

Einführung in OCL (Object Constraint Language)

Dr. Birgit Demuth

It is commonly thought that 10 years is needed for technology to pass from its initial conception into wide–spread use.

W. E. Riddle

The magic number eighteen plus or minus three: a study of software technology maturation.

SIGSOFT Softw. Eng. Notes 9(2):21–37, 1984

Was wollen wir lernen?

- Einführung in die Thematik (Vertragsmodell, Zusicherungen, Überblick OCL)
- Sprachkonzepte und „OCL by Example“
 - OCL-Typen und OCL-Ausdrücke
 - Weitere OCL-Anweisungen
- Anwendungsfälle für OCL und Diskussion
- OCL Tools

EINFÜHRUNG

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
 - Wenn die Klasse K1 eine Methode M der Klasse K2 in Anspruch nimmt, muss K1 sicherstellen, dass vor Ausführung von M deren Vorbedingungen erfüllt ist. K2 garantiert dann, dass nach Abschluss der Methode M die Nachbedingung gilt.

Zusicherungen

- **Vorbedingung:**
 - garantiert die „Kundenklasse“
- **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 Modellierungssprachen (UML), hybride Sprache
- formale Sprache für die Definition von Constraints (Zusicherungen) und Anfragen auf UML-Modellen
- standardisiert (OMG), derzeit OCL 2.3.1 (January 2012)
- deklarativ
- seiteneffektfrei
- typisiert
- 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
- [4] OMG OCL, <http://www.omg.org/spec/OCL/>
- [5] Wolfgang Zuser et al: Softwaretechnologie für Einsteiger.
Vorlesungsunterlage für die Veranstaltungen an der TU Dresden.
Pearson Studium, 2009, S. 145-152

• [About OCL Portal](#)

• [News](#)

OCL Portal

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.

Although OCL is thus an important tool, it can be hard to find information about it. For example, there is still a lack of good teaching modules for OCL. Furthermore, case studies on projects could help potential users to find out whether or not OCL is the proper formalism for their problems.

The worldwide OCL community has developed various (open-source) OCL tools. These can and should be integrated into UML CASE tools to support precise specification of UML models beyond the pure specification of OCL expressions as strings.

For this reason, the OCL community decided at the [2005 Workshop on OCL](#) at the [MoDELS conference](#) to set up an OCL portal website collecting all information about OCL. This is it!

This page relies on your collaboration. Therefore you are invited to put your OCL activities/knowledge at the portal! The OCL Portal provides editing capabilities both to members of the academic/research OCL community and OCL users/developers of CASE tools in industrial environments. Please register and then you will (after the confirmation procedure) be able to add your own teaching material, projects, toolkits, case studies etc. Furthermore, we are always happy on feedback about this site and will make every effort to improve the site in accordance with your needs and suggestions.

search...

> SEARCH

LATEST ENTRIES

- [News about Dresden OCL](#)
- [OCL 2.3.1 Specification](#)
- [SimpleOCL](#)
- [VMTS](#)
- [Visual OCL](#)

Constraint

Definition nach [1]

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

In deutschen Lehrbüchern:

- Zusicherung
- Einschränkung
- Integritätsbedingung
- Randbedingung

Invariante

Definition

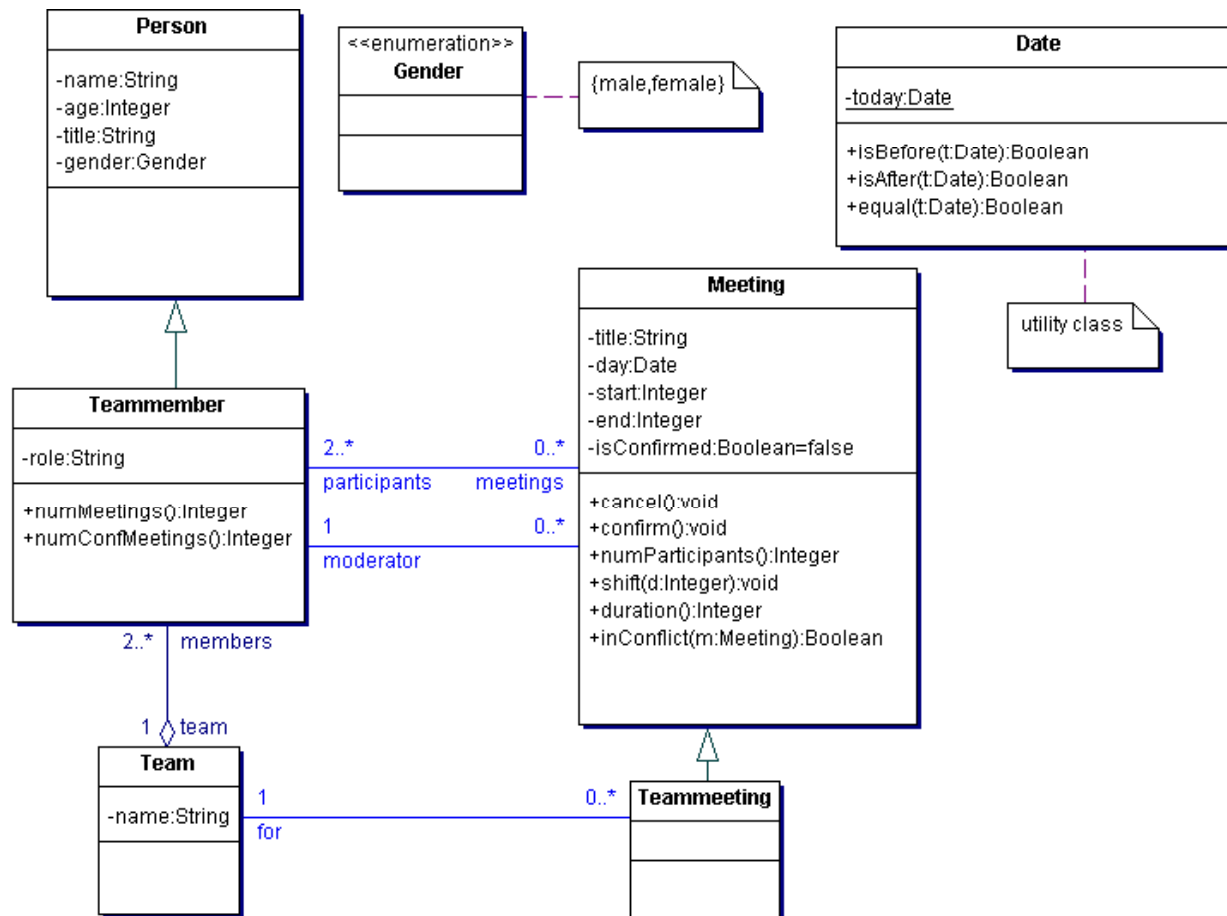
- Eine **Invariante** ist ein Constraint, das für ein Objekt während seiner ganzen Lebenszeit wahr sein sollte.

