TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut SMT, Lehrstuhl Softwaretechnologie

Einführung in OCL (Object Constraint Language)

Dr. Birgit Demuth

Warum OCL?

Eine formale Sprache?

Was wollen wir lernen?

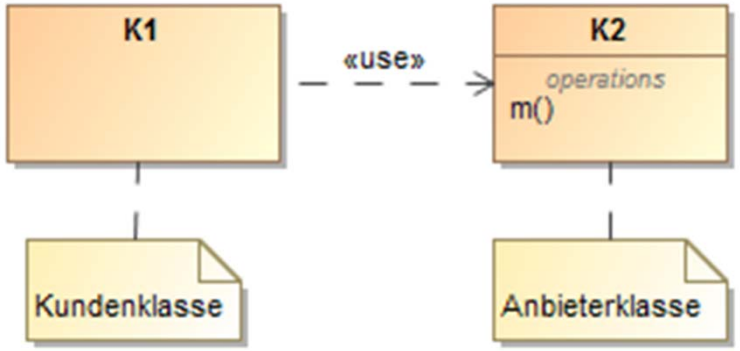
- Einführung in die Thematik (Vertragsmodell, Zusicherungen, Überblick OCL)
- Sprachkonzepte und „OCL by Example“
  - OCL-Typen und OCL-Ausdrücke
  - Weitere OCL-Anweisungen
- Anwendungsfälle für OCL und Diskussion
- OCL Tools

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

→ Erweiterung von Methoden oder Klassen um Code für Vor-, Nachbedingungen und Invarianten („Defensives Programmieren“)

Formulierung von Zusicherungen

- *Modellbasiert*:
  - OCL
- *In Programmiersprachen*:
  - EIFFEL
  - JASS (Java with ASSertions)
  - JML (Java Modeling Language)
  - Java (assert)
  - Spec# für C#

Beispiel `assert` *in Java:*

```
public void doSomething(String str)
{
  assert str != null;
  if (str.length > 0)
  ...
}
```

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 2.4 (2014)
- 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] Birgit Demuth (Hrsg.): Softwaretechnologie für Einsteiger. Pearson Studium, 2. geänderte Auflage, 2014

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

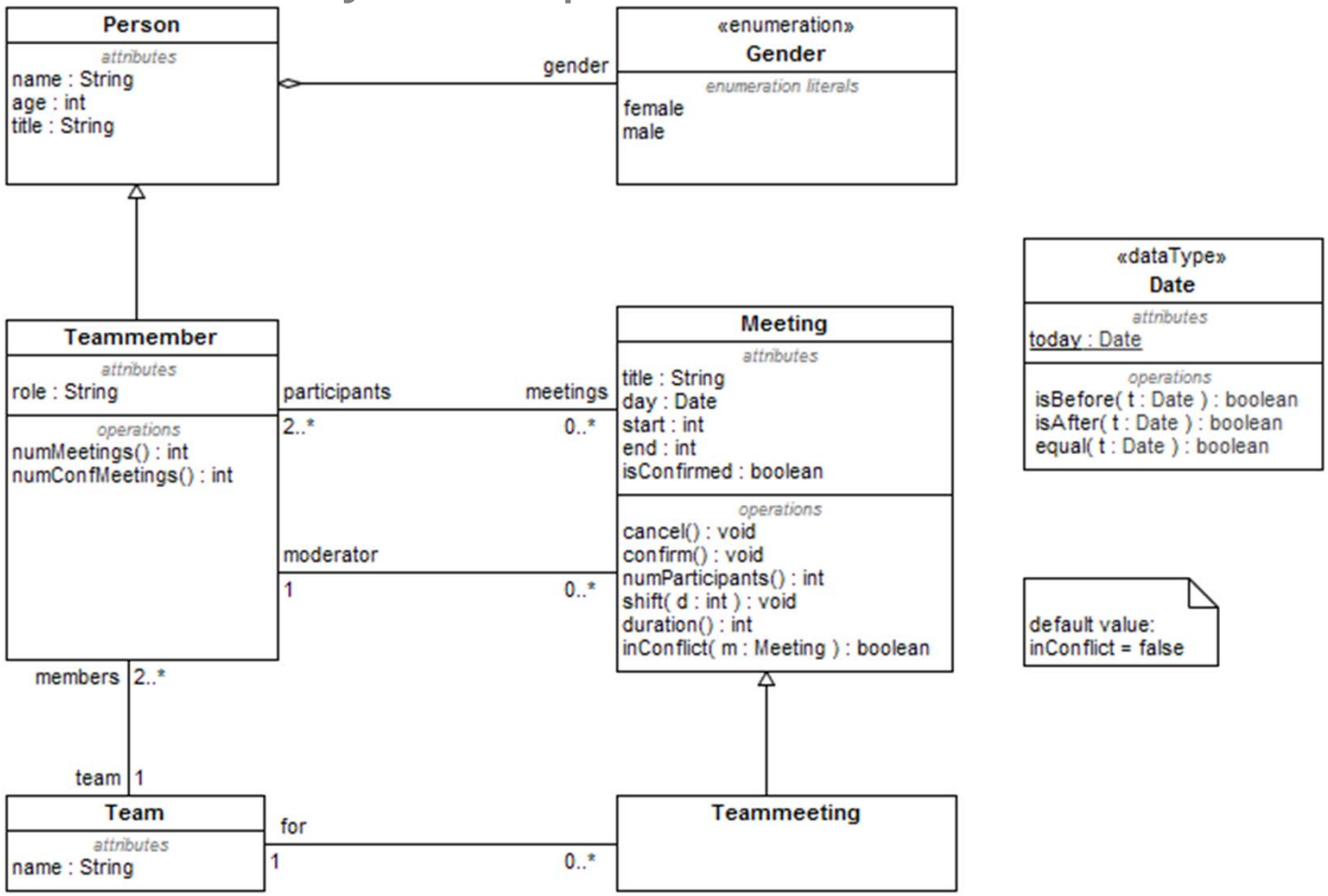
Definition

- 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

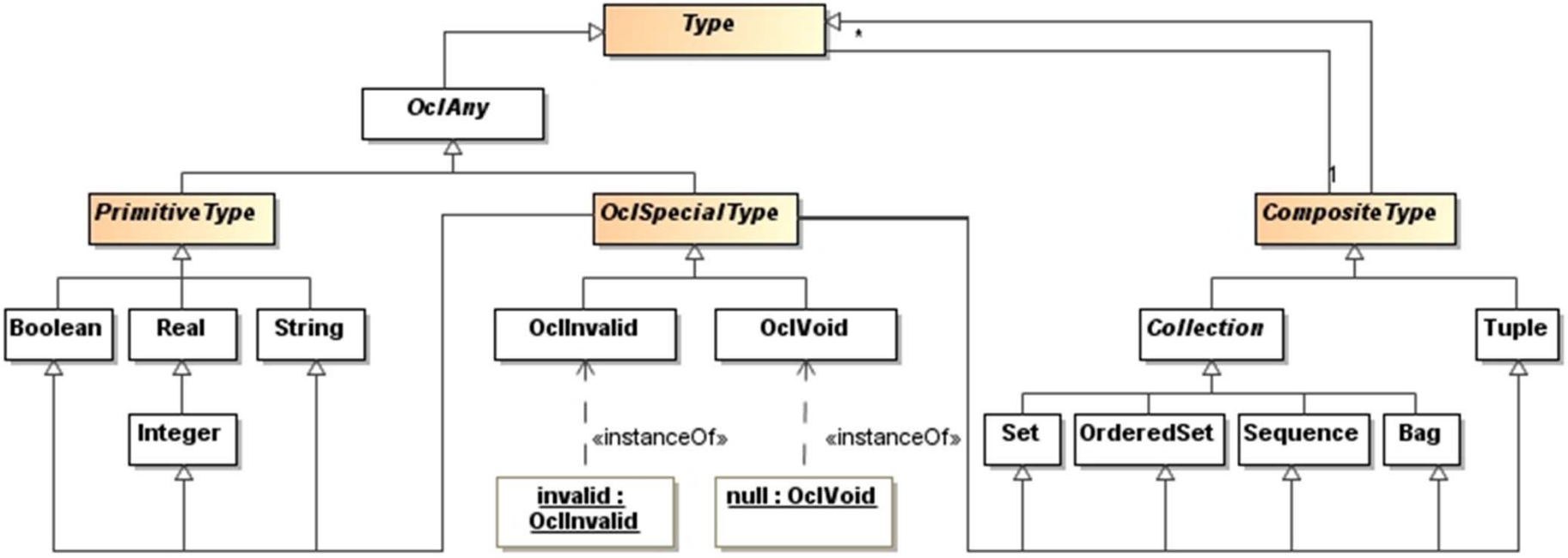
```
context Meeting::duration():Integer
post: result = self.end – self.start
```
-- *result* bezieht sich auf den Rückkehrwert der Operation

