An Introduction into the Dresden OCL2 Toolkit for Eclipse

by Ronny Brandt and Claas Wilke

These tutorial describes the work with the *Dresden OCL2 Toolkit for Eclipse*. This version of the Toolkit is based on the new infrastructure called the "pivot model". The pivot model was developed by Matthias Bräuer and is described in his "Großer Beleg" [Brä07]. Further information about the toolkit is available at the website of the Dresden OCL2 Toolkit [Sofb].

The tutorial starts with the installation of the needed *Eclipse* plugins. Then it is described how to load a domain specific model and a model instance. Afterwords the importation and interpretation of OCL expressions will be explained.

The procedure described in this tutorial is realized and tested with *Eclipse 3.3.2* [Thea]. The tutorial should also run with *Eclipse 3.4*. If method or menu names have been changed in *Eclipse 3.4* this will be noted in the tutorial. Besides *Eclipse* you also need to install the required plugins of the *Eclipse Modeling Framework (EMF)*. During this tutorial the *EMF* plugins of version 2.3.2 were used.

To install the *EMF* plugins you have to download them from the *EMF* website [Theb] and to extract them into the "plugins" directory of *Eclipse*. Afterwords you can start the *Eclipse SDK*. Alternatively you can install the plugins by using the *Eclipse Update Manager* after starting the *Eclipse SDK*.

1 How to install the Dresden OCL2 Toolkit for Eclipse

To use the *Dresden OCL2 Toolkit for Eclipse* you need to install them as *Eclipse* plugins, or to import them into your *Eclipse workspace*. Both possibilities are explained in the following.

1.1 Installing the Eclipse plugins

To install the *OCL2 Toolkit* as *Eclipse* plugins, you need to have the jar archives of the toolkit. The jar archives are available at [Sofa]. You need to copy the jar archives into the "plugins" directory of your *Eclipse SDK* distribution. Then you can start the *Eclipse SDK* and you can work with the Toolkit.

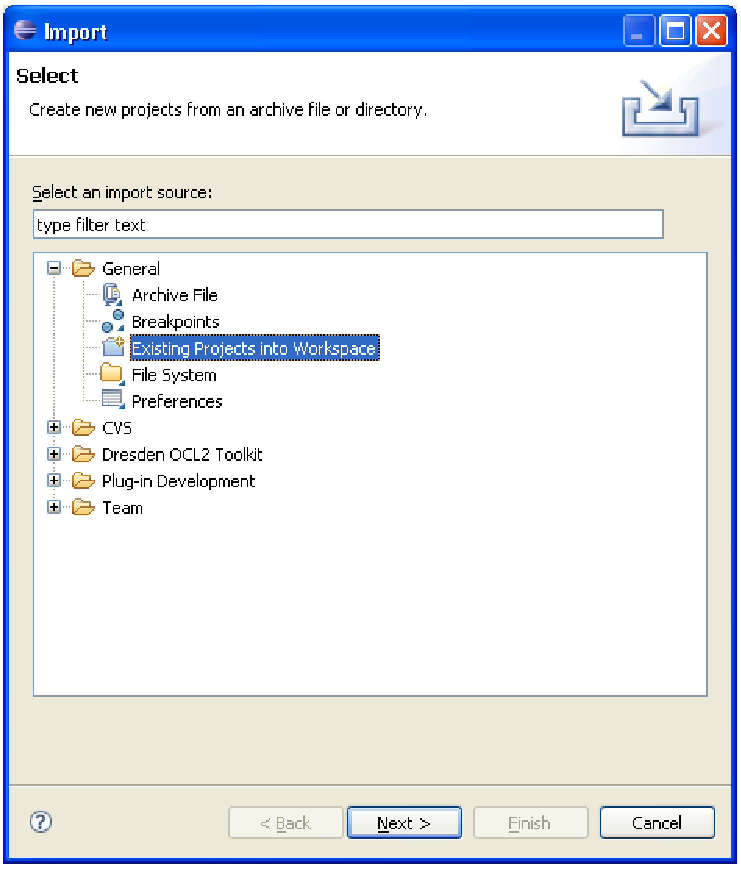
1.2 Importing the Toolkit into an Eclipse Workspace

Alternatively you can import the *OCL2 Toolkit* as plugin projects into an *Eclipse* workspace. There are two different options to do that.

On the one hand you can import the plugins from your file system, if you already have downloaded them (e.g. from [Sofa] as a source code distribution). One the other hand you can import the plugins directly from the *SVN* (*Subversion*) directory of the *Dresden OCL2 Toolkit*. Both possibilities are described below.

1.2.1 Import the plugins from the local file system

Let's say you have the plugins located in a directory XYZ of your file system. To import them into your *Eclipse* workspace you can use the *Eclipse import wizard*. Open the wizard via the menu "File > Import..." and select "General > Existing Projects into Workspace" (see figure 1). In the following window you select the directory XYZ as root. Then you select the plugins you want to import (if not selected automatically) and activate the check box "Copy projects into workspace" (see figure 2). After pressing the button "Finish" the plugins will be imported as projects into your workspace.


Select

Create new projects from an archive file or directory.

