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 \(\text{assert}\), 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

OCL Portal
“The center for OCL related information.”

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

The OCL portal
Monday, 21 November 2005
The Object Constraint Language (OCL) is an integral 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 ...
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 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>
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**

```plaintext
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

```cpp
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 classifiers 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

OCLAny

- allInstances: Set
-oclIsTypeOf(): Boolean
-oclIsUndefined(t: OCLType): Boolean

Real

String

Boolean

OCLType

Collection

- isEmpty(): Boolean
- size(): Integer
- excludes(object: T): Boolean
- includes(object: T): Boolean
- sum(): T
- count(object: T): Integer

Real

String

Boolean

OCLType

Collection

Set

OrderedSet

Bag

Sequence

one object for each Classifier in a UML model
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 parameterized 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 (\(\text{asBag}()\), \(\text{asSet}()\), \(\text{asOrderedSet}()\), \(\text{asSequence}()\))
  - \(\text{including}(\text{object})\) and \(\text{excluding}(\text{object})\)
  - \(\text{flatten}()\) for example
    \[
    \text{Set}\{\text{Bag}\{1,2,2\},\text{Bag}\{2\}} \rightarrow \text{Set}\{1,2\}
    \]
  - Typical set operations
    \(\text{union}, \text{intersection}, \text{minus}, \text{symmetricDifference}\)
  - Operations on ordered collections only (\(\text{OrderedSet}, \text{Sequence}\)) (such as \(\text{first}()\), \(\text{last}()\), \(\text{indexOf}()\))
Collection Operations (2)

Loop operations (Iterators) on all collection types

any(expr)
collect(expr)
eexists(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 (\texttt{forall}):

  \begin{verbatim}
  context Teammeeting
  inv: participants->forall(team=self.for)
  \end{verbatim}

  \begin{verbatim}
  context Meeting inv: oclIsTypeOf(Teammeeting)
  implies participants->forall(team=self.for)
  \end{verbatim}
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

\[ \text{self.participants.meetings} \]

in the context „Meeting“

What happens?

- \text{self.participants} delivers a \text{Set(Person)}
- \text{self.participants.meetings} delivers a \text{Bag(Set(Person)}
- Results in a \text{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
```

```
class 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

```plaintext
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

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

```ocl
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

\[ \text{oclIsUndefined}(): \text{Boolean} \]

- 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\} \times \{UML, CWM, ODM, SBVR, PRR, DSLs\}

<table>
<thead>
<tr>
<th>Model Layer</th>
<th>Examples</th>
</tr>
</thead>
</table>
| **M2** (Metamodel) | • Specification of **WFRs** in OMG standards  
|               | • Definition of **Modeling Guidelines** for DSLs  
|               | • Specification of **Model Transformations**            |
| **M1** (Model) | • Model Verification (\rightarrow 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

```ocl
class Classifier inv:
   not self.allParents->includes(self)
-- Generalization cycles are not allowed
```

- UML modeling guideline for Java developers

```ocl
class Classifier inv SingleInheritance:
   self.generalization->size() \leq 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

- **1995**: First OCL version developed by IBM
- **1997**: UML 1.1 and UML OCL 1.1 Standard
- **1998**: Start of UML2.0 and OCL2.0 (Request for Proposals)
- **1999**: Release of the Dresden OCL Toolkit (DOT)
- **2000**: Start of the Dresden OCL2 Toolkit (DOT2) development
- **2002**: Release of the Dresden OCL2 Toolkit (DOT2)
- **2004**: UML 2.0 and UML 2.0 OCL Standard
- **2005**: Development of the Pivot Model
- **2006**: Release of Dresden OCL2 for Eclipse (DOT4Eclipse)
Dresden OCL2 for Eclipse
Dresden OCL2 for Eclipse

Tools

OCL22Java <-> OCL2 Parser <-> OCL2 Interpreter

Base

Model Bus <-> Pivot Model <-> Essential OCL

Back-End

Meta-Model
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)
Meeting

- title: String[1]
- day: Data[1]
- start: Integer[1]
- end: Integer[1]
- isConfirmed: Boolean[1]=false

[Image of software interface with object diagram and preferences settings]
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta-Object Facility</td>
</tr>
<tr>
<td>PRR</td>
<td>Production Rule Representation</td>
</tr>
<tr>
<td>QVT</td>
<td>Query Views Transformation</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>WFR</td>
<td>Well-Formedness Rule</td>
</tr>
</tbody>
</table>
Thank you for your attention!