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

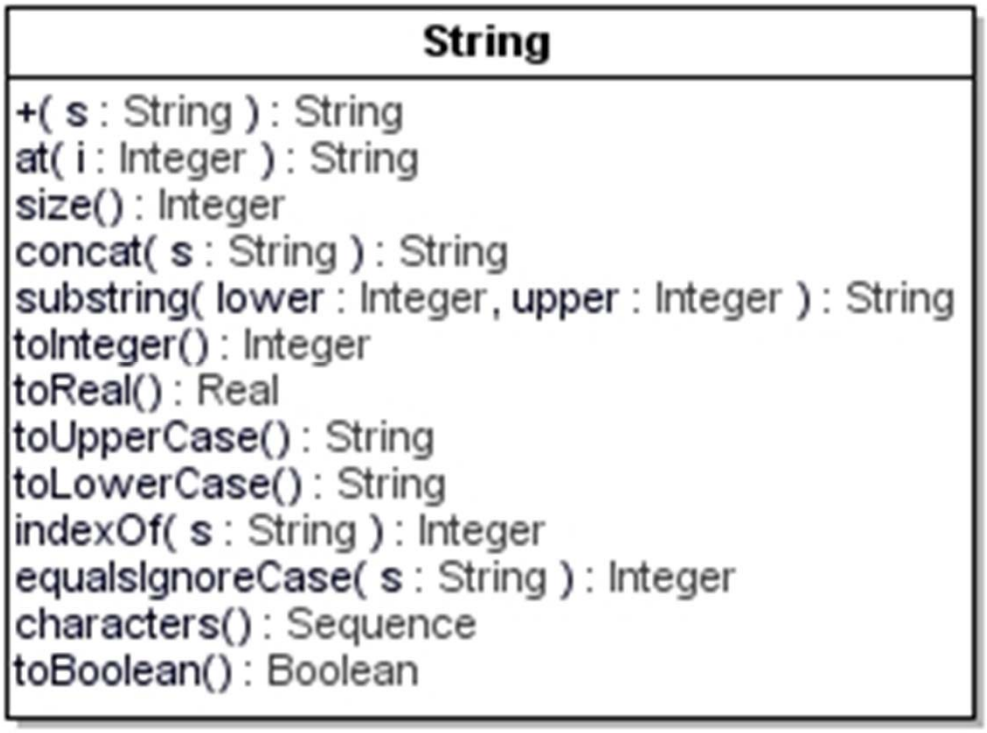
```
context Meeting::shift(d:Integer)
post: start = start@pre +d and end = end@pre + d
```
-- *start@pre* bezieht sich auf den Wert **vor** Ausführung der
-- Operation
-- *start* bezieht sich auf den Wert **nach** Ausführung der Operation
-- @pre ist nur in Postconditions erlaubt

OCL-TYPEN UND OCL-AUSDRÜCKE

Basistypen der OCL-Standardbibliothek



Standardoperationen auf Strings (Auswahl)



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 NullPointer- Exception.

- **`oclIsInvalid(): Boolean`**
  -- true falls das Objekt ungültig *(invalid)* ist,
  -- ansonsten false

Vierwertige Logik in OCL

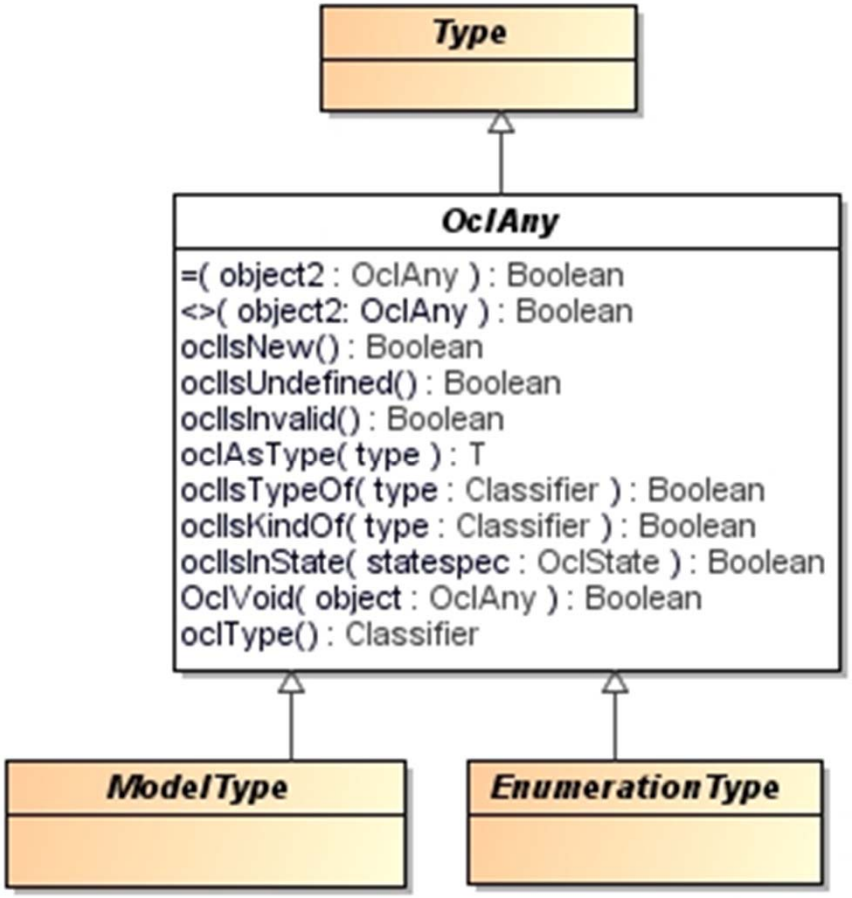
a	b	not a	a or b	a and b	a implies b	a xor b
false	**false**	**true**	**false**	**false**	**true**	**false**
false	**true**	**true**	**true**	**false**	**true**	**true**
false	null	**true**	invalid	**false**	**true**	invalid
false	invalid	**true**	invalid	**false**	**true**	invalid
true	**false**	**false**	**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	*IsOrdered*	*not IsOrdered*
IsUnique	OrderedSet	Set
not IsUnique	Sequence	Bag

- Collections können undefinerte 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 (1)