Syntax

`context <class name>`

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

OCL/UML By Example



Invariante - Beispiel

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

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

Äquivalente Formulierungen

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

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

-- Vergabe eines Namens für das Constraint

- Sichtbarkeiten von Attributen 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

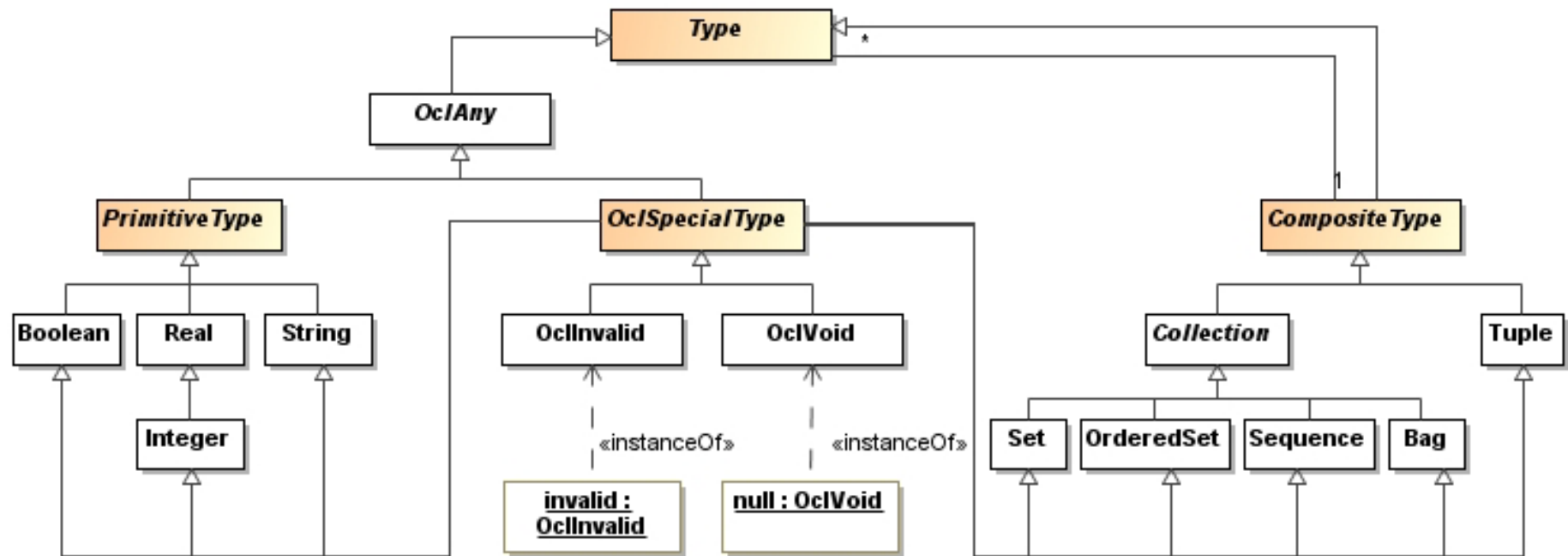
-- Operation

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

-- @pre ist nur in Postconditions erlaubt

OCL-TYPEN UND OCL-AUSDRÜCKE

Basistypen der OCL-Standardbibliothek



Standardoperationen auf Strings

String
<code>+(s : String) : String</code>
<code>at(i : Integer) : String</code>
<code>size() : Integer</code>
<code>concat(s : String) : String</code>
<code>substring(lower : Integer , upper : Integer) : String</code>
<code>toInteger() : Integer</code>
<code>toReal() : Real</code>
<code>toUpperCase() : String</code>
<code>toLowerCase() : String</code>
<code>indexOf(s : String) : Integer</code>
<code>equalsIgnoreCase(s : String) : Integer</code>
<code>characters() : Sequence</code>
<code>toBoolean() : Boolean</code>

