



TECHNISCHE  
UNIVERSITÄT  
DRESDEN

Fakultät Informatik, Institut SMT, Lehrstuhl Softwaretechnologie

# Einführung in OCL (Object Constraint Language)

Dr. Birgit Demuth

---

---

Rules and models destroy genius and art.

William Hazlitt  
*English writer, 1778 - 1830*

---

## Was wollen wir lernen?

- Einführung in die Thematik (Vertragsmodell, Zusicherungen, Überblick OCL, Literaturquellen)
- Sprachkonzepte und „OCL by Example“
- Anwendungsfälle für OCL
- Aktuelle OCL Tools

---

## Theoretische Grundlagen

- **Hoare-Tripel**  $\{ P \} S \{ Q \}$  [Hoare, 1969], z.B.  
 $\{ x=y \} y := y-x+1 \{ y=1 \}$
- **Design by Contract** (Vertragsmodell) [Meyer, 1997],  
Übertragung auf Klassen und Methoden

## Softwarepionier C.A.R. Hoare

**Software-Pioniere,  
sd&m-Konferenz 2001 in Bonn**



"What a wonderful set of resources!  
Congratulations on assembling this marvelous collection."  
Grady Booch

[www.sdm.de/de/publikationen/konferenzen/softwarepioniere2001/](http://www.sdm.de/de/publikationen/konferenzen/softwarepioniere2001/)

---

## Zusicherungen (defensive Programmierung)

- **Vorbedingung:**
  - muss die Kundenklasse einhalten
- **Nachbedingung:**
  - garantiert die Anbieterklasse
- **Klasseninvariante:**
  - muss von allen Methoden der Anbieterklasse eingehalten werden (vor und nach jeder Methodenausführung)
  - gilt während der gesamten Lebensdauer der Objekte der Klasse

---

## Formulierung von Zusicherungen

- *Modellbasiert:*
  - OCL
- *In Programmiersprachen:*
  - Eiffel
  - JASS (Java with ASSertions)
  - JML (Java Modeling Language)
  - Java (assert)

---

## OCL (Object Constraint Language)

- ergänzt die Unified Modeling Language (UML)
- formale Sprache für die Definition von Constraints (Zusicherungen) und Anfragen auf UML-Modellen
- standardisiert (OMG)
- deklarativ
- seiteneffektfrei
- fügt graphischen (UML-)Modellen präzisierte Semantik hinzu
- verallgemeinert für alle MOF-basierten Metamodelle
- inzwischen allgemein akzeptiert, viele Erweiterungen
- „Core Language“ von Modelltransformationssprachen (QVT), Regelsprachen (PRR) ...



---

## Literatur

- [1] Warmer, J., Kleppe, A.: The Object Constraint Language. Precise Modeling with UML. Addison-Wesley, 1999
- [2] Warmer, J., Kleppe, A.: The Object Constraint Language Second Edition. Getting Your Models Ready For MDA. Addison-Wesley, 2003
- [3] OMG UML specification,  
[www.omg.org/technology/documents/modeling\\_spec\\_catalog.htm#UML](http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML)
- [4] OMG UML 2.0 OCL,  
[www.omg.org/technology/documents/formal/ocl.htm](http://www.omg.org/technology/documents/formal/ocl.htm)
- [5] Brügge, B., Dutoit, A.H.: Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. PEARSON Studium, 2004 (kurze Zusammenfassung)

OCL Portal - Home - Mozilla Firefox

http://web.inf.tu-dresden.de/~s9600916/OCL-Portal/

Erste Schritte Aktuelle Nachrichten ...

Lehrstuhl für Softwaretechnologie www.softvis.org OCL Portal - Home

search...

# OCL Portal

*"The center for OCL related information."*

Home News OCL specification OCL software Activities Related ressources Courses Projects Publications Impressum Contact

## The OCL portal

Written by Administrator  
Montag, 21 November 2005

The Object Constraint Language (OCL) is a textual sublanguage of the Unified Modelling Language (UML). It can be used to express additional constraints on UML models that cannot be expressed, or are very difficult to express, with the graphical means provided by UML. OCL is based on first-order predicate logic but it uses a syntax similar to programming languages and closely related to the syntax of UML. It is, thus, more adequate for every-day modelling than pure first-order predicate logic.

Last Updated ( Freitag, 21 April 2006 )  
[Read more...](#)

### Popular

- [Dresden OCL Toolkit](#)
- [The OCL portal](#)
- [Impressum](#)
- [Contact person](#)
- [The professional site of Jos Warmer and Anneke Kleppe](#)

Username

Password

Remember me

[Lost Password?](#)  
[No account yet?](#)  
[Register](#)

# Constraint

## Definition nach [1]

- „A **constraint** is a restriction on one or more values of (part of) an object-oriented model or system.“

## Definition nach [5]

- „Eine **Einschränkung** ist ein Prädikat, dessen Wert wahr oder falsch ist.
- Boolesche Ausdrücke sind ... Einschränkungen. ...
- OCL erlaubt die formale Spezifikation von Einschränkungen für einzelne Modellelemente (z.B. Attribute, Operationen, Klassen) sowie für Gruppen von Modellelementen (z.B. Assoziationen)“
- Wir benutzen im folgenden weiter den Begriff des Constraints.

