D.B.E.

Digital Business Ecosystem

Contract n° 507953

WP16: Service Description Language
D16.1: Service description models and language definition

Part 1 -SDL Definition

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

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D16.1

Due date: 30th April 2005

Delivery Date: 14th October 2005

Short Description:

This document describes the Service Description Language.

Partners owning: Soluta.net

Partners contributed: Soluta.net

Made available to: DBE Consortium

Versioning		
Version	Date	Name, organisation
00.01	2004/05/26	Giulio Montanari, Valentino Trentin – Soluta.net
00.02	2005/02/18	Giulio Montanari – Soluta.net
00.03	2005/06/23	Giulio Montanari, Andy Neagu, Pierfranco Ferronato – Soluta.net
00.04	2005/08/31	Giulio Montanari – Soluta.net
01.00	2005/10/14	Giulio Montanari – Soluta.net

Quality check

1st Internal Reviewer: Miguel Vidal – SUN Microsystems

2nd Internal Reviewer: Angelo Corallo - ISUFI

Abstract

It concerns with defining the Service Description Language (SDL). It specifies the capabilities, interactions and dependencies of DBE services from the technical point of view. The SDL has to participate in enabling the self-organizational feature of Services, for this purpose the SDL has to support semantic descriptions.
This first version focuses on the services capability specification not on interaction and dependency between services.
The deliverable is released in two parts:

- SDL Definitions
- Service Manifest: Conceptual Model & Software Model

Contents

Figures

1. Executive Summary

The SDL (Service Description Language) is a language for describing services in a platform independent way, where for platform in this context we specifically mean middleware. In DBE a service is described from two different viewpoints: the business and the technical. While the BML (Business Modelling Language) describes the service from a business point of view, the SDL describes the technical interface of the service: By technical interface we mean the low-level/programmatic interface that a service offers to all the consumers. According to the MDA[1] approach, adopted by the DBE, the SDL is defined using a XMI[2] 1.2 MOF[3] 1.4 metamodel. The use of a MOF metamodel allows the use of a MDA tool for the automatically generation of the SDL Editor.
The SDL is similar to the WSDL[4] but in DBE is needed a description language that enables the automatic composition of services. 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, for instance the MessageList was introduced to promote the reuse in modelling and ease the composition of services.

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

2.Other languages

The first step in the SDL definition was the investigation of the existing, platform dependent, languages to describe services. For the purpose of this document the amplitude of the chapter about the WSDL and IDL might seem excessive but it represents the investigation over the existing standards and a quick reference for people involved in SDL Definition. For these reasons in CORBA IDL there is also a brief introduction to CORBA.

2.1 WSDL[5]

The Web Services Definition Language or WSDL is an XML based language for describing network services. A WSDL document is written in XML, and it describes a Web Service interface. It contains the definitions of *services* or *ports* as collections of network endpoints[6]. To allow the reuse, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. *Messages* represent the abstract descriptions of the data being exchanged, and *port types* represent abstract collection of *operations*. The concrete protocol and data format specifications for a particular port type form a reusable *binding*. A collection of ports which specifies a *service* and a *port* is defined by associating a network address with a reusable binding.

A WSDL document makes use of the following elements in the definition of web services:

- Definitions – the definition element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces, and contains all the service elements;
- Types – includes data type definitions using some predefined type system (e.g. XSD);
- Message – an abstract but typed definition of data being transmitted;
- Operation – an abstract definition of a method supported by the service;
- portType – an abstract set of operations supported by one or more endpoints;
- Binding – a concrete protocol and data format definition for a particular port type;
- Port – a single endpoint defined as a combination of a binding and a network address.
- Service – a collection of related endpoints.

It should be specified that WSDL instead of introducing a new type definition language, it uses the XML Schemas specification as type system. If this type system doesn't fit for some message format it allows the use of other type definition languages via extensibility.

To attach a specific protocol or data format to an abstract message, operation, or endpoint WSDL defines a *binding* mechanism which allows the reuse of abstract definitions. In addition to the core service definition framework it introduces specific *binding extensions* for the following protocols and message formats:

- SOAP
- HTTP GET/POST
- MIME

5 Refer to [WSDL] for more information.

6 An entity on one end of a transport layer connection.

The above language extensions are layered on top of the core service definition framework and nothing forbids the use of other binding extensions with WSDL.

2.1.1 WSDL document structure

A simplified structure of a WSDL document can be seen in Figure 1 - WSDL document components, below.

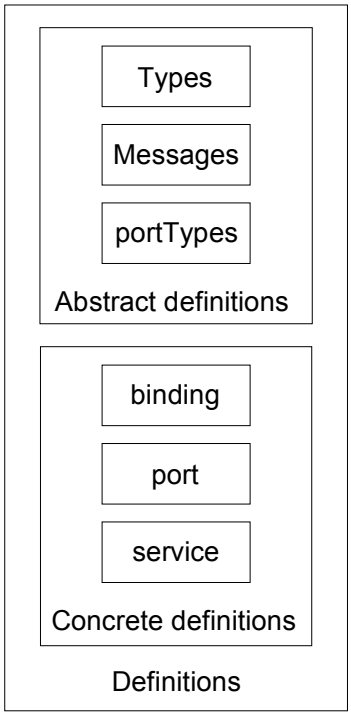

Figure 1 - WSDL document components

2.1.1.1 Types

The *types* element describes all the data types used to exchange information between the client and the server. The default type system used is the W3C XML Schema specification (see Figure 2 - WSDL Types element, below), but is not exclusively tied to this one. Since it is unreasonable to expect that a single type grammar will satisfy all the requests for abstract types an extensibility element may appear under the *types* element to identify the type definition system being used. If the service uses only built-in simple types from the XML Schema the *types* element isn't required.

```
<definitions .... >
   <types>
      <xsd:schema ... />
      ...
      <xsd:schema ... />
   </types>
   ...
</definitions>
```

Figure 2 - WSDL Types element

2.1.1.2 Message

The *message* element describes a one-way message, whether it is a request or a response. It defines the name of the message and contains none or

more message *part* elements, which may refer to message parameters and return values. Each *part* is associated with a type from a type system using a message-typing attribute. The set of message typing attributes is extensible. The syntax for defining a message can be seen in Figure 3 - WSDL Message element, below. The message *name* attribute provides an unique name among all messages defined by the WSDL document. The *part* name attribute provides an unique name among all messages defined for the current *message*. The *qname* represents an XML qualified name[7].

```
<definitions .... >
   <message name="nmtoken">
      <part name="nmtoken"  type="qname"/>
      ...
      <part name="nmtoken"  type="qname"/>
   </message>
</definitions>
```

Figure 3 - WSDL Message element

2.1.1.2.1 Message part

The message *parts* are a mechanism used to describe the abstract content of a message. Such *part* can be referenced by a *binding* in order to specific binding-specific information about that *part*. Multiple *part* elements are used if a message has more logical units, i.e. parameters and return type. If the message contents are enough complex, then an alternative syntax can be used to specify the composite structure of the message using the type system directly. In this usage, only one *part* may be specified.

2.1.1.3 PortType

The *portType* element defines the abstract *operations* that can be performed, and the abstract *messages* involved. The *portType* can be compared to a function library (or a class) in traditional programming language.

2.1.1.3.1 Operation

WSDL has four transmission primitives that an endpoint can support, and refers to these as *operations*:

- *one-way*: the endpoint receives a message – see Figure 4 - One-way operation, below;

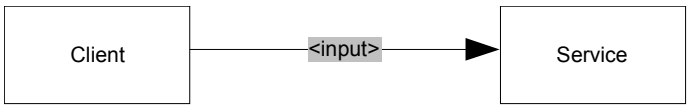

Figure 4 - One-way operation

- request-response: the endpoint receives a message, and sends a correlated message – see Figure 5 - Request-response operation, below;

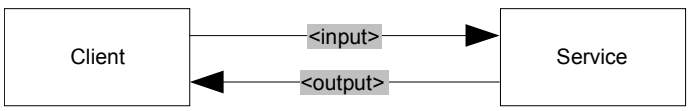

Figure 5 - Request-response operation

7 http://www.w3.org/TR/xmlschema-2/#QName

- *solicit-response*: the endpoint sends a message, and receives a correlated message – see Figure 6 - Solicit-response operation, below;

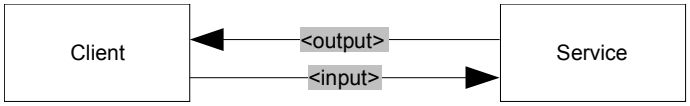

Figure 6 - Solicit-response operation

- *notification*: the endpoint sends a message – see Figure 7 - Notification operation, below.

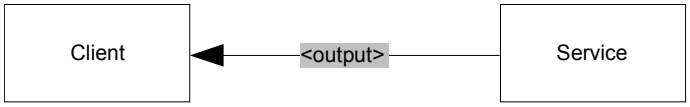

Figure 7 - Notification operation

Even if the request/response can be easily abstracted using two one-way messages, it is useful to model these as primitive operation. Despite the fact that the base WSDL structure supports bindings for these four transmission primitives, WSDL defines bindings only for the *one-way* and *request-response* primitives. For the rest of the transmission primitives it is expected that further specifications will define binding extensions that will allow the use of these *operation* primitives. The syntax for defining a *portType* is depicted in Figure 8 - portType definition, below. The presence of *input* or *output* messages inside of the *operation* definitions of a *portType* is optional and in accord with the transmission primitives.

```
<wsdl:definitions .... >
   <wsdl:portType .... > *
      <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
         <wsdl:output message="qname"/>
         <wsdl:input message="qname"/>
         <wsdl:fault message="qname"/>
      </wsdl:operation>
   </wsdl:portType >
</wsdl:definitions>
```

Figure 8 - portType definition

2.1.1.4 Binding

The *binding* element provides specific details on how a portType operation will actually be transmitted over the network. It defines message format and protocol details for a particular portType. There can be any number of bindings for a specific portType. Bindings are made available by multiple transports, including HTTP GET, HTTP POST, or SOAP. The syntax for specifying a binding can be seen in Figure 9 - Binding syntax, below.

```
<wsdl:definitions .... >
   <wsdl:binding name="nmtoken" type="qname">
      <wsdl:operation name="nmtoken">
         ...
         <wsdl:input name="nmtoken">
           ...
         </wsdl:input>
         <wsdl:output name="nmtoken">
           ...
         </wsdl:output>
         <wsdl:fault name="nmtoken">
           ...
         </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
</wsdl:definitions>
```

Figure 9 - Binding syntax

The *name* attribute provides an unique name among all bindings defined within the current WSDL document. An operation element within a binding specifies binding information for the operation with the **same name** within the binding's portType. In the case of an overloaded method (the operation names aren't constrained to be unique), the name attribute in the operation binding element might not be enough to uniquely identify the operation. In that case, the operation should be identified by providing the name attributes of the corresponding wsdl:input and wsdl:output elements.
The following constraints apply to one binding: it must specify only one protocol; and it must not specify address information.

2.1.1.5 Port

The *port* element defines an individual endpoint by specifying a single network address for a binding. The following constraints apply to one port definition: it must not specify more than one address; and it must not specify any binding information other than address information. The syntax for specifying a *port* can be seen in Figure 10 - Port syntax, below. The *name* attribute provides an unique name among all ports defined within the current WSDL document. The *binding* attribute refers to the binding.

```
<wsdl:definitions .... >
   ...
   <wsdl:service .... >
      <wsdl:port name="nmtoken" binding="qname">
         ...
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Figure 10 - Port syntax

2.1.1.6 Service

The *service* element specifies the location of the service while grouping together a set of related ports. The syntax for specifying a *port* can be seen in Figure 11 - Service syntax, below.

```
<wsdl:definitions .... >
   ...
   <wsdl:service name="nmtoken">
      <wsdl:port .... />
      ...
      <wsdl:port .... />
   </wsdl:service>
</wsdl:definitions>
```

Figure 11 - Service syntax

The *name* attribute provides an unique name among all services defined within the current WSDL document. The *service* element may include a *documentation* element in order to provide human-readable documentation.

2.1.2 A simple WSDL example

A simple WSDL example can be seen in Figure 12 - WSDL file example, below.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://localhost:8080/axis/Echo.jws"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://localhost:8080/axis/Echo.jws" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types />
  <wsdl:message name="echoRequest">
    <wsdl:part name="aMsg" type="s:string" />
  </wsdl:message>
  <wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="s:string" />
  </wsdl:message>
  <wsdl:portType name="Echo">
    <wsdl:operation name="echo" parameterOrder="aMsg">
      <wsdl:input name="echoRequest" message="tns:echoRequest" />
      <wsdl:output name="echoResponse" message="tns:echoResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="EchoSoapBinding" type="tns:Echo">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
    <wsdl:operation name="echo">
      <soap:operation soapAction="" />
      <wsdl:input name="echoRequest">
        <soap:body use="encoded" namespace="http://DefaultNamespace" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </wsdl:input>
      <wsdl:output name="echoResponse">
        <soap:body use="encoded" namespace="http://localhost:8080/axis/Echo.jws" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="EchoService">
    <wsdl:port name="Echo" binding="tns:EchoSoapBinding">
      <soap:address location="http://localhost:8080/axis/Echo.jws" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 12 - WSDL file example

This WSDL file implements a very simple web service on the Axis[8] project and the Tomcat[9] web server. More, the most simple way to implement a web service, for this configuration, was used. That was done by creating a file with the JWS (Java Web Service) extension in the root folder of the Axis implementation. This file contains a Java class which implements only one method which accepts a string as its argument and it returns it as its result. Using this method, our WSDL document was automatically generated by the server. The Java code for this example can be seen in Figure 13 - Echo.jws, below.

8 http://ws.apache.org/axis/
9 http://jakarta.apache.org/tomcat/

```
public class Echo {
  public String echo(String aMsg)
  {
    return aMsg;
  }
}
```

Figure 13 - Echo.jws

2.1.2.1 The structure of the WSDL example file

As it can be seen, our example, being very simple, has no <types ...> element. Some predefined XML schemas are included, but mostly the *wsdl*[10] and the *s*[11] namespaces are used for the types of the elements, and the *soap*[12] namespace for the binding part.

The WSDL file defines two messages (see Figure 14 - WSDL Example – The messages, below), each having only one message part. One message is used for receiving data and the other to send the answer back. Their parts are the argument passed to the Java method “echo” and its return.