Undefinierte Werte in OCL (OclVoid)

- Die Berechnung eines OCL-Teilausdruckes kann u.U. zu einen undefinierten Wert (**OclVoid**) führen
- Vergleichbar mit *null* in SQL oder Java
- Test auf undefinierten Wert mit

oclIsUndefined() : Boolean

-- true falls das Objekt undefiniert (*null*) ist,
-- ansonsten false

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

Ungültige Werte in OCL (OclInvalid)

- Vergleichbar mit Exceptions in Java
 - Methodenaufrufe auf *null* resultieren in einer NullPointerException.
- **oclIsInvalid(): Boolean**
 - true falls das Objekt ungültig (*invalid*) ist,
 - ansonsten false

Vierwertige Logik in OCL 2.3

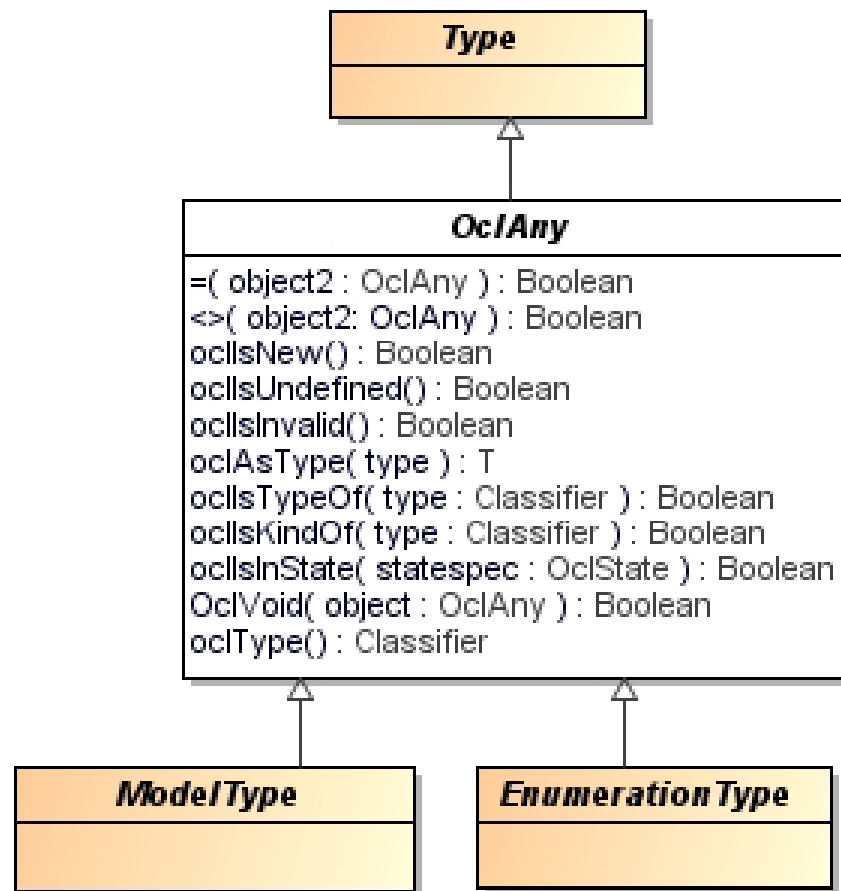
a	b	not a	a or b	a and b	a implies b	a xor b
false	false	true	false	false	true	false
false	true	true	true	false	true	true
false	null	true	invalid	false	true	invalid
false	invalid	true	invalid	false	true	invalid
true	false	false	true	false	false	true
true	true	false	true	true	true	false
true	null	false	true	invalid	invalid	invalid
true	invalid	false	true	invalid	invalid	invalid
null	false	invalid	invalid	false	invalid	invalid
null	true	invalid	true	invalid	invalid	invalid
null	null	invalid	invalid	invalid	invalid	invalid
null	invalid	invalid	invalid	invalid	invalid	invalid
invalid	false	invalid	invalid	false	invalid	invalid
invalid	true	invalid	true	invalid	invalid	invalid
invalid	null	invalid	invalid	invalid	invalid	invalid
invalid	invalid	invalid	invalid	invalid	invalid	invalid

Collection Types

Eigenschaften	<i>IsOrdered</i>	<i>not IsOrdered</i>
<i>IsUnique</i>	OrderedSet	Set
<i>not IsUnique</i>	Sequence	Bag

- Collections können undefinierte Werte (null), aber keine ungültigen Werte (invalid) enthalten.
- Falls eine Collection einen ungültigen Wert enthält, wird sie selber ungültig.

