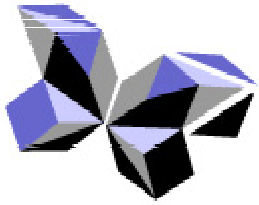

Digital Business Ecosystem

Contract n° 507953

WP16: Service Description Language

D16.2: SDL specification for the DBE Platform

Part 1 - SDL Full Definition

Information Society Technologies

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

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D16.2 – part 1

Due date: July 2006

Delivery Date: August 2006

Short Description:

This document illustrates the Service Description Language (SDL), a language for describing services in an abstract/platform independent way.

Author: Soluta.net

Partners contributed: Soluta.net

Made available to: DBE Consortium

Versioning		
Version	**Date**	**Author, Organisation**
00.01	22/03/2006	Giulio Montanari, Umberto Pernice – Soluta.net
00.02	18/06/2006	Giulio Montanari, Umberto Pernice – Soluta.net
01.00	30/08/2006	Giulio Montanari, Umberto Pernice – Soluta.net

Quality check

1st Internal Reviewer: Miguel Vidal – SUN Microsystems

2nd Internal Reviewer: Angelo Corallo – ISUFI

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License. To view a copy of this license, visit : http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

creative commons

COMMONS DEED

Attribution-NonCommercial-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Table of Contents

Table of Figures

Index of Tables

1. Executive Summary

This document provides a full definition of the Service Description Language (SDL) updating the previous SDL version - as released in D.16.1 - into the current SDL version 3.0 released in D.16.2 “SDL Specification for the DBE Platform”.
The SDL (Service Description Language) is a language for describing services in a platform independent way, where for platform in this context it is specifically meant middleware. In the DBE project, a service is described from two different viewpoints: the business and the technical viewpoint. The BML (Business Modelling Language) and the SBVR (Semantics of Business Vocabulary and Business Rules) describe the service from a business point of view, while the SDL describes the technical interface of the service. By technical interface it is meant the low-level/programmatic interface that a service presents to all the consumer applications. According to the MDA[1] approach, adopted by the DBE project, the SDL is defined using an XMI[2] 1.2 MOF[3] 1.4 metamodel (i.e. the XMI 1.2 is used for streaming out the SDL expressed as MOF 1.4 statements). The use of a MOF metamodel allows the use of an MDA tool for the automatic generation of the SDL Editor and the use of tools for the SDL to Java transformation or transformations among several other languages.
The SDL is similar to the WSDL[4] but in DBE it is needed a description language that enables the automatic composition of services. The SDL uses ontology to define the semantic of the services and data used. In addition the use of a “custom” language enables the development of new concepts to fulfil the DBE requirements.

1 Model Driven Architecture [MDA].
2 XML Metadata Interchange format [XMI].
3 Meta-Object Facility [MOF].
4 Web Services Description Language [WSDL].

2.Foreword

Alike the previous deliverable, D.16.2 is divided into two parts: part one, “SDL full definition” and part two, “Service Manifest software model full definition”. Concerning the first part, compared to D.16.1, this deliverable does not document the process followed to define the SDL, but focuses on the final achieved results. it explains the motivations for the adopted choices. ; as such it addresses the recommendations provided by the EU Commission Reviewers [DBE 2nd Review Report][5]

5 The Report contains the considerations made by the EU official reviewers to the DBE projects and set of deliverables during the second review meeting held in Tampere between the 17th and 19th of January 2006.

3.SDL Features

The use of SDL in DBE states a list of features characterizing the language itself, divided into two main categories: functional and extra functional features.

3.1 Functional Features

Feature name	*Description*
Definition of Service Interface	The main SDL responsibility is to define the service interface, i.e. “sequences of messages that a service sends and/or receives”. Messages may be annotated with ontology to define their semantics.
Definition of Services Semantics	The SDL defines semantics both of services and messages sent and received by services.
Supporting Service Composition	The SDL supports the service composition.
Enabling of Data Mapping	This feature is a sub-feature of Enabling services composition; data mapping allows communication between services that do not have the same exact messages structure but use messages containing the same information. The mapping may be driven by ontology.
No Inheritance	A service may extend a base service only by a “copy and modify” mechanism.
Sharing of Data Type	A service must easily use data type defined by other services in order to allow the reuse of data and standard generation.

3.2 Extra Functional Features

Feature name	*Description*
XMI 1.2 MOF 1.4 based	SDL has to be an XMI1.2 MOF1.4 document.
Enabling generation of Platform/Technology independent interfaces	The SDL is not platform/technology dependent.
Enabling generation of Technology/Protocol dependent interfaces	Using MDA tools and principles, the SDL enables the automatic generation of Technology/Protocol dependent interfaces, such as SOAP[6], WSDL or Java.

6 Simple Object Access Protocol. See [SOAP].

4.Introduction to SDL

Defining the SDL means to create a language for describing services in an abstract/platform independent way, where for platform, in this context, it is specifically meant middleware. Although interface description languages like CORBA[7], IDL[8] and WSDL[9] may be considered platform independent, they are strictly middleware dependent. It is interesting to notice that a PIM[10] is a PSM[11] from an upper abstraction point of view, for example IDL may be a PIM because is independent from the CORBA implementation/vendor but a PSM for DBE purposes because IDL is independent from platform but is dependent from CORBA specification. Same consideration can be extended to the WSDL.
From a logical point of view, the SDL is similar to WSDL v2.0 but it offers some interesting evolution of compared to WSDL, as it is middleware independent (there is no binding specification) and semantically rich.
The Figure 1 - SDL dependency with other standards, below shows an overview of the SDL and its dependencies with other concepts of the DBE.

7 Common Object Request Broker Architecture [CORBA]
8 Interface Definition Language [IDL]
9 WebServices Description Language[WSDL]
10 Platform Independent Model (PIM) [MDA].
11 Platform Specific Model (PSM) [MDA].

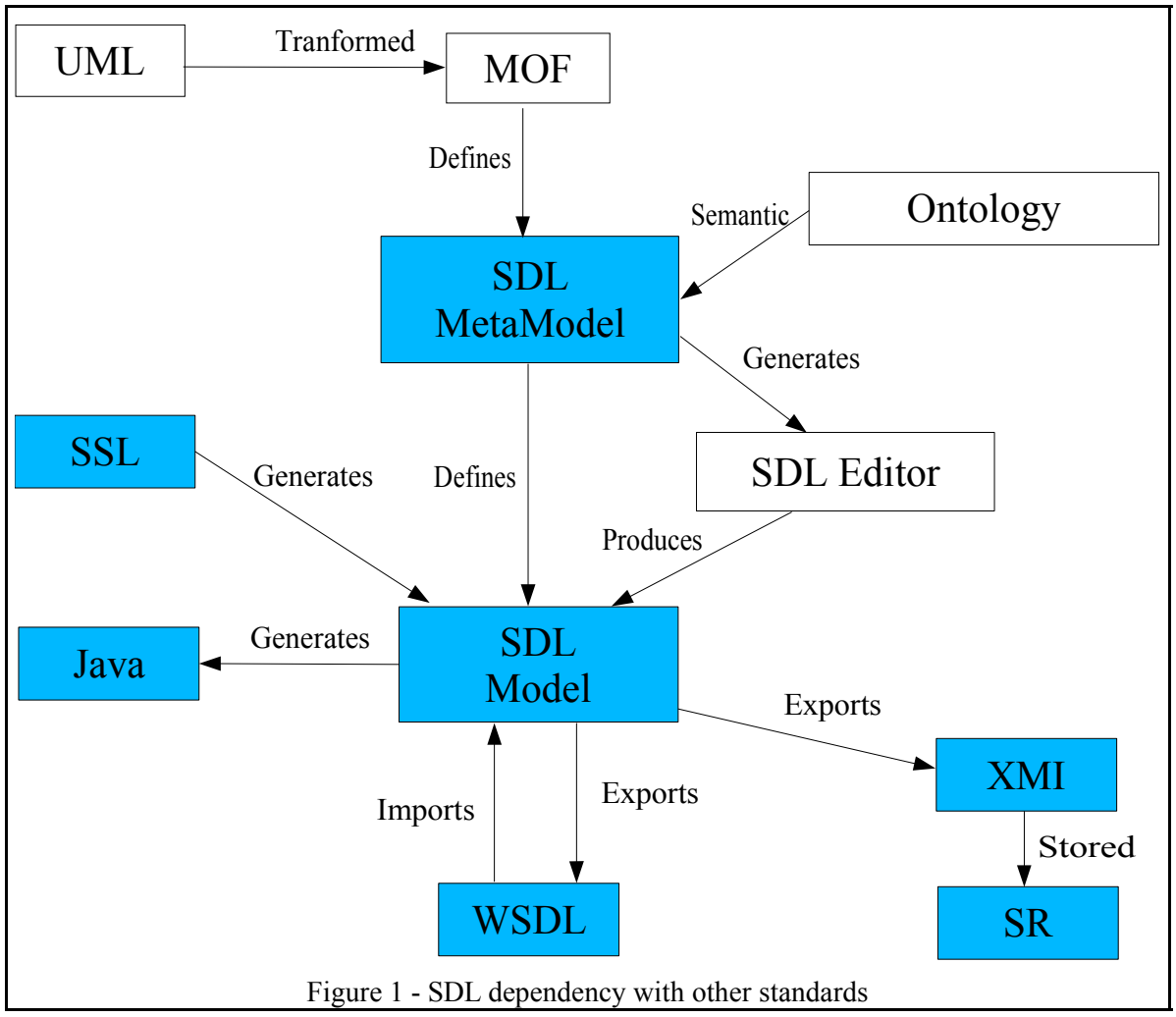

Figure 1 - SDL dependency with other standards

A detailed explanation of the Figure 1 - SDL dependency with other standards is provided in the next paragraph “Methodology applied“ together with a description of the process followed to define the SDL.

4.1 Methodology applied

Since the SDL has to be an XML[12] specification, its syntax can be defined in very simple way using the XML Schema[13], but the adoption of the MDA approach by the DBE project produces, as a consequence, that the SDL Metamodel is defined using XMI-MOF which allows to generate the SDLEditor and to store the SDL Metamodel in XMI-MOF format in the Semantic Registry. In addition, since the DBE project adopts an Open Source or Free Software approach, the way to obtain the SDL Metamodel and to generate the SDL Editor has required a keen usage of several different tools because it was not possible

12 eXtensible Markup Language. See [XML].
13 See [XMLS].

to find a single FLOSS tool that allow to define the Metamodel, export it in MOF XMI format, generate the SDL Editor, etc. etc.
In this chapter the reader is supposed to be proficient with XMI, UML and MOF basic concepts; this because the document does not aim at introducing the XMI, rather it discusses an XMI characteristic that might be not well known to all the readers.

4.1.1 MOF XMI and UML XMI

Despite XMI means XML Metadata Interchange (it easier to find “Metamodel” or “Model” instead of the correct OMG[14] adopted term: “Metadata”) it is not so easy to interchange models or metamodels between different UML or MDA tools. To better comprehend this consideration we have just to think that XMI is a standard for representing Metadata according to a Metamodel (i.e. a language). For example, adopting the OMG's naming conventions, XMI may represent Metamodels using a MOF notation and may represents a Model using an UML notation. Without introducing other complexity investigating the difference, “if it really exists”, between model and meta-model, we have two different XMI, MOF XMI and UML XMI; the first indicated that XMI contains MOF specification of a metamodel and the latter an UML specification of a metamodel. As such it is not possible to interchange metadata between tools that use different languages even though they are streamed in XMI, that act more or less as a simple data carrier. The receiving application need to be able to understand the semantic of MOF if we want to import XMI MOF or UML if we want to import XMI UML: XMI is not a silver bullet for metamodel interchange despite its name.
In addition there are a lot of versions for XMI (such as XMI 1.0, 1.1, 1.2 and 2.0) as well as for UML and MOF. The previous discussion refers to the OMG XMI standard, but there are also dialects, such as, for instance, the ECORE XMI used by Eclipse[15] EMF, which contribute to generate a plethora of standards. The drawback of such a multitude of standards and versions is a correspondence of the same multitude of tools to convert from an XMI standard/version to another.

4.1.2 Soluta.net contribution to UML2MOF

UML2MOF is a “*command-line tool that lets you to convert models created using UML 1.3 into MOF 1.4 metamodels*”[16]. To make it easier to convert files using UML2MOF tool, Soluta.net published an on-line version of it (a web application built on top of UML2MOF) which can be used for converting UML models to MOF metamodels. The application is available at http://www.soluta.net/xmi2mof.php
Soluta.net also helped to test and debug the UML2MOF tool and received an appreciation for such a task from Martin Matula in the Netbeans UML2MOF official website[17].
We are currently developing an Eclipse plug-in for the UML2MOF tool to allow the Eclipse user to use it easier.

14 Object Management Group. See http://www.omg.org/
15 See http://www.eclipse.org/
16 See [U2M].
17 Ibidem 16.

4.1.3 Define the UML Metamodel

The first step to the SDL definition was the creation of the SDL Metamodel, using the tool PoseidonUML CE[18], as shown in Figure 3 - SDL v3.0 Class Diagram and Figure 4 - SDL v 3.0 Inheritance Diagram. Poseidon allows to export the metamodel in XMI UML format.

4.1.4 Define the MOF Metamodel

While the first version of the SDL was defined by an XML Schema generated from the UML Metamodel, the current version of the SDL is defined using an XMI 1.2 MOF 1.4 Metamodel that is still generated from the UML Metamodel.
Since Poseidon can export the metamodel in XMI UML format, the XMI MOF metamodel may be obtained transforming the UML model into the MOF model using UML2MOF. The UML2MOF transforms a XMI 1.2 UML 1.3 model in a XMI 1.2 MOF 1.4 model.
The XMI-MOF SDL Metamodel is the normative definition of the SDL and it is stored in the SR.

4.1.5 Generate the SDL Editor

The SDL Metamodel is used to generate the SDL Editor. As previously mentioned, it is not very easy to interchange models between different UML or MDA tools. The generation path of SDL Metamodel may be a clear example of this difficulty but also of the possibility to reuse and transform models across different standards, formats and standard versions (for instance from XMI 1.2 to XMI 2.0). The SDL Editor was generated using a customization of the Eclipse EMF. The main trouble resolved was related to the metamodel format used by EMF: XMI 2.0 ECORE 2.0 that is not an OMG standard. Also the EMF standard representation for the models, created with the generated editor, is XMI 2.0 ECORE 2.0 but in DBE the format for the models is XMI 1.2 MOF1.4, the serializer and the parser was customized to XMI 1.2 MOF 1.4 standards. The Figure 2 - SDL Metamodel Transformations shows the SDL Metamodel generation path.

4.1.6 SDL Metamodel Transformation

The goal of this process is to obtain two representations of the SDL Metamodel: one representation in XMI 2.0 Ecore 2.0 used by Eclipse EMF to generate the SDL Editor, as stated in chapter 4.1.5 - Generate the SDL Editor and the other representation in XMI 1.2 MOF 1.4 needed by the Semantic Registry to store the SDL.
The choice to start with an XMI 1.0 UML 1.2 Metamodel was somehow forced by the Zargo2Ecore plug-in, as in that circumstance that was the only available plug-in to convert an XMI-UML model into a XMI-ECORE model (and Zargo2Ecore transforms only XMI 1.0 UML 1.2 metamodels).