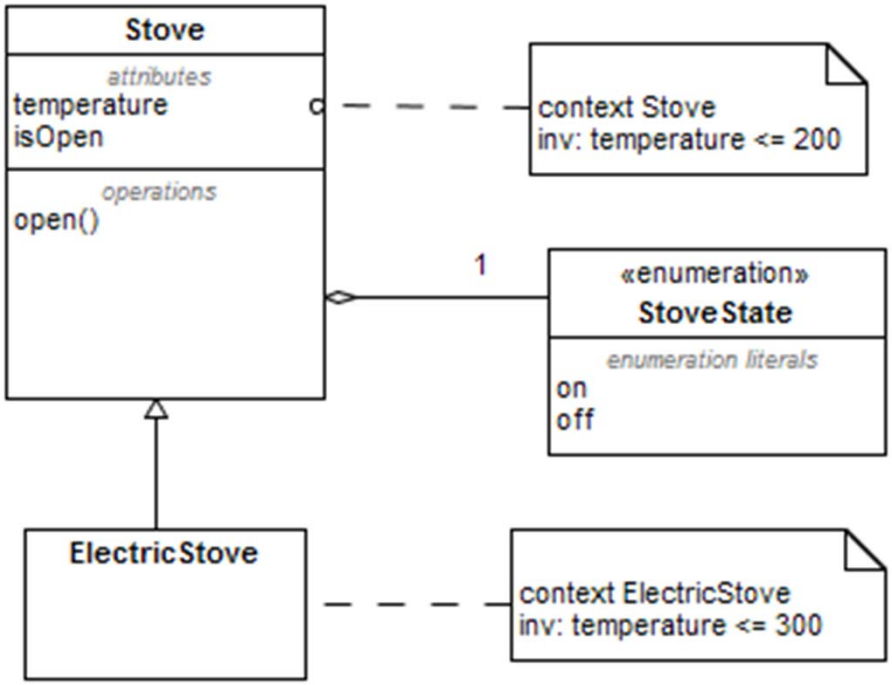
Constraints allgemein

- Constraints einer Superklasse werden von den Subklassen geerbt.

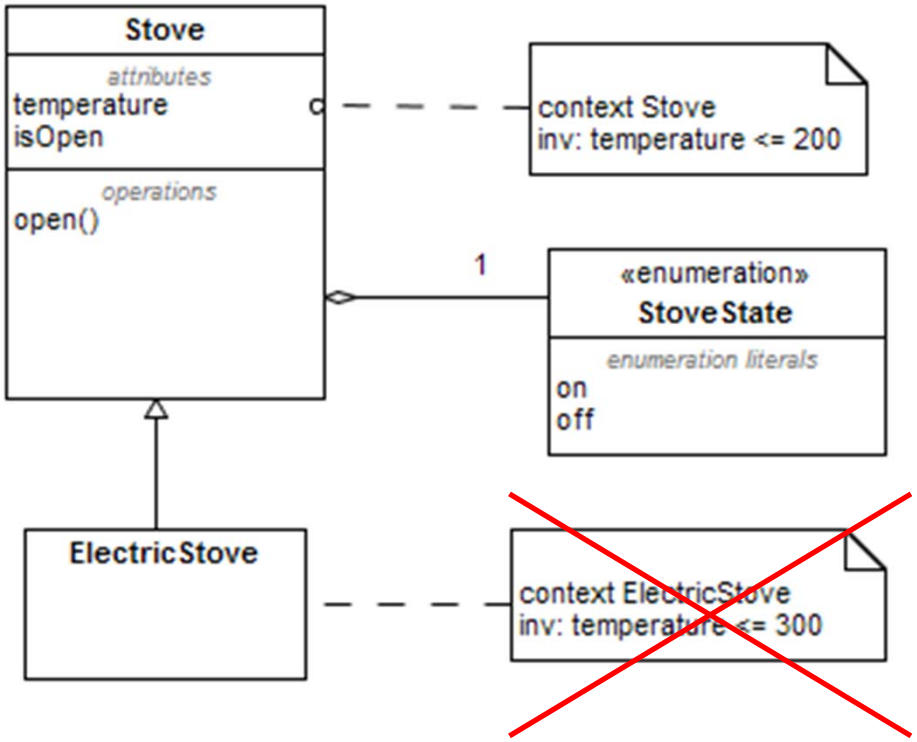
Invarianten

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

OCL Constraints und Vererbung – Beispiel (1)



OCL Constraints und Vererbung – Beispiel (1)



--Abschwächung

OCL Constraints und Vererbung (2)

Constraints allgemein

- Constraints einer Superklasse werden von den Subklassen geerbt.

Preconditions

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

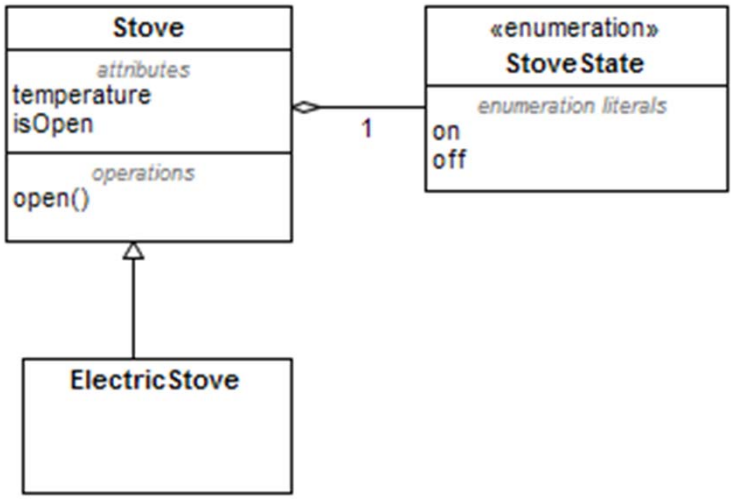
Postconditions

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

OCL Constraints und Vererbung – Beispiel (2)

```
context Stove::open()
pre : status = StoveState::off
post: status = StoveState::off
   and isOpen

context ElectricStove::open()
pre : status = StoveState::off
   and temperature <= 100
   -- Verstärkung
post: isOpen -- Abschwächung
```

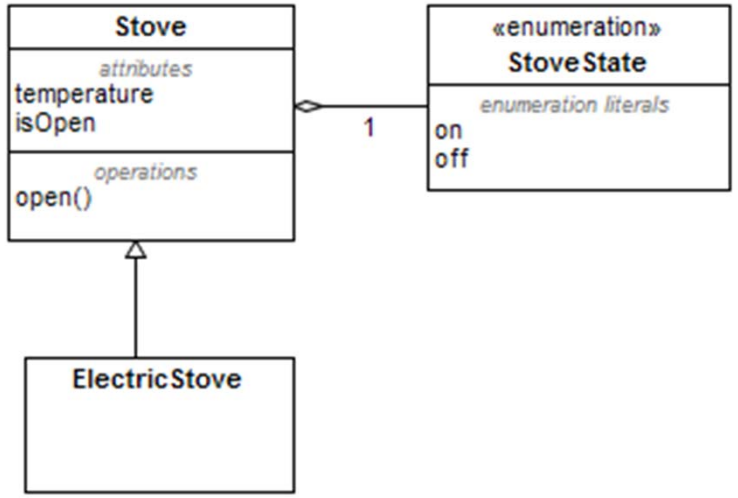


OCL Constraints und Vererbung – Beispiel (2)

```
context Stove::open()
pre : status = StoveState::off
post: status = StoveState::off and
   isOpen
```

```
context ElectricStove::open()
pre : status = StoveState::off and
   temperature <= 100
```

```
post: isOpen
```



Navigationsausdrücke

- Assoziationsenden (Rollennamen) können verwendet werden, um von einem Object im Modell/System zu einem anderen zu navigieren (**Navigation**)
- Navigation wird in OCL als Attribut 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.
    `Set{Bag{1,2,2},Bag{2}}` → `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.