Select an import source:

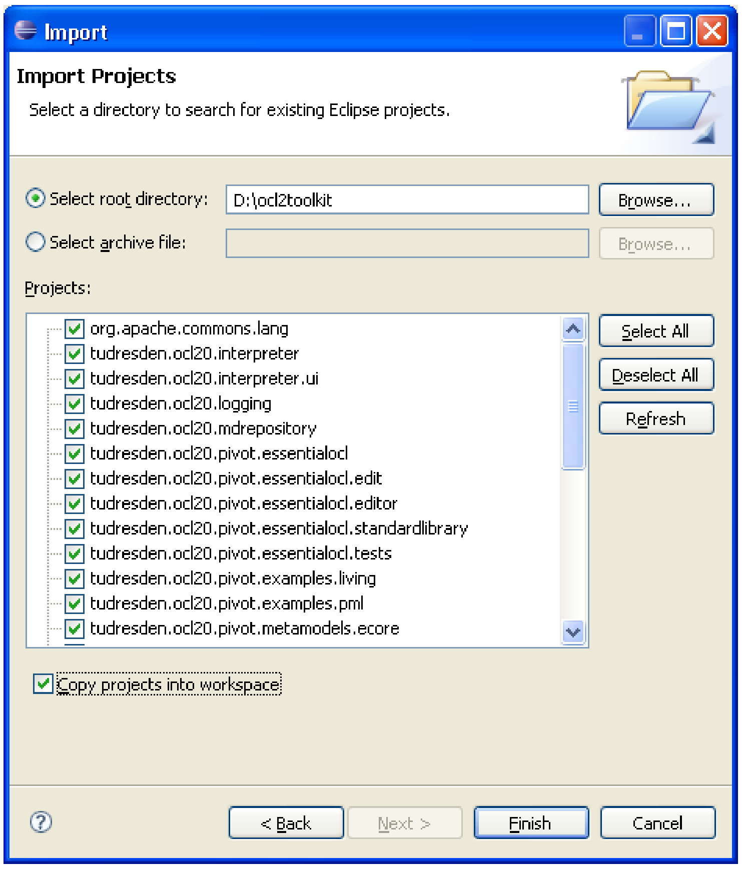
type filter text

Figure 1: Plugin import from local file system (1).

1.2.2 Import the plugin projects from SVN

To import the plugins directly from the *SVN* repository, you need to install an additionally *Eclipse* plugin to connect with the *SVN*. The needed plugin is called *Subclipse* ([Tigb]). The installation of *Subclipse* is explained on the *Supclipse* website [Tiga].

Attention: At the moment the *Subclipse* plugin does not work with *Eclipse 3.4*. To access the *SVN* Repository with *Eclipse 3.4* use another *SVN* plugin for *Eclipse* or use an external tool to access the *SVN* respository.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: D:\ocl2toolkit

( ) Select archive file:

Projects:

- [✓] org.apache.commons.lang
- [✓] tudresden.ocl20.interpreter
- [✓] tudresden.ocl20.interpreter.ui
- [✓] tudresden.ocl20.logging
- [✓] tudresden.ocl20.mdrepository
- [✓] tudresden.ocl20.pivot.essentialocl
- [✓] tudresden.ocl20.pivot.essentialocl.edit
- [✓] tudresden.ocl20.pivot.essentialocl.editor
- [✓] tudresden.ocl20.pivot.essentialocl.standardlibrary
- [✓] tudresden.ocl20.pivot.essentialocl.tests
- [✓] tudresden.ocl20.pivot.examples.living
- [✓] tudresden.ocl20.pivot.examples.pml
- [✓] tudresden.ocl20.pivot.metamodels.ecore

[✓] Copy projects into workspace

Figure 2: Plugin import from local file system (2).

After installing *Subclipse* a new *Eclispe perspective* for access to *SVN* should exist. The perspective can be opened via the menu “Window > Open Perpective > Other... > SVN Repository Exploring“. In the view *SVN Repository* you can add a new repository (see figure 3) using the URL “https://dresden-ocl.svn.sourceforge.net/svnroot/dresden-ocl/“. After pressing the button “Finish“ the SVN repository root should the visible in the *repository view*.

To checkout the plugins, you now select them in the repository directory “trunk/ocl20forEclipse/eclipse“ and use the “Checkout...“ function in the context menu (see figure 4). The given settings could be used and after a click on the “Finish“ button the plugins should be imported.

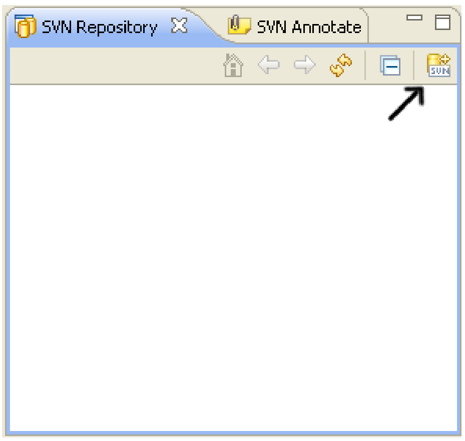


Figure 3: Adding an SVN repository.

2 Building the OCL2 Parser

If you decided to run the *OCL2 Toolkit for Eclipse* as project plugins into your workspace, you need to build the *OCL2 Parser* of the *Dresden OCL2 Toolkit for Eclipse* via an *Ant* build script. If you installed the Toolkit using jar archives, you can skip this chapter of the tutorial.

To build the *OCL2 Parser* select the file “build.xml“ in the project “tudresden.ocl20.pivot.ocl2parser“ and open the context menu via a right mouse click. Select the function “Run As ... > Ant Build“ (see figure 5).

If an error like “Problem: failed to create task or type eclipse.refreshLocal“ occurs, you need to change the configuration of the *Ant* script. Open the function “Properties“ in the context menu of the “build.xml“. A new window should open. Select the topic “Run/Debug settings“ and then the configuration for “tudresden.ocl20.pivot.oclparser build.xml“. Click on the button “Edit“. In the new window select in the sub menu “JRE“ the check box “Run in the same JRE as the workspace“ and click on the button “OK“ (see figure 6). Afterwords the *Ant* script should be executable without errors.

