

# Eine Einführung in das Dresden OCL2 Toolkit for Eclipse

von Ronny Brandt und Claas Wilke

Dieses Dokument beschreibt den Einstieg in die Arbeit mit dem *Dresden OCL2 Toolkit for Eclipse*. Diese Version des Toolkits basiert auf der neuen Infrastruktur des Pivotmodells, das im Großen Beleg von Matthias Bräuer [Brä07] näher beschrieben ist. Weitere Informationen zum Toolkit befinden sich auch auf der Webseite des *Dresden OCL2 Toolkit* [Sofb].

Das Tutorial gliedert sich wie folgt: Zunächst wird das Installieren der nötigen *Eclipse*-Plugins für das Toolkit beschrieben. Anschließend wird das Laden eines domänen-spezifischen Modells und einer Modell-Instanz beschrieben. Dann wird das Laden von OCL Expressions erläutert. Zum Abschluss wird noch auf das Interpretieren von OCL Exressions eingegangen.

Die im Folgenden beschriebene Vorgehensweise wurde mit *Eclipse 3.3.2* getestet [Thea]. Das Tutorial sollte aber auch mit *Eclipse 3.4* ausführbar sein. Sofern sich Namen von Menüs und Funktionen in *Eclipse 3.4* geändert haben, ist dies im Tutorial angemerkt. Neben dem *Eclipse SDK* müssen auch die erforderlichen Plugins des *Eclipse Modeling Framework (EMF)* installiert werden. Für dieses Tutorial wurden die *EMF*-Plugins in Version 2.3.2 verwendet.

Um die *EMF*-Plugins zu installieren, müssen diese von der *EMF*-Homepage [Theb] heruntergeladen und in das Plugin-Verzeichniss von *Eclipse* entpackt werden. Anschließend kann das *Eclipse SDK* gestartet werden. Alternativ können die Plugins auch nach dem Start des *Eclipse SDK* über den *Eclipse Update Manager* installiert werden.

## 1 Das Dresden OCL2 Toolkit for Eclipse installieren

Um die Werkzeuge des *Dresden OCL2 Toolkit for Eclipse* nutzen zu können, müssen diese entweder als *Eclipse*-Plugins installiert werden, oder als Plugin-Projekte in den *Eclipse-Workspace* importiert werden. Beide Vorgehensweisen werden im folgenden kurz erläutert.

### 1.1 Installieren der Eclipse-Plugins

Um das Toolkit als *Eclipse*-Plugins zu installieren, müssen die Plugins als jar-Archive vorliegen. Diese sind z. B. unter [Sofa] verfügbar. Die jar-Archive müssen einfach in den „plugins“-Unterordner des *Eclipse SDK* kopiert werden. Anschließend kann *Eclipse* gestartet und mit dem *Dresden OCL2 Toolkit for Eclipse* gearbeitet werden.

### 1.2 Import als Projekte in den Workspace

Alternativ können die Plugins auch als Projekte in den aktuellen Workspace des Eclipse SDK importiert werden. Dieser Import kann auf zwei verschiedene Arten erfolgen. Zum einen können die Plugins direkt importiert werden, wenn

diese offline vorliegen, z. B. durch eine Quellcode-Distribution, die unter [Sofa] erhältlich ist.

Die zweite Möglichkeit ist der Import der Plugins aus dem SVN-Verzeichnis (*Subversion*) des *Dresden OCL2 Toolkits*. Diese beiden Möglichkeiten werden nun beschrieben.

### 1.2.1 Die Plugin-Projekte aus lokalem Verzeichnis importieren

Es wird davon ausgegangen, dass die Plugins sich in einem lokalen Verzeichnis XYZ befinden. Für den Import in *Eclipse* öffnet man den *Import-Wizard* über das Menü „File > Import...“, und wählt „Existing Projects into Workspace“ aus der Gruppe „General“ (Siehe Abbildung 1). Im folgenden Fenster wählt man das Verzeichnis XYZ als Wurzel-Verzeichnis. Anschließend werden die Plugins markiert (falls dies nicht bereits automatisch erfolgt ist) und die Checkbox „Copy projects into workspace“ aktiviert (Siehe Abbildung 2). Nach der Bestätigung mittels des Buttons „Finish“ werden die Projekte in den Workspace importiert.

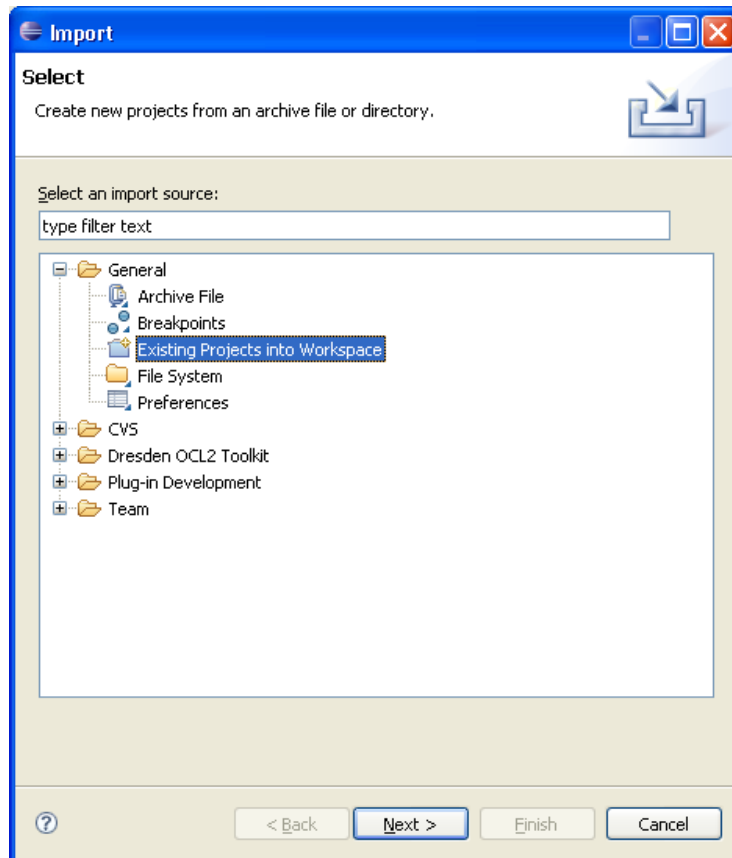


Abbildung 1: Import vorhandener Plugins (1).