---

# Invariante

## Definition

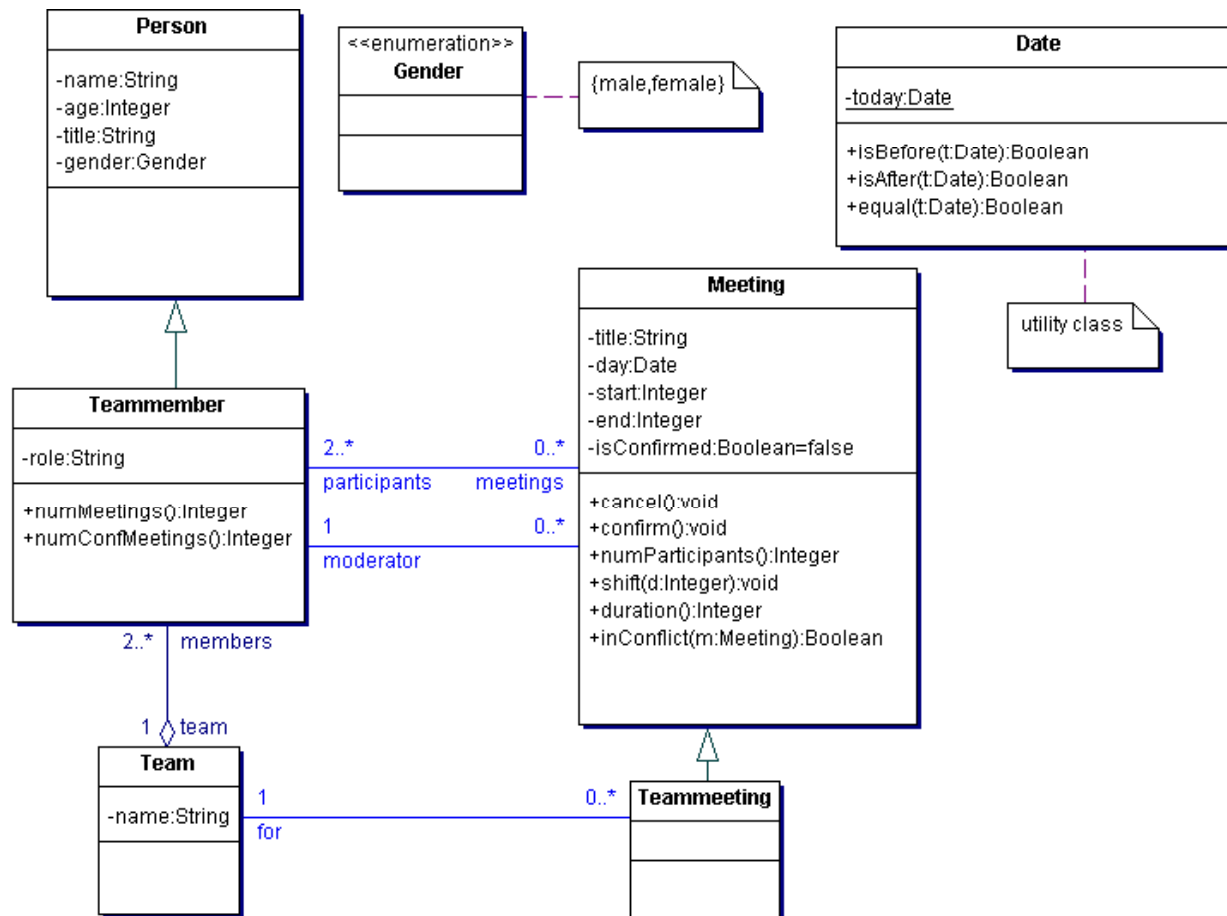
- Eine **Invariante** ist ein Constraint, das für ein Objekt während seiner ganzen Lebenszeit wahr sein sollte.

## Syntax

`context <class name>`

`inv [<constraint name>]: <OCL expression>`

# OCL/UML By Example



---

## Invariante - Beispiel

```
context Meeting inv: self.end > self.start
```

### Äquivalente Formulierungen

```
context Meeting inv: end > start
```

-- *self* bezieht sich immer auf das Objekt, für das das  
Constraint berechnet wird

```
context Meeting inv startEndConstraint:  
self.end > self.start
```

-- Vergabe eines Namens für das Constraint

- Sichtbarkeiten von Attributen u.ä. werden durch OCL standardmäßig ignoriert.

## Precondition (Vorbedingung)

- Pre- und Postconditions sind Constraints, die die Anwendbarkeit und die Auswirkung von Operationen spezifizieren, ohne dass dafür ein Algorithmus oder eine Implementation angegeben wird.

### Definition

- Eine **Precondition** ist ein Boolescher Ausdruck, der zum Zeitpunkt des Beginns der Ausführung der zugehörigen Operation wahr sein muss.

### Syntax

```
context <class name>::<operation> (<parameters>)  
pre [<constraint name>]: <OCL expression>
```

## Precondition - Beispiele

```
context Meeting::shift(d:Integer)
pre: self.isConfirmed = false
```

```
context Meeting::shift(d:Integer)
pre: d>0
```

```
context Meeting::shift(d:Integer)
pre: self.isConfirmed = false and d>0
```



## Postcondition (Nachbedingung)

### Definition

- Eine **Postcondition** ist ein Boolescher Ausdruck, der unmittelbar nach der Ausführung der zugehörigen Operation wahr sein muss.

### Syntax

```
context <class name>::<operation> (<parameters>)  
post [<constraint name>]: <OCL expression>
```

## Postcondition - Beispiele

```
context Meeting::duration():Integer
post: result = self.end - self.start
```

-- *result* bezieht sich auf den Rückkehrwert der Operation

```
context Meeting::confirm()
post: self.isConfirmed = true
```

```
context Meeting::shift(d:Integer)
post: start = start@pre +d and end = end@pre + d
```