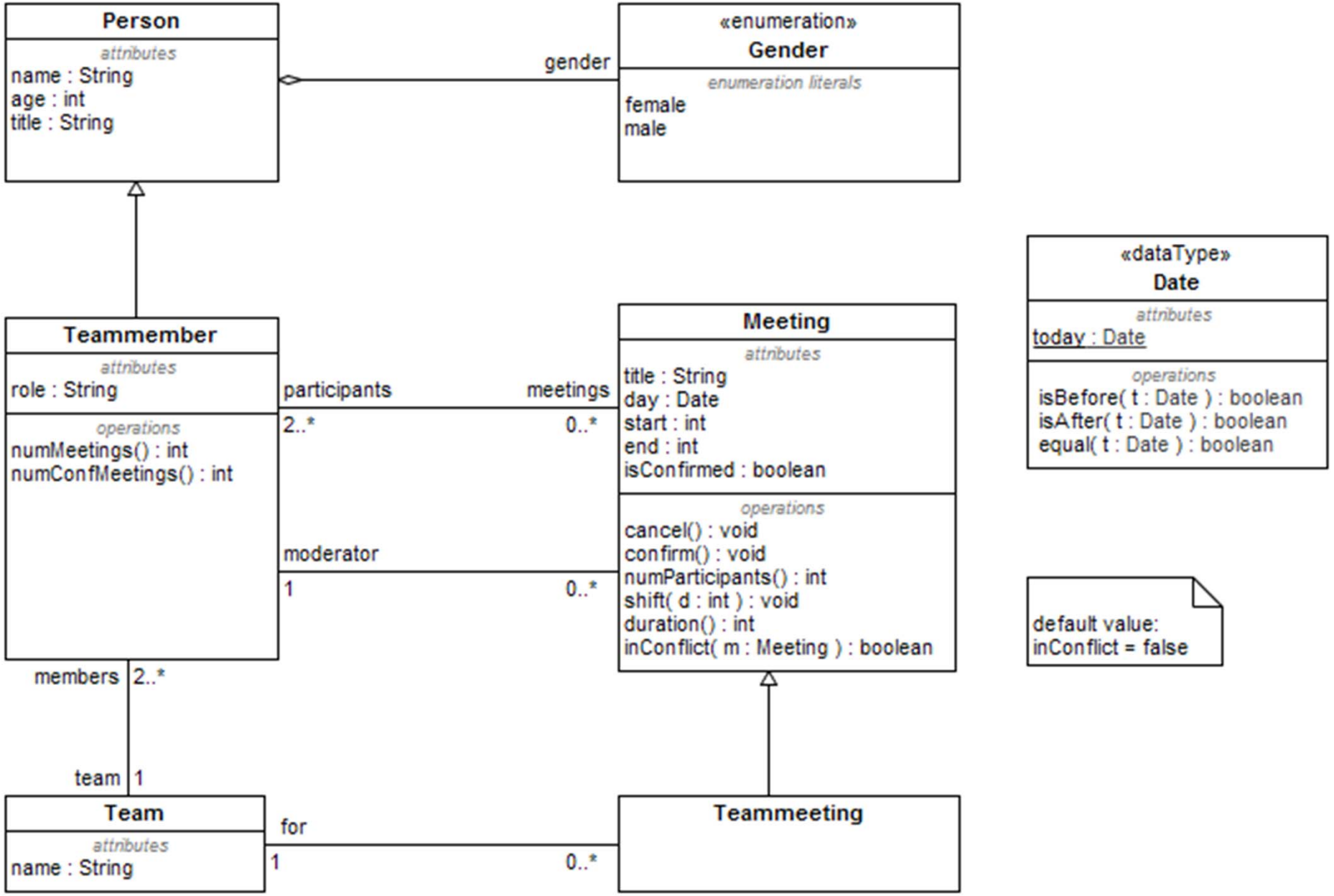
Set {1,2,3}->sum() durch

```
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)
```


Person
attributes
name : String
age : int
title : String
gender
«enumeration»
Gender
enumeration literals
female
male
«dataType»
Date
attributes
today : Date
operations
isBefore(t : Date) : boolean
isAfter(t : Date) : boolean
equal(t : Date) : boolean
Teammember
attributes
role : String
operations
numMeetings() : int
numConfMeetings() : int
participants
2..*
meetings
0..*
Meeting
attributes
title : String
day : Date
start : int
end : int
isConfirmed : boolean
operations
cancel() : void
confirm() : void
numParticipants() : int
shift(d : int) : void
duration() : int
inConflict(m : Meeting) : boolean
moderator
1
0..*
default value:
inConflict = false
members
2..*
team
1
Team
attributes
name : String
for
1
0..*
Teammeeting

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)
```

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.

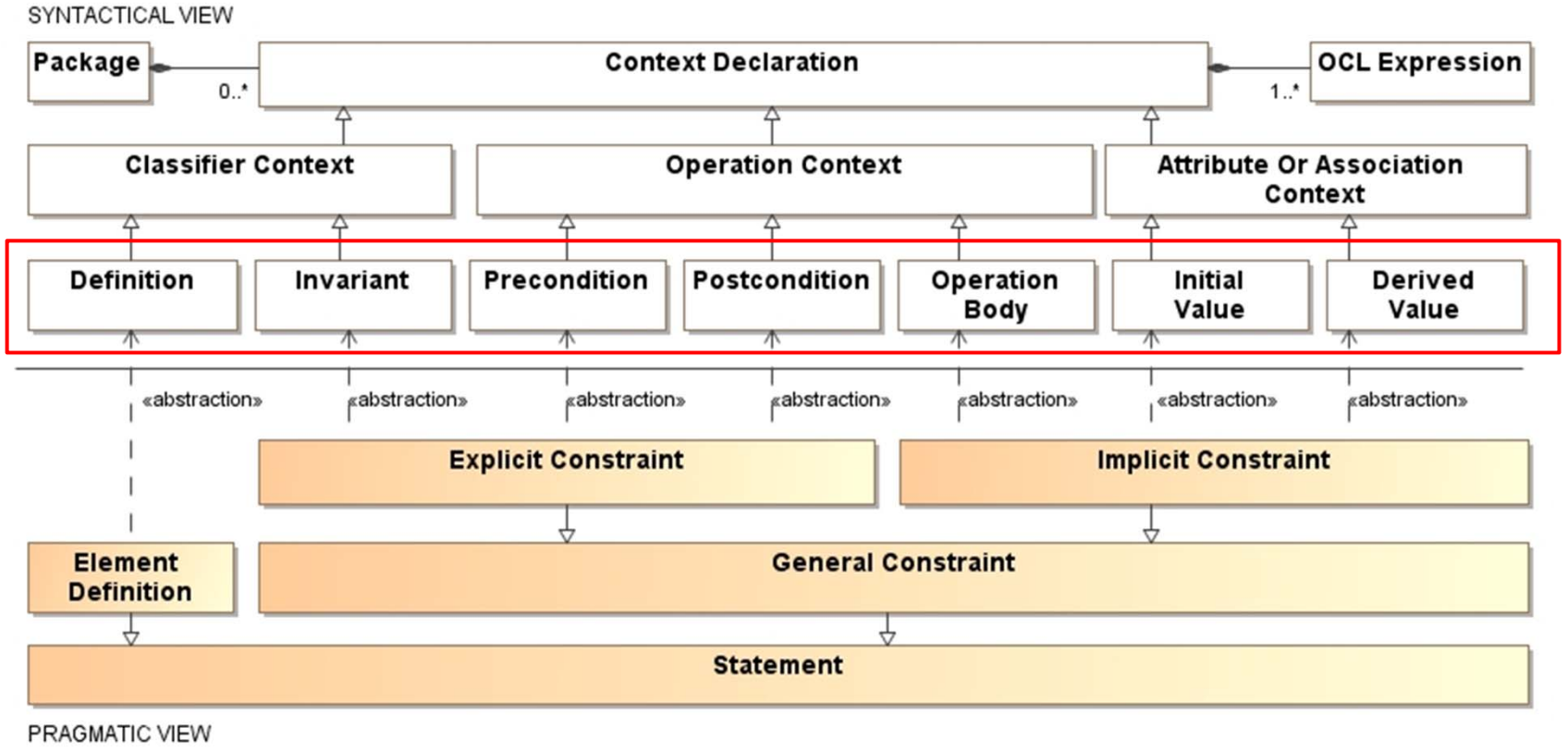
Beispiel
(Ein bestätigtes Meeting darf nicht in Konflikte mit anderen Meetings stehen.)