After executing the build script successfully you need to update the projects in your workspace. Update the project “tudresden.ocl20.pivot.oclparser“ via context menu (“Refresh“, see figure 7).

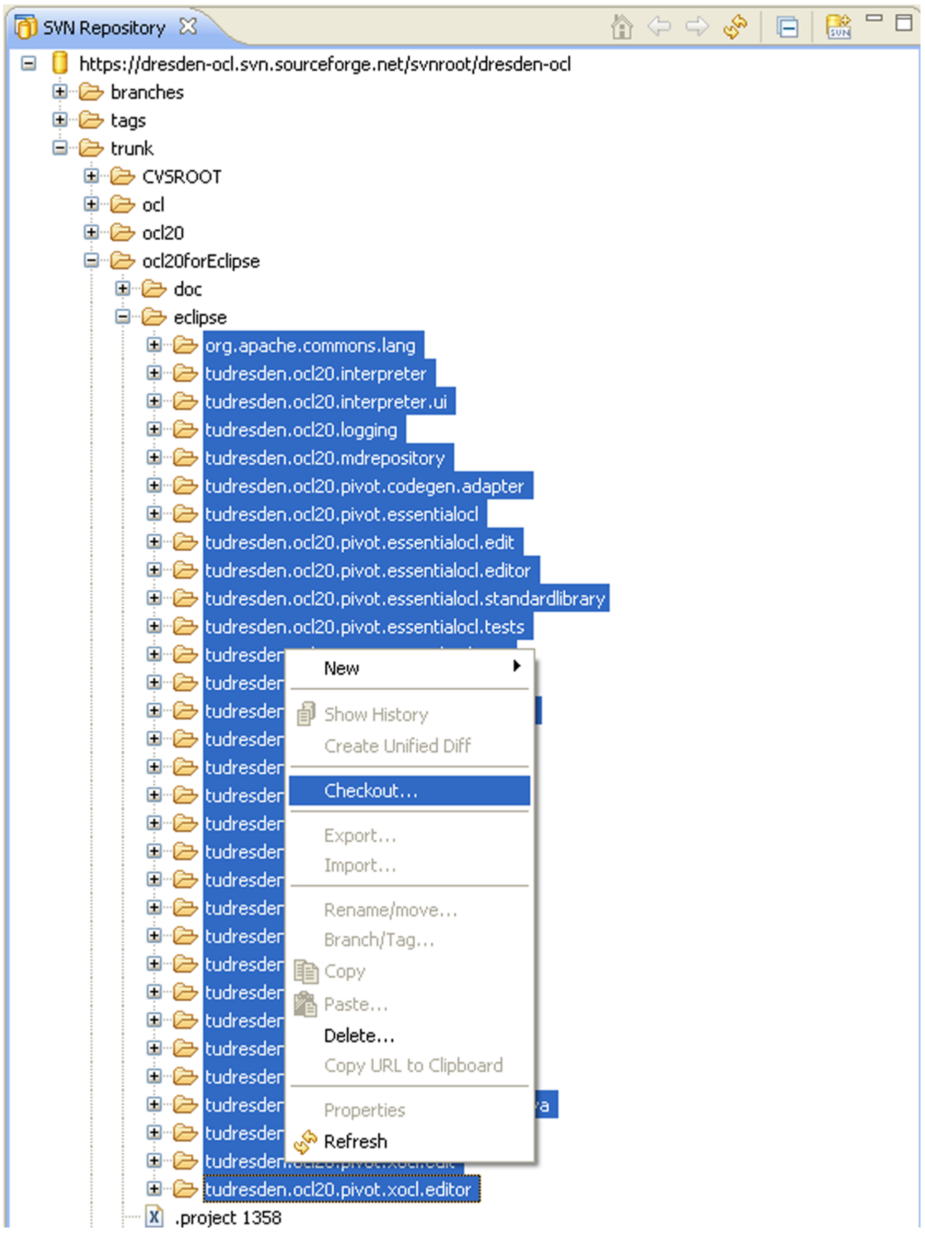

SVN Repository
https://dresden-ocl.svn.sourceforge.net/svnroot/dresden-ocl
branches
tags
trunk
CVSROOT
ocl
ocl20
ocl20forEclipse
doc
eclipse
org.apache.commons.lang
tudresden.ocl20.interpreter
tudresden.ocl20.interpreter.ui
tudresden.ocl20.logging
tudresden.ocl20.mdrepository
tudresden.ocl20.pivot.codegen.adapter
tudresden.ocl20.pivot.essentialocl
tudresden.ocl20.pivot.essentialocl.edit
tudresden.ocl20.pivot.essentialocl.editor
tudresden.ocl20.pivot.essentialocl.standardlibrary
tudresden.ocl20.pivot.essentialocl.tests
tudresden.ocl20.pivot.xocl.editor
.project 1358
New
Show History
Create Unified Diff
Checkout...
Export...
Import...
Rename/move...
Branch/Tag...
Copy
Paste...
Delete...
Copy URL to Clipboard
Properties
Refresh

Figure 4: Checkout of the Dresden OCL2 Toolkit plugin projects.

