SIXTH FRAMEWORK PROGRAMME

MESOR

Management and Exploitation of Solar Resource

D.2.1 – "Web Service Tutorial"

Project/Contract no.: 038655

Date of preparation of the document: July 15, 2008

Version	Date	Changes made	by	Sent to
1.0	08/12/2008	First Official Version	ARMINES	MESoR Members
1.1	14/01/2009	Version 1.1	ARMINES	MESoR Members

Table of Contents

1 Introduction

1.1 Foreword

This tutorial intends to help the partners of the MESoR project to build, set-up, and deploy Web Service providing therefore a standard mean to access their inner resources. Resource denotes any data, data set, relational database, application… that a provider may wish to offer to the community.

In the framework of the MESoR project we have already selected standard components (platform, programming language, protocols…) for supporting web services development and thus, this tutorial does not claims to be exhaustive regarding all possible means to build Web Services. The justification of the choice of those components has already been given during the project and approved by the MESoR consortium. In addition, several of them have been selected for the realisation of two of the three portals built by the GEOSS Architecture and Data Committee. All the components used in MESoR are based on Open-Source software and respect as much as possible international existing standards.

1.2 Web Service Strategy

Before beginning to develop your Web Service, you must select the resource(s) that you want to provide to the community. The figure below explains where the different components of a Web Service "ecosystem" stand. The three red boxes represent from right to left:

1. the data or resource that you own and want to share,
2. the Web Service components that your are going to build in this tutorial (box #4),

The Client that will exploit your Web Service. This client will be built by ARMINES based upon your Web Service description.

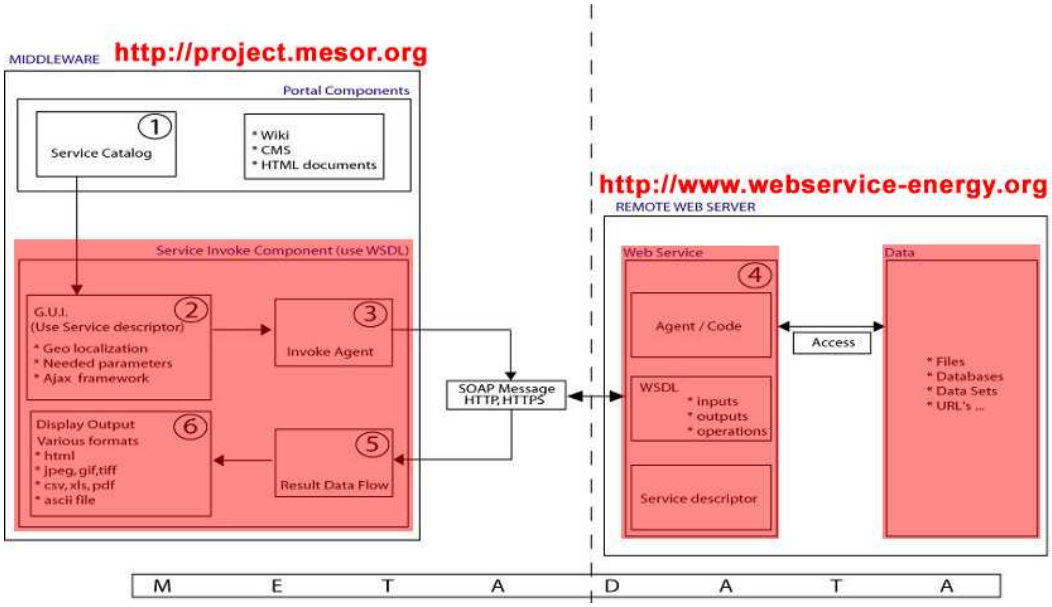

Figure 1 Description of the various components of the MESoR information system and Web Services

In a distributed approach, Web Services can be set-up in different manners. For example, you might want to both develop the Web Service and operate the needed platform (application server) that ensures the visibility and persistence of the service. Conversely, you might only wish to build the Web Service component, but do not want to operate the application server. Whatever your choice you must ensure that the resource is accessible for the given case. For example, if the resource is not allowed to be accessed remotely through the network by a Web Service, you will have to build and operate this Web Service at your own premise. If the resource can be remotely accessed, you have both choices:

1. build, deploy and persistently operate your Web Service at your own premise,
2. build deploy and test your Web service and then send the Web Service archive to ARMINES which will deploy and host it on its own platform at www.webservice-energy.org.

The matrix below summarizes the various approaches that are offered in the framework of MESoR.

	Type of access to the resource	**Build and test Web Services**	**Deploy and persistently operate Web Service**
Case #1	Local only	At your premise	At your premise
Case #2	Possibly remote	At your premise	At your premise
Case #3	Possibly remote	At your premise	At webservice-energy.org

Table 2

Let give examples to illustrate these cases. Assume you own a database that you want to share. The format of your database and the software used for the management have no importance at all. In order to exploit the database, you certainly have an application that extracts data. What is denoted by resource is this application. This application can be natively a Web service; otherwise, a Web service is developed to encapsulate this application. When invoked, a Web service must exchange messages in certain formats with the Web, e.g. SOAP and HTTP formats (Fig. 1). The application server (remote Web server in Fig. 1) is in charge of these processes of encoding/decoding/exchanging messages. In Case #1, the Web service can only be invoked by your application server. In Case #2, it can be invoked remotely and also by your application server(. In Case #3, you do not operate an application server; the Web service archive is stored in a warehouse (e.g., webservice-energy.org) from which it can be invoked. Note that in each case, your computer is requested to perform the access to your database.

2 Requirements

In order to be able to install the needed components for this tutorial you must have a basic understanding of Unix/Linux.

Windows Operating Systems (OS) are also candidates though we have not tested it.

All the command lines written down in this tutorial refer to a Linux OS.

2.1 Operating System

The selected Operating System (OS) for this tutorial is Ubuntu a Linux based operating system. You can download this OS at: http://www.ubuntu.com/

Most of the Unix/Linux OS are compatible. The choice is up to you.

2.2 Application Server Jboss

The Application Server for testing and deploying your Web Service is Jboss. For this tutorial, we have used the version Jboss Application server 4.2.2.

Go to http://www.jboss.org/ and under the "*Download*" section select the "Jboss Application Server" link.

Select the latest stable version 4.2.2.GA with which I wrote this tutorial.

2.3 Java JDK (Java Development Kit)

We currently need one version of Java (JDK 6) for operating Eclipse and one version of Java (JDK 5) for operating Jboss. As stated on the Jboss web site JDK 5 is required to run Jboss AS 4.2 and therefore we recommend to use JDK 5.

Note: If you plan to let ARMINES hosts your Web Service you must follow the two Java JDK approach.

For the JDK 6:

On the Sun java web site http://java.sun.com download Java JDK1.6 or JDK 6.0 (same version different name).

To do so select *Java SE* (Standard Edition).

Select *JDK 6 Update 6* (at the time of writing this tutorial) or a higher version.

Select "*Linux*" as Platform and let" *Multi-language*" as default language.

Finally select "*Linux self-extracting file*" "jdk-6u6-linux-i586.bin".

For the JDK 5:

Select the "*Previous Releases*" tab link under the Java SE page.

Select "*J2SE 5.0 Downloads*" link.

Select JDK 5.0 Update 15 (at the time I'm writing this tutorial) or a higher version.

Select "*Linux*" as Platform and let" *Multi-language*" as default language.

Finally select "*Linux self-extracting file*" "*jdk-1_5_0_15-linux-i586.bin*".

2.4 Eclipse IDE (Integrated Development Environment)

Go to the Eclipse Web site http://www.eclipse.org

Follow the "*Download Eclipse*" link and select "*Eclipse Classic 3.4 (151 MB)*" link.

3 Component installation

3.1 Install Jboss

Open a terminal window, go to your home directory and create a new folder eg. "*www*" .

Go in your home directory:

```
mesor:~$ cd
```

Unzip the Jboss archive in this folder. "*Desktop*" is the place where the archive has been stored after download. This may vary according to your browser's download preferences.

```
mesor:~$ unzip /Desktop/jboss-4.2.2.GA.zip
```

You must have now a folder called "*jboss-4.2.2.GA*" .

3.2 Unzip the JDK's

In order to unzip the JDK's (5 and 6) that come as bash script file (.bin extension), you must run the shell interpreter "*bash*" command.

```
mesor:~$ bash Desktop/jdk-6u6-linux-i586.bin
```

Do the same for the JDK 5.

3.3 Gunzip and Untar Eclipse

```
mesor:~$ tar –xzf Desktop/eclipse-SDK-3.4-linux-gtk.tar.gz
```

3.4 Get plugins for Eclipse

Go to the eclipse folder that has been created. In the command line window run eclipse with the following options.

```
mesor:~/eclipse$ ./eclipse –vm ~/jdk1.6.0_07/bin/java
```

where "*–vm*" is the Virtual Machine option that indicate witch java version to use.

At the Eclipse prompt for the workspace, leave it as default.

You should now get the Eclipse window like this one:

Figure 3

Select the “Workbench” icon and you get the following Eclipse window:

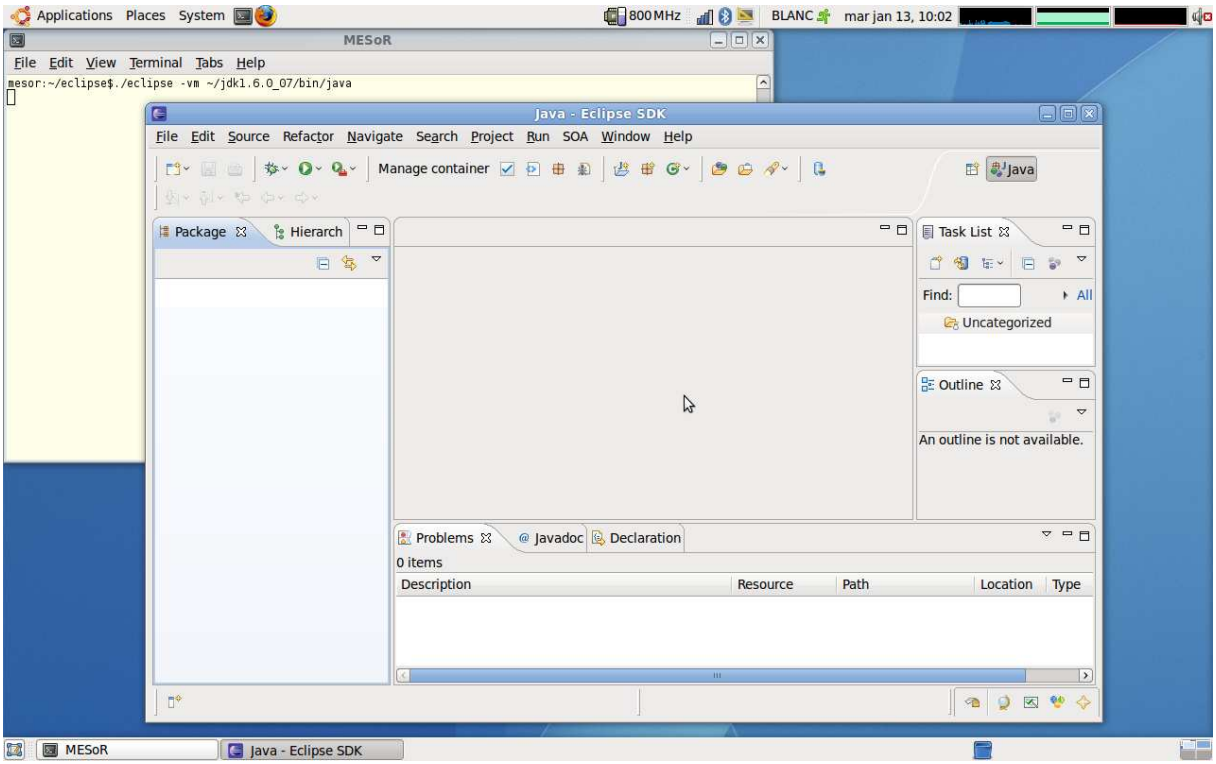

Figure 4

Before you start using Eclipse you need to download extra plugins needed for writing Web Services.

Select the "*Help/Software Update…*" menu.

Select the "*Available Software*" tabs at the top of the window.

Click on the "*Ganymede Update Site*" arrow. If this name does not exist, you can add it with the button "Add Site" under the location

"http://*download.eclipse.org/releases/ganymede*".

Wait until modules are displayed.

Then tick on the "*SOA Development*" and the "*Web and Java EE Development*" boxes.

Finally click on the "*Install*" button.

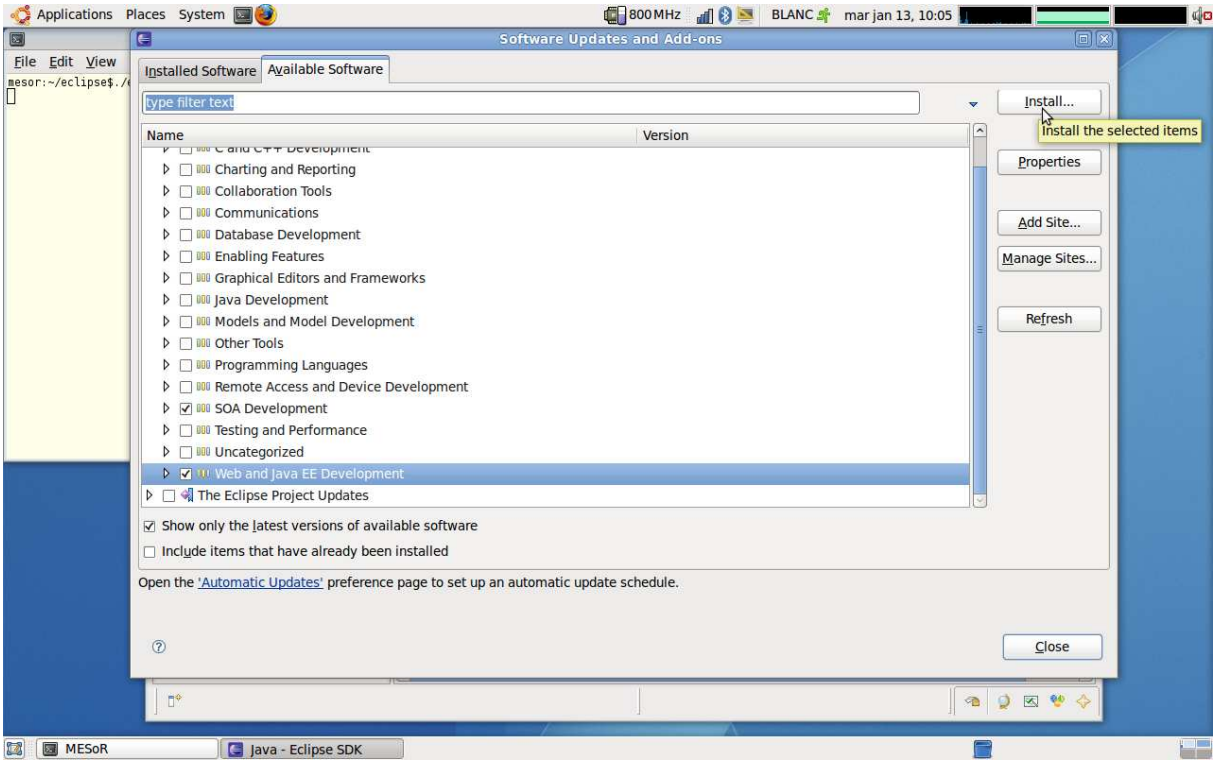

Figure 5

Then Eclipse display the list of the package that will be installed.

Finally accept the licence and click on the "*Finish*" button.

The installation of the package will start. This may take few minutes.

You might be asked to restart the system.

3.5 Configure Eclipse

First run Eclipse

```
mesor:~/eclipse$ ./eclipse –vm ~/jdk1.6.0_07/bin/java
```

Click on the "*Workbench*" icon.

The default view is "Java" (button on the top right) and we are going to set some preferences.

Select "*Windows/Preferences*" menu.

In the list on the left select "*java/installed JREs*" and tick the "*jdk1.5.0_15"* item.

Eclipse already has the JRE 6 as default and we need to add the JRE 5 to the list in order to be able to write our Jboss compliant web service.

Click the "*Add*" button.

Select the "*Standard VM*" JRE Type.

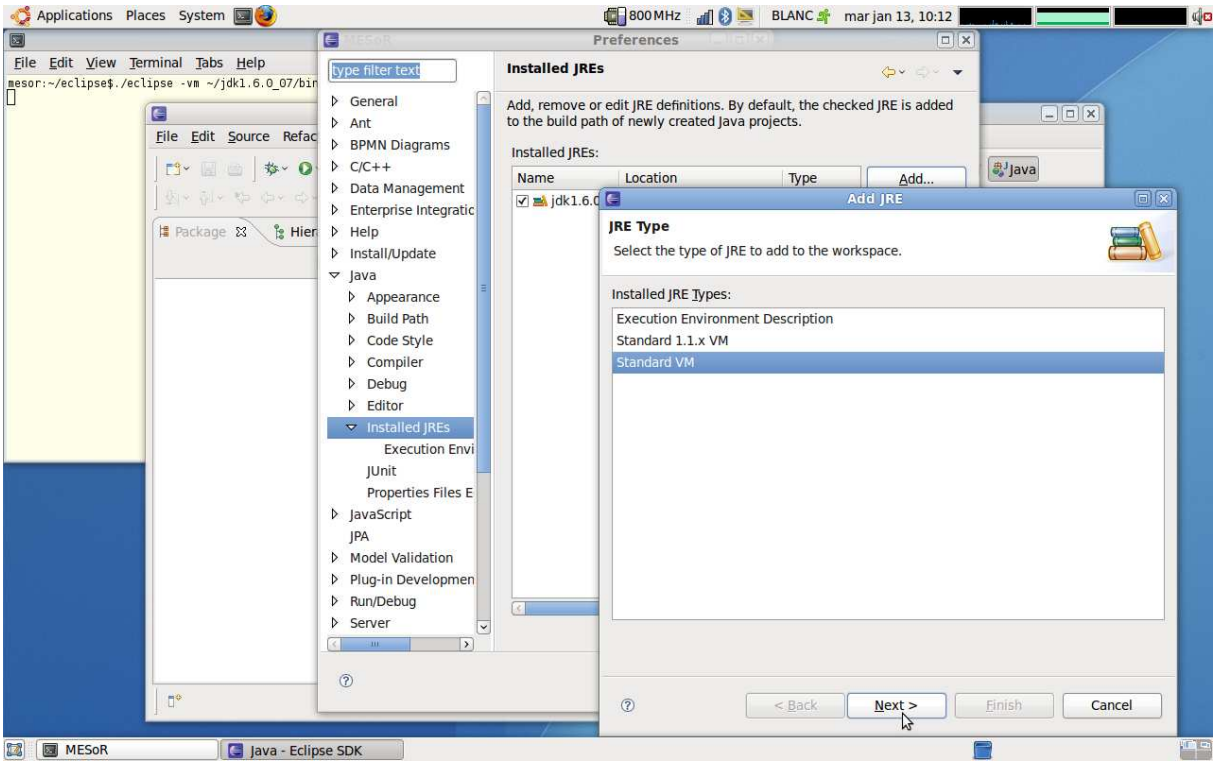

Figure 6

Click "*Next*".