Nutzerdefinierte Typen



Modelltyp

- Nutzerdefinierte Klasse
- 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“)

OCL-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.

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.

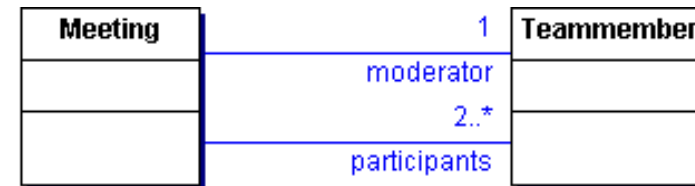
Postconditions

- 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

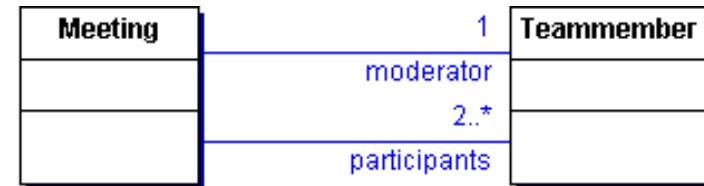
z.B. moderator

Navigation von Meeting: Typ Teammember

context Meeting

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

Navigationsausdrücke - Beispiele



Collection

- Z.B. `participants` Navigation von `Meeting` ist vom Typ `Set (Teammember)`
- Operationen auf Collections 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 Collections (1)

- 22 Operationen mit unterschiedlicher Semantik in Abhängigkeit vom Collection-Typ, 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()`)

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 }
```

Operationen auf Collections (2)

Vordefinierte Iteratoren auf allen Collection-Typen, z.B.

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

Weitere Beispiele für Collection-Operationen (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 Collection-Operationen (2)

- Weitere Nachbedingungen (Operation `select()`):

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

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

allInstances()-Operation

- Erlaubt für
 - Nutzerdefinierte Typen erlaubt, z.B. **Person.allInstances()**
 - Boolean, OCLVoid und OCLInvalid
- Für unendliche Mengen von Typen nicht erlaubt, z.B. **Integer.allInstances()**

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
```

WEITERE OCL-ANWEISUNGEN

Wiederverwendbare Ausdrücke (Definition)

- 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))
```

Anfrageoperationen (Operation Body)

- 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)

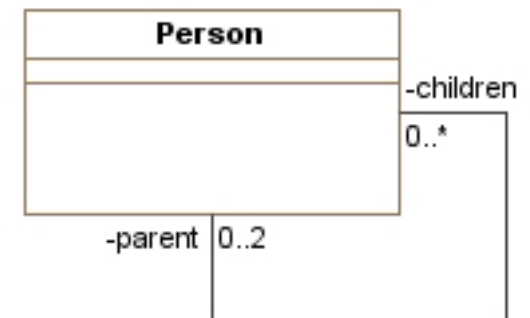
body: meetings->collect(title)

Transitive Hülle

- Wird sehr oft benötigt
 - Neu in OCL 2.3
 - Transitive Closure-Operator closure()
 - Definiert als Iterator
- ```
source->closure(expression)
```
- Problem ist effiziente Implementation

```
context Person::allDescendants():Set(Person)
body: self.parents -> closure(children)
```

```
context Person::allAncestors():Set(Person)
body: self->OrderedSet()->closure(parents)
```



## Anfangswerte (Initial Value)

### 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!

---

## Abgeleitete Attribute und Assoziationen (Derived Value)

- 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))
```

---

## Zusammenfassung von OCL-Anweisungen zu Packages

```
package MeetingExample
```

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

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

```
..
```

```
endpackage
```

---

# ANWENDUNGSFÄLLE FÜR OCL

---

## OCL in weiteren Modellen

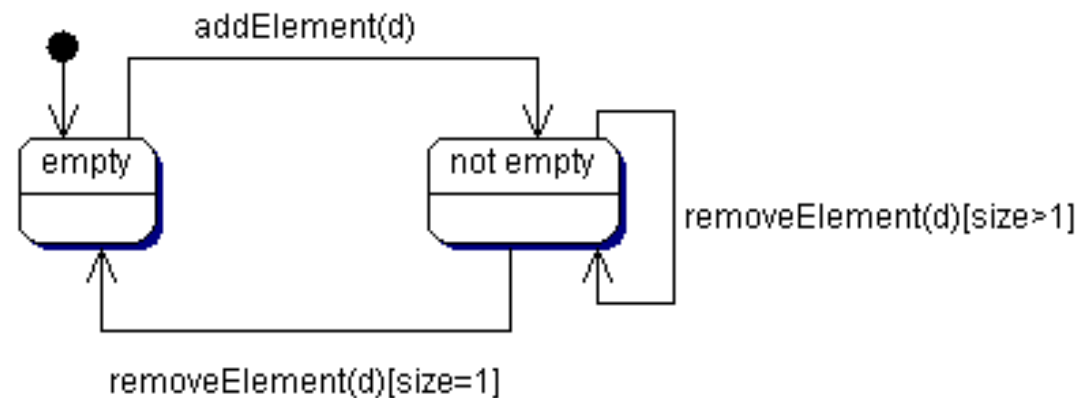
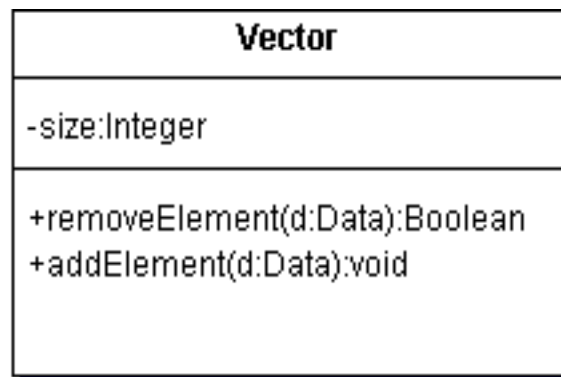
- Zustandsdiagramm (Guards, Pre- und Post-Conditions)
- Sequenzdiagramm
- Aktivitätsdiagramm
- Anwendungsfalldiagramm
- Komponentendiagramm



