TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Institute for Software and Multimedia Technology

OCL (Object Constraint Language) by Example

Dr. Birgit Demuth

In theory, there is no difference
between theory and practice.
But, in practice, there is.

Jan L. A. van de Snepscheut/Yogi Bera

Main Goals of the Lecture

- Bridge the gap between practically used software specifications (UML) and formal languages
- Introduce into OCL (history, outline, literature)
- Learn how to specify semantics using OCL
- Learn what are interesting OCL use cases
- Inform what OCL tools can already be used

Foundation: Assertions

- An **assertion** is a predicate (i.e., a true–false statement) placed in a program to indicate that the developer *thinks* that the predicate is always true at that place [Wikipedia].

- Usage in
  - Hoare logic [Hoare 1969]
  - Design by contract [Meyer 1986, Eiffel]
  - For run-time checking (Java (`assert`), JML, JASS, SQL, ...)
  - During the development cycle (debugging)
  - Static assertions at compile time

Object Constraint

- Model-based assertion

- [Warmer and Kleppe] define a constraint as follows:

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

- OCL as specification language for object constraints

History of OCL

- Developed at IBM in 1995 originally as a business engineering language
- Adopted as a formal specification language within UML
- Part of the official OMG standard for UML (from version 1.1 on)
- Used for precisely defining the well-formedness rules (WFRs) for UML and further OMG-related metamodels
- Current version is OCL 2.0

OCL (Object Constraint Language)

- Extends the Unified Modeling Language (UML)
- Formal language for the definition of constraints and queries on UML models
- Declarative
- Side effect free
- Add precise semantics to visual (UML-) models
- Generalized for all MOF based metamodels
- Meanwhile generally accepted
- Many extensions such as for temporal constraints
- „Core Language“ of other OMG languages (QVT, PRR)

Literature

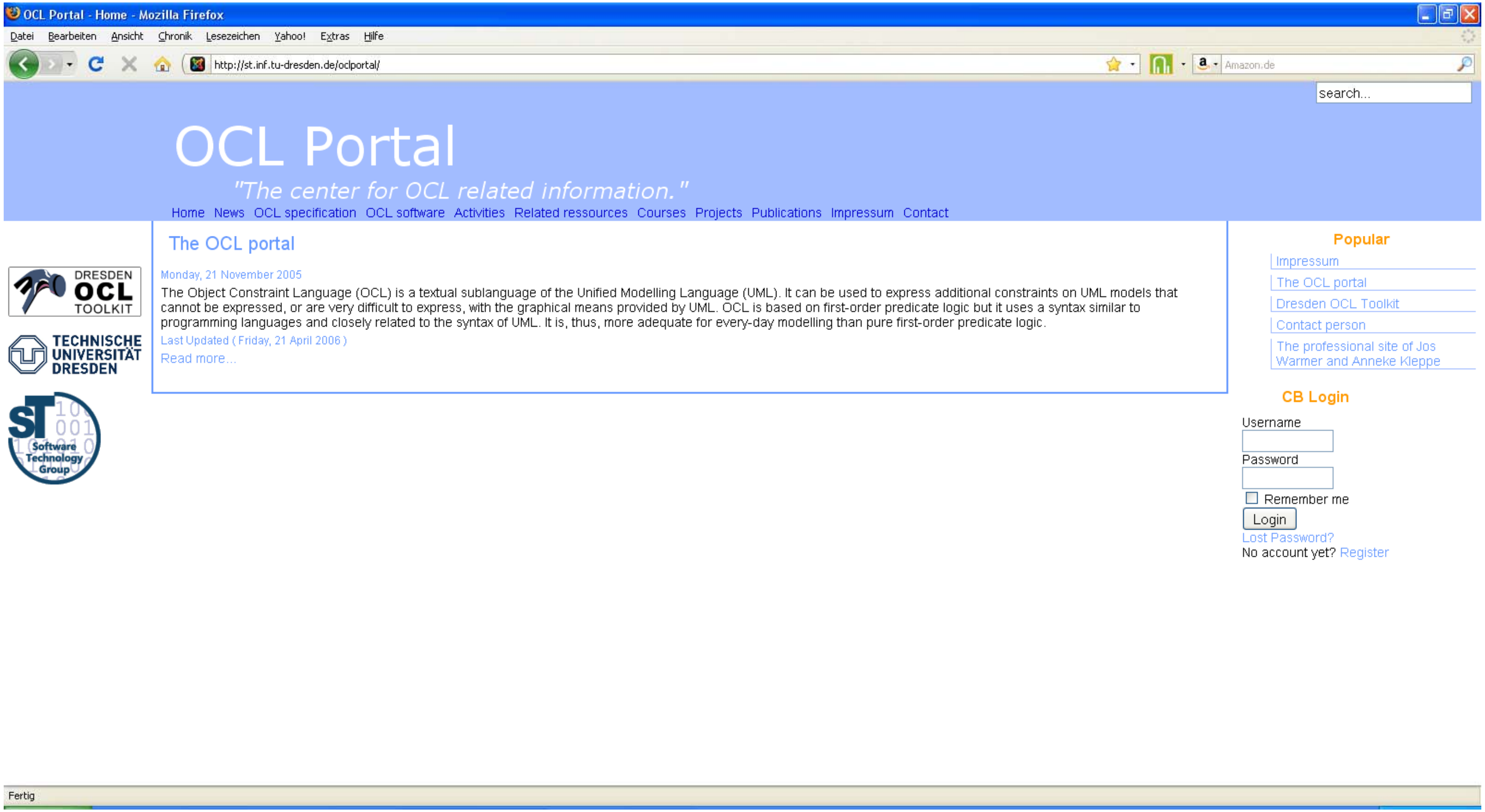
[1] Warmer, J., Kleppe, A.: The Object Constraint Language. Precise Modeling with UML. Addison-Wesley, 1999

[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 UML 2.0 OCL, www.omg.org/technology/documents/formal/ocl.htm

[5] Heinrich Hußmann: Formal Specification of Software Systems. Course, 2000, Technische Universität Dresden


OCL Portal - Home - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Yahoo! Extras Hilfe
http://st.inf.tu-dresden.de/oclportal/
Amazon.de
search...
OCL Portal
"The center for OCL related information."
Home News OCL specification OCL software Activities Related ressources Courses Projects Publications Impressum Contact
DRESDEN OCL TOOLKIT
TECHNISCHE UNIVERSITÄT DRESDEN
Software Technology Group
The OCL portal
Monday, 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 (Friday, 21 April 2006)
Read more...
Popular
Impressum
The OCL portal
Dresden OCL Toolkit
Contact person
The professional site of Jos Warmer and Anneke Kleppe
CB Login
Username
Password
Remember me
Login
Lost Password?
No account yet? Register
Fertig

TECHNISCHE UNIVERSITÄT DRESDEN

Constraint

Definition

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

- A constraint is formulated on the level of classes, but its semantics is applied on the level of objects.
- originally formulated in the syntactic context of a UML UML model (i.e. a set of UML diagrams)

Invariant

Definition

– An **invariant** is a constraint that should be true for an object during its complete lifetime.