18 See http://www.gentleware.com/

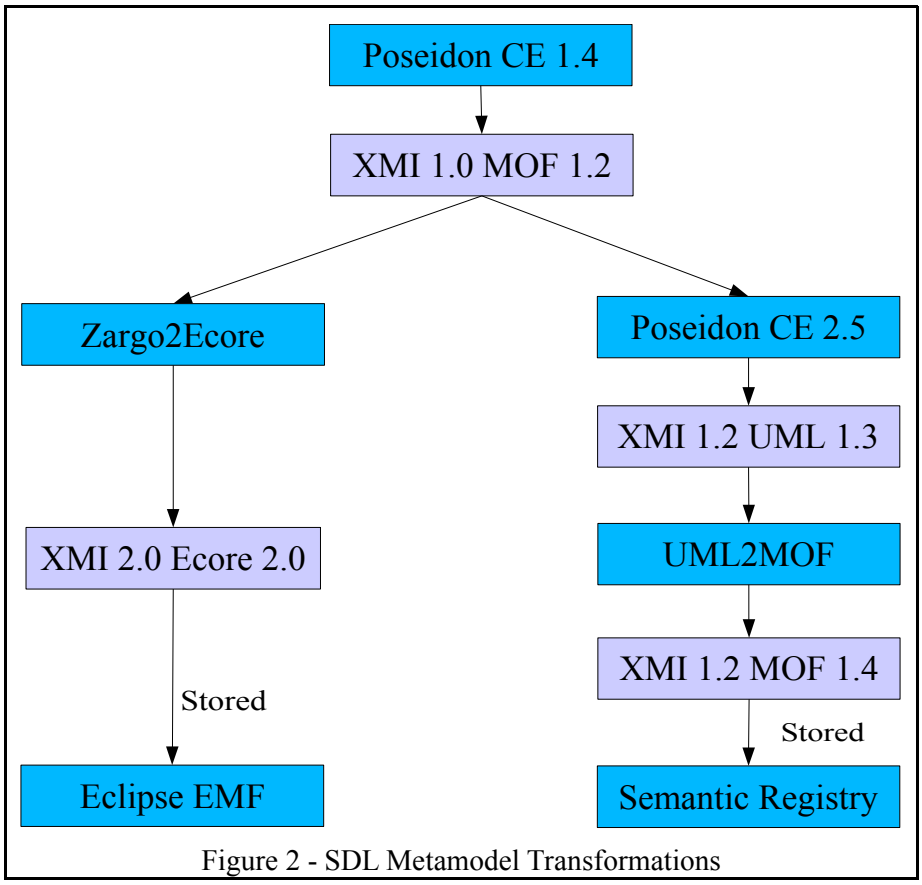

Figure 2 - SDL Metamodel Transformations

4.1.7 Edit the SDL Model

Using the SDL Editor the user can edit valid SDL Models. They will be validated against the SDL metamodel by the validation tool within the SDL Editor that also offers an effective tree based graphic interface that allows the editing of SDL Models in an intuitive way. The reader can refer to chapter 6 - SDL Example to get a detailed information about the SDL Editor interface.
The SDL Model may be stored in the file system in XMI 1.2 MOF 1.4 compliant format or it may be stored, in the same format, in the SR.
Note that an SDL Model is not expressed in MOF but is compliant to a MOF Metamodel; an SDL Model is in SDL XMI format, i.e. an XMI compliant to the SDL Metamodel, only the SDL Metamodel is expressed in MOF. Since the SDL Metamodel is a MOF XMI Metamodel it may be concluded that an SDL Model is a MOF compliant model, as such the SDL Models can be stored in MDR.

4.1.8 SDL Compiler

From the SDL Model it is hence possible to generate XMI, WSDL and Java using MDA based tools. Soluta.net implemented the SDL Compiler both to cross-compile the SDL to WSDL and to reverse WSDL to SDL so as to facilitate the creation of SDL from existing services. The SDL cross-compilation to Java is divided into two main parts: one dedicated to create the Java Interface representing the service as

it is and another dedicated to generate the whole Java DBE infrastructure strongly depending on the technical architecture of DBE. Such a kind of generation was thought to facilitate, as much as possible, the DBE services codification and then it should generate the greatest possible part of code (such as, for example, the complete implementation of the methods to access attributes of classes). More detailed information on SDL cross-compiling can be found in chapter 7 - Transformations Rules based on SDL.

4.2 MDA Consideration

Besides the technical approach for the definition and disseminations, the SDL specification is provided to the DBE project teams in a MOF compliant format; more specifically it is contained in an XMI MOF file. Such approach will enable XMI based tools to import the specification. For instance the SDL Metamodel was imported in Eclipse EMF to generate the SDL Editor and in ATL[19], by the SSL Compiler, to generate mappings between SSL and SDL models. The use of MDA tools eases the work of developer to obtain high quality code allowing the automatic generation of code.
As stated in chapter 4 - Introduction to SDL, the SDL Editor is generated using the MDA approach: one of the most interesting effects is that when the SDL Model changes the Editor is updated in just regenerating. This aspect might seem trivial but there are two main advantages: the first is that it is possible to test the SDL Metamodel as soon as it is ready, the second aspect is that the SDL may change very often and then, in this way, it can be avoided the manual effort of recoding the Editor. During the maintenance of the SDL, the necessary work for updating the Editor might create a sort of psychological block which could even lead to stop the evolution of the SDL.
Another important effect of the MDA approach is the possibility of using the transformation tool between the metamodels; for the SSL2SDL Compiler we use ATL, a QVT[20] implementation. Any change to the SDL Metamodel, or SSL, does not requires modification to the SSL2SDL Compiler rather it only determines some update to the SSL-SDL mapping.

4.3 Semantic Description in SDL

4.3.1 Why Should Semantics Be Described?

When a client and a Web Service interact, an agreement must exist between them regarding not only the mechanism of the interaction - the message formats, data types and protocols - but also the meaning and purpose of the interaction:agreement on the *semantics.* For example: *which are the results of sending a "PurchaseOrder" message?* The buyer will send money to the seller, and goods will be sent to the buyer. *What occurs if a printer service receives a "PrintDocument" message?* The result consists of printed pages. Additionally, a way to select a printer service with appropriate semantics such as color printing, should be offered to your PDA, instead of any service that accepts "PrintDocument" message formats.

19 Atlas Transformation Language[ATL]
20 Query/ Views/ Transformations[QVT]

Semantics can be regarded by another point of view, that of the context in which client and service interact each other. For example, a “PurchaseOrder” message is designated to happen in the context of a business-level agreement to buy goods. Or a “ShippedReceipt” may be sent in the context of an imminent PurchaseOrder. The semantics of the interaction are determined by the context.
In the daily humankind situations, interactions are not explicit but take place under tacit agreements, semantics or contexts. For example, you might receive a shipping receipt that does not correspond to the purchase order to which belongs to. Explicit semantics or context of the interaction would be clearer, even if the flexibility of humans allows them to manage to the ambiguity of the situation. Moreover, to automate human processes using Web Service they must be explicit, as machines do not manage ant ambiguity as humans can rather do.

4.3.2 Where Should Semantics Be Described?

Lack of knowledge of Web Service semantics makes its description not functional. The main issues are: where semantics must be described and how a client can find them. The related options may be:

- Using an ad-hoc, out-of-band[21] mechanism that is not shown in the Web Service description. This is obviously not a good option, as it will be a non deterministic manner for an agent who discovers a Web Service description to find the semantics of descriptions.
- Describing the semantics in an indicated section of the Web Service description document. This is not a practical option, because the parser that processes the Web Service description and the parser that interprets the semantics may be different and with different purposes.
- The Web Service description can reference a distinct document that describes the semantics, by including the URI of the document as exemplified below. WSDL 2.0 chooses this manner of describing semantics, using the “targetNamespace” URL. This is the choice adopted for the SDL.

4.3.3 Web Service Discovery

Web Service discovery indicates the capacity to dynamically find a Web Service for what an application requires. To give an example, finding, in a dynamically manner, a nearby printer service for printing out a document from my PDA.
The ambiguity of the term “discovery” is indicated for two main reasons. The first reason is that people usually discuss about two types of “discovery”:

- how to *find* a previously unknown service
- how to *select* a service from other already known services

These two types are different taking in consideration their requirements and implications.
The second reason refers to who is going to *perform* the “discovery” - a human or an application, having also different interface and implications.
Recommendations on different types of Web Service discovery distinguish between centralized (an example being UDDI) and decentralized (as conventional Web searching). Although WSDL 2.0 does not deal with the problem of Web

21 out-of-band data is a separate stream of data from the main data stream.

Service discovery, discovery mechanisms intend to provide improved capabilities to a client which searches for a specific type of Web Service that are going to use WSDL 2.0. For example, useful ways to categorize Web Services are possibly offered by discovery mechanisms, or data on the trustworthiness of the salesmen whose Web Services are listed.
Semantics is important to discovery as it fulfil the client' s requirements of selecting a Web Service with a requested semantics. Anyway, a complete Web Service description of the syntax for interacting with the service and of the semantics of the interaction, allow the client to use a possible service regardless the service discoveryprotocol.

4.4 Abstract Interface versus Concrete Protocols

The SDL supplies an Abstract Interface of the service, this interface has to define the service capability using a platform independent model (PIM). In this document the reader is supposed to be proficient with the CIM-PIM-PSM concept, but it may be interesting to point out a particular aspect of the PIM concept. As stated previously, the SDL has to be a language for platform independent modelling, and, for SDL purpose, this means mainly middleware independent because the platform, for the DBE, is the “interoperability platform”, i.e. the middleware. The concept of platform independent is a little bit strange: it is accepted that a model can not be *completely* platform independent but it may be *enough* independent, that is the model is based on a *virtual* platform which may be implemented by the largest possible number of concrete platforms. The SDL is platform independent as much as the DBE architecture because it is based on the DBE Virtual Platform[22].
The DBE Virtual Platform has to be enough abstract to represent the largest number of concrete platforms but also enough concrete to allow the generation of the PSM model related to each concrete protocol that may be used by DBE. The old chicken and egg problem.
The MDA approach can help to point out the problem and to ease the changes to the entire infrastructure when the metamodel changes.

22 The virtual platform defined by the DBE

5.SDL Definition

5.1 Building an UML Class Diagram to Model SDL

Version 2.0 of SDL is represented in the UML Class Diagrams shown in Figure 3 - SDL v3.0 Class Diagram and in UML Inheritance Diagram shown in Figure 4 - SDL v 3.0 Inheritance Diagram: they have been produced with Poseidon UML editor [HMAD].

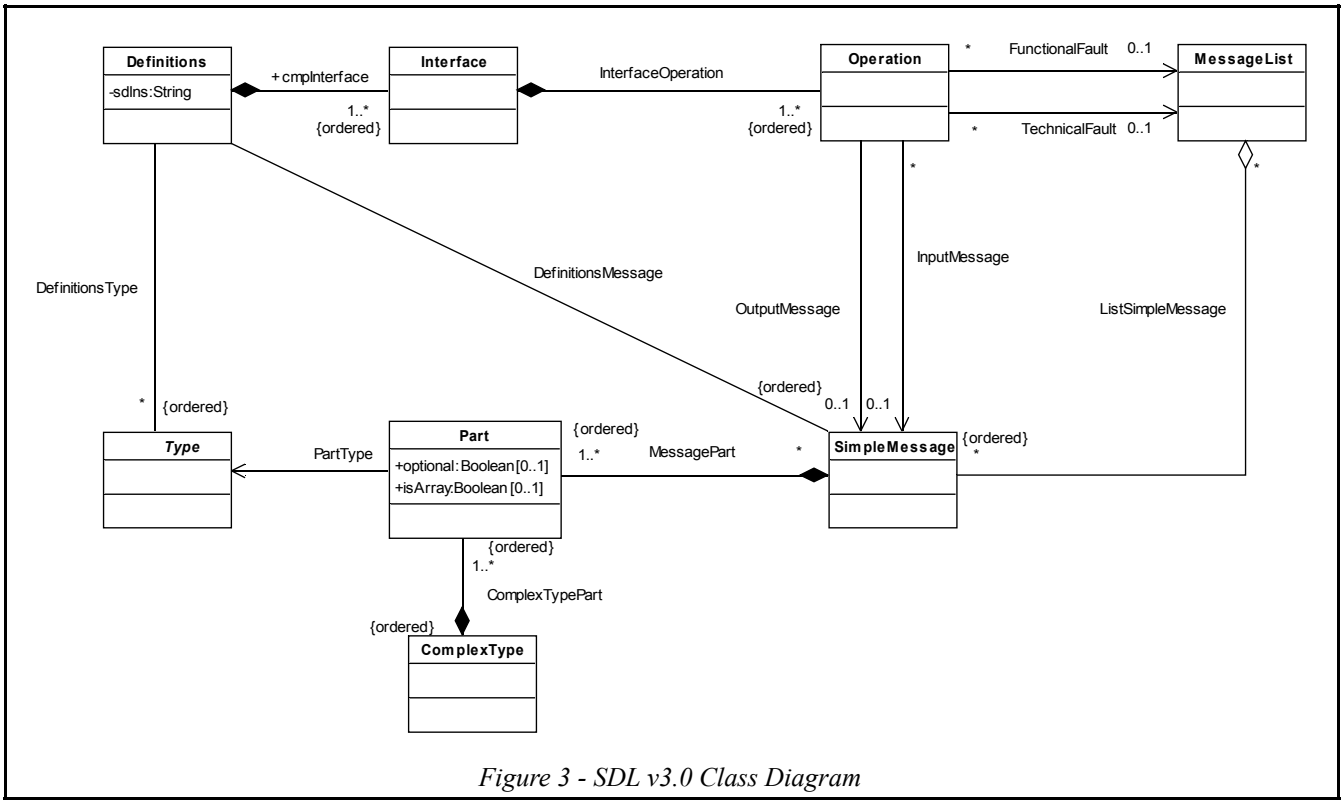

Figure 3 - SDL v3.0 Class Diagram

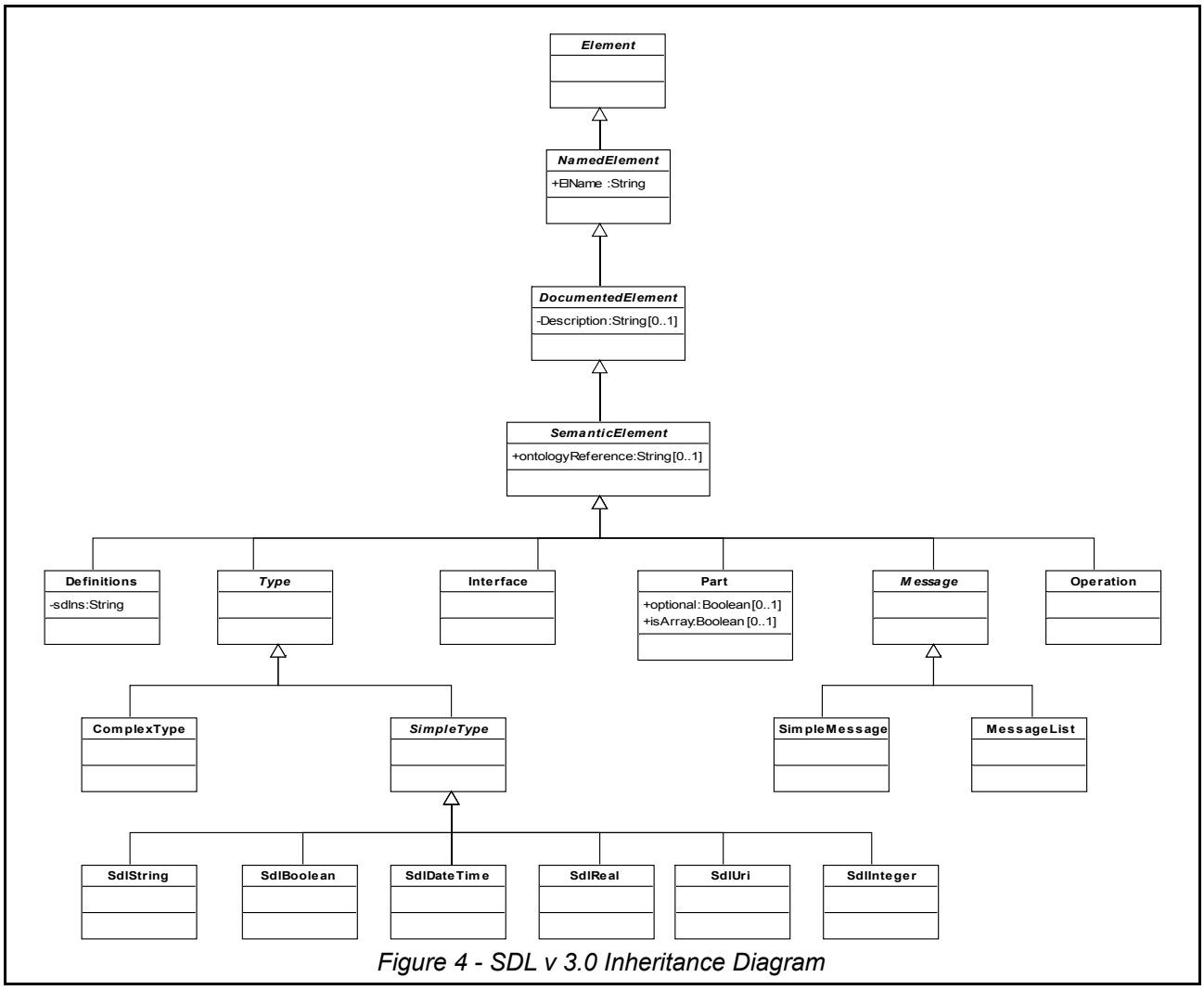

Figure 4 - SDL v 3.0 Inheritance Diagram

As stated in chapter 4 - Introduction to SDL the SDL Metamodel defines the SDL Language. The normative definition of SDL is the XMI 1.2 MOF 1.4 defined in chapter 5.3 - The SDL Metamodel.