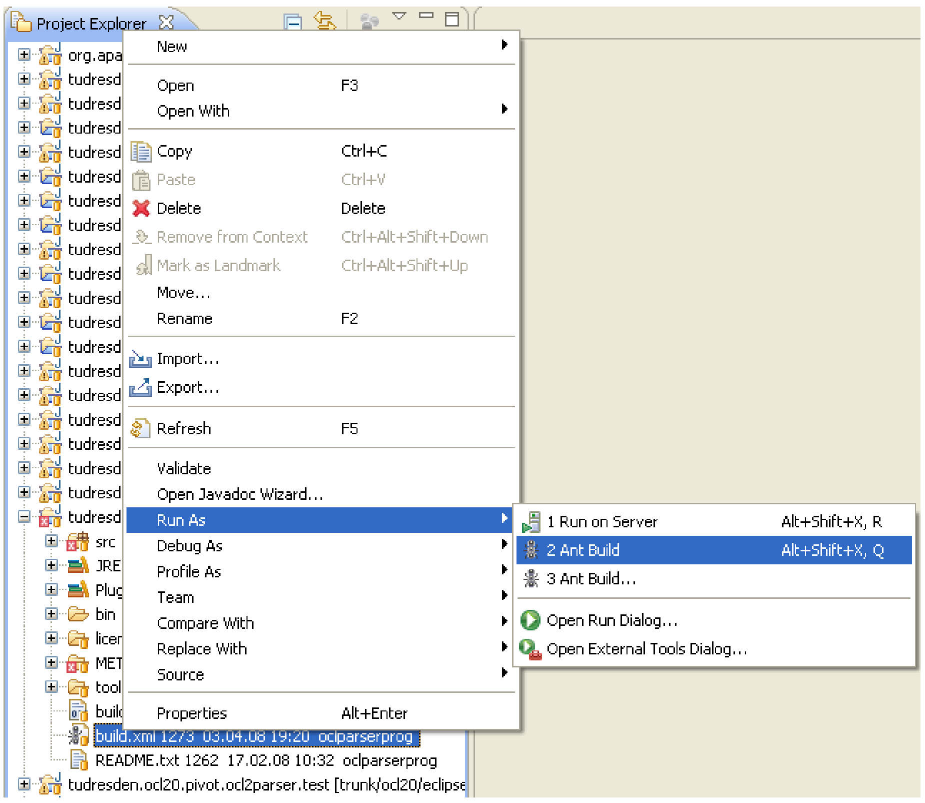
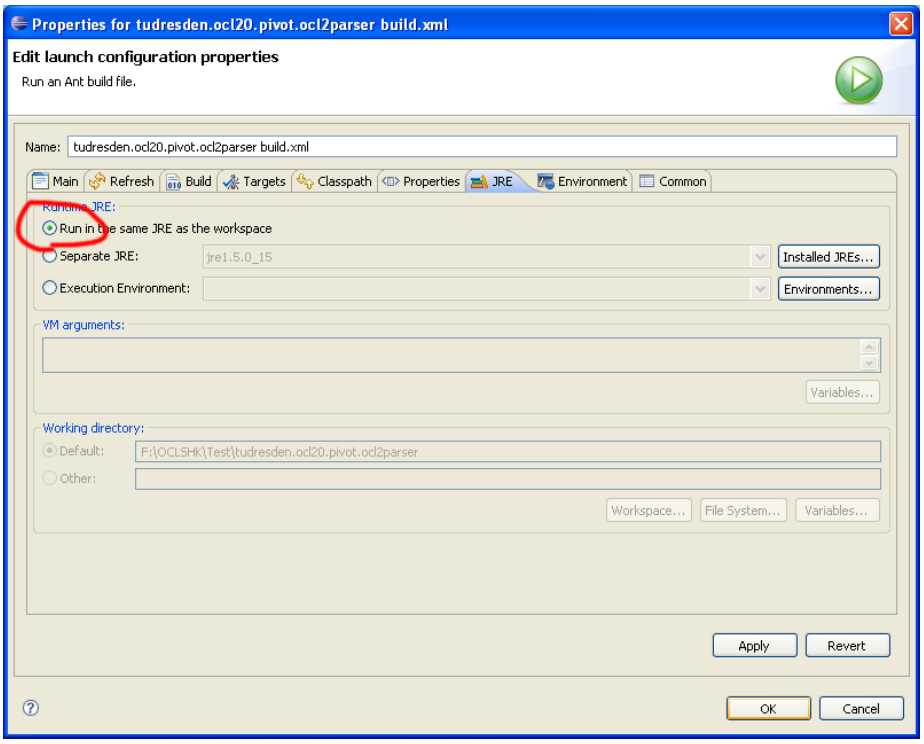

Project Explorer
New
Open F3
Open With
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Mark as Landmark Ctrl+Alt+Shift+Up
Move...
Rename F2
Import...
Export...
Refresh F5
Validate
Open Javadoc Wizard...
Run As
Debug As
Profile As
Team
Compare With
Replace With
Source
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 Ant Build Alt+Shift+X, Q
3 Ant Build...
Open Run Dialog...
Open External Tools Dialog...
build.xml 1273 03.04.08 19:20 oclparserprog
README.txt 1262 17.02.08 10:32 oclparserprog
tudresden.ocl20.pivot.ocl2parser.test [trunk/ocl20/eclipse

Figure 5: Executing the OCL2 parser build script.



Properties for tudresden.ocl20.pivot.ocl2parser build.xml

Edit launch configuration properties

Run an Ant build file.

Name: tudresden.ocl20.pivot.ocl2parser build.xml

Runtime JRE:

[✓ Run in the same JRE as the workspace]

Separate JRE: jre1.5.0_15

Execution Environment:

VM arguments:

Working directory:

Default: F:\OCLSHK\Test\tudresden.ocl20.pivot.ocl2parser

Other:

Figure 6: Settings of the JRE for the Ant build script.

