



| | | |
|---|---|--|
|  | <p>FP6-IP 511731</p> <p>MODELWARE</p> <p><i>MODELLing solution for softWARE systems</i></p> |  |
|---|---|--|

D3.1.3 - ModelBus Tool Integration Kit and Guidelines document

Due date of deliverable:

Actual submission date: 28/07/06

Start date of project: 01/08/2004

Duration: 24 months

Organisation name of lead contractor for this deliverable: LIP6

Revision: 0.8

Dissemination level:

| | | |
|-----------|---|-------------------------------------|
| PU | Public | <input checked="" type="checkbox"/> |
| PP | Restricted to other programme participants (including the Commission Services) | <input type="checkbox"/> |
| RE | Restricted to a group specified by the consortium (including the Commission Services) | <input type="checkbox"/> |
| CO | Confidential, only for members of the consortium (including the Commission Services) | <input type="checkbox"/> |



Project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006)

| Rev. | Date (dd/mm/yy) | Author | Checked by | Internal approval | Commission approval | Description |
|---------|--------------------|---|------------|----------------------|------------------------|---|
| 0.1-0.6 | 09/03/06 | Fateh Bekhouche, Fatima Fadil, Nicolas Garandeau, Nils Henner, Saoussen Kraiem, Philippe Kych, Rémy-Christophe Schermesser, Marc Schwitzguébel | | | | Initial Contribution |
| 0.7 | 12/03/06 | Andrey Sadovykh | | | | Stub usage section. Reformatting |
| 0.8 | 28/07/06 | Andrey Sadovykh | | | | Skeleton Usage Generation from Eclipse Editor |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

Table of contents

| | |
|---|-----------|
| Table of contents | 3 |
| 1. Introduction | 5 |
| 2. SETUP | 6 |
| 2.1. Install SUN Java J2SE SDK 1.4.2 | 6 |
| 2.2. Install Eclipse 3.0.1 | 6 |
| 2.3. Install the Eclipse Modeling Framework (EMF) | 6 |
| 2.4. Install the ModelBus Adapter and Toolkit..... | 8 |
| 3. CREATE A PROJECT | 9 |
| 3.1. Launch Eclipse | 9 |
| 3.2. Edit the tool description | 14 |
| 4. FORMATING RULES | 16 |
| 4.1. Create a description file..... | 16 |
| 4.2. Xmi referencement rules: | 16 |
| 4.3. Node description and containment rules..... | 17 |
| Event nodes | 17 |
| Error nodes | 17 |
| ModelBusPackaging nodes | 17 |
| ModelingServiceInterface nodes | 17 |
| ModelingService nodes | 17 |
| Parameter nodes | 18 |
| Property nodes | 18 |
| Tool nodes18 | |
| MetaclassSpecification nodes | 18 |
| Primitive / ModelType nodes | 19 |
| 4.4. Attribute constraints | 19 |
| Lower and Upper constraints..... | 19 |
| Name attributes | 19 |
| 5. GENERATE STUB AND SKELETON FOR YOUR TOOL..... | 20 |
| 5.1. Create a Folder | 20 |
| 5.2. Validate Tool Description | 21 |

| | | |
|------|---|----|
| 5.3. | Generate Stubs/Skeletons | 21 |
| 5.4. | Expected Result | 22 |
| 6. | USE STUB IN CLIENT APPLICATION | 23 |
| 7. | USE SKELETON IN SERVER APPLICATION..... | 24 |
| 7.1. | Test | 25 |
| 8. | Acknowledgements..... | 25 |
| 9. | Appendix..... | 26 |
| 9.1. | OCLTool Description | 26 |
| 9.2. | OCLToolServiceInterface | 26 |
| 9.3. | OCLToolServiceInterfaceStub | 29 |
| 9.4. | OCLToolProviderInterface..... | 32 |
| 9.5. | OCLToolSkeleton | 33 |
| 9.6. | SampleOclClient..... | 34 |
| 9.7. | SampleOclProviderImpl..... | 35 |
| 9.8. | SampleOclProviderDeploy | 35 |

1. Introduction

ModelBus is a distributed environment purposed to integrate heterogeneous MDA tools. It provides a standard approach to exchange models and to execute remote modelling services. It is based on Web Services and proposes deployment for several platforms: Java stand-alone and Eclipse. Moreover, for tools located in the same JVM, a local access mode is automatically chosen. For more information about ModelBus, please refer to the ModelBus documentation.

ModelBus Toolkit intends to help in integration of modelling tools. It uses Eclipse as a platform for graphical user interfaces. In addition, it requires the ModelBus adapter installed in order to plug the tool to the data bus. The complete installation requirements are listed in the SETUP section.

This tutorial shows how to use the ModelBus Toolkit to generate an adapter from an XMI description of the tool.

The following scenario is considered in this tutorial:

1. SETUP

The first step is installation of all prerequisites for the Toolkit.

Download Eclipse, install EMF, install ModelBus Adapter and Toolkit - all you need to get a functional environment.

2. CREATE A PROJECT

Create a simple eclipse project to use the ModelBus Toolkit.

3. Edit the tool description

How to edit manually a Tool description file.

4. FORMATING RULES

Constraints on structure and naming of the description file elements.

5. GENERATE STUB AND SKELETON FOR YOUR TOOL

Use the Toolkit to generate a stub for the tool you have described.

6. USE STUB IN CLIENT APPLICATION

Use generated stub in your client application.

Limitations: Stub generation engine has the following limitations.

- Only, consumer side generation is available.
- Generated interfaces are only compatible with default serializer / model representation format.

2. SETUP

2.1. Install SUN Java J2SE SDK 1.4.2

1. Get it from java.sun.com
<http://java.sun.com/j2se/1.4.2/download.html>
2. Launch the installation software, and proceed with the default installation

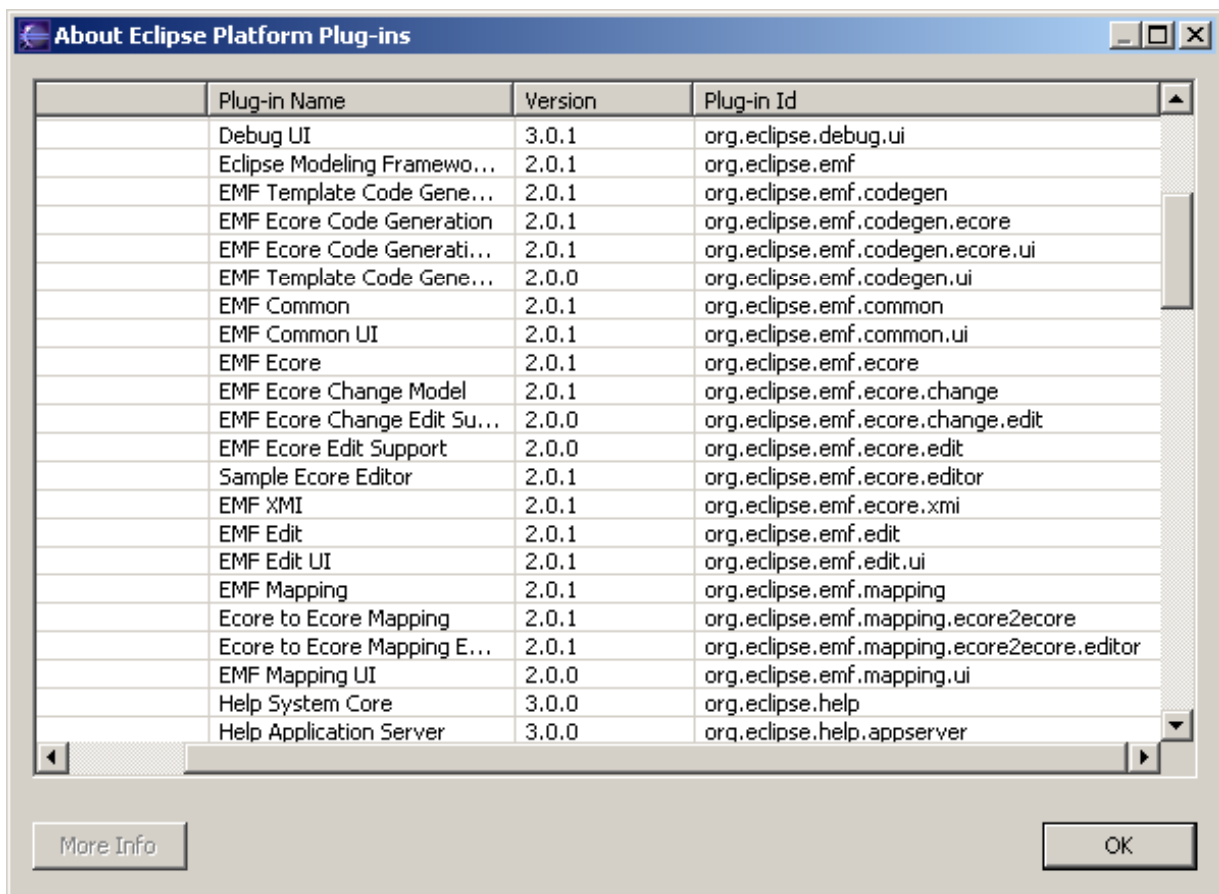
2.2. Install Eclipse 3.0.1

1. Get it from eclipse.org
<http://download.eclipse.org/eclipse/downloads/drops/R-3.0.1-200409161125/index.php>
2. Extract files from the downloaded archive to the desired folder