5.2 The SDL specification

The Figure 3 - SDL v3.0 Class Diagram shows all the SDL classes and their relations; in this paragraph they are all individually discussed.

5.2.1 SemanticElement

All the SDL Elements derive from the SemanticElement. The SemanticElement diagram is shown in Figure 5 - Semantic Element.

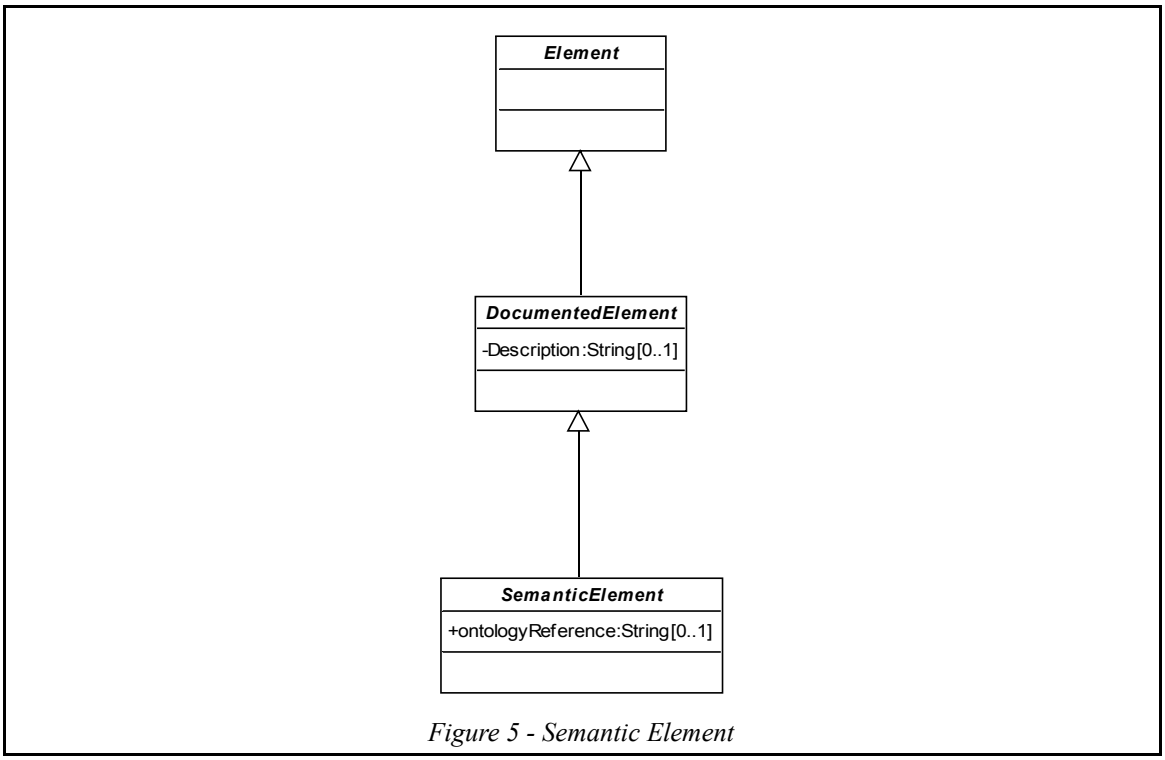

Figure 5 - Semantic Element

The ontologyReference has been introduced in accordance with the consideration expressed in 4.3 Semantic Description in SDL stating that the link to an ontology seems to be the simplest way to introduce semantics in the SDL.

5.2.2 Definitions

Definitions is the root of a Service Description, it contains the Type, Interface and Message definition. The Figure 6 - Definitions UML Diagrams shows its structure.

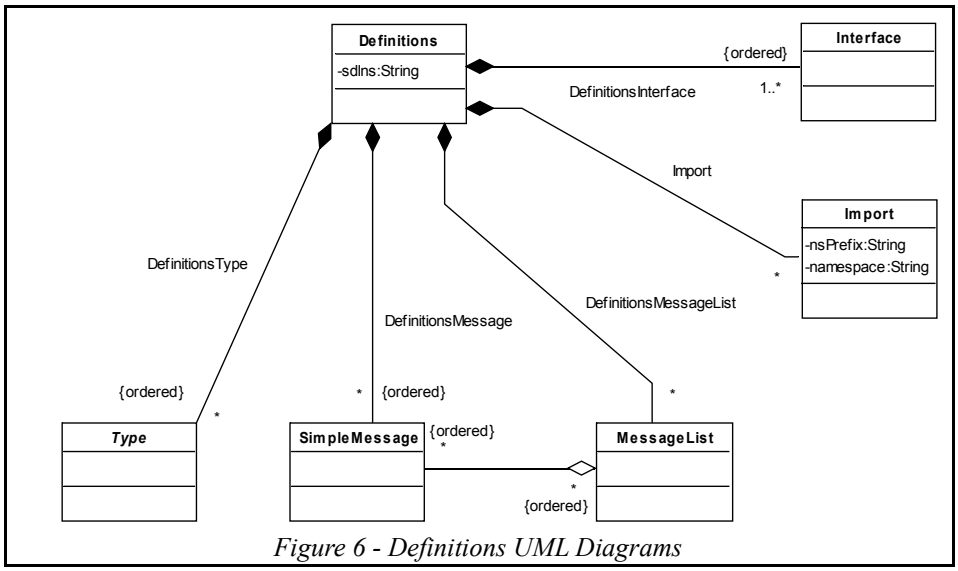

Figure 6 - Definitions UML Diagrams

Definitions is a ***composition*** of Interfaces, Types, Import and Messages (SimpleMessage and MessageList). Composition means that all the objects *belong* to the *Definitions* and an object can only belong to a single *Definitions*.
Another way to define the *Definitions* is to use XMI-MOF notation, refers to 5.3.2-XMI SDL Metamodel for the normative SDL Metamodel.

5.2.3 Interface

Interface is a logical grouping of operations, it represents an abstract Service type, it may be annotated with ontology.

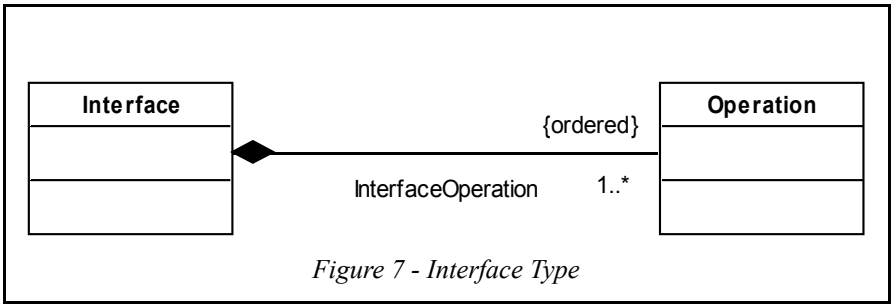

Figure 7 - Interface Type

"Operation" describes the elementary operations that an interface supports; it is a Composition, i.e. each single operation may be part of only one Interface.

5.2.4 Interface Operation

Operation is a sequence of messages related to a single Service action. Each operation defines the input and the output and the fault/exception messages. There are two types of fault messages: Technical and Functional; a Technical Exception rises, for instance, when the data base is down, a Functional Exception is returned when the transaction can not be concluded for a "business reason". An example of business reason may be that, in the case of a Hotel reservation service, rooms for a certain selected date are not available. The management of the Technical and Functional Exception is, obviously, different: if the database is down it is possible to retry the transaction after a while, if there are not rooms the customer may be invited to change the reservation date. Refers to chapter 5.2.6 - Message List for further information about Functional and Technical exception.

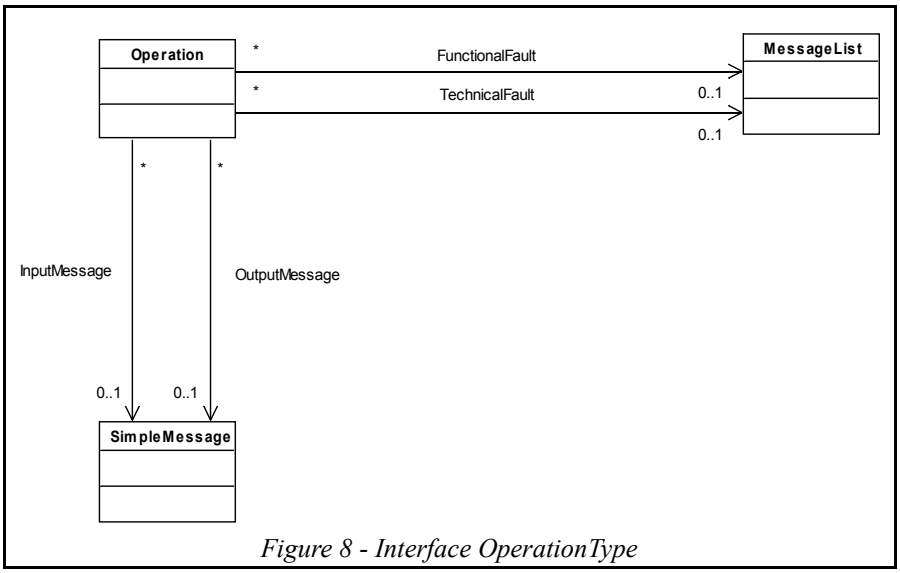

Figure 8 - Interface OperationType

Note that in SDL the granularity of operation is more detailed than in SSL/BML[23]. While in SSL/BML only the semantically significant operations are expressed, in SDL also the "technical operations" are expressed. If referring again to the MakeReservation operation in the HotelReservation service case: in SSL/BML only the MakeReservation operation and the related messages are defined; the InputMessage for MakeReservation may contain, for instance, a RoomType and the reservation date. In SDL it is possible to define also the method getRoomTypeList that returns a list of Strings (or RoomType) describing the RoomType available in that Hotel. In SDL all the support operations that may help to perform the transaction as well as all the auxiliaries methods are defined.

5.2.5 Simple Message

A "message" is the basic unit of communication between a Service and a Client and vice versa. "Message" is annotated with ontology and is composed by Part Component. Each Operation specifies messages for Input, Output and Fault.

23 Semantic Service Language[SSL] See [DBECoreArch]./Business Modelling Language[BML]

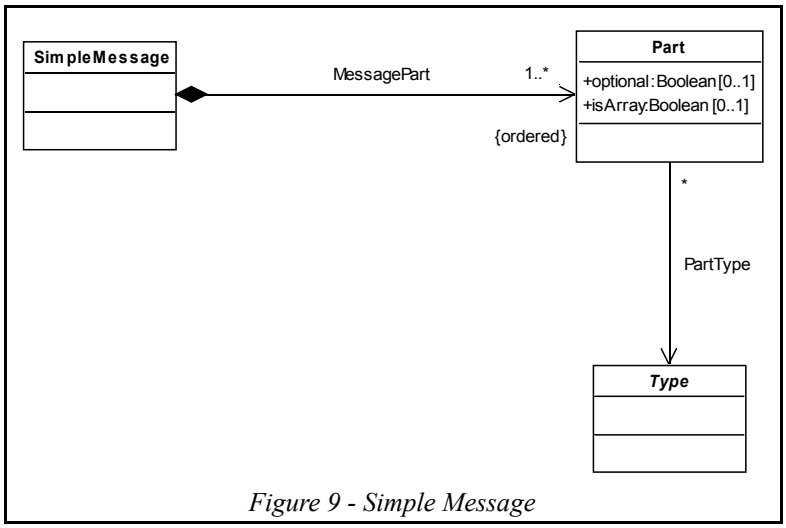

Figure 9 - Simple Message

Part is the elementary part of a Message, it may be optional or mandatory. The concept of Array is only referred to the multiplicity of the part, not to the possible implementation as array, collection etc.

5.2.6 Message List

The MessageList has been introduced to realize the concept of Technical and Functional Exception.

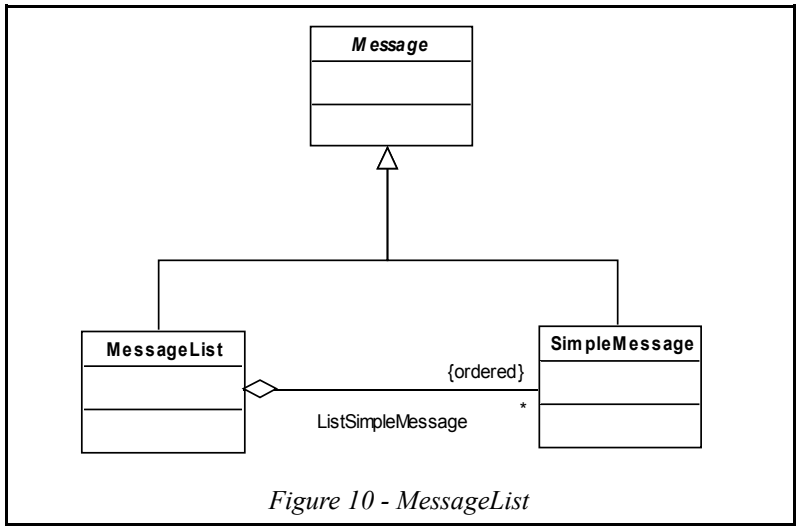

Figure 10 - MessageList

The concept of MessageList is shown in Figure 10 - MessageList and, as the name suggests, is an aggregation of SimpleMessages.
More interesting is the concept of Technical and Functional Exception which the MessageList depends on.
The MessageList is a list of possibilities when a Technical or Functional error occurs. The picture Figure 8 - Interface OperationType shows the relation between the Operation and the MessageList.

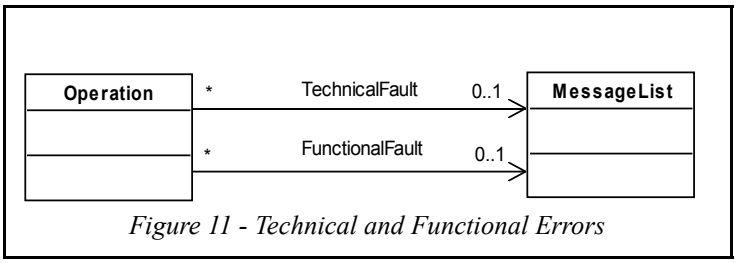

Figure 11 - Technical and Functional Errors

When an Exception occurs, it is suggested that the result may be specialized in relation to that exception. For instance, if a proxy is not available it may be useful that the Technical Exception contains the ProxyID while if, for instance, a Network is down it may be useful to know the number of attempts to connect and the URL of the entry node. In this way it is possible to understand the type of the error and take on the right actions.

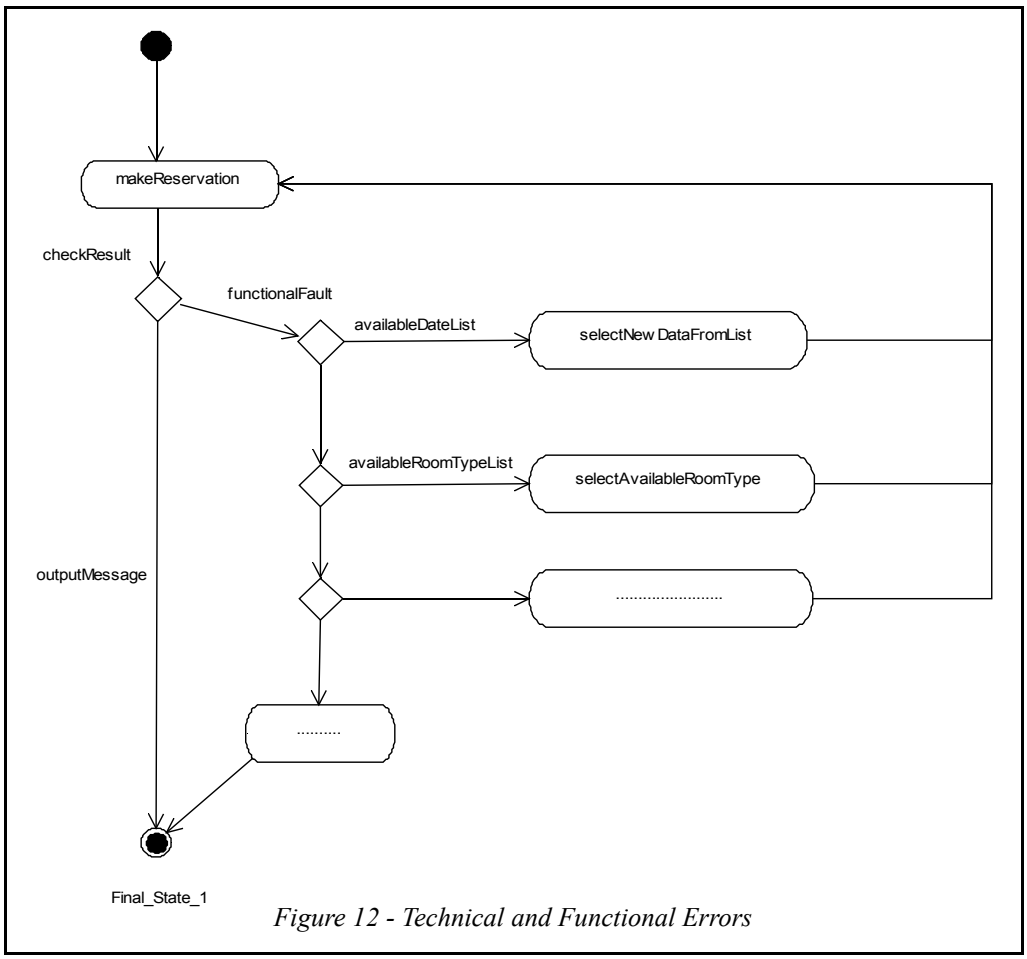

Figure 12 - Technical and Functional Errors

These considerations are even more important when we think about the Functional Exception. The Service Composer might use the MessageList as “guard” to decide how to proceed with the Service Chain Execution, or how to invoke operations of the single service. In the example in Figure 11 - Technical and Functional Errors a possible use of MessageList is shown.