-- *start@pre* bezieht sich auf den Wert **vor** Ausführung der  
-- Operation

-- *start* bezieht sich auf den Wert **nach** Ausführung der Operation

-- *@pre* ist nur in Postconditions erlaubt

---

## OCL-Ausdrücke <OCL expression> (1)

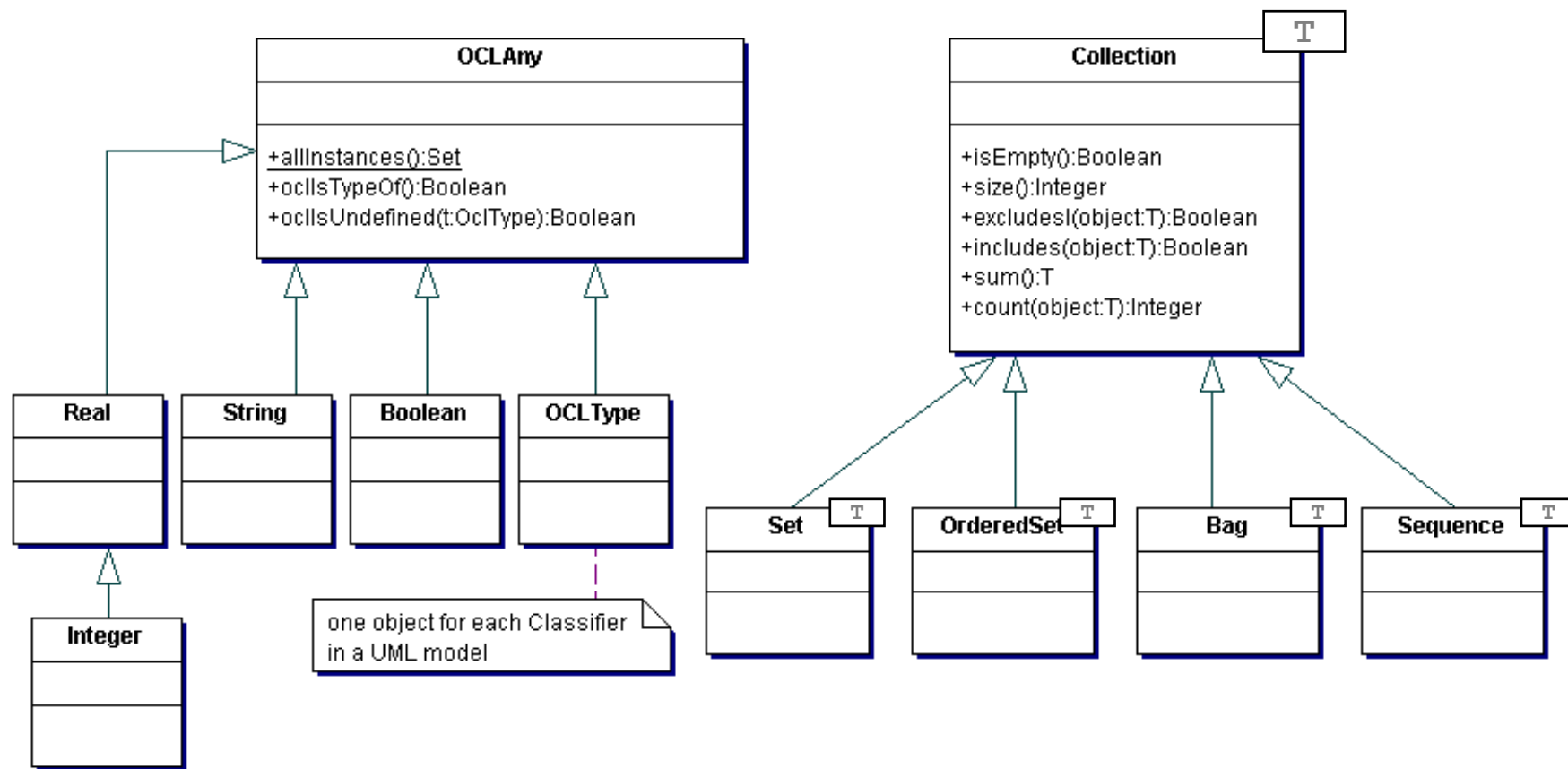
- Boolesche Ausdrücke
- Standardbibliothek
  - *Basistypen:*
    - Boolean
    - Integer
    - Real
    - String
  - *Kollektionstypen:*
    - Collection
    - Set
    - Ordered Set (nur OCL2)
    - Bag
    - Sequence

## OCL-Ausdrücke <OCL expression> (2)

### Nutzerdefinierte Typen (OCLType)

- **Klassentyp (Modelltyp):**
  - Klasse in einem Klassendiagramm
  - Generalisierung zwischen den Klassen führt zu **Supertypen**.
  - Eine Klasse besitzt die folgenden **Features**:
    - Attribute (z.B. `start`)
    - Operationen (nur *query operations*) (z.B. `duration()`)
    - Klassenattribute (z.B. `Date::today`)
    - Klassenoperationen
    - Assoziationsenden („Navigationsausdrücke“)
- **Aufzählungstyp (enumeration typ)**

## OCL-Typhierarchie mit Standardtypen (Ausschnitt)



## OCL-Typ-Konformitätsregeln

OCL ist eine **typsichere** Sprache.

Der Parser prüft OCL-Ausdrücke auf *Konformität*:

- Typ 1 ist konform zu Typ 2, wenn eine Instanz von Typ 1 an jeder Stelle ersetzt werden kann, wo eine Instanz vom Typ 2 erwartet wird.

### Allgemeine Regeln

- Typ 1 ist konform zu Typ 2, wenn Sie identisch sind.
- Jeder Typ ist konform zu jedem seiner Supertypen.
- Typkonformität ist transitiv.

---

# OCL Constraints und Vererbung

## Constraints allgemein

- Constraints einer Superklasse werden von den Subklassen geerbt. Dabei gilt das Liskovsche Substitutionsprinzip.

## Invarianten

- Eine Subklasse kann die Invariante verstärken, sie aber nicht abschwächen.

## Preconditions

- Eine Vorbedingung kann bei einem Überschreiben einer Operation einer Subklasse aufgeweicht, aber nicht verstärkt werden.

## Postconditions

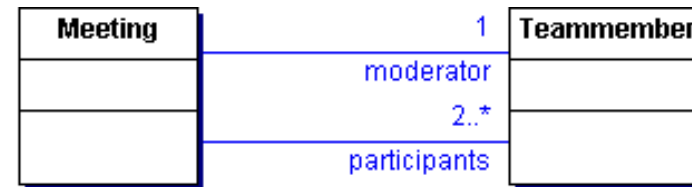
- Eine Nachbedingung kann bei einem Überschreiben einer Operation einer Subklasse verstärkt, aber nicht aufgeweicht werden.

## Navigationsausdrücke

- Assoziationsenden (Rollennamen) können verwendet werden, um von einem Object im Modell/System zu einem anderen zu navigieren (**Navigation**)
- Navigationen werden in OCL als Attribute behandelt (*dot-Notation*).
- Der Typ einer Navigation ist entweder
  - **Nutzerdefinierter Typ** (Assoziationsende mit Multiplizität maximal 1)
  - **Kollektion** von nutzerdefinierten Typen (Assoziationsende mit Multiplizität  $> 1$ )



## Navigationsausdrücke - Beispiele



### Nutzerdefinierter Typ

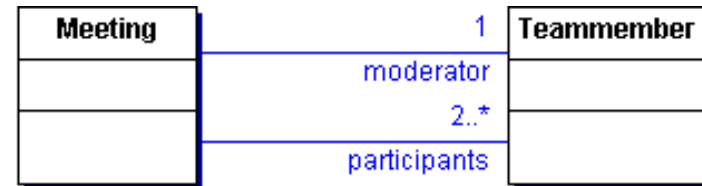
z.B. moderator

Navigation von Meeting ist vom Typ Teammember

context Meeting

inv: **self.moderator**.gender = Gender::female

## Navigationsausdrücke - Beispiele



### Kollektion

- Z.B. `participants` Navigation von `Meeting` ist vom Typ `Set (Teammember)`
- Operationen auf Kollektionen werden in der „Pfeilnotation“ (`->`) geschrieben
- Kurznotation für die `collect`-Operation ist die `dot`-Notation (für `self->collect(participants)` besser `self.participants`)

