Value;
            reference theMaxCardinality[0-1] : NonNegativeInteger;
            reference theMinCardinality[0-1] : NonNegativeInteger;
            reference theAValueRange[0-1] : ValueRange;
            reference theSValuesRange[0-1] : ValueRange;
        }

    }


    package Datatypes {

        class Enumeration extends DataRange, NamedElement {
            reference Enumerator[1-*] ordered : Literal;
        }

    }

}

```

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p align="center"><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p align="center">Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p align="center">SSL to SDL</p>	<p align="center">Date: 20/02/2006</p>

```

class PrimitiveType extends Datatype {
}

class TypedLiteral extends Literal {
    reference theType : URIreference;
}

class PlainLiteral extends Literal {
    attribute Language : String;
}

class URIreference extends Literal, AnnotationObject {
}

class NonNegativeInteger extends TypedLiteral {
}

class Literal extends Value, Element, AnnotationObject {
    attribute lexicalForm : String;
}

abstract class DataRange extends ValueRange, Element {
}

class DeprecatedDatatype extends Datatype {
}

class Datatype extends DataRange, NamedElement {
    reference TypeDefinitionURI : URIreference;
}


}

package Individuals {

    class DataTypePropertyThing extends Thing {
        reference Type : DatatypeProperty;
        reference TheDTPRange : Literal;
        reference TheDTPDomain : ClassThing;
    }

    class ObjectPropertyThing extends Thing {
        reference TheOPDomain : ClassThing;
        reference TheOPRange : ClassThing;
        reference Type : ObjectProperty;
    }
}

```

 <b>SOLUTA.NET</b> <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class ClassThing extends Value, Thing, AnnotationObject {
    reference theAnnotationProperty[*] : AnnotationProperty;
    reference Type[1-*] : Class;
    attribute name : String;
}

class AllDifferent extends Element {
    reference Member[*] : Thing;
}

abstract class Thing extends Element {
    reference theTargetSameThing[*] : Thing;
    reference theTargetDifferentThing[*] : Thing;
    attribute id : String;
}

}

package Ontology {

    class OntologyProperty extends NamedElement {
        reference Domain[1-*] : Ontology;
        reference Range[1-*] : Ontology;
    }

    class Ontology extends NameSpace {
        reference theAnnotationProperty[*] : AnnotationProperty;
    }

}

package Properties {

    class deprecatedDatatypeProperty extends DatatypeProperty {
    }


    class FunctionalDatatypeProperty extends DatatypeProperty {
    }

    class DeprecatadObejctProperty extends ObjectProperty {
    }

    class FunctionalObjectProperty extends ObjectProperty {
    }
}

```



 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

class InverseFunctionalProperty extends ObjectProperty {
}

class TransitiveProperty extends ObjectProperty {
}

class SymmetricProperty extends ObjectProperty {
}

class DatatypeProperty extends Property {
    reference theDataRange : DataRange;
}

class ObjectProperty extends Property {
    reference theRange[*] : Class;
    reference theTargetInverse[0-1] : ObjectProperty;
}

abstract class Property extends NamedElement {
    reference theDomain[*] : Class;
    reference theTargetEquivalent[*] : Property;
    reference theAnnotationProperty[*] : AnnotationProperty;
    reference theSuperProperty[*] : Property;
}

}

package Core {


    abstract class NamedElement extends Element {
        attribute name : String;
        attribute id : String;
    }

    abstract class NameSpace extends NamedElement {
        reference OwnedElement[*] container : Element;
    }


    abstract class AnnotationObject {
    }

    class AnnotationProperty extends NamedElement {
        reference theAnnotationObject : AnnotationObject;
    }
}

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">iridumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```
        abstract class Element {
            }
    }
package PrimitiveTypes {
    datatype Integer;
    datatype Long;
    datatype Float;
    datatype Double;
    datatype Boolean;
    datatype String;
}
```

 <p>SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a></p>	<p style="text-align: center;"><b>ATL TRANSFORMATION EXAMPLE</b></p>	<p style="text-align: center;">Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a></p>
	<p style="text-align: center;">SSL to SDL</p>	<p style="text-align: center;">Date: 20/02/2006</p>

## Appendix IV - The SSL to SDL ATL code

```

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 of
-- ODM!DatatypeProperty elements that have in their theDomain property
-- a reference to a class with the id attribute equal to that String
-- value.
helper def: getDataTypeProperties(input : String) :
    Sequence(ODM!DatatypeProperty) =
    ODM!DatatypeProperty.allInstances() -> select(e | e.theDomain ->
        collect(d | d.id) -> includes(input))
;

-- 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()
;

-- This helper returns the name of the SSL!ServiceInput that has as

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

-- 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 corresponding to SDL!String.
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.
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
-- returned 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

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006


```

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
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)
;


```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

-- 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 referred (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.getDataTypeProperties(
        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))(

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006


```

        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.getDataTypeProperties(
                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.

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

-- 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 <- e.name,
        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 <- e.name,
        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

```



 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006


```

-- 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 <- e.name,
        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 <- e.name,
        cmpPart <- part
    ),
    part : SDL!Part(
        ElName <- e.name + 'ComplexType',
        refPart <- e.Type
    )
}

-- For every SSL!ServiceFunctionality that has a single input and a
-- single 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
    )
}

```


 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:iridumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

-- For every SSL!ServiceFunctionality that has multiple inputs and a
-- single 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 a single 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
  )
}

```


 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

-- 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{

```

 SOLUTA.NET <a href="http://www.soluta.net">www.soluta.net</a>	<b>ATL TRANSFORMATION EXAMPLE</b>	Irina Dumitrascu <a href="mailto:idumitrascu@soluta.net">idumitrascu@soluta.net</a> Natalia Rebeja <a href="mailto:nrebeja@soluta.net">nrebeja@soluta.net</a>
	SSL to SDL	Date: 20/02/2006

```

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'
)
}

```

- end of document -