Click on "*Directory*" and select the home directory of the JDK 5.

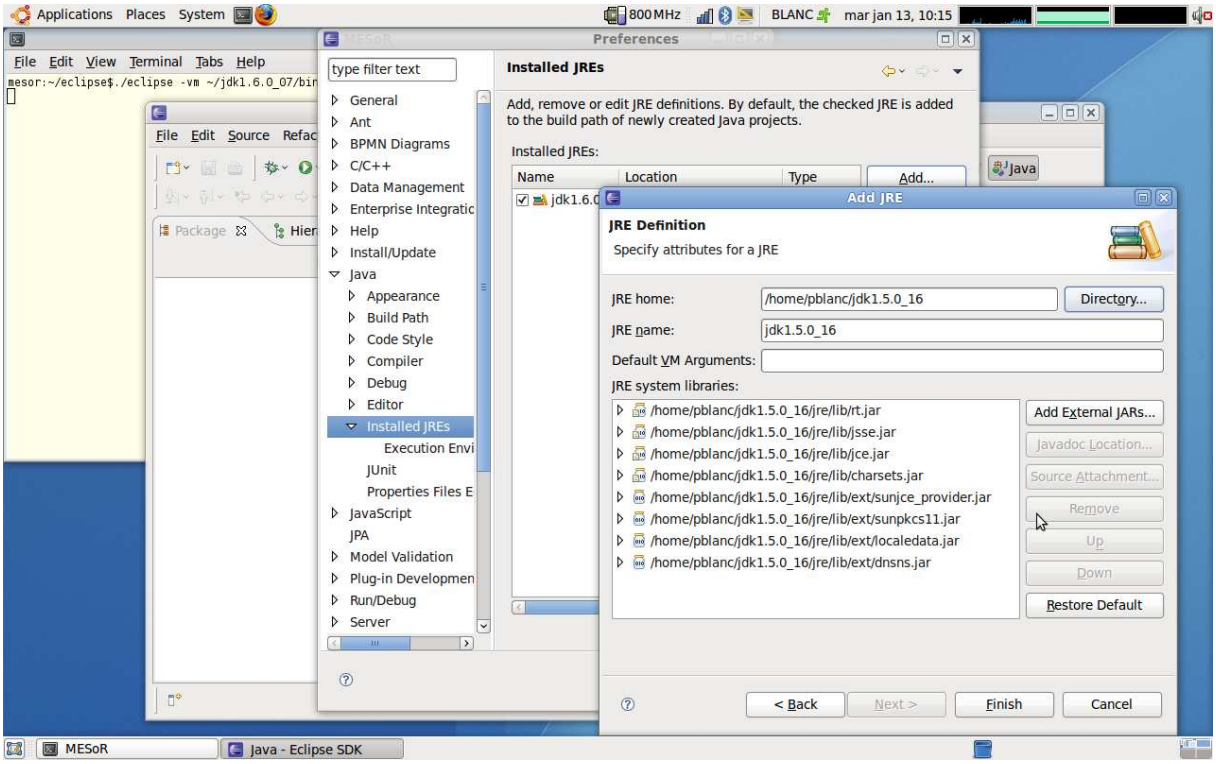

Figure 7

Once the JRE system libraries have been loaded, click the “Finish” button.

Now you must have two JRE in the list of installed JRE’s.

Select the JRE5 as default as we are going to develop Web Services.

Now under the “*Java*” arrow menu on the left, select “*Compiler*” sub-menu.

Set the “*Compiler compliance level*” to “*1.5”* in the top right option box.

Click the “*Apply*” button and validate with “*yes*” at the windows pop-up.

Under the “*Server*” arrow menu on the left select “*Runtime Environment*” sub-menu.

Click the “*Add button”*.

In the list of “*Server/Euntime Environment*”, locate the “*Jboss*” folder and select “*Jboss V4.2*”.

Click the “*Next*” button.

Let the JRE as “*Default JRE*”.

Click on the “*Browse*” button to select the “*Application Server Directory*” .

Locate and select the “*jboss-4.2.2.GA*” folder.

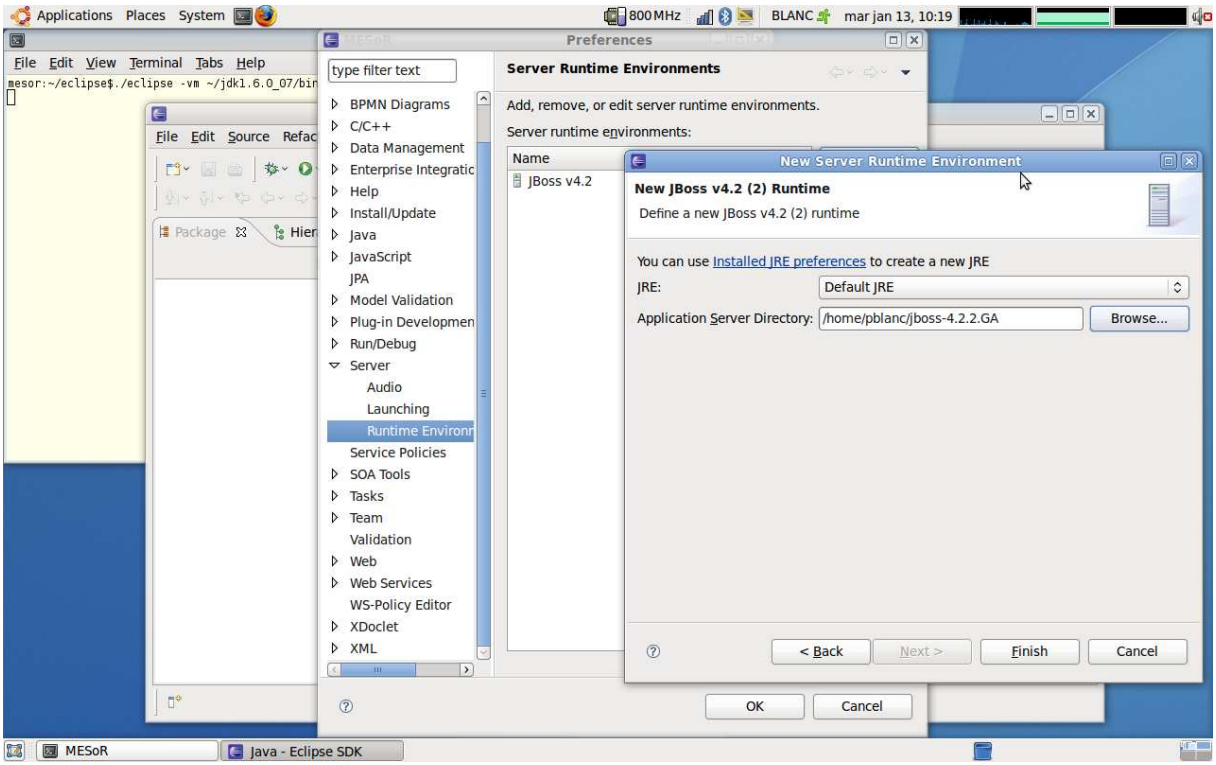

Figure 8

Click on the “*Finish*” and then on the “*Ok*” button.

Congratulations, your Eclipse environment is set-up.

Close eclipse to make sure all your configuration options will be saved.

4 Web Service Creation Process

It is important to recall that at that stage of the tutorial you should already have identified the resource that you want to provide by the mean of Web Services approach. In this tutorial we have selected a resource that copes with the case #3 of the Table 1 as an example.

This resource is accessible through a request to a web server:

http://www.helioclim.net/com/ncep_climate.php

4.1 Create WSDL file

We are going now to create the structure of the WSDL file by using Eclipse.

Run eclipse as usual:

```
mesor:~/eclipse$ ./eclipse –vm ~/jdk1.6.0_07/bin/java
```

Once you have the Eclipse default windows, select a new "*Perspective*".

Click on the "*Perspective*" icon  on the top right of the window.

Select "*Other/Java EE*".

Now create a new project:

Select the top menu "*File/New/Dynamic Web Project*".

Make sure that the configuration "*Custom*" is set on Java 5.

To do so click on the "*Modify*" button and set the "Java" Project Facets" to 5.0.

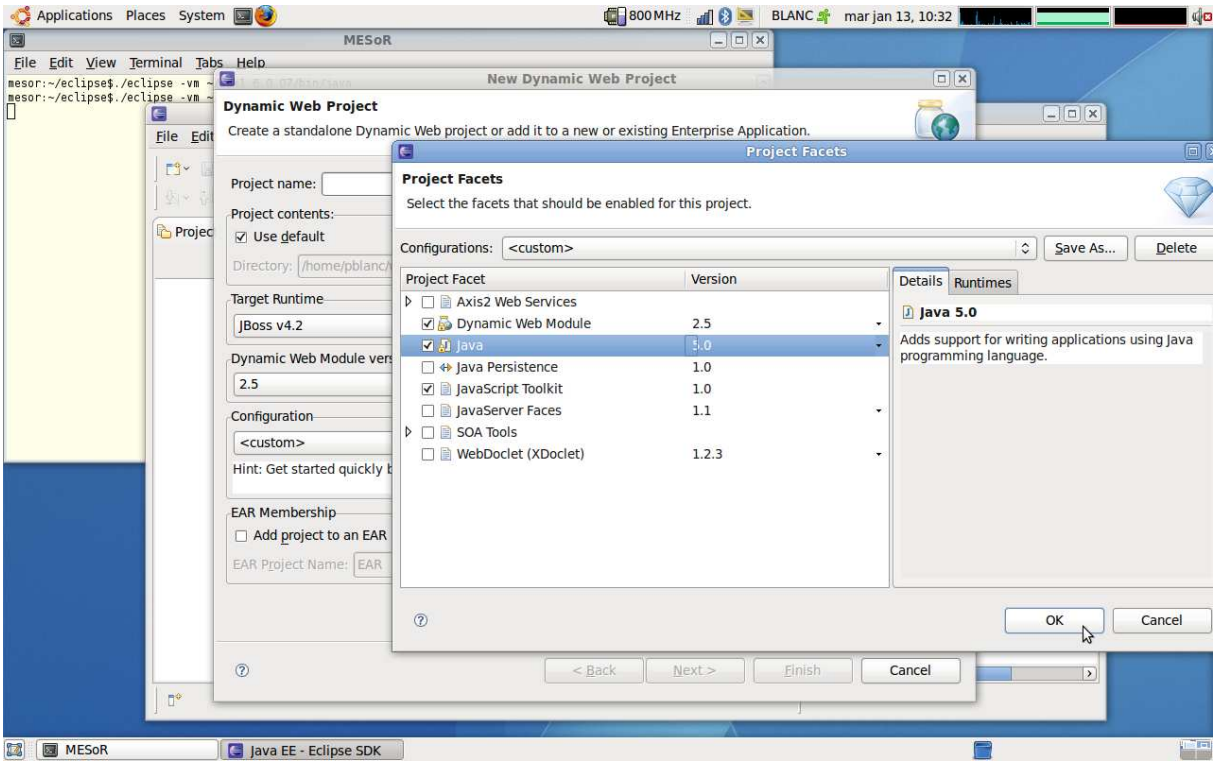

Figure 9

Click “OK”.

Enter the name of your project (“*EMPClimate*” in our case).

Click “Finish”. You should end-up with a screen like this:

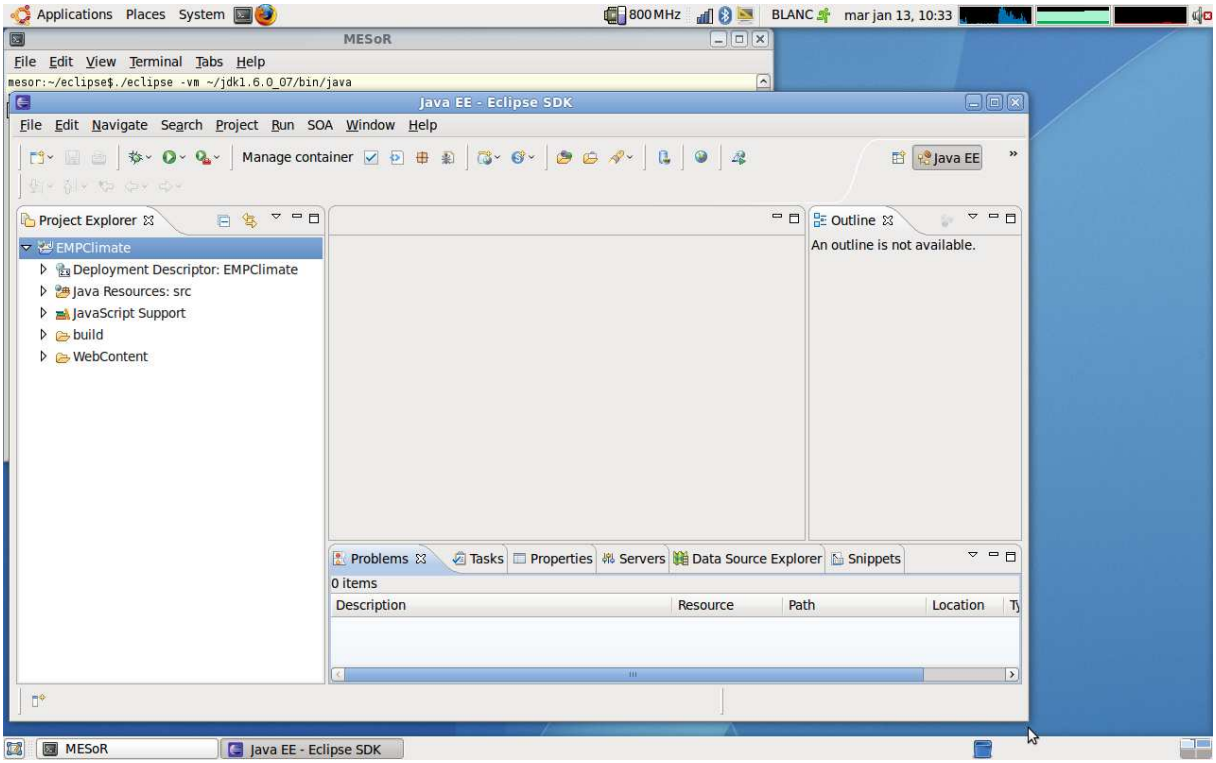

Figure 10

Now let's create the WSDL file of the Service.

Right click in the "*WebContent/WEB-INF*" arrow folder on the left side of the window.

Select "*New/Folder*" and name it "*wsdl*".

Once the "*wsdl*" folder is in the Project Explorer left window, right click and select:

"*New/Other*" .

In the list of the "*Select a Wizard*" window select "*Web Services*" folder and "*WSDL*" item.

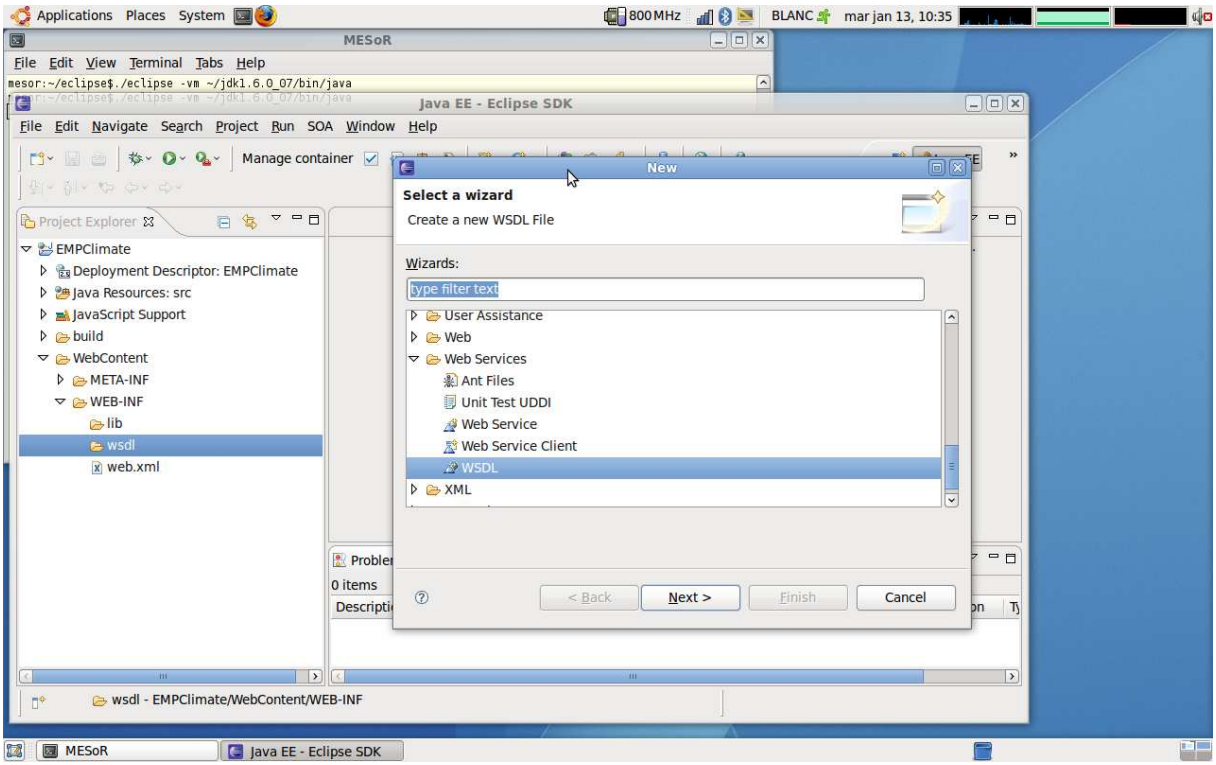

Figure 11

Click "*Next*" and enter a name for the WSDL file. In our case "*EMPClimate.wsdl*".

Click "*Next*" and change the "*target namespace field*" to a name that recalls the resource that you are going to provide. Any name is possible it is just easier for everyone if you do so. In our case *http://www.soda-is.com/EMPClimate/*.