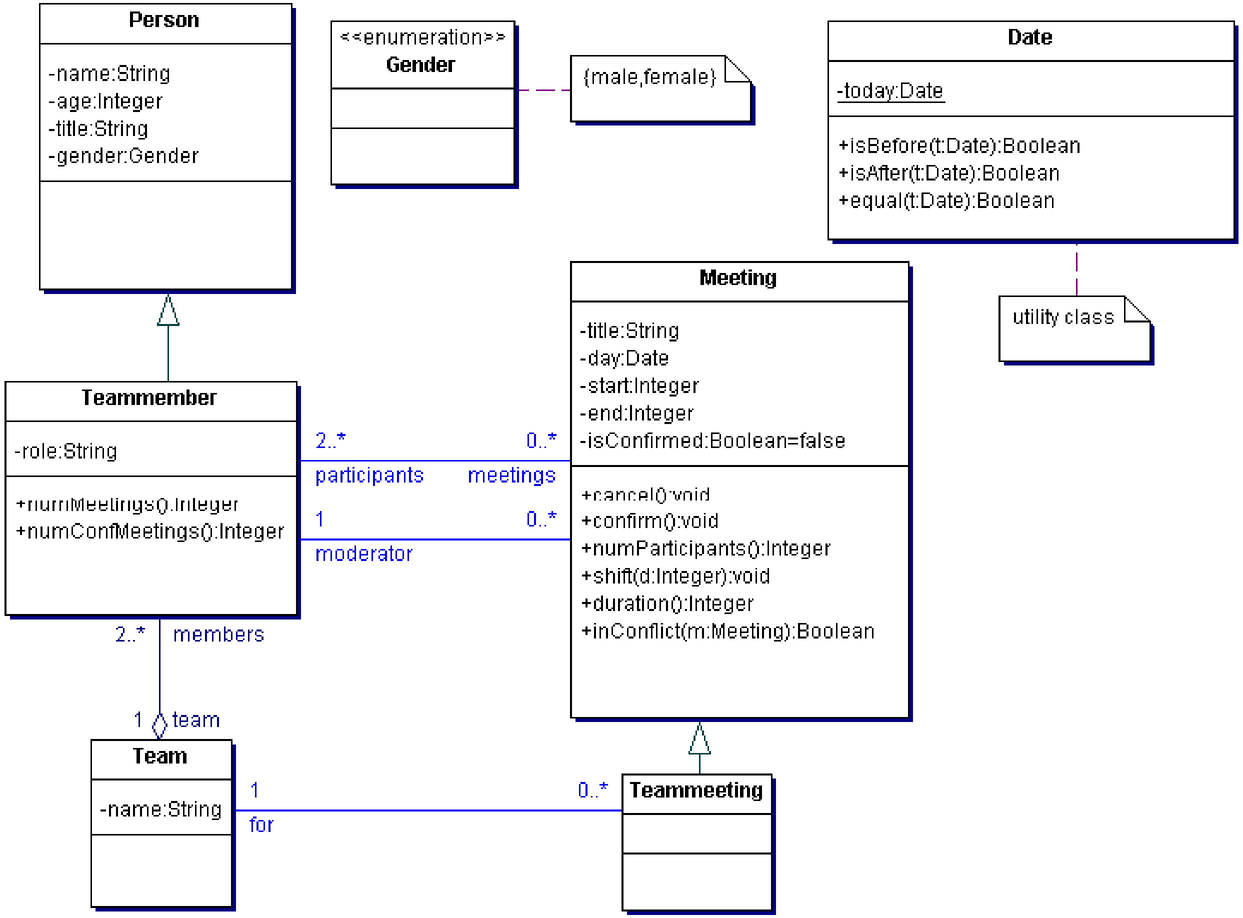
- Invariants often represent rules that should hold for the real-life objects after which the software objects are modeled.

Syntax

```
context <classifier>
inv [<constraint name>]: <Boolean OCL expression>
```

TECHNISCHE UNIVERSITÄT DRESDEN

OCL/UML By Example



Invariant - Examples

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

Equivalent Formulations

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

-- "self" always refers to the object identifier from which the constraint is evaluated.

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

-- Names can be given to the constraint

Precondition /Postcondition

- Constraint that specify the applicability and effect of an operation without stating an algorithm or implementation
- Are attached to an operation in a class diagram
- Allow a more complete specification of a system

Precondition

Definition

- Constraint that must be true just prior to the execution of an operation

Syntax

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

Precondition - Examples

```
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

Definition

- Constraint that must be true just after to the execution of an operation

- Postconditions are the way how the actual effect of an operation is described in OCL.

Syntax

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

Postcondition - Examples

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

-- keyword *result* refers to the result of the operation

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

Postcondition – Examples (cont.)

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

-- *start@pre* indicates a part of an expression
-- which is to be evaluated in the original state
-- before execution of the operation

-- *start* refers to the value upon completion of the operation

-- @pre is only allowed in postconditions

Postcondition – Examples (cont.)

- **messaging** only in postconditions
- is specifying that communication has taken place
- **hasSent (“^“)** operator

