Centre for Environmental Data Archival

Date: 1st June 2010

Version: 0.0.1

Author: Maurizio Nagni

CEDA Geographical Enterprise Application

Table of Contents

i. Preface

This document specifies the features, included or planned, of the GEA web application. The main goal of GEA is to implement a fully complaint implementation of the OGC CSW specifications (vers. 2.0.2). Anyway as will be clear in the following pages the GEA architecture easily allows to expands the number of the supported interfaces.

There are several CSW implementation but only a few are distributed under an OpenSource licence:

- degree – University of Bonn (Tomcat, PostgreSQL/Oracle)
- eXcat Server – Dutch Geospatial Data Service Centre (Tomcat, eXcat)

but only the commercial ones are, at the moment, officially Certified OGC Compliant; at the same time while the commercial ones offer not only a JSP/Servlet support through Tomcat but also a full J2EE support through servers as JBoss or WebSphere allowing a seamless integration with the existing J2EE resources in the company.

With those premises GEA would like to approach the CSW implementation problem using a commercial-like point of view but exploiting its Open Source spirit. The actual GEA configuration includes Java 6, JBoss (J2EE, ESB) and, postgreSQL server as supporting database.

GEA General Architecture

Maintain multiple logic for each of the possible versions of a specification could soon became a nightmare if not a real spaghetti-code. For this reason the fundamental idea behind GEA is to digest the incoming request message in a GEA standard internal XML document format: this process is know as *NORMALIZATION* (see Fig. 1).

The NORMALIZATION uses an XSLT transformation to carry the incoming request to a GEA XML internal description and one or more XSD schema (defining the GEA internal data description) to validate the previous transformation.

The normalized XML is then transformed in a Java class which is suitable to be dispatched to the GEA logic module then, after this module, the response follows a reverse path.

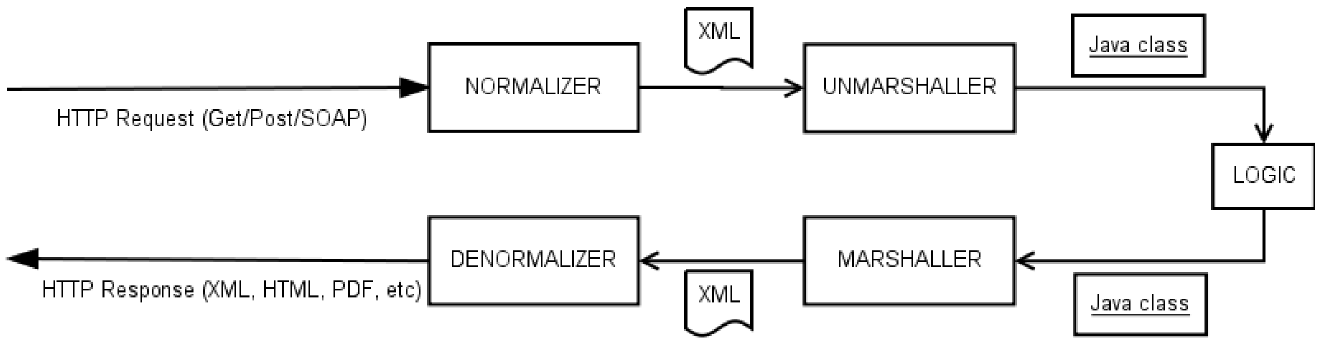

Fig. 1

A detailed description of the single technologies supporting each process step is:

- ***XML normalize/denormalize - validate***: Apache Xerces
- ***XML marshall/unmarshall***: JAXB (Java Architecture for XML Binding)
- ***Logic***: EJB (Enterprise Java Bean technology)
- ***Model**:* both the last two modules share java library containing all the XSD schemas involved in the specific GEA implementation. Such library is generated using the XML Java Compiler – XJC

All the application is deployed in the JBoss application server using three files:

- an EAR file containing all the EJBs
- an ESB file containing all the message pre/post logic processing
- an XML file defining the connection to the database(or databases)

All the project as been developed with Eclipse as IDE and MAVEN as source builder as well for the management of the custom created libraries.