```
<wsdl:message name="echoRequest">
  <wsdl:part name="aMsg" type="s:string" />
</wsdl:message>
<wsdl:message name="echoResponse">
  <wsdl:part name="echoReturn" type="s:string" />
</wsdl:message>
```

Figure 14 - WSDL Example – The messages

The *portType* element of the WSDL example file represents the abstract representation of the *echo* method of the *Echo* Java class. This can be seen in Figure 15 - WSDL Example - The portType element, below. It has the name of the Java class as its name attribute, and contains an operation specification which has its name attribute identical with the name of the method of the Java class. The input and the output declarations represent the message received by the method and the result returned to the caller.

```
<wsdl:portType name="Echo">
  <wsdl:operation name="echo" parameterOrder="aMsg">
    <wsdl:input name="echoRequest" message="tns:echoRequest" />
    <wsdl:output name="echoResponse" message="tns:echoResponse" />
  </wsdl:operation>
</wsdl:portType>
```

Figure 15 - WSDL Example - The portType element

The *binding* element uses the SOAP protocol to transport data over the wire. Its declaration inside our WSDL example can be seen in Figure 16 - WSDL Example - The binding element, below.

10 http://schemas.xmlsoap.org/wsdl/
11 http://www.w3.org/2001/XMLSchema/
12 http://schemas.xmlsoap.org/wsdl/soap/

```
<wsdl:binding name="EchoSoapBinding" type="tns:Echo">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
   <wsdl:operation name="echo">
    <soap:operation soapAction="" />
    <wsdl:input name="echoRequest">
     <soap:body use="encoded" namespace="http://DefaultNamespace" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </wsdl:input>
    <wsdl:output name="echoResponse">
     <soap:body use="encoded" namespace="http://localhost:8080/axis/Echo.jws" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </wsdl:output>
   </wsdl:operation>
 </wsdl:binding>
```

Figure 16 - WSDL Example - The binding element

Finally, the *service* element, which specifies the physical location of the web service for our example is shown in Figure 17 - WSDL Example - The service element, below. There it can be seen that the real network address is the local computer, the port 8080 used by Tomcat, and the “Echo.jws” file inside the “axis” domain.