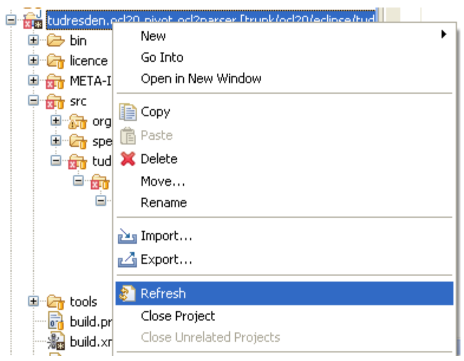


Figure 7: Refreshing the project “tudresden.ocl20.pivot.oclparser“.

Additionally you need to recompile all depending projects. Select the function "Project > Clean... > Clean all projects" in the *Eclipse* menu to clean all projects. All the projects should not contain any errors anymore and should be executable.

3 Using the Dresden OCL2 Toolkit for Eclipse

If you installed the *OCL2 Toolkit for Eclipse* using jar archives, you can execute the Toolkit by starting your *Eclipse SDK*. If you imported the Toolkit as plugin projects into an *Eclipse* workspace, you have to start a new *Eclipse SDK* instance. You can start a new instance via the menu "Run > Run As > Eclipse Application". If the menu "Eclipse Application" is not available or disabled you need to select one of the plugins of the toolkit first. Eventually you also have to install the *Eclipse Plugin Development Environment* first. After starting the new *Eclipse* instance you can use the *Dresden OCL2 Toolkit for Eclipse* as described below.

3.1 Loading a domain specific model

After starting the *Eclipse* instance you have to load a model into the toolkit. You can load a model by using the *import wizard* (File > Import...). Select the wizard "Dresden OCL2 Toolkit > Domain-Specific Model". In a new opened window you have to select a model file and a meta model for the model (see figure 8). During this tutorial the *PML* Model is used which you can find in the project "tudresden.ocl20.pivot.examples.pml" in the file "model/pml.ecore". Other examples delivered with the *Dresden OCL2 Toolkit for Eclipse* are listed at the end of this tutorial.

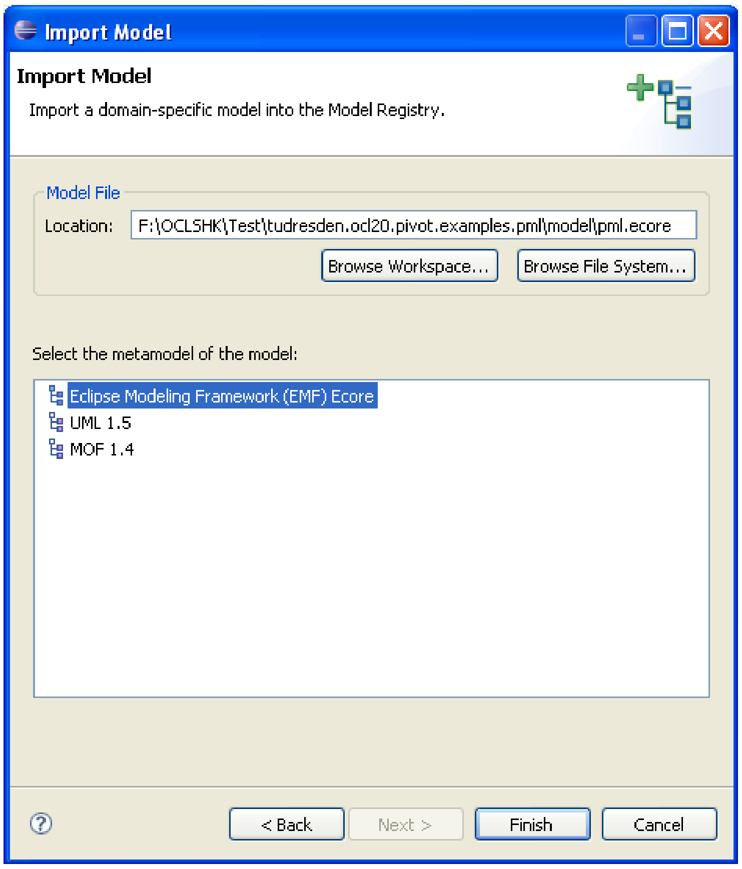
If the plugins of the *Dresden OCL2 Toolkit for Eclipse* were installed as jar archives, the *PML* plugin has to be imported into the *Workspace* first. Create a new projekt into your *Workspace* and select the *import wizard* „General -> Archive File"; in the following windo select the „plugins" directory into your *Eclipse* root folder, select the archive „tudresden.ocl20.pivot.examples.living_1.0.0.jar" and click the „Finish" button.

Figure 9 shows the loaded *PML* model, which uses *Ecore* as its meta model. Via the menu button (the little triangle in the right top corner) you can switch between different models in the model browser (see figure 10).

3.2 Loading a model instance

After loading the model, you can load a *model instance* using another *import wizard*. Use the wizard "Dresden OCL2 Toolkit > Model Instance". You have to select a model instance (in this tutorial we used the file "model instance/Testmodell.pml" of the project "tudresden.ocl20.pivot.examples.pml") and the domain specific model loaded before (see figure 11).

Figure 12 shows the loaded model instance of the *PML* model. Like in the model browser you can switch between different model instances (after loading a model instance you could have to select the model instance first, before it appears in the model instance browser). Note that the model instance browser only shows the model instances of the model actually selected in the model



Import Model

Import Model

Import a domain-specific model into the Model Registry.