**context Meeting**

**inv: `self->collect(participants)->size()>=2`**

**context Meeting inv: `self.participants->size()>=2`**

## Operationen auf Kollektionen (1)

22 Operationen mit unterschiedlicher Semantik in Abhängigkeit vom Kollektionstyp, z.B.

- Vergleichsoperationen (`=`, `<>`)
- Konvertierungsoperationen (`asBag()`, `asSet()`, `asOrderedSet()`, `asSequence()`)
- Verschiedene including- und excluding-Operationen
- Operation `flatten()` erzeugt aus einer Kollektion von Kollektionen eine Kollektion mit einzelnen Objekten, z.B.  
 $\text{Set}\{\text{Bag}\{1, 2, 2\}, \text{Bag}\{2\}\} \rightarrow \text{Set}\{1, 2\}$
- Mengenoperationen  
(`union`, `intersection`, `minus`, `symmetricDifference`)
- Operationen auf sortierten Kollektionen (z.B. `first()`, `last()`, `indexOf()`)

---

## Operationen auf Kollektionen (2)

**Iterierende** Operationen auf allen Kollektionstypen, z.B.

```
any(expr)
collect(expr)
exists(expr)
forall(expr)
isUnique(expr)
one(expr)
select(expr)
reject(expr)
sortedBy(expr)
```

## Operation iterate()

```
Collection->iterate( element : Type1;  
                    result  : Type2 = <expression>  
                    | <expression with element and result> }
```

- Alle anderen iterierenden Operationen sind ein Spezialfall von iterate() und können damit ausgedrückt werden, z.B.

Set {1,2,3}->**sum()** durch

Set{1,2,3}->

```
iterate{i: Integer, sum: Integer=0 | sum + i }
```

## Weitere Beispiele für Kollektionsoperationen (1)

- Ein Teammeeting muss für ein ganzes Team organisiert werden (Operation `forall()`):

```
context Teammeeting
```

```
inv: participants->forall(team=self.for)
```

```
context Meeting inv: oclIsTypeOf(Teammeeting)
```

```
implies participants->forall(team=self.for)
```

## Weitere Beispiele für Kollektionsoperationen (2)

- Weitere Nachbedingungen (`select()`):

```
context Teammember::numMeeting():Integer
post: result=meetings->size()
```

```
context Teammember::numConfMeeting():Integer
post:
result=meetings->select(isConfirmed)->size()
```

## OCL für abgeleitete Attribute und Assoziationen (Ableitungsregeln, derive, OCL2)

- Beispiel für ein **abgeleitetes Attribut** (size)

```
context Team::size:Integer  
derive:members->size()
```

- Beispiel für eine **abgeleitete Assoziation**
  - conflict definiert miteinander (zeitlich) in Konflikt stehende Meetings

```
context Meeting::conflict:Set(Meeting)  
derive: select(m|m<>self and  
self.inConflict(m))
```



## OCCL zur Spezifikation von Anfangswerten (init, OCL2)

### Beispiele

```
context Meeting::isConfirmed : Boolean  
init: false
```

```
context Teammember:meetings : Set(Meetings)  
init: Set{}
```

- Man beachte den Unterschied zu Invarianten und Ableitungsregeln: Ein Anfangswert muss nur zum Zeitpunkt der Erzeugung des Objektes gelten!

## OCIL zur Spezifikation von Anfrageoperationen (body, OCL2)

- Spezifikation von Operationen ohne Seiteneffekte (d.h. Operationen, die nicht den Zustand irgendeines Objektes im System ändern)
- Volle Ausdruckskraft einer Anfragesprache (vergleichbar mit SQL)

### Beispiel

**context**

**Teammember::getMeetingTitles(): Bag(String)**

**body: meetings->collect(title)**

## Teilausdrücke in OCL (let)

- Interessant in komplexen OCL-Ausdrücken
- Ein let-Ausdruck definiert eine Variable (z.B. noConflict), die anstelle eines Teilausdruckes benutzt werden kann.

### Beispiel

**context Meeting inv:**

```
let noConflict : Boolean =  
    participants.meetings->forall(m|m<>self  
    and  
    m.isConfirmed implies not  
    self.inConflict(m))
```

**in**

```
isConfirmed implies noConflict
```

---

## OCL zur Spezifikation von wiederverwendbaren Ausdrücken (def, OCL2)

- Definition von Attributen und Anfrageoperationen
- Verwendung wie normale Attribute und Operationen
- Syntax ist ähnlich dem let-Ausdruck
- Gedacht für die Wiederverwendung von OCL-Teilausdrücken in **verschiedenen** Constraints

**context Meeting**

```
def: noConflict : Boolean =  
    participants.meetings->forall(m|m<>self  
        and  
        m.isConfirmed implies not  
        self.inConflict(m))
```

---

## Zusammenfassung von OCL Expressions zu Packages

```
package MeetingExample
```

```
context Meeting::isConfirmed : Boolean  
init: false
```

```
context Teammember:meetings : Set(Meetings)  
init: Set{}
```

```
..
```

```
endpackage
```

## Grenzen von OCL

- Für **inkonsistente** Spezifikationen (Kombination von Constraints, die sich widersprechen) gibt es keine Unterstützung, diese aufzufinden.
- **„Frame Problem“**
  - Nachbedingungen beziehen sich nur auf lokale Attribute und Assoziationen.
  - Für den Rest der Objektkonfiguration gilt die Annahme „es ändert sich nichts“.
  - Das kann zu Konflikten führen (Forschungsgegenstand).
- **Transitive Hülle** kann nicht spezifiziert werden.
- **allInstances()-Problem:**
  - Für Klassentypen erlaubt, z.B. `Person.allInstances()`
  - Für unendliche Mengen von Typen nicht erlaubt, z.B. `Integer.allInstances()`