```
context Subject::hasChanged()
post: observer^update(2,4)
```

/* standard **observer pattern:**
results in true if an update message with arguments 2 and 4 was sent to the observer object during execution of the operation *hasChanged()*
*/

Building OCL Expressions <OCL expression> (1)

- Boolean expressions

- **Standard library** of
  primitive types and associated operations

  - **Basic types** (Boolean, Integer, Real, String)

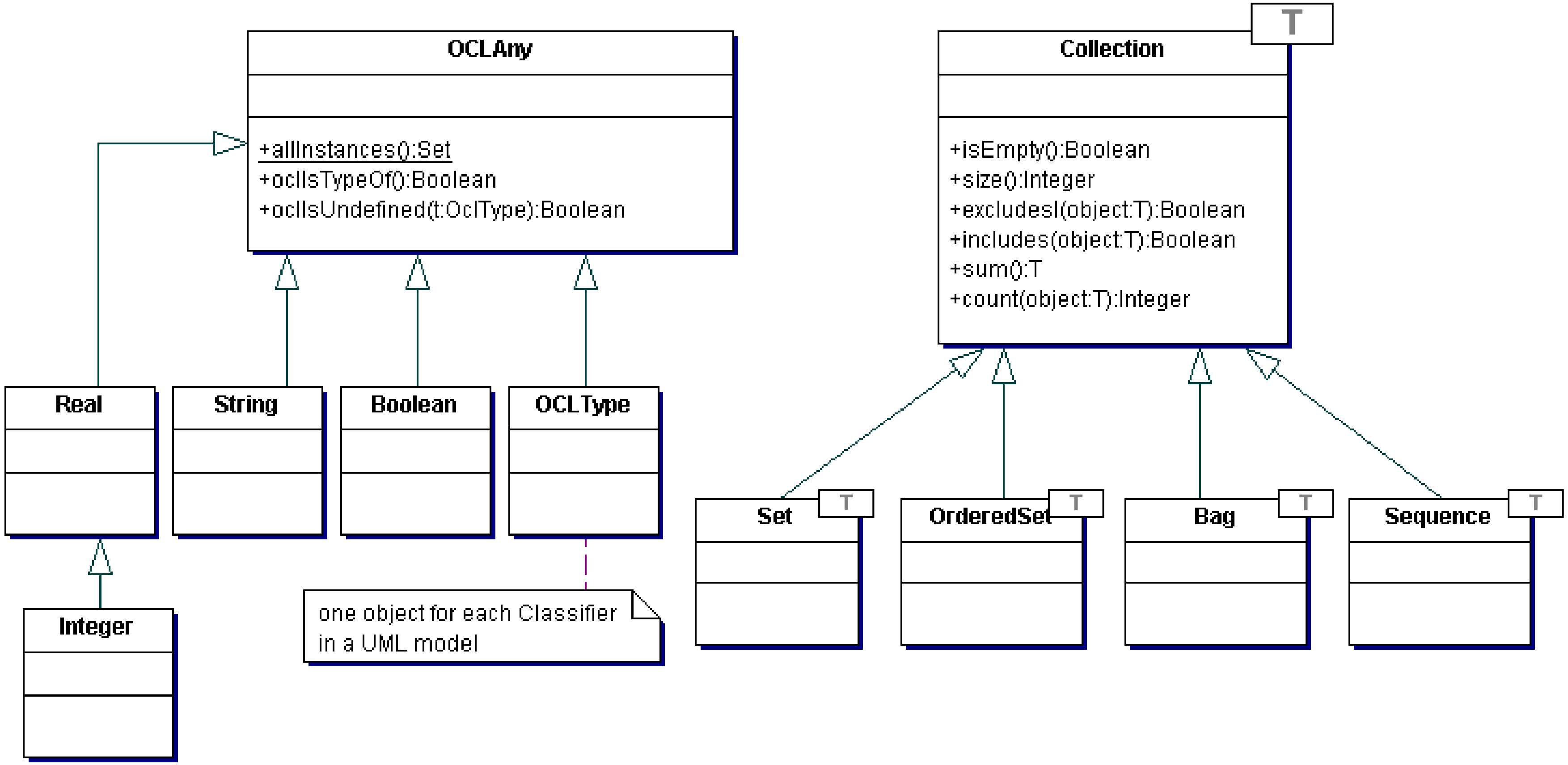
  - **Collection types:**
    - Collection
    - Set
    - Ordered Set (only OCL2)
    - Bag
    - Sequence

Building OCL Expressions <OCL expression> (2)

User defined types (OCLType)

- **Class type (Model type):**
  - Classifier in a class diagram (implicitly defined)
  - Generalisation among classiefiers leads to **Supertypes**
  - A class has the following **Features***:*
    - Attributes (`start`)
    - Operations (`duration()`)
    - Class attributes (`Date::today`)
    - Class operations
    - Association ends („navigation expressions“)
- **Enumeration type** (`Gender, Gender::male`)

OCL Type Hierarchy



OCL Type Conformance Rules

OCL is a strongly typed language .

The parser has to check the **conformance**:

- Type1 conforms to Type2 if an instance of Type1 can be substituted at each place where an instance of Type2 is expected.

General rules:

- Each Type conforms to each of its supertypes.
- Type conformance is transitive.
- A paramerized type T(X) conforms to T(Y) if X conforms to Y.

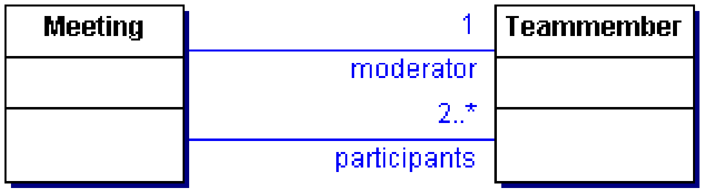
OCL Constraints and Inheritance

- Constraints are inherited.
- **Liskov's Substitution Principle**
  - Wherever an instance of a class is expected, one can always substitute an instance of any of its subclasses.
- An **invariant** for a superclass is inherited by its subclass. A subclass may strengthen the invariant but cannot weaken it.
- A **precondition** may be weakened but not strengthened in a redefinition of an operation in a subclass.
- A **postcondition** may be strengthened but not weakened in a redefinition of an operation in a subclass.

Navigation Expressions

- Association ends (role names) are be used to „navigate“ from one object in the model to another object.
- Navigations are treated as attributes (*dot-Notation*).
- The type of a navigation expression is either a
  - **User defined type**
    (association end with multiplicity at most 1)
  - **Collection**
    (association end with multiplicity > 1)

Navigation Expressions - Examples



User defined type

– Navigation from Meeting to moderator results in type Teammember

```
context Meeting
inv: self.moderator.gender = Gender::female
```

Navigation Expressions - Examples

Collection

– Navigation von Meeting to participants results in type Set(Teammember)

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

or with **shorthand** notation:

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

Collection Operations (1)

- 22 operations with variant meaning depending on the collection type such as
  - equals (**=**) and not equals operation (**< >**)
  - Transformations (**asBag(), asSet(), asOrderedSet(), asSequence()**)
  - **including(object)** and **excluding(object)**
  - **flatten()** for example

    `Set{Bag{1,2,2},Bag{2}}` → `Set{1,2}`
  - Typical set operations (**union,intersection,minus,symmetricDifference)**
  - Operations on ordered collections only (OrderedSet, Sequence) (such as **first(), last(), indexOf()**)

Collection Operations (2)

Loop operations (Iterators) on all collection types

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

Loop Operation **iterate()**

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

- All other loop operations can be described as a special case of *iterate*() such as in the following simple example:

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

Set{1,2,3}->
iterate{i: Integer, sum: Integer=0 | sum + i }
```

Further Examples for Collection Operations (1)

- A teammeeting has to be organized for a whole team (**forAll()**):

```
context Teammeeting
inv: participants->forAll(team=self.for)

context Meeting inv: oclIsTypeOf(Teammeeting)
  implies participants->forAll(team=self.for)
```

TECHNISCHE UNIVERSITÄT DRESDEN

Further Examples for collection operations (2)

- Postconditions (**select()**):

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

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

Flattening of Collections

Automatic flattening rule for all nested collections

`self.participants.meetings`

in the context „Meeting“

What happens?

- `self.participants` delivers a Set(Person)
- **`self.participants.meetings`** delivers a Bag(Set(Person))
- Results in a Bag(Person)

Derivation Rule (derive, OCL2)

- **Derived attribute** (size)

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

- **Derived association** (conflict)
  - defines a set of meetings that are in conflict with each other

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

Initial Value (init, OCL2)

Examples

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

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

- Note that an initial value must be valid only at the object creation time!

