FOOSE UND OCL IN DER PRAXIS

OCL im Sinn der Vorlesung

- DresdenOCL
  - *Integrierbar in Eclipse*
  - *Unterstützt OCL-Constraints für UML Klassendiagramme*
    - *Klasseninvarianten*
    - *Vor- / Nachbedingungen für Methoden*
    - *Auswertung der Constraints für Java-Instanz des Modells (Instanzen von modellkonformen Java-Klassen)*
  - *Automatische Generierung von AspectJ-Code*
    - *Wird in den Code der Java-Implementierung eingewebt*
    - *Auswertung der Constraints bei Methodenaufrufen und beim Zugriff auf Instanzvariablen*
    - *Auslösen von Exceptions wenn Constraints verletzt werden*

Dresden OCL continued...

- Fazit
  - Schöner Ansatz aber in der Praxis scheitert DresdenOCL leider an komplexeren Constraints
  - Wenig verbreitet
  - Es gibt aber (meines Wissens) keine andere OCL-Implementierung, die die Auswertung auf (Java-)Instanzen von UML-Modellen unterstützt

Design By Contract

- **Generelle Idee:** Einhaltung der Verplichtungen aus dem Vertragskonzept für Methodenaufrufe zur Laufzeit überprüfen
- Erstmals als integrales Konzept der Sprache Eiffel
- Heute gibt es DBC-Frameworks und Spezifikationssprachen für C/C++, Java, Python, etc.

Java Modelling Language

- Die Java Modelling Language (JML) ist eine der bekanntesten Design By Contract Spezifikationssprachen für Java
- Vor- und Nachbedingungen als Kommentare im Java-Quelltext
- JML-Compiler generiert Bytecode, der Assertions gemäß der Spezifikation enthält

JML-Beispiel

OCL:

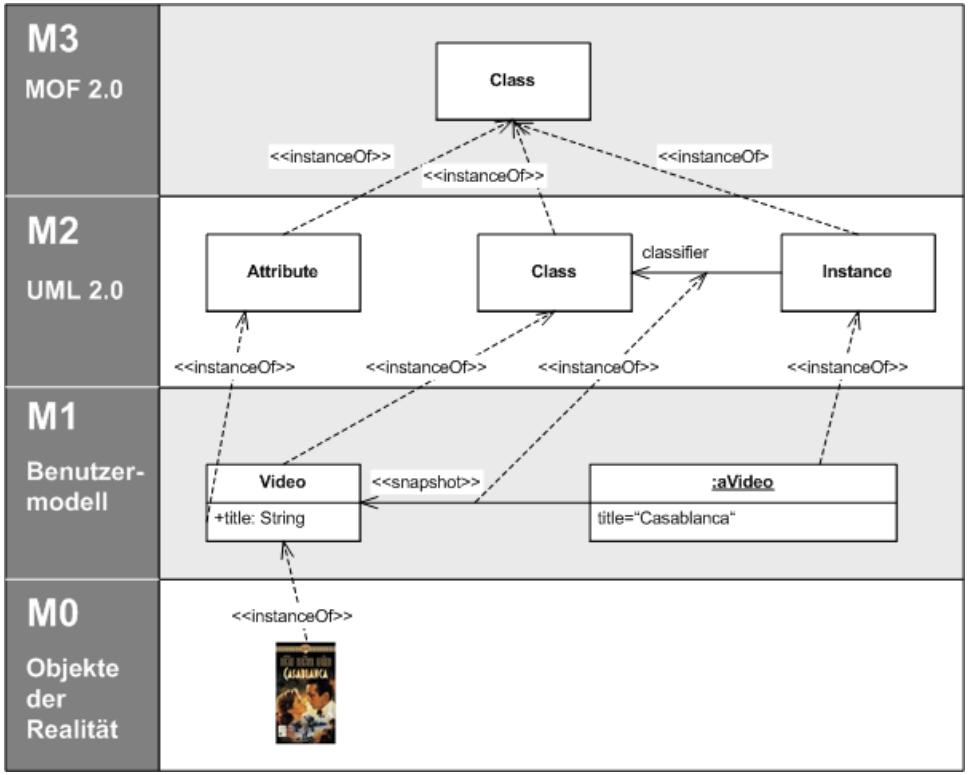
```
context Person::addKgs(kgs:Integer)
pre: (kgs >= 0) and (weight + kgs >= 0)
post: self.weight = self.weight@pre + kgs
```

JML:

```
/*@
@ requires kgs >= 0;
@ requires weight + kgs >= 0;
@ ensures weight == \old(weight + kgs);
@*/
public void addKgs(int kgs) {
  …
}
```

Metamodellierung

- Konzepte der UML (Klasse, Assoziation, …) als *Metaklassen* im *UML-Metamodell* (M2) definiert.
- UML-Modell besteht aus Instanzen von UML-Metaklassen
- UML-Metamodell selbst ist Instanz eines Meta-Meta-Modells (MOF)



OCL auf Metaebene M1

- In OCL lassen sich auch Invarianten für die Instanzen von Metamodellen (=Modelle auf M1) formulieren.
  - Beschreibung, welche Eigenschaften ein (UML-)Modell erfüllen muss
  - Signatur für OCL-Ausdrücke: UML-Metamodell (M2)
  - Z.B.
    ```
    Class.allInstances()->forAll(c | c.name <> null)
    ```
    - „Alle Klassen im Modell müssen einen Namen haben“
  - Dient also zur Definition von Modellierungs-Konventionen
  - Unterstützung in den verbreiteten UML-Tools vorhanden
    - Z.B. TopCased (Open Source)

OCL zur Sprachdefinition

- UML-Spezifikation verwendet OCL für sogenannte *well-formedness rules*
  - *Z.B. „Interfaces dürfen keine Attribute enthalten“*

```
context Interface
inv: features->select(f |
f.oclIsKindOf( Attribute ) )->isEmpty()
```

- *well-formedness rules* auch für eigene *domänenspezifische* Metamodelle (aufbauend auf MOF)
- Neben komplett neu-definierten eigenen Metamodellen: *UML-Profile*
  - *Stereotypen* fügen Semantik zu bestehenden UML-Metaklassen hinzu

OCL & Model Driven Architecture

- Model Driven Architecture (MDA)
  - Erstelle *platform independent model (PIM)* in UML + domänenspezifisches UML-Profil
  - Verwende Modelltransformationen, um aus dem PIM ein *platform specific model (PSM)* für J2EE / .NET etc. zu erzeugen
  - Verwende Modell-zu-Text-Transformationen um den fertigen Code zu erzeugen
- OCL zur Validierung von *well-formedness rules* und als Grundlage für Transformationssprachen (z.B. QVT, ATL)

Links

- DresdenOCL
  - http://dresden-ocl.sourceforge.net
- AspectJ
  - http://www.eclipse.org/aspectj/
- Design by Contract
  - http://en.wikipedia.org/wiki/Design_by_contract
- JML
  - http://www.eecs.ucf.edu/~leavens/JML/
  - http://www.eecs.ucf.edu/~leavens/JML//jmldbc.pdf
- Metamodellierung
  - http://en.wikipedia.org/wiki/Metamodeling
- MDA
  - http://en.wikipedia.org/wiki/Model-driven_architecture
  - http://www.omg.org/mda/
- TopCased UML
  - http://www.topcased.org
- Modelltransformationssprachen
  - http://de.wikipedia.org/wiki/MOF_QVT