Check the "*rpc literal*" option in the "*SOAP Binding Options*" radio button list. This is the only encoding message currently supported by Jboss.

Click the "*Finish*" button.

You should end-up with a screen like this displaying the your newly created WSDL file.

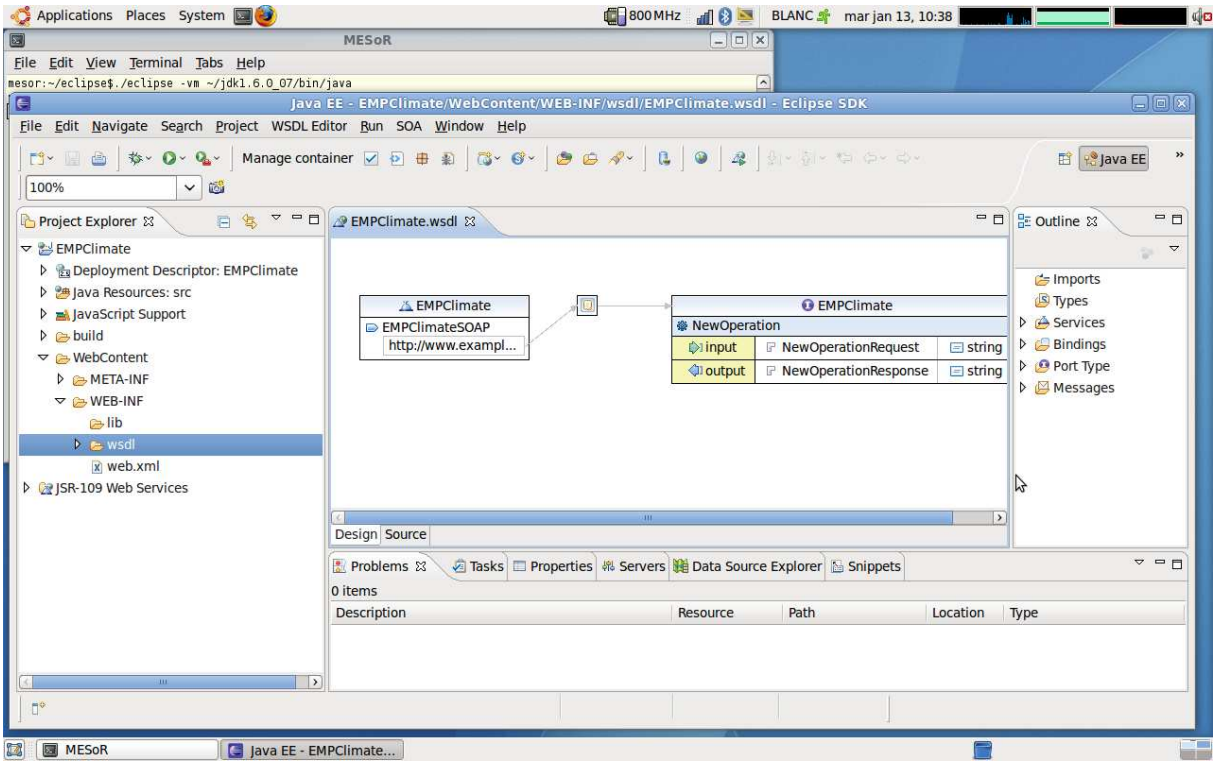

Figure 12

If we take a closer look at the graphical representation of the components of this WSDL file, we have:

- on the left, the service and port definition,
- on the right, the interface including the inputs outputs definition and the operations,
- in the middle, the binding between the service and the interface.

As both components have by default the same name eg. "*EMPClimate*" we are going to rename them:

The service (left box) became "*EMPClimateService*".

The interface became "*EMPClimatePortType*".

You should end up with a screen like this:

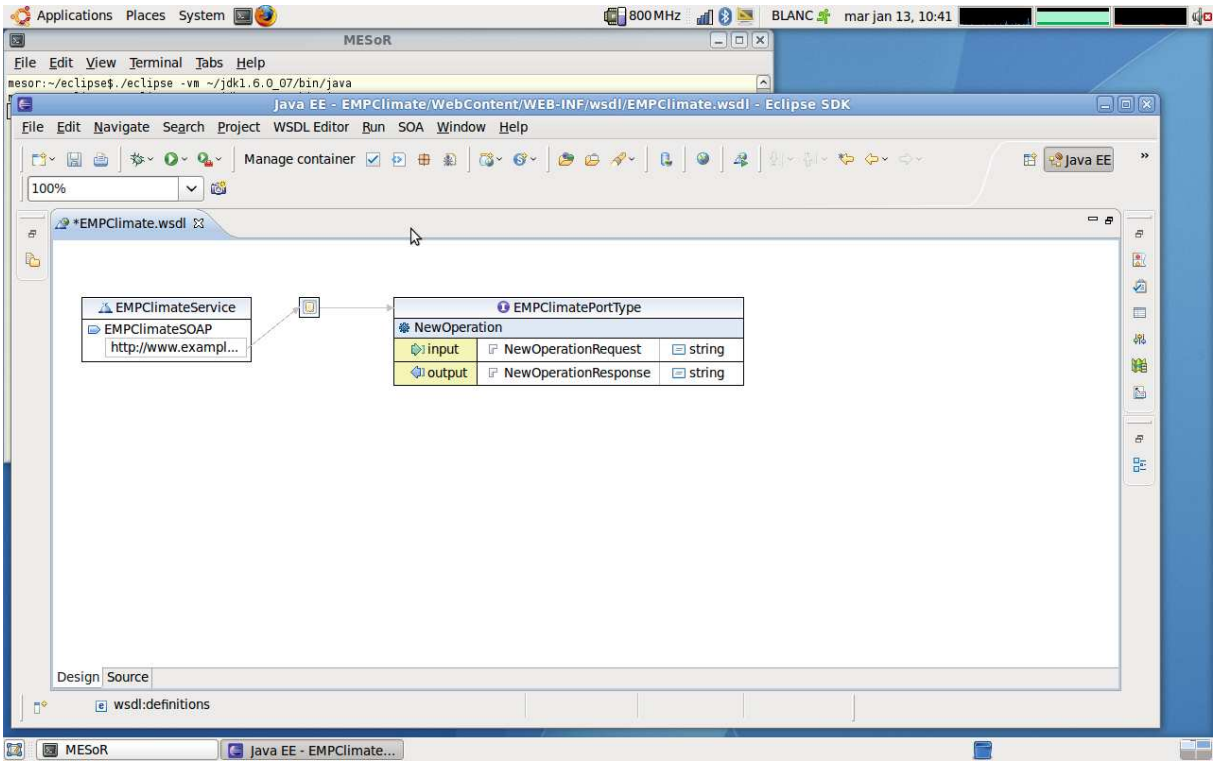

Figure 13

On the "*EMPClimatePortType*" box, operations are defined by default. We are going to change those operations to match the ones that come out from our resource.

To do so in our example (not mandatory) we will use a standard approach by using a XSD (XML Schema Definition) schema that provides prescriptions on types for Web Service definition. This schema has been provided to you with this tutorial. It can also be downloaded at: http://www.soda-is.com/schemas/ where you have to select the corresponding XSD schema.

This schema is already defined and we are going to import it in our project.

Select the "*wsdl*" folder.

Click on the top menu "*File/Import*".

Select the "*General/File System*" folder.

Click the "*Next*" button.

Then locate where is your XSD Schema with the top "*Browse*" button.

Leave options as is.

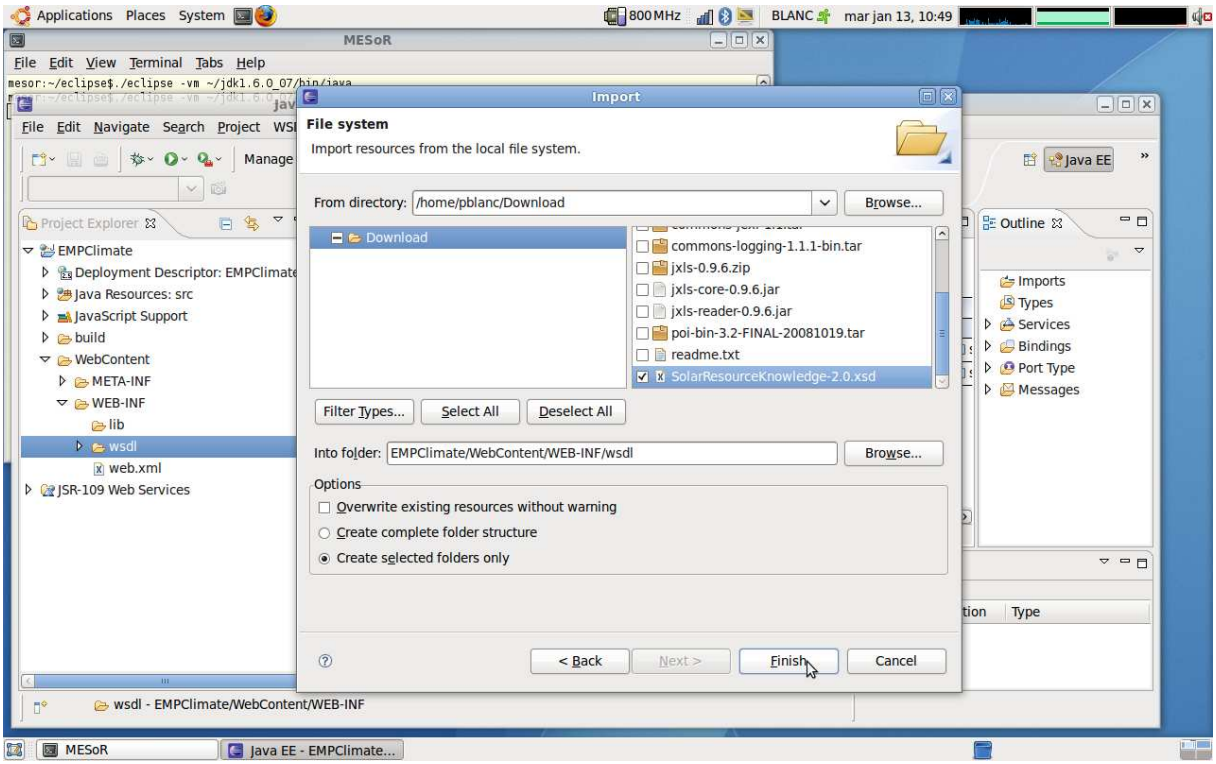

Figure 14

Then click on the “Finish” button to load the file in your project.

You should see now the XSD Schema file in the “*wsdl*” folder of your “*Project Explorer*” .

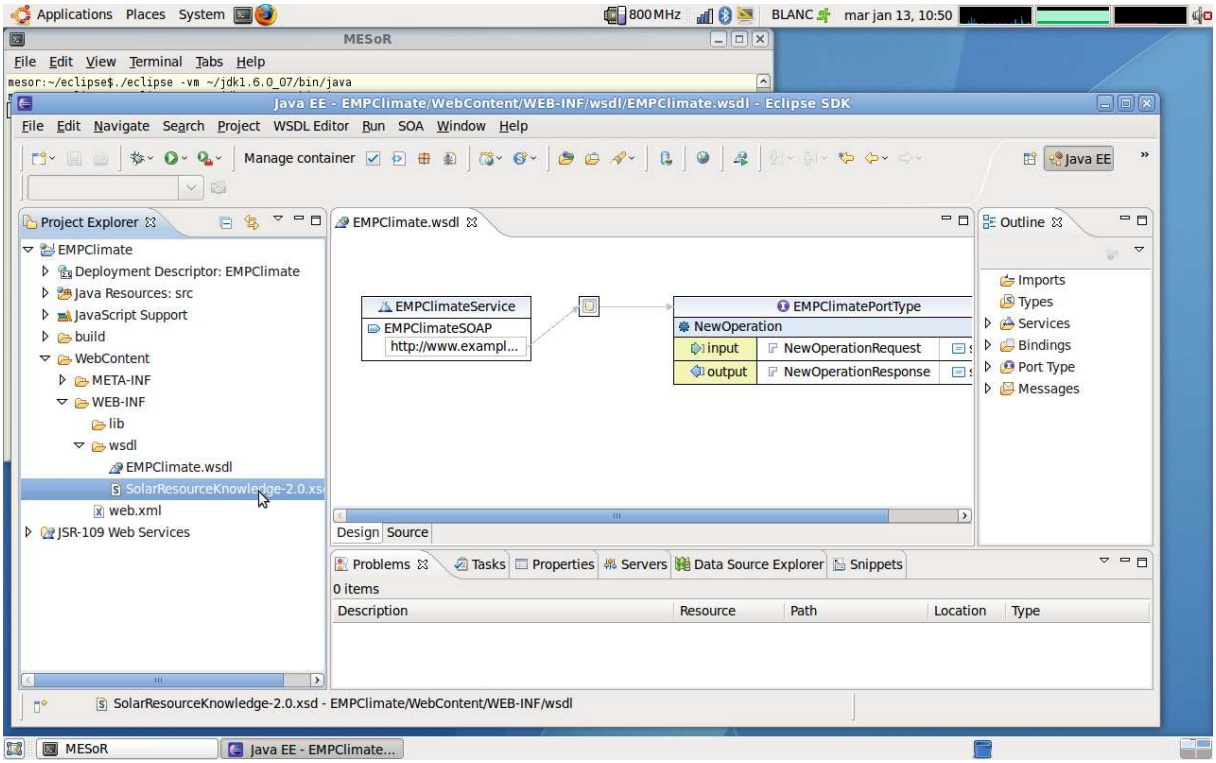

Figure 15

We will now use this Schema for defining the various type of our Web Service ensuring therefore a standard approach of type definition.

Lets go back to the "*EMPClimatePortType*" box.

We are going to tie the XSD Schema to the type definition of our Web Service.

Lets start by changing the name of the operation on the "*EMPClimatePortType*" box.

Change "*NewOperation*" with "*GetAllValues*".

Note that the "*input*" and "*output*" parameters names are changed on the fly according to the new operation name.

Now we are going to change the type of the input and use the XSD Schema.

Select the "*string*" value in the "*EMPClimatePortType*" box.

Click on the "*Properties*" tab in the bottom tabs list.

In the "*Type*" menu of the "*Properties*" tab instead of the default value "*string*" select "*New*".

Enter the new type value name. In our case "*GetAllValuesRequestType*".

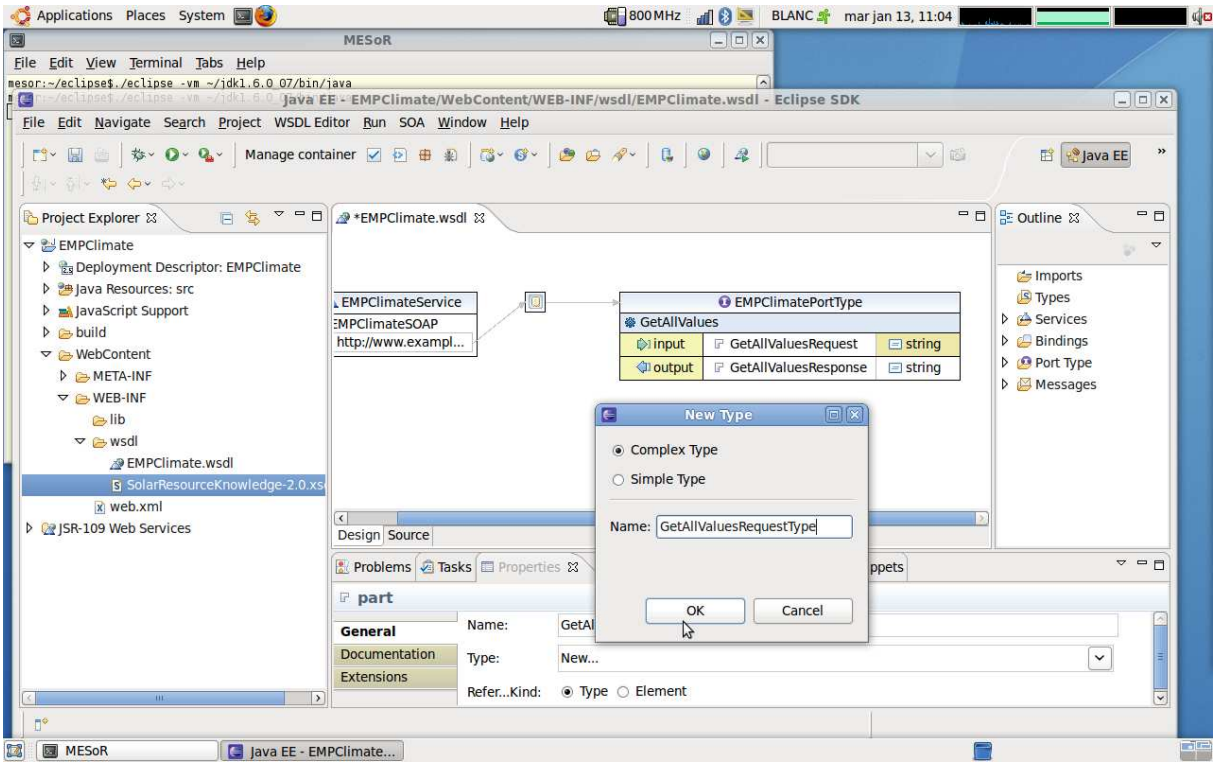

Figure 16

Tick the "*Complex Type*" option if not already set and click the "*OK*" button.

You should now be able to view the detail of the Service by clicking on the blue arrow on the right of the "*EMPClimatePortType*" box.