```
<wsdl:service name="EchoService">
  <wsdl:port name="Echo" binding="tns:EchoSoapBinding">
    <soap:address location="http://localhost:8080/axis/Echo.jws" />
  </wsdl:port>
 </wsdl:service>
```

Figure 17 - WSDL Example - The service element

2.2 CORBA[13] IDL

2.2.1 CORBA

The Common Object Request Broker Architecture (CORBA) is an OMG open architecture and infrastructure that applications can use to work together over networks. Using CORBA, an application can interoperate with any other application that also uses CORBA, regardless of the operating system, programming language, and network. CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and unmarshalling; and operation dispatching.

An application which uses CORBA to interact with another CORBA enabled application uses objects. For each object type (for example an invoice) an interface in OMG IDL (Interface Definition Language) must be defined. The interface represents the contract that the server object offers to the clients that may invoke it. Any client that uses an object on the server must use this IDL interface to indicate the operations it wants to perform, and to marshal the arguments. When an operation invocation arrives at the target object, the same interface definition is used to unmarshal the arguments in order that the object can execute the requested operation with them. After the execution, the interface definition is used to marshal the results of the operation, and to unmarshal them when they reach their destination.

The separation of the interface from the implementation is a key element in CORBA, because in this way the interoperability is enabled. The interface is defined very clearly and the implementation of the object which manages the functionalities defined by the interface is hidden from the client which invokes it.

13 Version 3.0.

Clients access objects only trough their interfaces, invoking strictly only those methods that the object exposes through its IDL interface.
To make things work, the IDL interface is compiled into client stubs and object skeletons, and an object implementation is written in a chosen language. Stubs and skeletons serves as proxies for clients and servers. In order to invoke the remote object instance, the client needs to obtain its object reference. To make the remote invocation, the client uses the same code that it used for a local invocation. The only difference is the way in which the object reference is obtained. The process of invocation is illustrated in Figure 18- A client passing a request trough ORB to the server, below.

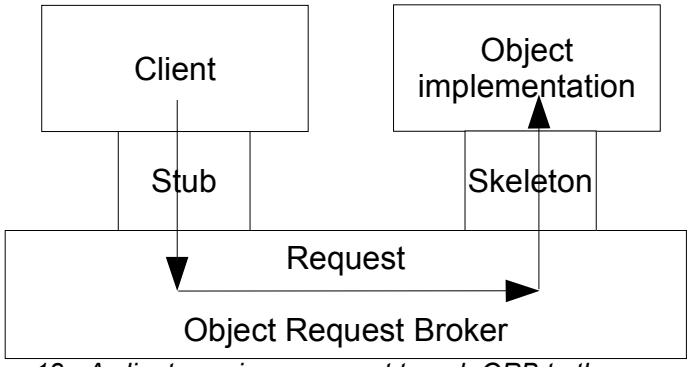

Figure 18 - A client passing a request trough ORB to the server

This process is standardised at two levels. First, the client stub and object skeleton are obtained from the same IDL. That means that the client knows exactly which operations can invoke with their return and parameters types. Second, an agreement on a common protocol must be made between the client and the object trough their ORBs. Although the ORBs can use any protocol, the OMG defines for that the standard protocol IIOP (Internet Inter-ORB Protocol). The process of remote invocation and location transparency is illustrated in Figure 19 - Location transparency (ORB to ORB invocation) below.

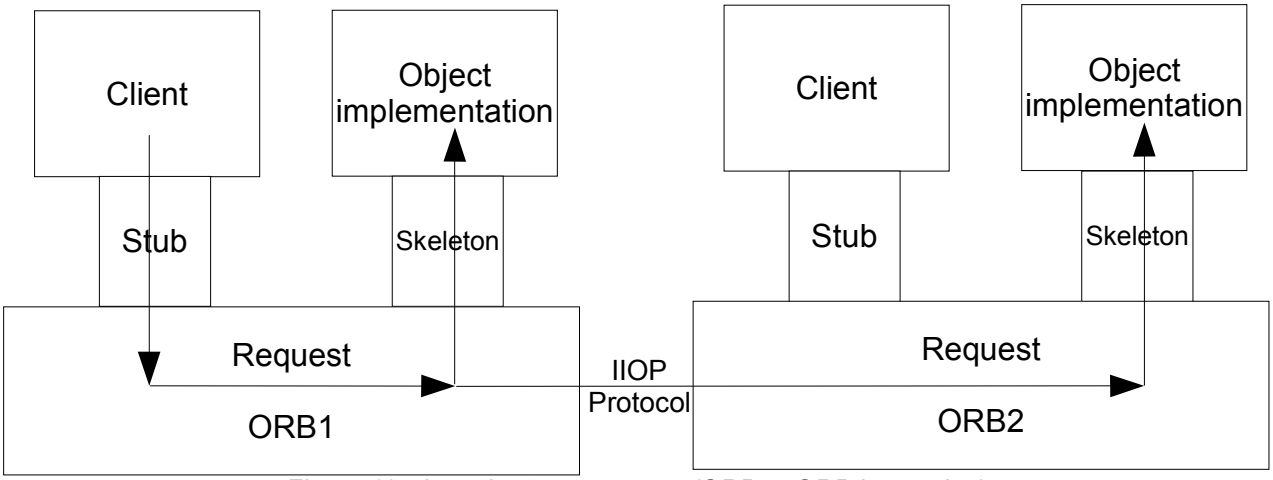

Figure 19 - Location transparency (ORB to ORB invocation)

The Object Request Broker or ORB is the central component of CORBA. It encloses all the infrastructure of the communication which is necessary to identify and locate objects, handle connection management and deliver data.
The most important functionality of the ORB is instead to pass the requests from clients to the object implementations on which they are invoked. In order to make a

request the client has to communicate with the ORB through the IDL stub. As we said, the stub represents the mapping between the programming language used by the client and the ORB. Thus the client can be written in any language as long as there are mappings for this language. The ORB then transfers the request to the object implementation which receives the request through an IDL skeleton.

2.2.2 IDL

The IDL is the language used to describe CORBA interfaces. A definition of an interface written in IDL defines the interface and specifies all the parameters of each operation. Such a definition is used to create the client stubs and the object skeleton in any chosen language for which mappings are defined. It should be said that the client stub and the object skeleton aren't necessary written using the same programming language. The IDL is designed to be independent of programming languages, but maps are defined to almost all popular programming languages (C, C++, Java, Smalltalk, COBOL, Lisp etc).
IDL is a strong typed language and it provides a full type system, exceptions and events included. An IDL specification consists of one or more type definitions, constant definitions, exception definitions, or module definitions. The syntax is:

- optional **import** pragmas
- one or more **definitions** that can be one of the following:
  - **type** declaration
  - **constant** declaration
  - **exception** declaration
  - **module** declaration
  - **interface** declaration
  - **value** declaration
  - **type id** declaration
  - **type prefix** declaration
  - **event** declaration
  - **component** declaration
  - **home** declaration

2.2.2.1 The IDL specification

In this chapter all the elements of the IDL will be discussed individually.

2.2.2.1.1 Import declaration

The import declaration imports a scope, i.e. a name space. The imported scope can be either an IDL name scope, or a string containing the ID of an IDL name scope from the interface repository, i.e., a definition object in the repository whose interface derives from CORBA::Container. The import declaration prevents the need to define the IDL constructs in terms of *file scopes*. Some of its effects are:

- the content of the imported name scope is visible in the context of the import specification. Names that may be used in IDL declarations within the importing specification can be resolved to definitions from imported scope
- importing a name scope imports recursively all name scopes included within it

- an imported IDL name scope exists in the same space name as names defined in following declarations in the importing specification
- the union of all imported scopes is visible to the importing scope, the redundant imports are disregarded
- modules defined in imported name scopes may be re-opened by IDL module definitions in the importing scope

2.2.2.1.2 Module declaration

The module declaration is used to scope the IDL identifiers, resolving the name conflicts. Usually it consists of one or more interfaces of the same application.

2.2.2.1.3 Interface declaration

An interface declaration is a description of a class in terms of methods and attributes. It consists of an interface header and an interface body. Interface declarations consisting only of interface header are called *forward* interface declarations.

2.2.2.1.3.1 Interface header

The interface header consists of three elements: an optional modifier specifying if the interface is abstract or local, the interface name, and optional inheritance specifications.

2.2.2.1.3.2 Interface body

The interface body contains one of the following declarations:

- constant declarations, which specify the constants that the interface exports
- type declarations, which specify the type definitions that the interface exports
- exception declarations, which specify the exception structures that the interface exports
- attribute declarations, which specify the attributes exported by the interface
- operation declarations, which specify the operations that the interface exports and their format, including operation name, the data type returned, the types of all parameters of an operation, exception that may be returned as a result of an invocation

2.2.2.1.4 Forward declaration

A forward declaration of an interface declares its name without defining it. This allows to refer an interface that wasn't yet defined from another interface (interfaces which refer each other).
It is not allowed to inherit from a forward declared interface whose definition hasn't yet been seen.

2.2.2.1.5 Interface inheritance

An interface can inherit from another interface, which is called a *base* interface, all the elements of the base interface. A derived interface can

declare its own new elements (types, constants, attributes, operations, and exceptions). The elements of the base interface can be referred to as if they were elements of the derived interface. The derived interface can redefine any of the constant, type, and exception names defined in the base interface. Anyway, if an element of the base interface was redefined, it still can be accessed using the "::" operator.
The purpose of interface inheritance is to make visible in the derived interface name scope all the identifiers defined in the base interface name scope.

2.2.2.1.6 Abstract interface

Abstract interfaces allow to choose at runtime whether the object will be passed by value or by reference. An abstract interface is an interface which contains in its header declaration the keyword **abstract**. There are special rules that applies to abstract interfaces:

- abstract interfaces can inherit only from other abstract interfaces
- value types can support any number of abstract interfaces

2.2.2.1.7 Local interface

A local interface is an interface which contains in its header declaration the keyword **local**. If an interface declaration does not contain the keyword **local** in its header then it is referred to as an unconstrained interface. An object implementing a local interface is referred to as an local object. There are special rules that applies to local interfaces:

- an unconstrained interface cannot inherit from a local interface. An interface which inherits from a local interface must be explicitly declared as local
- a local interface can inherit from any other interface, local or not
- a **valuetype** can support a local interface
- any IDL type may appear in the declaration of a local interface
- a local interface is a local type, consequently any non-interface type declaration constructed using a local interface is a local type. For example, an union with a member that is a local interface is itself a local type
- a local type can be used as a parameter, attribute, return type, or exception declaration of a local interface or of a **valuetype**
- a local type cannot be used as a parameter, attribute, return type, or exception declaration of an unconstrained interface or as a state member of a **valuetype**

2.2.2.2 Value declaration

The **valuetype** IDL type is used to pass state data over the net, in other words the IDL keyword **valuetype** declares an object that will be passed by value. A **valuetype** is best thought of as a **struct** type with inheritance and methods. **Valuetypes** differ from normal interfaces in that they contain properties to describe the **valuetype**'s state. Each state member defines an element of the state, which is marshaled and sent to the receiver when the **valuetype** is passed as a parameter.

There are several kinds of value type declarations: “regular” value type, boxed value types, abstract value types, and forward declarations.

