MODEL AND OBJECT VERIFICATION

Use Cases of OCL and the Dresden OCL Toolkit

Claas Wilke and Birgit Demuth

Dresden, Oct. 15th 2009
Overview

- Introduction
- Dresden OCL Toolkit
- Verification Approaches
  - Interpretative Approach
  - Generative Approach
- OCL Use Cases
  - Interpretative Use Cases
  - Generative Use Cases
- Summary
Introduction

What is OCL?

- Part of UML
- Additional Semantics
- Queries and Constraints
- On different MOF layers
- On different Meta-Models
Introduction

An Example

def isAdult: Boolean =
  if (age >= 18)
    then true
    else false
  endif

context Person
inv: self.age >= 0

context Person: birthdayHappens()
post: self.age = self.age@pre + 1

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>age: int</td>
</tr>
</tbody>
</table>

birthdayHappens()
Dresden OCL Toolkit

About the Project

- Started in 1999
- A toolkit to extend case tools with OCL
Dresden OCL Toolkit

Dresden OCL2 for Eclipse

- Developed since 2007
- Meta-Model independent (Based on a Pivot Model)
- **Supported Meta-Models:**
  - EMF Ecore
  - Eclipse MDT UML2
  - Java
- **Provided Tools:**
  - OCL2 Parser
  - OCL2 Interpreter
  - OCL2toJava Code Generator
  - Meta-Model Adapter Generator
Dresden OCL Toolkit

The Generic Three Layer Metadata Architecture

Mn+1
Meta-Model <<is-a>> Pivot Model <<extends>> Essential OCL

Mn
Model <<instance-of>> Constraints

Mn-1
Model Instance <<are-evaluated-on>>
Verification Approaches

Two different approaches exist:

1. **Interpretative Approach**
   - Verification by interpretation

2. **Generative Approach**
   - Verification through generated check code
Verification Approaches

Interpretative Approach

1. Modeling
Verification Approaches

**Interpretative Approach**

1. Modeling
2. Constraint specification

TU Dresden, Oct. 15th 2009
Verification Approaches

**Interpretative Approach**

1. Modeling
2. Constraint specification
3. Model Instance definition/generation
Verification

Approaches

Interpretative Approach

1. Modeling
2. Constraint specification
3. Model Instance definition/generation
4. Interpretation

Verification is part of the Interpretative Approach
Verification Approaches

**Generative Approach**

1. Modeling
Verification Approaches

Generative Approach

1. Modeling
2. Constraint specification
Verification
Approaches

**Generative Approach**

1. Modeling
2. Constraint specification
3. Code generation
Verification Approaches

Generative Approach

1. Modeling
2. Constraint specification
3. Code generation

Execution and Verification are not part of the Generative Approach
OCL Use Cases

• **Interpretative Approaches**
  - Model Verification
  - Testing
  - Run-time (Object) Verification
  - Simulation/Animation
  - Querying

• **Generative Approaches**
  - Testing
  - Run-time (Object) Verification
  - Simulation/Animation
  - Model Transformation
OCL Use Cases

- **Interpretative Approaches**
  - Model Verification
  - Testing
  - Run-time (Object) Verification
  - Simulation/Animation
  - Querying

- **Generative Approaches**
  - Testing
  - Run-time (Object) Verification
  - Simulation/Animation
  - Model Transformation
OCL Use Cases

Interpretative Approaches: Model Verification

- Constraints on Meta-Models
  - Well-Formedness Rules
  - Modeling Guide-Lines

- Interpretation/Verification of Models during Modeling
OCL Use Cases

Interpretative Approaches: Model Verification - WFRs and Modeling Guidelines in UML [OMG09]

context Interface
inv featuresArePublic:
self.feature
→forall(f | f.visibility = #public)

context Class
inv SingleInheritance:
self.generalization→size()<=1
OCL Use Cases

Interpretative Approaches: Run-Time (Object) Verification

- Constraints are defined on a Model
- Interpretation/Verification of objects during run-time
OCL Use Cases

Interpretative Approaches:
Run-Time (Object) Verification in Treaty [DJ08, Tre09]

```
context DateFormatter::format(aDate: Date): String
post containsDay:
  let day: String = aDate.toString().substring(9, 10)
  in result.contains(day)
```
OCL Use Cases

Interpretative Approaches: Querying

- Queries on Models or Meta-Models
- Interpreter queries on an instance of the Model or Meta-Model
- Collect invalid Objects
- Compute Model Metrics
OCL Use Cases