2.3. Install the Eclipse Modeling Framework (EMF)

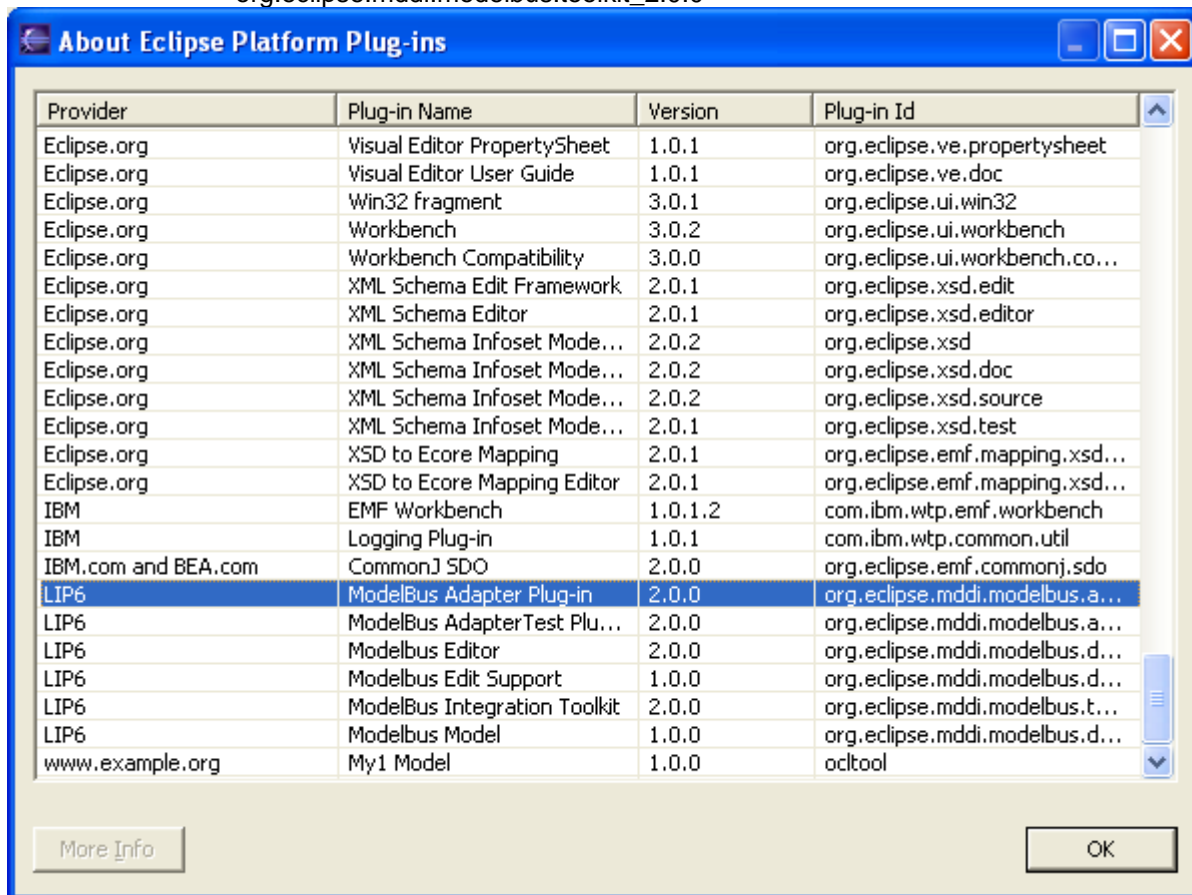
1. Get EMF for Eclipse 3.0.1 from eclipse.org.
<http://download.eclipse.org/eclipse/downloads/index.php>.
Or install it through the Eclipse platform («Help / Software Updates» menu)

2. In the Eclipse « Help » menu, press the « About Eclipse / Plug-in Details » button and verify that you have the following entries in the list :
 - org.eclipse.emf_2.0.1
 - org.eclipse.emf.codegen_2.0.1
 - org.eclipse.emf.codegen.ui_2.0.0
 - org.eclipse.emf.codegen.ecore_2.0.1
 - org.eclipse.emf.codegen.ecore.ui_2.0.1
 - org.eclipse.emf.common_2.0.1
 - org.eclipse.emf.common.ui_2.0.1
 - org.eclipse.emf.ecore_2.0.1
 - org.eclipse.emf.ecore.edit_2.0.0
 - org.eclipse.emf.ecore.editor_2.0.1
 - org.eclipse.emf.ecore.change_2.0.1
 - org.eclipse.emf.ecore.change.edit_2.0.0
 - org.eclipse.emf.ecore.xmi_2.0.1



2.4. Install the ModelBus Adapter and Toolkit

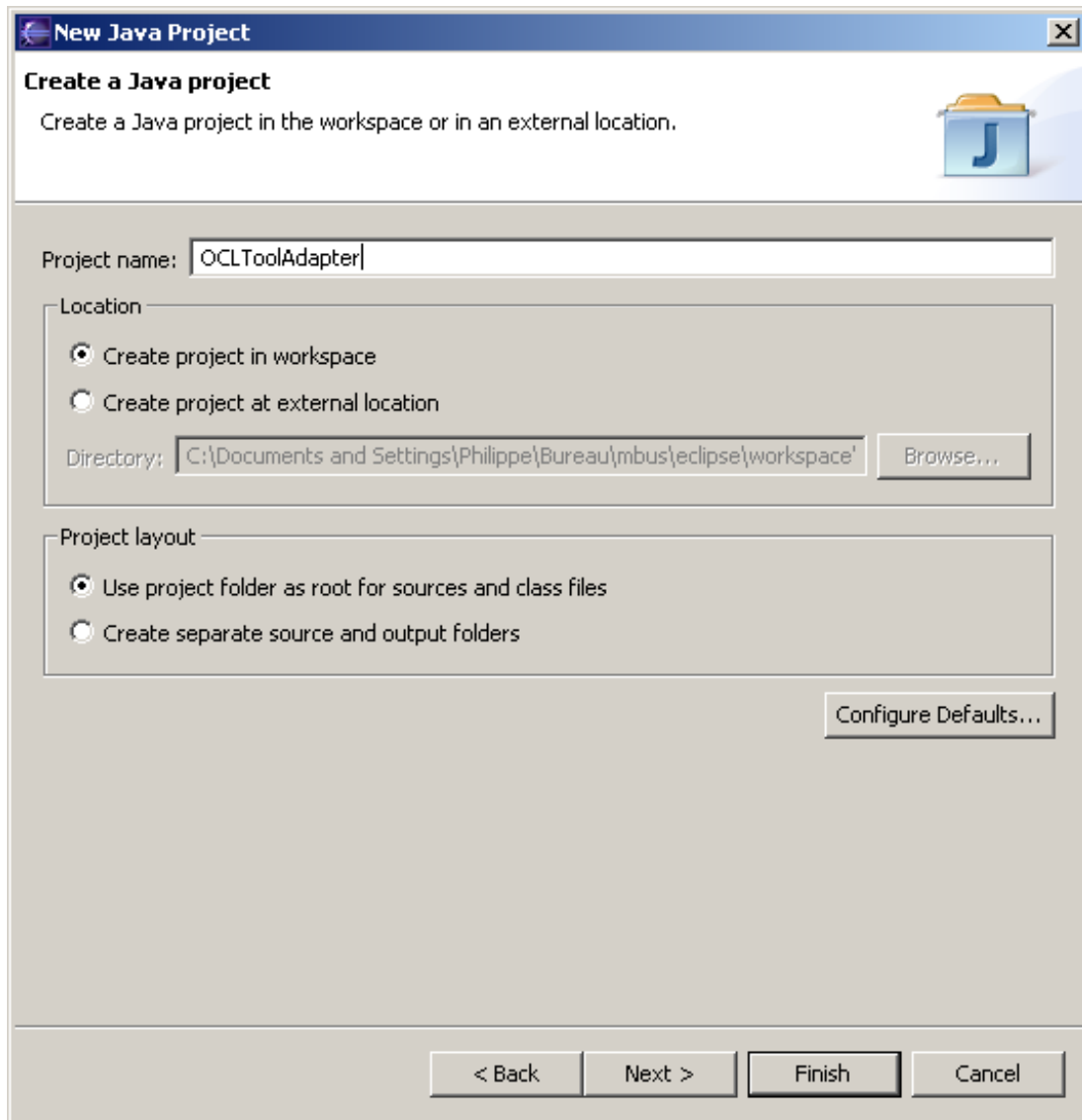
1. Get it from eclipse.org
<http://www.eclipse.org/mddi/download.php>
2. Extract files from the downloaded archive to a temporary folder
3. Copy the content of the folder « release/eclipse/plugins » into the subdirectory « plugins » of your Eclipse installation path.
4. Launch Eclipse, select « Help / About Eclipse platform » from the main menu bar, press the « Plug-in Details » button and verify you have the following entries in the list :
 - org.eclipse.mddi.modelbus.adapter_2.0.0
 - org.eclipse.mddi.modelbus.description_1.0.0
 - org.eclipse.mddi.modelbus.description.editor_2.0.0
 - org.eclipse.mddi.modelbus.description.edit_1.0.0
 - org.eclipse.mddi.modelbus.toolkit_2.0.0