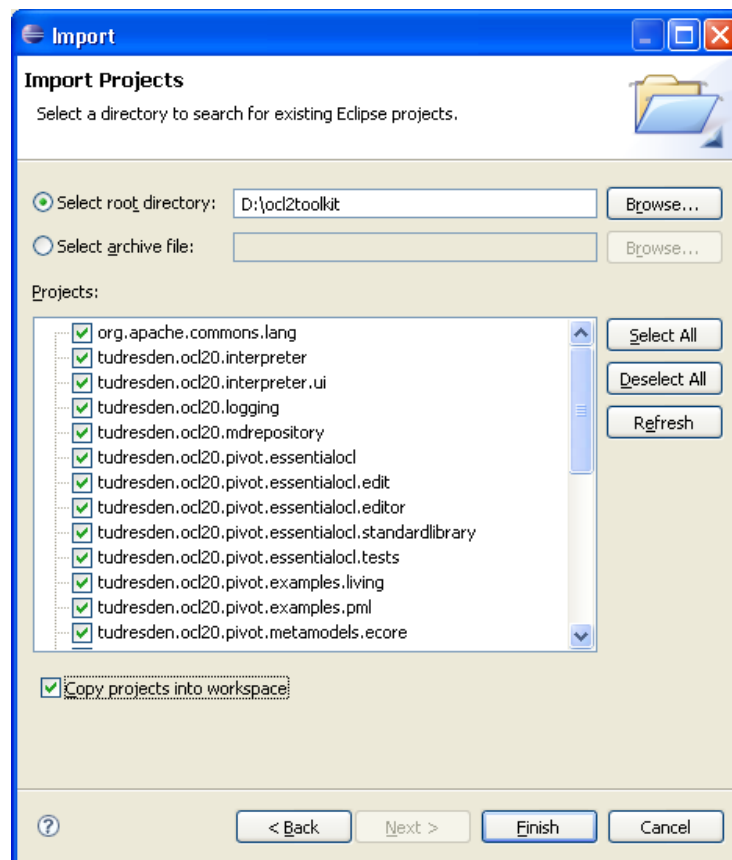


Abbildung 2: Import vorhandener Plugins (2).

### 1.2.2 Die Plugin-Projekte aus dem SVN herunterladen

Um die Plugins direkt aus dem *SVN*-Repository zu importieren, muss zusätzlich ein *Eclipse-Plugin* für den Zugriff auf *SVN* installiert werden. Das nötige Tool heißt *Subclipse* und ist unter [Tiga] verfügbar. Die Installation von *Subclipse* wird auf der Webseite des *Supclipse*-Plugins [Tigb] erläutert.

**Achtung: Derzeit funktioniert das *Subclipse*-Plugin nicht unter *Eclipse 3.4*.** Unter *Eclipse 3.4* sollte ein ähnliches *Eclipse*-Plugin oder ein anderes Tool für den Zugriff auf *SVN-Repositories* außerhalb von *Eclipse* verwendet werden.

Nach der Installation von *Supclipse* existiert eine neue *Perspective* für den Zugriff auf *SVN* im *Eclipse SDK*. Diese kann über das Menü „Window > Open Perspective > Other...“ durch Auswahl von „SVN Repository Exploring“ geöffnet werden. Im View *SVN Repository* fügt man ein neues Repository hinzu (Siehe Abbildung 3) und gibt „https://dresden-ocl.svn.sourceforge.net/svnroot/dresden-ocl/“ als URL ein. Nach der Auswahl von „Finish“ erscheint das *SVN*-Wurzelverzeichnis im *Repository-View*.

Um die Plugins nun zu importieren, markiert man diese im Repository-Verzeichnis „trunk/ocl20forEclipse/eclipse“ und wählt „Checkout...“ im Kontextmenü (Siehe Abbildung 4). Die vorgegebenen Einstellungen können übernommen und die Plugins mit einem Klick auf „Finish“ importiert werden.

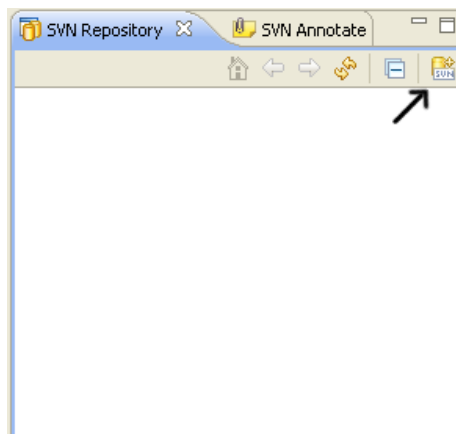


Abbildung 3: Hinzufügen eines *SVN*-Repositories.