When a reservation service is executed the result may be that an outputMessage representing the invocation of the service is completed in a correct way, but "*how the service can report that for the selected date there are no rooms"?* And more than this, *"how can the service return the list of the nearest available data"?* And *"how can the client understand the type of error"?* In the Figure 12 - Technical and Functional Errors a possible process is shown. If the reservation is not possible a functional error is returned: if the requested room type is not available a list of possible room types is returned and then the client service may ask the user to select a new type of available room. Then the reservation may be done. If the reservationNumber (standard outputMessage) is obtained the invocation is completed otherwise the Functional or Technical error has to be managed.

5.2.7 Type

There are two kinds of type: SimpleType and ComplexType, the Figure 13 - Type shows the relationships between types.
ComplexType and SimplexType are Types, the main difference is that ComplexType is composed by an ordered list of parts, and each part may be an element of a particular Type (REFPartType), Simple or Complex.

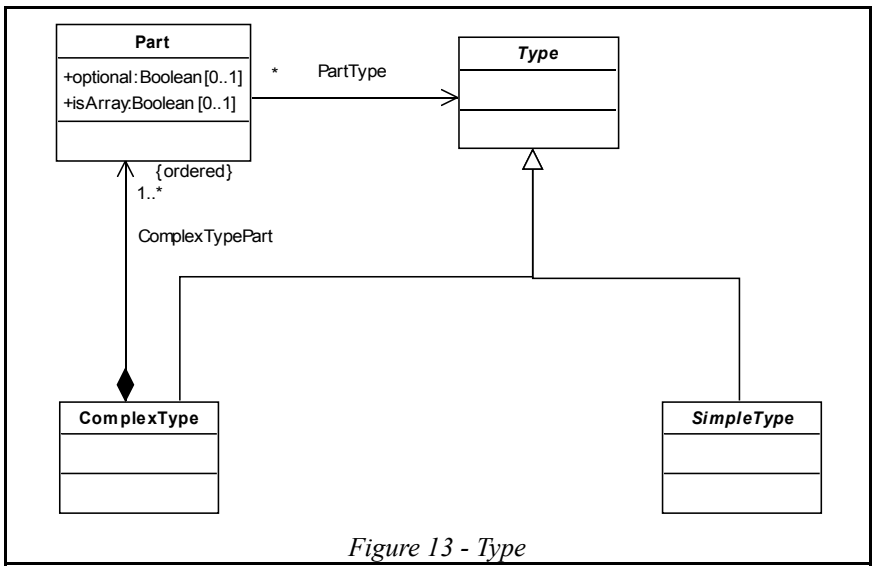

Figure 13 - Type

SimpleType is an abstract concept, as shown in Figure 14 - SimpleType the SDL SimpleType are: SdlBoolean, SdlInteger, SdlReal, SDLDateTime, SdlUri and SdlString. This is the SimpleType used in SDL Models.

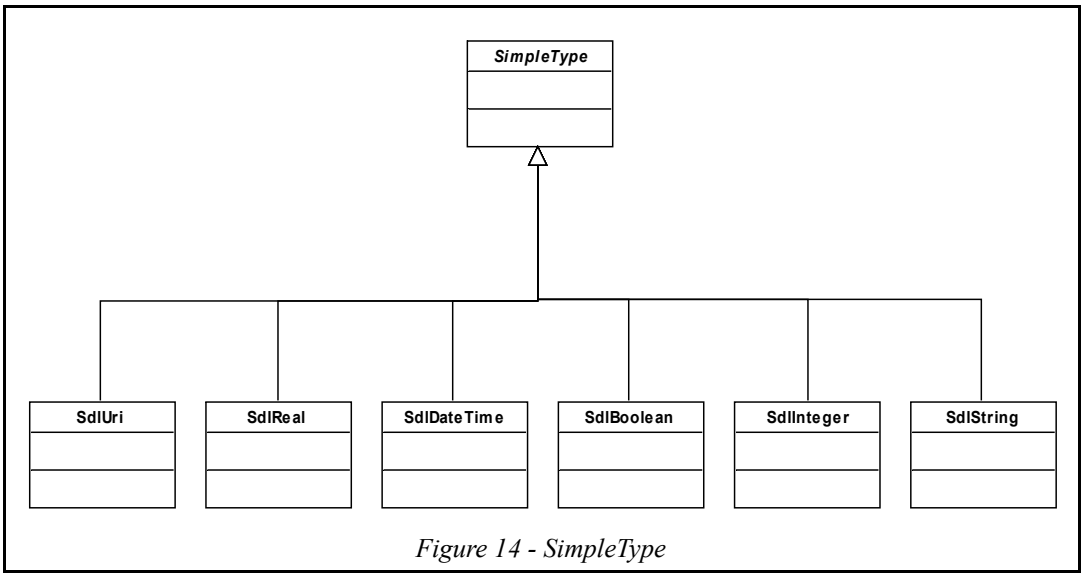

Figure 14 - SimpleType

5.2.8 Import

Import allows to use ComplexTypes and messages defined in other SDL models . Such a possibility to import types has been introduced in version 3.0 of SDL. It represents a way to simplify the SDL editing and it facilitates types and messages reuse. More in particular, import is essential to use ComplexTypes such as, for example, those related to security or “base services” (i.e. e-mail, sms). Import should also facilitate automatic composition of services which becomes easier when services use the same kind of messages (therefore not necessitating of data mapping). The reuse of services should also facilitates the emergence of standards to define the most used Types.
Conceptually, import should refer to a model of which some parts are used; however it has been preferred to opt for an import that states the physic copy of types and messages within the SDL, for the following reasons:

- DBE is a distributed environment and it is not possible to guarantee access to all the resources.
- As a subject exerting a centralized control to define types does not exist, it is not possible to exclude at all unauthorized modifications or deletions of models referenced by the import.
- Modifications to a model have not to interfere with those models that have imported it:
  - when a model is modified a new version must be released to avoid bad performances on other models that are using it.
- Off-line modifications of SDL must be possible.
- Service Manifest should be self-contained as well as the related contained models:
  - such a consideration is only partially true because SDL contains some reference to ontology, but this should not influence the general concept, as a service is defined anyway in a formal way also without referring to the ontology.

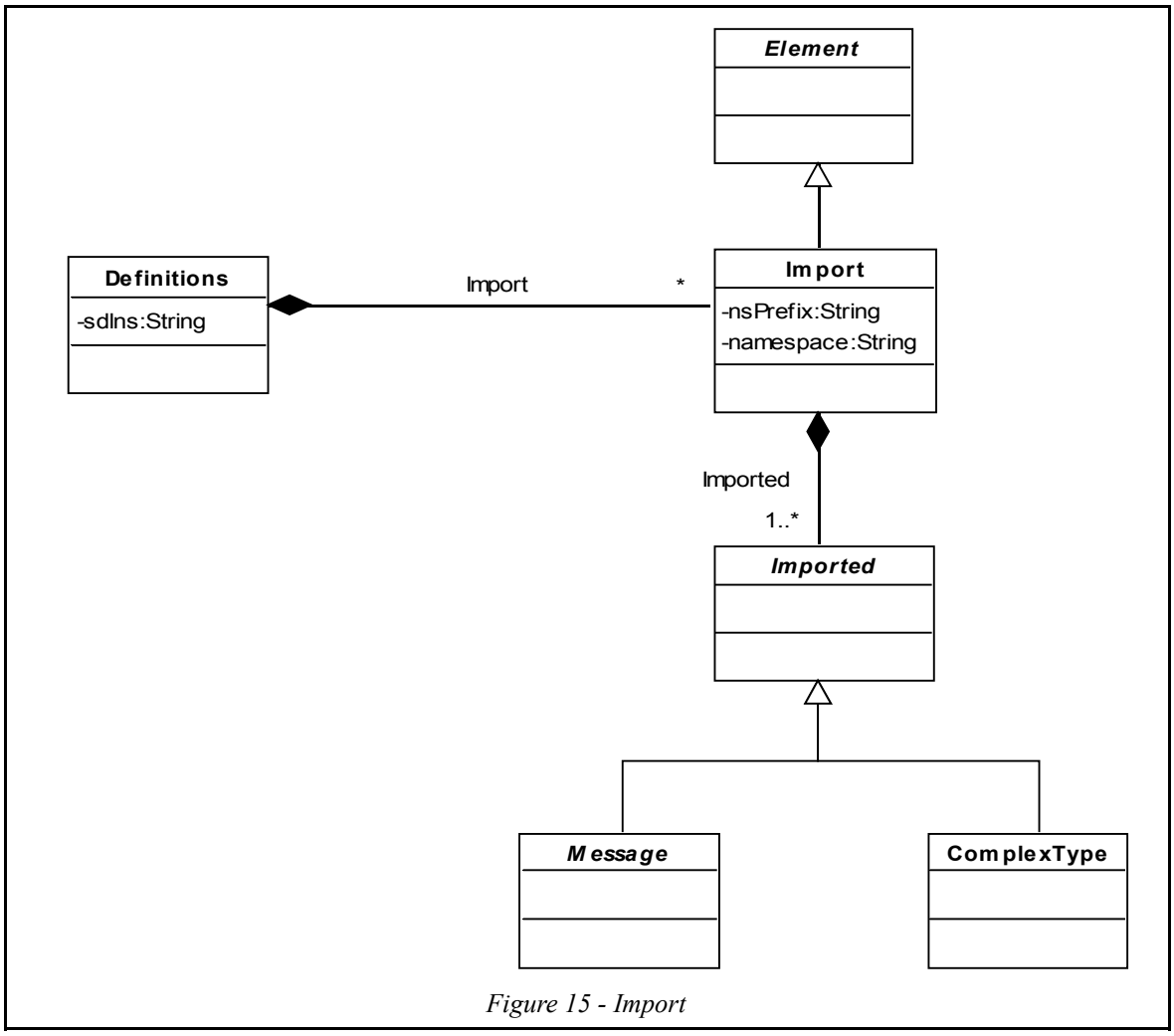

Figure 15 - Import

As shown in Figure 15 - Import ComplexTypes and messages can be imported.
It is not possible to modify the imported Types but in case of further extensions or personalization they can be copied and thus modified: inheritance is not supposed in SDL.

5.3 The SDL Metamodel

5.3.1 SDL, XMI SDL and XML SDL

The first version of SDL, called XML-SDL or SDL1.0, was very similar to WSDL. This first version was defined to allow all DBE developers to use it an early stage of the project before the SDL Editor was made available. The SDL1.0 was defined by an XML Schema and had an important peculiarity: it was easy to write and validate using any XML editor.
The second version of SDL was a MOF based language, a language that can be transformed using MDA tools such as Eclipse EMF or ATL (QVT implementation).

In such a transition changes have not effected the model rather the way to represent it. This difference is made clear by comparing the sample models shown in Figure 17 - TicketAgent SDL1.0 (XML SDL) and in Figure 18 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types – Part 1.

5.3.2 XMI SDL Metamodel

As stated in chapter 3-SDL Features the SDL has to be an XMI1.2 MOF1.4 specification. In chapter 4-Introduction to SDL the process used to obtain the SDL Metamodel has been explained in detail. In this chapter the SDL Metamodel is listed. The SDL Metamodel is delivered in the Collabnet in the project *languages,* folder *WP-16SDL* file *SDL Metamodel 3.0 XMI-MOF.*[24]

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' xmlns:Model = 'org.omg.xmi.namespace.Model' timestamp = 'Wed Jul 19 11:36:43 CEST 2006'>
 <XMI.header>
  <XMI.documentation>
   <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
   <XMI.exporterVersion>1.0</XMI.exporterVersion>
  </XMI.documentation>
 </XMI.header>
 <XMI.content>
  <Model:Package xmi.id = 'a1' name = 'PrimitiveTypes' annotation = '' isRoot = 'true'
   isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'>
   <Model:Namespace.contents>
    <Model:PrimitiveType xmi.id = 'a2' name = 'Integer' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
    <Model:PrimitiveType xmi.id = 'a3' name = 'Long' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
    <Model:PrimitiveType xmi.id = 'a4' name = 'Float' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
    <Model:PrimitiveType xmi.id = 'a5' name = 'Double' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
    <Model:PrimitiveType xmi.id = 'a6' name = 'Boolean' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
    <Model:PrimitiveType xmi.id = 'a7' name = 'String' annotation = '' isRoot = 'true'
     isLeaf = 'true' isAbstract = 'false' visibility = 'public_vis'/>
   </Model:Namespace.contents>
  </Model:Package>
  <Model:Package xmi.id = 'a8' name = 'org.dbe.sdl' annotation = '' isRoot = 'false'
   isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'>
   <Model:Namespace.contents>
     <Model:Tag xmi.id = 'aaa2' name = 'org.omg.mof.idl_prefixorg.omg.mof.idl_prefix'
     annotation = '' tagId = 'org.omg.mof.idl_prefixorg.omg.mof.idl_prefix'>
     <Model:Tag.values>urn:sdl.ecore</Model:Tag.values>
     <Model:Tag.elements>
      <Model:Package xmi.idref = 'a8'/>
     </Model:Tag.elements>
    </Model:Tag>
    <Model:Tag xmi.id = 'aaa3' name = 'org.omg.xmi.namespace' annotation = ''
     tagId = 'org.omg.xmi.namespace'>
     <Model:Tag.values>sdl</Model:Tag.values>
     <Model:Tag.elements>
```

24 https://languages.digital-ecosystem.net/files/documents/18/930/SDLModelMOF.xmi

```
      <Model:Package xmi.idref = 'a8'/>
    </Model:Tag.elements>
  </Model:Tag>
  <Model:Association xmi.id = 'a9' name = 'CMPIMportImported' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a10' name = 'cmpImportImp' annotation = ''
        isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a11'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a12' name = 'cmpImportedImp' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a13'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a13' name = 'Imported' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'/>
  <Model:Association xmi.id = 'a14' name = 'CMPDefinitionImport' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a15' name = 'cmpDefinitionImp' annotation = ''
        isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a16'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a17' name = 'cmpImportDef' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
```

```
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a11'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a11' name = 'Import' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Attribute xmi.id = 'a18' name = 'nsPrefix' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a7'/>
        </Model:TypedElement.type>
      </Model:Attribute>
      <Model:Attribute xmi.id = 'a19' name = 'namespace' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a7'/>
        </Model:TypedElement.type>
      </Model:Attribute>
      <Model:Reference xmi.id = 'a20' name = 'cmpImportedImp' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a13'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a12'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a21'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
```