---

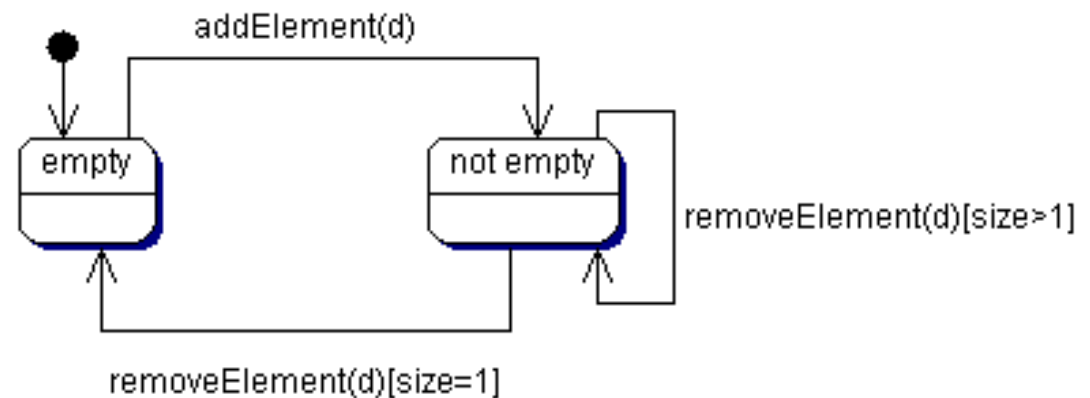
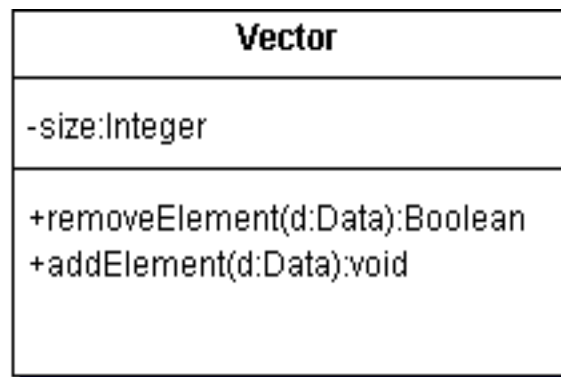
## OCL in weiteren Modellen [2]

- Zustandsdiagramm
- Sequenzdiagramm
- Aktivitätsdiagramm
- Anwendungsfalldiagramm
- Komponentendiagramm

## OCL in Zustandsmodellen – Beispiel (oclInState())

- vordefiniertes Prädikat für alle Objekte (Typ `OclAny`)

**`oclInState(s: OclState) : Boolean`**



```

context Vector::removeElement(d:Data)
pre:  oclInState(notEmpty)
post: size@pre = 1 implies oclInState(empty)
    
```



## Undefinierte Werte in OCL (OclVoid)

- Die Berechnung eines OCL-Teilausdruckes kann u.U. zu einen undefinierten Wert (**OclVoid**) führen
- dreiwertige Logik analog SQL
- Test auf undefinierten Wert mit

### **oclIsUndefined(): Boolean**

-- true falls das Objekt (vom Typ OclAny) undefiniert ist,  
-- ansonsten false

- typischer Fall des Auftreten undefinierter Werte ist der Zugriff auf einen nicht existierenden Attributwert

## Typische Anwendungsfälle für OCL

**Metamodelle:** {MOF-, Ecore-basiert} X {UML, CWM, ODM, SBVR, PRR, nutzerdefinierte DSLs, ...}

MOF-Modellebene	Beispiele für die Verwendung von OCL
<b>M2 (Metamodell)</b>	<ul style="list-style-type: none"> <li>•Spezifikation von <b>Well-Formedness Rules (WFRs)</b> in OMG-Standards</li> <li>•Definition von Modellierungsrichtlinien für <b>DSLs (Domain Specific Languages)</b></li> <li>•Spezifikation von <b>Modellabbildungen</b></li> </ul>
<b>M1 (Modell)</b>	<ul style="list-style-type: none"> <li>•Überprüfung der Konsistenz von Modellen mit dem Metamodell (→ CASE-Tool)</li> <li>•Evaluation von Modellierungsrichtlinien in DSL-Instanzen</li> <li>•Ausführung von Modellabbildungen</li> </ul>
	<ul style="list-style-type: none"> <li>•Spezifikation von <b>Geschäftsregeln/Constraints</b></li> <li>•Spezifikation von <b>Testfällen</b></li> </ul>
<b>M0 (Objekte)</b>	<ul style="list-style-type: none"> <li>•Evaluation von Geschäftsregeln/Constraints</li> <li>•Ausführung von Testfällen</li> </ul>

## Beispiele für OCL auf der Metamodellebene

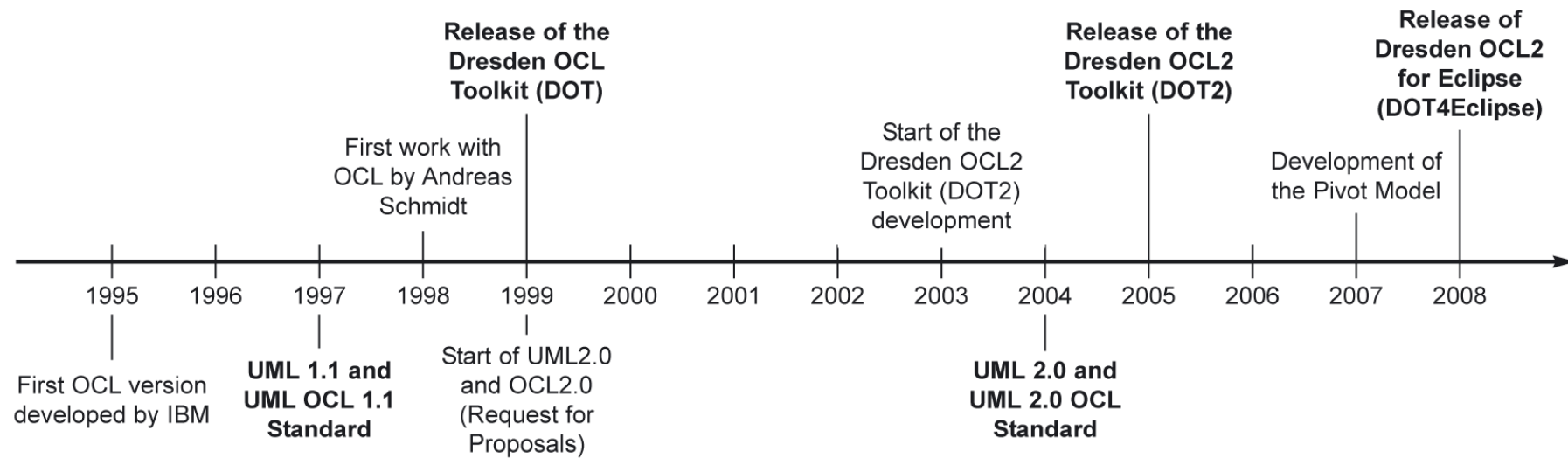
- WFR im UML-Metamodell

```
context GeneralizableElement inv:  
not self.allParents->includes(self)  
-- Zyklen in der Vererbungshierarchie sind nicht erlaubt
```