## 2 Build des Parsers

Falls die Plugins des *Dresden OCL2 Toolkits for Eclipse* als *Plugin-Projekte* importiert wurden, muss nun noch der *OCL2-Parser* per Build-Skript erstellt werden. Bei einem Installieren der Plugins als jar-Archive ist dieser Schritt zu überspringen. Dazu muss im Projekt „tudresden.ocl20.pivot.ocl2parser“ die Datei „build.xml“ ausgewählt und mit Rechtsklick aus dem Kontextmenü die Funktion „Run As ... > Ant Build“ ausgewählt werden (Siehe Abbildung 5).

Falls die Fehlermeldung „Problem: failed to create task or type eclipse.refresh-Local“ auftritt, muss die Konfiguration des *Ant*-Skripts angepasst werden. Dazu

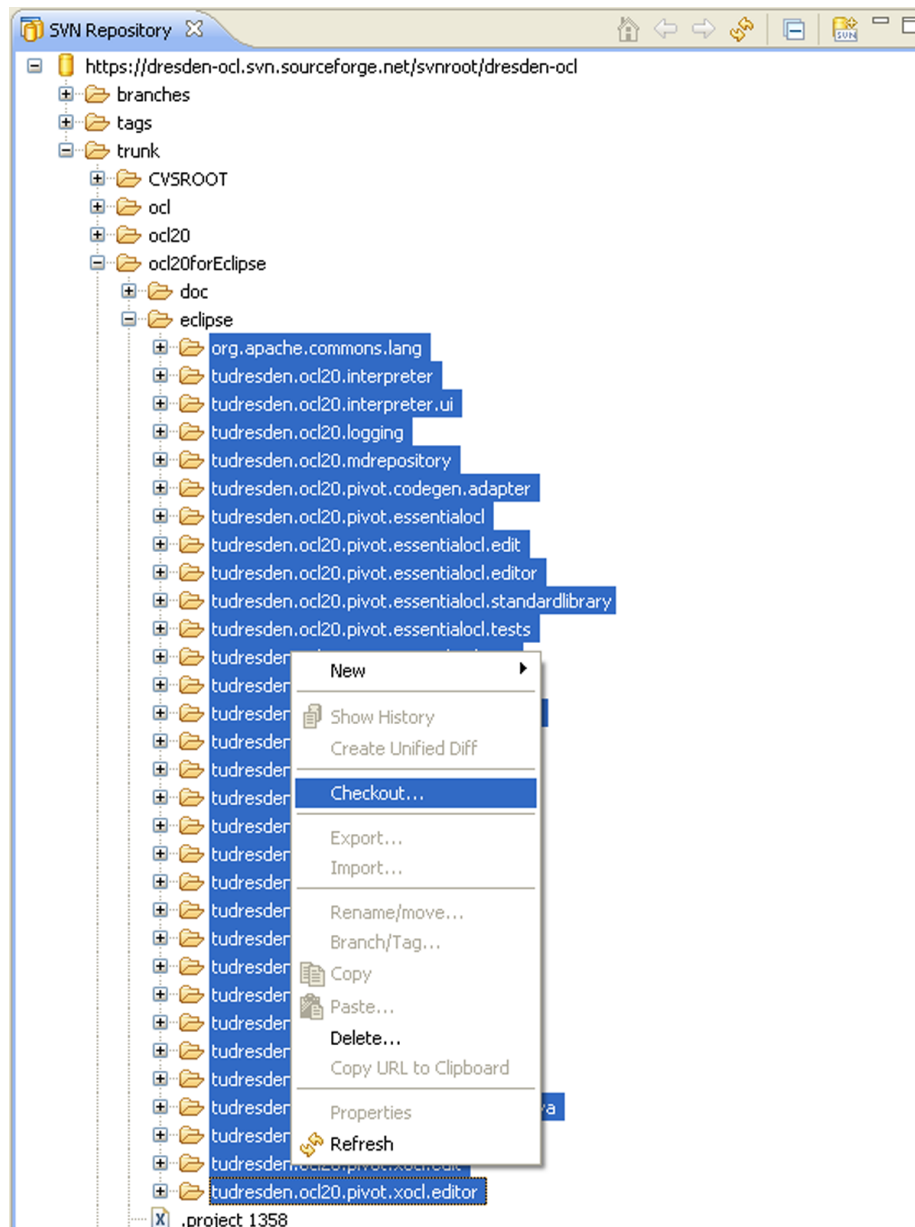


Abbildung 4: Auschecken der erforderlichen Dresden OCL2 Toolkit Plugins.