## 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)

```

## 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 Classifier inv SingleInheritance:
```

```
self.generalization->size() <= 1
```



```
-- Java-spezifisch: kein mehrfache Vererbung
```

---

---

# OCL TOOLS

# Dresden OCL ([dresden-ocl.org/](http://dresden-ocl.org/))

|                                                                                   | UML 0.8                             | UML 0.9 | UML 1.1                              | UML 1.2 | UML 1.3                          | UML 1.4 | UML 1.5                                                                       | UML 2.0 | UML 2.1.1<br>UML 2.1.2                           | UML 2.2 | UML 2.3                                                                      |                    |      |      |      |      |
|-----------------------------------------------------------------------------------|-------------------------------------|---------|--------------------------------------|---------|----------------------------------|---------|-------------------------------------------------------------------------------|---------|--------------------------------------------------|---------|------------------------------------------------------------------------------|--------------------|------|------|------|------|
|  | OCL (IBM)                           |         | OCL 1.1                              |         |                                  |         |                                                                               |         | OCL 2.0                                          |         | OCL 2.2                                                                      |                    |      |      |      |      |
|                                                                                   | 1995                                | 1996    | 1997                                 | 1998    | 1999                             | 2000    | 2001                                                                          | 2002    | 2003                                             | 2004    | 2005                                                                         | 2006               | 2007 | 2008 | 2009 | 2010 |
|  | <b>Dresden OCL Toolkit</b>          |         |                                      |         | <b>Dresden OCL2 Toolkit</b>      |         |                                                                               |         | <b>Dresden OCL2 for Eclipse</b>                  |         |                                                                              | <b>Dresden OCL</b> |      |      |      |      |
| <b>Design, Implementation and Refactoring</b>                                     | Frank Finger<br>Ralf Wiebicke       |         | Steffen Zschaler<br>Sten Löcher      |         | Stefan Ocke<br>Nikolai Krambrock |         | Florian Heidenreich<br>Ansgar Konermann<br>Christian Wende<br>Kai Uwe-Gartner |         | Matthias Bräuer<br>Ronny Brandt<br>Nils Thieme   |         | Claas Wilke<br>Michael Thiele<br>Björn Freitag<br>Lars Schütze<br>Ronny Marx |                    |      |      |      |      |
| <b>Theoretical Foundations, Analysis and Case Studies</b>                         | Andreas Schmidt<br>Heinrich Hußmann |         | Birgit Demuth<br>Sven Obermaier      |         | Andrea Kling                     |         | Katrin Eisenreich<br>Sebastian Käßler                                         |         | Birgit Demuth<br>Martin Krebs<br>Christian Wende |         |                                                                              |                    |      |      |      |      |
| <b>Adaptation to and Coupling with other Tools</b>                                | Frank Finger                        |         |                                      |         | Christian Nil                    |         | Irina Soudnik<br>Mirko Stölzel                                                |         | Claas Wilke                                      |         |                                                                              |                    |      |      |      |      |
| <b>OMG Standardization Activities</b>                                             |                                     |         | Heinrich Hußmann<br>Steffen Zschaler |         |                                  |         |                                                                               |         |                                                  |         |                                                                              |                    |      |      |      |      |

Dresden OCL - tudresden.oc120.pivot.examples.simple/constraints/allConstraints.oc1 - Eclipse Platform

File Edit Refactor Dresden OCL Navigate Search Project Run Window Help

Quick Access

Project Explorer Navigator

- tudresden.oc120.pivot.examples.simple
  - src
  - JRE System Library [JavaSE-1.6]
  - Plug-in Dependencies
  - constraints
    - allConstraints.oc1
    - defs.oc1
    - invariants.oc1
    - postconditions.oc1
    - preconditions.oc1
  - META-INF
  - model
    - simple.javamodel
    - simple.uml
  - tudresden
    - README

allConstraints.oc1

```

/* @model{../model/simple.uml} */
package tudresden::oc120::pivot::examples::simple

-- The age of Person can not be negative.
context Person
inv: age >= 0

-- Students should be 16 or older.
context Student
inv: age > 16

-- Professors should be at Least 30.
context Professor
inv: not (age < 30)

-- Returns the age of a Person.
context Person
def: getAge(): Integer = age

-- Before returning the age, the age must be defined.
context Person::getAge()
pre: not age.oc1IsUndefined()