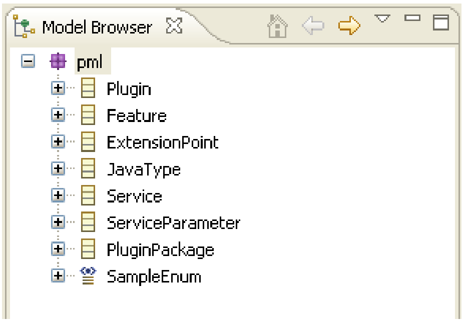
Model File

Location: F:\OCLSHK\Test\tudresden.ocl20.pivot.examples.pml\model\pml.ecore

Select the metamodel of the model:

- Eclipse Modeling Framework (EMF) Ecore
- UML 1.5
- MOF 1.4

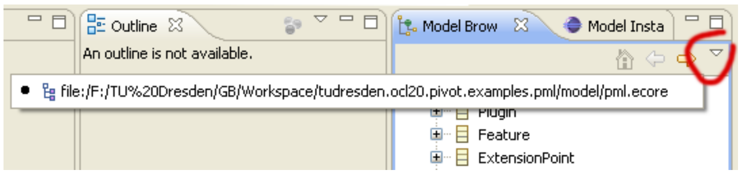
Figure 8: Loading a domain specific model.



Model Browser

- pml
  - Plugin
  - Feature
  - ExtensionPoint
  - JavaType
  - Service
  - ServiceParameter
  - PluginPackage
  - SampleEnum

Figure 9: The loaded PML model in the model browser.

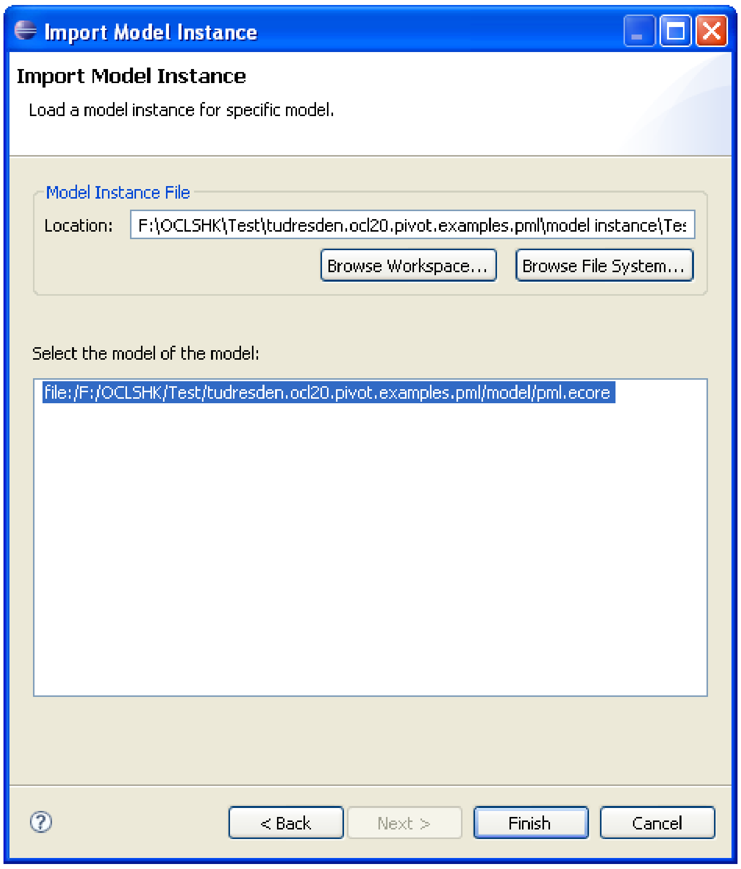


An outline is not available.

● file:/F:/TU%20Dresden/GB/Workspace/tudresden.ocl20.pivot.examples.pml/model/pml.ecore

- Plugin
- Feature
- ExtensionPoint

Figure 10: You can switch between different models using the little triangle.


Import Model Instance

Import Model Instance

Load a model instance for specific model.

Model Instance File

Location: F:\OCLSHK\Test\tudresden.ocl20.pivot.examples.pml\model instance\Tes

Select the model of the model:

file:/F:/OCLSHK/Test/tudresden.ocl20.pivot.examples.pml/model/pml.ecore

Figure 11: Loading a *PML* model instance.

browser. By switching the domain specific model, you also switch the pool of model instances available in the model instance browser.

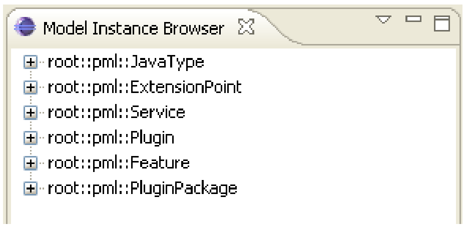
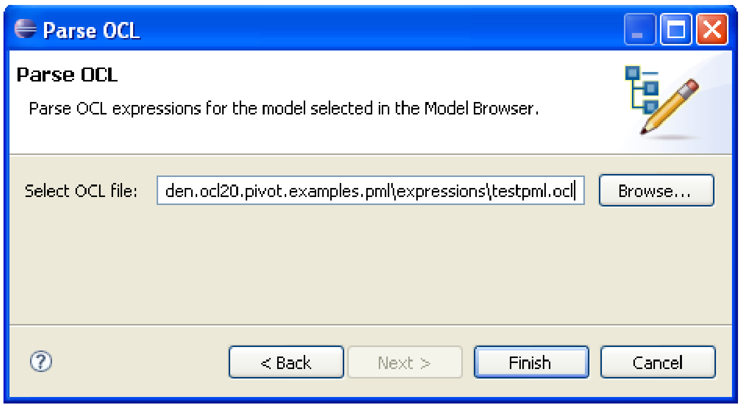


Figure 12: A loaded *PML* model instance in the Model Instance Browser.