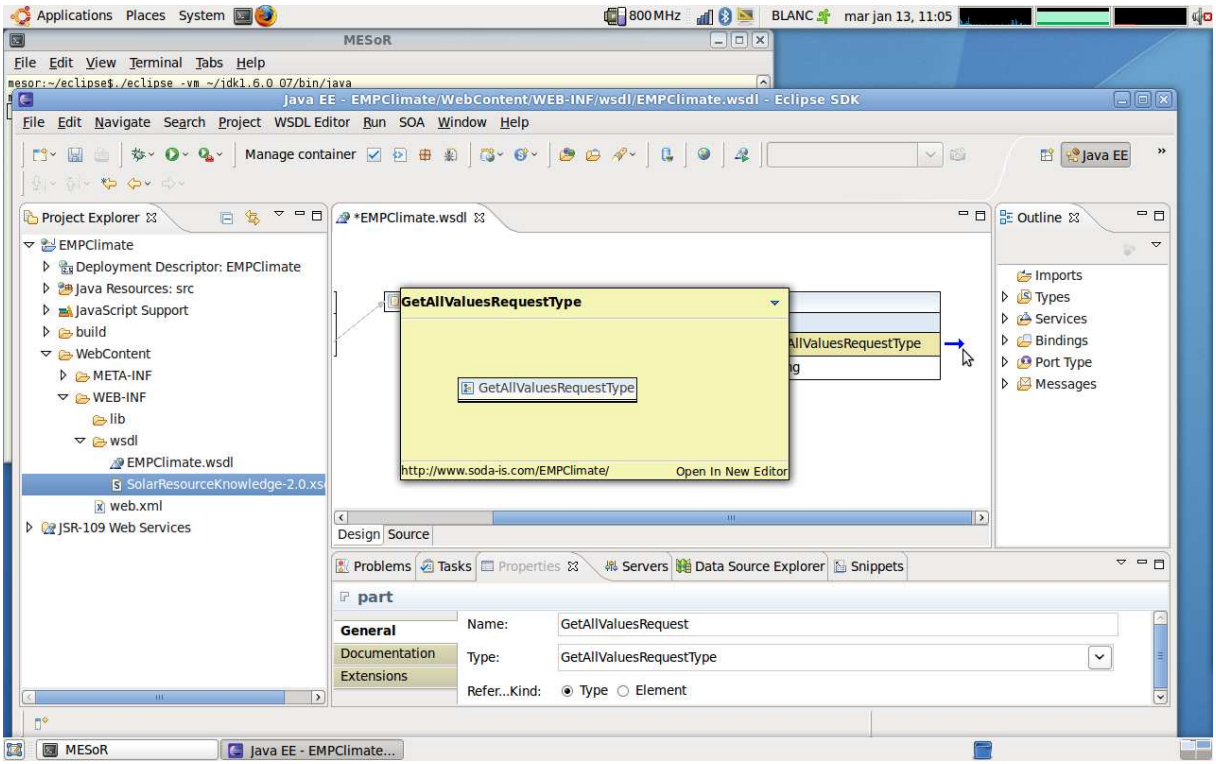

Figure 17

This will open a new tabs call "*Inline Schema of EMPClimate.wsdl*".

Now we are going to change the default type by an existing one that comes from the XSD Schema.

Select the "*Inline Schema of EMPClimate.wsdl*" tab.

Right click on the "*GetAllValuesRequestType*" box and select "*Add Element*".

Select the "*NewElement string*" line of the object and change the type of this element by clicking in the bottom "*Type*" menu and by selecting the "*Browse*" option instead of the default "*xsd:string*" option.

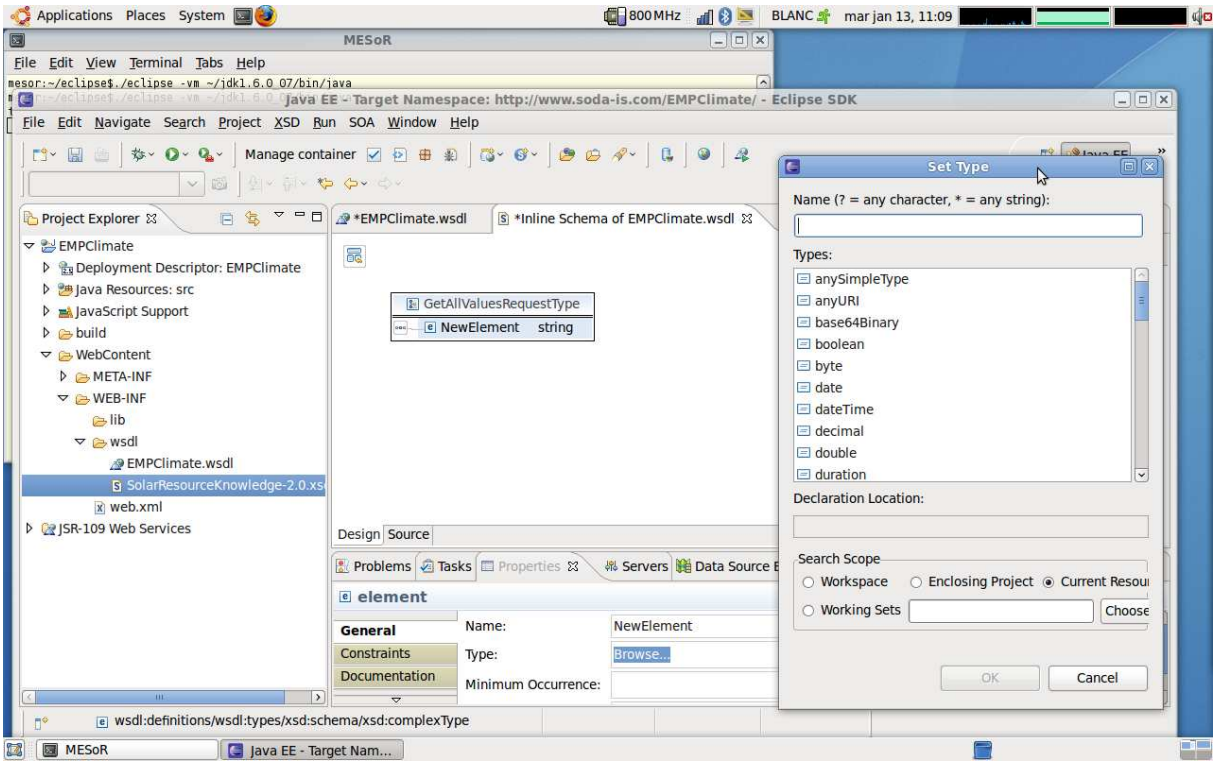

Figure 18

Now select the "*Enclosing Project*" radio button at the bottom of the pop-up window.

In the upper "Name" form start typing "geo…" and the auto completion will provide you with the full name that are available in the XSD Schema that match your name (*ie. geopointType*).

Click the "*Ok*" button to add the "*geopointType*" type to the selected object. You can see that the values "*latitude*", "*longitude*", "*elevation*" defined as "*floats*" are correctly bind to the "*geopointType*" element.

Rename the "*NewElement*" parameter name of the "*GetAllValuesRequestType*" box into "*geopoint*".

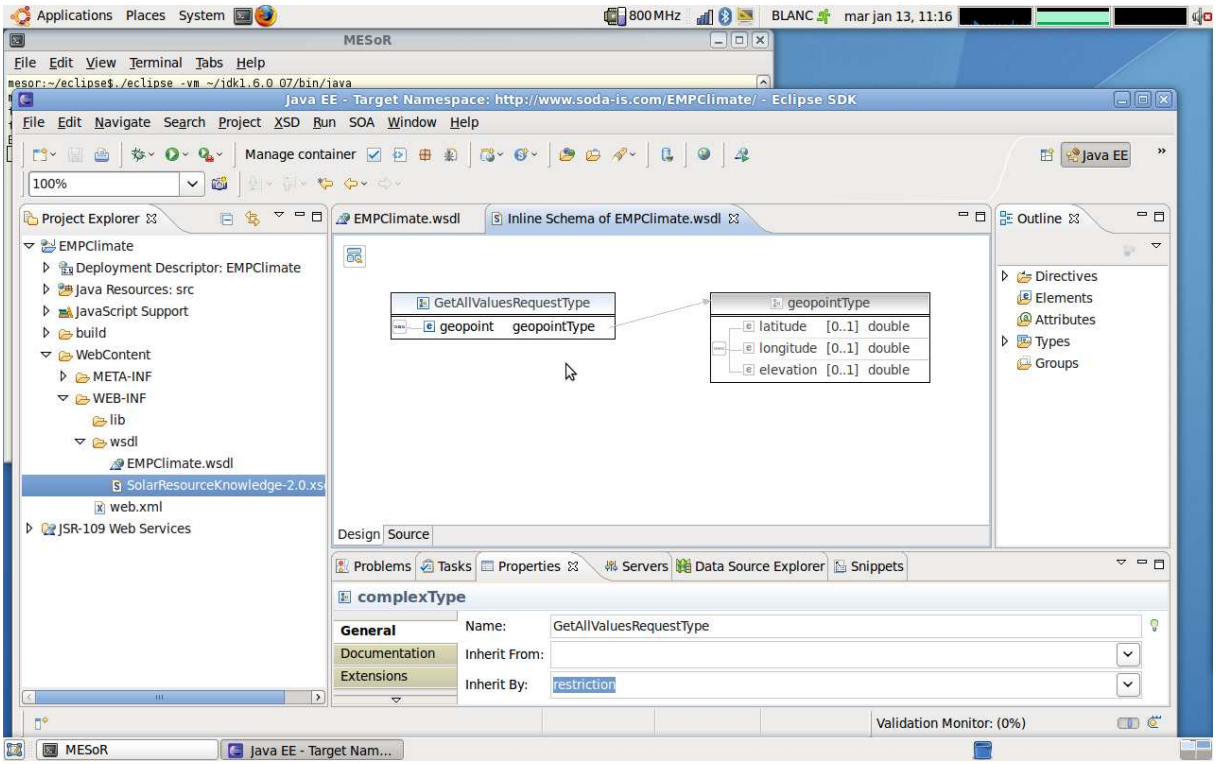

Figure 19

Save your works by clicking on the "*Save*" (or Ctrl + S) icon in the top menu.

Now you have to do the same for the "*output*" values of the "*EMPClimatePortType*" in your "*EMPClimate.wsdl*" file.

Change the current type "string" by a "New" (*Complex Type*) one called "*GetAllValuesResponseType*".

Click on the blue arrow to open the "*Inline Schema of EMPClimate.wsdl*" tab.

Then right click on the new type and select "*AddElement*".

Change the element name from "*NewElement* to "*ghi*". In our example this new type will handle the Global Horizontal Irradiance values coming out from the resource.

In the bottom "*Properties*" tab, change the "*Type*" from "*xsd:string*" to "*sequenceOfObservationType*" coming from the XSD Schema. To do so, click on the "*Type*" bottom menu, select "*Browse*…" and try the completion of the "Name" with "*seq*". Select the item "*sequenceOfObservationType*" in the rolling list.

Note that this time you do not have to click the "*Enclosing Project*" toggle button to find the type by completion. This is because the XSD Schema has already been loaded once when we selected the type for the input value.

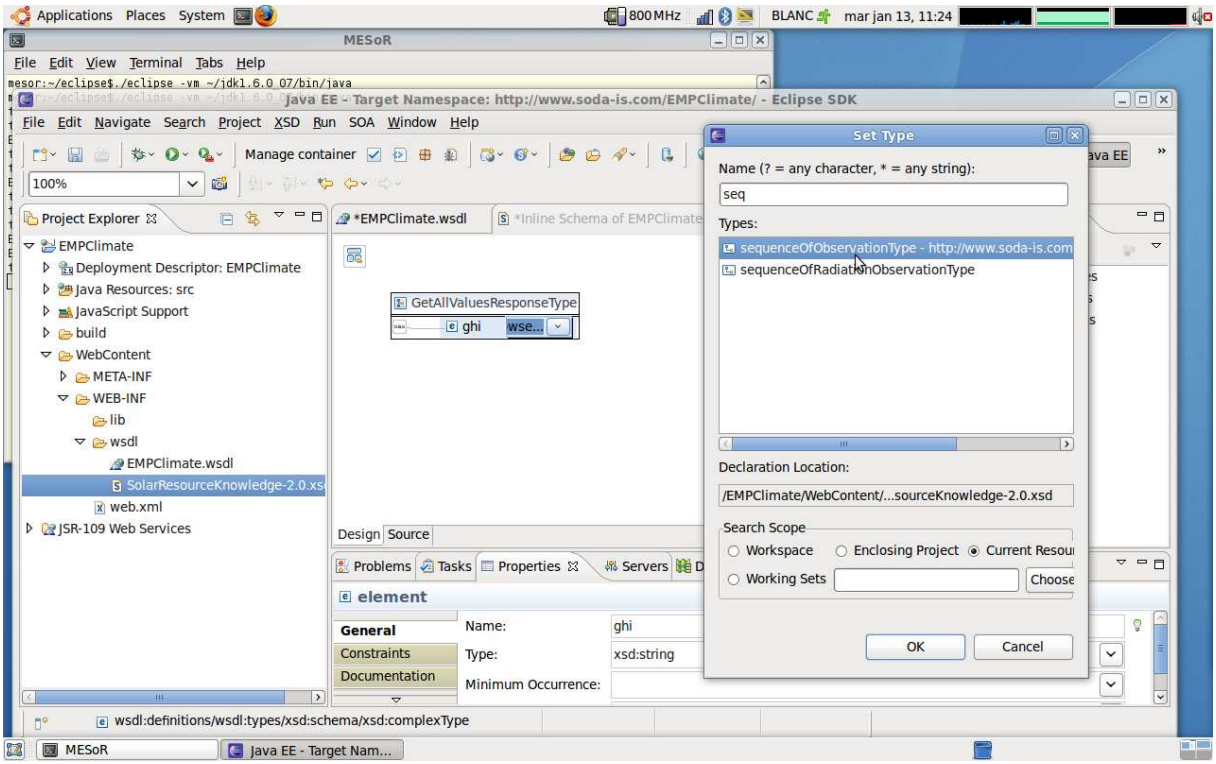

Figure 20

You should end-up with a screen like this, showing the first two outputs objects and type of your Web Service.

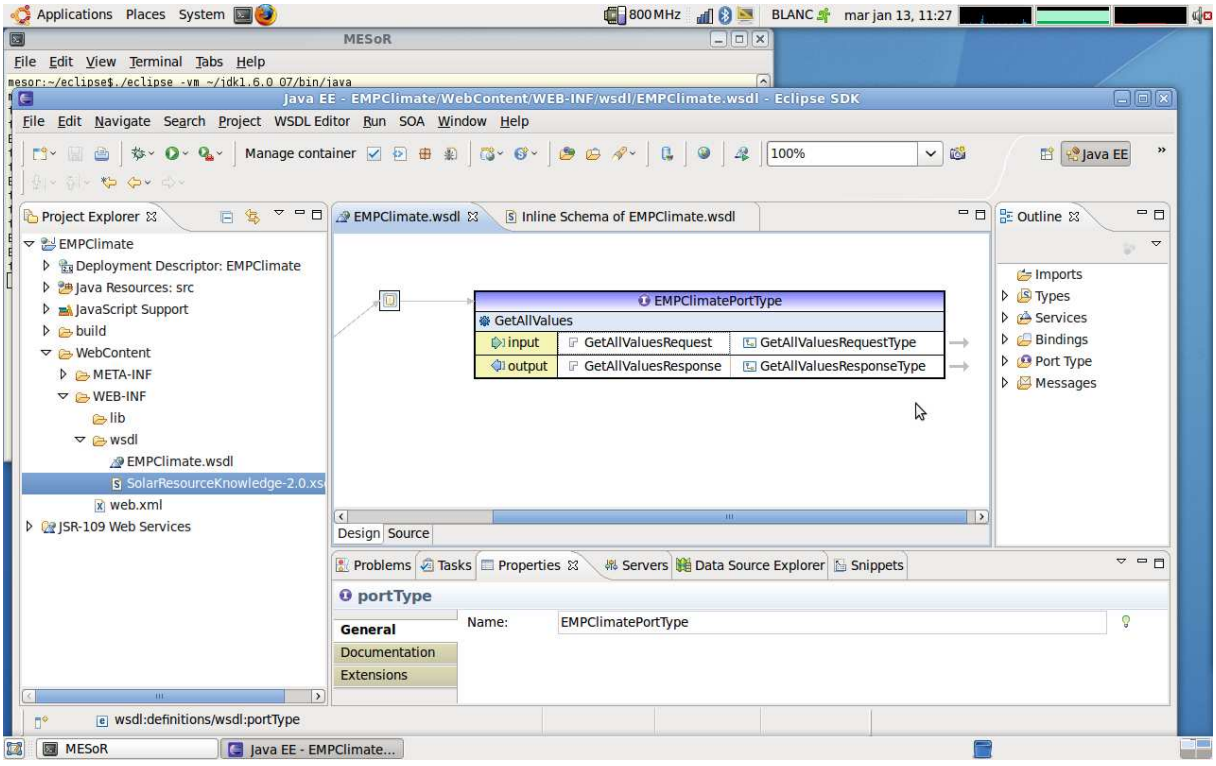

Figure 21

We know that our resource is accessible on-line (case #3):

http://www.helioclim.net/com/ncep_climate.php?latlon=10,40

The output of the resource gives the following data flow for latitude 10° and longitude 40°:

	Lat.	Long.	Alt.	Jan.	Feb.	Mar.		Oct.	Nov.	Dec.	Yearly Mean
GHI	10.00	40.00	1425	232	256	256	…	256	252	242	250
TempMin	10.00	40.00	1425	10.0	11.5	12.5	…	12.0	11.5	10.0	12.0
TempMax	10.00	40.00	1425	17.0	18.0	18.5	…	18.0	17.5	16.5	18.5
TempMean	10.00	40.00	1425	13.0	15.0	15.5	…	15.0	14.5	13.5	15.5
HumMin	10.00	40.00	1425	43.5	41.5	40.5	…	43.0	43.5	44.0	41.5
HumMax	10.00	40.00	1425	55.5	53.0	52.5	…	54.5	55.0	56.0	53.0
HumMean	10.00	40.00	1425	49.5	47.5	47.0	…	49.0	49.5	50.0	47.5

Table 2

We could create as many new elements or “Operations” in our EMPClimate wsdl file as we wish our web Service to deliver. In this tutorial we’ve chosen to provide only one operation that delivers all the seven values (temp_min, temp_max, temp_mean, hum_min, hum_max, hum_mean).

We have generated the corresponding WSDL file that contains all the needed parameters to build your Web Service.

First, close the eclipse application

Then, replace your existing WSDL file (*EMPClimate.wsdl*) by the one provided within this tutorial archive file named (*EMPClimate_tutorial_v1.1.wsdl*). Keep the original name (*EMPClimate.wsdl*). In the following prompt command, we assume that you have unzipped the archive in your home directory. If this is not the case just copy and rename accordingly.

```
mesor:~$cp ~/EMPClimate_tutorial_v1.1.wsdl
~/workspace/EMPClimate/WebContent/WEB-INF/wsdl/EMPClimate.wsdl
```

Restart Eclipse.

In the “*Project Explorer*” tab, double click on the “*EMPClimate*” project.

Select “*WebContent/WEB-INF/wsdl*” folder and open “*EMPClimate.wsdl*” file.

Click on the arrow of the “*GetAllValuesResponseType*” box to display the “*Inline Schema of EMPClimate.wsdl*” tab.

You should now be able to view all the elements of the ”*GetAllValuesResponseType*”. like the following figure.