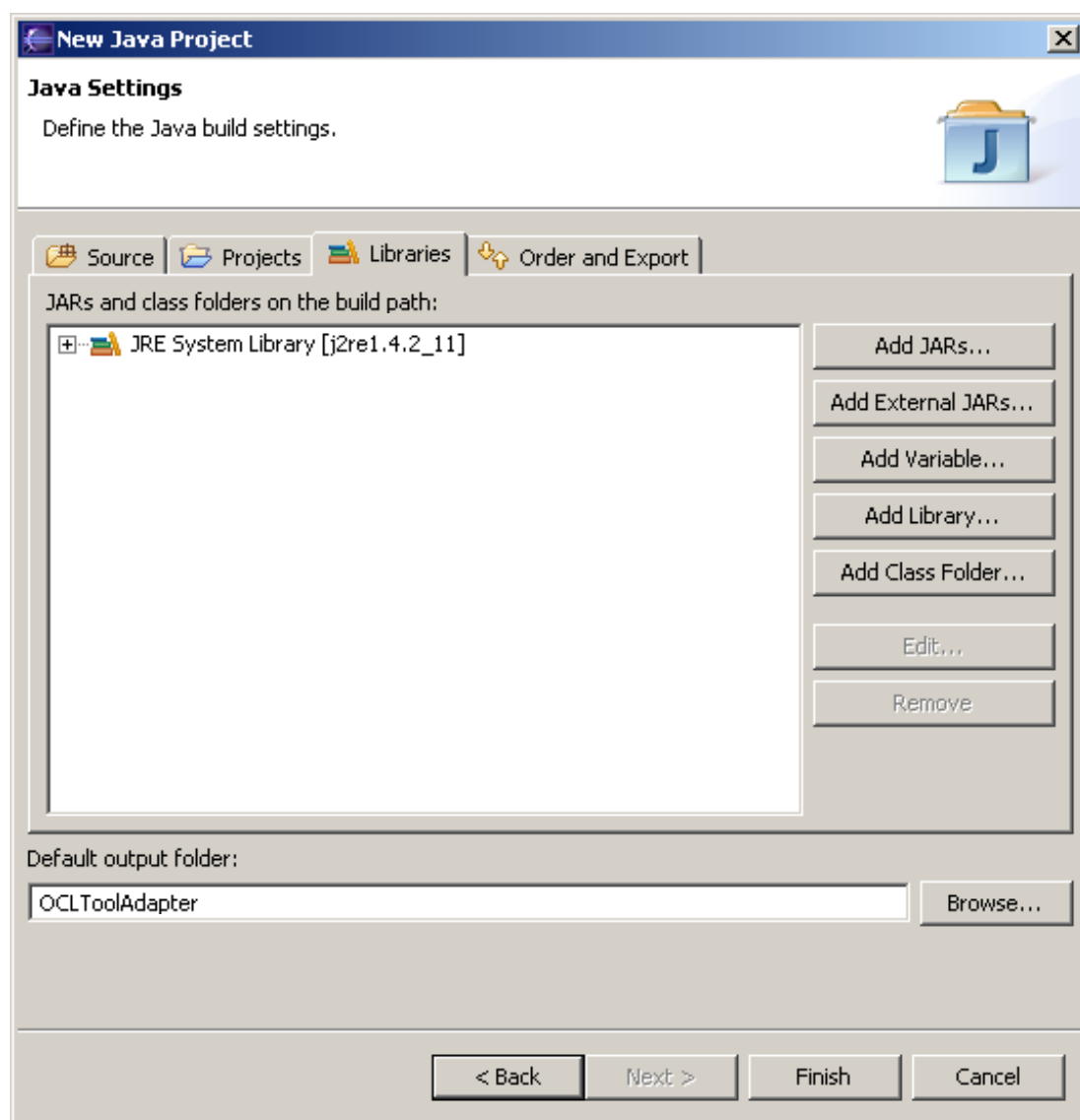
3. CREATE A PROJECT

3.1. Launch Eclipse

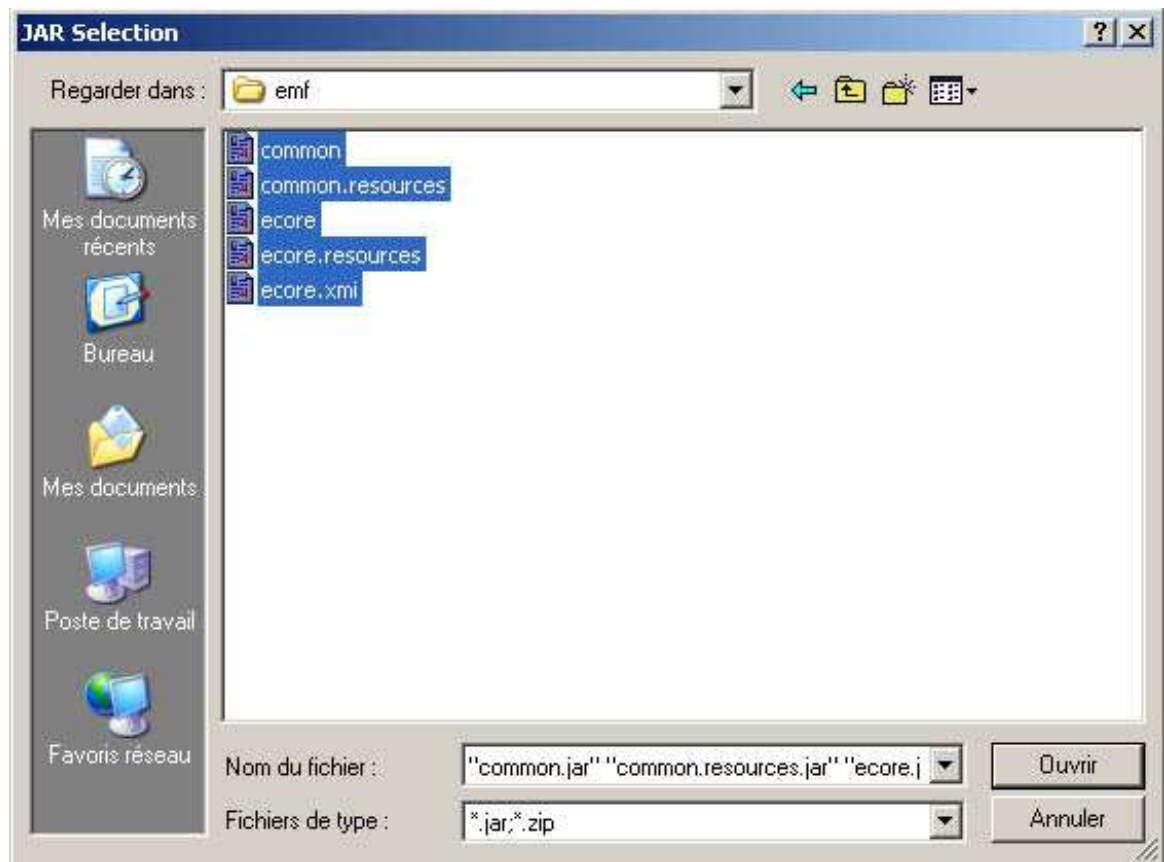
1. Launch Eclipse
2. Create a new Java project by selecting « File / New / Project... » from the main menu bar, select « Java Project » and press « Next »
3. Name it “OCLToolAdapter” and press « Next »



4. On the next screen, select the « Libraries » tab button and press « Add External JARs... »

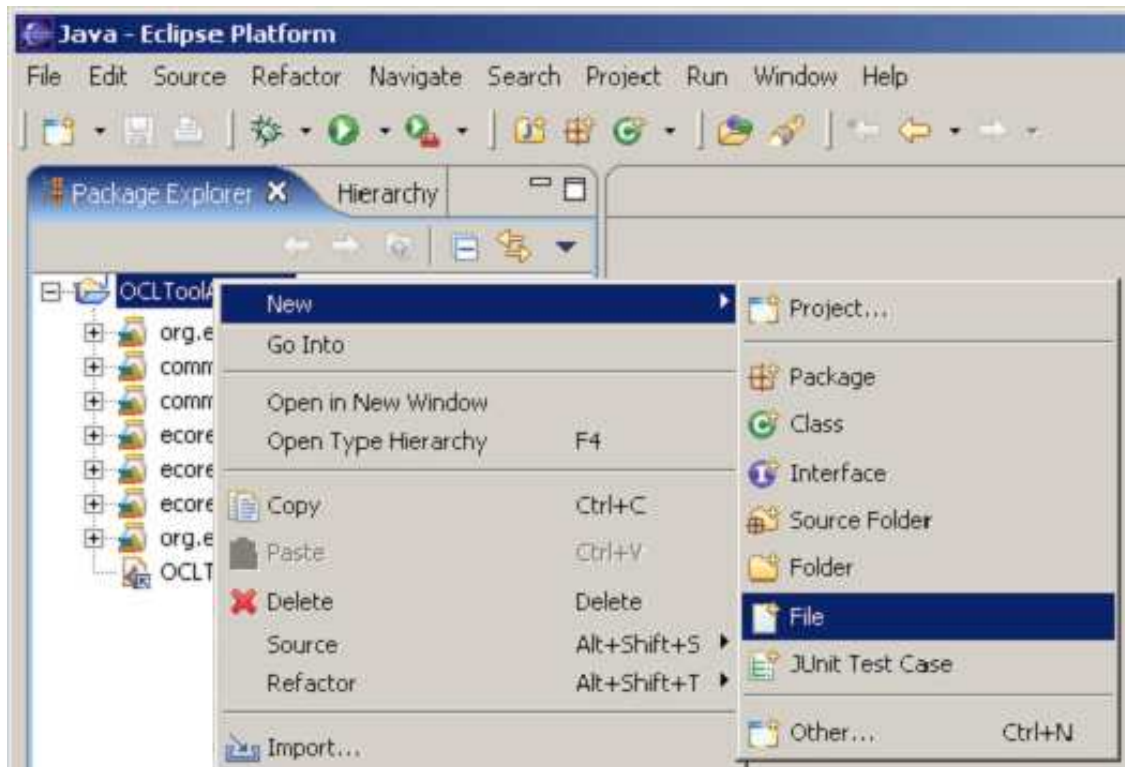


5. Browse your file system to your ModelBus Adapter extract folder ("plugin" subdirectory of your Eclipse installation Path), select all the files from the « lib /emf » directory and press « Open »