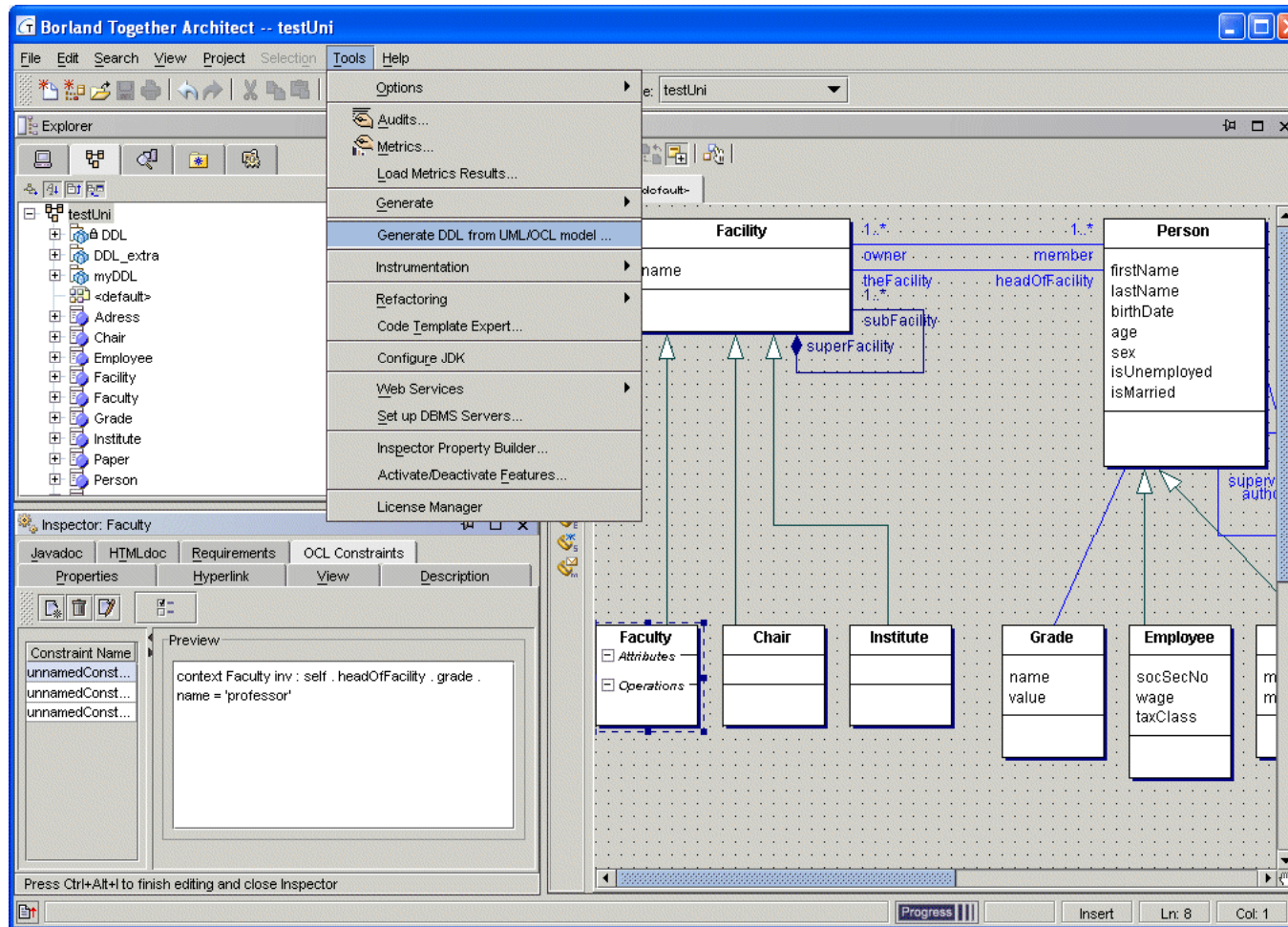
- UML Modellierungsrichtlinie/Teil eines UML-Profil bzw. einer DSL

```
context GeneralizableElement inv:  
self.generalization->size() <= 1  
-- Java-spezifisch: kein mehrfache Vererbung
```

## Dresden OCL (1)



# Dresden OCL Toolkit



The screenshot shows the Borland Together Architect interface for a project named 'testUni'. The main workspace displays a UML class diagram with the following elements:

- Facility Class:** Contains an attribute 'name'. It is the superclass for 'Chair' and 'Institute'. It has a self-referencing association 'superFacility' and an association 'owner' to the 'Person' class.
- Person Class:** Contains attributes 'firstName', 'lastName', 'birthDate', 'age', 'sex', 'isUnemployed', and 'isMarried'. It has an association 'member' to the 'Facility' class and a 'superv authority' association to the 'Employee' class.
- Chair Class:** Inherits from 'Facility'.
- Institute Class:** Inherits from 'Facility'.
- Grade Class:** Inherits from 'Person' and has an attribute 'value'.
- Employee Class:** Inherits from 'Person' and has attributes 'socSecNo', 'wage', and 'taxClass'.