Technical Specification

- Java 6
- JBoss 5.1.0GA with JBoss ESB4.8

Maven project under Eclipse

GEA Workflow

The request's workflow heavily relies on messages exchange (the XML documents) so to improve, as much as possible, the system flexibility the system should couple as weakly as possible each single process. Using the JBoss ESB technology GEA is able to transform/route/log/invoke, synchronously or asynchronously, any kind of incoming request message because all the messages dispatched among the ESB container are wrapped in a standard form.

GEA uses the ESB's features to pass from one module to another not only the various XML documents, but also a small set of parameters, actually related just to the OGC specifications, which allow the module to identify the kind of received message:

group: represents the specific service (CSW, WMP, etc)
subgroup: GET, POST, SOAP (other?)
operation: the specific operation required for the specified group
originalVersion: the message version of the incoming request
acceptedVersion: specific for the GetCapability operation

A few other parameters are used internally but are necessary mainly to plug one module into the next one but these are well like to be reduced in future.

The workflow defined in the ESB container is composed by two services: the *HttpGateway* and the *ServiceExecutor* (actually the last service is repeated for each group/operation but in future a unique instance will be called using the appropriate parameters).

The Gateway listener intercept the HTTP requests (actually intercept only the GET requests) and forward them to the MarkHttpMethod which extract the basic informations to exploit the request: group, subgroup, operation, oroginalVersion, acceptedVersion. Then the message with the extracted parameters is dispatched to a JMS (a message queue).

Has to be noted that a proper handling of the exception is still under development but the general idea is to use the message container to transport the raised exception up to the response action where the proper output will be formatted.

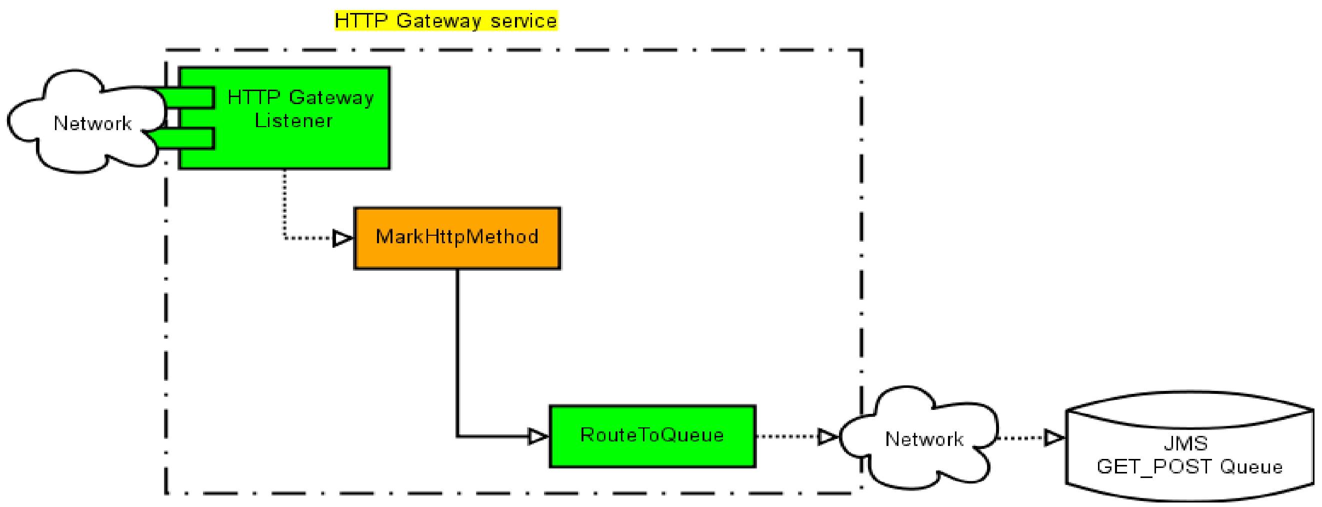

Figure 2

The message is then processed in the ServiceExecutor (see Fig. 3). The ServiceExecutor owns a specific listener on the JMS queue where the previous service has its output, which acts as a filter extracting only messages with specific parameters. At the side of each method call are visible the parameters (configured through the ESB configuration xml) required for its execution; is worth to note that only the values in the brackets must be configured for each specific service/method.