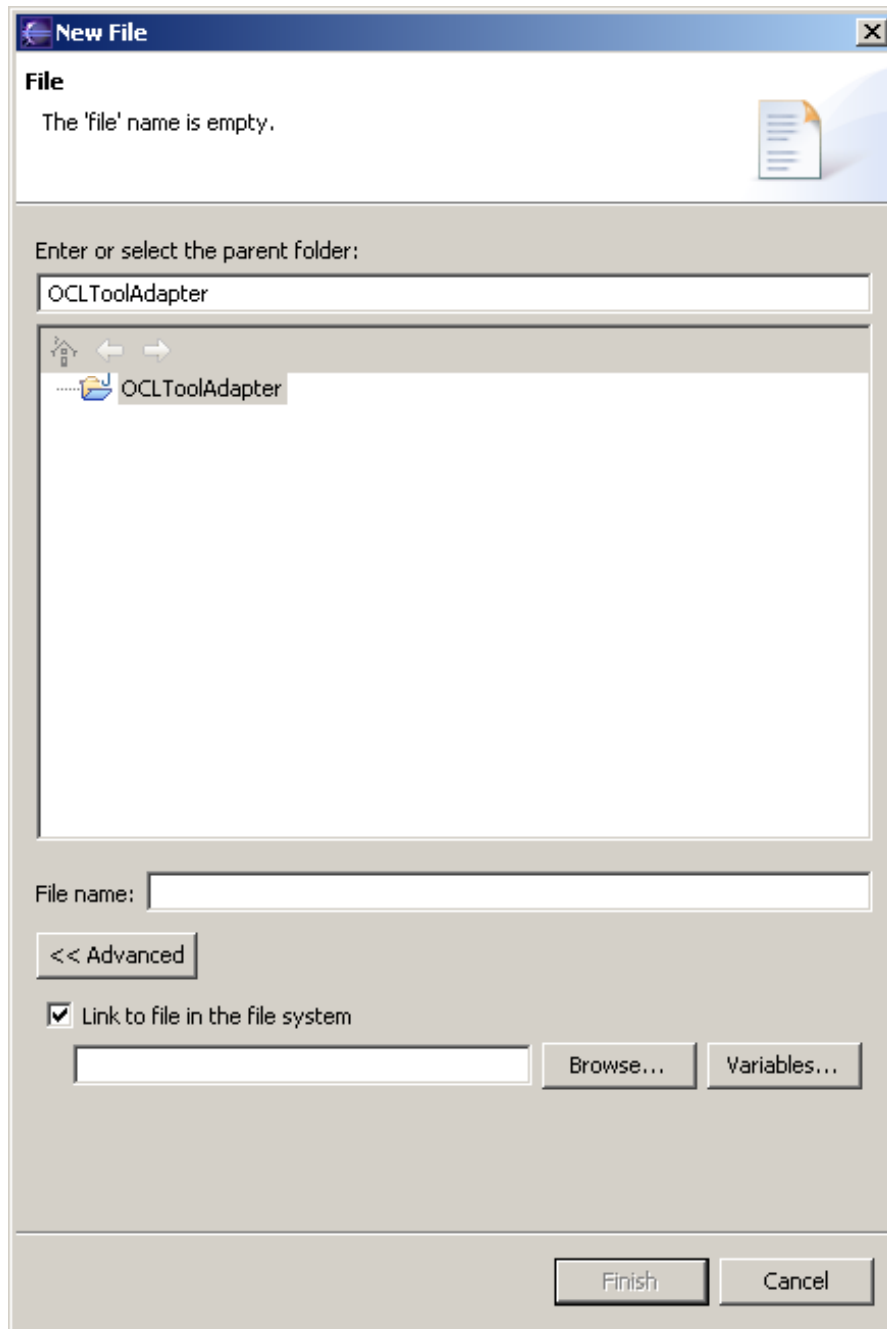


6. Do the same for the other jar file "org.eclipse.mddi.modelbus.description" from the « release /stand-alone » directory.
7. Finally add the « org.eclipse.mddi.modelbus.toolkit.jar » in the « release » subdirectory of your ModelBus Toolkit extract path
8. Press « Finish »

9. Import a description file, right-clicking the top node in the « Packages Explorer » tree view that holds your project name, selecting « New / File » from the context menu



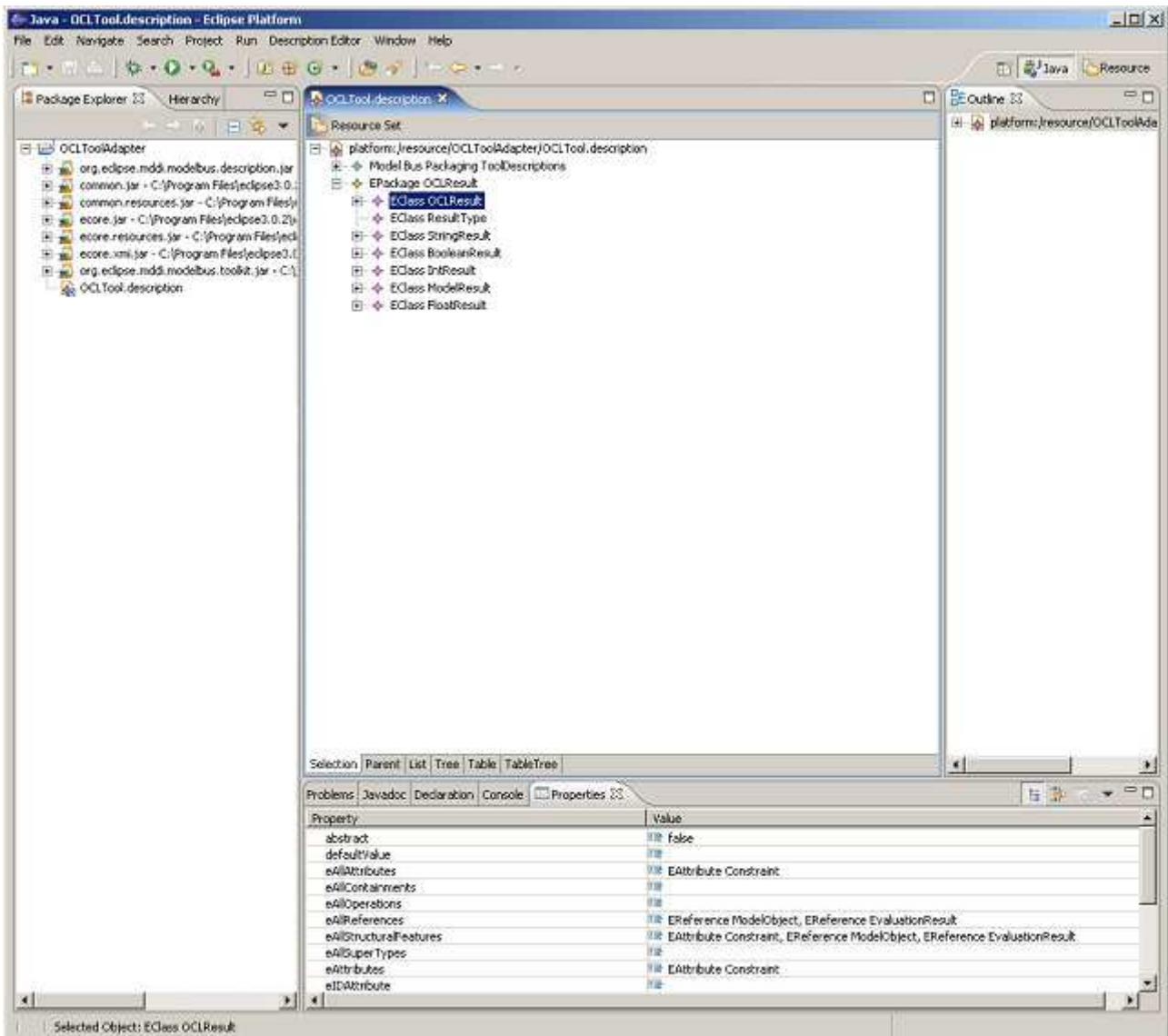
10. On the new file window, press the « Advanced >> » button, click the « Link to file in the file system » check box, and press « Browse »



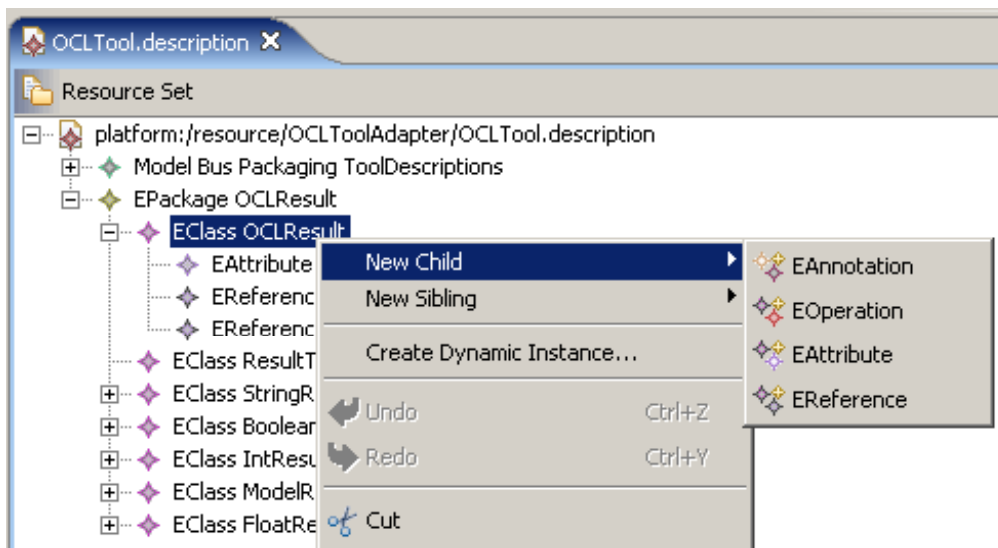
11. Browse your file system to select a description file, for instance the existing « OCLTool.description » example (section 9.1), located in the « resources » subdirectory of your ModelBus Toolkit extract path. Otherwise, create your own new .description file and continue tutorial steps from 3.2.
12. Name the file « OCLTool.description » and press « Finish »

3.2. Edit the tool description

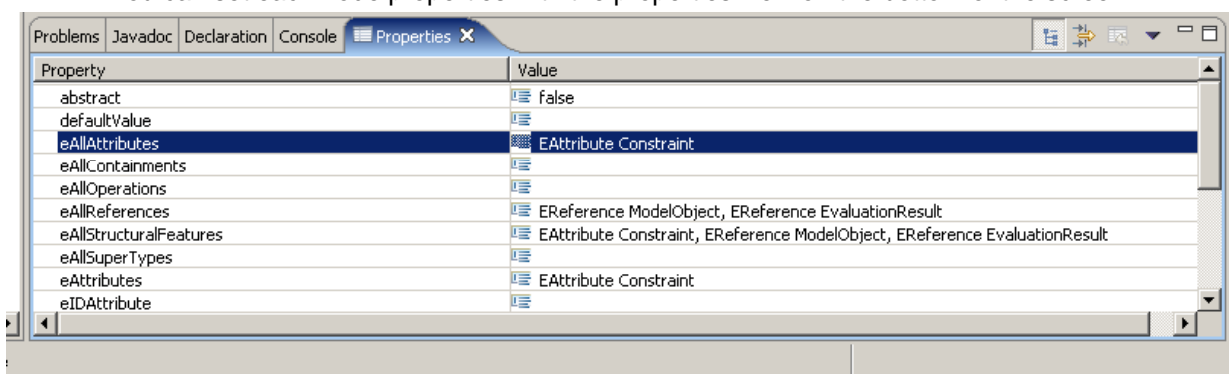
1. Open the description file by double-clicking the « OCLTool.description » node in the left view



2. Browse the descriptor, folding and unfolding nodes. You can add and remove child nodes with the contextual menu according to the FORMATING RULES(see section 4 for further information)



3. Open the properties view, selecting "Window / Show View / Other..." from the main menu bar, select "Properties" and press "OK"
4. You can set each node properties with the properties view on the bottom of the screen



4. FORMATING RULES

4.1. Create a description file

- You may create a description file using EMF (Eclipse project) and the Modelbus metamodel.ecore : load the.ecore into EMF and create a new description model. Add a Tool child to its main node (a special ModelBusPackaging node without name). See par 1.10 how to go on and complete your tool description.
- You may create "manually" a description file : it is an XML 2.0 file.
Its main node is a special ModelBusPackaging node. This main node must contain exactly one Tool node (see below for further information) and must contain at least one ModelingServiceInterface (as child or in its other ModelbusPackaging children) it is necessary for generating the adapter (your Tool node references a ModelingServiceInterface). This special main ModelbusPackaging can contains, Type, Event or other ModelbusPackaging.