```
<Model:Class xmi.id = 'a21' name = 'Element' annotation = '' isRoot = 'false'
 isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'/>
<Model:Association xmi.id = 'a22' name = 'cmpDefinitionsMessageList' annotation = ''
 isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
 isDerived = 'false'>
 <Model:Namespace.contents>
  <Model:AssociationEnd xmi.id = 'a23' name = 'cmpDefinitions' annotation = ''
   isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
   <Model:AssociationEnd.multiplicity>
    <XMI.field>1</XMI.field>
    <XMI.field>1</XMI.field>
    <XMI.field>false</XMI.field>
    <XMI.field>true</XMI.field>
   </Model:AssociationEnd.multiplicity>
   <Model:TypedElement.type>
    <Model:Class xmi.idref = 'a16'/>
   </Model:TypedElement.type>
  </Model:AssociationEnd>
  <Model:AssociationEnd xmi.id = 'a24' name = 'cmpMessageList' annotation = ''
   isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
   <Model:AssociationEnd.multiplicity>
    <XMI.field>0</XMI.field>
    <XMI.field>-1</XMI.field>
    <XMI.field>false</XMI.field>
    <XMI.field>true</XMI.field>
   </Model:AssociationEnd.multiplicity>
   <Model:TypedElement.type>
    <Model:Class xmi.idref = 'a25'/>
   </Model:TypedElement.type>
  </Model:AssociationEnd>
 </Model:Namespace.contents>
</Model:Association>
<Model:Association xmi.id = 'a26' name = 'REFFaultMessageListOperation'
 annotation = '' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
 isDerived = 'false'>
 <Model:Namespace.contents>
  <Model:AssociationEnd xmi.id = 'a27' name = 'refMessageFaultOperationMessageList'
   annotation = '' isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
   <Model:AssociationEnd.multiplicity>
    <XMI.field>0</XMI.field>
    <XMI.field>-1</XMI.field>
    <XMI.field>false</XMI.field>
    <XMI.field>true</XMI.field>
   </Model:AssociationEnd.multiplicity>
   <Model:TypedElement.type>
    <Model:Class xmi.idref = 'a28'/>
   </Model:TypedElement.type>
  </Model:AssociationEnd>
  <Model:AssociationEnd xmi.id = 'a29' name = 'refTechnicalException' annotation = ''
   isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
   <Model:AssociationEnd.multiplicity>
    <XMI.field>0</XMI.field>
    <XMI.field>1</XMI.field>
    <XMI.field>false</XMI.field>
    <XMI.field>true</XMI.field>
   </Model:AssociationEnd.multiplicity>
```

```
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a25'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a30' name = 'Message' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a13'/>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Association xmi.id = 'a32' name = 'refOperationMessageList' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a33' name = 'refOperationOperationMessageList'
        annotation = '' isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a28'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a34' name = 'refFunctionalException' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a25'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a35' name = 'CMPMessageListSimpleMessage' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a36' name = 'cmpSimpleMessage' annotation = ''
        isNavigable = 'true' aggregation = 'shared' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
```

```
          <Model:Class xmi.idref = 'a25'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a37' name = 'cmpMessageList' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a25' name = 'MessageList' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Reference xmi.id = 'a39' name = 'cmpMessageList' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a37'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a30'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a40' name = 'SdlUri' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a41'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a42' name = 'SdlDateTime' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a41'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a43' name = 'SdlBoolean' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a41'/>
```

```
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Association xmi.id = 'a44' name = 'CMPComplexTypePart' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a45' name = 'cmpComplexType' annotation = ''
        isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a46'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a47' name = 'cmpPart' annotation = '' isNavigable = 'true'
        aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a48'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a49' name = 'REFOutputMessageOperation' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a50' name = 'refOperationOutputOperationMessage'
        annotation = '' isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a28'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a51' name = 'refOutputMessage' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
```

```
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a52' name = 'REFPartType' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a53' name = 'refPart' annotation = '' isNavigable = 'true'
        aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a54'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a55' name = 'refType' annotation = '' isNavigable = 'true'
        aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a48'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a56' name = 'REFImputMessageOperation' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a57' name = 'refOperationInputOperationMessage'
        annotation = '' isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a28'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a58' name = 'refInputMessage' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
```

```
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a59' name = 'CMPDefinitionsType' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a60' name = 'cmpDefinitionsDefinitionsType'
        annotation = '' isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a16'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a61' name = 'cmpType' annotation = '' isNavigable = 'true'
        aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a54'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a62' name = 'CMPInterfaceOperation' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a63' name = 'cmpInterfaceInterfaceOperation'
        annotation = '' isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a64'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a65' name = 'cmpOperation' annotation = ''
```

```
      isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
      <Model:AssociationEnd.multiplicity>
        <XMI.field>1</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>true</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:AssociationEnd.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a28'/>
      </Model:TypedElement.type>
    </Model:AssociationEnd>
  </Model:Namespace.contents>
</Model:Association>
<Model:Class xmi.id = 'a66' name = 'SdlInteger' annotation = '' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
  <Model:GeneralizableElement.supertypes>
    <Model:Class xmi.idref = 'a41'/>
  </Model:GeneralizableElement.supertypes>
</Model:Class>
<Model:Class xmi.id = 'a67' name = 'SdlReal' annotation = '' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
  <Model:GeneralizableElement.supertypes>
    <Model:Class xmi.idref = 'a41'/>
  </Model:GeneralizableElement.supertypes>
</Model:Class>
<Model:Class xmi.id = 'a68' name = 'SdlString' annotation = '' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
  <Model:GeneralizableElement.supertypes>
    <Model:Class xmi.idref = 'a41'/>
  </Model:GeneralizableElement.supertypes>
</Model:Class>
<Model:Class xmi.id = 'a41' name = 'SimpleType' annotation = '' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
  <Model:GeneralizableElement.supertypes>
    <Model:Class xmi.idref = 'a54'/>
  </Model:GeneralizableElement.supertypes>
</Model:Class>
<Model:Class xmi.id = 'a46' name = 'ComplexType' annotation = '' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
  <Model:Namespace.contents>
    <Model:Reference xmi.id = 'a69' name = 'cmpPart' annotation = '' scope = 'instance_level'
      visibility = 'public_vis' isChangeable = 'true'>
      <Model:StructuralFeature.multiplicity>
        <XMI.field>1</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>true</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:StructuralFeature.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a48'/>
      </Model:TypedElement.type>
      <Model:Reference.referencedEnd>
        <Model:AssociationEnd xmi.idref = 'a47'/>
      </Model:Reference.referencedEnd>
    </Model:Reference>
  </Model:Namespace.contents>
```

```
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a13'/>
      <Model:Class xmi.idref = 'a54'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a54' name = 'Type' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Association xmi.id = 'a70' name = 'CMPMessagePart' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a71' name = 'cmpMessageMessagePart' annotation = ''
        isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a72' name = 'cmpPart' annotation = '' isNavigable = 'true'
        aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a48'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Association xmi.id = 'a73' name = 'CMPDefinitionsMessage' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a74' name = 'cmpDefinitionsDefinitionsMessage'
        annotation = '' isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a16'/>
        </Model:TypedElement.type>
```

```
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a75' name = 'cmpMessage' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a48' name = 'Part' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Attribute xmi.id = 'a76' name = 'optional' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a6'/>
        </Model:TypedElement.type>
      </Model:Attribute>
      <Model:Attribute xmi.id = 'a77' name = 'isArray' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a6'/>
        </Model:TypedElement.type>
      </Model:Attribute>
      <Model:Reference xmi.id = 'a78' name = 'refPart' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a54'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a53'/>
        </Model:Reference.referencedEnd>
```

```
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a28' name = 'Operation' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Reference xmi.id = 'a79' name = 'refTechnicalException' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a25'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a29'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
      <Model:Reference xmi.id = 'a80' name = 'refOutputMessage' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a51'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
      <Model:Reference xmi.id = 'a81' name = 'refInputMessage' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a38'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a58'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
      <Model:Reference xmi.id = 'a82' name = 'refFunctionalException' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
```

```
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a25'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a34'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a38' name = 'SimpleMessage' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Reference xmi.id = 'a83' name = 'cmpPart' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a48'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a72'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a30'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Association xmi.id = 'a84' name = 'CMPDefinitionsInterface' annotation = ''
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis'
    isDerived = 'false'>
    <Model:Namespace.contents>
      <Model:AssociationEnd xmi.id = 'a85' name = 'cmpdefinitionsDefinitionsInterface'
        annotation = '' isNavigable = 'true' aggregation = 'composite' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a16'/>
        </Model:TypedElement.type>
```

```
      </Model:AssociationEnd>
      <Model:AssociationEnd xmi.id = 'a86' name = 'cmpInterface' annotation = ''
        isNavigable = 'true' aggregation = 'none' isChangeable = 'true'>
        <Model:AssociationEnd.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:AssociationEnd.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a64'/>
        </Model:TypedElement.type>
      </Model:AssociationEnd>
    </Model:Namespace.contents>
  </Model:Association>
  <Model:Class xmi.id = 'a64' name = 'Interface' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Reference xmi.id = 'a87' name = 'cmpOperation' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a28'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a65'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a16' name = 'Definitions' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Attribute xmi.id = 'a88' name = 'sdlns' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>1</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a7'/>
        </Model:TypedElement.type>
      </Model:Attribute>
      <Model:Reference xmi.id = 'a89' name = 'cmpMessage' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
```

```
        <XMI.field>0</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>true</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:StructuralFeature.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a38'/>
      </Model:TypedElement.type>
      <Model:Reference.referencedEnd>
        <Model:AssociationEnd xmi.idref = 'a75'/>
      </Model:Reference.referencedEnd>
    </Model:Reference>
    <Model:Reference xmi.id = 'a90' name = 'cmpInterface' annotation = '' scope = 'instance_level'
      visibility = 'public_vis' isChangeable = 'true'>
      <Model:StructuralFeature.multiplicity>
        <XMI.field>1</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>true</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:StructuralFeature.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a64'/>
      </Model:TypedElement.type>
      <Model:Reference.referencedEnd>
        <Model:AssociationEnd xmi.idref = 'a86'/>
      </Model:Reference.referencedEnd>
    </Model:Reference>
    <Model:Reference xmi.id = 'a91' name = 'cmpMessageList' annotation = ''
      scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'>
      <Model:StructuralFeature.multiplicity>
        <XMI.field>0</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>false</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:StructuralFeature.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a25'/>
      </Model:TypedElement.type>
      <Model:Reference.referencedEnd>
        <Model:AssociationEnd xmi.idref = 'a24'/>
      </Model:Reference.referencedEnd>
    </Model:Reference>
    <Model:Reference xmi.id = 'a92' name = 'cmpImportDef' annotation = '' scope = 'instance_level'
      visibility = 'public_vis' isChangeable = 'true'>
      <Model:StructuralFeature.multiplicity>
        <XMI.field>0</XMI.field>
        <XMI.field>-1</XMI.field>
        <XMI.field>false</XMI.field>
        <XMI.field>true</XMI.field>
      </Model:StructuralFeature.multiplicity>
      <Model:TypedElement.type>
        <Model:Class xmi.idref = 'a11'/>
      </Model:TypedElement.type>
      <Model:Reference.referencedEnd>
        <Model:AssociationEnd xmi.idref = 'a17'/>
      </Model:Reference.referencedEnd>
```

```
      </Model:Reference>
      <Model:Reference xmi.id = 'a93' name = 'cmpType' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>-1</XMI.field>
          <XMI.field>true</XMI.field>
          <XMI.field>true</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:Class xmi.idref = 'a54'/>
        </Model:TypedElement.type>
        <Model:Reference.referencedEnd>
          <Model:AssociationEnd xmi.idref = 'a61'/>
        </Model:Reference.referencedEnd>
      </Model:Reference>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a31'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a31' name = 'SemanticElement' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Attribute xmi.id = 'a94' name = 'ontologyReference' annotation = ''
        scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true'
        isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a7'/>
        </Model:TypedElement.type>
      </Model:Attribute>
    </Model:Namespace.contents>
    <Model:GeneralizableElement.supertypes>
      <Model:Class xmi.idref = 'a95'/>
    </Model:GeneralizableElement.supertypes>
  </Model:Class>
  <Model:Class xmi.id = 'a95' name = 'DocumentedElement' annotation = '' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
    <Model:Namespace.contents>
      <Model:Attribute xmi.id = 'a96' name = 'Description' annotation = '' scope = 'instance_level'
        visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
        <Model:StructuralFeature.multiplicity>
          <XMI.field>0</XMI.field>
          <XMI.field>1</XMI.field>
          <XMI.field>false</XMI.field>
          <XMI.field>false</XMI.field>
        </Model:StructuralFeature.multiplicity>
        <Model:TypedElement.type>
          <Model:PrimitiveType xmi.idref = 'a7'/>
        </Model:TypedElement.type>
```

```
        </Model:Attribute>
      </Model:Namespace.contents>
      <Model:GeneralizableElement.supertypes>
        <Model:Class xmi.idref = 'a97'/>
      </Model:GeneralizableElement.supertypes>
    </Model:Class>
    <Model:Class xmi.id = 'a97' name = 'NamedElement' annotation = '' isRoot = 'false'
      isLeaf = 'false' isAbstract = 'true' visibility = 'public_vis' isSingleton = 'false'>
      <Model:Namespace.contents>
        <Model:Attribute xmi.id = 'a98' name = 'ElName' annotation = '' scope = 'instance_level'
          visibility = 'public_vis' isChangeable = 'true' isDerived = 'false'>
          <Model:StructuralFeature.multiplicity>
            <XMI.field>1</XMI.field>
            <XMI.field>1</XMI.field>
            <XMI.field>false</XMI.field>
            <XMI.field>false</XMI.field>
          </Model:StructuralFeature.multiplicity>
          <Model:TypedElement.type>
            <Model:PrimitiveType xmi.idref = 'a7'/>
          </Model:TypedElement.type>
        </Model:Attribute>
      </Model:Namespace.contents>
      <Model:GeneralizableElement.supertypes>
        <Model:Class xmi.idref = 'a21'/>
      </Model:GeneralizableElement.supertypes>
    </Model:Class>
    <Model:Import xmi.id = 'a99' name = 'PrimitiveTypes' annotation = '' visibility = 'public_vis'
      isClustered = 'false'>
      <Model:Import.importedNamespace>
        <Model:Package xmi.idref = 'a1'/>
      </Model:Import.importedNamespace>
    </Model:Import>
  </Model:Namespace.contents>
 </Model:Package>
</XMI.content>
</XMI>
```

5.4 Inheritance

In order to reduce management complexity and to improve performance Object Orientation inheritance is not allowed in SDL. The suggested way to extend a service is by a "*copy and modify" operation*. The SDL Editor offers the features to copy and versioning an SDL Model.

6.SDL Example

The following chapter aims to provide some examples useful to better understand the SDL structure by comparing different languages. It has been thought to make easier the comprehension of a such a new concept by bringing it back to already known similar concepts. For such a reason the following example here proposed is very simple and it is presented in both formats WSDL and Java, hoping that the reader will be familiar at least with one of these two languages.

6.1 TicketAgent

TicketAgent is a WSDL 1.2 example supplied by the W3C (http://www.w3.org/2002/02/21-WSDL-RDF-mapping/wsdl12ticketAgent.wsdl). This example eases the comparison between WSDL 2.0 and SDL 1.0, and SDL 2.0. Differences between SDL2.0 and SDL3.0, related to this example are illustrated in Figure 21 - TicketAgent SDL3.0 – Import.

6.1.1 WSDL Definition

In Figure 16 - TicketAgent WSDL the WSDL source for TicketAgent example.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TicketAgent" targetNamespace="http://airline.wsdl/ticketagent/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://airline.wsdl/ticketagent/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsd1="http://airline/">
    <import location="TicketAgent.xsd" namespace="http://airline/"/>
    <message name="listFlightsRequest">
        <part name="depart" type="xsd:dateTime"/>
        <part name="origin" type="xsd:string"/>
        <part name="destination" type="xsd:string"/>
    </message>
    <message name="listFlightsResponse">
        <part name="result" type="xsd1:ArrayOfString"/>
    </message>
    <message name="reserveFlightRequest">
        <part name="depart" type="xsd:dateTime"/>
        <part name="origin" type="xsd:string"/>
        <part name="destination" type="xsd:string"/>
        <part name="flight" type="xsd:string"/>
    </message>
    <message name="reserveFlightResponse">
        <part name="result" type="xsd:string"/>
    </message>
    <interface name="TicketAgent">
        <operation name="listFlights" parameterOrder="depart origin destination">
            <input message="tns:listFlightsRequest" name="listFlightsRequest"/>
            <output message="tns:listFlightsResponse" name="listFlightsResponse"/>
        </operation>
        <operation name="reserveFlight" parameterOrder="depart origin destination flight">
            <input message="tns:reserveFlightRequest" name="reserveFlightRequest"/>
            <output message="tns:reserveFlightResponse" name="reserveFlightResponse"/>
        </operation>
    </interface>
</definitions>
```

Figure 16 - TicketAgent WSDL

6.1.2 SDL Definition

The Figure 17 - TicketAgent SDL1.0 (XML SDL) shows the example in SDL 1.0 format.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://dbe.org/schemas/sdl01.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://dbe.org/schemas/sdl20.xsd
http://www.soluta.net/schemas/sdl01.xsd"
targetNamespace="http://dbe.org.examples/TicketAgent">
    <message name="listFlightsRequest">
        <sdl:part name="depart" type="dateTime"/>
        <sdl:part name="origin" type="string"/>
        <sdl:part name="destination" type="string"/>
    </message>
    <message name="listFlightsResponse">
        <sdl:part name="flight" type="string"/>
    </message>
    <message name="reserveFlightRequest">
        <sdl:ontologyid>http://link.to.ontology/concep1t</sdl:ontologyid>
        <sdl:part name="depart" type="dateTime"/>
        <sdl:part name="origin" type="string"/>
        <sdl:part name="destination" type="string"/>
        <sdl:part name="flight" type="string">
            <sdl:ontologyid>http://link.to.ontology/concept2</sdl:ontologyid>
        </sdl:part>
    </message>
    <message name="reserveFlightResponse">
        <sdl:part name="result" type="string"/>
    </message>
    <message name="reserveFlightResponseOutfault">
        <sdl:part name="errorResult" type="string"/>
        <sdl:part name="errorDescription" type="string"/>
    </message>
    <interface name="TicketAgent">
        <sdl:operation name="listFlights">
            <sdl:input name="listFlightsRequest"/>
            <sdl:output name="listFlightsResponse"/>
        </sdl:operation>
        <sdl:operation name="reserveFlight">
            <sdl:input name="reserveFlightRequest"/>
            <sdl:output name="reserveFlightResponse"/>
            <sdl:outfault name="reserveFlightResponseOutfault"/>
        </sdl:operation>
    </interface>
</definitions>
```

Figure 17 - TicketAgent SDL1.0 (XML SDL)

Note that the SDL source is really similar to WSDL for this simple example. The main difference is the presence of the ontology link. In the pictures Figure 18 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types – Part 1 and Figure 20 - TicketAgent SDL2.0 (XMI-SDL) - Interface the same example is defined in SDL 2.0. The differences with the WSDL are more conspicuous because the SDL1.0 was XML-SDL while the SDL2.0 is XMI-SDL. Refer to 5.3.1 SDL, XMI SDL and XML SDL for further information.

The SDL2.0 example is divided in two parts: the first in the Figure 18 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types – Part 1 shows the messages and the

types used, the second Figure 20 - TicketAgent SDL2.0 (XMI-SDL) - Interface shows the definition of the service. The SDL was split to ease the reading, the XMI-SDL is less “human readable” then XML-SDL but is MOF compliant.