```
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

OCL-Metamodell aus pragmatischer Sicht

[Chimiak-Opoka, Demuth 2009]



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)
```

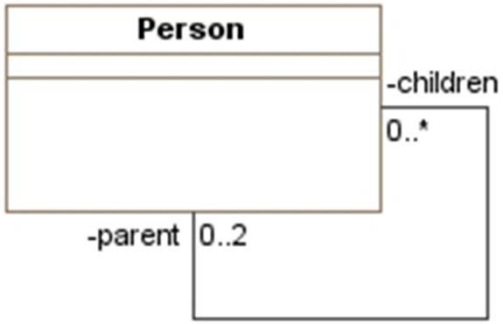
Transitive Hülle

- Wird sehr oft benötigt
- Neu seit 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::allAnchestors ():Set(Person)
body: self->OrderedSet()->closure(parents)
```



Anfangswerte (Initial Value)

Beispiele

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

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

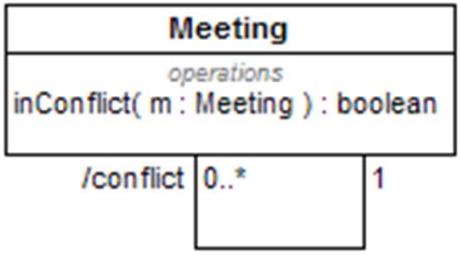
- Man beachte den Unterschied zu Invarianten und Ableitungsregeln: Ein Anfangswert muss nur zum Zeitpunkt der Erzeugung des Objektes gelten!

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)
  - 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

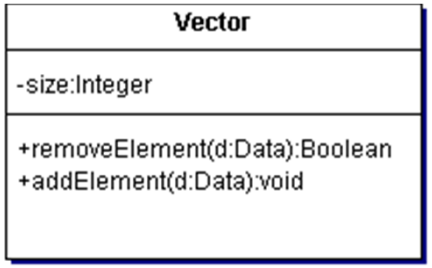
OCL in weiteren Modellen

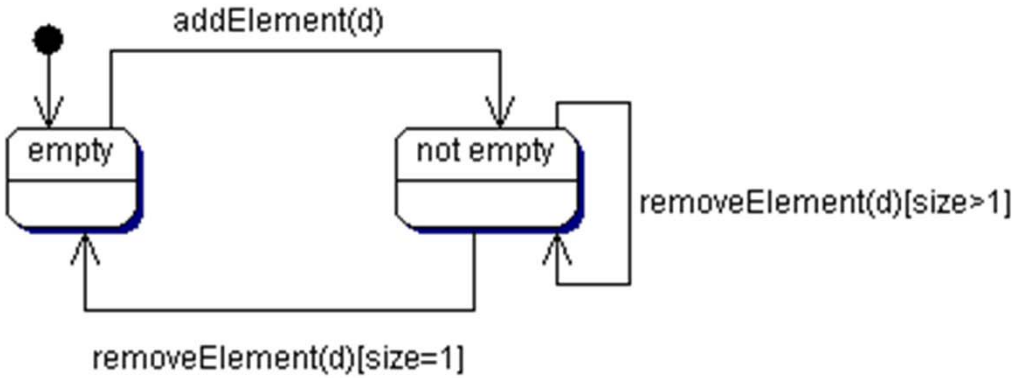
- Zustandsdiagramm (Guards, Pre- und Post-Conditions)
- Sequenzdiagramm
- Aktivitätsdiagramm
- Anwendungsfalldiagramm
- Komponentendiagramm