Query Operation (body, OCL2)

- Operations that do not change the state of the system
- Can be used as a query language
- Power of SQL

Example

```
context
Teammember::getMeetingTitles(): Bag(String)
body: meetings->collect(title)
```

Let Expression (let)

- Interesting for complex expressions
- Define a local variable (`noConflict`) that can be used instead of a sub-expression

```
context Meeting inv:
   let noConflict : Boolean =
      participants.meetings-> forAll(m|m<>self and
      m.isConfirmed implies not self.inConflict(m))
   in  isConfirmed implies noConflict
```

Defining New Attributes and Operations(def, OCL2)

- Adding attributes and query operations to the model
- Syntax is similar to the let expression
- Helpful for the reuse of OCL expressions in several constraints

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

Packaging OCL Expressions

```
package MeetingExample

context Meeting::isConfirmed : Boolean
init: false

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

..

endpackage
```

Limitations of OCL

- No support for **inconsistency detection** for OCL
- **„Frame Problem“**
  - Operations are specified by what they change (in post-conditions), with the implicit assumption that everything else (the frame) remains unchanged
- **Limited recursion**
- **allInstances()** Problem:
  - `Person.allInstances()` allowed
  - not allowed for **infinite** types such as `Integer.allInstances()`

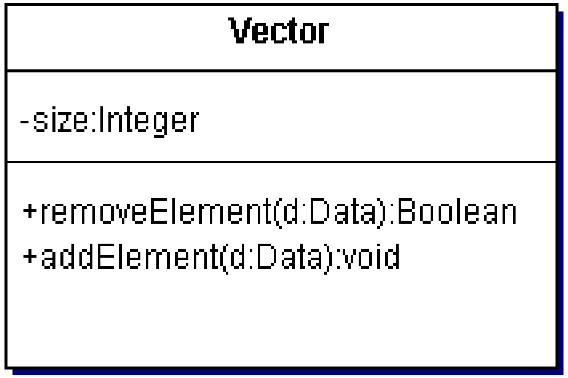
Building complete models with OCL

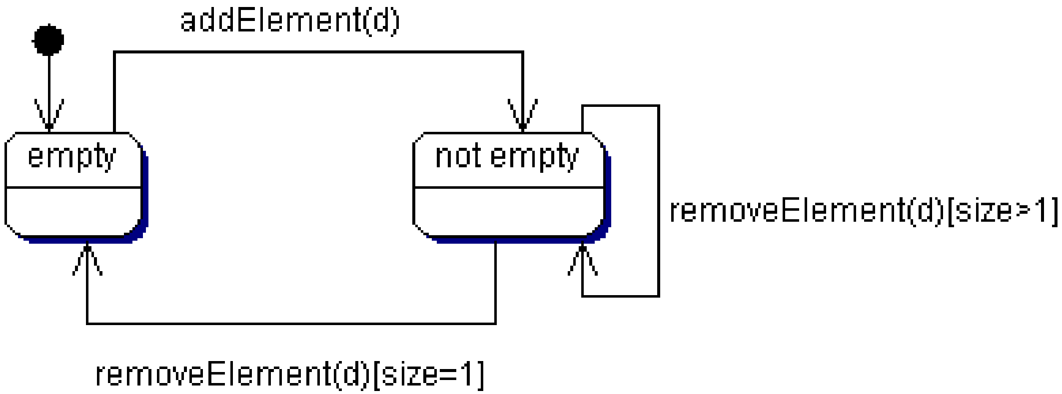
- Statechart diagram
- Interaction diagram
- Activity diagram
- Component diagram
- Use case diagram

OCL in Statecharts – Example (oclInState())

operation on all objects (Typ `OclAny`)

`oclInState(s: OclState) : Boolean`






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

Undefined Values in OCL

- An OCL expression can evaluate to „undefined“ (**OclVoid**)
  - For example: Access to an attribute value or navigation where no value is existent in the respective object

- **Strictness Principle**
  - Whenever a subexpression of an OCL expression evaluates to undefined, then the whole term evaluates to undefined
  - Exceptions
    - True or undefined = True
    - False and undefined = False
    - False implies undefined = True

The OclVoid Type

- Undefined value is the only instance
- Operation for testing if the value of an expression is undefined

```
oclIsUndefined(): Boolean
```

-- true if the object is undefined

-- otherwise false

Some Tips for Writing OCL Expressions

Constraints should be easy to read and write:

- Avoid complex navigation expressions
- Choose appropriate context
- Avoid allInstances()
- Split „and“ constraints by writing multiple constraints
- Use the „collect“ shorthand
- Use association end names (role names) instead of association names in modeling

Typical Use Cases for OCL

Metamodels: {MOF-, Ecore-based} X {UML, CWM, ODM, SBVR, PRR, DSLs}

Model Layer	Examples
M2 (Metamodel)	•Specification of **WFRs** in OMG standards •Definition of **Modeling Guidelines** for DSLs •Specification of **Model Transformations**
M1 (Model)	•Model Verification (→ CASE-Tool) •Evaluation of modeling guidelines •Execution of model transformations
	•Specification of **Business Rules/Constraints** •Specification of **Test Cases**
M0 (Objects)	•Evaluation of Business Rules/Constraints •Testing

Examples for OCL on Metamodel

- WFR in UML metamodel

```
context Classifier inv:
not self.allParents->includes(self)
```

-- Generalization cycles are not allowed

- UML modeling guideline for Java developers

```
context Classifier inv SingleInheritance:
self.generalization->size()<= 1
```