```
<sdl:Definitions.cmpType>
 <sdl:SdlInteger xmi:type="sdl:SdlInteger" xmi.id="_0.@cmpType.0" ElName="Integer"/>
 <sdl:SdlReal xmi:type="sdl:SdlReal" xmi.id="_0.@cmpType.1" ElName="Real"/>
 <sdl:SdlString xmi:type="sdl:SdlString" xmi.id="_0.@cmpType.2" ElName="String"/>
 <sdl:SdlBoolean xmi:type="sdl:SdlBoolean" xmi.id="_0.@cmpType.3" ElName="Boolean"/>
 <sdl:SdlDateTime xmi:type="sdl:SdlDateTime" xmi.id="_0.@cmpType.4" ElName="Date Time"/>
 <sdl:SdlUri xmi:type="sdl:SdlUri" xmi.id="_0.@cmpType.5" ElName="Uri"/>
</sdl:Definitions.cmpType>
<sdl:Definitions.cmpMessageList>
 <sdl:MessageList xmi.id="_0.@cmpMessageList.0"
ElName="msgListReserveFlightResponseOutfault" cmpMessageList="_0.@cmpMessage.4"/>
</sdl:Definitions.cmpMessageList>
<sdl:Definitions.cmpMessage>
 <sdl:SimpleMessage xmi.id="_0.@cmpMessage.0" ElName="listFlightsRequest">
  <sdl:SimpleMessage.cmpPart>
   <sdl:Part xmi.id="_0.@cmpMessage.0.@cmpPart.0" ElName="depart">
    <sdl:Part.refPart>
     <sdl:SdlDateTime xmi.idref="_0.@cmpType.4"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.0.@cmpPart.1" ElName="origin">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.0.@cmpPart.2" ElName="destination">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
  </sdl:SimpleMessage.cmpPart>
 </sdl:SimpleMessage>
 <sdl:SimpleMessage xmi.id="_0.@cmpMessage.1" ElName="listFlightsResponse">
  <sdl:SimpleMessage.cmpPart>
   <sdl:Part xmi.id="_0.@cmpMessage.1.@cmpPart.0" ElName="flight">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
  </sdl:SimpleMessage.cmpPart>
 </sdl:SimpleMessage>
</sdl:Definitions.cmpMessage>
```

Figure 18 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types – Part 1

```
 <sdl:SimpleMessage xmi.id="_0.@cmpMessage.2" ElName="reserveFlightRequest"
ntologyReference="http://link.to.ontology/concept1">
  <sdl:SimpleMessage.cmpPart>
   <sdl:Part xmi.id="_0.@cmpMessage.2.@cmpPart.0" ElName="depart">
    <sdl:Part.refPart>
     <sdl:SdlDateTime xmi.idref="_0.@cmpType.4"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.2.@cmpPart.1" ElName="origin">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.2.@cmpPart.2" ElName="destination">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.2.@cmpPart.3" ElName="flight"
ontologyReference="http://link.to.ontology/concept2">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
  </sdl:SimpleMessage.cmpPart>
 </sdl:SimpleMessage>
 <sdl:SimpleMessage xmi.id="_0.@cmpMessage.3" ElName="reserveFlightResponse">
  <sdl:SimpleMessage.cmpPart>
   <sdl:Part xmi.id="_0.@cmpMessage.3.@cmpPart.0" ElName="result">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
  </sdl:SimpleMessage.cmpPart>
 </sdl:SimpleMessage>
 <sdl:SimpleMessage xmi.id="_0.@cmpMessage.4" ElName="reserveFlightResponseOutfault">
  <sdl:SimpleMessage.cmpPart>
   <sdl:Part xmi.id="_0.@cmpMessage.4.@cmpPart.0" ElName="errorResult">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
   <sdl:Part xmi.id="_0.@cmpMessage.4.@cmpPart.1" ElName="errorDescription">
    <sdl:Part.refPart>
     <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
    </sdl:Part.refPart>
   </sdl:Part>
  </sdl:SimpleMessage.cmpPart>
 </sdl:SimpleMessage>
```

Figure 19 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types – Part 2

```
<sdl:Definitions.cmpImportDef>
 <sdl:Import xmi.id="_0.@cmpImportDef.0" nsPrefix="myt" namespace="org.dbe.myTravelAlliance">
  <sdl:Import.cmpImportedImp>
   <sdl:SimpleMessage xmi:type="sdl:SimpleMessage" xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.0"
     ElName="listFlightsRequest">
    <sdl:SimpleMessage.cmpPart>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.0.@cmpPart.0" ElName="depart">
      <sdl:Part.refPart>
       <sdl:SdlDateTime xmi.idref="_0.@cmpType.4"/>
      </sdl:Part.refPart>
     </sdl:Part>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.0.@cmpPart.1" ElName="origin">
      <sdl:Part.refPart>
       <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
      </sdl:Part.refPart>
     </sdl:Part>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.0.@cmpPart.2" ElName="destination">
      <sdl:Part.refPart>
       <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
      </sdl:Part.refPart>
     </sdl:Part>
    </sdl:SimpleMessage.cmpPart>
   </sdl:SimpleMessage>
   <sdl:SimpleMessage xmi:type="sdl:SimpleMessage" xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.1"
     ElName="reserveFlightRequest" ontologyReference="http://link.to.ontology/concept1">
    <sdl:SimpleMessage.cmpPart>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.1.@cmpPart.0" ElName="depart">
      <sdl:Part.refPart>
       <sdl:SdlDateTime xmi.idref="_0.@cmpType.4"/>
      </sdl:Part.refPart>
     </sdl:Part>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.1.@cmpPart.1" ElName="origin">
      <sdl:Part.refPart>
       <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
      </sdl:Part.refPart>
     </sdl:Part>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.1.@cmpPart.2" ElName="destination">
      <sdl:Part.refPart>
       <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
      </sdl:Part.refPart>
     </sdl:Part>
     <sdl:Part xmi.id="_0.@cmpImportDef.0.@cmpImportedImp.1.@cmpPart.3" ElName="flight"
ontologyReference="http://link.to.ontology/concept2">
      <sdl:Part.refPart>
       <sdl:SdlString xmi.idref="_0.@cmpType.2"/>
      </sdl:Part.refPart>
     </sdl:Part>
    </sdl:SimpleMessage.cmpPart>
   </sdl:SimpleMessage>
  </sdl:Import.cmpImportedImp>
 </sdl:Import>
</sdl:Definitions.cmpImportDef>
.....
```

Figure 21 - TicketAgent SDL3.0 – Import

In Figure 21 - TicketAgent SDL3.0 – Import, above all the messages used in the TicketAgent example are not defined in the TicketAgent SDL model but imported from other SDL models that contain all the messages and types used in a specific domain. In that way the creation of the SDL model is simplified, as well as model reuse and interoperability are promoted.

6.1.3 Java Interface

Figure 22 - TicketAgent Java shows a Java representation of the TicketAgent service. The Java Interface may be generated automatically, in this example it is used only to better define the service in a way to be easily grasped by a wide range of readers. The DBE Java Interface may fundamentally differ from this, for instance it will be decided to use XML parameters and not a Java type, in this way all the Java methods will have an XML Document as parameter. Refers to 7.2 SDL to Java to better understand the difference between Java Interface and DBE Java Interface.

```
package dbe.org.examples;
import java.lang.*;
import java.util.*;

interface listFlightsRequest {
        public Date getDepart();
        public void setDepart(Date depart);
        public String getOrigin();
        public void setOrigin(String origin);
        public String getDestination();
        public void setDestination(String destination);
}
interface listFlightsResponse {
        public String[] getResult();
        public void setResult(String[] result);
}
interface reserveFlightRequest {
        public Date getDepart();
        public void setDepart(Date depart);
        public String getOrigin();
        public void setOrigin(String origin);
        public String getDestination();
        public void setDestination(String destination);
        public String getFlight();
        public void setFlight(String flight);
}
interface reserveFlightResponse {
        public String getResult();
        public void setResult(String result);
}