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)
```

Anwendungsfälle für OCL

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

MOF-Modellebene	Beispiele für die Verwendung von OCL
M2 (Metamodell)	•Spezifikation von **Well-Formedness Rules (WFRs)** in OMG-Standards •Definition von Modellierungsrichtlinien für **DSLs (Domain Specific Languages)** •Spezifikation von **Modellabbildungen**
M1 (Modell)	•Überprüfung der Konsistenz von Modellen mit dem Metamodell (→ CASE-Tool) •Evaluation von Modellierungsrichtlinien in DSL-Instanzen •Ausführung von Modellabbildungen
	•Spezifikation von **Geschäftsregeln / Constraints** •Spezifikation von **Testfällen**
M0 (Objekte)	•Evaluation von Geschäftsregeln/Constraints •Ausführung von Testfällen

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

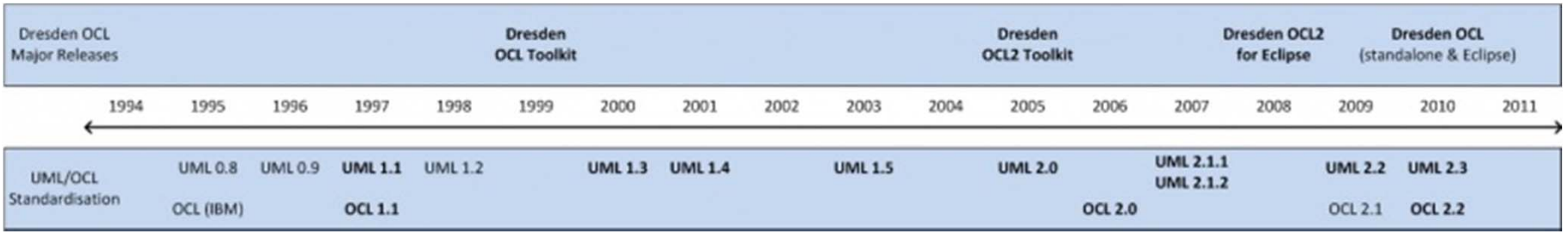
```
context Classifier inv SingleInheritance:
self.generalization->size()<= 1
-- Java-spezifisch: keine mehrfache Vererbung
```

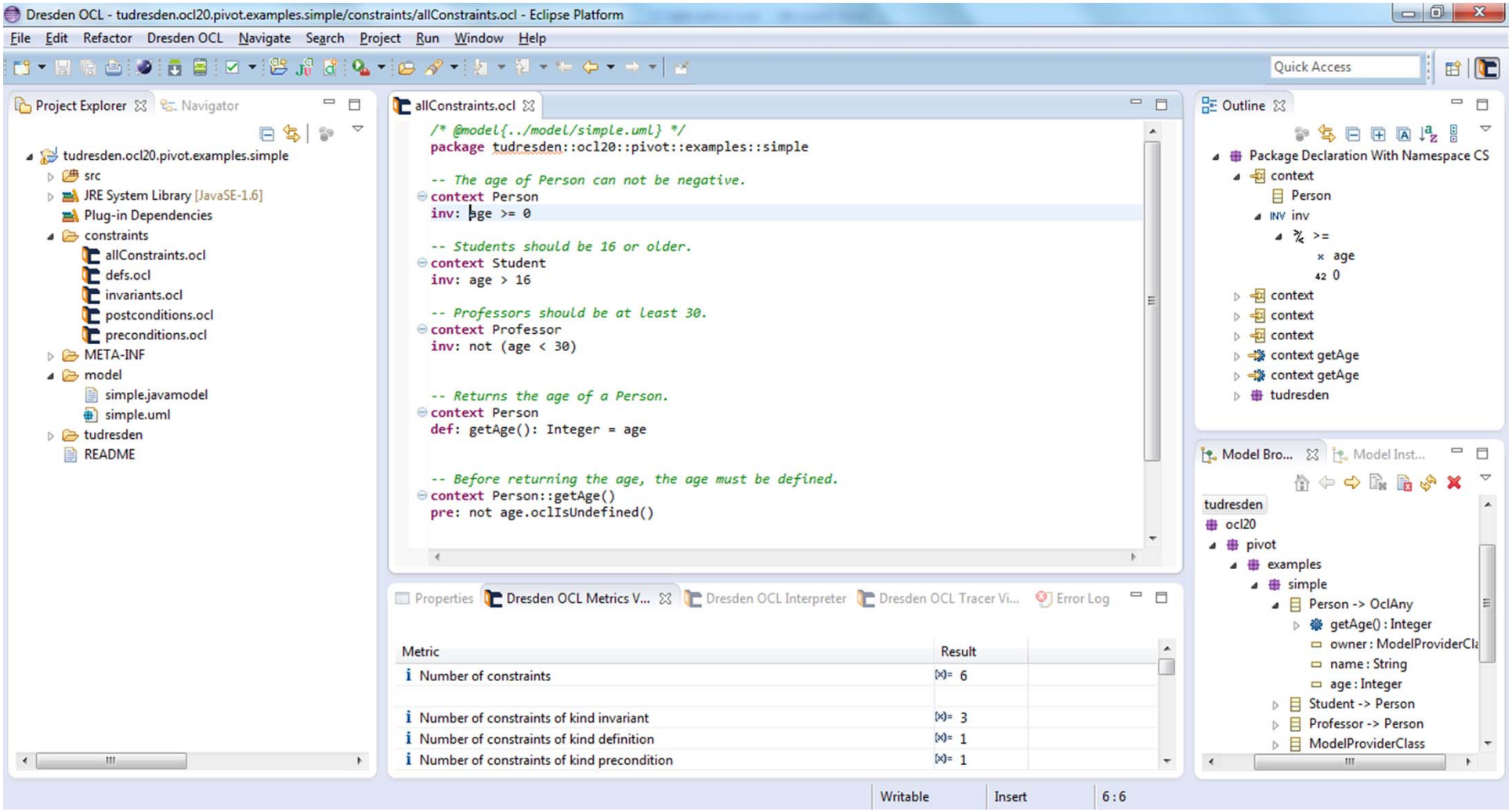
ÜBERBLICK OCL TOOLS

Dresden OCL

dresden-ocl.org/
https://github.com/dresden-ocl/

aktuell: Dresden OCL v3.3.0



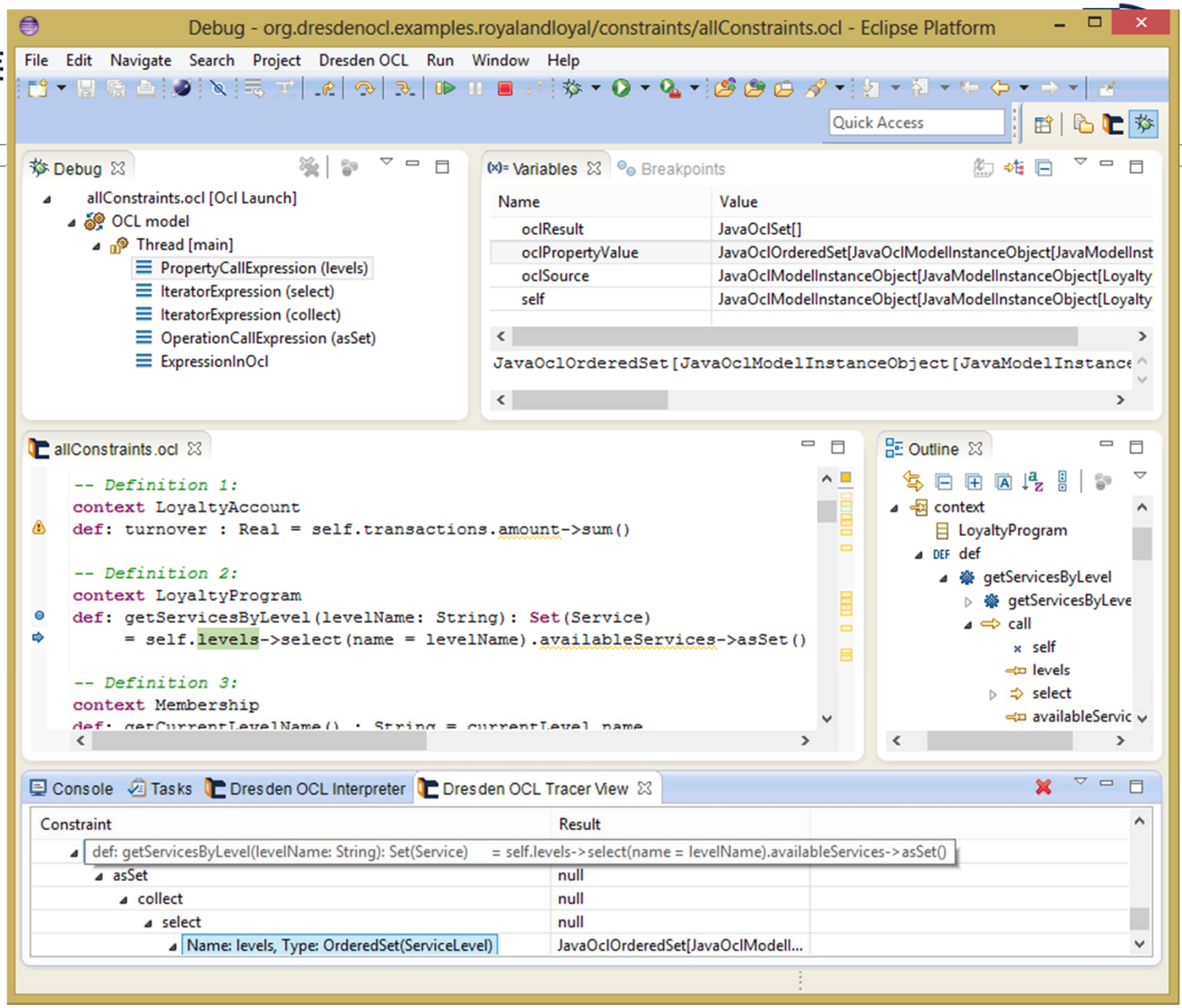

Dresden OCL - tudresden.ocl20.pivot.examples.simple/constraints/allConstraints.ocl - Eclipse Platform
File Edit Refactor Dresden OCL Navigate Search Project Run Window Help
Quick Access
Project Explorer
Navigator
tudresden.ocl20.pivot.examples.simple
src
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
constraints
allConstraints.ocl
defs.ocl
invariants.ocl
postconditions.ocl
preconditions.ocl
META-INF
model
simple.javamodel
simple.uml
tudresden
README
allConstraints.ocl
/* @model{../model/simple.uml} */