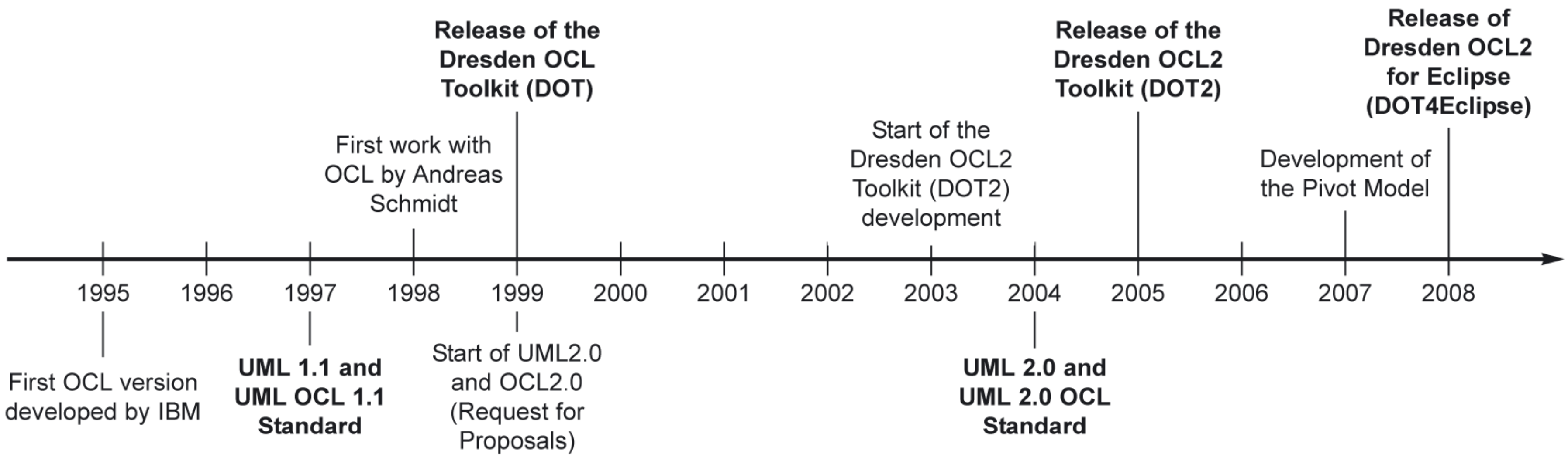
-- Multiple inheritance is not allowed

Some UML/OCL Tools

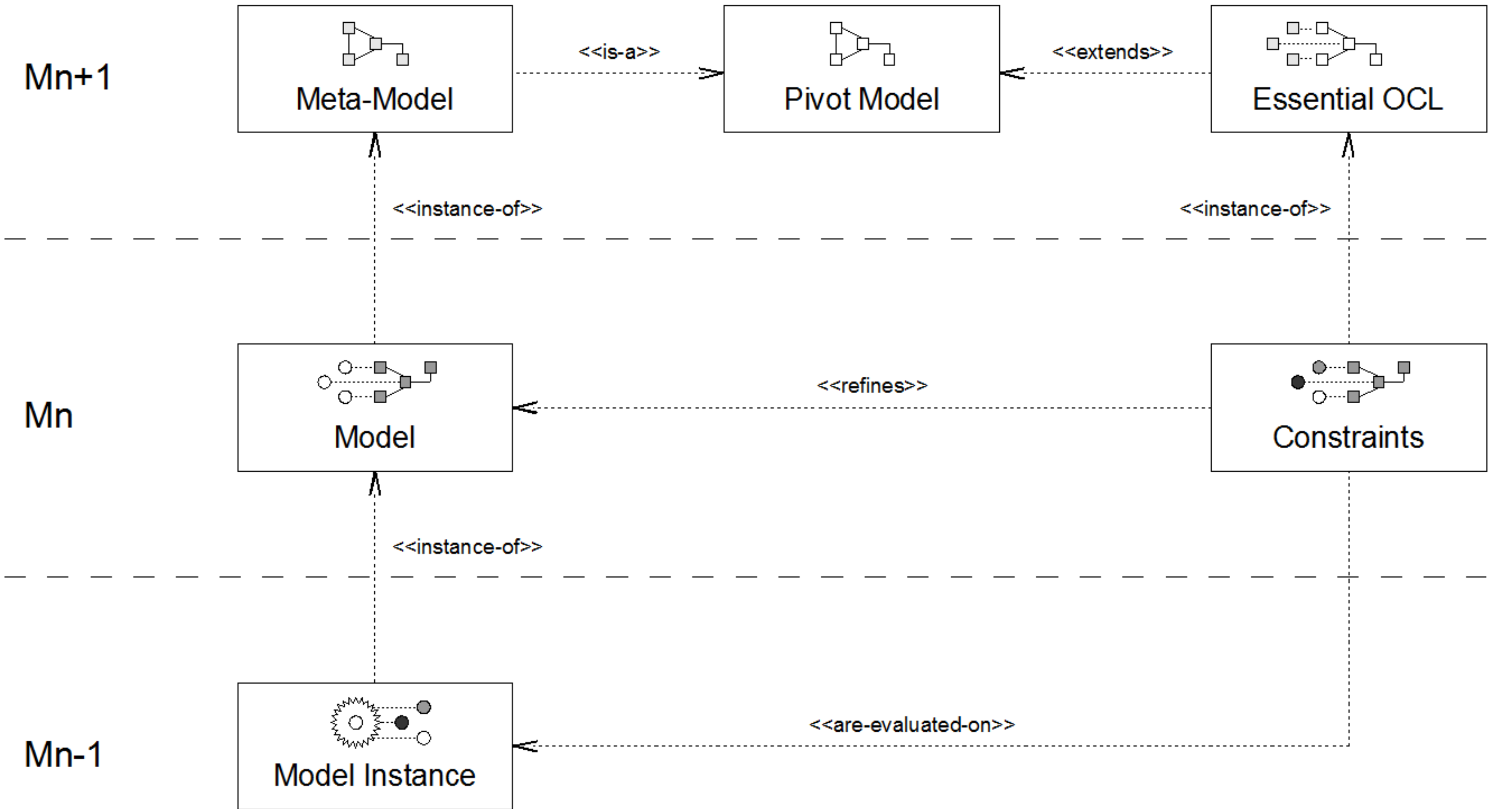
- 12 OCL tools/libraries (see OCL Portal)

- Integrations into UML environments
  - **MagicDraw** Enterprise Edition v16.5
  - Borland **Together** 2008 (OCL/QVT)
  - **Eclipse MDT/OCL** for EMF Based Models
  - **ArgoUML**
  - **Fujaba4Eclipse**