For instance :

```
<tooldescription:ModelBusPackaging xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abstract="http://modelbus/description/abstract"
xmlns:concrete="http://modelbus/description/concrete"
xmlns:tooldescription="http://modelbus/description">
```

//here the description of your tool

```
<content xsi:type="concrete:Tool" name="toolname" interface="//@content.0" platform="" /> //the
interface references a ModelingServiceInterface of your description ( it means that your
description file must have one)
</tooldescription:ModelBusPackaging>
```

4.2. Xmi referencement rules:

eventInfo="//@content.4" means that the type referenced by eventInfo is the Type specified by the fifth child of the main node

eventInfo="//@content.0/@content.4" means that the type referenced by eventInfo is the Type specified by the fifth child of the first child of the main node.

eventInfo="//@content.0 //@content.4" means that the types referenced by eventInfo are the Types specified by the fifth and first child of the main node

4.3. Node description and containment rules

Event nodes

- EventType nodes have a name (cf. "Name constraints" below) and a type.

For example :

```
<content xsi:type="abstract:Event" name="eventName" eventInfo="//@content.0
//@content.4"/>
// eventInfo references the type(s) of the Event
// (eventInfo is the only attribute to allow multiple type)
```

Error nodes

- Error nodes have a name (cf. "Name constraints" below) and a type and can have upper and lower bounds (cf. "Lower and Upper constraints" below).

For Example :

```
//with bounds
< serviceError name="errorName" type="//@content.3" upper="-1" lower="1"/>
// without bounds
<serviceError name="errorName" type="//@content.3"/>
// for now, default bounds are lower 1, upper 1 (cf. ModelBus metamodel)
```

ModelBusPackaging nodes

- ModelBusPackaging nodes have a name (cf. "Name constraints" below), and can contain ModelBusPackaging nodes, ModelingServiceInterface nodes, Event nodes and Type nodes (PrimitiveTypes or ModelTypes).

At least one ModelBusPackaging must contain a ModelingServiceInterface.

For example:

```
<content xsi:type="tooldescription:ModelBusPackaging" name="ServicePackage">
//contents of package (ModelingServiceInterface, Type, ModelbusPackaging, Event)
</content>
```

ModelingServiceInterface nodes

- ModelingServiceInterface nodes have a name (cf. "Name constraints" below), and can contain ModelingService nodes and/or can reference EventType nodes (referenced by the publishedEvent attribute).

For example :

```
<content xsi:type="abstract:ModelingServiceInterface" name="msi1"
publishedEvent="//@content.2 //@content.3"> //msi1 references two Events
//contents of interface (i.e ModelingServices)
</content>
```

ModelingService nodes

- ModelingService nodes have a name (cf. "Name constraints" below) and a type, and can contain Parameter nodes (one at least, according to ModelBus metamodel) and Error nodes.

For example :

```
<service name="serviceName"> //ModelingService node
//contents of service ( parameters, errors)
</service>
```

Parameter nodes

- Parameter nodes have a name and type, optionally you may specify parameter direction (the options are "in", "out", "inout", default is "in"), and upper and lower bounds (cf. "Lower and Upper constraints") for the parameter.

```
//without direction and upper and lower bounds
<parameter name="param1" type="//@content.2"/>
```

```
// with direction
<parameter name="param2" type="//@content.3" direction="out"/>
```

```
// with upper and lower bounds
<parameter name="param2" type="//@content.3" upper="1" lower="0"/>
```

Property nodes

- Property nodes have a name (cf. "Name constraints" below) and a value, expressed as strings.

For example :

```
<property name="URL" value="http://localhost:8081/testTool"/>
```

Property names are :
Session (possible values : true, false),
Communication (synchronous, asynchronous),
Identification (true, false),
ModelEncoding (XMI2.0, XMI2.1, EMF2.1).

Tool nodes

- Tool node has a name (cf. "Name constraints" below), an interface and optionally a platform attribute and Property nodes.

For example :

```
<content xsi:type="concrete:Tool" name="testTool" interface="//@content.0" platform="">
//contents of Tool properties if needed
<property name="URL" value="http://localhost:8081/testTool"/>
<property name="Session" value="true"/>
</content>
```

MetaclassSpecification nodes

- MetaclassSpecification nodes have a name (cf. "Name constraints" below) and can have upper and lower bounds (cf. "Lower and Upper constraints" below).

For example :

```
//with bounds
<content name="metaClassName" upper="-1" lower="1"/>

// without bounds
<content name="metaS1"/>
// for now, default bounds are lower 0, upper 0 (cf. ModelBus metamodel)
```

Primitive / ModelType nodes

- Type nodes (either PrimitiveType or ModelType) have a name, and if they are ModelType nodes can have MetaclassSpecification nodes.

For example :

```
// names of primitive type nodes must be "boolean", "double", "integer" or "string"
<content xsi:type="abstract:PrimitiveType" name="string"/>
//model type nodes
<content xsi:type="abstract:ModelType" name="modelTypeName">
//contents of ModelType if needed
<content name="metaClassName" upper="-1" lower="1"/>
<content name="metaClassName2" upper="1" lower="1"/>
...
</content>
```

4.4. Attribute constraints

Lower and Upper constraints

Lower and upper attributes must respect following rules:

- lower and upper are integer
- lower ≥ 0
- upper > 0 or upper = -1 (for infinity)
- lower \leq upper

Lower and upper attributes have default values.

But these default values are able to change (depending from the ModelBus metamodel), so be aware of error message when generating an adapter.

Name attributes

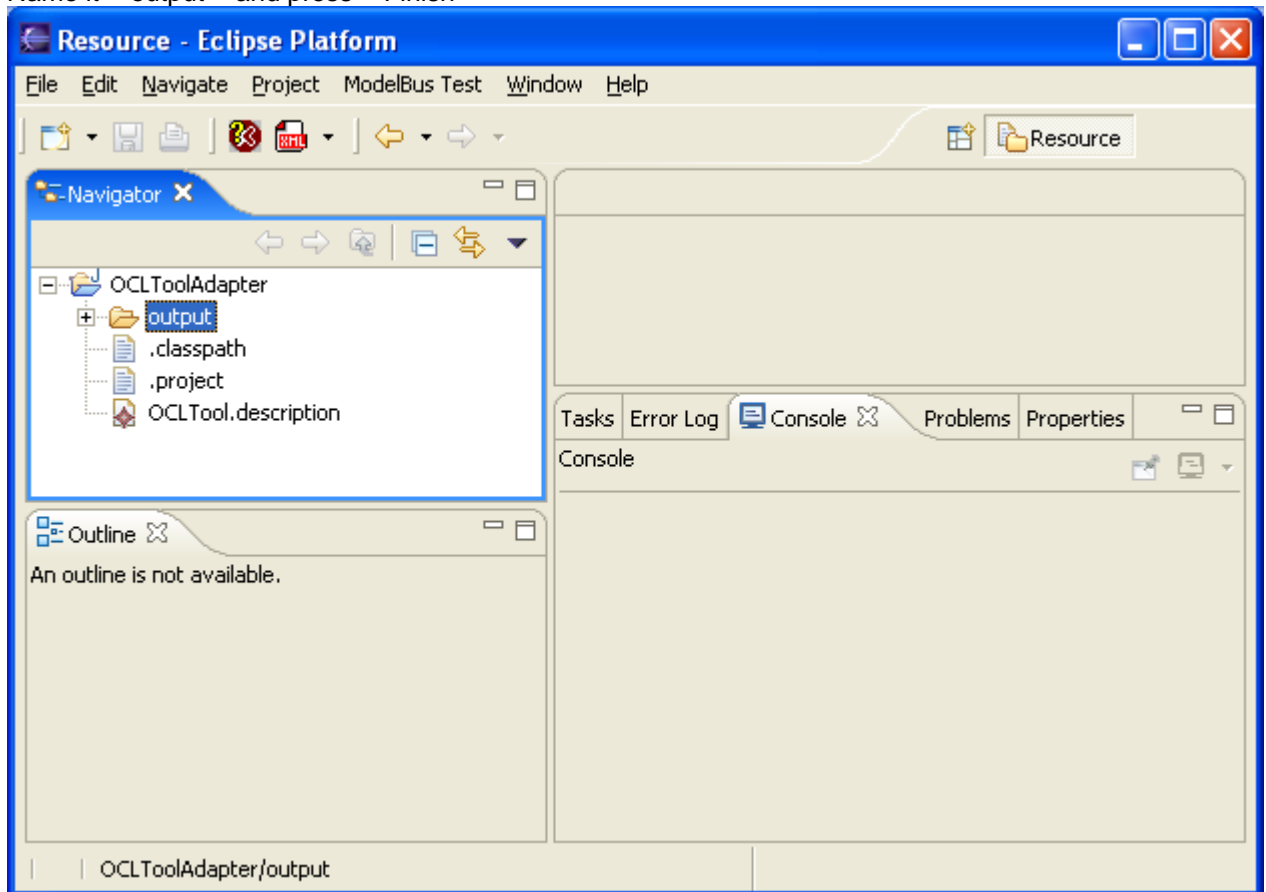
Some attribute of the tool description must respect these rules, according to Java code:
A name must only contains these characters

- a-z
- A-Z
- 0-9 (except for the first character)
- _
- \$
- and the name must not correspond to a Java reserved keywords, and must not be empty.