2.2.2.2.1 Regular value type

A regular value type consists of a header and body elements.

2.2.2.2.1.1 Value header

The value header consists of:

- a name and optional modifier specifying if the value type uses custom marshalling
- an optional inheritance specification

2.2.2.2.1.2 Value element

A value type may contain all the elements that a regular interface can, as well as definition of state members, and initializers for that state.

2.2.2.2.1.3 Value inheritance specification

A value type can inherit from other value types and *support* optional IDL interfaces. A value type that supports a local interface doesn't become itself *local* (unmarshalable).

2.2.2.2.1.4 Initializers

To guarantee the portability of value type implementations, designers can define the signatures of initializers/constructors for non abstract value types. These look like local operation signatures except that they are prefixed with the keyword **factory**, have no return type, and are constrained to use only **in** parameters. There are no restrictions about the number of factory declarations. Initializers defined in a value type aren't inherited by its derivatives.
There is no default initializer, and without initializers the value type does not provide a portable way of creating an instance of its type at runtime.

2.2.2.2.2 Boxed value type

Some times it is useful to define a value type with no operations or inheritance and with a single state member. There is a shorthand IDL notation used to simplify the use of value types for this particular case, and it is referred to as a “value box”.

2.2.2.2.3 Abstract value type

A value type can also be an abstract one. It is called abstract because an abstract value type cannot be instantiated. An abstract value type doesn't have state members or initializers, but only operations. Syntactically the keyword **abstract** must appear in the value type header specification.

2.2.2.2.4 Value forward declaration

Similar to an interface forward declaration the value forward declaration declares the name of a value type without defining it. This allows the definition of value types that refer to each other. Syntactically it consists only in a value type header declaration.
More than one forward declarations of the same value type are permitted. It isn't allowed to inherit from a forward-declared value type whose definition has not yet been seen. Boxed value types cannot be forward declared.

2.2.2.2.5 Value type inheritance

The value type inheritance uses a terminology similar to that used to describe the interface inheritance. The name scoping and name collision rules are identical with those used for interface inheritance. In addition, an abstract value type cannot be specified as a direct base of a derived value type.
Value types can inherit from other value types and can support an interface and any number of abstract interfaces. The interface and abstract interfaces that a value type may support are listed following the **support** keyword.
In Table 1 - Inheritance relationships, below can be found the complete summary of allowable inheritance and supporting relationships between interfaces and value types.

May inherit from:	***Interface***	***Abstract Interface***	***Abstract Value***	***Stateful Value***	***Boxed Value***
Interface	multiple	multiple	no	no	no
Abstract Interface	no	multiple	no	no	no
Abstract Value	supports single	supports multiple	multiple	no	no
Stateful Value	supports single	supports multiple	multiple	single (may be truncatable)	no
Boxed Value	no	no	no	no	no

Table 1 - Inheritance relationships

2.2.2.3 Constant declaration

Constants are named variables that are restricted from being modified after being initialized.
The declaration of a constant has the following syntax: **const** <type> <identifier> = <expression>, where <type> is a base IDL type or a scoped name and <expression> an IDL expression which includes literals, IDL base types, scoped names, and operators.

2.2.2.4 Type declaration

A type declaration associates an identifier with a type. The mechanism is similar with the one used in C/C++. The IDL uses the keyword **typedef** to associate the name with the data type; the data type can be a simple type or

a complex one which uses the **struct**, **union**, or **enum** declarations. The "simple" type can be a base IDL type or a previously defined named scope. The base IDL type are enlisted in Figure 20 - Base IDL types, below.

- floating point types
  - ✗ **float**
  - ✗ **double**
  - ✗ **long double**
- integer types
  - ✗ signed int
    - ✔ signed short int: **short**
    - ✔ signed long int: **long**
    - ✔ signed longlong int: **long long**
  - ✗ unsigned int
    - ✔ unsigned short int: **unsigned short**
    - ✔ unsigned long int: **unsigned long**
    - ✔ unsigned longlong int: **unsigned long long**
- char type: **char**
- wide char type: **wchar**
- boolean type: **bool**
- octet type: **octet**
- any type: **any**

Figure 20 - Base IDL types

2.2.2.4.1 Integer Types

The IDL integer types are **short**, **unsigned short**, **long**, **unsigned long**, **long long** and **unsigned long long**. Their range of values are indicated in Table 2 - Ranges of integer types, below.

Type	*Range*
short	-2^{15} ... $2^{15} - 1$
long	-2^{31} ... $2^{31} - 1$
long long	-2^{63} ... $2^{63} - 1$
unsigned short	0 ... $2^{16} - 1$
unsigned long	0 ... $2^{32} - 1$
unsigned long long	0 ... $2^{64} - 1$

Table 2 - Ranges of integer types

2.2.2.4.2 Floating point types

The IDL floating point types are **float**, **double** and **long double** and they have correspondences in the IEEE[14] definitions of floating point numbers. The **float** type represents the single-precision floating point numbers; the **double** type represents double-precision floating point numbers; the **long double** type represents the double-extended floating point numbers.

2.2.2.4.3 Char type

The **char** data type is an 8 bit length which encodes a single byte character from any byte oriented code set, or when it is used in an array

14 IEEE stands for Institute of Electrical and Electronics Engineering. See www.ieee.org for more informations.

can encode a multi-byte character from a multi-byte code set. An implementation is free to use any code set internally for encoding character data, but for transmission a conversion to another code set may be required. Such conversions can change the representation of a character but maintain its meaning.

2.2.2.4.4 Wide char type

The **wchar** data type encodes wide characters from any character set. An implementation is free to use any cod set internally for encoding wide characters, as for **char** types. Again, conversion from one code set to another may be required. The size of **wchar** depends on the implementation.

2.2.2.4.5 Boolean type

The **boolean** data type defines an item that can have only two values: **TRUE** and **FALSE**.

2.2.2.4.6 Octet type

The **octet** type is 8 bit length and is guaranteed to not be subject of any conversion when transmitted over the net.

2.2.2.4.7 Any type

The **any** data type allows to specify values that can have any IDL type.

2.2.2.4.8 Constructed types

The IDL constructed types are: **struct**, **union**, and **enums**. The structure type allows to group more IDL variables of different types using only one name. The IDL unions are a hybrid between the C **union** and **switch** statements. IDL unions must be discriminated. The enumerations represent sequences of identifiers.

2.2.2.5 Template types

An IDL template type can be a sequence type, a string type, a wide string type, or a fixed type.

2.2.2.5.1 Sequence type

The sequence type is defined as a one-dimensional array with two characteristics: a fixed maximum size (determined at compile time) and a length(determined at runtime).

2.2.2.5.2 String type

The string type consists in a sequence of char including all possible 8 bit characters excepting *null*. In IDL strings are a separate type because many languages have dedicated functions for string manipulation. Keeping string type as a separate type allows substantial optimizations in the handling of strings.

2.2.2.5.3 Wstring type

The wide string type is a sequence of wchar, with the exception of the wide character null. The wide string type is similar to the string type, except that its elements have the type **wchar**.

2.2.2.5.4 Fixed type

The **fixed** type is a fixed point decimal number of length up to 31 significant digits. Usually the **fixed** data type is mapped to a native fixed point capability of a programming language, if available.

2.2.2.5.5 Arrays

The IDL defines multidimensional, fixed sized arrays. An array declaration must include explicit sizes for each of its dimensions. The implementation of array indexes is language mapping specific and those should not be passed as parameters or incorrect results may occur.

2.2.2.5.6 Native types

The native types are a mechanism provided by IDL to define an opaque type whose representation is specific by the language mapping for that object adapter. This kind of type declarations defines a new type, similar to an IDL base type. A native type can be used only to define results and parameters of an operation and exceptions. Native type parameters are allowed only in **local interface** or **valuetypes**.

2.2.2.6 Exception declaration

Exception declarations are in fact data structures, which if returned indicate that an exceptional condition has occurred during the request.
Each exception is defined by its IDL identifier, an exception type identifier, and the type of the associated return value. The value of an exception becomes accessible to the programmer when it is returned as a result of a request, for determining exactly which exception was raised. If the exception declarations have members, then the values of those are accessible to the programmer.

2.2.2.7 Operation declaration

The operation declarations in IDL are similar to C function declaration (or even the Java method declaration).
An operation declaration consists of:

- an optional operation attribute
- the type of the result returned by the operation. The type can be any IDL type that may be defined in IDL. If the operation doesn't return a result then it must specify the **void** type.
- the operation name which identifies the operation in the scope of the interface in which is declared
- a parameter list for the operation, that consists in zero or more parameter declarations

2.2.2.7.1 Attribute declaration

The attribute declaration specifies which invocation semantics the communication service must provide for the operation invocation. The keyword for this is **oneway**, and when it is used in front of the declaration of an operation the invocation semantics are best-effort; best-effort implies that the operation will be invoked at most once. The constraints of a method defined with the **oneway** attribute are that the operation must not contain any output parameters and must specify a **void** return type (i.e. no result returned).

2.2.2.7.2 Parameter declaration

The parameter declaration consists of a directional attribute of the parameter, the type of the parameter that can be string, wide string, or a named scope, and the parameter name which identifies the parameter in the scope of the operation for which it is declared.