3.3 Loading OCL expressions

Before you can interprete OCL constraints you have to load them like the domain specific model and the model instance before. Use the import wizard "Dresden OCL2 Toolkit > OCL Expressions" and select an OCL file (in this tutorial we used the OCL file "espressions/testpml.ocl" of the project "tudresden.ocl20.pivot.examples.pml", see figure 13).



Parse OCL

Parse OCL expressions for the model selected in the Model Browser.

Select OCL file: den.ocl20.pivot.examples.pml\expressions\testpml.ocl

Figure 13: The import of OCL expressions of a model loaded before.

The expressions of the selected OCL file will be loaded into the actually selected model instance. Figure 14 shows the "Model Browser" containing the *PML* model and the parsed expressions.

3.4 Interpretation of constraints

The interpretation of constraints is possible via the *Interpreter View*. Eventually you have to open the *View* via the menu option „Window -> Show View -> Other..." and select „TU Dresden OCL2 Toolkit -> Interpreter".

The *Interpeter View* provides a context menu containing all needed functionality. The context menu is shown in figure 15 (You can open the context menu via the little triangle in the right top corner of the *Interpreter View*).

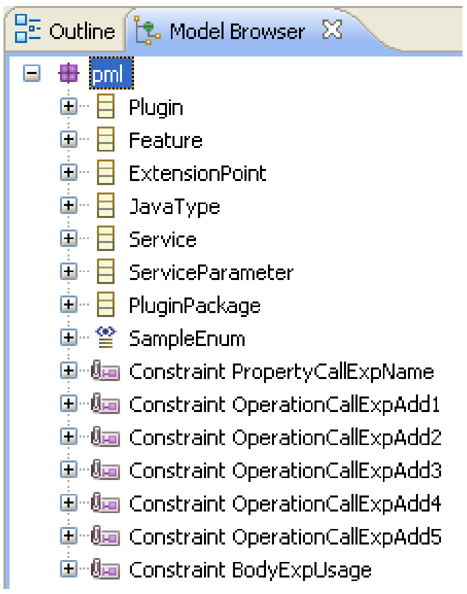

Outline
Model Browser
pml
Plugin
Feature
ExtensionPoint
JavaType
Service
ServiceParameter
PluginPackage
SampleEnum
Constraint PropertyCallExpName
Constraint OperationCallExpAdd1
Constraint OperationCallExpAdd2
Constraint OperationCallExpAdd3
Constraint OperationCallExpAdd4
Constraint OperationCallExpAdd5
Constraint BodyExpUsage

Figure 14: Parsed expressions of the *PML* model in the Model Browser.

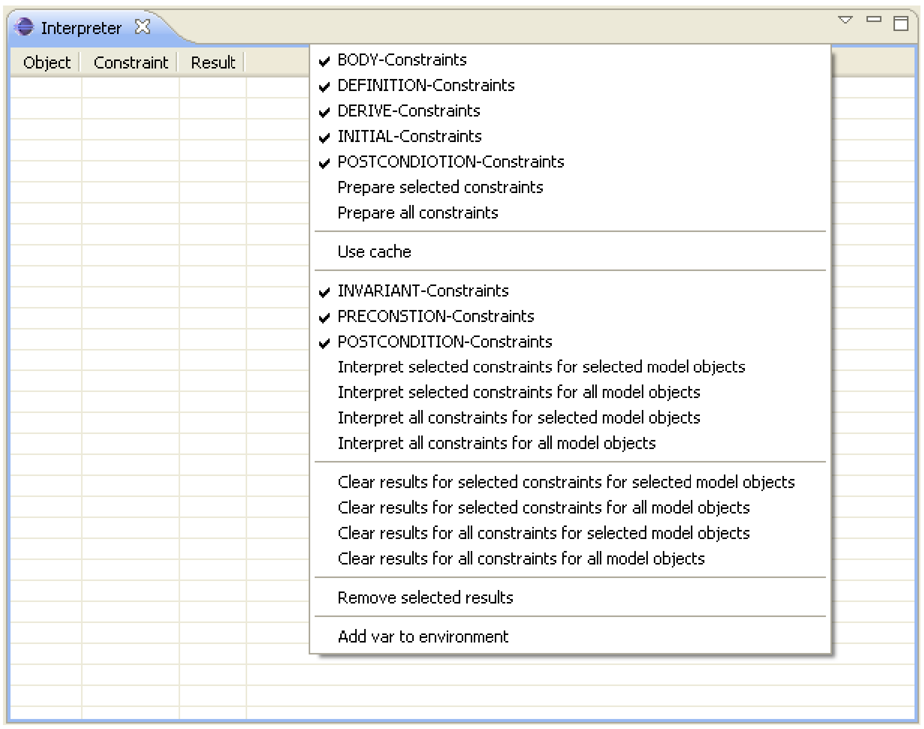
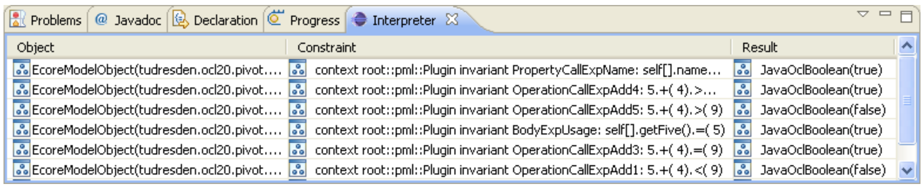

Interpreter
Object
Constraint
Result
✓ BODY-Constraints
✓ DEFINITION-Constraints
✓ DERIVE-Constraints
✓ INITIAL-Constraints
✓ POSTCONDIOTION-Constraints
Prepare selected constraints
Prepare all constraints
Use cache
✓ INVARIANT-Constraints
✓ PRECONSTION-Constraints
✓ POSTCONDITION-Constraints
Interpret selected constraints for selected model objects
Interpret selected constraints for all model objects
Interpret all constraints for selected model objects
Interpret all constraints for all model objects
Clear results for selected constraints for selected model objects
Clear results for selected constraints for all model objects
Clear results for all constraints for selected model objects
Clear results for all constraints for all model objects
Remove selected results
Add var to environment

Figure 15: The context menu of the *Interpreter View*.