Interpretative Approaches: Querying non-well-formed Model Objects in a PML model [Brä07]

context Plugin

def getIllegalPlugins() :
    Set(Plugin) =

        self.allInstances()
        ->select(id.oclIsUndefined())
OCL Use Cases

Generative Approaches: Testing

- Constraints are defined on Models
- Code Generator generates Test Code
- Test Code tests Model Instances
OCL Use Cases

Generative Approaches: Testing - JUnit Code Generation

context Person
inv ageIsPositive: self.age >= 0

In Java:

```
@Test
class TestPerson {
    @Test
    public void testAgeIsPositive() {
        Person aPerson = new Person();
        assertEquals(0, aPerson.age);
    }
}
```
Generative Approaches: Run-Time (Object) Verification

- Constraints are defined on Models
- Code Generator generates Constraint Code
- Constraint code is instrumented or woven into Model code
- Constraints are verified during Model Instance execution

TU Dresden, Oct. 15th 2009 Model and Object Verification
OCL Use Cases

Generative Approaches:
Run-Time Verification - AspectJ Code Generation [Wil09]

context Person
inv ageIsPositive: self.age >= 0

In Java:

pointcut ageChanged(Person aPerson):
  set(* Person.age) && this(aPerson);

after(Person aPerson) : ageChanged(Person) {
  if (!aPerson.age >= 0) {
    throw new RuntimeException("The age of a person must not be negative");
  }
}

TU Dresden, Oct. 15th 2009 Model and Object Verification slide 21
OCL Use Cases

Generative Approaches: Run-time Verification - OCL2 to SQL Transformation [Hei05]

**context** Person

**inv** ageIsPositive: self.age >= 0

**SQL Integrity View**
(contains all objects that violate the constraint):

```sql
create view AGEISPOSITIVE as
select * from PERSON SELF
where not (SELF.AGE >= 0)
```
Summary

• **Object Constraint Language**
  – Model Verification
  – Object Verification

• **Dresden OCL2 for Eclipse**
  – Generic Three Layer Metadata Architecture
  – Supports both Model, and Object Verification
  – A Set of Tools for other Case Tools

• **Two groups of Verification Use Cases**
  – Interpretative Approaches
  – Generative Approaches
Summary

- We are interested in other OCL use cases and your own experiences with OCL!
- Feedback is welcome!
- Dresden OCL Toolkit
  http://dresden-ocl.sourceforge.net/
- Use our mailinglists at
  http://sourceforge.net/projects/dresden-ocl/
- Direct Contact: info@claaswilke.de
References I

BRÄUER, Matthias:  
*Models and Metamodels in a QVT/OCL Development Environment.*  

DIETRICH, Jens ; JENSON, G.:  
*Treaty - A Modular Component Contract Language.*  

HEIDENREICH, Florian:  
*SQL-Codegenerierung in der metamodellbasierten Architektur des Dresden OCL Toolkit.*  
Großer Beleg (Minor Thesis), TU Dresden, May 2005. – Published in German
References II

HEIDENREICH, Florian: 
*OCL-Codegenerierung für deklarative Sprachen.* 
Diploma Thesis, TU Dresden, April 2006. – Published in German

Object Management Group (OMG): 
*OMG Unified Modeling Language*™ (*OMG UML), *Superstructure.* 
Version: 2.2, February 2009

*Treaty Project Website.* 
Google Code Project Website. 
http://code.google.com/p/treaty/. 
Version: July 2009
References III

WILKE, Claas:
Java Code Generation for Dresden OCL2 for Eclipse.
Großer Beleg (Minor Thesis), TU Dresden, February 2009
OCL Use Cases

Interpretative Approaches: WFRs in PML [Brä07]

context Plugin
inv: not self.id.oclIsUndefined()

class Plugin {
  id: String
  name: String
  version: String
  provider: String
  extensionPoints
  0..*
  services
  0..*
}

context Feature
inv: self.plugins ->isUnique(plugin | plugin.id)

class Feature {
  id: String
  name: String
  version: String
  plugins
  0..1
  features
  0..*
}

TU Dresden, Oct. 15th 2009  Model and Object Verification  slide 28
OCL Use Cases

Generative Approaches - Run-time Verification - OCL2 to XMLSchema/XQuery Transformation [Hei06]

**XMLSchema:**

```xml
<xs:complexType name="Person">
  <xs:element name="age" type="xs:integer" />
</xs:complexType>
```

**XQuery Integrity Query**

(contains all objects that violate the constraint):

```xquery
for $SELF in fn:doc("modelInstance.xml")/Person
where not ($SELF/age >= 0)
return $SELF
```