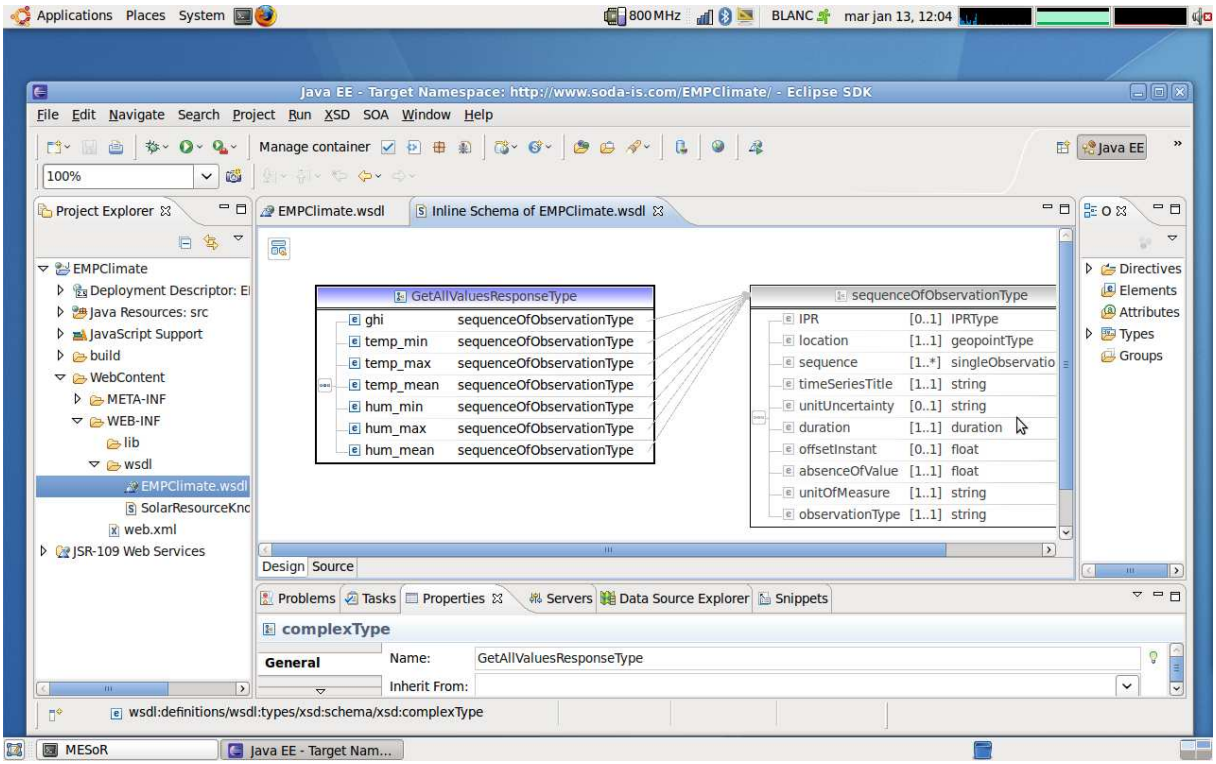

Figure 22

In this list, all of the types that have been bind to the inputs and outputs come from the XSD Schema. If you need extra elements that are not contained in the XSD Schema you can add one. There is no difficulty to mix elements descriptions that are supported or not by such a Schema.

Don't forget to re-generate the binding content when you modify or reload the WSDL file.

To do so, click on the little square box between the "*EMPClimateService*" box and the "*EMPClimatePortType*" box.

Select the "*Properties*" tab in the bottom window and click on the "*Generate Binding Content…*" button.

Tick the "*Overwrite existing binding information*" option.

Leave "*rpc literal*" option and click on the "*Finish*" button.

You can save your work and quit Eclipse.

4.2 Create Java Classes from the WSDL file

Now we are going to use some Jboss tools to create the Java Classes of our Web Service.

Let go back to the Xterm.

Set the following variables to your environment:

Set the JAVA_HOME environment variable to the JDK 5

```
mesor:~$ export JAVA_HOME= ~/jdk1.5.0_16
```

Set the JBOSS_HOME environment variable to jboss-4.2.2.GA

```
mesor:~$ export JBOSS_HOME=~/jboss-4.2.2.GA
```

Go the directory where your WSDL file is. It should be in the" …*WEB-INF/wsdl*" directory of your installation.

```
mesor:~$ cd ~/workspace/EMPClimate/WebContent/WEB-INF/wsdl/
```

Run the "*wsconsume.sh*" script with the following option to generate the Java Classes of your WSDL file.

```
mesor:~/[..]/wsdl$ bash $JBOSS_HOME/bin/wsconsume.sh –k
EMPClimate.wsdl
```

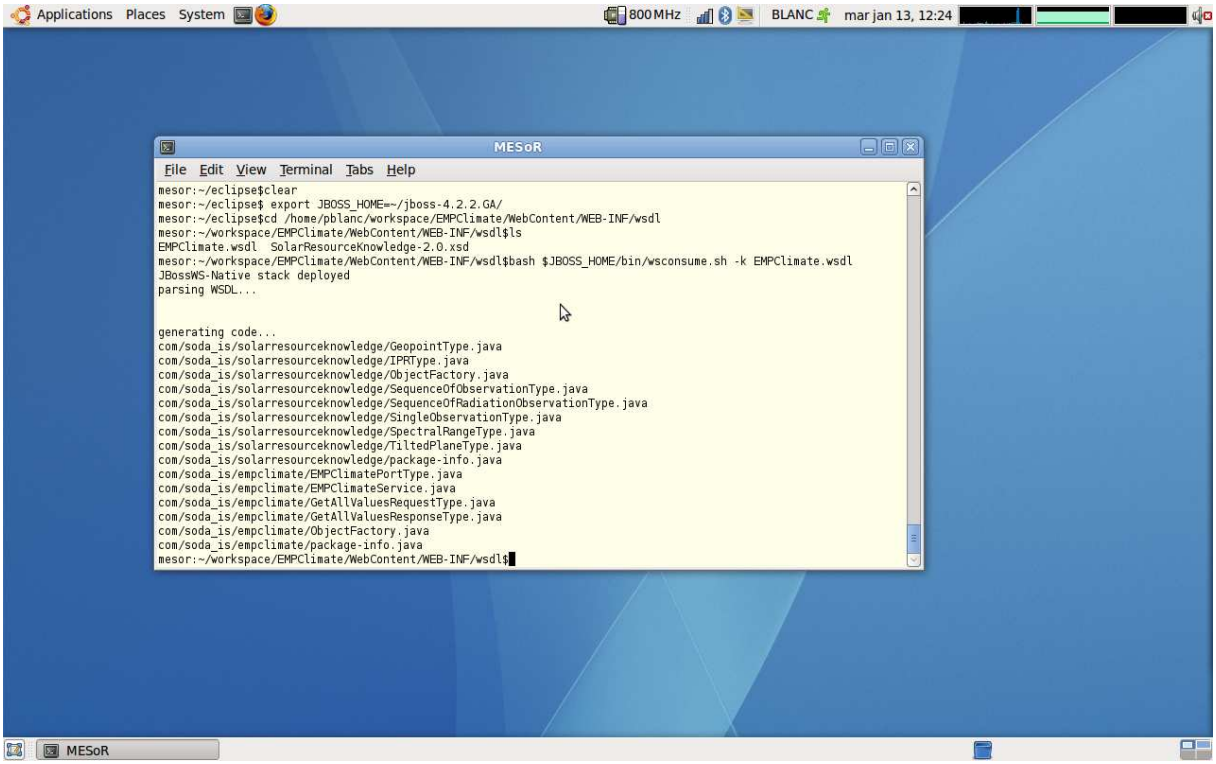

Figure 23

All the files that have been generated by the "*wsconsume.sh*" script are located in two directories ("*empclimate*" and "*solarresourceknowledge*") at:

~/workspace/EMPClimate/WebContent/WEB-INF/wsdl/output/com/soda_is

Those names are inherited from namespace values of the wsdl and the Schema files.

4.3 Run Eclipse to create the code of the Web Service

Now that all the classes of the skeleton of our Web Service have been created we need to run Eclipse to start programming our application based on those already existing classes.

Start a new Xterm.

Go to the Eclipse directory an run:

```
mesor:~/eclipse$ ./eclipse –vm ~/jdk1.6.0_07/bin/java
```

When your Project has been loaded in the Project Explorer window, right click on the project name (in our example "*EMPClimate*") and select "*Refresh"* or *"F5 key* " to make visible all the newly created Java Classes of the "*output*" directory.

In the "*Project Explorer*" menu, click on the following arrow folders to view the Java Classes:

EMPClimate/WebContent/WEB-INF/wsdl/output/com/soda-is/empclimate

or

EMPClimate/WebContent/WEB-INF/wsdl/output/com/soda-is/solarresourceknowledge

You need now to move the "*com*" folder under the "*output*" folder in the "*src*".

To do so, right click on the "*com*" folder.

Select the "*Move*…" option.

Select the "*src*" folder.

Click the "*OK*" button.

After the move is completed, delete the empty "*output*" folder.

Now you can view the Java Classes that are available in the "*Project Explorer*" as a menu called: "*java Resources:src*"

You have two packages called:

"com.soda_is.empclimate"

and

"com.soda_is. solarresourceknowledge"

In the "*com.soda_is.empclimate"* package double-click on "*EMPClimatePortType.java*" .

This file is the interface of the Web Service and contains the Java code that comes from the WSDL file. It's in this file that you will add your own Java code for doing your own calculation on the various elements that are available.

Take a look at the following screen (PS: this is the "*Maximise*" display available by clicking the icon on the top right of the window). You should get something similar.

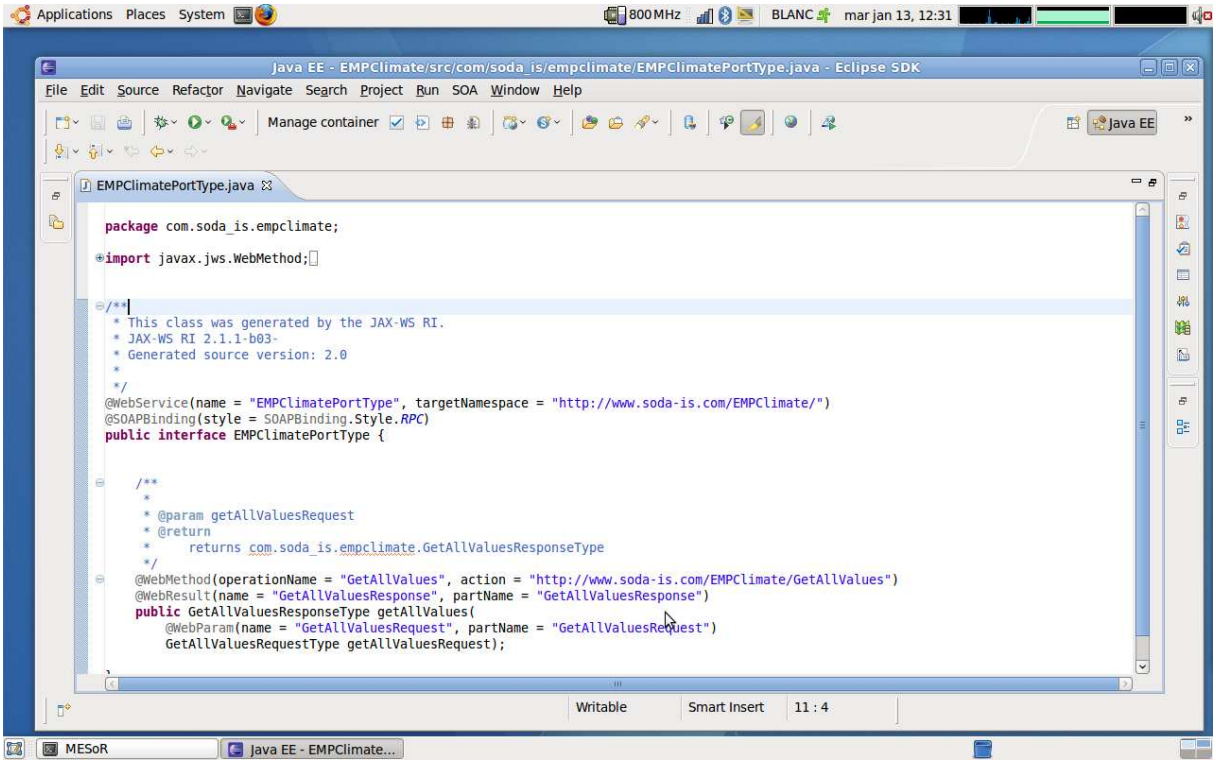

Figure 24

To start modifying the interface file (*EMPClimatePortType.java*), you should change the "*public interface*" directive to "*public class*".

Save the file.

Now you need to implement the operation (getAllValues) of this newly created class.

As a starter just make this Method return a "*null*" value. This will avoid Eclipse to complain. Save your work.

See the following figure with the highlighted modifications:

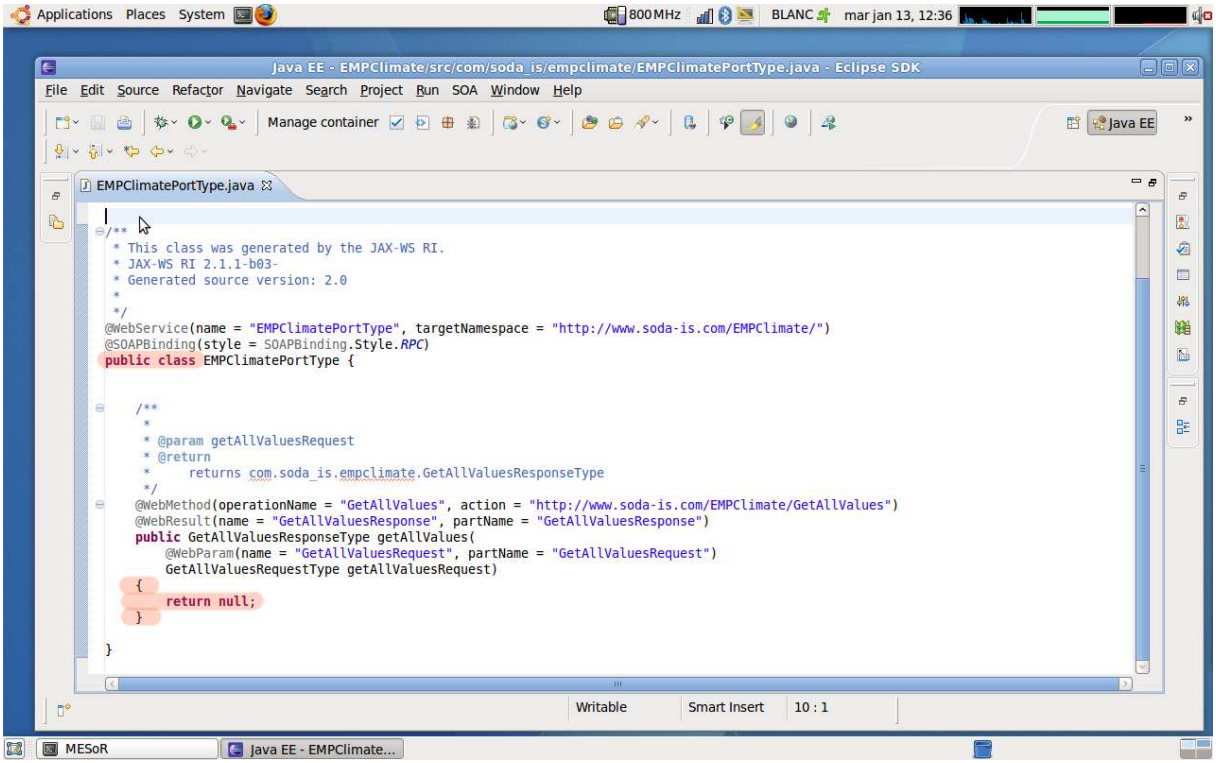

Figure 25

We have appended to this tutorial the modified "*EMPClimatePortType.java* " file in order to show the code that we have added to the operation binded with the "*EMPClimate.wsdl*" file that we have provided.

You should load this file in place of the current "*EMPClimatePortType.java*".

To do so, close Eclipse.

Replace the file "*EMPClimatePortType.java*" that is in your project by the one (*EMPClimatePortType_tutorial_v1.1.java*) provided with this tutorial. Keep the "*EMPClimatePortType.java*" name.

```
mesor:~$cp ~/EMPClimatePortType_tutorial_v1.1.java
~/workspace/EMPClimate/src/com/soda_is/empclimate/EMPClimatePor
tType.java
```

Restart Eclipse.

When the project is loaded, Eclipse ask for a "*Refresh*".

Press "*F5*" key of go to "*File > Refresh*".

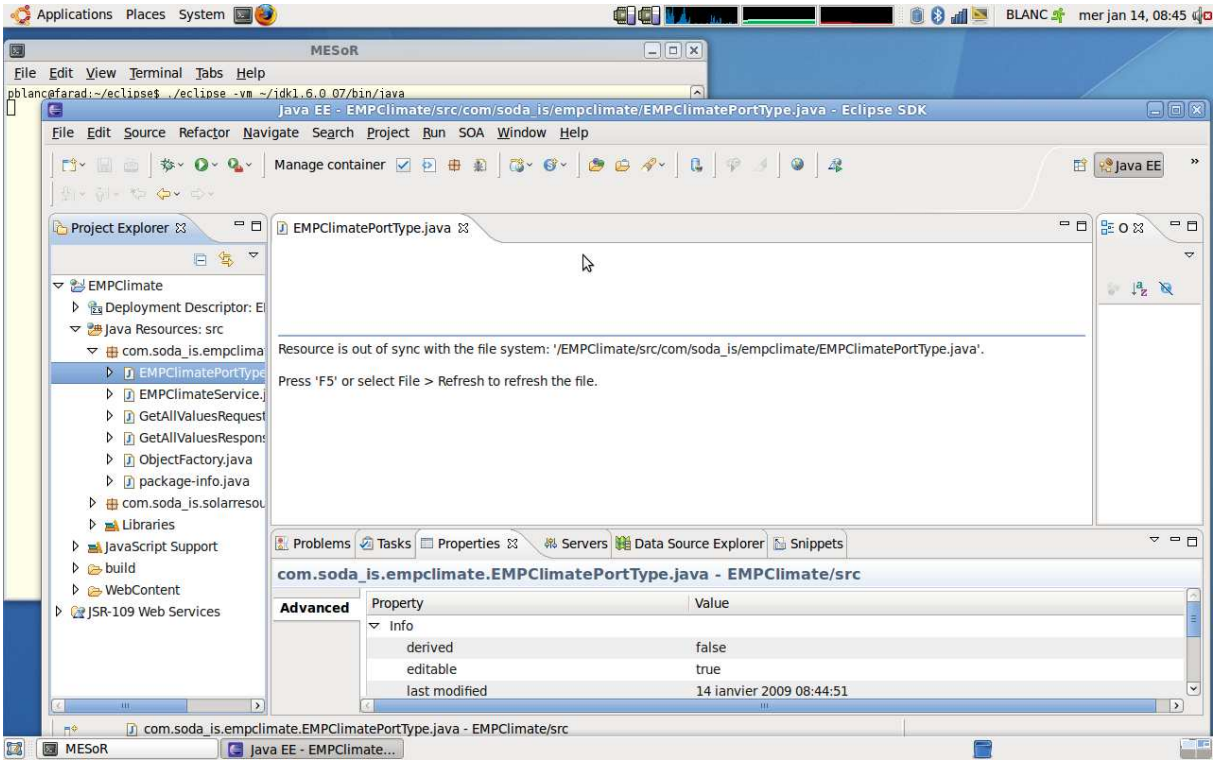

Figure 26

The new "*EMPClimatePortType.java*" file is loaded and contains the code for accessing the remote resource.