For the interpretation of constraints you can use the functions “Interpret ...“ in the context menu. First you can select constraints you want to interpret in the *Model Browser* and the model objects for which the constraints should be interpreted in the *Model InstanceBrowser*. The M*odel Instance Browser* shows only the model objects, for which you can interpret the constraints selected before.

In the *interpreter menu* you can select the types you want to interpret (invariants, pre and post conditions). After interpretation the table in the *Interpreter View* shows all results. These are filtered by the selection of constraints and model objects. Figure 16 shows a table with some results. If the column “Result“ does not contain enough space for a result, you can open a result via a double mouse click in a new window.



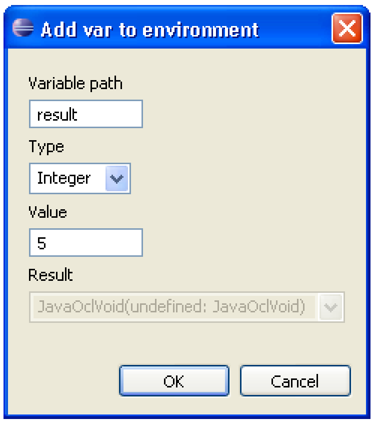
Object	Constraint	Result
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant PropertyCallExpName: self[].name...	JavaOclBoolean(true)
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd4: 5.+(4).>...	JavaOclBoolean(true)
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd5: 5.+(4).>(9)	JavaOclBoolean(false)
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant BodyExpUsage: self[].getFive().=(5)	JavaOclBoolean(true)
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd3: 5.+(4).=(9)	JavaOclBoolean(true)
EcoreModelObject(tudresden.ocl20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd1: 5.+(4).<(9)	JavaOclBoolean(false)

Figure 16: Some results in the *Interpreter View*.

3.5 Additional functionality of the interpreter

Before interpretation you can prepare some or all constraints like described in [Bra07, Abschnitt Environment] (“Prepare ...“).

You can add variables to the environment via “Add var to environment“. This functionality can be used for parameters and results of operations or the name of parameters. Figure 17 shows a window to add variables to the environment. After entering a variable name (e.g. “result“ or the name of a parameter) a primitive type object (e.g. Integer) can be created or the result of an already interpreted constraint can be used.



Variable path: result

Type: Integer

Value: 5

Result: JavaOclVoid(undefined: JavaOclVoid)

Figure 17: Window to add a variable to the environment.

PML example	
Plugin	tudresden.ocl20.pivot.examples.pml
Meta model	Ecore
Model	model/pml.ecore
OCL expressions	expressions/testpml.ocl
Model instance	model instance/Testmodell.pml
Living example	
Plugin	tudresden.ocl20.pivot.examples.living
Meta modell	UML 1.5 or UML 2.0
Model	model/UmlExample.xmi
OCL expressions	expressions/living.ocl
Model instance	src/tudresden.ocl20.pivot.examples.living/ModelProviderClass.java

Table 1: Examples provided with the *Dresden OCL2 Toolkit for Eclipse*.

Using the "Clear ..." functionality, results can be removed from the interpreter view. You can remove all results or select results which you want to remove before ("Remove Selected Results").

Using the check box "Use cache", you can activate the caching mechanism described in [Bra07] to improve the performance of the interpreter.

4 Conclusion

This tutorial described how to use the *Dresden OCL2 Toolkit for Eclipse*. It explained how to install or import and start the Toolkit's plugins. Afterwards the usage of the interpreter of the *Dresden OCL2 Toolkit for Eclipse* was shown.

This tutorial explained the *Dresden OCL2 Toolkit for Eclipse* using the PML example. Table 1 shows all examples proviced whith the *Dresden OCL2 Toolkit for Eclipse*.

As mentioned before, more information about the *Dresden OCL2 Toolkit* is available at the website of *Dresden OCL2 Toolkit* [Sofb].

Literatur

[Brä07] Bräuer, Matthias: *Models and Metamodels in a QVT/OCL Development Environment*, Technische Universität Dresden, Großer Beleg, 2007. `http://dresden-ocl.sourceforge.net/gbbraeuer/index.html`

[Bra07] Brandt, Ronny: *Ein OCL-Interpreter für das Dresden OCL2 Toolkit basierend auf dem Pivotmodell*, Technische Universität Dresden, Diplomarbeit, 2007

[Sofa] Software Technology Group (Hrsg.): *Sourceforge project website of the Dresden OCL2 Toolkit.* `http://sourceforge.net/projects/dresden-ocl/`

[Sofb] Software Technology Group (Hrsg.): *Website of the Dresden OCL2 Toolkit.* `http://dresden-ocl.sourceforge.net/`

[Thea] The Eclipse Foundation. (Hrsg.): *Eclipse Website.* `http://www.eclipse.org/`

[Theb] The Eclipse Foundation. (Hrsg.): *Website of the EMF project.* `http://www.eclipse.org/modeling/emf/`

[Tiga] Tigris.org (Hrsg.): *Installation of the Subclipse plugin.* `http://subclipse.tigris.org/install.html`

[Tigb] Tigris.org (Hrsg.): *Subclipse Website.* `http://subclipse.tigris.org/`