5. GENERATE STUB AND SKELETON FOR YOUR TOOL

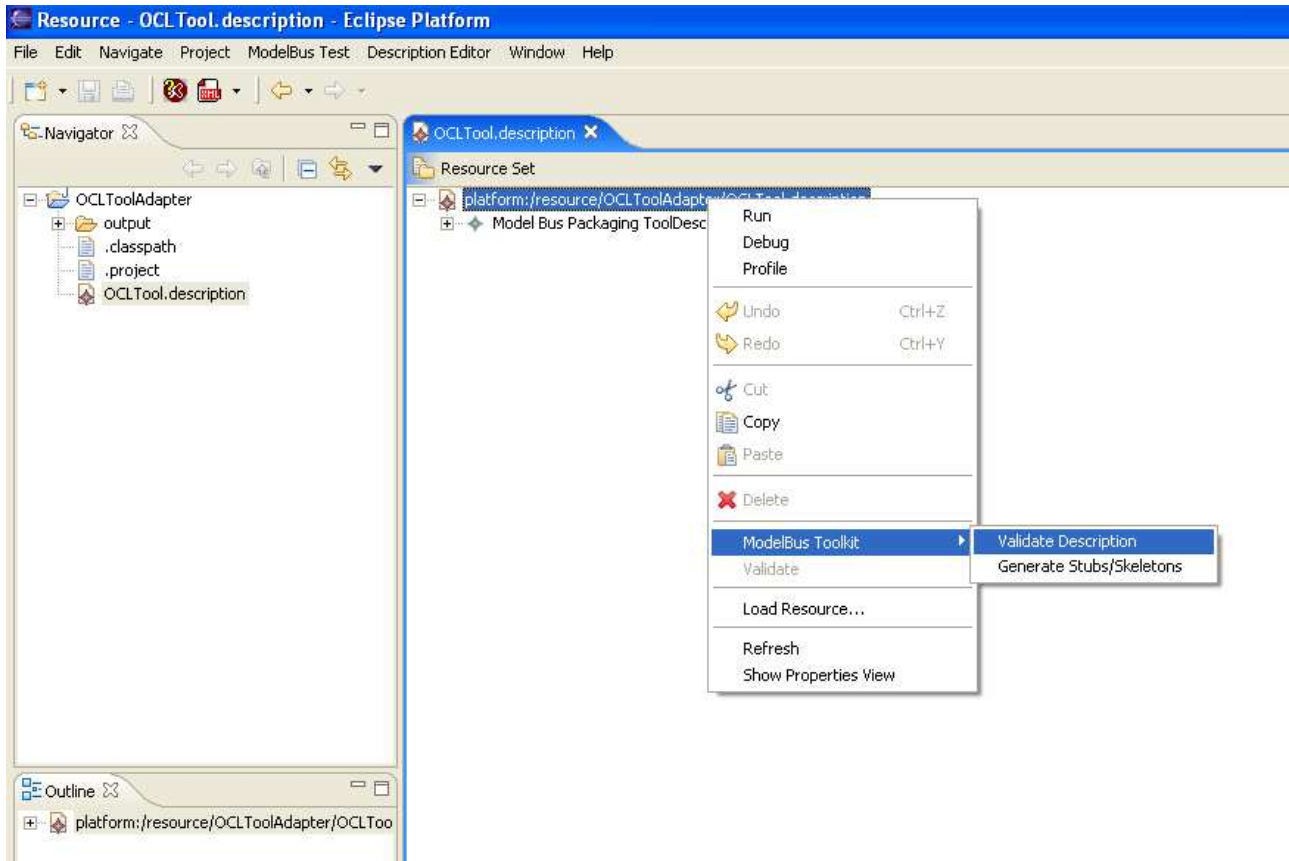
5.1. Create a Folder

1. Create a folder, right-clicking the top node in the « Packages Explorer » tree view that holds your project name, selecting « New / Folder » from the context menu
2. Name it « output » and press « Finish »



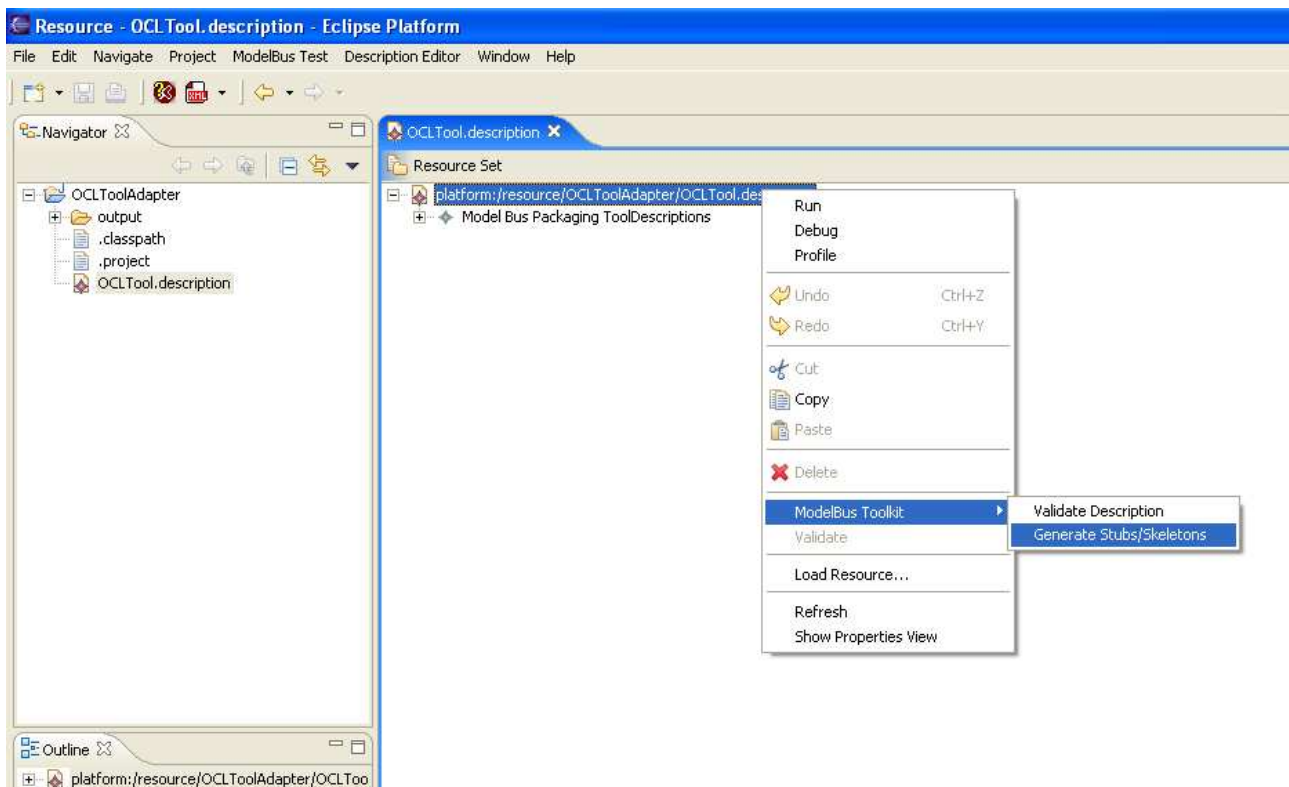
5.2. Validate Tool Description

3. Open Tool Description file
4. Right-click on any description element and chose ModelBus Toolkit->Validate Description

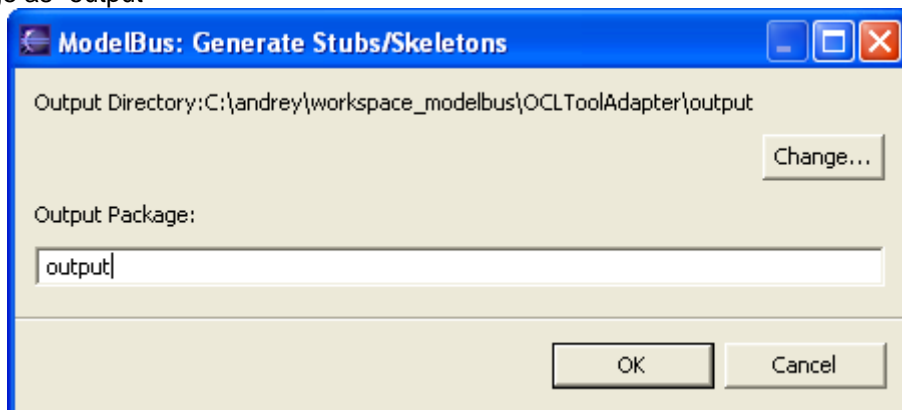


5.3. Generate Stubs/Skeletons

5. Open Tool Description file
6. Right-click on any description element and chose ModelBus Toolkit->Generate Stubs/Skeletons

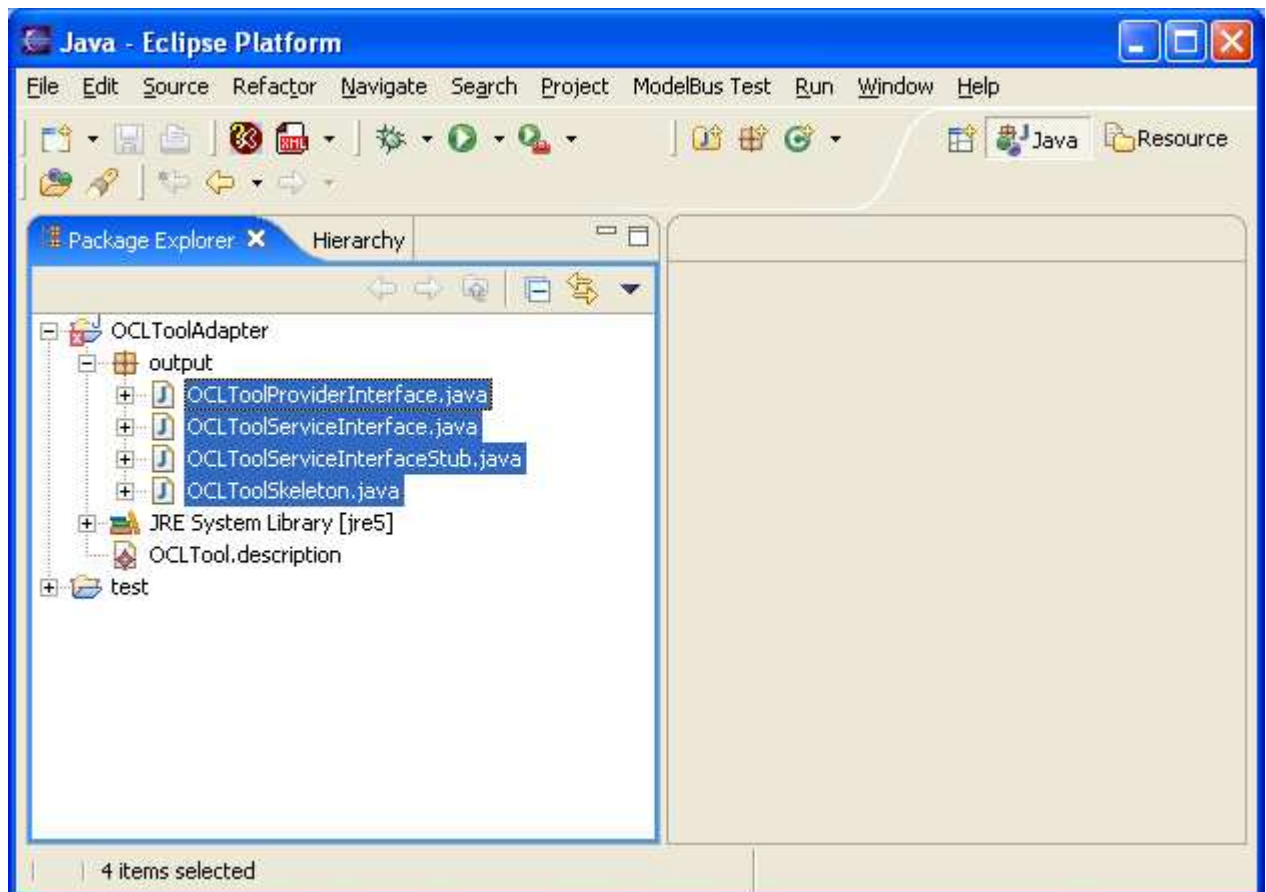


7. In the dialog windows change Output Directory to "...\OCLToolAdapter\output" and define Output Package as "output"



8. Click Ok

5.4. Expected Result



9. Press "F5" (refresh) on your output directory (Eclipse notify an error because "output" package name is different from the "ocltool" package name in the java files generated, nothing's serious ...) :
10. Here you are! Your adapter is generated and ready to be filled-in to plug to Modelbus!
11. The generated files can be found in the Annexes 9.2, 9.3, 9.4 and 9.5.