You can browse this code to get a flavour of Java code for Web Services.

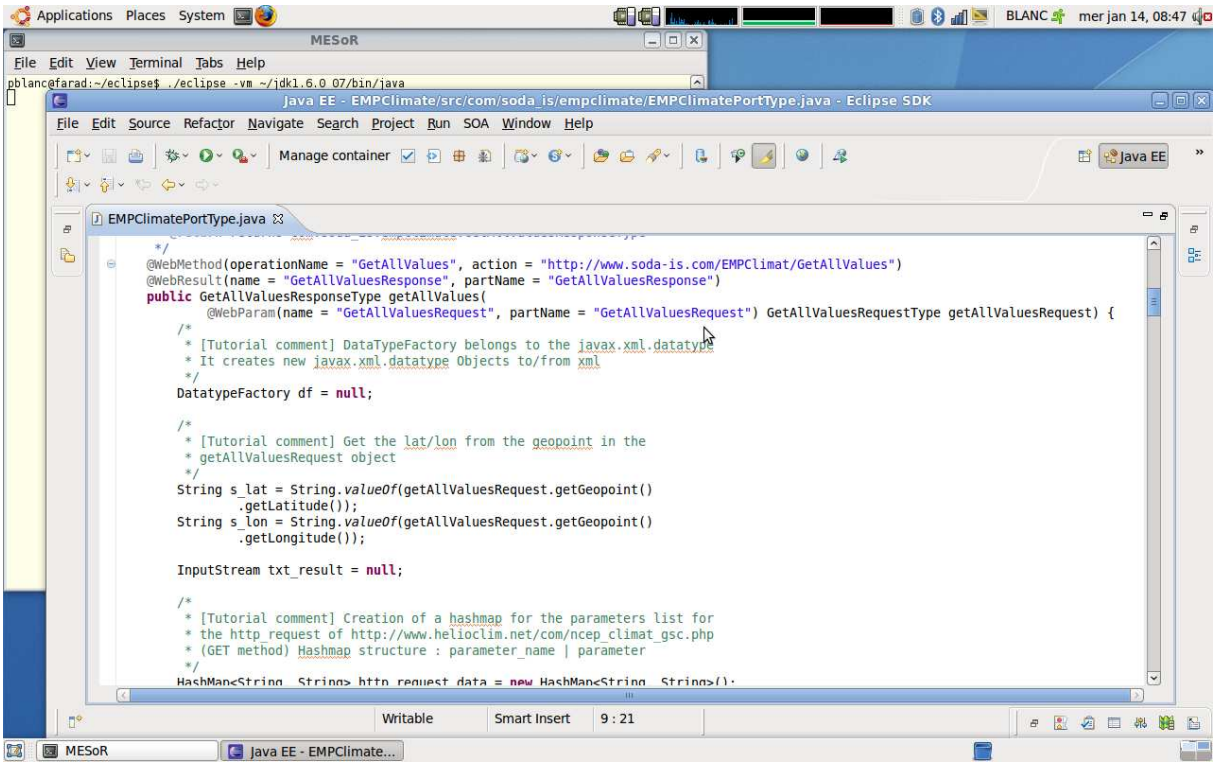

Figure 27

4.4 Edit the web.xml file and create the war file

Now that the code is written, we are going to create the Web archive (war) file to be deployed on the Jboss Application Server.

Go in Eclipse and in the "*Project Explorer*" window right click on:

"*JavaResources:src/EMPClimatePortType.java/EMPClimatePortType*" (Class symbol)

Select "*Copy Qualified Name*" option.

Open the file "web.xml" under the folder:

"*WebContent/WEB-INF*"

In the main window, right click on the "*web-app*" item and select:

"*Add Child/message-destination – welcome-file-list/servlet*"

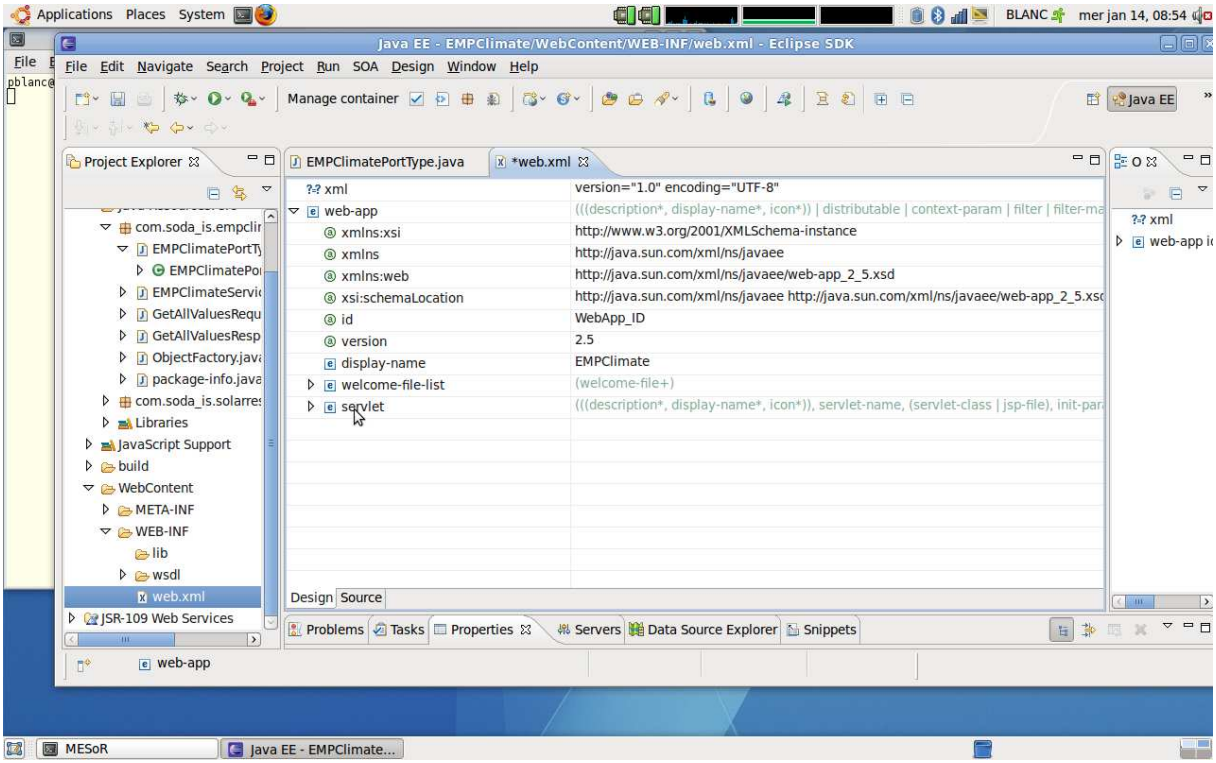

Figure 28

Click the "*servlet*" item in the bottom of the list and change the following fields:

for "*servlet-name*" enter "*EMPClimate*",

for "*servlet-class*" right click in the field and past the "*Qualified Name*" that we copied previously (*com.soda_is.empclimate.EMPClimatePortType*).

You should have something similar to the following screen.

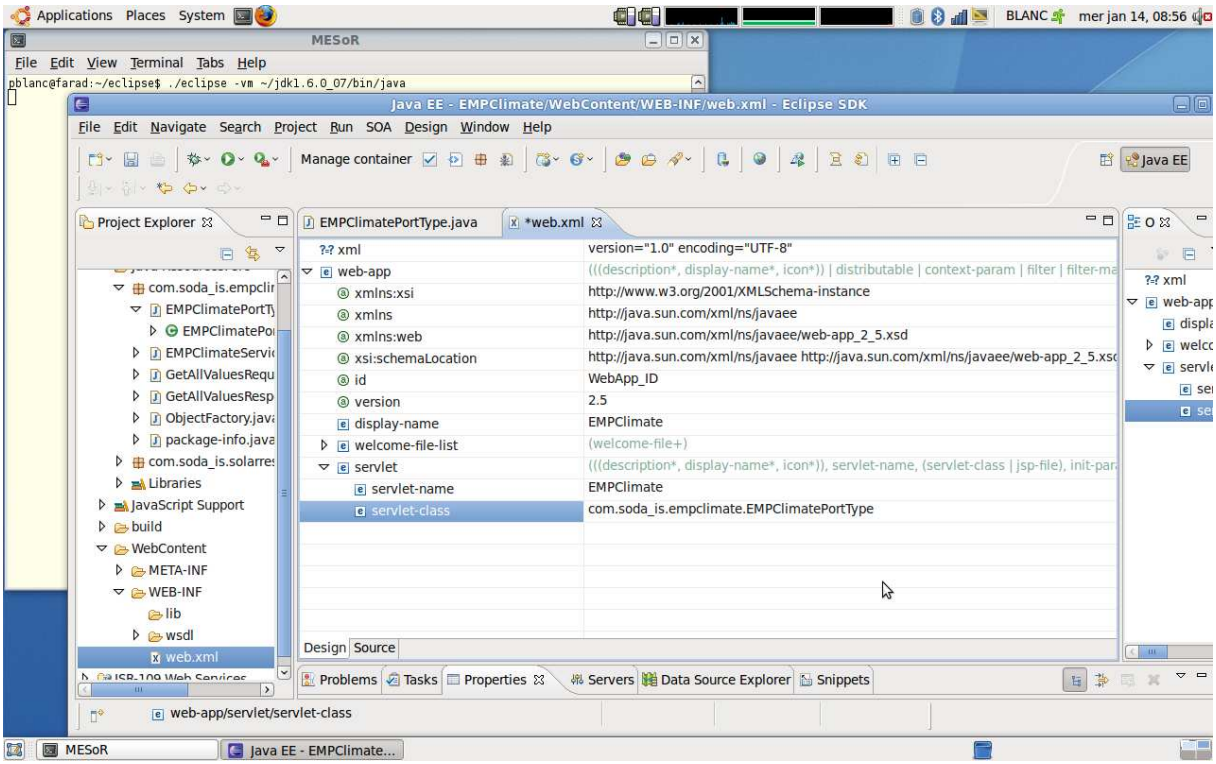

Figure 29

Similarly to adding a "*servlet*", you must add a "*servlet mapping*" to the web.xml file. This will give a URL to our service.

In the main window, right click on the "*web-app*" item and select:

"*Add Child/message-destination – welcome-file-list/servlet-mapping*"

Click the "*servlet-mapping*" item in the bottom of the list and change the following fields:

for "*servlet-name*" enter "*EMPClimate*",

for "*url-pattern*" enter *"/service*".

Save your work (Ctrl S).

You should have something similar to the following screen.

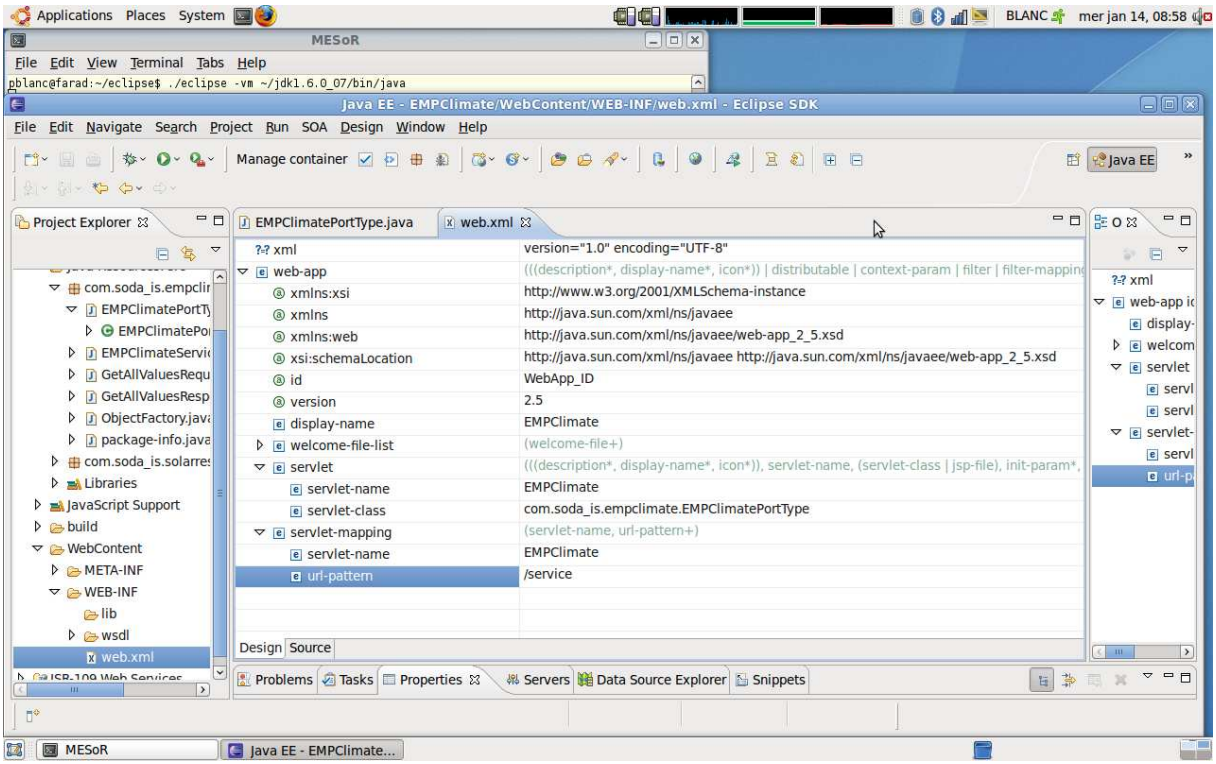

Figure 30

We have finished to set-up the Web Service and we need now to export it as a Web archive or WAR file.

In the "*Project Explorer*", right click on "*EMPClimate*" project and select:

"*Export/WAR file*".

Keep the default name eg. "*EMPClimate*" and select a destination folder of your choice. In our case the home directory of the user.

Un-tick the option "*Optimize for a specific server runtime*".

Tick the option "*Overwrite existing file*".

Click the "*Finish*" button.

Close eclipse.

4.5 Deploy the web service on Jboss

We are testing the deployment of our Web Service on the Jboss Application server.

Open a Xterm.

Define the correct JBOSS_HOME environment:

```
mesor:~$ export JBOSS_HOME=~/jboss-4.2.2.GA
```

Define the correct JAVA_HOME variable:

```
mesor:~$ export JAVA_HOME=~/jdk1.5.0.16
```

Go the Jboss directory

```
mesor:~$ cd $JBOSS_HOME
```

Start Jboss:

```
mesor:~/jboss-4.2.2.GA$ ./bin/run.sh
```

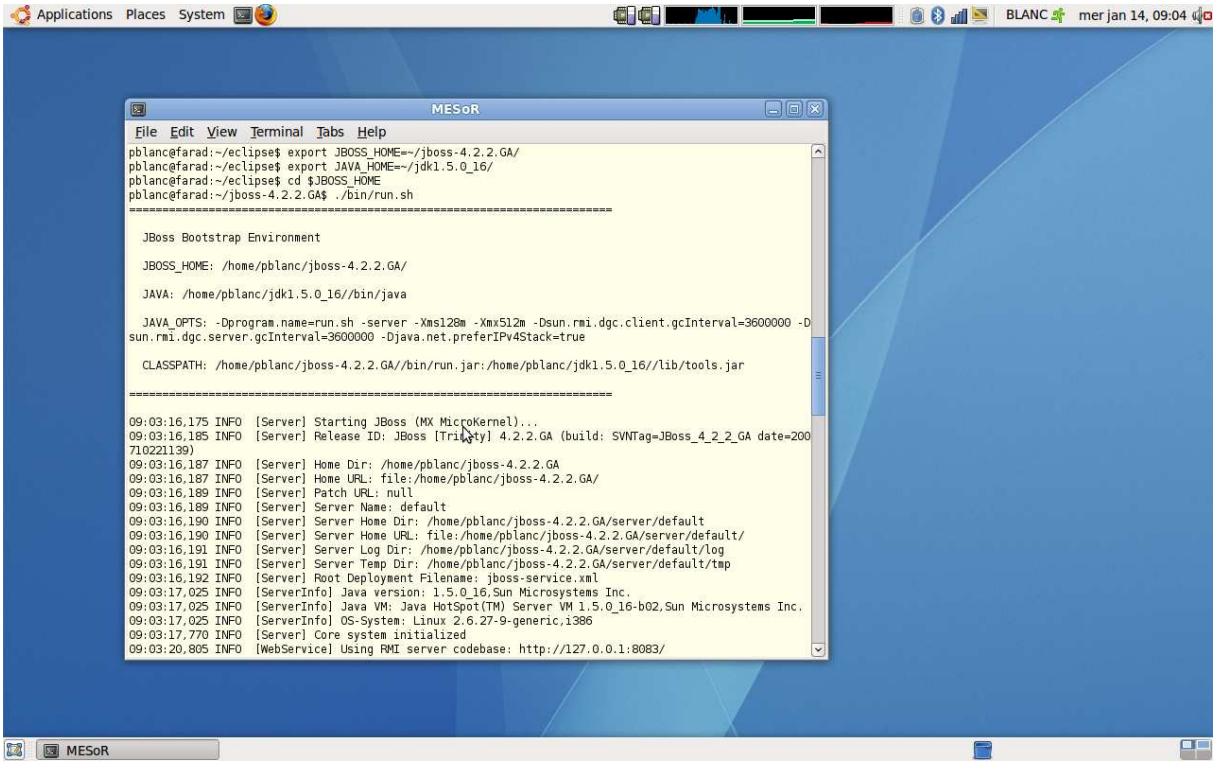

Figure 31

It may take few second while Jboss filled the Xterm with logs.