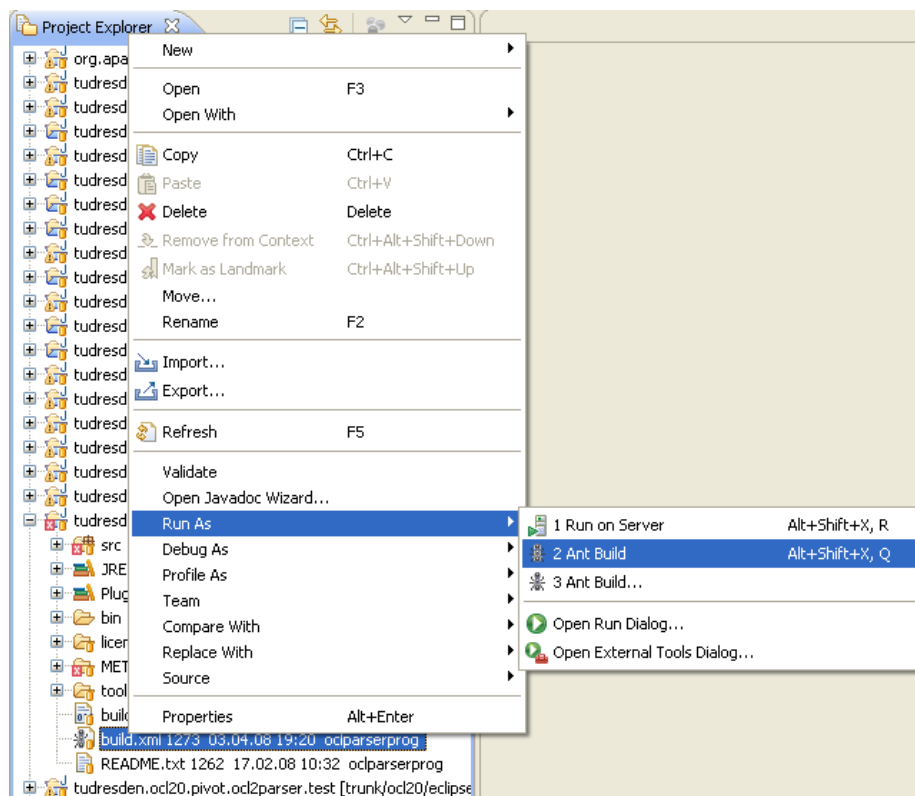


Abbildung 5: Ausführen des Build-Skripts für den OCL2-Parser.

per Rechtsklick auf der „build.xml“, die Funktion „Properties“ auswählen. Dann im neu geöffneten Fenster links die Unterseite „Run/Debug settings“ auswählen. Dort die Konfiguration „tudresden.ocl20.pivot.oclparser build.xml“ auswählen und auf die Schaltfläche „Edit“ klicken. Es öffnet sich ein weiteres Fenster, in diesem im Reiter „JRE“ die Einstellung „Run in the same JRE as the workspace“ auswählen und diese Einstellung mit der Schaltfläche „OK“ bestätigen (Siehe Abbildung 6). Anschließend sollte das *Ant*-Skript ohne Fehlermeldung ausführbar sein.

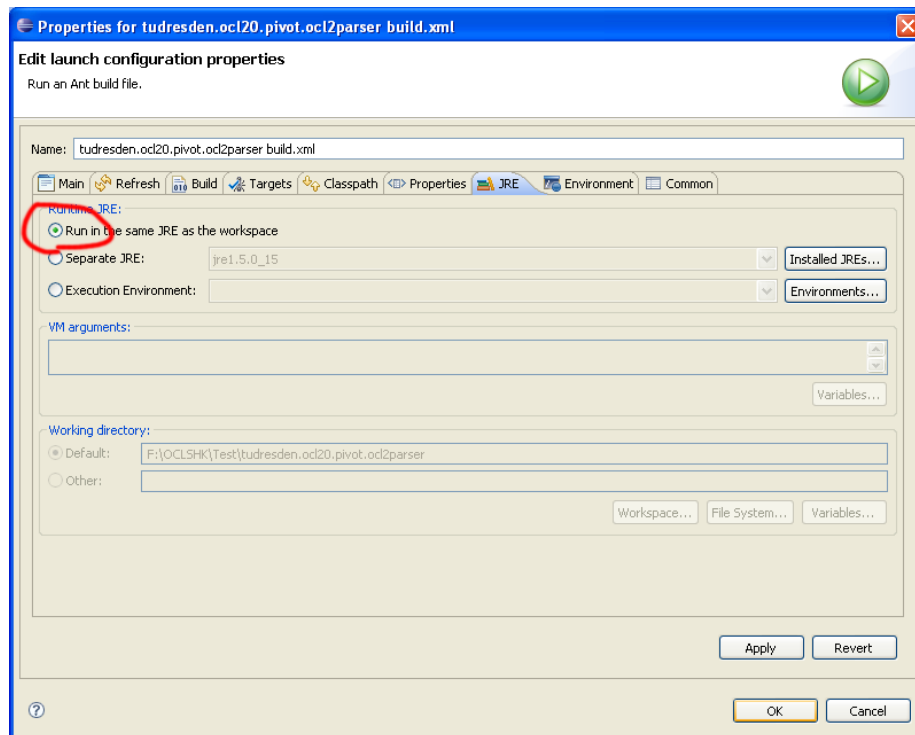


Abbildung 6: Einstellen der JRE des Build-Skripts.

Nach dem das Build-Skript erfolgreich ausgeführt wurde, müssen die Projekte im Workspace noch aktualisiert werden. Zunächst muss das Projekt „tudresden.ocl20.pivot.oclparser“ per Rechtsklick aktualisiert werden („Refresh“, siehe Abbildung 7).

Anschließend müssen die Projekte noch neu kompiliert werden. Dazu ist im *Eclipse*-Menü die Funktion „Project > Clean... > Clean all projects“ auszuführen. Danach sollten die Projekte keine Fehler mehr enthalten und ausführbar sein.