The directional attribute can be:

- **in** – the parameter is passed from client to server
- **out** – the parameter is passed from server to client
- **inout** – the parameter is passed in both directions.

If the operation ends with the rise of an exception the values of the return result and of **inout** parameters aren't defined.

2.2.2.8 Attribute declaration

In addition of operations an interface can have attributes which are defined as part of the interface. An attribute definition can be seen as a pair of accessor/mutator functions (**_get/_set**), one for retrieving the value, and another to set the value of the attribute. In terms of programming languages an attribute declaration is equivalent to a C++ or Java instance variable declaration. The optional keyword **readonly** may specify that the attribute declaration which precedes is write protected.

2.2.2.9 Event declaration

The event type is a specialization of a value type, and it is dedicated to asynchronous communication. An event declaration can be a regular one, abstract, or forward declared.

2.2.2.9.1 Regular event declaration

An event type declaration is composed by an event header and an event element. The event header consists of the event's type name and an optional value inheritance. The event element can contain all the elements that a value can include.

2.2.2.9.2 Abstract event declaration

If the event header is prefixed with the keyword **abstract** then the event is an abstract one and cannot be instantiated. In fact, an abstract event declaration is a package of operations signatures implemented only locally, and no state members or initializers can be specified.

2.2.2.9.3 Forward event declaration

The forward declared event types consists only in the name declaration of the event without its definition. The syntax consists only in the keyword **eventtype** followed by the identifier which names the event. This mechanism is used to allow the cross reference between events, i.e. events that refer each other. It is illegal to inherit from a forward declared event type before seen its definition.
Because the event type is derived from value type then event type inheritance is similar to value type inheritance.

2.2.2.10 Component and Home declarations

The component and the home declarations are part of the CORBA Component Model (CCM), which is an extension to CORBA and it is by itself an entire specification. More informations on CCM and CORBA can be found on the OMG's site (www.omg.org).

3.SDL Features

3.1 Functional Features

Feature name	*Description*
Define Service Interface	The main SDL responsibility is to define the service's interface, i.e. “sequences of messages that a service sends and/or receives”. The messages may be annotated with ontology to define their semantic.
Define Services Semantic	The SDL doesn't have to define only the service semantic but also the semantic of all the messages sent and received by the services.
Support Service Composition	The SDL has to support the service composition.
Enable Data Mapping	This feature is a sub-feature of Enabling services composition; data mapping allows the communication between services that don't 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.

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.
Enable Generation of Platform/Technology Independent Interfaces	The SDL doesn't have to be platform/technology dependent.
Enable Generation of Technology/Protocol Dependent Interfaces	Using MDA tools and principles, SDL has to enable the automatic generation of Technology/Protocol dependent interfaces, like SOAP[15], WSDL or Java.

15 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 we specifically mean middleware. Interface description languages like CORBA[16] IDL[17] and WSDL[18] may be considered platform independent but are strictly middleware dependent. It is interesting to notice that a PIM is a PSM 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. The same may be said for the WSDL.
The SDL from the logical point of view is similar to WSDL v2.0 but it offers some interesting evolution from WSDL, it is middleware independent and it is semantically rich.

The Figure 21 - SDL Principal View, below shows an overview of the SDL and its dependencies with other concepts of the DBE.

16 Common Object Request Broker Architecture [CORBA]
17 Interface Definition Language [IDL]
18 WebServices Description Language[WSDL]

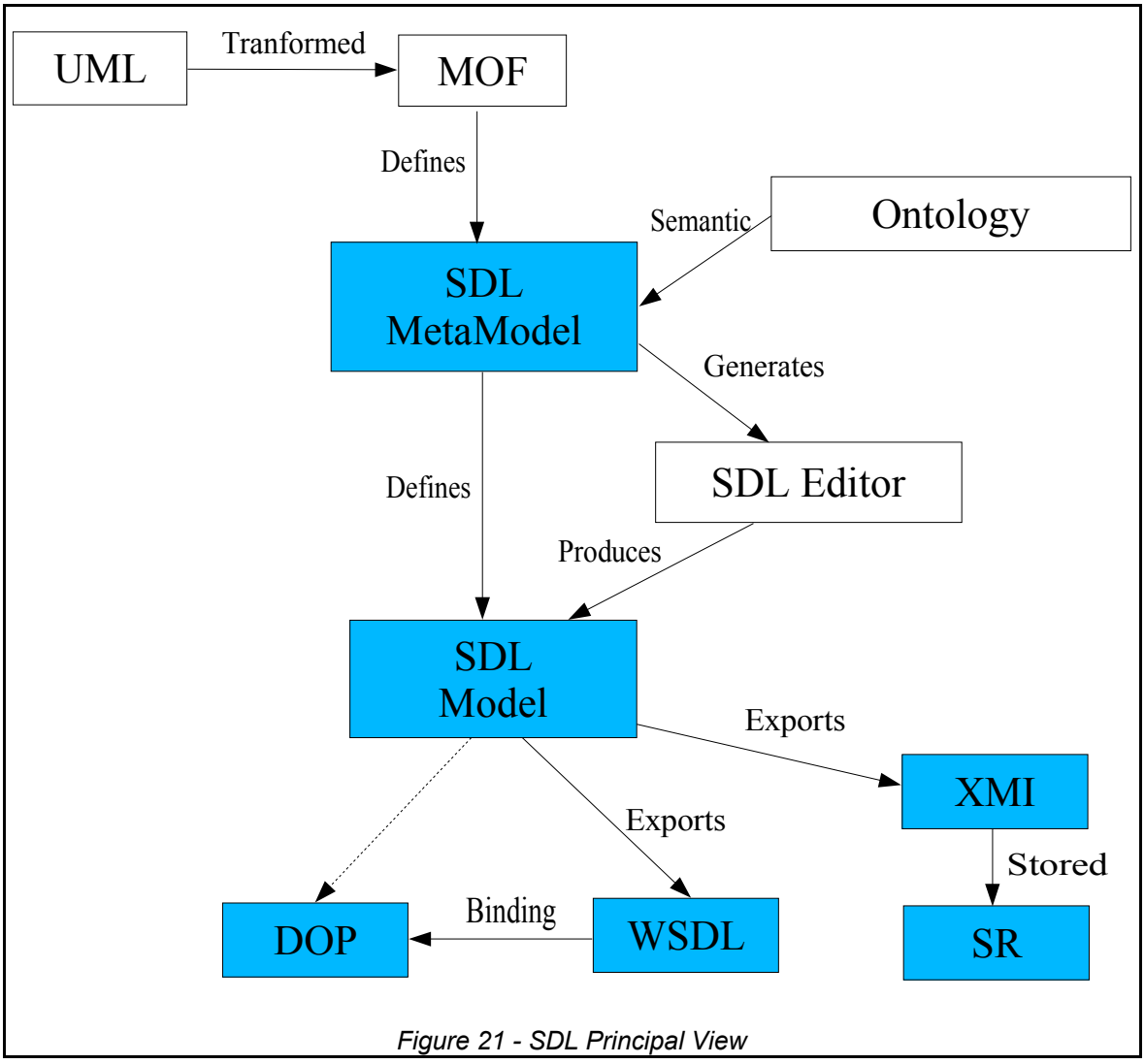

Figure 21 - SDL Principal View

In the next paragraph How we did it the Figure 21 - SDL Principal View is explained in detail.

4.1 How we did it

Since the SDL has to be an XML[19] specification, its syntax can be defined in very simple way using XML Schema[20], but the DBE adopts MDA and then the SDL Metamodel had been defined using XMI-MOF allowing to generate the SDLEditor and to store the SDL Metamodel in XMI-MOF format in the Semantic Register. Since in DBE project was decided to use only Open Source or Free Software the way to obtain the SDL Metamodel and to generate the SDL Editor from it was not really direct, the usage of many tools was needed to reach the goal.
In this chapter the reader is supposed to be proficient with XMI, UML and MOF basic concepts, this document does not aim at introduce the XMI, but we have to point out a XMI characteristic that might be not well known by all the reader.

19 eXtensible Markup Language. See [XML].
20 See [XMLS].

4.1.1 MOF XMI and UML XMI

Despite XMI means XML Metadata Interchange (easy to find Metamodel or Model instead of Metadata, but the OMG[21] definition is Metadata) is not so easy to interchange models or metamodels between different UML or MDA tools. To understand this sentence we have just to think that XMI is a standard to represent Metadata according to a Metamodel, i.e., using OMG naming convention, XMI may represent Metamodels using a MOF notation and may represents Model using a UML notation. Without introducing other complexity investigating the difference, if really exists, between model and meta-model, we have two different XMI, MOF XMI and UML XMI, and it is not possible to interchange metadata between tools using different XMI.
In addition there are a lot of versions of XMI, 1.0, 1.1, 1.2 and 2.0 and the same for UML and MOF. The previous discussion refers to OMG XMI standard, but there are also the dialects, for instance the ECORE XMI used by Eclipse[22] EMF, and then we obtain a plethora of standards. The drawback of such multitude of standards and versions is that there are the same multitude of tools to convert from an XMI standard/version to another.
For instance in the SDL Definition we use the ZARGO2ECORE Eclipse plug in to convert UML XMI to ECORE XMI, and the UML2MOF to convert UML XMI to MOF XMI.

4.1.2 Soluta.net contribution to UML2MOF

UML2MOF is a "*command-line tool that lets you convert models created using UML 1.3 into MOF 1.4 metamodels*"[23]. To make it easier to convert files using UML2MOF tool, Soluta.net published an on-line version of it (a servlet built on top of UML2MOF) which can be used for converting UML models to MOF metamodels. The servlet is available at http://www.soluta.net/xmi2mof.php
Soluta.net also helped to test and debug of the UML2MOF tool and received special thanks from Martin Matula in netbean UML2MOF official site[24].
We are currently developing an Eclipse plug-in in for the UML2MOF tool to allow the Eclipse user to use it easier.

4.1.3 Define the UML Metamodel

The first step to the SDL definition was the creation, using PoseidonUML CE[25], of the SDL Metamodel: it is shown in Figure 23 - SDL v2.0 Class Diagram and Figure 24 - SDL v 2.0 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 has been defined by an XML Schema generated from the UML Metamodel, the current version of the SDL has been defined using a XMI 1.2 MOF 1.4 Metamodel that is still generated from the UML Metamodel.

21 Object Management Group. See http://www.omg.org/
22 See http://www.eclipse.org/
23 See [U2M].
24 Idem 23.
25 See http://www.gentleware.com/

Since Poseidon can export the metamodel in XMI UML format the XMI MOF metamodel may be obtained transforming the UML model to 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 will be used to generate the SDL Editor. Despite XMI means XML Metamodel Interchange is not so easy to interchange models between different UML or MDA tools. The history of SDL Metamodel may be a clear example of this difficulty but also of the chance to reuse and transform models across different standards, formats and versions. 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 had been customized to XMI 1.2 MOF 1.4 standards. The Figure 22 - SDL Metamodel Transformations shows the SDL Metamodel history.

4.1.6 SDL Metamodel Transformation

The goal of this process is to obtain two representations of the SDL Metamodel: one 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 one in XMI 1.2 MOF 1.4 needed by the Semantic Registry to store the SDL.
The choice to start with a XMI 1.0 UML 1.2 Metamodel had been forced by the Zargo2Ecore plug-in; it is the only one plug-in available, at the beginning of the project, to convert an XMI-UML model into a XMI-ECORE model and Zargo2Ecore transforms only XMI 1.0 UML 1.2 metamodels. Once the metamodel for Eclipse EMF was ready still remains the problem to supply a XMI 1.2 MOF 1.4 metamodel for the Semantic Registry. During this process was strongly suggested to the entire DBE team to choose an unique standard in metamodel representation but this was not possible for “technical” reasons: the MDA tool selected (Eclipse MDA) and the storage support used by the SR (MDR) used different standards. However it is strongly suggested to reach an agreement on the metamodels representation in the second part of the DBE project.

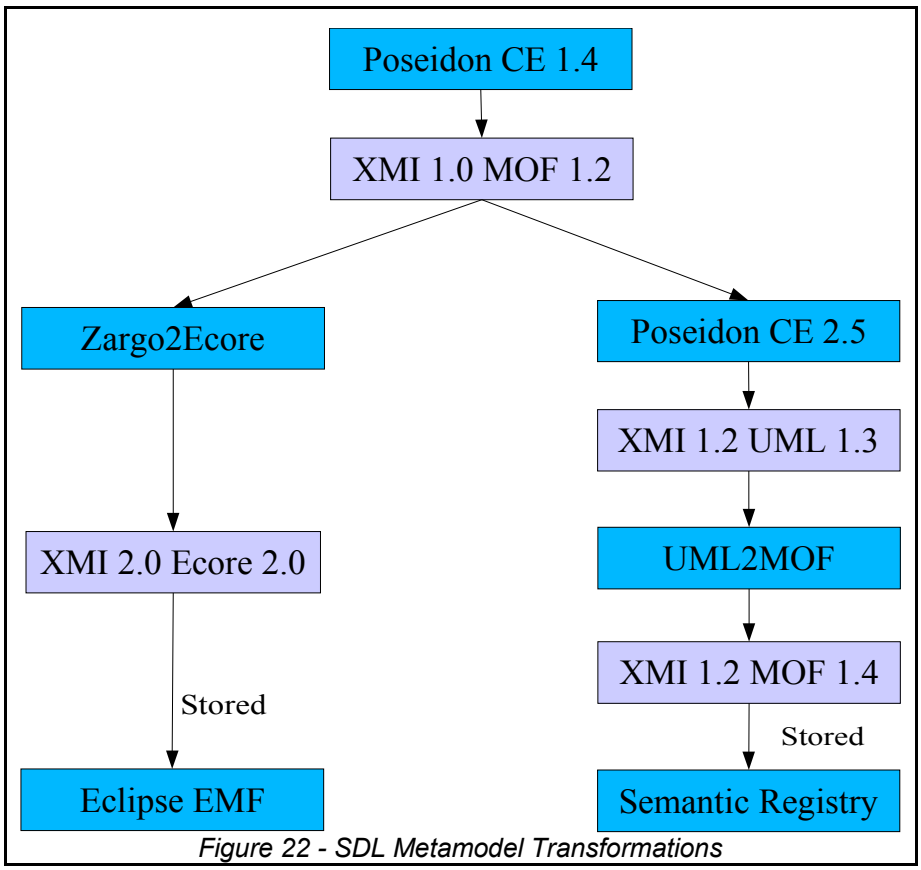

Figure 22 - SDL Metamodel Transformations

4.1.7 Edit the SDL Model

Using the SDL Editor the user can edit valid SDL Models, the model will be validated against the SDL metamodel by the validation tool within the SDL Editor. The SDL Editor offers a simple tree based graphic interface that allows the editing of SDL Models in an intuitive way. Refer to chapter 6 - SDL Example to make an idea 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 may be stored, in the same format, in the SR.
Note that a SDL Model is not expressed in MOF but is compliant to a MOF Metamodel; the SDL Model is in SDL XMI format, i.e. a XMI compliant to the SDL Metamodel, only the SDL Metamodel that is expressed in MOF. Since the SDL Metamodel is a MOF XMI Metamodel we may say that a SDL Model if MOF compliant model, then the SDL Models may be stored in MDR.

4.1.8 SDL Compiler

From the SDL Model it is hence possible to generate XMI, WSDL and Java representations using MDA based tools, Soluta.net has implemented the first version of the SDL Compiler to cross-compile the SDL to WSDL. The SDL cross-compilation to Java Interface and to Java DBE stub/skeleton is still on going.

4.2 MDA Consideration