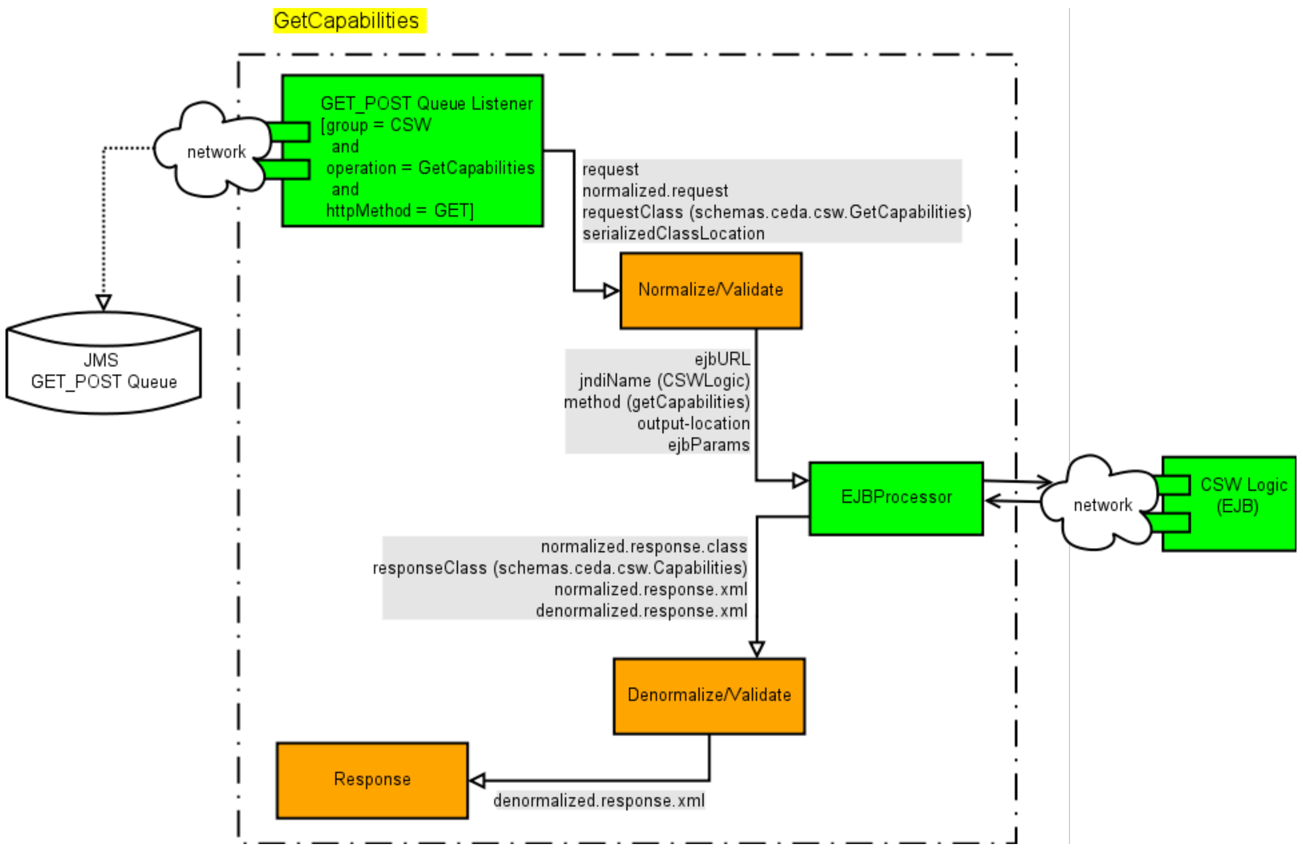

Figure 3

Normalize/Validate structure

The normalization operation (see Fig.4) involve two external resources: a database table which contains the relevant information for each transformation and an external file server where to download the XSD schema necessary to validate the executed transformation.