You must end-up with a line similar to this:

"*…Started in 25s:117ms*"

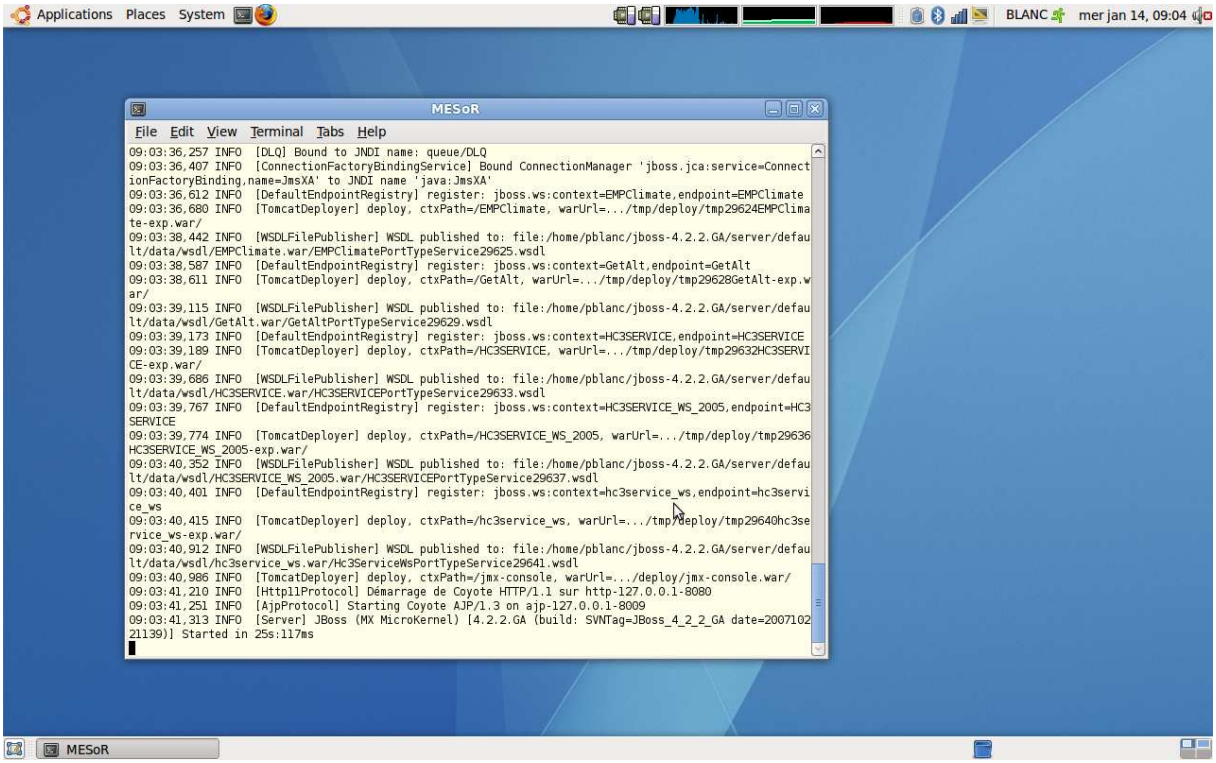

Figure 32

Now that Jboss is started, we are going to deploy our "*EMPClimate.war*" file in the "*deploy*" section of the Jboss server.

Open a new Xterm and do the following:

Define the correct $JBOSS_HOME environment:

```
mesor:~$ export JBOSS_HOME=~/jboss-4.2.2.GA
```

and copy the war-file "*EMPClimate.war"* in the deploy directory of Jboss

```
mesor:~$ cp EMPClimate.war $JBOSS_HOME/server/default/deploy/
```

If you look at the Jboss logs in the first Xterm where Jboss has been started, after the "*…Started in 25s:117ms*" line you can see that our WAR file has been deployed and that the WSDL file has been published.

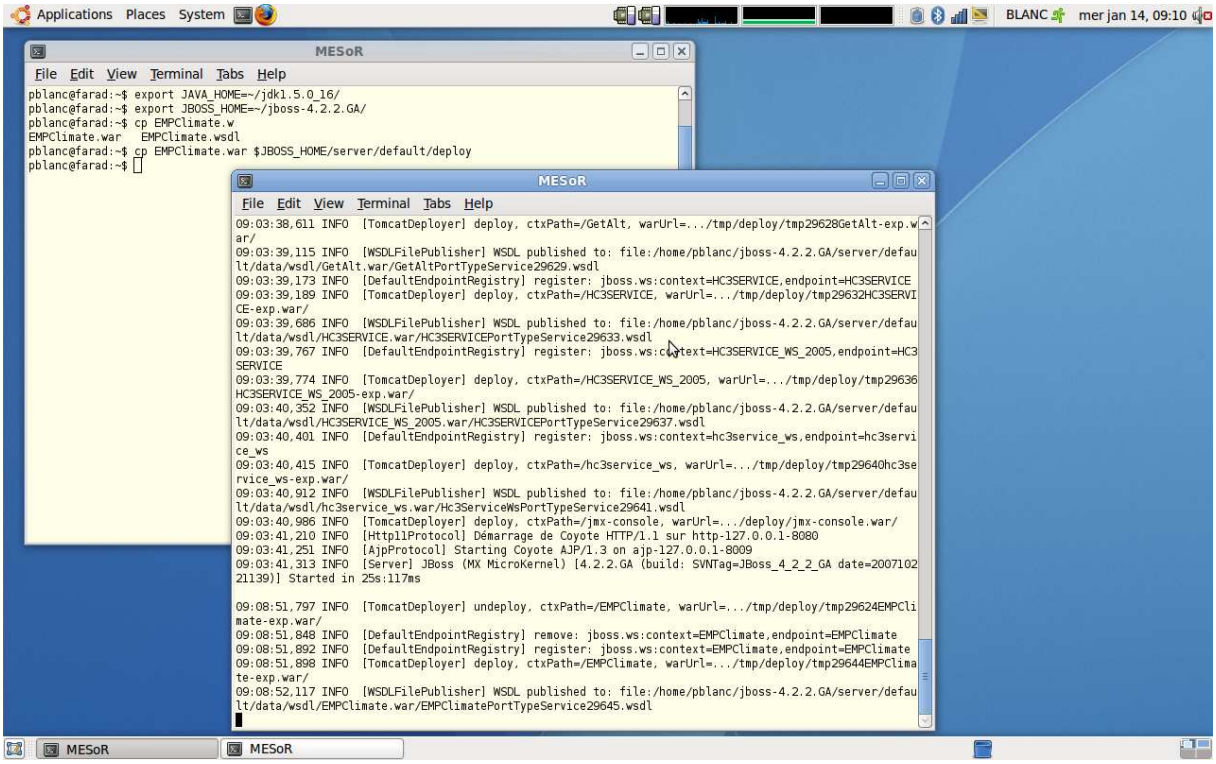

Figure 33

To check if Jboss is correctly launched and has correctly deployed your Web Service, open your favorite browser and enter the following URL:

http://localhost:8080/jbossws

This is sub-menu "*Jboss Web Service*" of the Jboss Application Server.

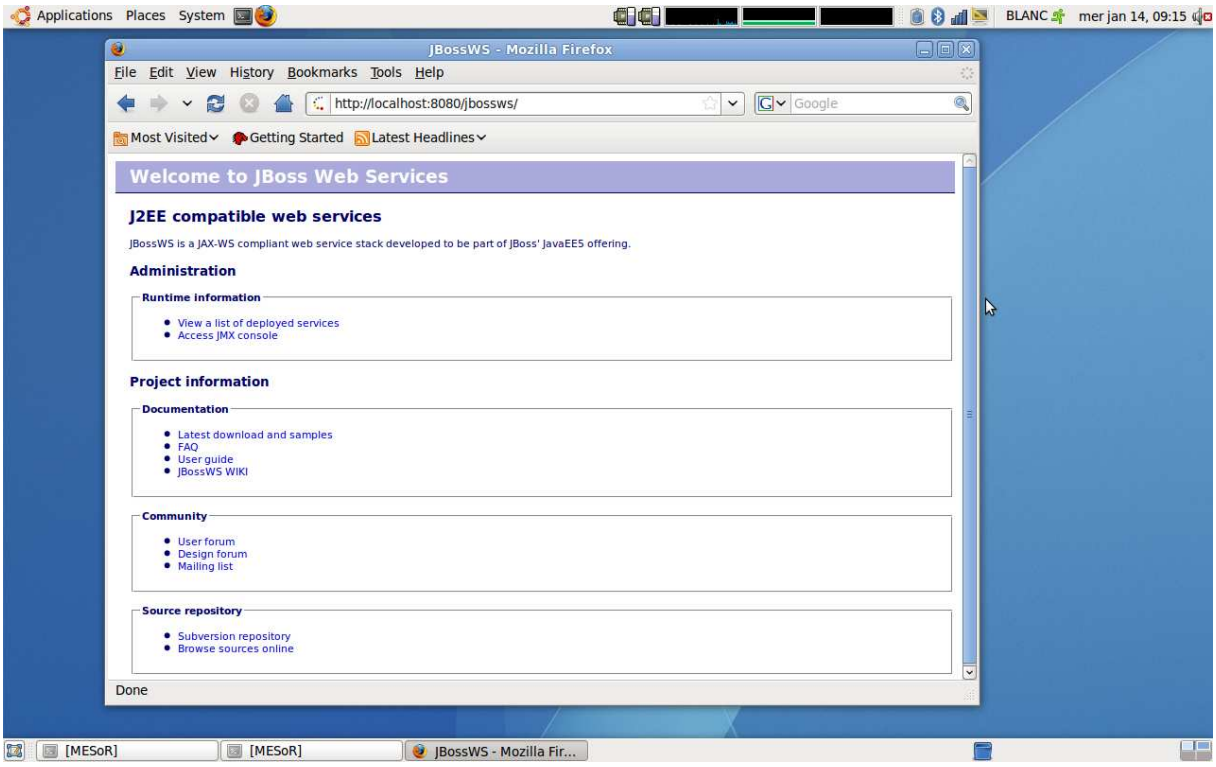

Figure 34

If you click on the link: "*View a list of deployed service*" you will be led to: http://localhost:8080/jbossws/services witch show you the list of deployed Web Services.

You should have something similar to the following screen.

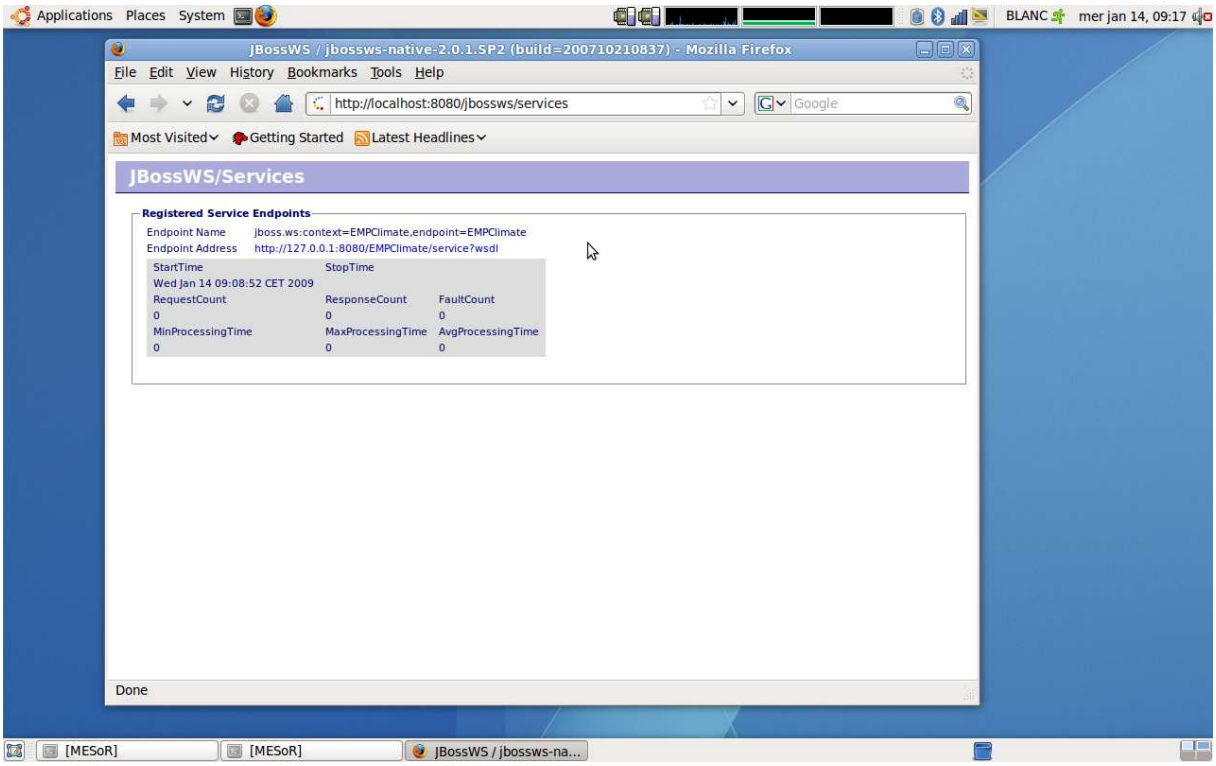

Figure 35

Note on the "*Endpoint Address*" URL the "*/service*" that we have previously entered in Eclipse as "*service-mapping/url-pattern*" value of the file "*web.xml*".

Click on the link to see the WSDL display as a XML grammar in your browser.

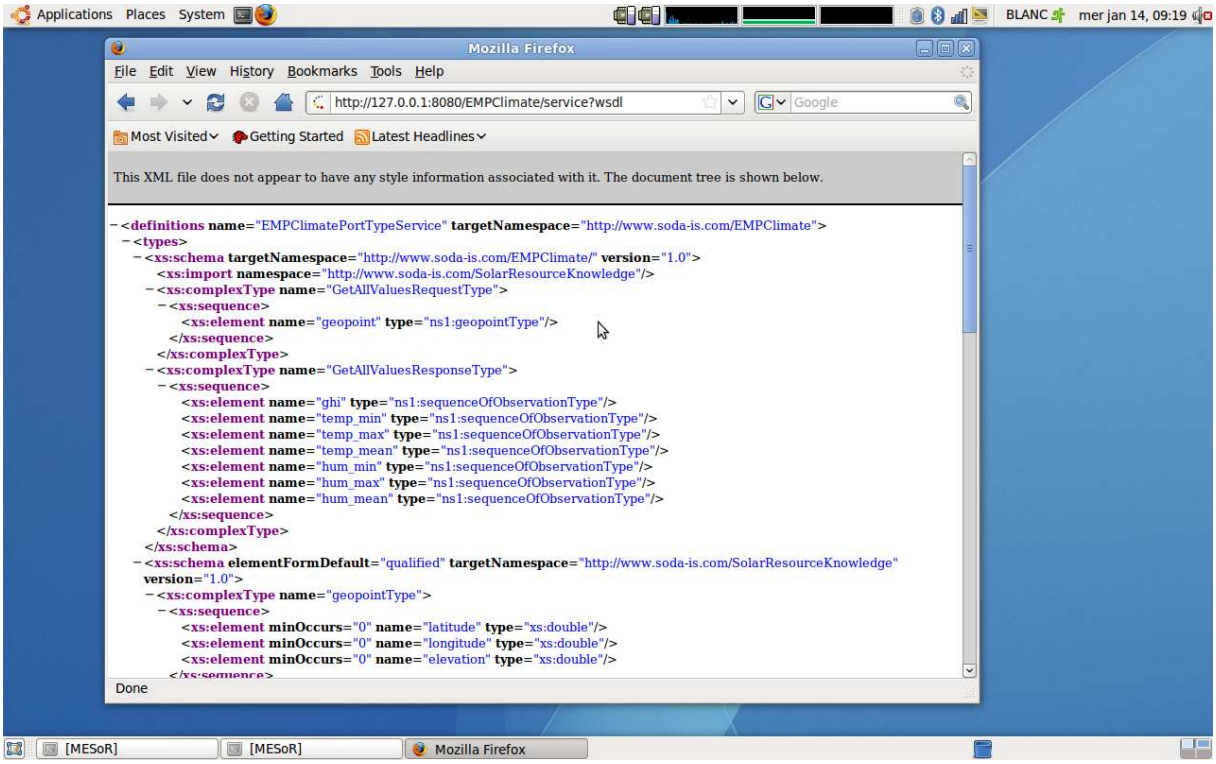

Figure 36

Congratulations your Web Service is correctly deployed on Jboss.

This WSDL could be used latter on to build the skeleton of a client.

4.6 Use Eclipse to test (consume) the Web Service

To make sure your Web Service will correctly answer to the request and provide the correct result, we are going to test it in Eclipse.

First in the browser copy the URL of the WSDL file:

"http://127.0.0.1:8080/EMPClimate/service?wsdl"

Start Eclipse, if it was not already running.

Start the *"Web Service Explorer"* by clicking on the icon "*Launch the Web Services Explorer*" of the top icon menu.