```

Outline

- Package Declaration With Namespace CS
  - context
    - Person
      - INV inv
        - >=
          - x age
            - 42 0
  - context
  - context
  - context
  - context getAge
  - context getAge
  - tudresden

Model Bro... Model Inst...

tudresden

- oc120
  - pivot
    - examples
      - simple
        - Person -> OclAny
          - getAge(): Integer
            - owner: ModelProviderClass
            - name: String
            - age: Integer
          - Student -> Person
          - Professor -> Person
          - ModelProviderClass

Properties Dresden OCL Metrics V... Dresden OCL Interpreter Dresden OCL Tracer Vi... Error Log

| Metric                                     | Result |
|--------------------------------------------|--------|
| Number of constraints                      | x = 6  |
| Number of constraints of kind invariant    | x = 3  |
| Number of constraints of kind definition   | x = 1  |
| Number of constraints of kind precondition | x = 1  |

Writable Insert 6 : 6

Properties Dresden OCL Metrics V... Dresden OCL Interpreter Dresden OCL Tracer Vi... Error Log

| Object                                   | Constraint                                      | Result                |
|------------------------------------------|-------------------------------------------------|-----------------------|
| JavaModelInstanceObject[Person[nam...]   | context getAge(): pre: not age.oclIsUndefined() | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Professor[na...] | inv: not (age < 30)                             | JavaOclBoolean[false] |
| JavaModelInstanceObject[Professor[na...] | context getAge(): pre: not age.oclIsUndefined() | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Professor[na...] | inv: age >= 0                                   | JavaOclBoolean[false] |
| JavaModelInstanceObject[Person[nam...]   | inv: age >= 0                                   | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Professor[na...] | def: getAge(): Integer = age                    | JavaOclInteger[-42]   |
| JavaModelInstanceObject[Person[nam...]   | context getAge(): post: result = age            | JavaOclBoolean[false] |
| JavaModelInstanceObject[Person[nam...]   | def: getAge(): Integer = age                    | JavaOclInteger[25]    |
| JavaModelInstanceObject[Professor[na...] | context getAge(): post: result = age            | JavaOclBoolean[false] |
| JavaModelInstanceObject[Student[nam...]  | inv: age > 16                                   | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Student[nam...]  | inv: age >= 0                                   | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Student[nam...]  | def: getAge(): Integer = age                    | JavaOclInteger[23]    |
| JavaModelInstanceObject[Student[nam...]  | context getAge(): pre: not age.oclIsUndefined() | JavaOclBoolean[true]  |
| JavaModelInstanceObject[Student[nam...]  | context getAge(): post: result = age            | JavaOclBoolean[false] |





---

## XMI/UML Import für Dresden OCL

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

---

## Einige weitere UML/OCL Tools

- **MagicDraw** Enterprise Edition v17 (mit Dresden OCL2 Toolkit 😊)
  - Evaluations-Lizenz
- Borland **Together** 2008 (OCL/QVT)
- **Eclipse MDT/OCL** for Ecore 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 v3.mdzip [D:\Lehre\Demuth\Ocl Vorlesungen\OCL SS 2009\MagicDraw Samples\]

File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help

D:\Le... Lecture Samples v3.mdzip

Containment

Containment

Data

- Classes
- Instances
- ModelingGuideline
- SingleInheritance=self.generalization->size()<=1
- Object verification
- inv1=self.participants->size()>=2
- inv2=end>start
- OCLByExample1
- Code engineering sets

OCLByExample1

Common

- Note
- Text Box
- Anchor
- Containment
- Dependency
- Image Shape
- Separator
- Class Diagram
- Class
- Interface
- Package
- Generalizat...
- Association
- Aggregation
- Composition
- Interface ...
- Usage
- Abstraction
- Collaboration
- Use Case Diagram
- Implementation Di...
- Composite Struct...
- Information Flows
- Profiling Mechanism

Meeting

(self.participants->size()>=2,  
end>start)

-start : Integer = 12  
-end : Integer = 10

Teammember

TM1

STMeeting : Meeting

end = 10  
start = 12

Chef : Teammember

Sebastian : Teammember

Birgit : Teammember

Katrin : Teammember

Claas : Teammember

DresdenOCLMeeting : Meeting

end = 16  
start = 14

Validation Results

Validation Results

Filter: >=debug <ALL> <ALL> Not Ignored

| Element       | Severity | Abbreviation      | Error Message                                                                | Is Ignored |
|---------------|----------|-------------------|------------------------------------------------------------------------------|------------|
| TM1 [Classes] | error    | SingleInheritance | Element contains multiple generalizations - only 1 generalization is allowed |            |

No symbol at (254, 440)

Start Total Commander 6.0... Posteingang von birgi... LEO Ergebnisse für "2... generalization - Wind... MagicDraw UML 16.5 ... DE 18:26