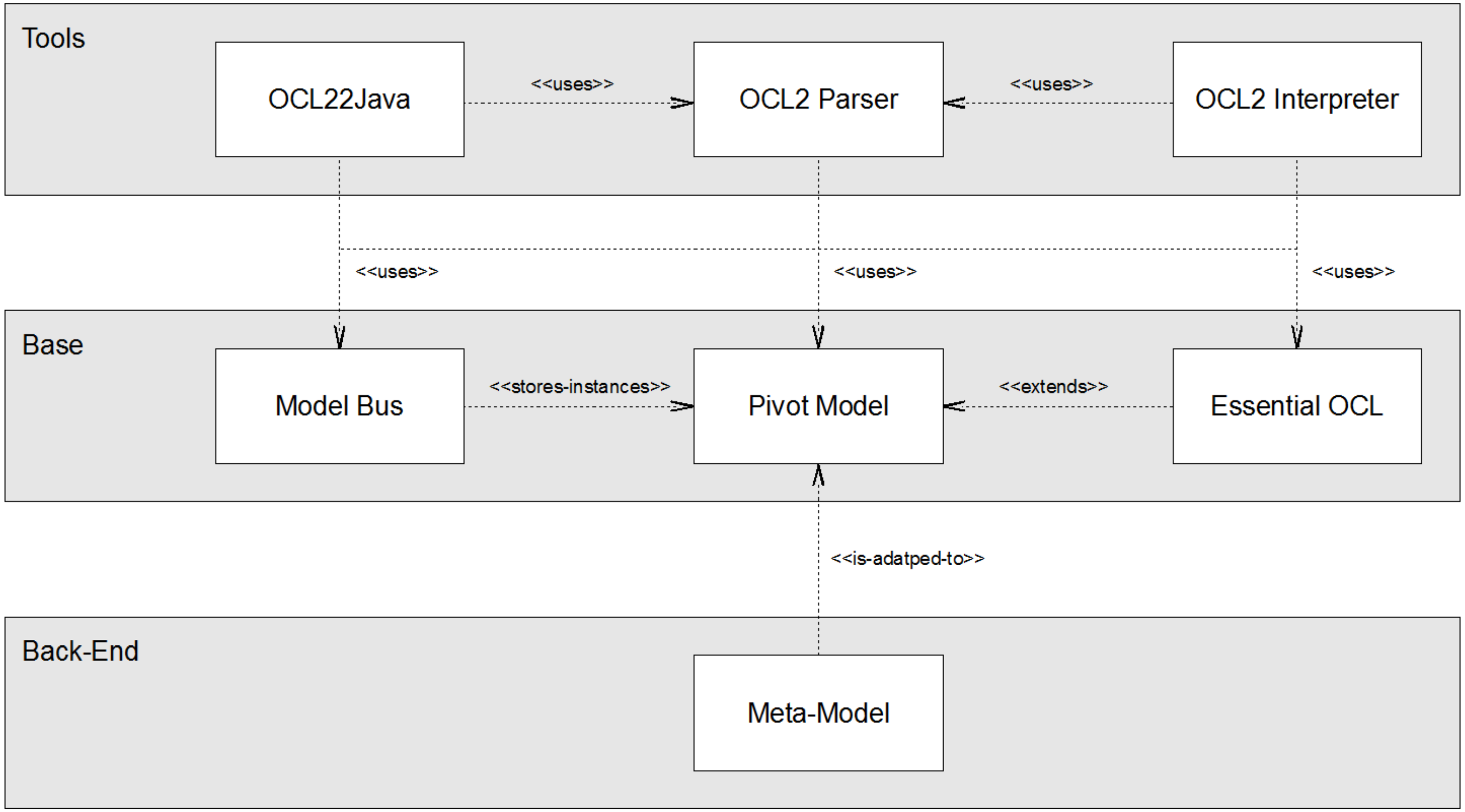
Decennial Anniversary of Dresden OCL in 2009



Dresden OCL2 for Eclipse



Dresden OCL2 for Eclipse



XMI Import **into** Dresden OCL2 for Eclipse

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

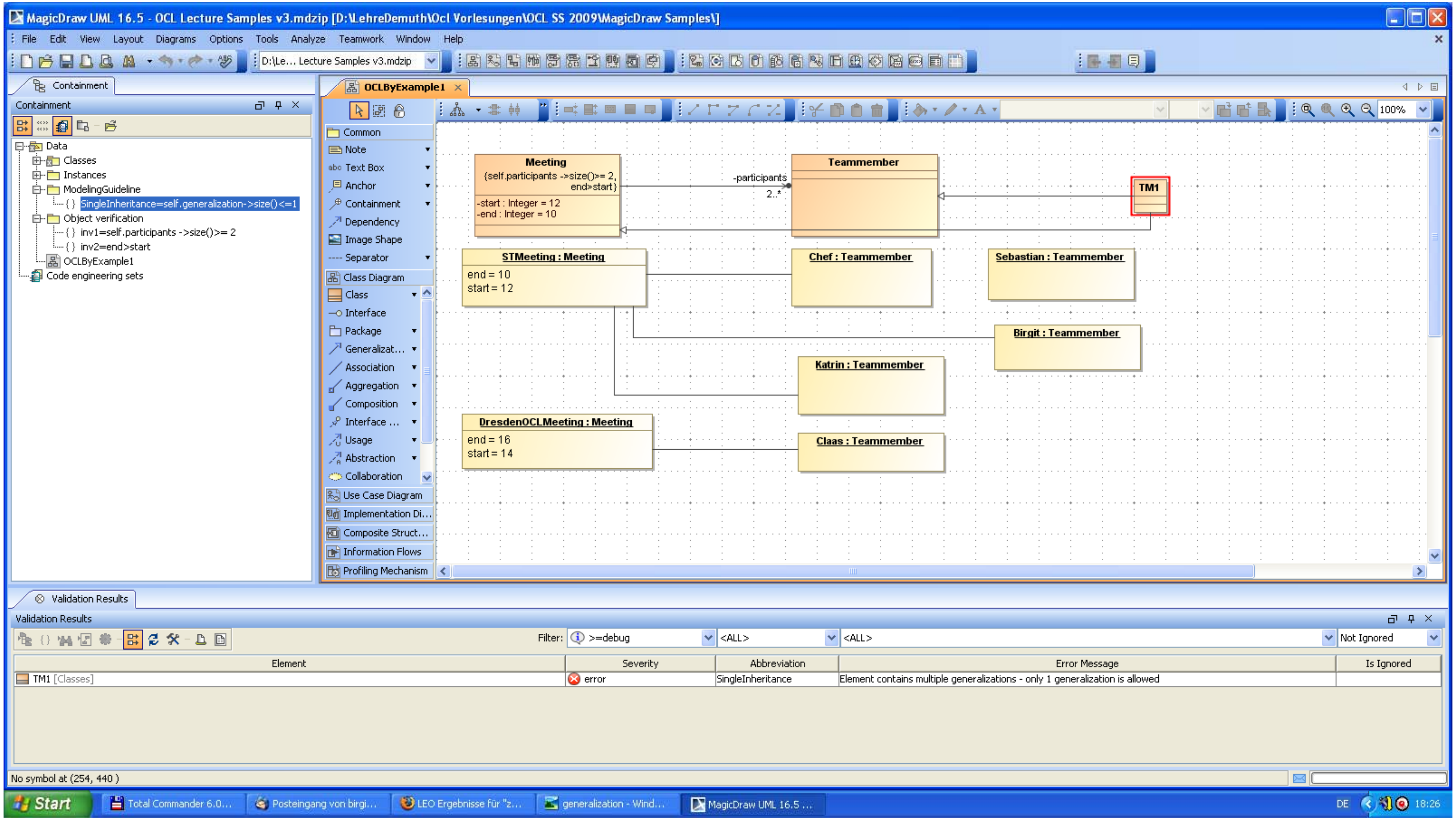
OCL Support in MagicDraw Enterprise Edition