Besides the technical approach for the definition and disseminations, the SDL specification has been provided to the DBE project teams in a MOF compliant format; more specifically it will be contained in an XMI MOF file. Such approach will enable XMI based tools to import the specification. For instance the SDL Metamodel has been imported in Eclipse EMF to generate the SDL Editor and in ATL, by the SDL Compiler, to generate mappings between SDL and Java 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 has been generated using MDA approach, one of the most interesting effect is that the Editor is updated in “real time” when the SDL Model changes. This aspect might seem trivial but there are two main aspects: the first is that we can test the SDL Metamodel as it is ready, the second is that the SDL may change very often and then the total work to realign the Editor many times may be not so negligible. Sometimes the work needed to change the Editor might also stop a evolution to the SDL.
Another really important effect of MDA approach is the possibility to use MDA transformation tool between the metamodels, for the SSL2SDL Compiler we use ATL[26], a QVT[27] implementation. The change to the SDL Metamodel, or SSL, doesn't imply change to the SSL2SDL Compiler but only 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 mechanics of the interaction -- the message formats, data types and protocols but also the meaning and purpose of the interaction, signifying an 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.
Semantics can be regarded by another point of view, that of the context in which the client and the service interact. 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 a 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, for human processes to be automated using Web Service, it is important to be explicit, as the machines don't manage to ambiguity as humans do.

26 Atlas Transformation Language[ATL]
27 Query/ Views/ Transformations[QVT]

4.3.2 Where Should Semantics Be Described?

Not knowing a Web Service semantics makes its description not functional. The main issues are where the semantics to be described and how can a client find them. The options may be:

- Using an ad-hoc, out-of-band mechanism that is not shown in the Web Service description. This is obviously not a good option, as it will be no deterministic manner for an agent who discovers a Web Service description to find the description's semantics.
- Describing the semantics in an indicated section of the Web Service description document. This is not a practical option, because the system that processes the Web Service description and the system that will read it may be different.
- The Web Service description can *reference* a distinct document that describes the semantics, by including the URL of the document as exemplified below. WSDL 2.0 chooses this manner of describing semantics, using the “targetNamespace” URL.

4.3.3 Web Service Discovery

Web Service discovery indicates the capacity of finding in a dynamically way, a Web Service for what an application requires. For example, finding a nearby printer service for printing out a document from my PDA, in a dynamically manner.
The ambiguity of the term “discovery” is indicated by two 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.
There have been recommendations on different types of Web Service discovery, centralized (an example being UDDI) and decentralized (as conventional Web searching). Although WSDL 2.0 does not deal with the problem of Web Service discovery, discovery mechanisms intended to provide improved capabilities to a client which searches for a specific type of Web Service 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.
Due to the client requirements of selecting a Web Service with the requested semantics, semantics are important to discovery. Anyway, a complete Web Service description of the syntax for interacting with the service and of the semantics of the interaction allows the client to use a possible service regardless the discovery of the service.

4.4 Abstract Interface Vs. 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 may be interesting pointing out a particular aspect of the PIM concept. As stated previously, the SDL has to be PIM, and, for SDL purpose, PIM 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.
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.

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 23 - SDL v2.0 Class Diagram and in UML Inheritance Diagram shown in Figure 24 - SDL v 2.0 Inheritance Diagram: they have been made with Poseidon[HMAD].

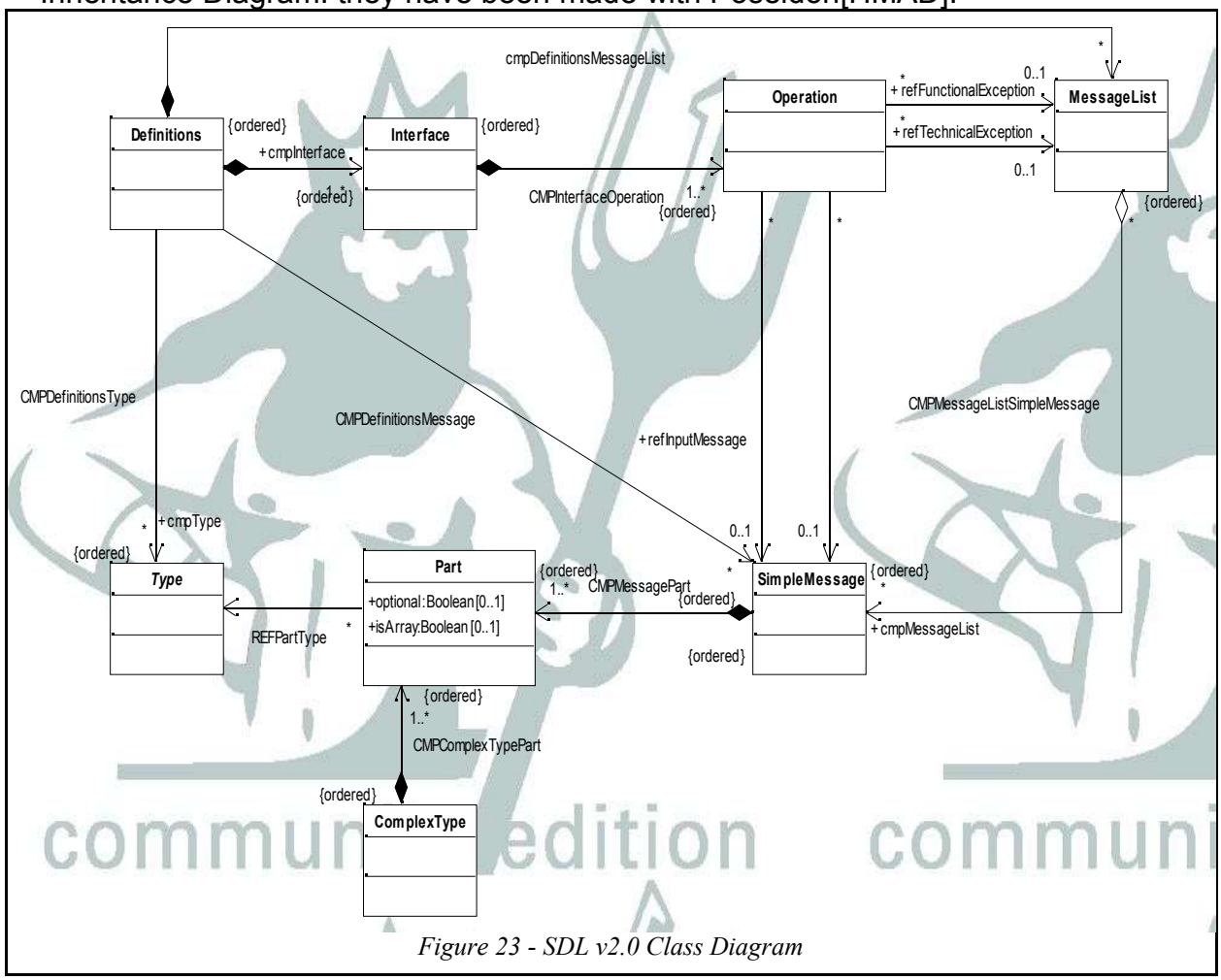

Figure 23 - SDL v2.0 Class Diagram

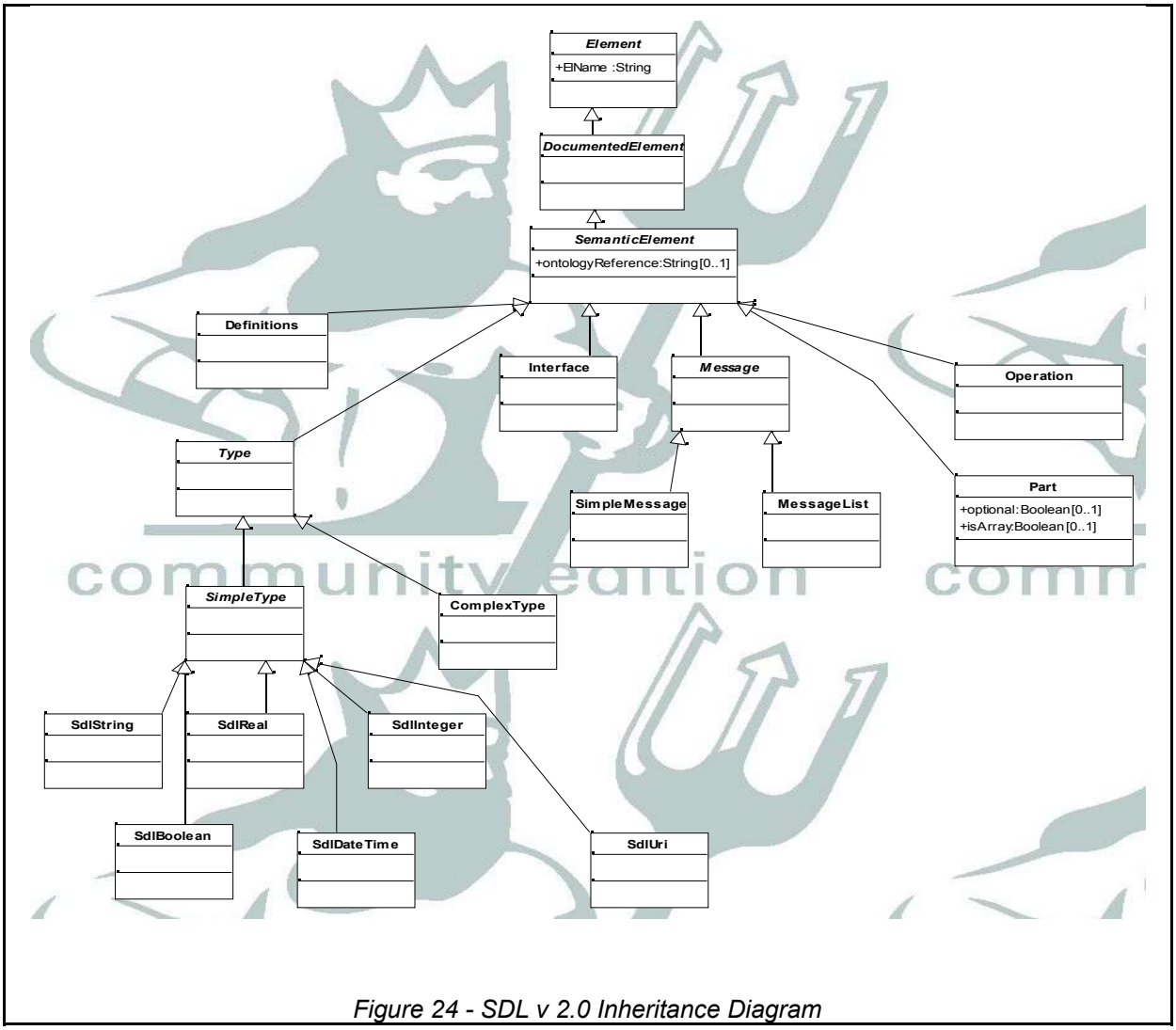

Figure 24 - SDL v 2.0 Inheritance Diagram

As stated in chapter 4 - Introduction to SDL the SDL Metamodel define 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 23 - SDL v2.0 Class Diagram shows all SDL components and their relations, in this paragraph all the main components are individually discussed.

5.2.1 SemanticElement

All the SDL Elements derive from the SemanticElement. The SemanticElement diagram is shown in Figure 25 - SemanticElement.

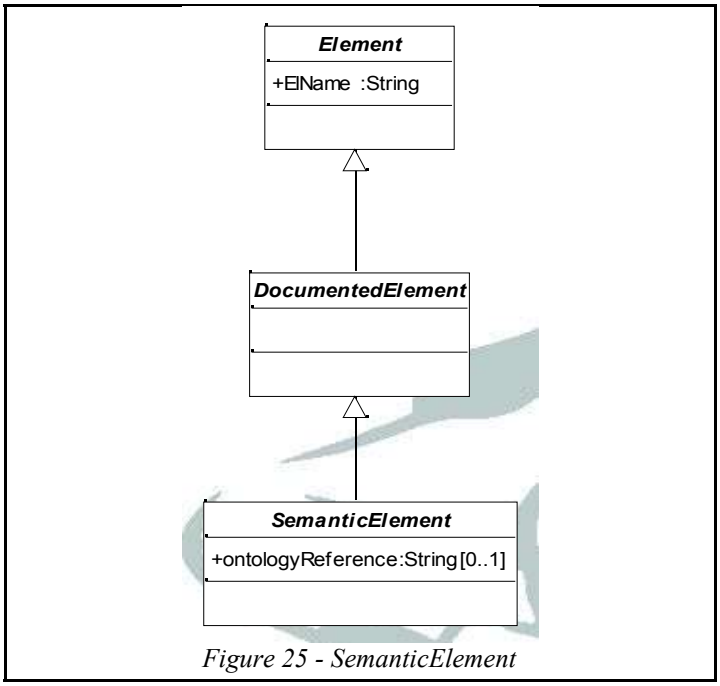

Figure 25 - SemanticElement

The ontologyReference had been introduced in accordance with the consideration expressed in the link to an ontology seams to be the simplest way to introduce semantic in the SDL.

Note also that, at the moment, the DocumentedElement is "empty": since the documentation might be related to the Versioning system.

5.2.2 Definitions

Definitions is the root of a Service Description, it contains the Type, Interface and Message definition. The Figure 26 - Definitions UML Diagrams shows its structure.

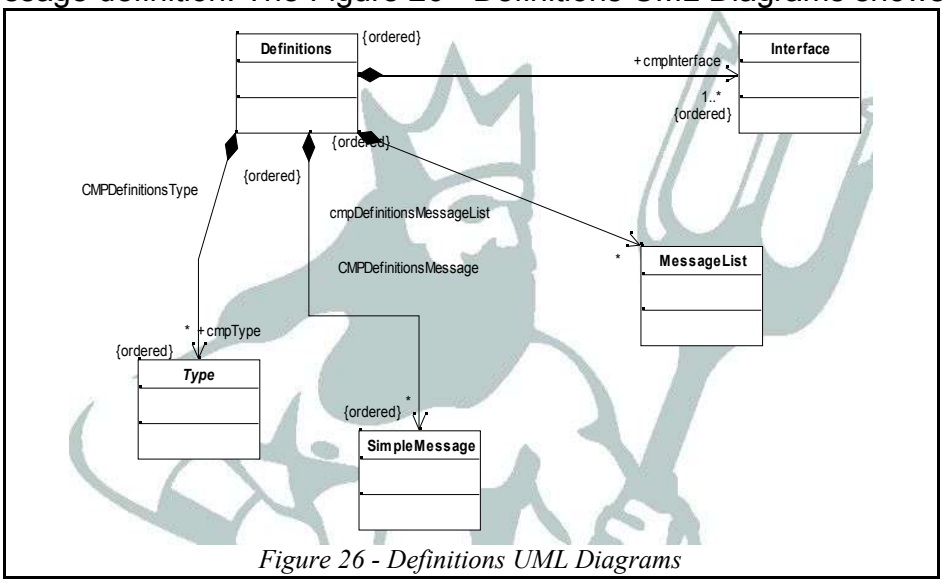

Figure 26 - Definitions UML Diagrams

The *Definitions* is a ***composition*** of Interfaces, Types and Messages (SimpleMessage and MessageList), composition means that all the objects *belong* to the *Definitions*, an object can belong only 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.

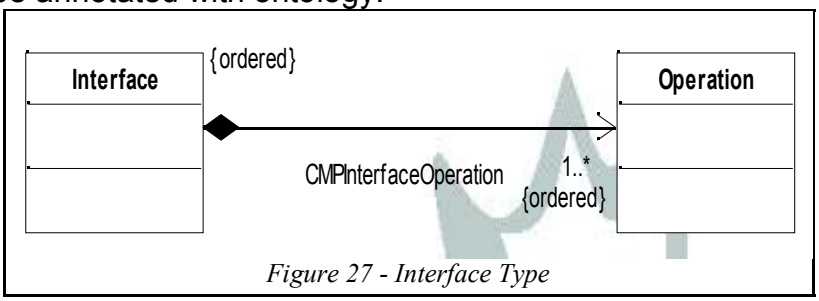

Figure 27 - Interface Type

"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 message the 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". A "business reason may be that, for a Hotel reservation service, that there are not rooms for the selected date. The management of the Technical and Functional Exception is, obviously, different: if the database is down 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.

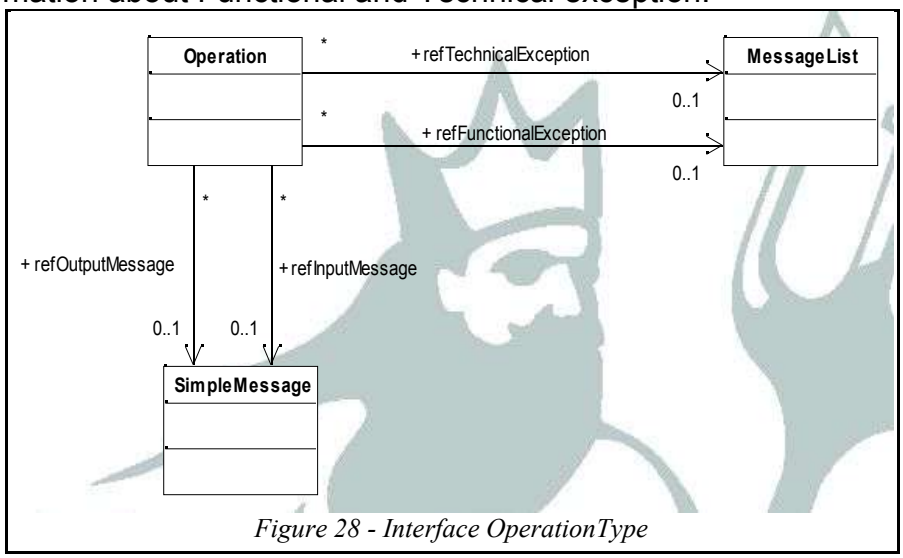

Figure 28 - Interface OperationType

Note that in SDL the granularity of operation is more particularised than in SSL/BML[28]. While in SSL/BML only the semantically significant operations are expressed, in SDL are expressed also the "technical operations". Think to the MakeReservation operation in an HotelReservation service : in SSL/BML only the MakeReservation operation and the related messages are defined; the

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

InputMessage for MakeReservation may contain, for instance, a RoomType and the reservation date. In SDL may be defined also the method getRoomTypeList that returns a list of Strings (or RoomType) describing the RoomType available in that Hotel. In SDL will be modelled all the support operations that may help to perform the transaction, all the auxiliaries methods.

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.

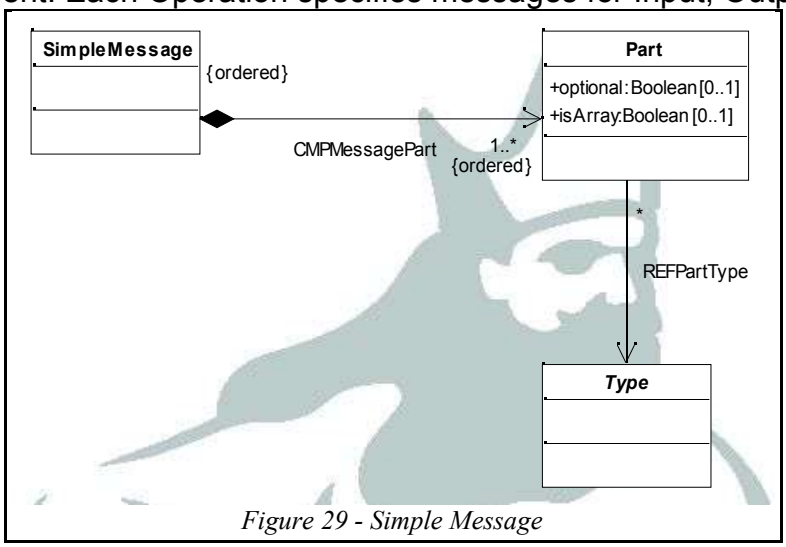

Figure 29 - Simple Message

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

5.2.6 Message List

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

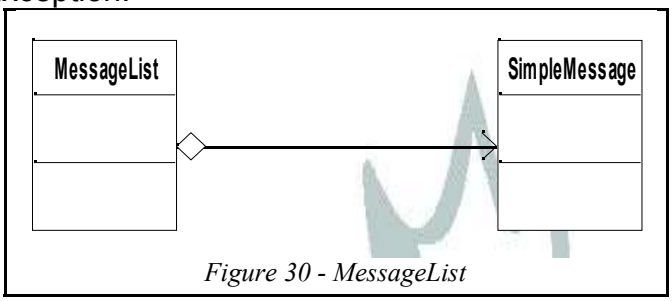

Figure 30 - MessageList

The concept of MessageList is shown in Figure 30 - 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 that it is needed a list of possibilities when a Technical or Functional error occurs. The picture Figure 28 - Interface OperationType shows the relation between the Operation and the MessageList.

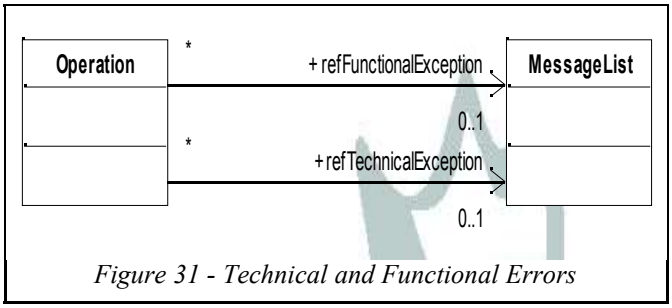

Figure 31 - Technical and Functional Errors

It is suggested that when an Exception occurs the result may be specialized in relation to the exception. For instance if a proxy is not available it may be useful that the Technical Exception contains the ProxyID while if is a Network to be 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.

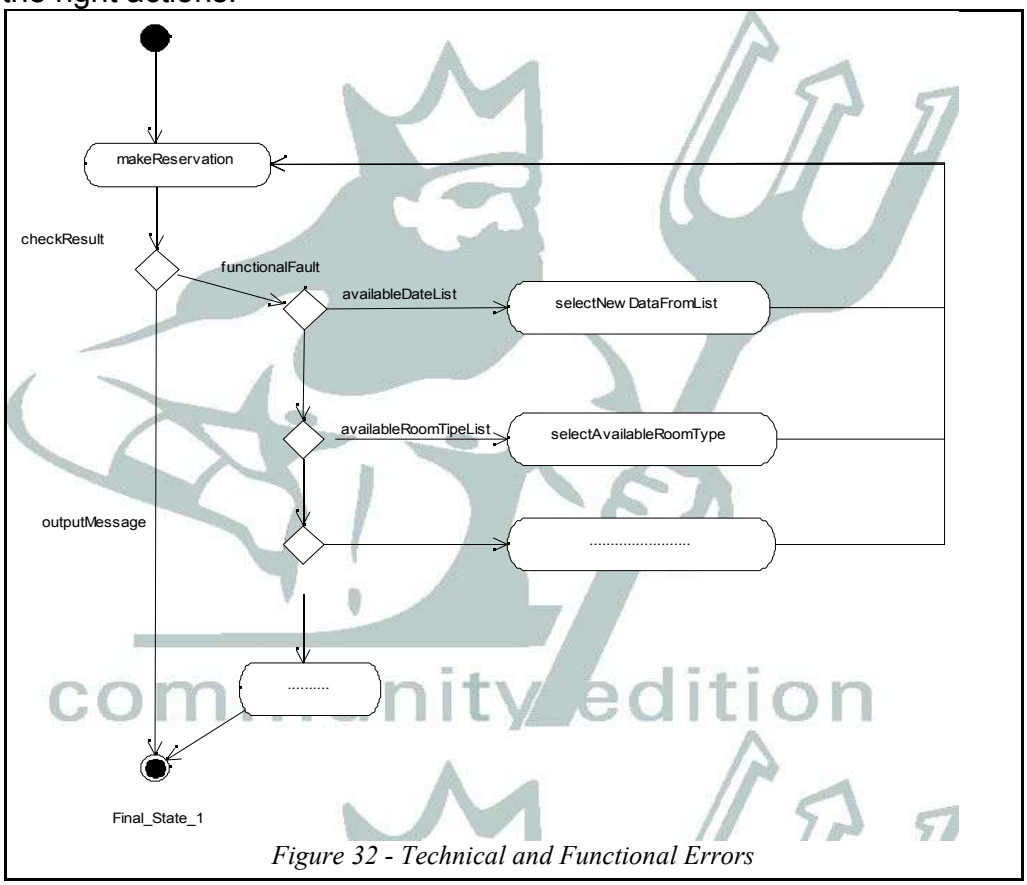

Figure 32 - Technical and Functional Errors

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 31 - 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 the service can return the list of the nearest available data? And how the client can understand the type of error? In the Figure 32 - 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 picture Figure 33 - 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.

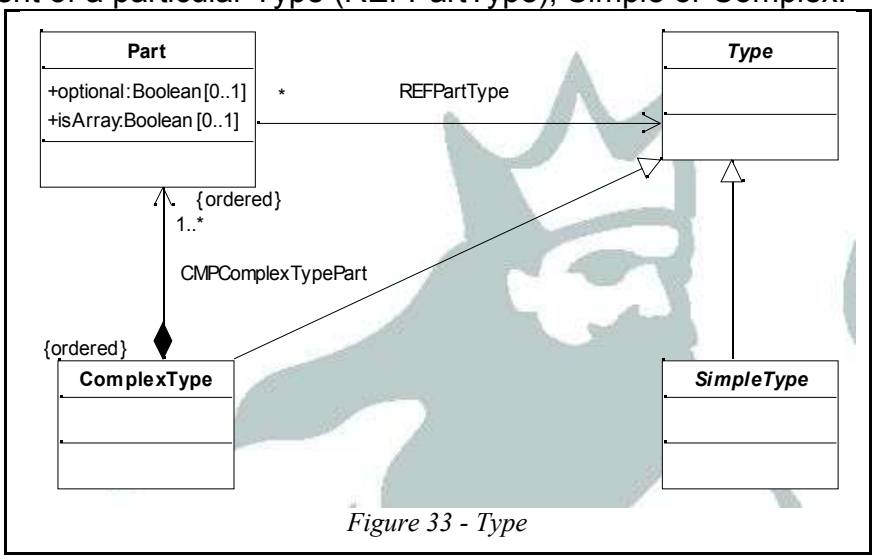

Figure 33 - Type

SimpleType is an abstract concept, as shown in Figure 34 - SimpleType the SDL SimpleType are: SDLBoolean, SDLInteger, SDLReal, SDLDateTime, SDLUri and SDLString. This is the SimpleType used in SDL Models.

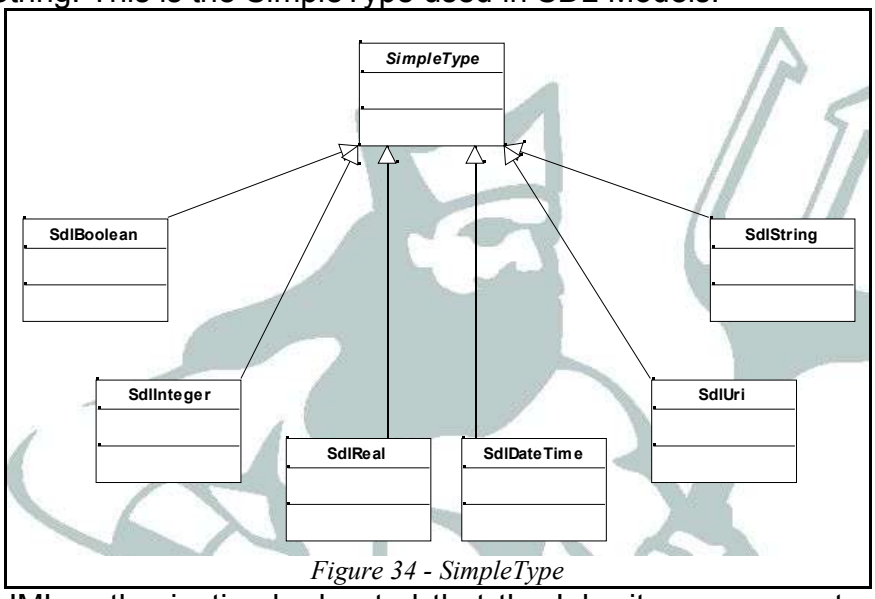

Figure 34 - SimpleType

The UML enthusiastics had noted that the Inheritance seems to be an “inclusive inheritance” when this is illogic. The problem is that the modelling tool used (Poseidon CE) does not implement the concept of “inclusive” and “exclusive”

inheritance. Inclusive inheritance means that an object can be at the same time a SDLInteger and a SDLDateTime, exclusive inheritance means that an object can be a SDLInteger or a SDLDateTime but not together.

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 really similar to WSDL this first version was defined to allow all DBE developers to use it early and easy also before the SDL Editor was ready. The SDL1.0 was defined by an XML Schema and had an important peculiarity: it was easy to write and validate using an XML editor.
The second version of SDL aims to be a MOF Based language, a language that can be transformed using MDA tools such as Eclipse EMF or ATL[29] (QVT[30] implementation).
The model didn't change a lot, only the way to represent the model had been changed. This difference may be clear comparing the sample models shown in Figure 36 - TicketAgent SDL1.0 (XML SDL) and in Figure 37 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types.

5.3.2 XMI SDL Metamodel

As stated in Ch. 3-SDL Features the SDL has to be an XMI1.2 MOF1.4 specification. In the Ch. 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 collabnet in the project *languages,* folder *WP-16SDL* file *SDL Metamodel 2.0 XMI-MOF.*[31]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<XMI xmi.version="1.2" xmlns:Model="org.omg.xmi.namespace.Model" timestamp="Mon Jan 24 12:40:24 CET 2005">
    <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="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="a1"/>
                    </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>
                        <Model:Package xmi.idref="a1"/>
                    </Model:Tag.elements>
                </Model:Tag>
                <Model:Class xmi.id="a2" name="Element" annotation="" isRoot="false" isLeaf="false" isAbstract="true" visibility="public_vis" isSingleton="false">
                    <Model:Namespace.contents>
                        <Model:Attribute xmi.id="a3" 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="a4"/>
                            </Model:TypedElement.type>
                        </Model:Attribute>
```