### 3 OCL2 Toolkit for Eclipse nutzen

Bei einer Installation des *OCL2 Toolkits for Eclipse* als jar-Archive reicht zum Nutzen des Toolkits das Starten des *Eclipse SDK* aus. Beim Arbeiten mit importieren Plugin-Projekten im *Eclipse Workspace*, muss zunächst eine neue Instanz

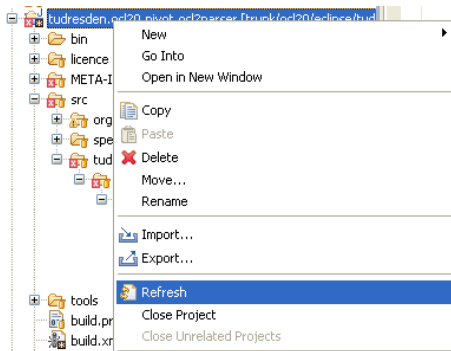


Abbildung 7: Aktualisieren des Projekts „tudresden.ocl20.pivot.oclparser“.

der *Eclipse SDK* gestartet werden. Diese wird über das Menü „Run > Run As > Eclipse Application“ ausgeführt. Sollte der Menüpunkt „Eclipse Application“ nicht vorhanden sein, so muss zunächst eines der Plugins markiert werden (Eventuell muss zum Ausführen auch noch das *Eclipse Plugin Development Environment* installiert werden). Wenn die neue Instanz gestartet wurde können die Plugins - wie in den folgenden Unterabschnitten beschrieben - verwendet werden.

### 3.1 Ein Modell laden

Nach dem Starten der *Eclipse*-Umgebung mit den Plugins des *Dresden OCL2 Toolkit for Eclipse* muss zunächst ein Modell geladen werden. Dies geschieht mit Hilfe eines *Import-Wizards* (File > Import...). In diesem Wizard wird nach der Auswahl des Punktes „Dresden OCL2 Toolkit > Domain-Specific Model“ die Modell-Datei und das zugehörige Meta-Modell gewählt (Siehe Abbildung 8). Das im Tutorial verwendete *PML*-Modell befindet sich im Projekt „tudresden.ocl20.pivot.examples.pml“ unter dem Pfad „model/pml.ecore“. Andere Beispiele die im *Dresden OCL2 Toolkit for Eclipse* werden am Ende dieses Tutorials aufgeführt.

Abbildung 9 zeigt das geladene *PML*-Modell, welches *Ecore* als Meta-Modell nutzt. Über den Menü-Button (das kleine Dreieck in der rechten oberen Ecke) kann zwischen verschiedenen geladenen Modellen gewechselt werden (Siehe Abbildung 10).

### 3.2 Eine Modell-Instanz laden

Nach dem Laden des Modells kann, ebenfalls über einen *Import-Wizard*, eine *Modell-Instanz* geladen werden. Hierfür ist der Wizard „Dresden OCL2 Toolkit > Model instance“ zu wählen. Es ist eine Modell-Instanz auszuwählen (im Tutorial handelt es sich um die Datei „model instance/Testmodell.pml“ des Projekts „tudresden.ocl20.pivot.examples.pml“) und zusätzlich das zuvor geladene *domänen-spezifische Modell* für die Modell-Instanz auszuwählen (Siehe Abbildung 11).

Falls die Plugins des *Dresden OCL2 Toolkit for Eclipse* als jar-Archive installiert wurden, muss dass entsprechende *PML*-Plugin zunächst noch in den



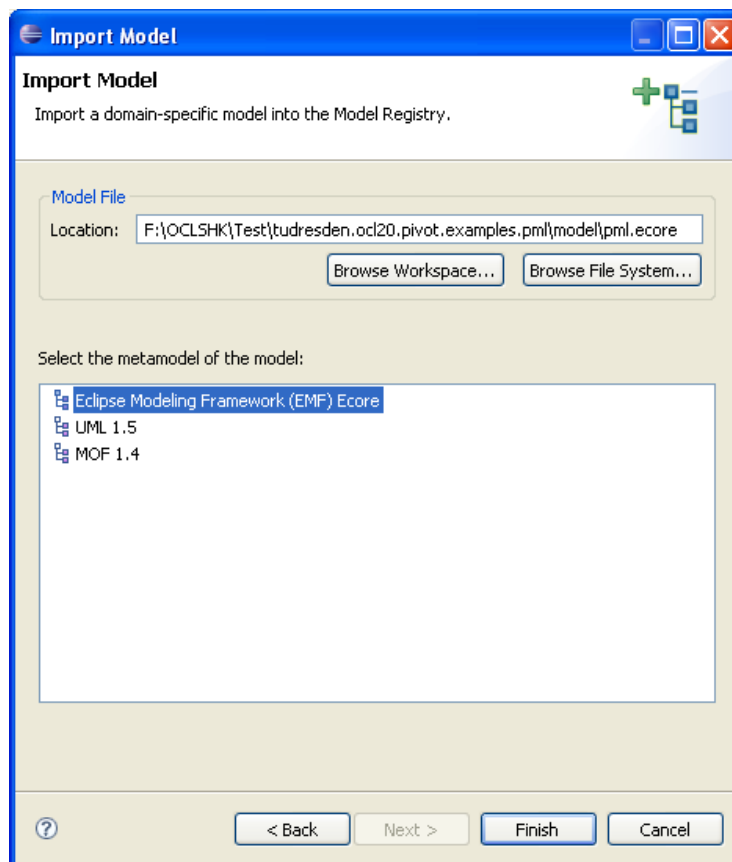


Abbildung 8: Laden eines domänen-spezifischen Modells.

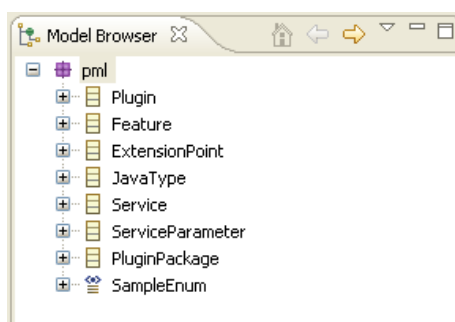


Abbildung 9: Geladenes *PML*-Model im Model Browser.