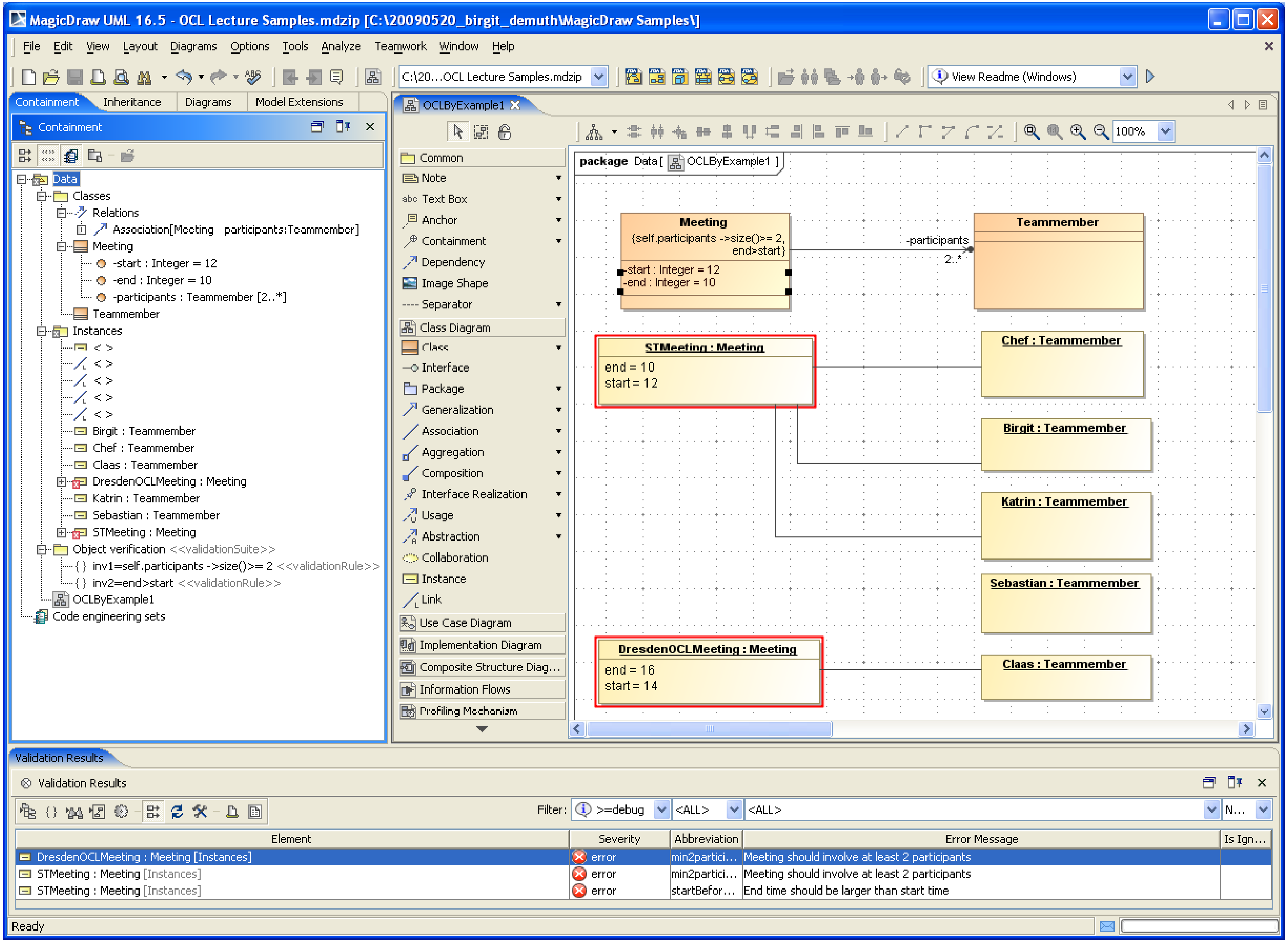
"OCL validation rules"
(based on Dresden OCL2 Toolkit)

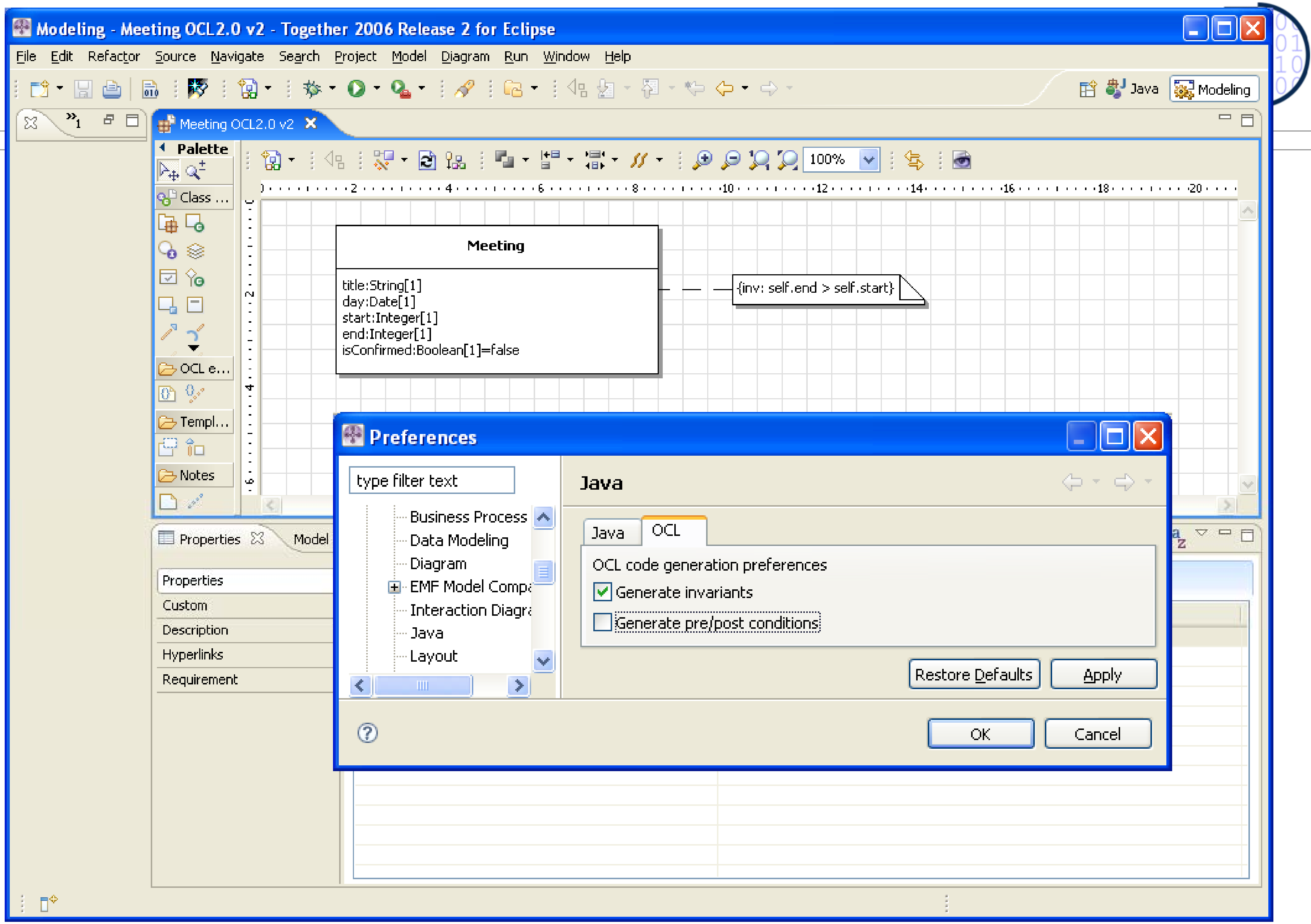
1. Specification on UML metamodel (M2) /
   Verification on UML models (M1)