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

OCLByExample1

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 "2..*" -- "*" Teammember : -participants
 Meeting "1" -- "1" Teammember : DresdenOCLMeeting
 Meeting "1" -- "1" Teammember : STMeeting
 Meeting "1" -- "1" Teammember : STMeeting

```

Meeting

{self.participants ->size()>= 2, end>start}

-start : Integer = 12  
-end : Integer = 10

Teammember

**STMeeting : Meeting**

end = 10  
start = 12

**DresdenOCLMeeting : Meeting**

end = 16  
start = 14

**Chef : Teammember**

**Birgit : Teammember**

**Katrin : Teammember**

**Sebastian : Teammember**

**Clas : Teammember**

Validation Results

Validation Results

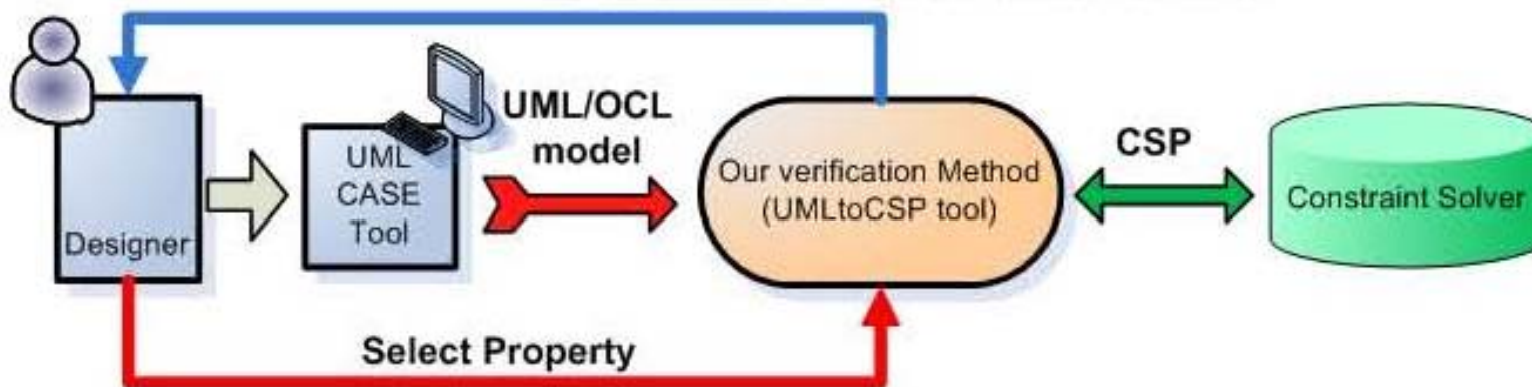