29 Atlas Transformation Language[ATL]
30 Query/ Views/ Transformations[QVT]
31 https://languages.digital-ecosystem.net/files/documents/18/456/SDLModel20-MOF.xmi

```
                </Model:Namespace.contents>
            </Model:Class>
            <Model:Class xmi.id="a5" name="DocumentedElement" annotation="" isRoot="false" isLeaf="false" isAbstract="true" visibility="public_vis" isSingleton="false">
                <Model:GeneralizableElement.supertypes>
                    <Model:Class xmi.idref="a2"/>
                </Model:GeneralizableElement.supertypes>
            </Model:Class>
            <Model:Class xmi.id="a6" name="SemanticElement" annotation="" isRoot="false" isLeaf="false" isAbstract="true" visibility="public_vis" isSingleton="false">
                <Model:Namespace.contents>
                    <Model:Attribute xmi.id="a7" 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="a4"/>
                        </Model:TypedElement.type>
                    </Model:Attribute>
                </Model:Namespace.contents>
                <Model:GeneralizableElement.supertypes>
                    <Model:Class xmi.idref="a5"/>
                </Model:GeneralizableElement.supertypes>
            </Model:Class>
            <Model:Class xmi.id="a8" name="Definitions" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                <Model:Namespace.contents>
                    <Model:Reference xmi.id="a9" 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="a10"/>
                        </Model:TypedElement.type>
                        <Model:Reference.referencedEnd>
                            <Model:AssociationEnd xmi.idref="a11"/>
                        </Model:Reference.referencedEnd>
                    </Model:Reference>
                    <Model:Reference xmi.id="a12" 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="a13"/>
                        </Model:TypedElement.type>
                        <Model:Reference.referencedEnd>
                            <Model:AssociationEnd xmi.idref="a14"/>
                        </Model:Reference.referencedEnd>
                    </Model:Reference>
                    <Model:Reference xmi.id="a15" 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="a16"/>
                        </Model:TypedElement.type>
                        <Model:Reference.referencedEnd>
                            <Model:AssociationEnd xmi.idref="a17"/>
                        </Model:Reference.referencedEnd>
                    </Model:Reference>
                    <Model:Reference xmi.id="a18" 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="a19"/>
                        </Model:TypedElement.type>
                        <Model:Reference.referencedEnd>
                            <Model:AssociationEnd xmi.idref="a20"/>
                        </Model:Reference.referencedEnd>
                    </Model:Reference>
                </Model:Namespace.contents>
                <Model:GeneralizableElement.supertypes>
                    <Model:Class xmi.idref="a6"/>
                </Model:GeneralizableElement.supertypes>
            </Model:Class>
            <Model:Class xmi.id="a13" name="Interface" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                <Model:Namespace.contents>
```

```
                        <Model:Reference xmi.id="a21" 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="a22"/>
                            </Model:TypedElement.type>
                            <Model:Reference.referencedEnd>
                                <Model:AssociationEnd xmi.idref="a23"/>
                            </Model:Reference.referencedEnd>
                        </Model:Reference>
                    </Model:Namespace.contents>
                    <Model:GeneralizableElement.supertypes>
                        <Model:Class xmi.idref="a6"/>
                    </Model:GeneralizableElement.supertypes>
                </Model:Class>
                <Model:Association xmi.id="a24" name="CMPDefinitionsInterface" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                    <Model:Namespace.contents>
                        <Model:AssociationEnd xmi.id="a25" 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="a8"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                        <Model:AssociationEnd xmi.id="a14" 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="a13"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                    </Model:Namespace.contents>
                </Model:Association>
                <Model:Class xmi.id="a10" name="SimpleMessage" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                    <Model:Namespace.contents>
                        <Model:Reference xmi.id="a26" 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="a27"/>
                            </Model:TypedElement.type>
                            <Model:Reference.referencedEnd>
                                <Model:AssociationEnd xmi.idref="a28"/>
                            </Model:Reference.referencedEnd>
                        </Model:Reference>
                    </Model:Namespace.contents>
                    <Model:GeneralizableElement.supertypes>
                        <Model:Class xmi.idref="a29"/>
                    </Model:GeneralizableElement.supertypes>
                </Model:Class>
                <Model:Class xmi.id="a22" name="Operation" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                    <Model:Namespace.contents>
                        <Model:Reference xmi.id="a30" 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="a19"/>
                            </Model:TypedElement.type>
                            <Model:Reference.referencedEnd>
                                <Model:AssociationEnd xmi.idref="a31"/>
                            </Model:Reference.referencedEnd>
                        </Model:Reference>
                        <Model:Reference xmi.id="a32" 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="a19"/>
                                </Model:TypedElement.type>
                                <Model:Reference.referencedEnd>
                                    <Model:AssociationEnd xmi.idref="a33"/>
                                </Model:Reference.referencedEnd>
                            </Model:Reference>
                            <Model:Reference xmi.id="a34" name="refInputMessage" 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="a10"/>
                                </Model:TypedElement.type>
                                <Model:Reference.referencedEnd>
                                    <Model:AssociationEnd xmi.idref="a35"/>
                                </Model:Reference.referencedEnd>
                            </Model:Reference>
                            <Model:Reference xmi.id="a36" name="refOutputMessage" 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="a10"/>
                                </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="a6"/>
                        </Model:GeneralizableElement.supertypes>
                    </Model:Class>
                    <Model:Class xmi.id="a27" name="Part" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                        <Model:Namespace.contents>
                            <Model:Attribute xmi.id="a38" 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="a39"/>
                                </Model:TypedElement.type>
                            </Model:Attribute>
                            <Model:Attribute xmi.id="a40" 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="a39"/>
                                </Model:TypedElement.type>
                            </Model:Attribute>
                            <Model:Reference xmi.id="a41" 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="a16"/>
                                </Model:TypedElement.type>
                                <Model:Reference.referencedEnd>
                                    <Model:AssociationEnd xmi.idref="a42"/>
                                </Model:Reference.referencedEnd>
                            </Model:Reference>
                        </Model:Namespace.contents>
                        <Model:GeneralizableElement.supertypes>
                            <Model:Class xmi.idref="a6"/>
                        </Model:GeneralizableElement.supertypes>
                    </Model:Class>
                    <Model:Association xmi.id="a43" name="CMPDefinitionsMessage" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a44" 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="a8"/>
                    </Model:TypedElement.type>
                </Model:AssociationEnd>
                <Model:AssociationEnd xmi.id="a11" 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="a10"/>
                    </Model:TypedElement.type>
                </Model:AssociationEnd>
            </Model:Namespace.contents>
        </Model:Association>
        <Model:Association xmi.id="a45" name="CMPMessagePart" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isDerived="false">
            <Model:Namespace.contents>
                <Model:AssociationEnd xmi.id="a46" 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="a10"/>
                    </Model:TypedElement.type>
                </Model:AssociationEnd>
                <Model:AssociationEnd xmi.id="a28" 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="a27"/>
                    </Model:TypedElement.type>
                </Model:AssociationEnd>
            </Model:Namespace.contents>
        </Model:Association>
        <Model:Class xmi.id="a16" name="Type" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a6"/>
            </Model:GeneralizableElement.supertypes>
        </Model:Class>
        <Model:Class xmi.id="a47" name="ComplexType" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
            <Model:Namespace.contents>
                <Model:Reference xmi.id="a48" 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="a27"/>
                    </Model:TypedElement.type>
                    <Model:Reference.referencedEnd>
                        <Model:AssociationEnd xmi.idref="a49"/>
                    </Model:Reference.referencedEnd>
                </Model:Reference>
            </Model:Namespace.contents>
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a16"/>
            </Model:GeneralizableElement.supertypes>
        </Model:Class>
        <Model:Class xmi.id="a50" name="SimpleType" annotation="" isRoot="false" isLeaf="false" isAbstract="true" visibility="public_vis" isSingleton="false">
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a16"/>
            </Model:GeneralizableElement.supertypes>
        </Model:Class>
        <Model:Class xmi.id="a51" name="SdlString" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a50"/>
            </Model:GeneralizableElement.supertypes>
        </Model:Class>
        <Model:Class xmi.id="a52" name="SdlReal" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a50"/>
            </Model:GeneralizableElement.supertypes>
        </Model:Class>
        <Model:Class xmi.id="a53" name="SdlInteger" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
            <Model:GeneralizableElement.supertypes>
                <Model:Class xmi.idref="a50"/>
            </Model:GeneralizableElement.supertypes>
```

```
                    </Model:Class>
                    <Model:Association xmi.id="a54" name="CMPInterfaceOperation" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a55" 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="a13"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a23" 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="a22"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a56" name="CMPDefinitionsType" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a57" 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="a8"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a17" 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="a16"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a58" name="REFImputMessageOperation" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a59" 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="a22"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a35" name="refInputMessage" 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="a10"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a60" name="REFPartType" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a42" 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="a16"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a61" 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="a27"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a62" name="REFOutputMessageOperation" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a63" 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="a22"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a37" name="refOutputMessage" 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="a10"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a64" name="CMPComplexTypePart" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a65" 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="a47"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a49" 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="a27"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Class xmi.id="a66" name="SdlBoolean" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                        <Model:GeneralizableElement.supertypes>
                            <Model:Class xmi.idref="a50"/>
                        </Model:GeneralizableElement.supertypes>
                    </Model:Class>
                    <Model:Class xmi.id="a67" name="SdlDateTime" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                        <Model:GeneralizableElement.supertypes>
                            <Model:Class xmi.idref="a50"/>
                        </Model:GeneralizableElement.supertypes>
                    </Model:Class>
                    <Model:Class xmi.id="a68" name="SdlUri" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
                        <Model:GeneralizableElement.supertypes>
                            <Model:Class xmi.idref="a50"/>
                        </Model:GeneralizableElement.supertypes>
                    </Model:Class>
                    <Model:Class xmi.id="a19" name="MessageList" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
```

```
                    <Model:Namespace.contents>
                        <Model:Reference xmi.id="a69" 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="a10"/>
                            </Model:TypedElement.type>
                            <Model:Reference.referencedEnd>
                                <Model:AssociationEnd xmi.idref="a70"/>
                            </Model:Reference.referencedEnd>
                        </Model:Reference>
                    </Model:Namespace.contents>
                    <Model:GeneralizableElement.supertypes>
                        <Model:Class xmi.idref="a29"/>
                    </Model:GeneralizableElement.supertypes>
                </Model:Class>
                <Model:Association xmi.id="a71" name="CMPMessageListSimpleMessage" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                    <Model:Namespace.contents>
                        <Model:AssociationEnd xmi.id="a72" 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="a19"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                        <Model:AssociationEnd xmi.id="a70" 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="a10"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                    </Model:Namespace.contents>
                </Model:Association>
                <Model:Association xmi.id="a73" name="refOperationMessageList" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                    <Model:Namespace.contents>
                        <Model:AssociationEnd xmi.id="a74" 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="a22"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                        <Model:AssociationEnd xmi.id="a31" 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="a19"/>
                            </Model:TypedElement.type>
                        </Model:AssociationEnd>
                    </Model:Namespace.contents>
                </Model:Association>
                <Model:Class xmi.id="a29" name="Message" annotation="" isRoot="false" isLeaf="false" isAbstract="true" visibility="public_vis" isSingleton="false">
                    <Model:GeneralizableElement.supertypes>
                        <Model:Class xmi.idref="a6"/>
                    </Model:GeneralizableElement.supertypes>
                </Model:Class>
                <Model:Association xmi.id="a75" name="REFFaultMessageListOperation" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                    <Model:Namespace.contents>
                        <Model:AssociationEnd xmi.id="a76" 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="a22"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a33" 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="a19"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Association xmi.id="a77" name="cmpDefinitionsMessageList" annotation="" isRoot="false" isLeaf="false" isAbstract="false" visibility="public_vis"
isDerived="false">
                        <Model:Namespace.contents>
                            <Model:AssociationEnd xmi.id="a78" 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="a8"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                            <Model:AssociationEnd xmi.id="a20" 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="a19"/>
                                </Model:TypedElement.type>
                            </Model:AssociationEnd>
                        </Model:Namespace.contents>
                    </Model:Association>
                    <Model:Import xmi.id="a79" name="PrimitiveTypes" annotation="" visibility="public_vis" isClustered="false">
                        <Model:Import.importedNamespace>
                            <Model:Package xmi.idref="a80"/>
                        </Model:Import.importedNamespace>
                    </Model:Import>
                </Model:Namespace.contents>
            </Model:Package>
            <Model:Package xmi.id="a80" name="PrimitiveTypes" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis">
                <Model:Namespace.contents>
                    <Model:PrimitiveType xmi.id="a81" name="Integer" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                    <Model:PrimitiveType xmi.id="a82" name="Long" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                    <Model:PrimitiveType xmi.id="a83" name="Float" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                    <Model:PrimitiveType xmi.id="a84" name="Double" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                    <Model:PrimitiveType xmi.id="a39" name="Boolean" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                    <Model:PrimitiveType xmi.id="a4" name="String" annotation="" isRoot="true" isLeaf="true" isAbstract="false" visibility="public_vis"/>
                </Model:Namespace.contents>
            </Model:Package>
        </XMI.content>
</XMI>
```