The 'Inspector: Facility' window is open, showing the 'OCL Constraints' tab. The preview area displays the following constraint:

```

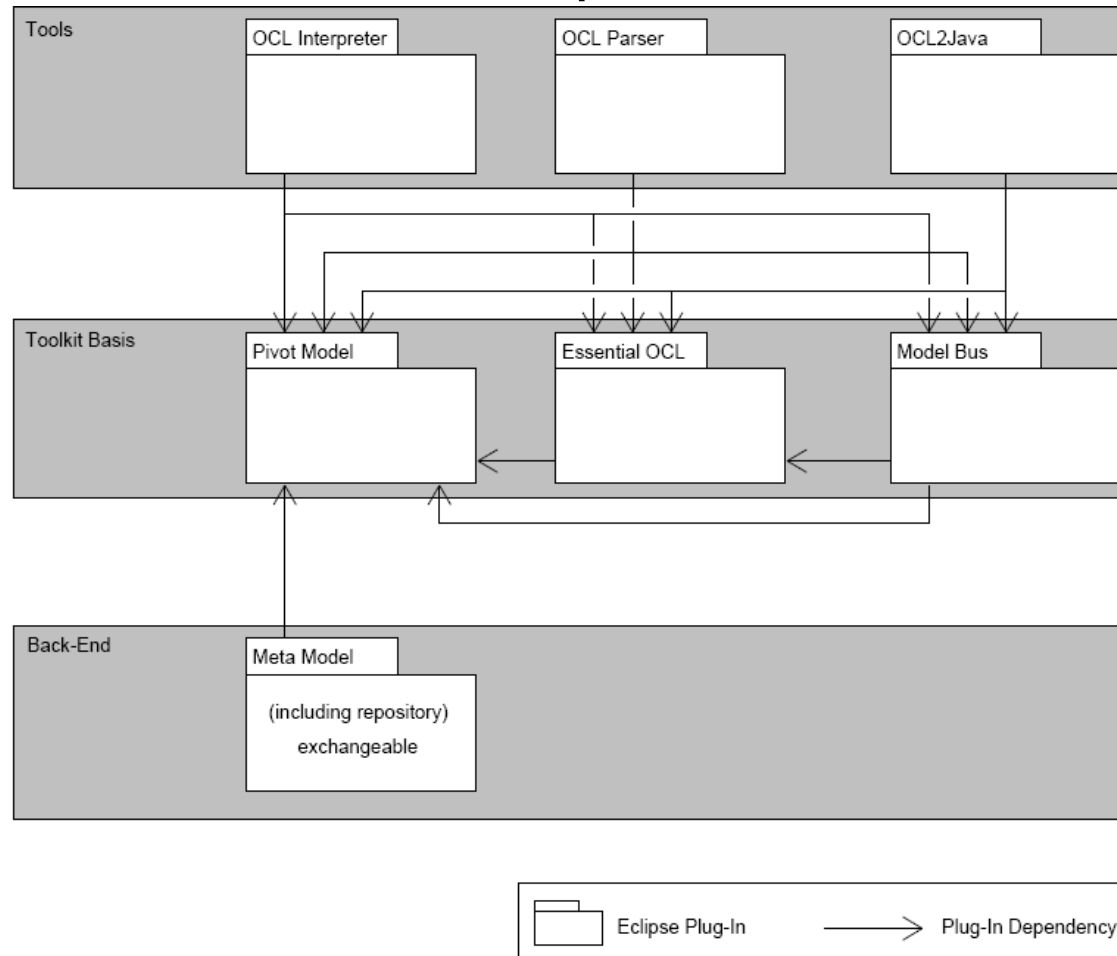
context Facility inv : self . headOfFacility . grade .
name = 'professor'
    
```

The 'Tools' menu is open, highlighting the 'Generate DDL from UML/OCL model ...' option.

## Dresden OCL (2)

- OCL2 Workbench (Stefan Ocke)/Infrastruktur für das OCL2 Toolkit v1.0
- OCL2 Parser (Ansgar Konermann, Version 2.0 Nils Thieme)
- SQL Codegenerator (Florian Heidenreich)
- Declarative Codegenerator (Florian Heidenreich)
- Transformation Framework for MDSD (Christian Wende)
- Visualisierung Plugin for Eclipse (Kai-Uwe Gärtner)
- Java Codegenerator (Ronny Brandt)
- OCL Editor for Eclipse (Mirko Stölzel)
- Anbindung an Fujaba (Mirko Stölzel)
- OCL2 Interpreter (Ronny Brandt)
- Dresden OCL2 Toolkit (Pivotmodell Matthias Bräuer)
- **Dresden OCL2 for Eclipse** Release 1.2 (Interpreter, Java-Code-Generator Claas Wilke)