interface TicketAgent {
        public listFlightsResponse listFlights (listFlightsRequest p1);
        public reserveFlightResponse reserveFlight(reserveFlightRequest p1);
}
```

Figure 22 - TicketAgent Java

6.1.4 SDL Editor Snapshot

The Figure 23 - TicketAgent SDL Editor Snapshot presents an SDL Editor snapshot of the TicketAgent service. The main frame (TicketAgent.sdl) presents the structure of the SDL with Definitions, Interface, Operations and Messages. The Operation ListFlight has been selected and its Properties are shown in the frame at the bottom of the window. The SDL Editor uses a tree based interface.

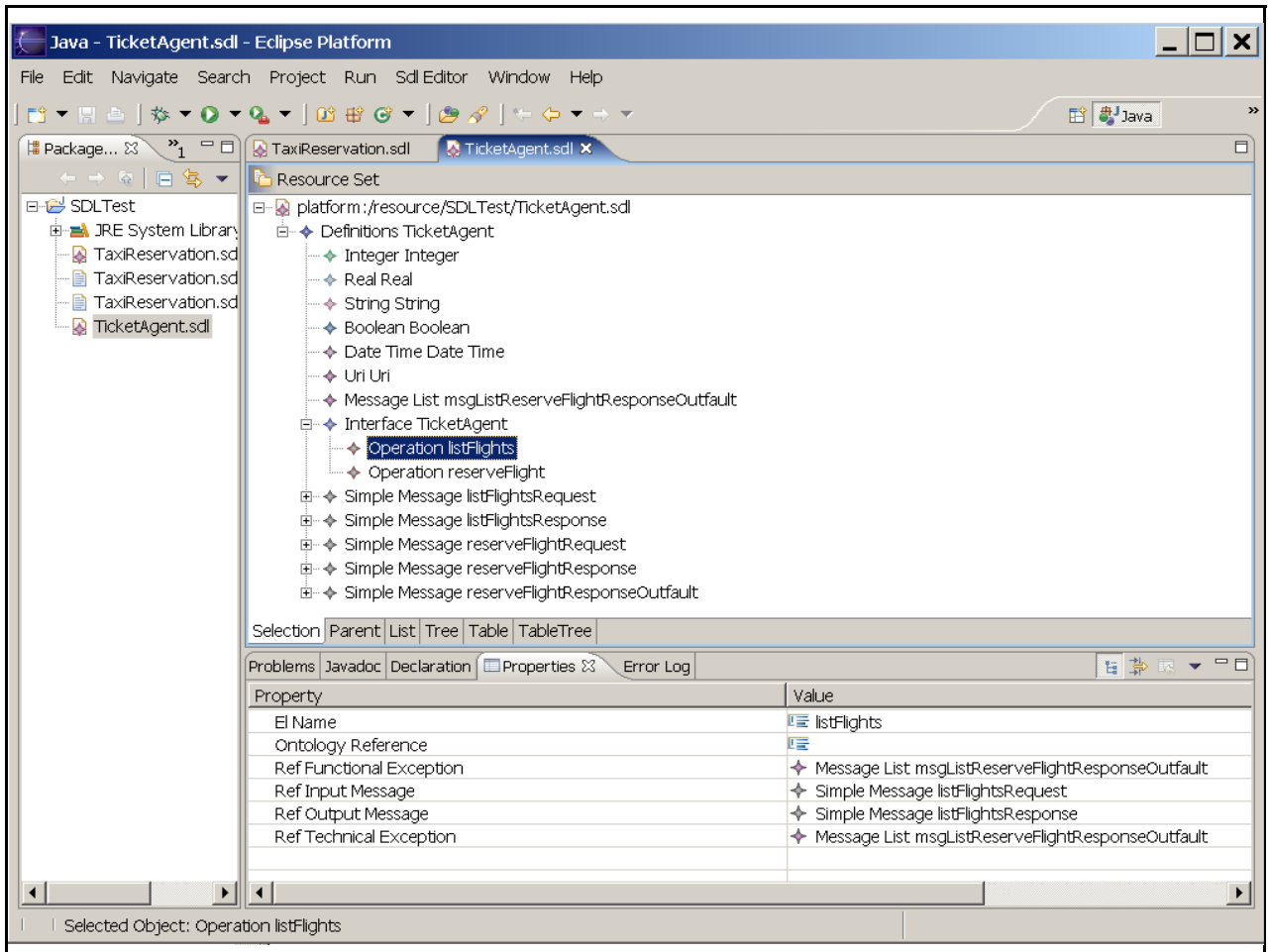

Figure 23 - TicketAgent SDL Editor Snapshot

7.Transformations Rules based on SDL

7.1 Transformation overview

The following chapter offers a general view of the transformation rules that involve the SDL, also explaining their meaning and the transformation guidelines followed.
Transformation rules have been introduced with the main purpose of facilitating the creation of services in DBE. The easiest way to publish a service in DBE is to copy a pre-existing Service Manifest and then to customize it, usually only modifying the BML Data (we not considering here the effort of exposing the SME back office legacy service).
On the contrary, if a Service Manifest has to be created from scratch all the operations would be heavier and longer. In other words, such transformations try to automatize as much as possible the editing of a service to be distributed within DBE. For example, some information contained in the SSL can be useful to define the SDL. It will not be possible to entirely generate the SDL, but according to the Pareto's principle<add wikipedia reference>, it will be very likely that the 80% of SDL can be automatically generated. Similar situations are related to the implementation of services: generating a Java infrastructure of services starting from the SDL allows time saving in codify, ignoring technical architectural details of DBE and thus producing a less error prone code.

The following paragraphs illustrate the transformation rules from SDL into other languages and from some other ones into SDL, as follows:

- SSL to SDL
- SDL to Java
- SDL to WSDL
- WSDL to SDL

The focuses on the part of the process related to the service creation which involves the SDL.
The BML Editor support the creation of BML models which also contains the SSL. Then, using the SSL the Compiler generates the SDL which can be used as it is or, after some adaptations made with the SDL Editor. Once defined the final version of the SDL, it is possible to generate both the WSDL and the Java code structure implementing the service. The Java classes generated will contain the code producing all the kind of data used and the structure of the control classes. The behaviour of methods, however, should be written by hand. The BML and SDL models could be used by the SMEditor to create, as a simply operation, the SM. The java code, suitably integrated, will be used to distribute the service. SSL to SDL

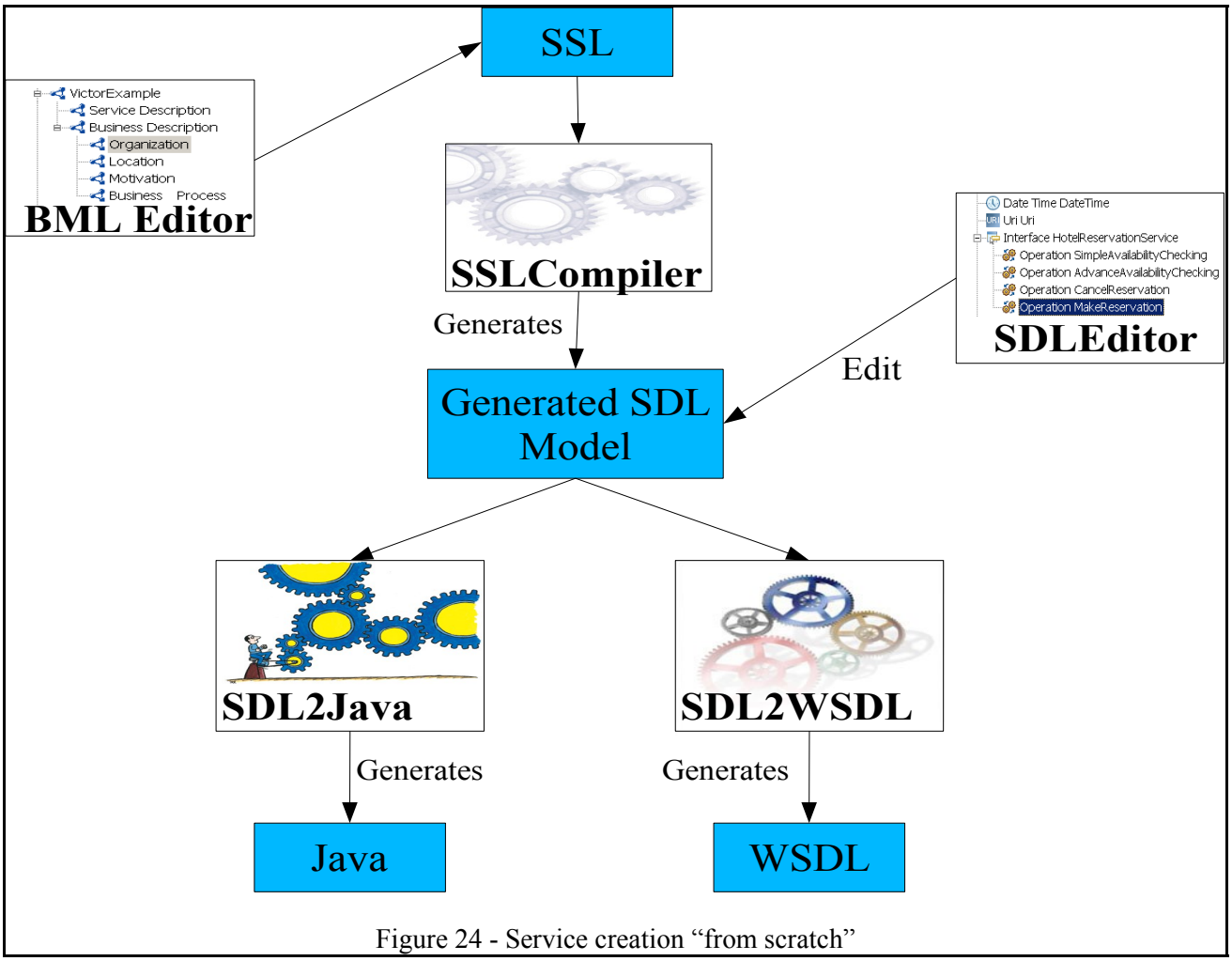

Figure 24 - Service creation "from scratch"

7.1.1 Introduction

The SSL (Semantic Service Language) is a metamodel that can be used to consistently define service description models in a business related matter, hence avoiding computational terms[25]. The SSL is integrated with the Ontology Defintion Metamodel (ODM)[26] in order to allow SSL models to make use of concepts defined in domain specific ontologies increasing the semantic interoperability among SMEs. Refers to [Knowledge Representation Models] and to [SSL Compiler] for further informations.

The SSL and SDL standards are mutually related since a technical interface for a certain service can be partially obtained from its semantic description.
The SSL to SDL transformation converts an SSL model into an SDL model also using an ODM model to include the necessary data from an ontology into the

25 *"The Semantic Service Language (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 outside world (the candidate service consumers)"* [Deliverable D14.1].

26 The ODM is a metamodel developed in the DBE project that can be used for defining business and service ontologies.

technical specification of the service. SSL and SDL are somehow similar for the fact that both describe a service, but SSL describes the semantics of the service (what the service does) while the SDL describes the technical interfaces of the service (how the service must be invoked). ODM is useful, for example, to indicate that a service has a “Customer” as input, while the SDL 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.
This paragraph does not provide a detailed explanation of the SSL or the ODM as it is supposed the reader is enough acknowledged on that. Further details can be found in [Deliverable D14.1]

In SSL, 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, and has the type SSL TypeURI.
As it can be seen in Figure 25 - SSL Metamodel - Type referencing scheme[27], the class SSL TypeURI is extended only by SSL DataRangeURI and OntologyClassURI, and DataRangeURI is in turn extended by EnumTypeURI and DataTypeURI. At the moment, enumerated types cannot be defined outside ontologies, so the enumerated type SSL EnumTypeURI is not considered, the types considered for this transformation being SSL OntologyClassURI and SSL DataTypeURI.

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

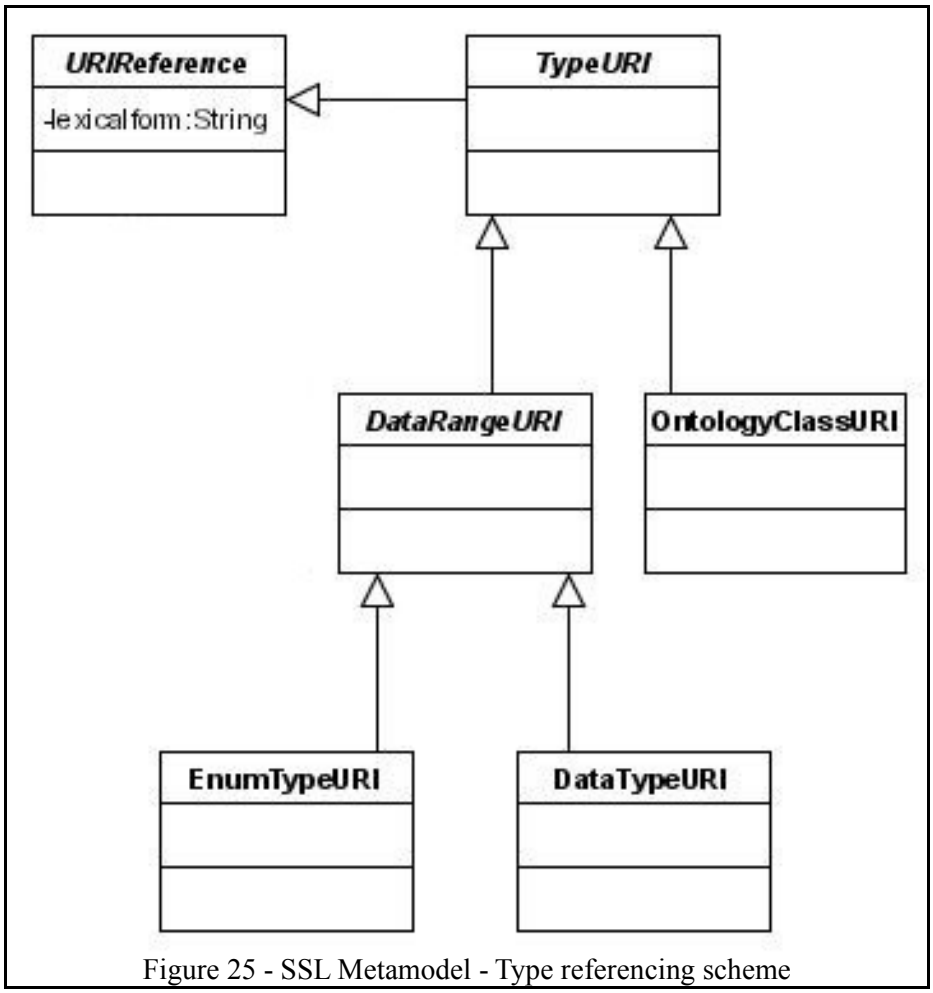

Figure 25 - SSL Metamodel - Type referencing scheme

In ODM, the types of the described classes are specified using ODM DatatypeProperty elements; each of these has a domain, that indicates that classes that have the current property, and an ODM DataRange element that specifies its type. This structure is presented in Figure 26 - ODM Metamodel - Properties in ODM[27] here below, that illustrates the following:

- 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.

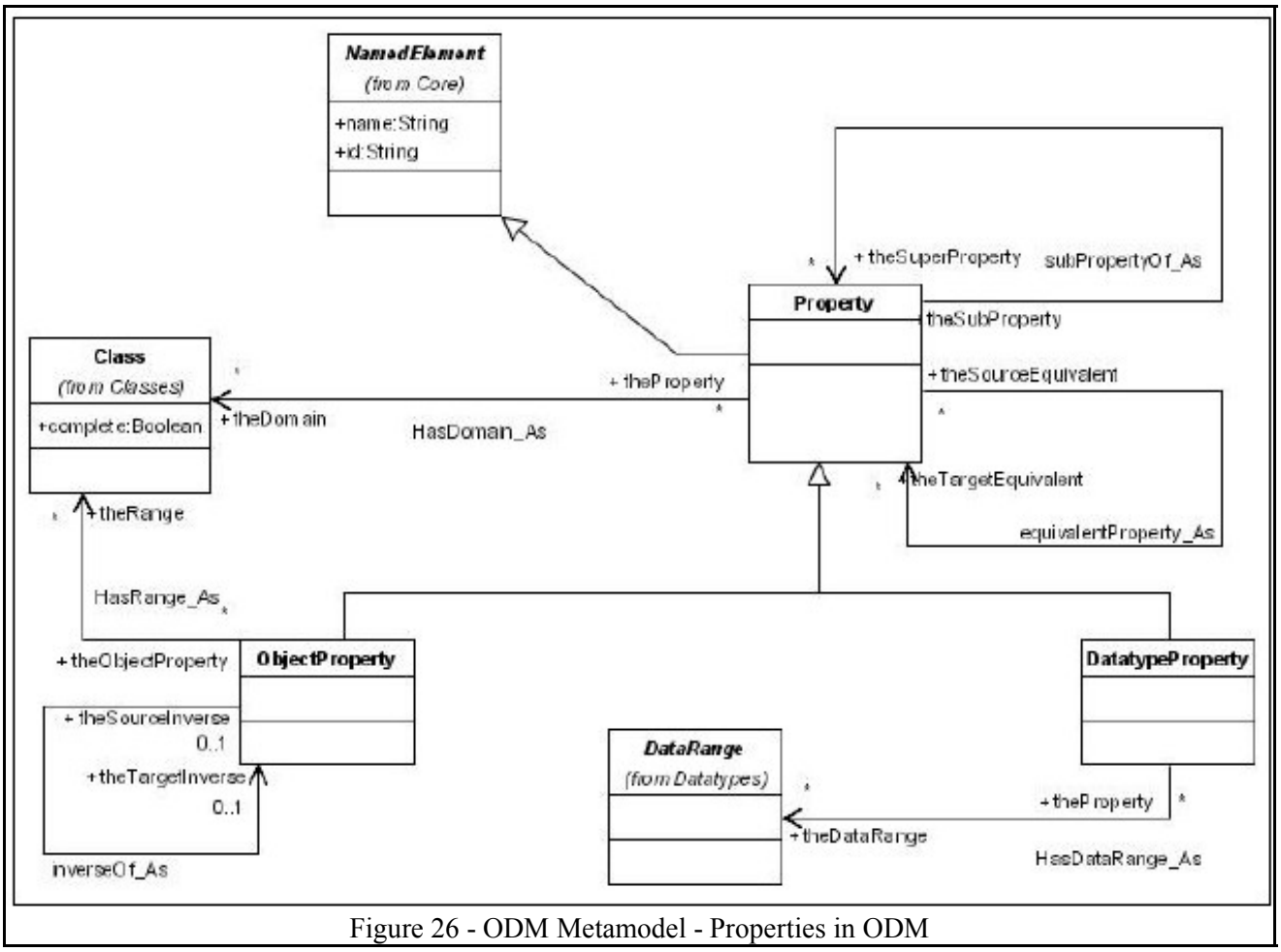

Figure 26 - ODM Metamodel - Properties in ODM

7.1.2 Transformation rules

A schematic description of the transformation between SSL and SDL elements is presented in Table 1 - SSL to SDL transformation, below.

SSL	*SDL*
ServiceProfile	Definitions
ServiceFunctionality	Operation
ServiceInput/ServiceOutput	Simple Message
DataTypeURI	Part
OntologyClassURI	ComplexType

Table 1 - SSL to SDL transformation

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 Interfaces (an 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 SimpleMessages.

From each SSL ServiceFunctionality an SDL Operation is created.
As an 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 an 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 an 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 an SSL ServiceFunctionality, an SDL SimpleMessage is constructed.
From each SSL ServiceOutput that is the only output of a SSL ServiceFunctionality, an SDL SimpleMessage is constructed.
From each SSL DataTypeURI (that is pointing to an XML Schema 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.
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 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 an SSL ServiceInput included in a SSL ServiceFunctionality with multiple inputs or if the SSL OntologyClassURI is the Type of an SSL ServiceOutput included in an SSL ServiceFunctionality with multiple outputs, then an 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 transformation.

This transformation is realized using ATL, a metamodel transformation language that responses to the OMG's QVT request for proposals[28]. It is possible to define the transformation at the level of metamodels because all the involved languages are defined using an MDA approach and, therefore, expressed using MOF.
A peculiar feature about this transformation is that it effectively uses two input metamodels. This means that when it is applied it creates the output SDL model by

28 Query / Views / Transformations Request for proposals - http://www.omg.org/docs/ad/02-04-10.pdf

combining, in a non-trivial way, the information from an SSL model and an ODM model. This is interesting since all the metamodel transformations that have been published so far in QVT communities are one-to-one: creating the output from only one input model.
The SSL to SDL transformation was published as a sample ATL usage containing the transformation code and explanatory documentetation on the site of the Eclipse Generative Modelling Tools project[29], as a highly appreciated contribution to the ATL community.

7.2 SDL to Java

7.2.1 Introduction

This transformation has as input an SDL model. From this model a set of Java classes are created and interfaces that are the skeleton for implementing a DBE service are defined by the SDL.
As mentioned in the introduction, the Java code generated can be divided into two parts:

- Java Interfaces and Java Types
- The adapter for the proxy

29 http://www.eclipse.org/gmt/atl/atlTransformations/#SSL2SDL

```
/** ************************************** */
/** Generated with SDL2Java Compiler    */
/** ************************************** */
package org.dbe.demos.ticketagent;
import org.dbe.servent.Adapter;
import org.dbe.servent.ServiceContext;
import javax.xml.rpc.holders.*;
import java.util.*;
import java.math.*;

/* Here packages to be imported.
 You should add to Java Build Path Libraries the servent-core.jar */

public class TicketAgentAdapter implements TicketAgent ,Adapter {
/*---------Service Methods--------*/
        public void listFlights( Calendar depart,String origin,String destination,StringHolder flight)
                throws java.rmi.RemoteException {
                // TO DO  write your code here
        };

        public void reserveFlight( Calendar depart,String origin,String destination,String flight,StringHolder result)
                throws java.rmi.RemoteException {
                // TO DO write your code here
        };