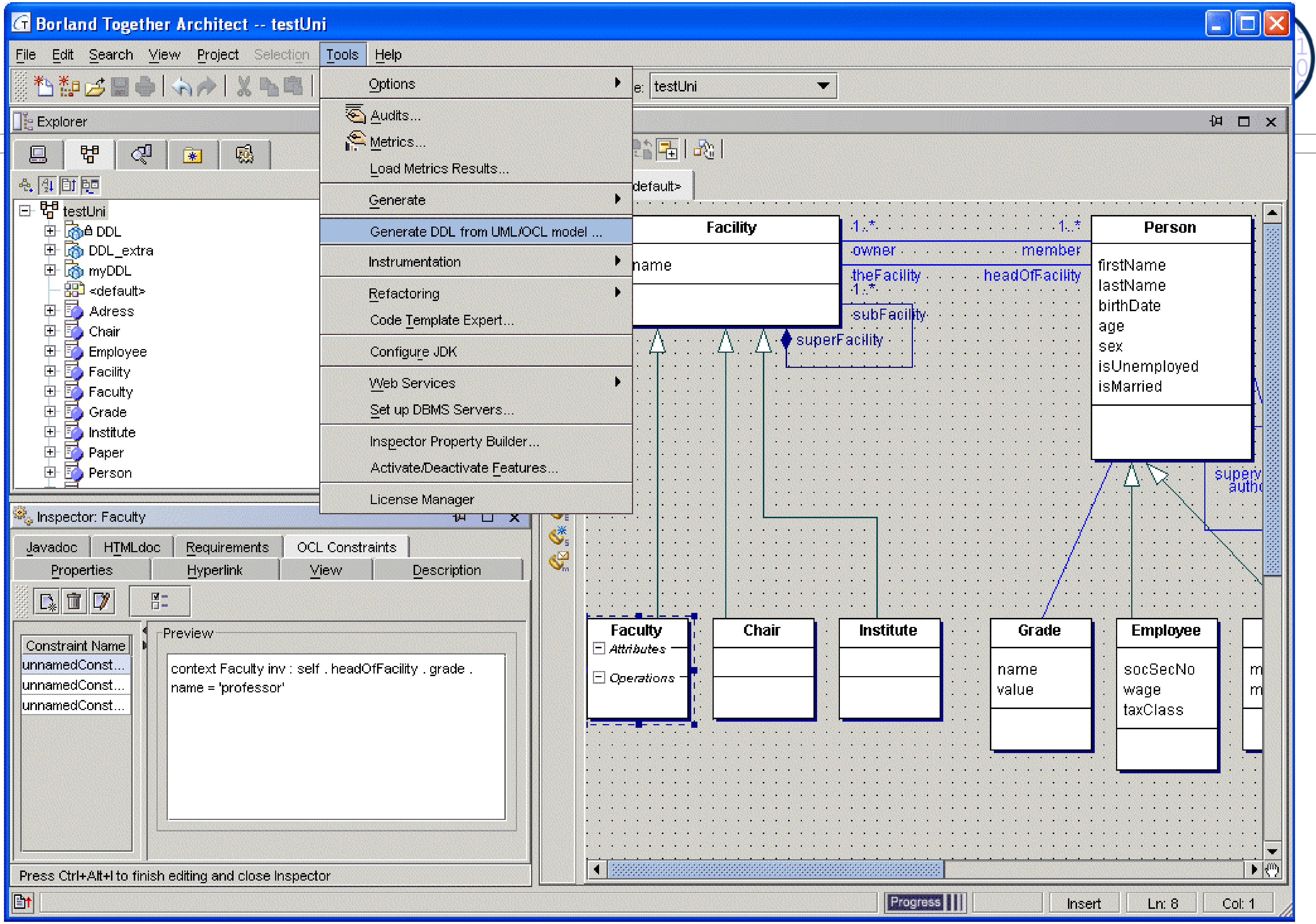
2. Specification of Stereotypes (M2) /
   Verification of UML models (M1)

3. Specification on UML models (M1) /
   Verification of UML instances (objects)


MagicDraw UML 16.5 - OCL Lecture Samples v3.mdzip [D:\LehreDemuth\Ocl Vorlesungen\OCL SS 2009\MagicDraw Samples\]
SingleInheritance=self.generalization->size()<=1
inv1=self.participants ->size()>= 2
inv2=end>start
Meeting
{self.participants ->size()>= 2, end>start}
-start : Integer = 12
-end : Integer = 10
Teammember
-participants
2..*
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
TM1 [Classes]
error
SingleInheritance
Element contains multiple generalizations - only 1 generalization is allowed


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:\20...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
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
100%
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
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


Modeling - Meeting OCL2.0 v2 - Together 2006 Release 2 for Eclipse
File Edit Refactor Source Navigate Search Project Model Diagram Run Window Help
Java
Modeling
Meeting OCL2.0 v2
Palette
Class ...
OCL e...
Templ...
Notes
100%
Meeting
title:String[1]
day:Date[1]
start:Integer[1]
end:Integer[1]
isConfirmed:Boolean[1]=false
{inv: self.end > self.start}
Properties
Model
Properties
Custom
Description
Hyperlinks
Requirement
Preferences
type filter text
Business Process
Data Modeling
Diagram
EMF Model Compa
Interaction Diagra
Java
Layout
Java
Java
OCL
OCL code generation preferences
Generate invariants
Generate pre/post conditions
Restore Defaults
Apply
OK
Cancel


Borland Together Architect -- testUni
File Edit Search View Project Selection Tools Help
Options
Audits...
Metrics...
Load Metrics Results...
Generate
Generate DDL from UML/OCL model ...
Instrumentation
Refactoring
Code Template Expert...
Configure JDK
Web Services
Set up DBMS Servers...
Inspector Property Builder...
Activate/Deactivate Features...
License Manager
Explorer
testUni
DDL
DDL_extra
myDDL
<default>
Adress
Chair
Employee
Facility
Faculty
Grade
Institute
Paper
Person
Inspector: Faculty
Javadoc HTMLdoc Requirements OCL Constraints
Properties Hyperlink View Description
Constraint Name
unnamedConst...
unnamedConst...
unnamedConst...
Preview
context Faculty inv : self . headOfFacility . grade . name = 'professor'
Press Ctrl+Alt+I to finish editing and close Inspector
Facility
name
1..*
owner
member
theFacility
headOfFacility
subFacility
superFacility
Person
firstName
lastName
birthDate
age
sex
isUnemployed
isMarried
Faculty
Attributes
Operations
Chair
Institute
Grade
name
value
Employee
socSecNo
wage
taxClass
Progress
Insert
Ln: 8
Col: 1

Acronyms

OCL	Object Constraint Language
OMG	Object Management Group
MOF	Meta-Object Facility
PRR	Production Rule Representation
QVT	Query Views Transformation
UML	Unified Modeling Language
WFR	Well-Formedness Rule

Thank you

for your attention!