5.4 Inheritance

In order to reduce management complexity and to improve performance the OO inheritance isn't allowed in SDL. The suggested way to extend a service is by *copy and modify*. The SDL Editor will provide the features to copy and versioning a SDL Model.

6.SDL Example

In the follow example the semantics aspect of the SDL is not pointed out because the work of the other partner related to semantic was not terminated at the time. Also the SDL Editor at the moment does not allow to edit the Ontology Reference.

6.1 TicketAgent

TicketAgent is a WSDL 1.2 example supplied by 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.

6.1.1 WSDL Definition

In Figure 35 - 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 35 - TicketAgent WSDL

6.1.2 SDL Definition

The Figure 36 - TicketAgent SDL1.0 (XML SDL) shows the example in SDL 1.0 format, the similarity with the WSDL is very strong, the main difference is the presence of the ontology link.

```
<?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 36 - 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 37 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types and Figure 38 - 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 to parts: the first in the Figure 37 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types shows the messages and the types used, the second Figure 38 - 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: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>
 </sdl:Definitions.cmpMessage>
```

Figure 37 - TicketAgent SDL2.0 (XMI-SDL) - Messages and Types

```
<sdl:Definitions.cmpInterface>
  <sdl:Interface xmi.id="_0.@cmpInterface.0" ElName="TicketAgent">
    <sdl:Interface.cmpOperation>
      <sdl:Operation xmi.id="_0.@cmpInterface.0.@cmpOperation.0" ElName="listFlights">
        <sdl:Operation.refOutputMessage>
          <sdl:SimpleMessage xmi.idref="_0.@cmpMessage.1"/>
        </sdl:Operation.refOutputMessage>
        <sdl:Operation.refInputMessage>
          <sdl:SimpleMessage xmi.idref="_0.@cmpMessage.0"/>
        </sdl:Operation.refInputMessage>
        <sdl:Operation.refTechnicalException>
          <sdl:MessageList xmi.idref="_0.@cmpMessageList.0"/>
        </sdl:Operation.refTechnicalException>
        <sdl:Operation.refFunctionalException>
          <sdl:MessageList xmi.idref="_0.@cmpMessageList.0"/>
        </sdl:Operation.refFunctionalException>
      </sdl:Operation>
      <sdl:Operation xmi.id="_0.@cmpInterface.0.@cmpOperation.1" ElName="reserveFlight">
        <sdl:Operation.refOutputMessage>
          <sdl:SimpleMessage xmi.idref="_0.@cmpMessage.3"/>
        </sdl:Operation.refOutputMessage>
        <sdl:Operation.refInputMessage>
          <sdl:SimpleMessage xmi.idref="_0.@cmpMessage.2"/>
        </sdl:Operation.refInputMessage>
        <sdl:Operation.refTechnicalException>
          <sdl:MessageList xmi.idref="_0.@cmpMessageList.0"/>
        </sdl:Operation.refTechnicalException>
        <sdl:Operation.refFunctionalException>
          <sdl:MessageList xmi.idref="_0.@cmpMessageList.0"/>
        </sdl:Operation.refFunctionalException>
      </sdl:Operation>
    </sdl:Interface.cmpOperation>
  </sdl:Interface>
</sdl:Definitions.cmpInterface>
```

Figure 38 - TicketAgent SDL2.0 (XMI-SDL) - Interface

6.1.3 Java Interface

Figure 39 - TicketAgent Java shows a Java representation of TicketAgent service. The Java Interface may be generated automatically, in this example it is used only to better define the service. The real 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 Java methods will have an XML Document as parameter.

```
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 39 - TicketAgent Java

6.1.4 SDL Editor Snapshot

The Figure 40 - TicketAgent SDL Editor Snapshot presents a 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 had been selected and its Properties are shown in the frame at the bottom of the window. The SDL Editor uses a tree based interface.

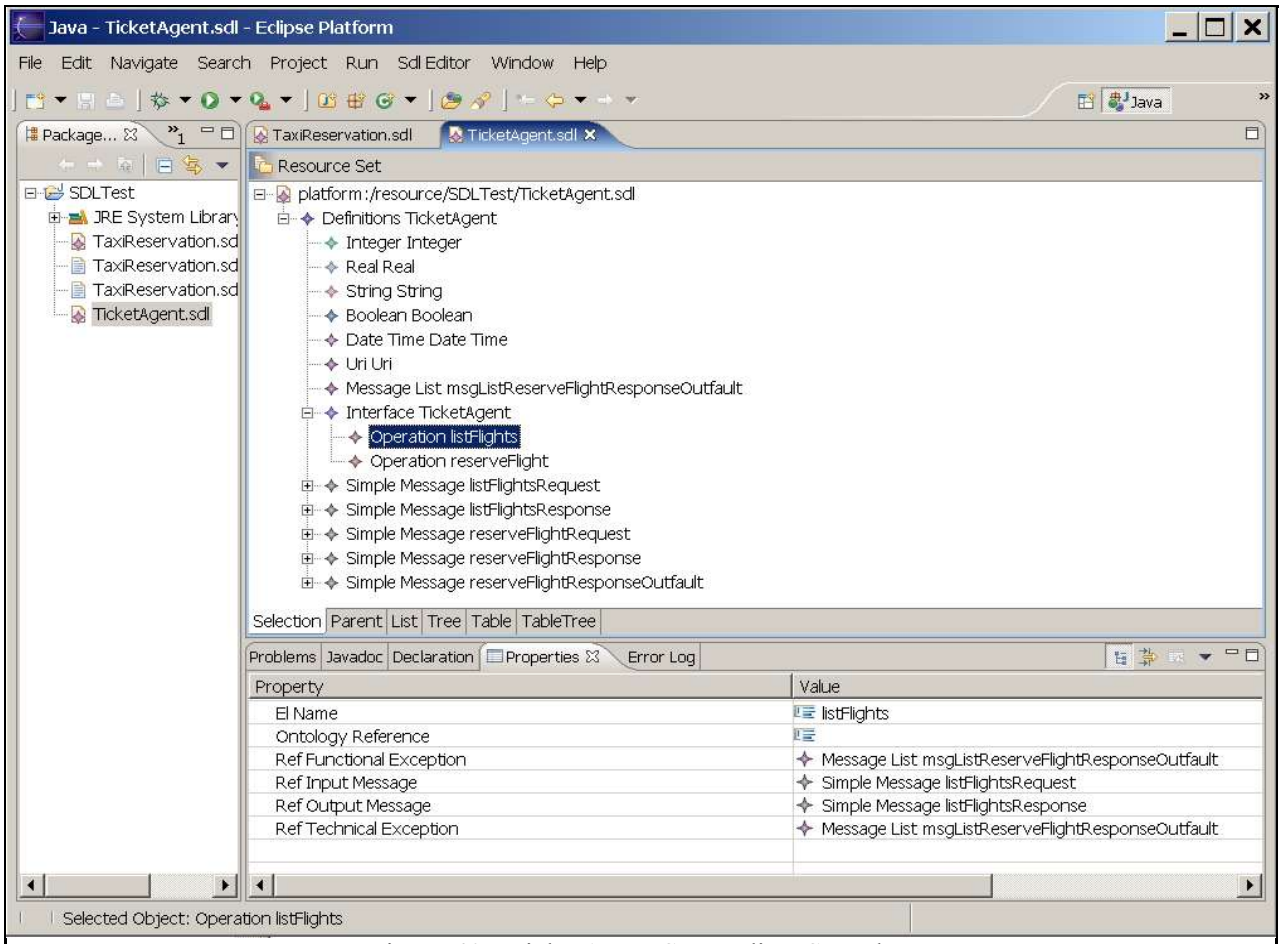

Figure 40 - TicketAgent SDL Editor Snapshot

7.Appendix

7.1 Glossary

Term	*Acronym*	*Synonymous*	*Obsolete*	*Description*
Atlas Transformation Language	ATL			ATL is a QVT implementation, a way to query, view and transform a metamodel into another metamodel.
Client				A Client is a software that makes use of a Web Service, acting as its 'user' or 'customer'.
Conceptual model				A conceptual model refers to the knowledge that a person has about the system and the way it should work.
Common Object Request Broker Architecture	CORBA			Is an OMG open architecture and infrastructure that applications can use to work together over networks.
Definitions Component				Is just a container for two categories of components; Message and Interface.
Feature Component				A feature component describes an abstract piece of functionality of the service like "reliability" or "security"
Interface component		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 Component				A Message is the basic unit of communication between a Service and a Client; data to be communicated to or from a Service as a single logical transmission.
Part Component		Message Part		A Part Component is the elementary part of a Message Component.
Property Component				A Property component describes the set of possible values for a particular property.
Query Views Transformation	QVT			OMG standard for Query, View and Trasform models
XSD		XML Schema		XML Schema Definition, a W3C Recommendation, specifies how to formally describe the elements in an XML document.

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

7.2 References

CORBA: CORBA® BASICS, http://www.omg.org/gettingstarted/corbafaq.htm
DBECoreArch: Pierfranco Ferronato, DBE Core Architecture Ver 01.3, 2004/06/04
HMAD: David Carlson, HyperModel analysis and design tool,
http://www.xmlmodeling.com/hyperModel/
IDL: OMG IDL, http://www.omg.org/gettingstarted/omg_idl.htm
MDA: David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, January 10, 2003
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/
U2M: UML2MOF Tool, Martin Matula, NetBeans / Sun Microsystems,
http://mdr.netbeans.org/uml2mof/
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

8.Open Issues

Investigating new SDL Requirements.
Orchestration vs choreography.
Common Language Metamodel introducing Versioning and Identification convention.

- end of document -