package tudresden::ocl20::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.oclIsUndefined()
Outline
Package Declaration With Namespace CS
context
Person
inv
>=
age
0
context
context
context
context getAge
context getAge
tudresden
Model Bro...
Model Inst...
tudresden
ocl20
pivot
examples
simple
Person -> OclAny
getAge() : Integer
owner : ModelProviderCla
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
6
Number of constraints of kind invariant
3
Number of constraints of kind definition
1
Number of constraints of kind precondition
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]

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden OCL Debugger



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)

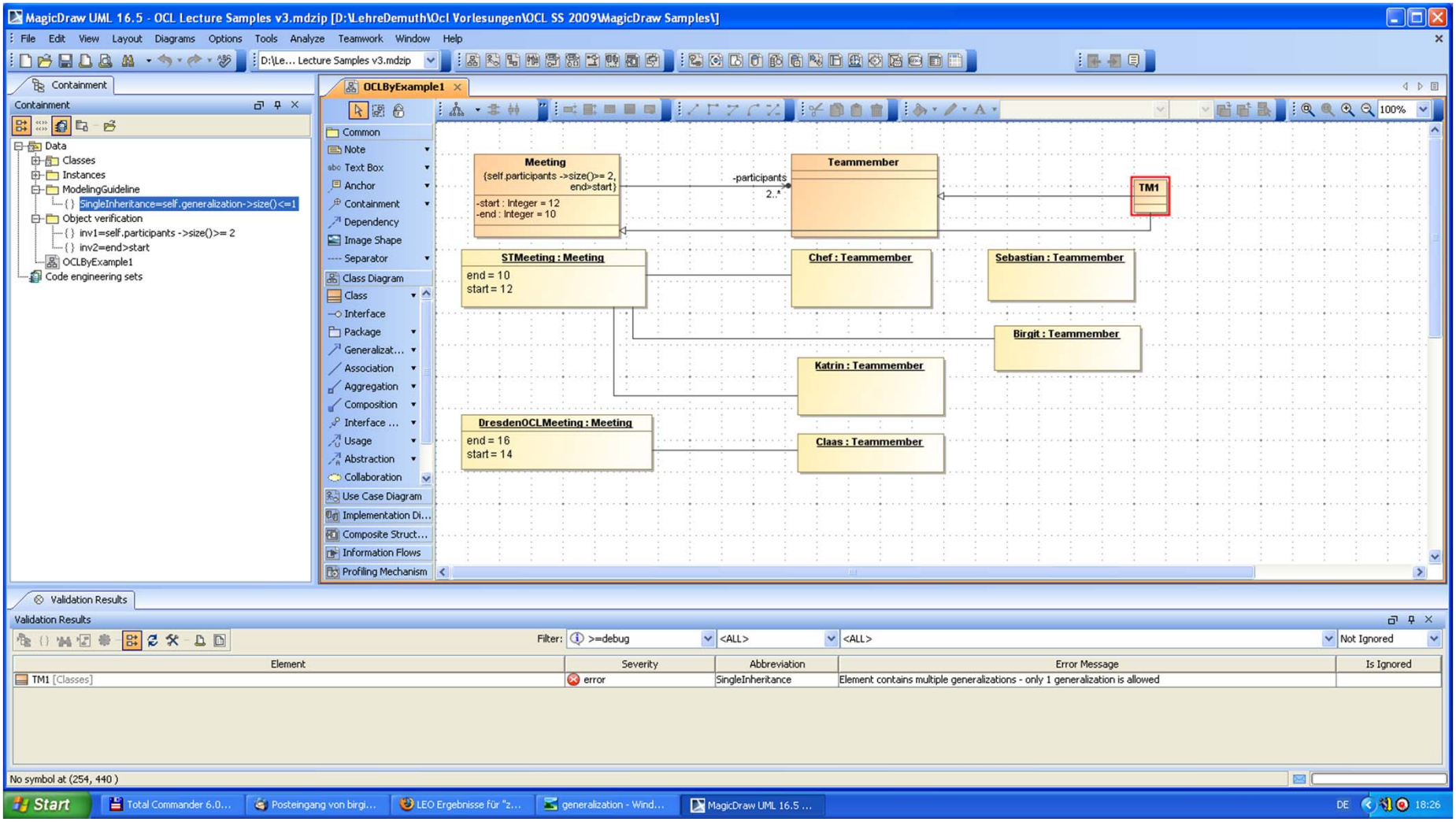
Einige weitere UML/OCL Tools

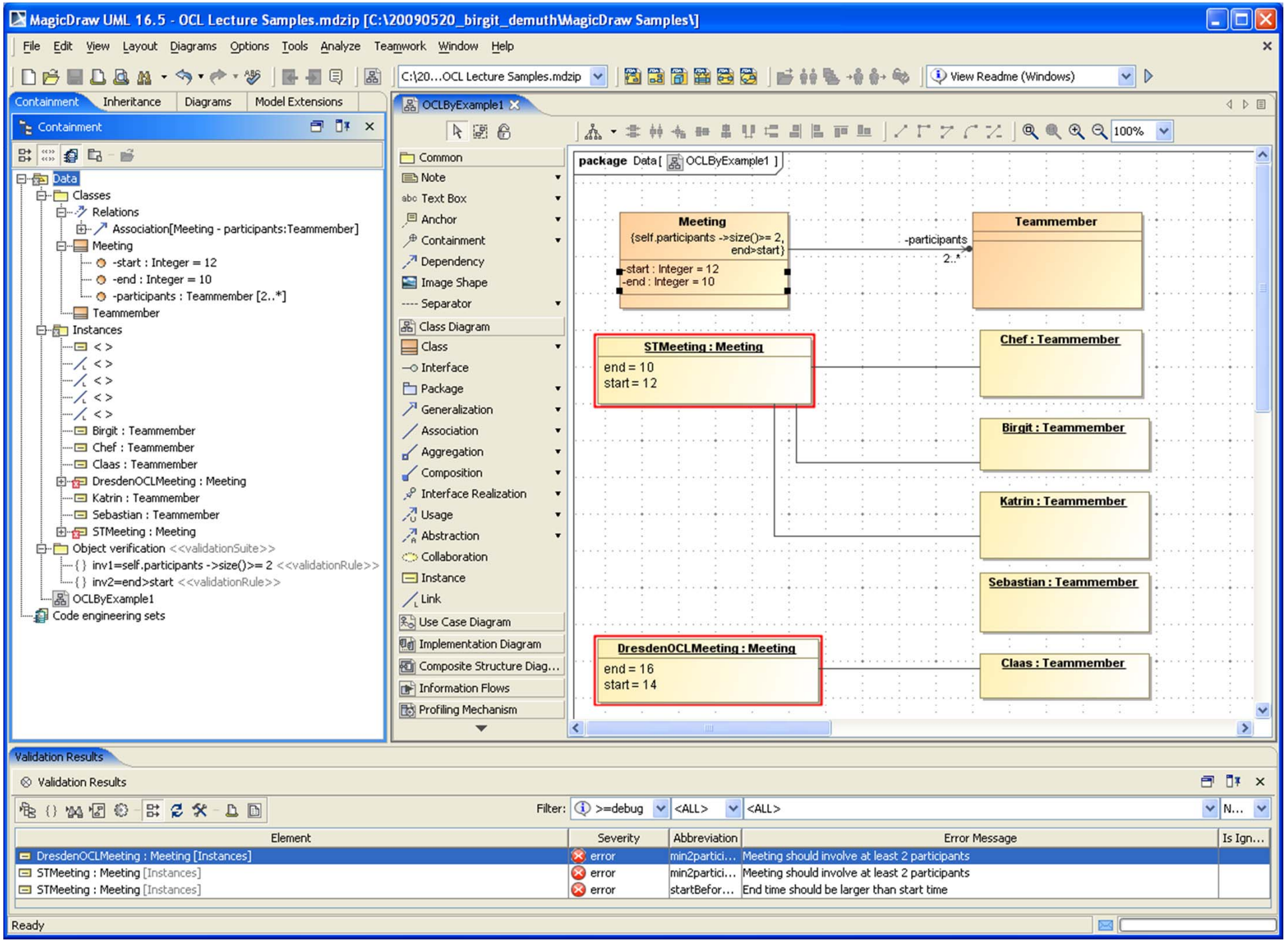
- **MagicDraw** Enterprise Edition v18 (mit Dresden OCL)
  - Evaluations-Lizenz
- **Eclipse MDT/OCL** for Ecore Based Models
  - Frei verfügbar
- **Use** (Universität Bremen)
  - Frei verfügbar
  - Animation, sehr schön geeignet für Übungs- und Animationszwecke
- **iOCL** (Interactive Tool for Specifying, Validating and Evaluating OCL Constraints)
  - https://modeling-languages.com/iocl-write-ocl-constraints-interactively/
  - Zum OCL Training gedacht

OCL Support in MagicDraw Enterprise Edition

“OCL validation rules”

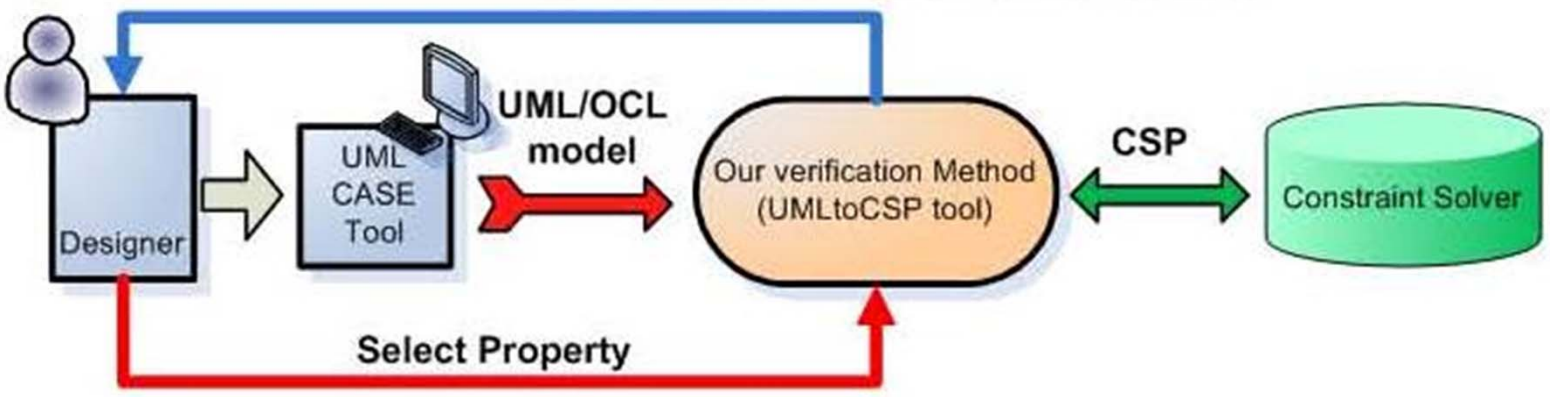
1. Spezifikation auf UML Metaklassen (M2) / Verifikation von UML-Modellen (M1)
2. Spezifikation von Stereotypen (M2) / Verifikation von UML-Modellen (M1)
3. Spezifikation auf UML-Modellen (M1) / Verifikation von UML-Instanzen (Objekten)


MagicDraw UML 16.5 - OCL Lecture Samples v3.mdzip [D:\LehreDemuth\Ocl Vorlesungen\OCL SS 2009\MagicDraw Samples\]