/*---------Adapter Methods--------*/
        public void init (ServiceContext context) {
                // TO DO write your code here
        }
        public void destroy () {
                // TO DO write your code here
        }
}
```

Figure 27 - Code generated for the Ticket Agent Example

The transformation is generated using the MDA pattern for PIM trasformation (further reference can be found at [MDA Guide ver 1.0.1]) illustrated in Figure 28 - A platform independent model transformed.

TicketAgentAdapter is a clear demonstration of the fact that the Java code generated is not simply the service Java interface but it also contains the DBE architecture. In this case the platform is DBE, because the code generated depends on the DBE platform and not only on the Java platform.

As illustrated in Figure 28 - A platform independent model transformed, to execute the transformation from PIM to PSM some information must be added. In this case, the added information represents the DBE architecture.

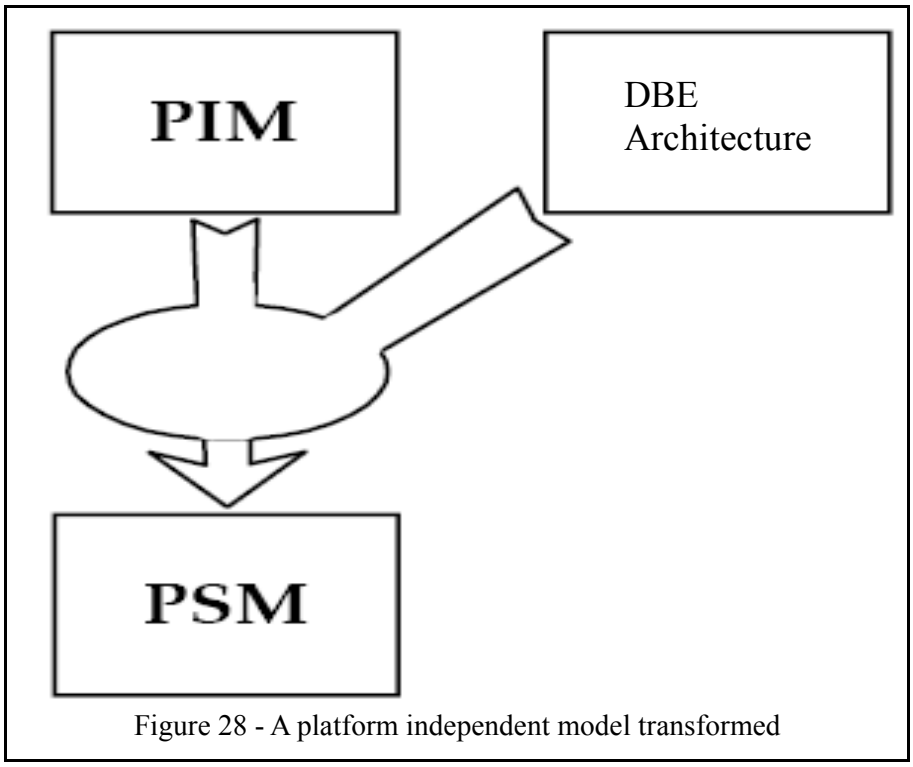

Figure 28 - A platform independent model transformed

7.2.2 Transformation rules

The main correspondences between SDL and Java elements are shown in Table 2 - SDL to Java transformation, below.

SDL	***JAVA***
Interface	interface
Operation	public void method
SimpleMessage -> Part	method parameters
ComplexType	class
Part (of ComplexType)	set and get methods
sdl<Type>	primitive type of parameters

Table 2 - SDL to Java transformation

- For each SDL Interface a Java interface is created, having as name the SDL Interface ElName followed by the string “Service”;
- For each SDL Operation a public void Java method for the interface is generated; it declares that it can throw a java.rmi.RemoteException;
- For each SDL SimpleMessage the corresponding method parameters are generated. These parameters are the SDL Part of a SDL SimpleMessage. The <Type> of the parameters is the type of the SDL Part.

The structure of the generated Java interface is shown in Figure 29 - Generated Java interface here below.

Also, from the SDL Interface, a public Java class that implements both the generated Java interface and the class org.dbe.servent.Adapter is generated. This

```
public interface <SDL Interface ElName>Service {
        public void <SDL Operation ElName> (<Type> <SDL Part ElName>, <Type> <SDL Part ElName>, ...)
                throws java.rmi.RemoteException;
        ...
}
```

Figure 29 - Generated Java interface

class has all the Java interface methods and the methods of the org.dbe.servent.Adapter class: init(ServiceContext context) and destroy(). The name of this class is the value of the attribute SDL Interface ElName, followed by the string “Adapter”.
The structure of the generated Java class is shown in Figure 30 - Generated Java class for complex type, below.

- For each SDL ComplexType a Java class is generated, having as name the value of the ElName attribute of the SDL ComplexType element. It implements java.io.Serializable. The structure of this class is shown in Figure 30 - Generated Java class for complex type here below;
- For each SDL Part of a SDL ComplexType a pair of set and get methods is generated.

```
public class <SDL ComplexType ElName> implements java.io.Serializable {
        private <Type> <SDL Part ElName>;
        ...
        public void set<SDL Part ElName>(<Type> <SDL Part ElName>) {
                this.<SDL Part ElName> = <SDL Part ElName>;
        }
        public <Type> get<SDL Part ElName>() {
                return <SDL Part ElName>;
        }
...
}
```

Figure 30 - Generated Java class for complex type

The SDL primitive types are transformed according to the following rules:

- SdlDateTime becomes Calendar in Java
- SdlReal type becomes BigDecimal in Java
- SdlURI type becomes String in Java
- SdlInteger type becomes BigInteger in Java
- SdlBoolean type becomes Boolean in Java
- SdlString type becomes String in Java
- if an SDL Part has the attribute isArray equal to true then the obtained type in Java is also an array (for example, if an SDL Part has the type SdlString and the isArray attribute true, it becomes String[] in Java). Otherwise, it is the correspondent of the SDL Part's type, using the type transformation rules.

7.3 SDL to WSDL

7.3.1 Introduction

The SDL to WSDL transformation has as input an SDL model and it creates a file that describes that service in the Web Services Definition Language (WSDL).

7.3.2 Transformation rules

The corresponding WSDL elements for the main SDL elements are shown in Table 3 - SDL to WSDL transformation, below.

SDL	***WSDL***
Definitions	definitions+service
Interface	PortType
Interface	binding
Interface	port (child of WSDL service)
Operation	operation (for WSDL portType and for WSDL binding)
Operation.refOutputMessage (the referred SimpleMessage)	output (for both WSDL portType and WSDL binding)
Operation.refInputMessage (the referred SimpleMessage)	input (for both WSDL portType and WSDL binding)
Operation.refTechnicalException (the referred MessageList)	fault (for both WSDL portType and WSDL binding)
Operation.refFunctionalException (the referred MessageList)	fault (for both WSDL portType and WSDL binding)
SimpleMessage	message
MessageList	message
Part (of a SimpleMessage)	part
ComplexType	complexType and its child element sequence
Part (of a ComplexType)	element
Sdl<Type> - for all SimpleType	the type attribute of a WSDL element or of a WSDL part

Table 3 - SDL to WSDL transformation

- For each SDL Definitions element, this transformation creates both a WSDL definition and a WSDL service element. The name of the WSDL service element is the value of the ElName attribute of SDL Definitions.
- For each SDL Interface a WSDL portType is generated, together with a WSDL binding and a WSDL port. The WSDL portType name attribute is the value of the SDL Interface ElName attribute. The WSDL binding name attribute is the value of the SDL Interface ElName attribute concatenated with the string “SoapBinding”. The WSDL port name attribute is the value of the SDL Interface ElName attribute and the binding attribute has the same value concatenated with the string “SoapBinding”.
- For each SDL Operation a WSDL operation is created as a child element of the WSDL portType element; also a WSDL operation is created as a child element of the WSDL binding element.
- For each SDL Operation.refOutputMessage, two WSDL output elements are created, one as the child of WSDL portType and one as the child of WSDL binding. The name attribute of each WSDL output has the value of the referenced SDL SimpleMessage ElName. The message attribute of the WSDL output element that is generated as a child of the WSDL portType element has the following value: the string “impl:” concatenated with the referenced SDL SimpleMessage ElName.
- For each SDL Operation.refInputMessage, two WSDL input elements are created, with the same rules as for SDL Operation.refOutputMessage.
- For each SDL Operation.refTechnicalException and SDL Operation.refFunctionalException two WSDL fault elements are created, each of them being the child of one of the corresponding WSDL operation elements. The name attribute of the WSDL fault has the value of the referenced SDL MessageList ElName. The message attribute of the WSDL fault has the following value: the string “impl:” concatenated with the referenced SDL MessageList ElName.
- For each SDL SimpleMessage a WSDL message is generated. The WSDL message has the same name as the SDL SimpleMessage.
- For each SDL MessageList a WSDL message is generated. The WSDL message has the same name as the SDL MessageList.
- For each SDL Part that is the child of an SDL SimpleMessage, a WSDL part element is generated as the child of the corresponding WSDL message. This WSDL part has the same name as the SDL part and has also an attribute named type which is generated from the SDL Sdl<Type> element (where <Type> could be String, Boolean, DateTime, Real, Integer, URI or ComplexType).
- For each SDL ComplexType a WSDL complexType and its child WSDL sequence element are generated. The name of the WSDL complexType is the same as the name of the SDL ComplexType.

The types transformation follows the rules listed bellow:

- SdlString becomes xsd:string
- SdlInteger becomes xsd:integer
- SdlReal becomes xsd:decimal

- SdlBoolean becomes xsd:boolean
- SdlUri becomes xsd:string
- SdlDateTime becomes xsd:dateTime
- SdlComplexType becomes a reference to the generated WSDL complexType.

- For each SDL Part that is the child of an SDL ComplexType, a WSDL element is generated as the child of the WSDL sequence. This WSDL element has the same name as the SDL part and an attribute named type which is generated from an SDL Sdl<Type> element (where <Type> could be String, Boolean, DateTime, Real, Integer, URI or ComplexType), using the types transformation rules that are listed above.

7.4 WSDL to SDL

7.4.1 Introduction

This transformation, sometimes referred to as “WSDL reverse engineering” has as an input a file that describes a service using the WSDL. Based on this description, it creates an SDL model.
This reverse is particularly useful to whom has implemented the service and exposed it as a web service or described it using WSDL. It allows to quickly create the SDL and the Java structure without rewriting it from scratch.

7.4.2 Transformation rules

Table 4 - WSDL to SDL transformation, below presents the main correspondences between WSDL and SDL elements.

WSDL	*SDL*
definitions	Definitions
portType	Interface
message	SimpleMessage
operation	Operation and its children: Operation.refInputMessage, Operation.refOutputMessage, Operation.refFunctionalException
operation	MessageList
part	Part (of a SimpleMessage)
element	Part (of a ComplexType)
complexType	ComplexType
input	SimpleMessage child of Operation.refInputMessage
output	SimpleMessage child of Operation.refOutputMessage
fault	MessageList child of Operation.refTechnicalException

Table 4 - WSDL to SDL transformation

- For each WSDL definition an SDL Definition is generated. The attribute ElName of the SDL Definitions has the value of the attribute name of the WSDL Service.
- For each WSDL portType an SDL Interface is generated.
- For each WSDL operation an SDL Operation is generated together with its children elements: Operation.refInputMessage, Operation.refOutputMessage and Operation.refFunctionalException.
- For each WSDL operation an SDL MessageList is generated, having as name the concatenation of all its the WSDL fault elements.
- For each WSDL input an SDL SimpleMessage is generated as a child of an SDL Operation.refInputMessage.
- For each WSDL output an SDL SimpleMessage is generated as the child of an SDL Operation.refOutputMessage.
- For each WSDL complexType an SDL ComplexType is generated.
- For each WSDL element an SDL Part is generated, this SDL Part being the child of an SDL ComplexType.
- For each WSDL message an SDL SimpleMessage is generated.
- For each WSDL part an SDL Part is generated as the child of an SDL SimpleMessage.

The rules for transforming the type attribute of a WSDL element into an SDL Sdl<Type> are the following:

- string becomes SdlString
- integer becomes SdlInteger
- int becomes SdlInteger
- long becomes SdlInteger
- base64Binary becomes SdlInteger
- decimal becomes SdlReal
- float becomes SdlReal
- double becomes SdlReal
- real becomes SdlReal
- boolean becomes SdlBoolean
- uri becomes SdlURI
- anyUri becomes SdlUri
- dateTime becomes SdlDateTime
- anyType becomes SdlString.

In all the other cases Sdl:ComplexType is used.
Also to be mentioned, the inheritance present in WSDL (for the ComplexType) is implemented in this transformation (WSDL to SDL) in this way: the corresponding SDL Complex Type will include also the elements inherited in WSDL from the base class, but there will be no reference to this base class; it is used a classic roll-down mechanism which implies that all the methods/attributes present in the father's class are added into the child's class. The user will not know about the base class. He will just see the inherited fields as parts of the SDL Complex Type. The cross-compilation process is particularly interesting for the overload and override mechanisms.

8.Glossary

Term	*Acronym*	*Synonymous*	*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.
Client			A Client is a software that makes use of a Web Service, acting as its 'user' or 'customer'.
CIM			The Computational Independent Model is a viewpoint of a system which focuses on the environment and the requirements for the system, but does not show details of the structure of the system.
Common Object Request Broker Architecture	CORBA		Is an OMG open architecture and infrastructure that applications can use to work together over networks.
DBE Knowledge Representation Framework			DBE Knowledge Representation Framework: The Framework of the interconnected DBE Knowledge Representation Models and/or Languages that capture the knowledge of the ecosystem.
ECore			Native metamodel format for the Eclipse Modeling Framework (EMF) http://www.eclipse.org/emf. It is generally interchangeable with the Essential MOF (EMOF) compliance level of MOF2
Interface		Port Type	A logical grouping of operations. An Interface represents an abstract Service type, independent of transmission protocol and data format. Describes sequences of messages that a service sends and/or receives.
Interface operation		Operation	Describes an operation that a given interface supports, it is an interaction with the service consisting of a set (ordinary and fault) messages exchanged between the service and the client.
Message			A Message is the basic unit of communication between a Service and a

Term	*Acronym*	*Synonymous*	*Description*
			Client; data to be communicated to or from a Service as a single logical transmission.
Meta Object Facility	MOF		A generalized facility and for specifying abstract information about very concrete object systems.
Model Driven Architecture	MDA		An approach (proposed by OMG) to IT system specification that separates the specification of system functionality for the specification of the implementation of that functionality on a specific technology.
ODM			Ontology Definition Metamodel: A MOF model (metamodel) developed in DBE for ontology representation.
Part Component		Message Part	A Part Component is the elementary part of a Message Component.
Platform Independent Model	PIM		The Platform Independent Model is a viewpoint of a system which focuses on the information about the specific technology that is used in the realization of a particular platform. A PIM combines the specifications in the PIM with details that specify how that system uses a particular type of platform.
Platform Specific Model	PSM		The Platform Specific Model is a viewpoint of a system which focuses on the operation a system while hiding the details necessary for a particular platform. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platform of similar type.
Query Views Transformation	QVT		OMG standard for Query, View and Transform models
Semantic Service Language	SSL		Semantic Service Language: A MOF-based language for semantically describing SME services in DBE.
Semantic Registry	SR		It is the component of the DBE Knowledge Base that hosts the service description published in the DBE environment and available for discovery and consumption.
Service Manifest	SM		All the specifications that describe an instance of a real service from the computing and business viewpoint.
Simple Object	SOAP		It is a lightweight protocol for exchanging

Term	Acronym	Synonymous	Description
Access Protocol			information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols.
Virtual Platform			The concept of Virtual Platform is close to the concept of Virtual Machine. As a virtual machine represents a group of physical machine, a Virtual Platform represents the common features of a group of different Platform.
Unified Modellng Language	UML		A method for specifying, visualizing, and documenting the artefacts of an object-oriented system under development; as well as for business modelling.
XML Metadata Interchange	XMI		XML Metadata Interchange: An SMIF (see SMIF description) standard specification based on XML.
XSD		XML Schema	XML Schema Definition, a W3C Recommendation, specifies how to formally describe the elements in an XML document.
Web Service	WS		A Web Service is a software application identified by an URI, whose interfaces and bindings are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.
Web Services Definition Language	WSDL		Is an XML based language for describing network services.

9.References

CORBA: CORBA® BASICS, http://www.omg.org/gettingstarted/corbafaq.htm
DBE 2nd Review Report: DG INFSO - EC, DBE Consolidated Second Review Report, 2006
Deliverable D14.1: TUC, DBE Knowledge Representation Models,
HMAD: David Carlson, HyperModel analysis and design tool,
http://www.xmlmodeling.com/hyperModel/
IDL: OMG IDL, http://www.omg.org/gettingstarted/omg_idl.htm
Knowledge Representation Models: , D.14.1: DBE Knowledge Representation Models, April 2005
MDA: David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, January 10, 2003
MDA Guide ver 1.0.1: , MDA Guide ver 1.0.1, 2003-06-01
MOF: Meta-Object Facility,
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF
SOAP: Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap/
SSL Compiler: , D.16.4 SSL Compler,
WSDL: Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl
XMI: XML Metadata Interchange (XMI), http://www.omg.org/technology/documents/formal/xmi.htm
XML: Extensible Markup Language (XML), http://www.w3.org/XML/
XMLS: XML Schema, http://www.w3.org/XML/Schema

- end of document