6. USE STUB IN CLIENT APPLICATION

In this section we describe how to configure and use the generated stub in a client application.

1. Add ModelBus adapter in class path of you project
2. Create java class for your client application : SampleOclClient
3. Add main method to the SampleOclClient class
4. Create a property referencing ModelBus Registry location.

```
//Default ModelBus Registry location
String registry_loc =
"http://monarch.fokus.fraunhofer.de:8080/WebRegistry/services/WebRegistry";
//configuration
Properties p = new Properties();
p.put(AdapterStub.PROP_REGISTRY_LOCATION, registry_loc);
```

5. Setup stub.

```
//Setup
OCLToolServiceInterface ocltool = new OCLToolServiceInterfaceStub(p);
```

6. The stub is ready to use

```
//papare inputs
Collection testmodel = new Vector();
EcoreFactory factory = EcorePackageImpl.init().getEcoreFactory();
```

```

EPackage pack = factory.createEPackage();
testmodel.add(pack);
String [] constraints = new String [1];
constraints [0] = "context Model inv: Package.allInstances()-
>forall(p|p.ownedMember->select(c|c.oclIsTypeOf(Class))->size())<6)";

//usage
Collection[] result = ocltool.consume_checkUML2(testmodel,constraints);

```

7. You can find source code of this example in section 9.6.
8. For more information about model type representation, please see Java type mapping section in the ModelBus Integration Guide.

7. USE SKELETON IN SERVER APPLICATION

In this section we describe how to configure and use the generated stub in a client application.

1. Add ModelBus adapter in class path of you project
2. Create java class for your modeling service : SampleOclProviderImpl which should implement OCLToolProviderInterface. You can use Eclipse to generate a default implementation as it is shown in 9.7
3. Implement generated ProviderInterface. For the OCLTool case look like in 9.7:

```

public class SampleOclProviderImpl implements OCLToolProviderInterface{

    /* (non-Javadoc)
     * @see OCLToolProviderInterface#execute_checkUML2(java.util.Collection,
     java.lang.String[])
     */
    public Collection[] execute_checkUML2(Collection UML2Model, String[] Constraints)
throws ModelingServiceError, SessionNeeded {
        // TODO Auto-generated method stub
        return null;
    }
...

```

Whereas you need to develop execute methods which represent service business logic (here, OCL checker).

4. Create a deployment class: SampleOclProviderDeploy as it is shown in 9.8. which deploys provider for Java stand-alone platform. The important part is the following one:
5. Setup Registry location and path to the OCLTool description file

```

//Default Registry
String registry_loc =
"http://monarch.fokus.fraunhofer.de:8080/WebRegistry/services/WebRegistry";
//Put absolute path to OCLTool.description here
String desc_file = "./OCLTool.description";

try {
    //prepare properties
    Properties p = new Properties();
    p.put(AdapterStub.PROP_REGISTRY_LOCATION, registry_loc);
    p.put(AdapterStub.PROP_TOOL_DESC_FILE, desc_file);

```

6. Create Adapter

```

//create adapter
AdapterStub adapter = new AdapterStubImpl(p);

```

7. Create Provider


```
//create provider
GenericProvider provider = new OCLToolSkeleton(new SampleOclProviderImpl());
```

8. Link Adapter with Provider

```
//link adapter and provider
adapter.getToolStub().setProvider(provider);
```

9. Deploy Adapter

```
//deploy adapter
adapter.deploy();
} catch (Exception e) {
    e.printStackTrace();
}
```

7.1. Test

10. Run SampleOclProviderDeploy
11. Run SampleOclClient

8. Acknowledgements

Authors: Modelbus Toolkit developers: Fateh Bekhouche, Fatima Fadil, Nicolas Garandeau, Nils Henner, Saoussen Kraiem, Philippe Kych, Rémy-Christophe Schermesser, Marc Schwitzguébel, Andrey Sadovykh

9. Appendix

9.1. OCLTool Description

```
<?xml version="1.0" encoding="UTF-8"?>
<tooldescription:ModelBusPackaging xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:abstract="http://modelbus/description/abstract"
xmlns:concrete="http://modelbus/description/concrete"
xmlns:tooldescription="http://modelbus/description" name="ToolDescriptions">
  <content xsi:type="tooldescription:ModelBusPackaging" name="ServicePackage">
    <content xsi:type="abstract:ModelingServiceInterface" name="OCLToolServiceInterface">
      <service name="checkUML2">
        <parameter name="UML2Model" type="//@content.1/@content.1"/>
        <parameter name="Constraints" type="//@content.1/@content.0" upper="-1"/>
        <parameter name="Result" type="//@content.1/@content.2" direction="out" upper="-1"/>
      </service>
      <service name="checkOCL">
        <parameter name="MetaModel" type="//@content.1/@content.3"/>
        <parameter name="InstanceModel" type="//@content.1/@content.4"/>
        <parameter name="Constraints" type="//@content.1/@content.0" upper="-1"/>
        <parameter name="Result" type="//@content.1/@content.2" direction="out" upper="-1"/>
      </service>
    </content>
  </content>
  <content xsi:type="tooldescription:ModelBusPackaging" name="TypePackage">
    <content xsi:type="abstract:PrimitiveType" name="string"/>
    <content xsi:type="abstract:ModelType" name="UML2Model"/>
    <content xsi:type="abstract:ModelType" name="OCLResult">
      <content name="content" lower="0" upper="1">
        <metaClass
href="file:/C:/andrey/workspace_modelbus/Test/OCLResult.ecore#//OCLResult"/>
      </content>
    </content>
    <content xsi:type="abstract:ModelType" name="MetaModel"/>
    <content xsi:type="abstract:ModelType" name="InstanceModel"/>
  </content>
  <content xsi:type="concrete:Tool" name="OCLTool" interface="//@content.0/@content.0"/>
</tooldescription:ModelBusPackaging>
```

9.2. OCLToolServiceInterface

```
import java.util.Collection;
import java.util.Properties;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ModelBusCommunicationException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ModelTypeMismatchException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.NoToolAvailableException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ServiceUnknownException;
import org.eclipse.mddi.modelbus.adapter.infrastructure.DeploymentException;
import org.eclipse.mddi.modelbus.adapter.user.impl.AdapterStubImpl;
import org.eclipse.mddi.modelbus.adapter.user.AdapterStub;
import org.eclipse.mddi.modelbus.adapter.user.consumer.GenericConsumer;

/**
 * OCLToolServiceInterface.java
 *
 * @author auto-generated
 *
 * @version $Revision: 1.2 $ $Date: 2006/03/13 12:33:31 $
 */
public interface OCLToolServiceInterface {
```

```

/**
 *
 * consume_checkUML2
 *
 * @param UML2Model
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] consume_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
;

/**
 *
 * consumeAsync_checkUML2
 *
 * @param UML2Model
 * @param Constraints
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public void consumeAsync_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
;

/**
 *
 * isResultReady_checkUML2
 *
 * @return boolean
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public boolean isResultReady_checkUML2() throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
;

/**
 *
 * getResult_checkUML2
 *
 * @param UML2Model
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] getResult_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
;

```

```

/**
 *
 * consume_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] consume_checkOCL(java.util.Collection MetaModel,
java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
ModelTypeMismatchException, Exception ;

/**
 *
 * consumeAsync_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public void consumeAsync_checkOCL(java.util.Collection MetaModel,
java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
ModelTypeMismatchException, Exception ;

/**
 *
 * isResultReady_checkOCL
 *
 * @return boolean
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public boolean isResultReady_checkOCL() throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
;

/**
 *
 * getResult_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */

```

```

        public java.util.Collection[] getResult_checkOCL(java.util.Collection MetaModel,
        java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
        ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
        ModelTypeMismatchException, Exception ;
    }

```

9.3. OCLToolServiceInterfaceStub

```

import java.util.Collection;
import java.util.Properties;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ModelBusCommunicationException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ModelTypeMismatchException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.NoToolAvailableException;
import org.eclipse.mddi.modelbus.adapter.user.consumer.ServiceUnknownException;
import org.eclipse.mddi.modelbus.adapter.infrastructure.DeploymentException;
import org.eclipse.mddi.modelbus.adapter.user.impl.AdapterStubImpl;
import org.eclipse.mddi.modelbus.adapter.user.AdapterStub;
import org.eclipse.mddi.modelbus.adapter.user.consumer.GenericConsumer;

/**
 * OCLToolServiceInterfaceStub.java
 *
 * @author auto-generated
 *
 * @version $Revision: 1.2 $ $Date: 2006/03/13 12:33:31 $
 */
public class OCLToolServiceInterfaceStub implements OCLToolServiceInterface {
    public AdapterStub adapter;
    public GenericConsumer consumer;

    /**
     *
     * @param pro Adapter properties
     * @throws DeploymentException
     */
    public OCLToolServiceInterfaceStub(Properties pro) throws DeploymentException {
        adapter = new AdapterStubImpl(pro);
        consumer = adapter.getGenericConsumer();
    }

    /**
     *
     * consume_checkUML2
     *
     * @param UML2Model
     * @param Constraints
     * @return java.util.Collection[]
     * @throws ModelBusCommunicationException
     * @throws ServiceUnknownException
     * @throws NoToolAvailableException
     * @throws ModelTypeMismatchException
     * @throws Exception
     */
    public java.util.Collection[] consume_checkUML2(java.util.Collection UML2Model,
    java.lang.String[] Constraints) throws ModelBusCommunicationException,
    ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
    {
        String serviceName = "OCLTool.OCLToolServiceInterface.checkUML2";
        Object[] inputs = new Object[2];
        inputs[0] = UML2Model;
        inputs[1] = Constraints;
        Object[] outputs = consumer.consume(serviceName, inputs);
        return (java.util.Collection[])outputs[0];
    }
}