The transformation action relies on an helper class (TransformerHelper) which acts as cache in order to minimize the access to the DB and maximize the retrieving of the required transformation. At the startup the helper load all (using an EJB) the transformation in the DB and create all the possible paths from one version (per service/operation) to another, eventually binding multiple transformations one after another (an policy to update the cache has to be implemented in future).

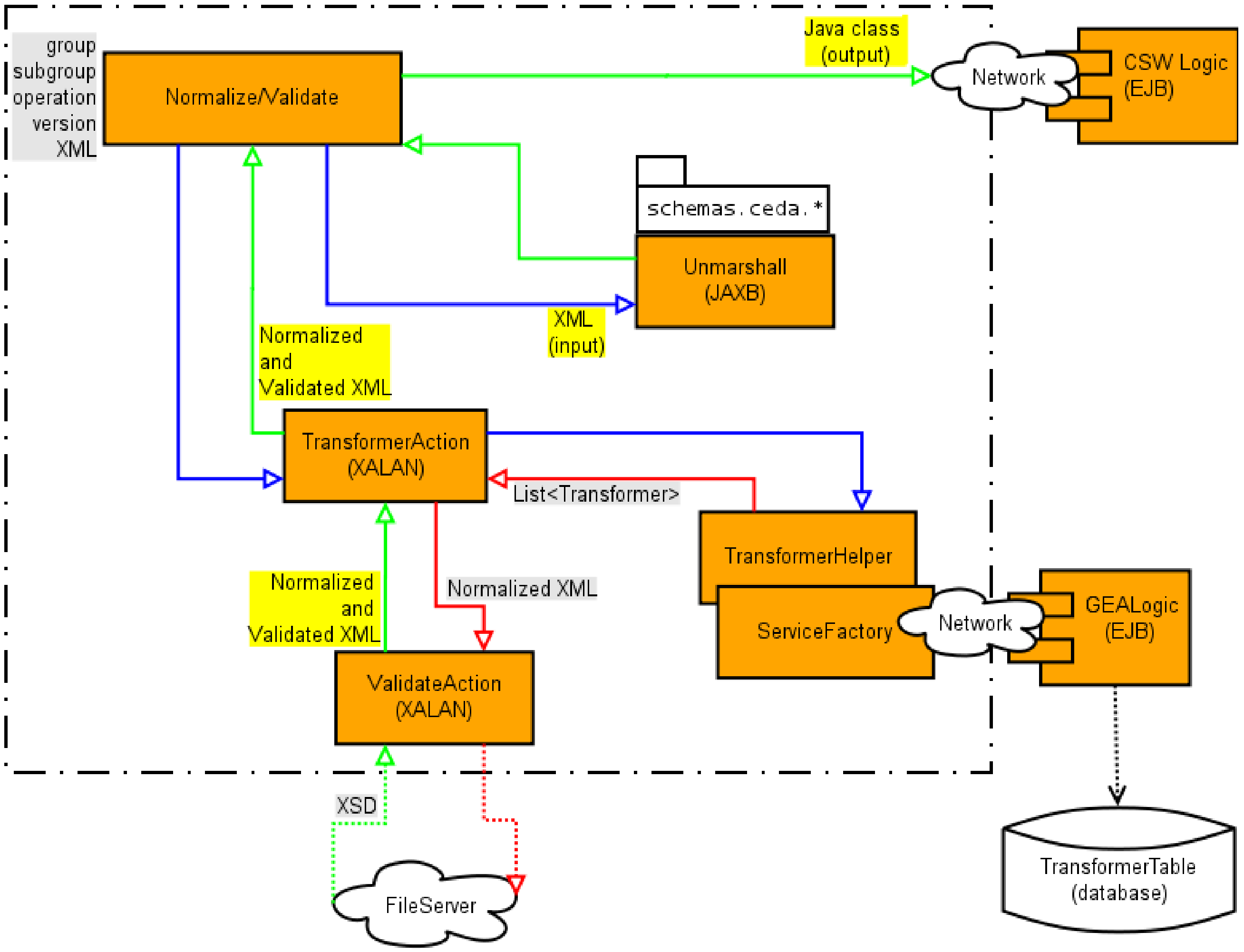

Figure 4

If the transformer succeeds then the message is validated (see *Validation* paragraph) marshalled to a Java class (using the Model) and dispatched to the CSWLogic. The Denormalize operation follows a similar (reverse) structure.