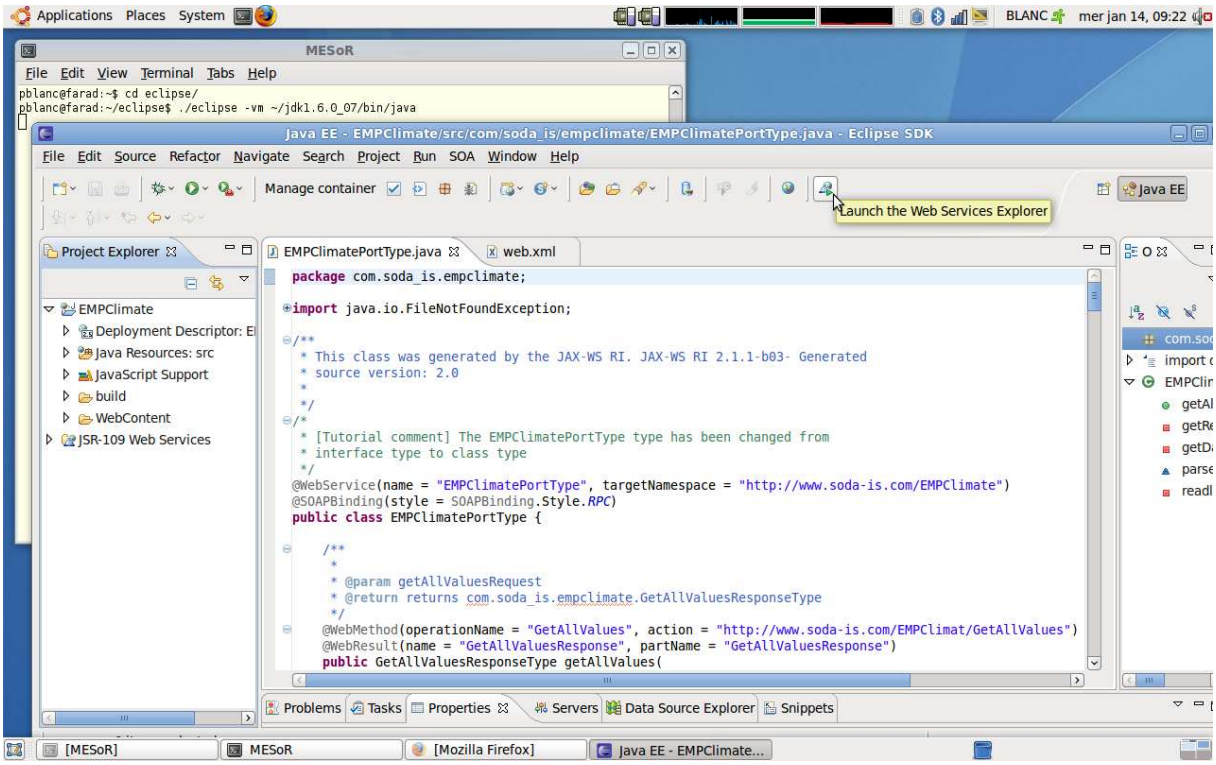

Figure 37

Note that the "*Web Services Explorer*" window could appear directly in the eclipse IDE or in your favourite browser. There is some advantages (eg. history of entered values) to tests the service in a browser. To latter do so, copy the URL that is in the top bar of the Eclipse application. In our example (see Figure 38):

"*http://127.0.0.1:37469/wse/wsexplorer/wsexplorer.jsp?org.ecplise.wst.ws.explorer=0*"

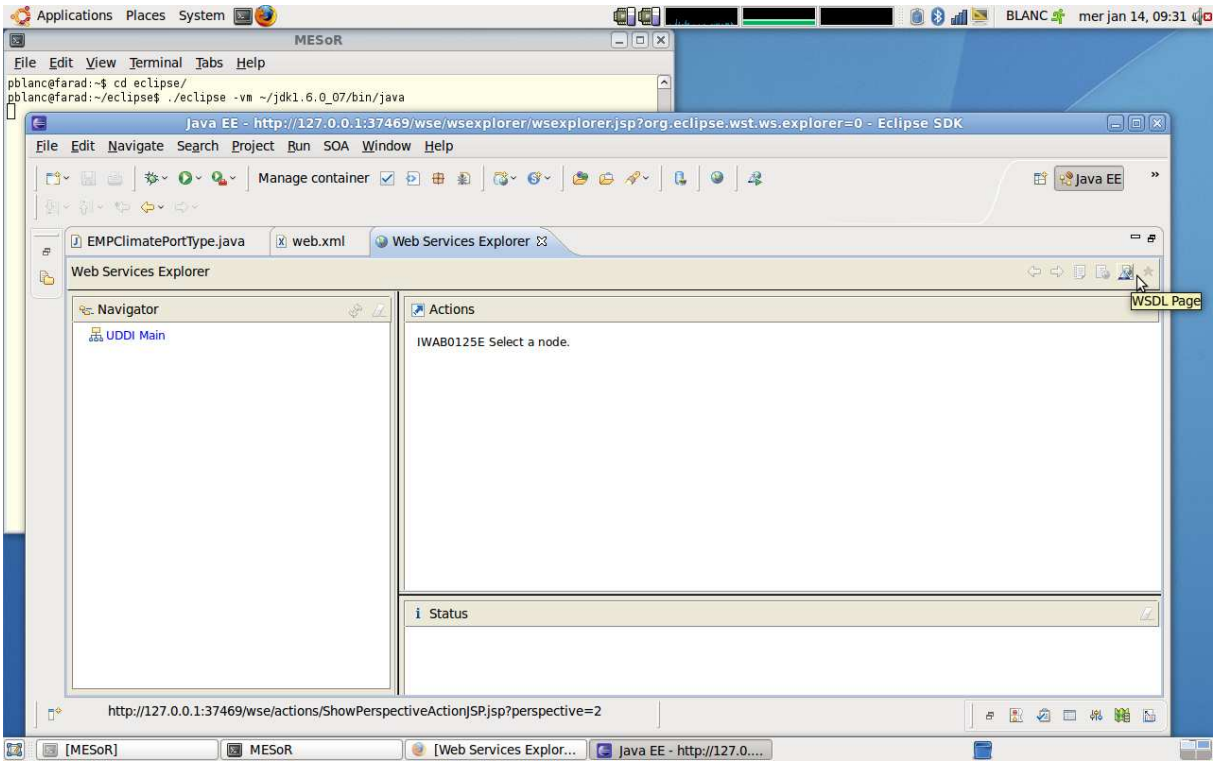

Figure 38

First click on the "*WSDL Page*" icon located in the top right menu bar of the "*Web Services Explorer*" window. (See tool tip in the Figure 38)

Then click on the link "*WSDL Main*" on the left part of the "*Web Service Explorer*" window.

In the form on the main window enter the URL of the WSDL file:

http://127.0.0.1:8080/EMPClimate/service?wsdl

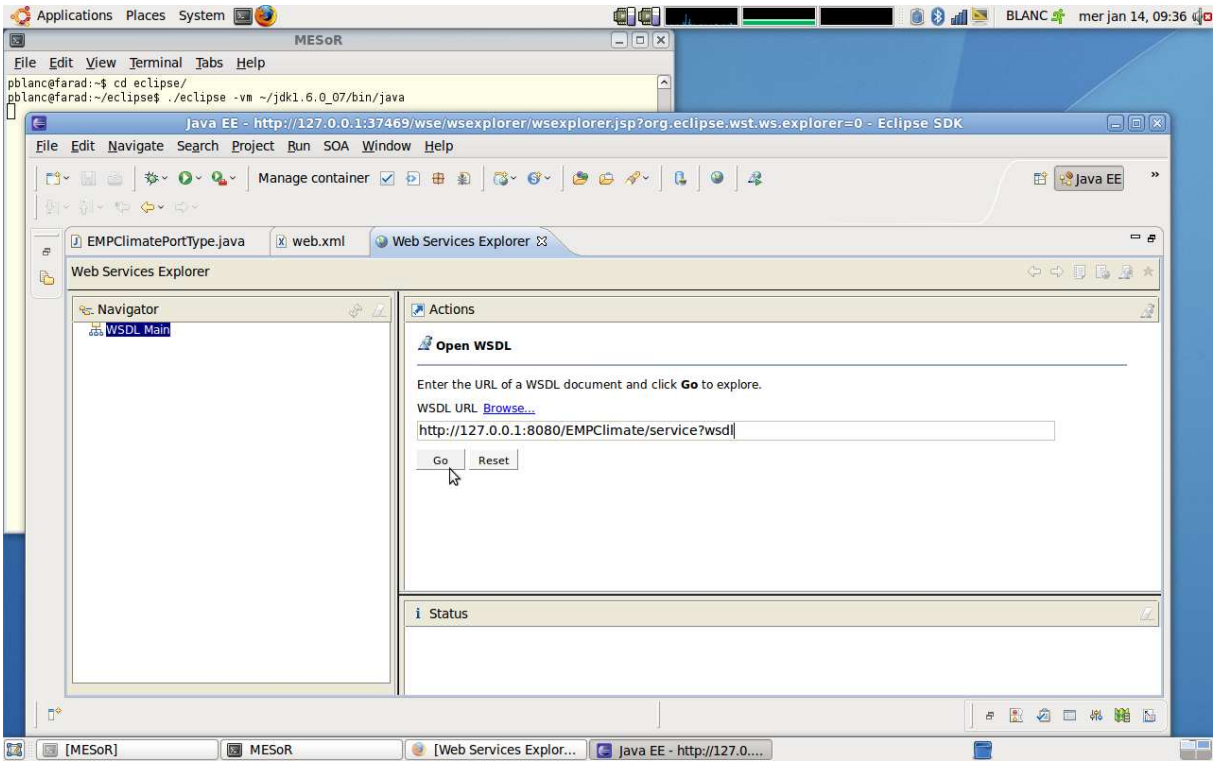

Figure 39

Press the “Go” button

You should see the “*WSDL binding details*” of the Web Service. This is the operation (GetAllValues) that we have implemented in our Web Service.

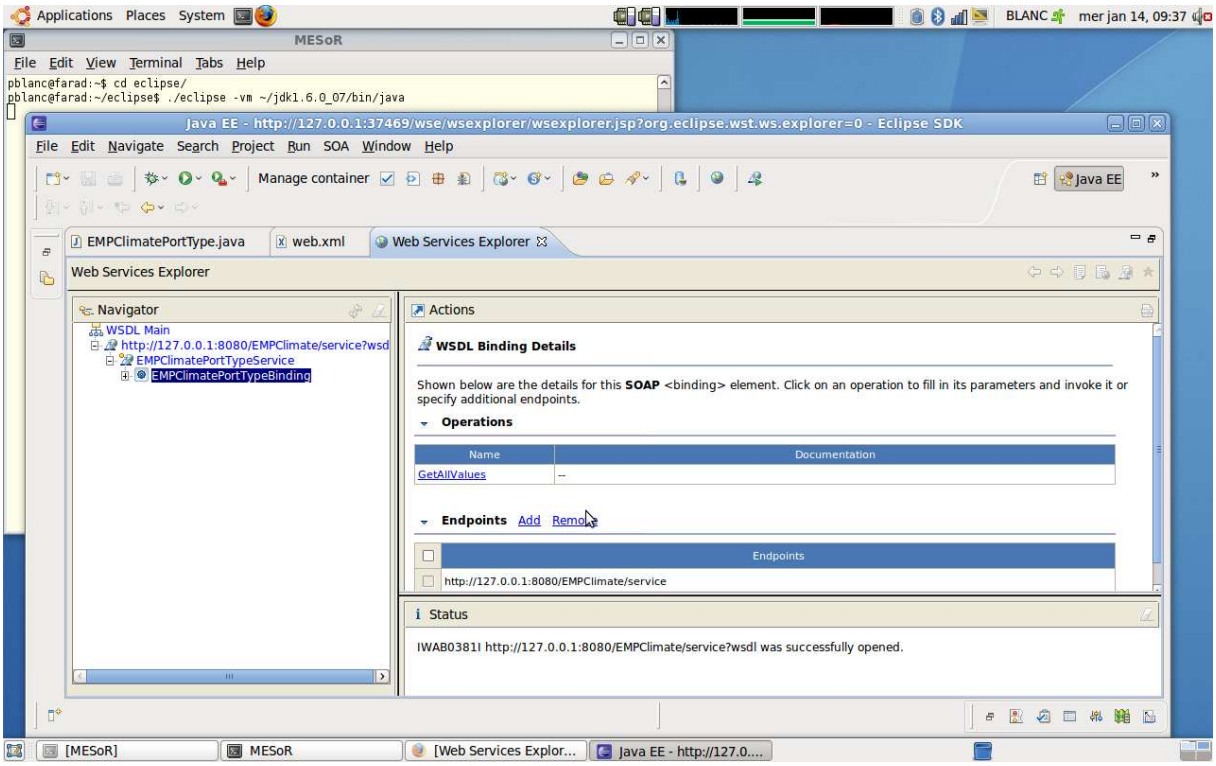

Figure 40

You can now test the operation. Click on the link "*GetAllValues*".

Click on the "*Add*" link for the elements "*latitude"* and "*longitude"* and enter your desired values. The element "*elevation*" is not relevant for this service.

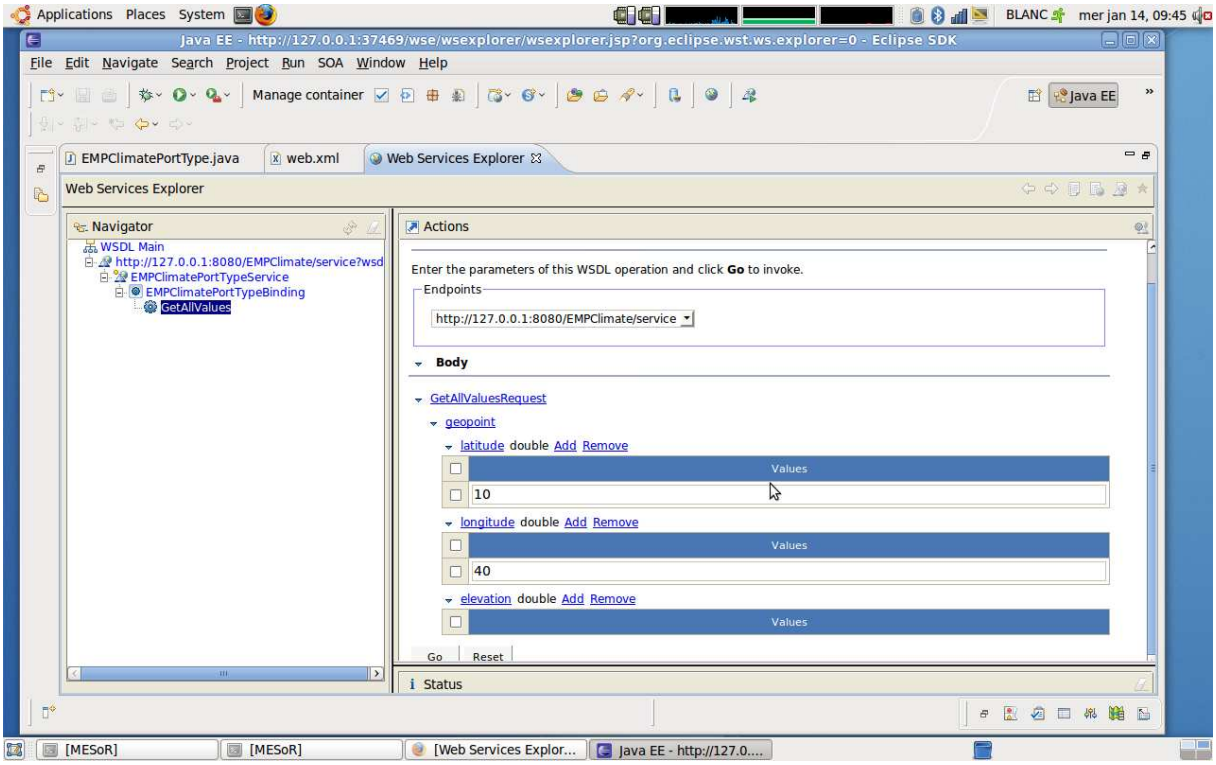

Figure 41

Click the "Go button.

Resize the "*Status*" window below and verify the response provided by your Web Service.

If you get something similar to following window you have successfully test your Web Service.

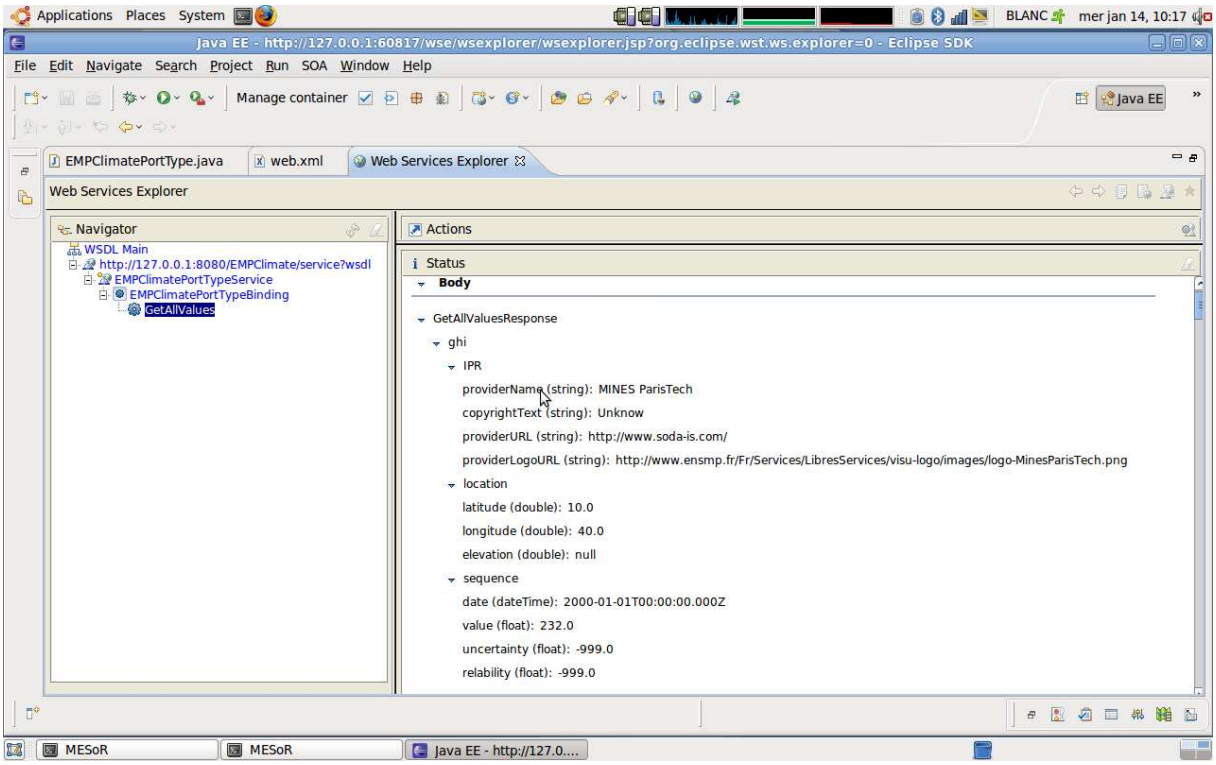

Figure 42

4.7 Web Service exploitation strategy

If you plan to operate your Web Service at your premise, you must make sure that the endpoint (*http://127.0.0.1:8080/EMPClimate/service?wsdl*) is accessible from the outside of your domain. The address of the web site must be public as opposed to the URL above. This is necessary to allow partners to access and possibly build a client onto your Web Service.

If you plan not to operate your Web Service by your own, ARMINES, in the framework of the MESoR project proposes you to host your Web Service in its platform named www.webservice-energy.org. The service that you want to provide must be remotely accessible (remember the different possible cases). ARMINES can deal with any restriction you might want to apply on the use of your Web Service.

To provide this Web Service to ARMINES, you must send the WAR file ("*EMPClimate.war*" in our case) by email if not too big or by any other appropriate mean.

The archive does not need to include the Java source. Only Java Classes and the "*web.xml*" file are mandatory.