```

```

/**
 *
 * consumeAsync_checkUML2
 *
 * @param UML2Model
 * @param Constraints
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public void consumeAsync_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
{
    String serviceName = "OCLTool.OCLToolServiceInterface.checkUML2";
    Object[] inputs = new Object[2];
    inputs[0] = UML2Model;
    inputs[1] = Constraints;
    consumer.consumeAsync(serviceName, inputs);
}

/**
 *
 * isResultReady_checkUML2
 *
 * @return boolean
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public boolean isResultReady_checkUML2() throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
{
    String serviceName = "OCLTool.OCLToolServiceInterface.checkUML2";
    return consumer.isResultReady(serviceName);
}

/**
 *
 * getResult_checkUML2
 *
 * @param UML2Model
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] getResult_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
{
    String serviceName = "OCLTool.OCLToolServiceInterface.checkUML2";
    Object[] outputs = consumer.getResult(serviceName);
    return (java.util.Collection[])outputs[0];
}

```

```

/**
 *
 * consume_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] consume_checkOCL(java.util.Collection MetaModel,
java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
ModelTypeMismatchException, Exception {
    String serviceName = "OCLTool.OCLToolServiceInterface.checkOCL";
    Object[] inputs = new Object[3];
    inputs[0] = MetaModel;
    inputs[1] = InstanceModel;
    inputs[2] = Constraints;
    Object[] outputs = consumer.consume(serviceName, inputs);
    return (java.util.Collection[])outputs[0];
}

/**
 *
 * consumeAsync_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public void consumeAsync_checkOCL(java.util.Collection MetaModel,
java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
ModelTypeMismatchException, Exception {
    String serviceName = "OCLTool.OCLToolServiceInterface.checkOCL";
    Object[] inputs = new Object[3];
    inputs[0] = MetaModel;
    inputs[1] = InstanceModel;
    inputs[2] = Constraints;
    consumer.consumeAsync(serviceName, inputs);
}

/**
 *
 * isResultReady_checkOCL
 *
 * @return boolean
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */

```

```

        public boolean isResultReady_checkOCL() throws ModelBusCommunicationException,
ServiceUnknownException, NoToolAvailableException, ModelTypeMismatchException, Exception
{
    String serviceName = "OCLTool.OCLToolServiceInterface.checkOCL";
    return consumer.isResultReady(serviceName);
}

/**
 *
 * getResult_checkOCL
 *
 * @param MetaModel
 * @param InstanceModel
 * @param Constraints
 * @return java.util.Collection[]
 * @throws ModelBusCommunicationException
 * @throws ServiceUnknownException
 * @throws NoToolAvailableException
 * @throws ModelTypeMismatchException
 * @throws Exception
 */
public java.util.Collection[] getResult_checkOCL(java.util.Collection MetaModel,
java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
ModelBusCommunicationException, ServiceUnknownException, NoToolAvailableException,
ModelTypeMismatchException, Exception {
    String serviceName = "OCLTool.OCLToolServiceInterface.checkOCL";
    Object[] outputs = consumer.getResult(serviceName);
    return (java.util.Collection[])outputs[0];
}
}

```

9.4. OCLToolProviderInterface

```

import java.util.Collection;
import org.eclipse.mddi.modelbus.adapter.user.ModelingServiceError;
import org.eclipse.mddi.modelbus.adapter.user.SessionNeeded;

/**
 * OCLToolProviderInterface.java
 *
 * @author auto-generated
 *
 * @version $Revision: $ $Date: $
 */

public interface OCLToolProviderInterface {

    /**
     *
     * execute_checkUML2
     *
     * @param UML2Model
     * @param Constraints
     * @return java.util.Collection[] - Result
     * @throws ModelingServiceError
     * @throws SessionNeeded
     */
    public java.util.Collection[] execute_checkUML2(java.util.Collection UML2Model,
java.lang.String[] Constraints) throws ModelingServiceError, SessionNeeded;
}

```



```

    /**
     *
     * execute_checkOCL
     *
     * @param MetaModel
     * @param InstanceModel
     * @param Constraints
     * @return java.util.Collection[] - Result
     * @throws ModelingServiceError
     * @throws SessionNeeded
     */
    public java.util.Collection[] execute_checkOCL(java.util.Collection MetaModel,
    java.util.Collection InstanceModel, java.lang.String[] Constraints) throws
    ModelingServiceError, SessionNeeded;
}

```

9.5. OCLToolSkeleton

```

import java.util.Collection;
import org.eclipse.mddi.modelbus.adapter.user.ModelingServiceError;
import org.eclipse.mddi.modelbus.adapter.user.SessionNeeded;
import org.eclipse.mddi.modelbus.adapter.user.provider.GenericProvider;

/**
 * OCLToolSkeleton.java
 *
 * @author auto-generated
 *
 * @version $Revision: $ $Date: $
 */

public class OCLToolSkeleton implements GenericProvider {
    private OCLToolProviderInterface concreteTool;

    public OCLToolSkeleton(OCLToolProviderInterface concreteTool) {
        this.concreteTool = concreteTool;
    }

    public Object[] execute(String serviceName, Object[] parameterValues) throws
    SessionNeeded, ModelingServiceError {
        if (serviceName.equals("checkUML2")){
            java.util.Collection[]
result=concreteTool.execute_checkUML2((java.util.Collection)
parameterValues[0],(java.lang.String[]) parameterValues[1]);
            Object[] returnValues = new Object[1];
            returnValues[0]=result;
            return returnValues;
        }else if (serviceName.equals("checkOCL")){
            java.util.Collection[]
result=concreteTool.execute_checkOCL((java.util.Collection)
parameterValues[0],(java.util.Collection) parameterValues[1],(java.lang.String[])
parameterValues[2]);
            Object[] returnValues = new Object[1];
            returnValues[0]=result;
            return returnValues;
        }else {
            throw new ModelingServiceError ("No such service: "+serviceName,
null);
        }
    }
}

```

9.6. SampleOclClient

```
import java.util.Collection;
import java.util.Properties;
import java.util.Vector;

import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EcoreFactory;
import org.eclipse.emf.ecore.impl.EcorePackageImpl;
import org.eclipse.mddi.modelbus.adapter.user.AdapterStub;

/*
 * SampleOclClient.java
 *
 * @author Andrey Sadovnykh (LIP6)
 * @version $Revision: $ $Date: $
 */

public class SampleOclClient {

    public static void main(String[] args) {

        try {
            //Default ModelBus Registry location
            String registry_loc =
"http://monarch.fokus.fraunhofer.de:8080/WebRegistry/services/WebRegistry";
            //configuration
            Properties p = new Properties();
            p.put(AdapterStub.PROP_REGISTRY_LOCATION, registry_loc);

            //setup
            OCLToolServiceInterface ocltool = new OCLToolServiceInterfaceStub(p);

            //prepare inputs
            Collection testmodel = new Vector();
            EcoreFactory factory = EcorePackageImpl.init().getEcoreFactory();
            EPackage pack = factory.createEPackage();
            testmodel.add(pack);
            String [] constraints = new String [1] ;
            constraints [0] = "context Model inv: Package.allInstances()-
>forAll(p|p.ownedMember->select(c|c.oclIsTypeOf(Class))->size()<6)";

            //usage
            Collection[] result = ocltool.consume_checkUML2(testmodel,constraints);

            System.out.println("Output "+result);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

9.7. SampleOclProviderImpl

```
import org.eclipse.mddi.modelbus.adapter.user.ModelingServiceError;
import org.eclipse.mddi.modelbus.adapter.user.SessionNeeded;

/*
 * SampleOclProviderImpl
 *
 */

public class SampleOclProviderImpl implements OCLToolProviderInterface{

    /* (non-Javadoc)
     * @see OCLToolProviderInterface#execute_checkUML2(java.util.Collection,
     java.lang.String[])
     */
    public Collection[] execute_checkUML2(Collection UML2Model, String[] Constraints)
throws ModelingServiceError, SessionNeeded {
        // TODO Auto-generated method stub
        return null;
    }

    /* (non-Javadoc)
     * @see OCLToolProviderInterface#execute_checkOCL(java.util.Collection,
     java.util.Collection, java.lang.String[])
     */
    public Collection[] execute_checkOCL(Collection MetaModel, Collection InstanceModel,
String[] Constraints) throws ModelingServiceError, SessionNeeded {
        // TODO Auto-generated method stub
        return null;
    }

}
```

9.8. SampleOclProviderDeploy

```
import java.util.Properties;

import org.eclipse.mddi.modelbus.adapter.user.AdapterStub;
import org.eclipse.mddi.modelbus.adapter.user.impl.AdapterStubImpl;
import org.eclipse.mddi.modelbus.adapter.user.provider.GenericProvider;

public class SampleOclProviderDeploy {

    public static void main(String[] args) {
        //Default Registry
        String registry_loc = "
http://monarch.fokus.fraunhofer.de:8080/WebRegistry/services/WebRegistry";
        //Put absolute path to OCLTool.description here
        String desc_file = "./OCLTool.description";

        try {
            //prepare properties
            Properties p = new Properties();
            p.put(AdapterStub.PROP_REGISTRY_LOCATION, registry_loc);
            p.put(AdapterStub.PROP_TOOL_DESC_FILE, desc_file);
        }
    }
}
```

```
        //create adapter
        AdapterStub adapter = new AdapterStubImpl(p);

        //create provider
        GenericProvider provider = new OCLToolSkeleton(new SampleOclProviderImpl());

        //link adapter and provider
        adapter.getToolStub().setProvider(provider);

        //deploy adapter
        adapter.deploy();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```