Validation

The validation is done per transformation (for each single transformation) not just on the result of the final transformation because, in general, GEA is aware only about the schema which are related to its logic. For the previous reason, the Transformer objects owns also a list of URL from where is possible to download the XSD necessary to validate the each transformation.

Actually all the XSD are available also through a file server. This solution has been preferred to a postgreSQL based one because all that schema, could be thought as the core for a department-wide reference schema library, that is, to allow other CEDA applications to share records between using a common collection of schema.

As last note is worth to note that according to the CSW specifications for the GetRecords

> The outputSchema parameter is used to indicate the schema of the output that is generated in response to a GetRecords request. The default value for this parameter shall be:
>
> http://www.opengis.net/cat/csw/2.0.2
>
> indicating that the schema for the core returnable properties (as defined in Subclause 10.2.5) shall be used. Application profiles may define additional values for outputSchema, but all profiles shall support the value
>
> http://www.opengis.net/cat/csw/2.0.2.

and this fit naturally with the transformation workflow (obviously adding some info in the ESB message wrapper)

The CSW operations

The OGC specification (vers. 2.0.2) require that the following operations must be available in any CSW server (in the brackets is defined which HTTP method is mandatory by CSW specifications) :

- ***GetCapabilities (GET)***: allows CSW clients to retrieve service metadata
- ***DescribeRecord (POST)***: allows a client to discover elements of the information model supported by the target catalogue service. The operation allows some or all of the information model to be described.
- ***GetRecords (POST)***: The primary means of resource discovery in the general model are the two operations search and present. In the HTTP protocol binding these are combined in the form of the mandatory GetRecords operation, which does a search and a piggybacked present.

Actually even such operations are available but there is a limited support both for their requests and their responses (GEA needs to full fill all the possible request/response parameters). Moreover, actually, the CSWLogic fills the response object with dummy data. Obviously such situation will change in the near future.

DescribeRecord

The DescribeRecord has to return the description of one, or all, the information models, goal which is exploited returning one XSD schema for each of the possible record structure. To implement such feature the CSW OGC standard requires the custom data schema to be wrapped into an CSW XSD abstract element: the ***csw:AbstractRecord.*** Extensible type is csw:DCMIRecordType(include the Dublin Core Metadata Initiative). Not extensible types are and the csw:Record, csw:SummaryRecord, csw:BriefRecord.

The Listing 1 is shown how this is implemented on a real situation for one element named *http://schemas.ceda.rl.ac.uk/medin:BriefRecord*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema id="medin-record" targetNamespace="http://schemas.ceda.rl.ac.uk/medin"
        xmlns:medin="http://schemas.ceda.rl.ac.uk/medin" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:csw="http://schemas.ceda.rl.ac.uk/cat/csw" elementFormDefault="qualified" version="2.0.2">
        <xsd:import namespace="http://schemas.ceda.rl.ac.uk/cat/csw" schemaLocation="http://localhost/csw/record.xsd"/>
        <xsd:element name="MedinBriefRecord" type="medin:MedinBriefRecordType" substitutionGroup="csw:AbstractRecord"/>
        <xsd:complexType name="MedinBriefRecordType">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                                A (very!) brief description of our records
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:complexContent>
                        <xsd:extension base="csw:AbstractRecordType">
                                <xsd:sequence>
                                        <xsd:sequence>
                                                <xsd:element name="Authors" type="xsd:string"/>
                                                <xsd:element name="Abstract" type="xsd:string"/>
                                                <xsd:element name="OriginalFormat" type="xsd:string"/>
                                        </xsd:sequence>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>
        <xsd:element name="MedinRecord" type="medin:MedinRecordType" substitutionGroup="csw:AbstractRecord"/>
        <xsd:complexType name="MedinRecordType">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                                A (very!) brief description of our records
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:complexContent>
                        <xsd:extension base="csw:DCMIRecordType">
                                <xsd:sequence>
                                        <xsd:sequence>
                                                <xsd:element name="Authors" type="xsd:string"/>
                                                <xsd:element name="Abstract" type="xsd:string"/>
                                                <xsd:element name="OriginalFormat" type="xsd:string"/>
                                        </xsd:sequence>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>
</xsd:schema>
```

Listing 1

This simple custom model could represents as well the base for a query or the result of a query as explained in the next paragraph.

From the GEA point of view this mean that the XSD description has to be available in the Model in order to output its schema with the DescribeRecord, marshall the class (i.e. filled by the CSWLogic) and as response after a GetRecords request and allow the CSWLogic to manipulate the request/response.

GetRecords

Input model and output model are two different concepts in CSW but they could be represented by the same entity.

The Input Model

The input model (one or more) is used to define the structure of the Query element. When the client requests a GetRecords operation, among the several other parameters, MUST specify the csw:Query@typeNames attribute which from the OGC specifications (OGC 07-006r1):

> [...] is a list of one or more names of queryable entities in the catalogue's information model that may be constrained in the predicate of the query.

In our simple example could be (note *medin* as XML namespace) could be just

typeName= medin:BriefRecord

anyway is worth to note that supposing the existence of different kinds of records (i.e. medin:BriefRecord, medin:SummaryRecord, medin:Record) the input type should be the most general one.

More complex is the syntax of the query which has to be sent with the GetRecord. The <csw:Constraint> element can contains two possible child: <csw:CqlText> (see Listing 2) or <ogc:Filter> (see Listing 3)..