# Dresden OCL2 for Eclipse

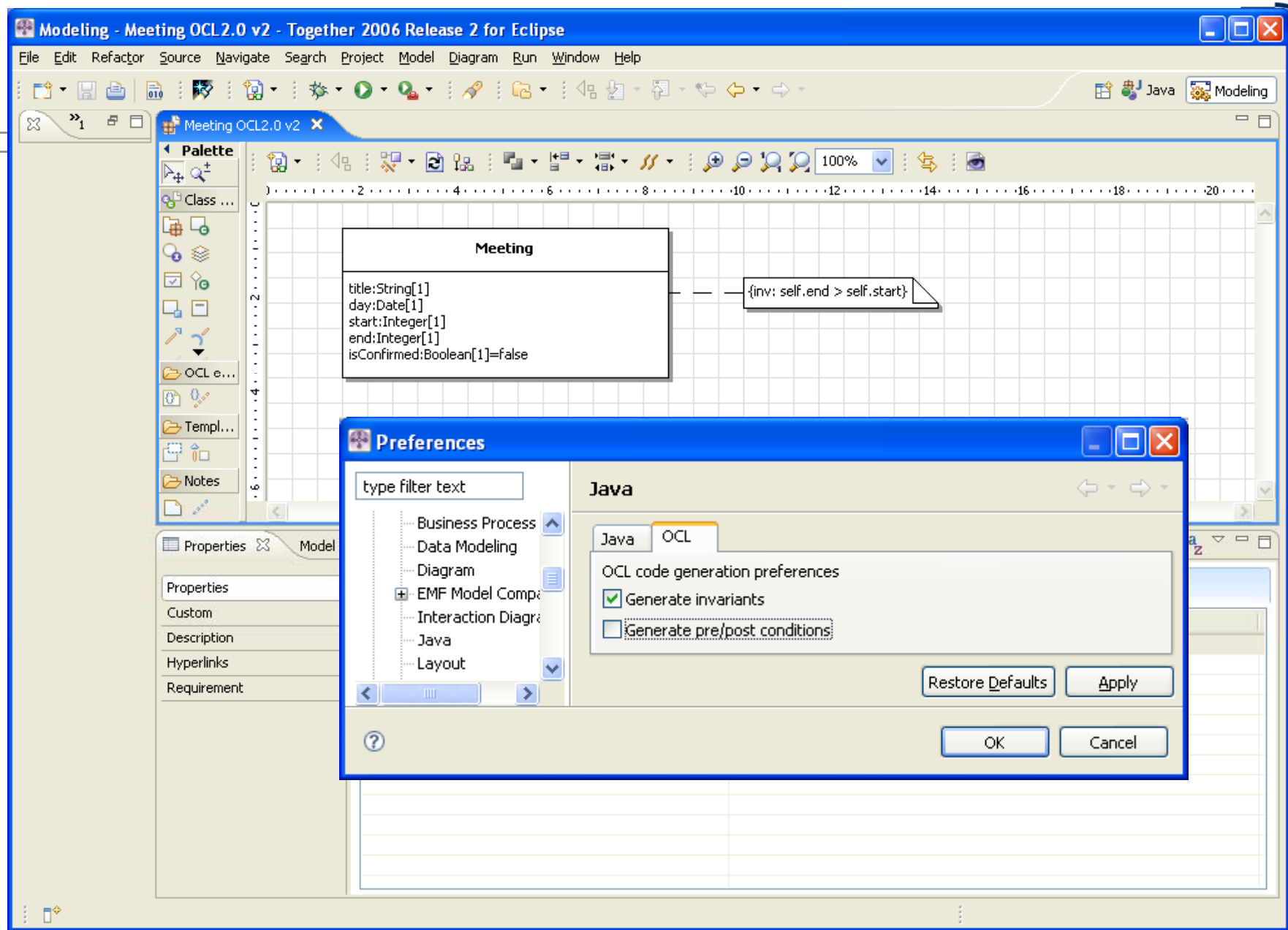


---

## Einige weitere UML/OCL Tools

- **MagicDraw** Enterprise Edition v16.5 (mit Dresden OCL2 Toolkit 😊)
  - Evaluations-Lizenz
- Borland **Together** 2008 (OCL/QVT)
  - Akademische Lizenz an der Fakultät
- **Eclipse MDT/OCL** for EMF Based Models
  - Frei verfügbar
- **Use** (Universität Bremen)
  - Frei verfügbar
  - Animation, sehr schön geeignet für Lehrzwecke





---

# OCL Support in MagicDraw Enterprise Edition

## “OCL validation rules”

1. Spezifikation auf UML Metaklassen (M2) / Verifikation von UML-Modellen (M1)
2. Spezifikation von Stereotypen (M2) / Verifikation von UML-Modellen (M1)
3. Spezifikation auf UML-Modellen (M1) / Verifikation von UML-Instanzen (Objekten)

MagicDraw UML 16.5 - OCL Lecture Samples.mdzip [C:\20090520\_birgit\_demuth\MagicDraw Samples\]

File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help

C:\20090520\_birgit\_demuth\MagicDraw Samples\ OCL Lecture Samples.mdzip View Readme (Windows)

Containment Inheritance Diagrams Model Extensions

Containment

- Data
  - Classes
    - Relations
      - Association[Meeting - participants:Teammember]
    - Meeting
      - start : Integer = 12
      - end : Integer = 10
      - participants : Teammember [2..\*]
    - Teammember
  - Instances
    - Birgit : Teammember
    - Chef : Teammember
    - Clas : Teammember
    - DresdenOCLMeeting : Meeting
    - Katrin : Teammember
    - Sebastian : Teammember
    - STMeeting : Meeting
  - Object verification <<validationSuite>>
    - { } inv1=self.participants ->size()>= 2 <<validationRule>>
    - { } inv2=end>start <<validationRule>>
  - OCLByExample1
  - Code engineering sets

Common

- Note
- abc Text Box
- Anchor
- Containment
- Dependency
- Image Shape
- Separator
- Class Diagram
  - Class
  - Interface
  - Package
  - Generalization
  - Association
  - Aggregation
  - Composition
  - Interface Realization
  - Usage
  - Abstraction
  - Collaboration
  - Instance
  - Link
- Use Case Diagram
- Implementation Diagram
- Composite Structure Diagram
- Information Flows
- Profiling Mechanism

package Data [ OCLByExample1 ]

```

classDiagram
    class Meeting {
        start : Integer = 12
        end : Integer = 10
    }
    class Teammember {
    }
    Meeting "1" -- "2..*" Teammember : -participants
    Meeting "1" -- "*" Meeting : -participants
    Meeting "1" -- "1" Teammember : DresdenOCLMeeting
    Meeting "1" -- "1" Teammember : STMeeting
    Meeting "1" -- "1" Teammember : STMeeting
  
```

Validation Results

Validation Results

Filter: >=debug <ALL> <ALL> N...

Element	Severity	Abbreviation	Error Message	Is Ign...
DresdenOCLMeeting : Meeting [Instances]	error	min2partici...	Meeting should involve at least 2 participants	
STMeeting : Meeting [Instances]	error	min2partici...	Meeting should involve at least 2 participants	
STMeeting : Meeting [Instances]	error	startBefor...	End time should be larger than start time	

Ready

---

## XMI Import für Dresden OCL2 for Eclipse

- TopCased (EMF UML2 XMI)
- MagicDraw (EMF UML2 XMI)
- Visual Paradigm (EMF UML2 XMI)
- Eclipse UML2 / UML2 Tools (EMF UML2 XMI)

## Stellenausschreibung 😊

Wir suchen Studentinnen/Studenten,  
die mit Lust und Liebe an **Dresden OCL2 for Eclipse** bauen.

Dies ist als längerfristiger Job anzusehen.

Die Einarbeitung in das Toolkits ist gleichzeitig  
eine hervorragende Vorbereitung für einen Großen Beleg/Bachelor-/  
Diplomarbeit im Rahmen unserer Forschungsprojekte!

Der Dank der Open Source-Gemeinde und des Lehrstuhl ist gewiss 😊  
Unterstützung in der technischen Arbeit gibt es insbesondere durch all  
jene Studenten, die sich bislang als hervorragende OCL-Toolentwickler  
erwiesen haben.

Kontakt: Birgit Demuth