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
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
-participants
2..*
Teammember
TM1
STMeeting : Meeting
end = 10
start = 12
Chef : Teammember
Sebastian : Teammember
Birgit : Teammember
Katrin : Teammember
DresdenOCLMeeting : Meeting
end = 16
start = 14
Claas : Teammember
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 "z...
generalization - Wind...
MagicDraw UML 16.5 ...
DE
18:26

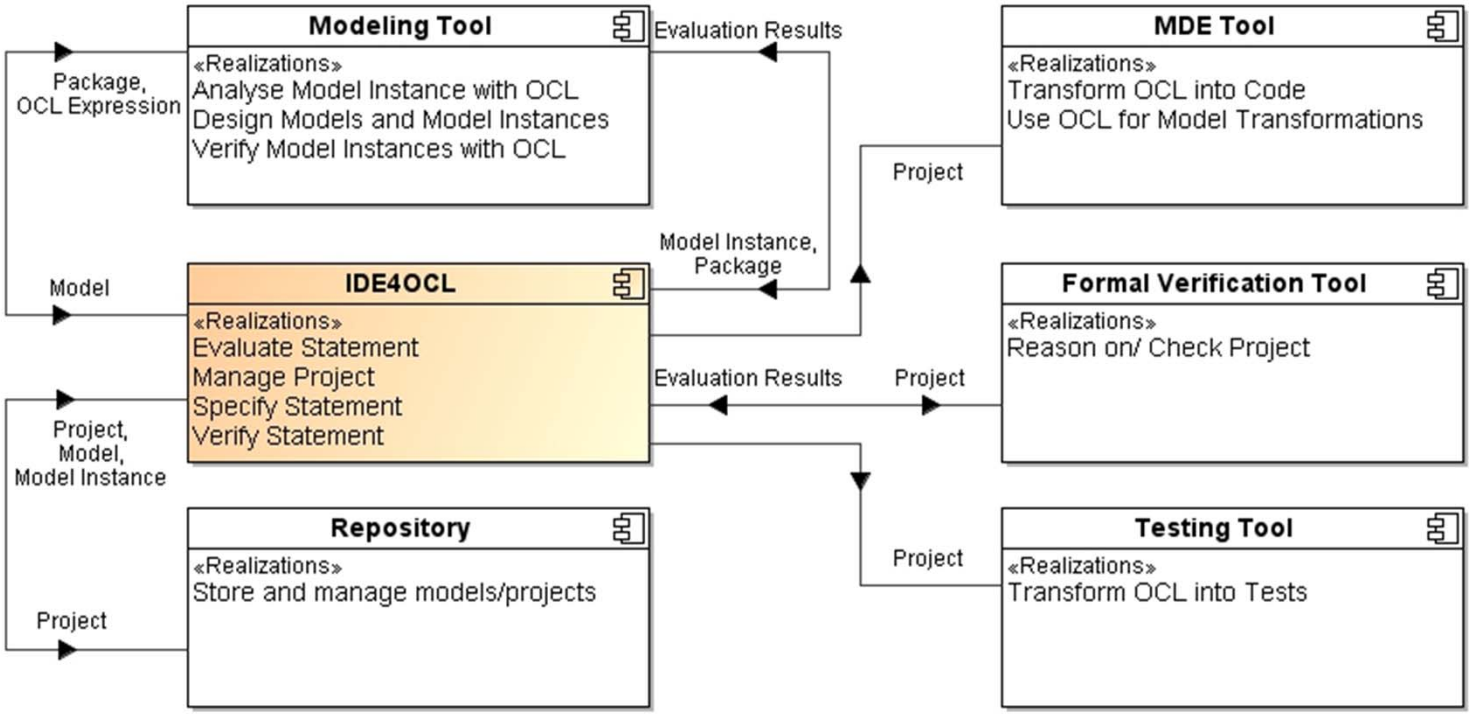

MagicDraw UML 16.5 - OCL Lecture Samples.mdzip [C:\20090520_birgit_demuth\MagicDraw Samples\]
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Containment Inheritance Diagrams Model Extensions
Containment
Data
Classes
Relations
Association[Meeting - participants:Teammember]
Meeting
-start : Integer = 12
-end : Integer = 10
-participants : Teammember [2..*]
Teammember
Instances
Birgit : Teammember
Chef : Teammember
Claas : 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
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 Diag...
Information Flows
Profiling Mechanism
package Data [OCLByExample1]
Meeting
{self.participants ->size()>= 2, end>start}
-start : Integer = 12
-end : Integer = 10
-participants
2..*
Teammember
STMeeting : Meeting
end = 10
start = 12
Chef : Teammember
Birgit : Teammember
Katrin : Teammember
Sebastian : Teammember
DresdenOCLMeeting : Meeting
end = 16
start = 14
Claas : Teammember
Validation Results
Filter: >=debug <ALL> <ALL>
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-Models in ein Constraint Satisfaction Problem(CSP)), Jordi Cabot et al
    - http://gres.uoc.edu/UMLtoCSP/



The OCL Tool Landscape [Chimiak-Opoka, Demuth 2009]



Kontakt:
birgit.demuth@tu-dresden.de

VIELEN DANK FÜR DIE AUFMERKSAMKEIT