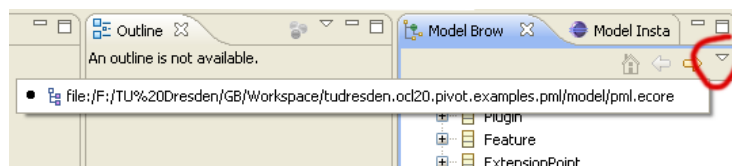


Abbildung 10: Wechsel zwischen den geladenenen Modell-Instanzen.

*Workspace* importiert werden. Legen sie dazu ein neues Projekt in Ihrem *Workspace* an und wählen sie dann den *Import-Wizard* „General -> Archive File“; im folgenden Fenster wählen sie aus dem „plugins“-Verzeichnis ihrer *Eclipse*-Instanz das Archiv „tudresden.ocl20.pivot.examples.living\_1.0.0.jar“ aus und klicken den „Finish“-Button.

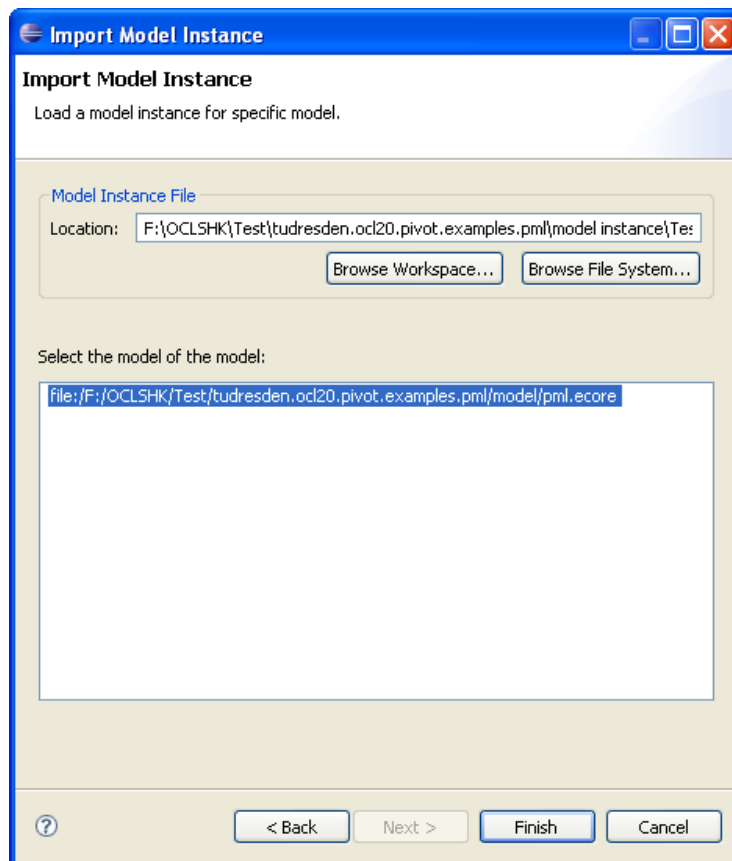


Abbildung 11: Laden einer *PML*-Modell-Instanz.

Abbildung 12 zeigt die geladene Modell-Instanz für das *PML*-Modell. Auch in dieser Ansicht kann (analog zur Modell-Ansicht) über den Menü-Button zwischen den verfügbaren Modell-Instanzen gewechselt werden (in einigen Fällen muss die Modell-Instanz nach dem Laden erst über das Dreieck ausgewählt werden, bevor sie im Browser angezeigt wird). In dem Auswahlmenü werden allerdings nur diejenigen Modell-Instanzen gezeigt, die zum aktiven Modell gehören. Mit dem Wechsel des Modells werden auch die verfügbaren Modell-Instanzen gewechselt.

### 3.3 OCL-Expressions laden

Vor der Interpretation von OCL-Constraints müssen diese noch über einen *Import-Wizard* importiert werden. Hierfür ist der *Wizard* „Dresden OCL2 Toolkit > OCL Expressions“ zu wählen. Es ist eine OCL-Datei auszuwählen (im

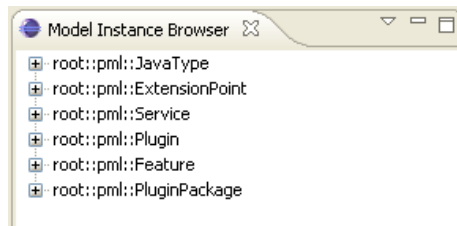


Abbildung 12: Geladene *PML*-Modell-Instanz im Model Instance Browser.

Tutorial handelt es sich um die Datei „espressions/testpml.ocl“ des Projekts „tudresden.ocl20.pivot.examples.pml“, siehe Abbildung 13).

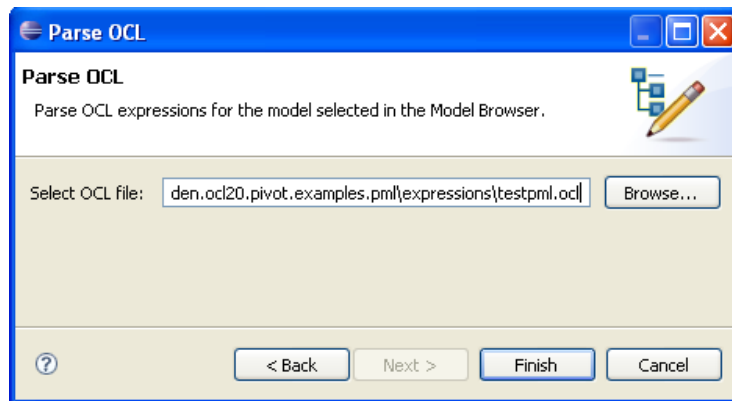


Abbildung 13: Der Import von OCL-Expressions für ein zuvor geladenes Modell.