```
<Constraint>
        <CqlText>prop1!=10</CqlText>
</Constraint>
```

Listing 2

```
<Constraint>
	<ogc:Filter>
		<ogc:Not>
			<ogc:PropertyIsEqualTo>
				<ogc:PropertyName>prop1</ogc:PropertyName>
				<ogc:Literal>10</ogc:Literal>
			</ogc:PropertyIsEqualTo>
		</ogc:Not>
	</ogc:Filter>
</Constraint>
```

Listing 3

To adapt this syntaxes to an existing service an Adapter class, from the CSW query syntax to the external service query syntax, has to be implemented at the CSWLogic level (that it in the EJB package). In a real case, to extract data from the existing Medin Discovery Service the following steps are necessary:

1. Precompile (as explained in the normalize/validate paragraph) in a specific library the Medin schema
2. Add the request/response Medin schema to the schema file server (necessary to validate the result of the marshaller)
3. Write an Adapter class which can translate the <csw:GetRecords> to a Discovery Service's <medin:doSearch> (the namespace is a problem discussed later)
4. Write (or update an existing one) a Dispatcher to dispatch the request to the DiscoveryService.

The output model

The output model is used to define which schema should be used to represent the result of the query.

The csw:GetRecords@outputSchema attribute is used to indicate the schema of the output that is generated in response to a GetRecords request (http://www.opengis.net/cat/csw/2.0.2 as default, or http://schemas.ceda.rl.ac.uk/medin in our example).

Optionally a csw:GetRecords/Query/ElementSetName can specify which particular subset should be used (brief, summary or full) or, alternatively, a more detailed output specification can be build using the csw:GetRecords/Query/ElementName.

Still missing

- ***Application core***
  - Exception handling
  - GetRecords Query parsing

- ***Application specific***
  - Create a database defining the GEA instance properties. Because it is a base for the GetCapabilities, that is, for CSW, WFS, WMS etc.
  - Verify if is possible to use the csw:Record as base for the CEDA existing dataset (i.e. MEDIN) or is worth to define a customized one

- ***Development environment related***
  - Code's Licence Declaration Header (“`This program is free software; you can redistribute it and/or modify.... blah..blah...`”)

  - packages naming: actually there are two main packages
    - ceda.service.* (code)
    - schemas.ceda.* (XSD compiled java library)

  - schemas namespace: actually are all under http://schemas.ceda.rl.ac.uk
    - http://schemas.ceda.rl.ac.uk/ows
    - http://schemas.ceda.rl.ac.uk/csw
    - ....

Technical Specification

- Java 6
- JBoss 5.1.0GA with JBoss ESB4.8
- Maven project under Eclipse