Filter: >=debug <ALL> <ALL> N...

| 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      |           |

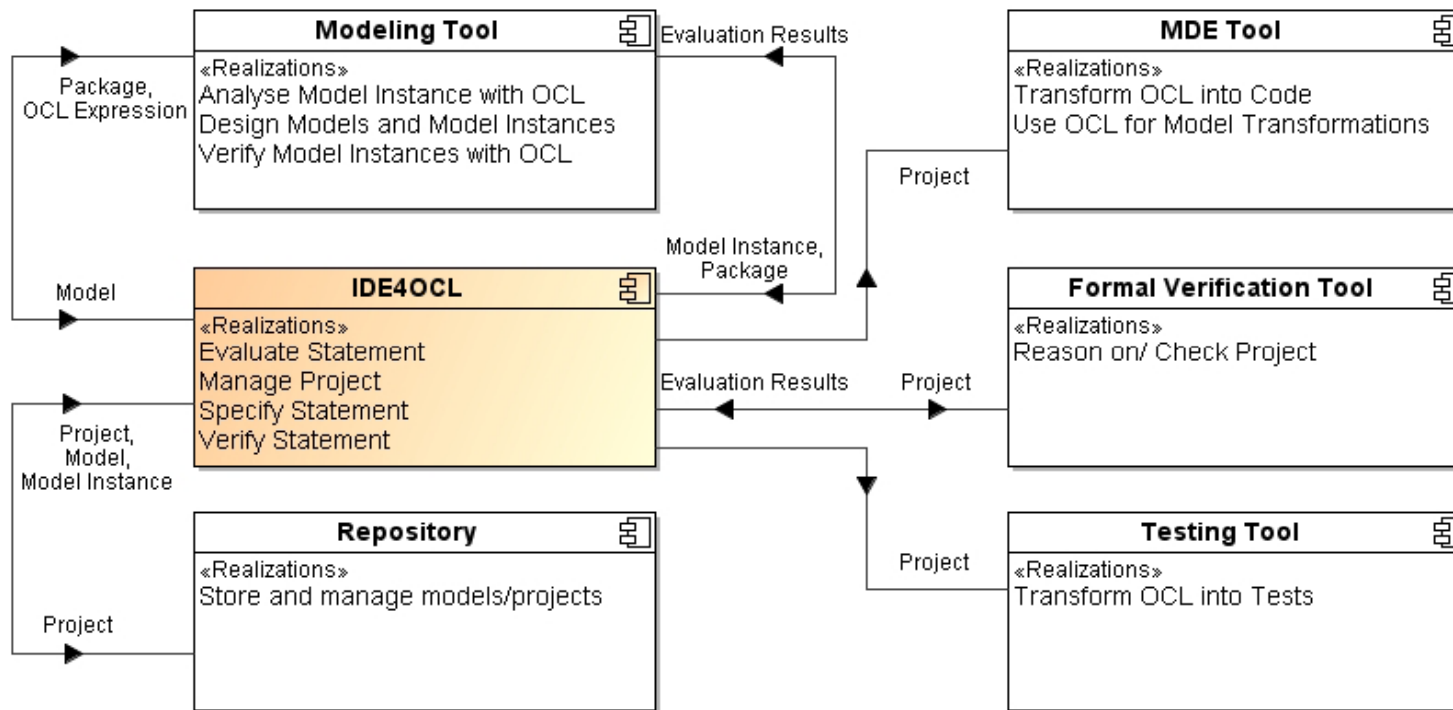
Ready

## Formale Verifikation von OCL-Constraints

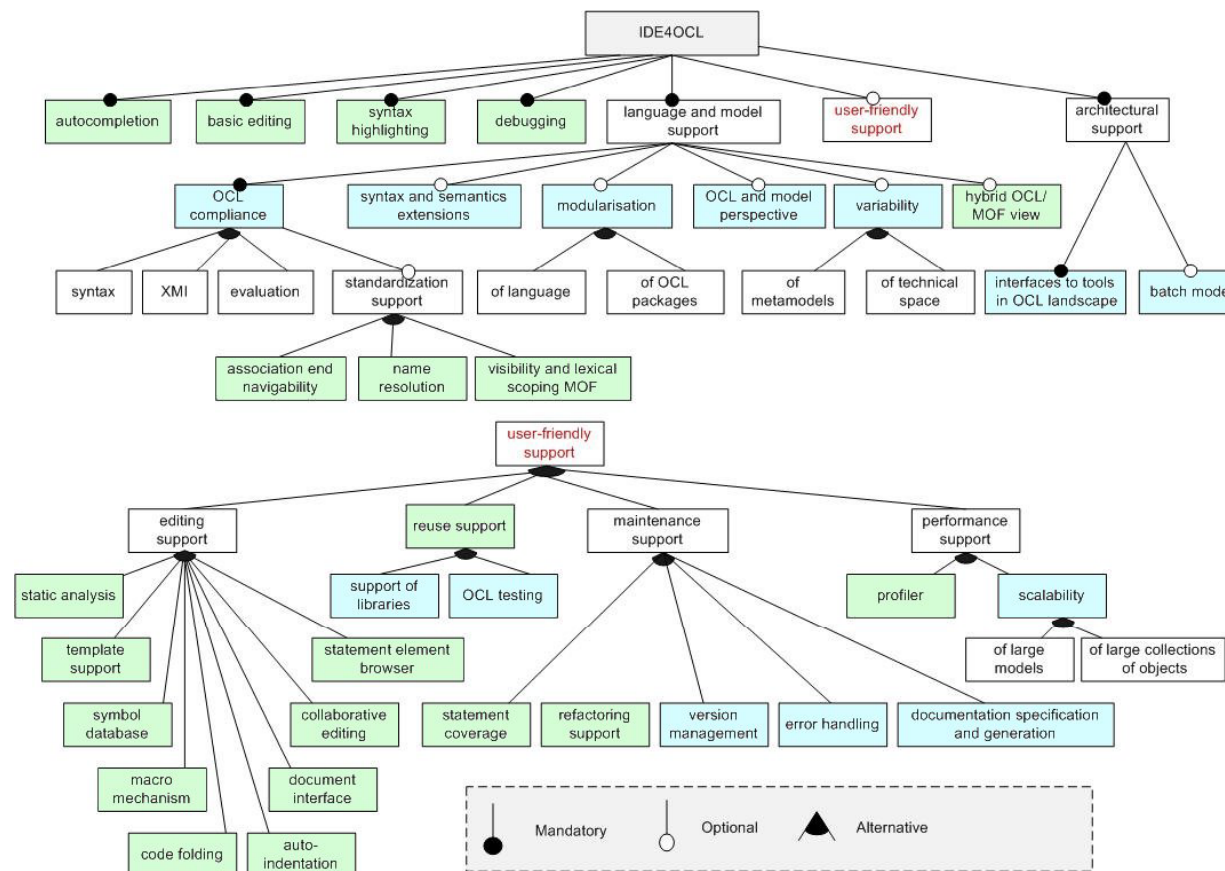
- Für **inkonsistente** Spezifikationen (Kombination von Constraints, die sich widersprechen) gibt es wenig Unterstützung, diese aufzufinden.
  - **UMLtoCSP** (Transformation eines UML/OCL-Modells in ein Constraint Satisfaction Problem(CSP)), Jordi Cabot et al
    - <http://gres.uoc.edu/UMLtoCSP/>



# The OCL Tool Landscape



# A Feature Model for an IDE4OCL





## **Kontakt:**

[birgit.demuth@tu-dresden.de](mailto:birgit.demuth@tu-dresden.de)

<http://dresden-ocl.org/>

# FRAGEN?

# INTERESSE AN DA/BA/GB?