Die Expressions der gewählten OCL-Datei werden in das aktive Modell geladen. Abbildung 14 zeigt den „Model Browser“ mit dem *PML*-Modell und den geladenen Expressions.

### 3.4 Interpretation von Constraints

Die Interpretation von Constraints erfolgt in der *Interpreter-View*. Eventuell muss diese zunächst geöffnet werden. Dies geschieht über die Menü-Funktion „Window -> Show View -> Other...“. Im sich öffnenden Fenster ist dann die View „TU Dresden OCL2 Toolkit -> Interpreter“ auszuwählen.

Die *Interpreter View* bietet ein Menü mit den notwendigen Optionen in Form eines Kontext-Menüs. Das Kontext-Menü zeigt Abbildung 15 (Es wird durch einen Klick auf das kleine Dreieck in der rechten oberen Ecke der *Interpreter-View* geöffnet).

Zur Interpretation der Constraints stehen die Menüpunkte „Interpret ...“ zur Verfügung. Zunächst können im *Model Browser* die gewünschten Constraints und im *Model Instance Browser* die gewünschten Modell-Objekte zu Interpretation gewählt werden. Der *Model Instance Browser* zeigt nur die Modell-Objekte, die zu den gewählten Constraints passen. Im Menü des Interpreters können die zu interpretierenden Typen (Invarianten, Pre- und Postconditions) gewählt werden. Nach der Interpretation zeigt die *Interpreter View* die Ergebnisse. Diese

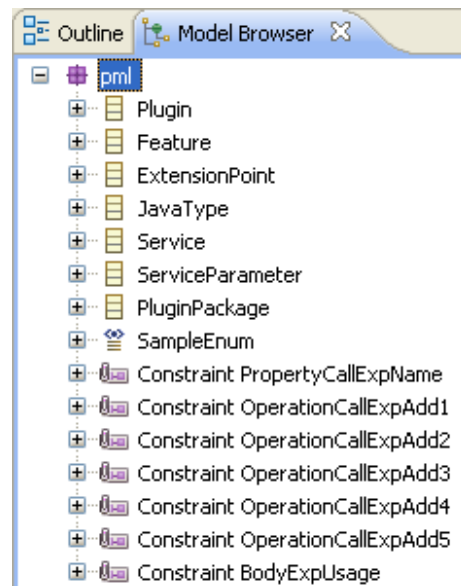


Abbildung 14: Geladene Expressions für das *PML*-Model im *Model Browser*.

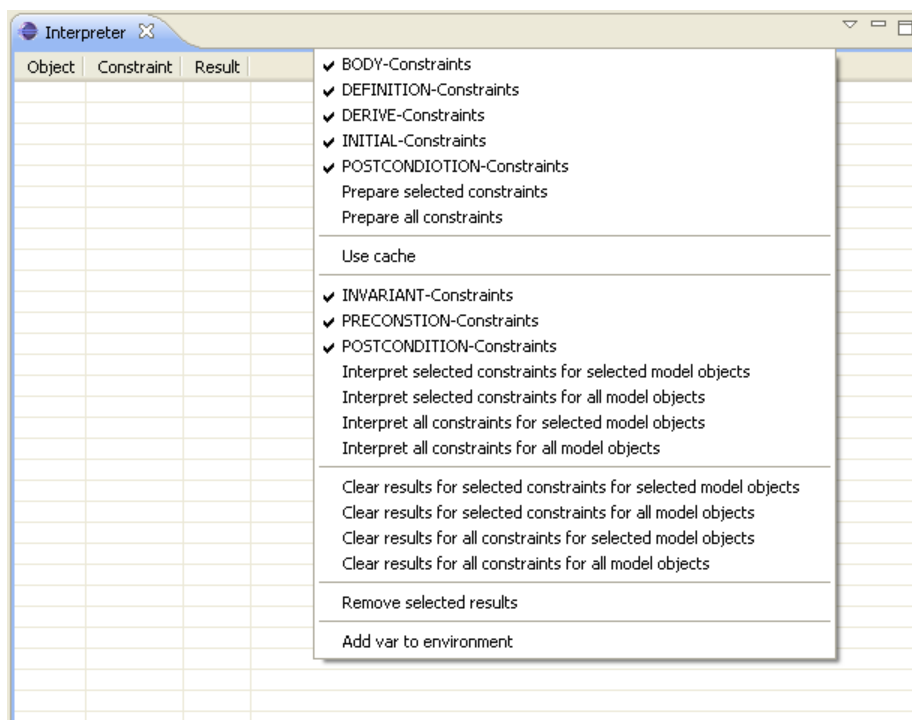
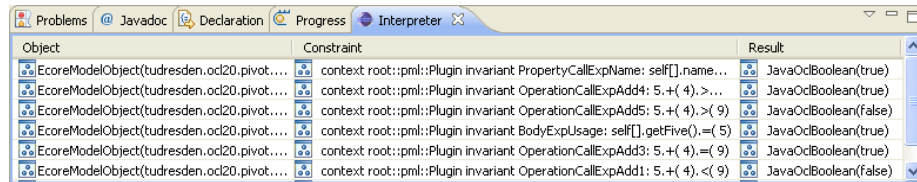


Abbildung 15: Das Kontext-Menü des Interpreters.

werden entsprechend der Auswahl von Constraints und Modell-Objekten gefiltert. Abbildung 16 zeigt die Tabelle mit einer Auswahl von Ergebnissen. Wenn die Länge der Spalte „Result“ für das Ergebnis nicht ausreichend ist, zeigt ein Doppelklick auf das Objekt in der Zeile das Ergebnis in einem Popup-Fenster.



Object	Constraint	Result
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant PropertyCallExpName: self[].name...	JavaOclBoolean(true)
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd4: 5.+( 4).>...	JavaOclBoolean(true)
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd5: 5.+( 4).>( 9)	JavaOclBoolean(false)
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant BodyExpUsage: self[].getFive().=( 5)	JavaOclBoolean(true)
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd3: 5.+( 4).=( 9)	JavaOclBoolean(true)
EcoreModelObject(tudresden.oc20.pivot....	context root::pml::Plugin invariant OperationCallExpAdd1: 5.+( 4).<( 9)	JavaOclBoolean(false)

Abbildung 16: Ausgewählte Ergebnisse in der *Interpreter View*.

### 3.5 Weitere Interpretationsmöglichkeiten

Vor der Interpretation ist es möglich, den in [Bra07, Abschnitt Environment] beschriebenen Vorbereitungsschritt („Prepare ...“) für einzelne oder alle Constraints durchzuführen. Das Vorgehen entspricht dem der Interpretation (Auswahl der Constraints, Auswahl der Modell-Objekte, Auswahl der Typen, „Prepare ...“).

Auch der Menüpunkt „Add var to environment“ ist zur Vorbereitung der Interpretation von Vor- und Nachbedingungen nutzbar. Mit dessen Hilfe ist es möglich, Variablen zum Environment hinzuzufügen. Diese werden zum Beispiel für Parameter und Rückgabewerte (result) von Operationen verwendet. Abbildung 17 zeigt das Fenster zum Hinzufügen von Variablen. Nach der Angabe des Variablennamens (zum Beispiel „result“ oder der Name eines Parameters) kann entweder ein Objekt eines primitiven Typs (zum Beispiel Integer) erstellt oder das Ergebnis eines bereits interpretierten Constraints verwendet werden.

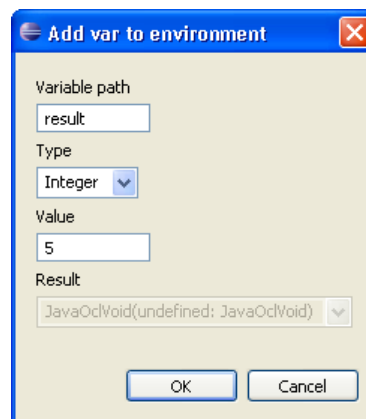


Abbildung 17: Dialog zum Hinzufügen von Variablen zum *Environment*.

Mit der „Clear ...“-Auswahl können die Ergebnisse basierend auf den gewählten Constraints und Modell-Objekten gelöscht werden. „Remove Selected Results“ entfernt die Ergebnisse, die in der Ergebnis-Tabelle selektiert sind.

<b>PML-Beispiel</b>	
Plugin	tudresden.ocl20.pivot.examples.pml
Meta-Modell	Ecore
Modell	model/pml.ecore
OCL-Expressions	expressions/testpml.ocl
Model-Instanz	model instance/Testmodell.pml
<b>Living-Beispiel</b>	
Plugin	tudresden.ocl20.pivot.examples.living
Meta-Modell	UML 1.5 oder UML 2.0
Modell	model/UmlExample.xmi
OCL-Expressions	expressions/living.ocl
Model-Instanz	src/tudresden.ocl20.pivot.examples.living/ModelProviderClass.java

Tabelle 1: Übersicht über Beispiele für das *Dresden OCL2 Toolkit for Eclipse*.

Mit der Checkbox „Use cache“ kann gewählt werden, ob der Interpreter den in [Bra07] beschriebenen Mechanismus zur Performance-Steigerung nutzt.

## 4 Zusammenfassung

Dieses Tutorial gab einen Einstieg in die Verwendung des *Dresden OCL2 Toolkit for Eclipse*. Es wurde erläutert, wie die Plugins des Toolkits installiert oder importiert und gestartet werden. Anschließend wurde auf die Arbeit mit dem Interpreter des *Dresden OCL2 Toolkit for Eclipse* eingegangen.

In diesem Tutorial wurde das Arbeiten mit dem *Dresden OCL2 Toolkit for Eclipse* anhand des PML-Modells erläutert. Tabelle 1 zeigt eine Übersicht über die derzeit mit dem Toolkit bereitgestellten Beispiele.

Bei weiteren Informationen zum *Dresden OCL2 Toolkit* sei hier nochmals auf die Webseite des *Dresden OCL2 Toolkits* verwiesen [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 Projekt-Webseite des Dresden OCL2 Toolkits*.  
<http://sourceforge.net/projects/dresden-ocl/>
- [Sofb] SOFTWARE TECHNOLOGY GROUP (Hrsg.): *Webseite des Dresden OCL2 Toolkits*. <http://dresden-ocl.sourceforge.net/>
- [Thea] THE ECLIPSE FOUNDATION. (Hrsg.): *Eclipse-Homepage*.  
<http://www.eclipse.org/>
- [Theb] THE ECLIPSE FOUNDATION. (Hrsg.): *Homepage des EMF-Projektes*.  
<http://www.eclipse.org/modeling/emf/>
- [Tiga] TIGRIS.ORG (Hrsg.): *Homepage von Subclipse*.  
<http://subclipse.tigris.org/>
- [Tigb] TIGRIS.ORG (Hrsg.): *Installation des Subclipse Plugins*.  
<http://subclipse.tigris.org/install.html>