Softwaretechnik

Lecture 15: OCL

Peter Thiemann

University of Freiburg, Germany

12.07.2012

What is OCL?

- OCL = object constraint language
- standard query language of UML 2
- specify expressions and constraints in
  - object-oriented models
  - object modeling artifacts
- "a formal language that remains easy to read and write"
- "a pure specification language"
- specification edited by OMG:
  `http://www.omg.org/spec/OCL/2.3.1/`

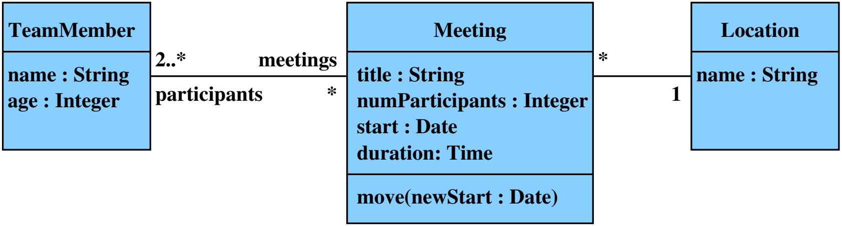
OCL/Expressions and Constraints

- Expressions
  - initial values, derived values
  - parameter values
  - body of operation (no side effects $\Rightarrow$ limited to queries)
  - of type: Real, Integer, String, Boolean, UnlimitedNatural, or model type
- Constraints
  - invariant (class): condition on the state of the class's objects which is always true
  - precondition (operation): indicates applicability
  - postcondition (operation): must hold after operation if precondition was met
  - guard (transition): indicates applicability

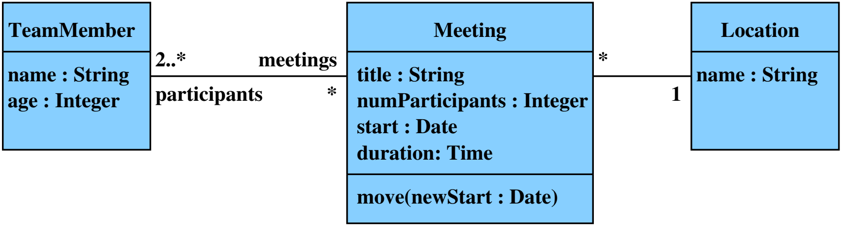
OCL/Context

- Each OCL expression is interpreted relative to a **context**
  - invariant wrt class, interface, datatype, component (a classifier)
  - precondition wrt operation
  - postcondition wrt operation
  - guard wrt transition
- Context is indicated
  - graphically by attachment as a note
  - textually using the context syntax
- Expression is evaluated with respect to a snapshot of the object graph described by the modeling artifact
- Expression evaluation does not change the object graph

OCL/Example


TeamMember
name : String
age : Integer
2..*
meetings
participants
*
Meeting
title : String
numParticipants : Integer
start : Date
duration: Time
move(newStart : Date)
*
1
Location
name : String

OCL/Example



Invariants

- `context TeamMember inv: age > 0`
- `context Meeting inv: duration > 0`

OCL/Types and Values

- Model types (class names)
- Basic types and notation for values:

`Boolean`	Values: `true`, `false`
`Integer`	Values: `1`, `-5`, `2`, `34`, `26524`
`Real`	Values: `1.4142`, `2.718`, `3.141`
`String`	Values: `'Sonntagmorgen um viertel vor ac`
`UnlimitedNatural`	Values: `0`, `1`, `33`, `...`, `*`

- Tuples
- Collection types: Collection, Set, Bag, Sequence
- Enumeration types (User-defined)
- Special types: `OclAny`, `OclType`

OCL/Operations on Basic Types

- `Boolean`: and, or, xor, not, implies, if-then-else (infix)
- `Integer`: *,+,-,/,abs,div(), mod(), max(),min()
- `Real`: *,+,-,/,floor
- `String`: size,toUpper,toLower, concat (), substring ()
- . . . and many more

OCL/Operations on Basic Types

- `Boolean`: and, or, xor, not, implies, if-then-else (infix)
- `Integer`: *,+,-,/,abs,div(), mod(), max(),min()
- `Real`: *,+,-,/,floor
- `String`: size,toUpper,toLower, concat (), substring ()
- ... and many more

Notation

- Symbols: infix notation
- Identifiers: method notation, unary methods w/o ()
- Examples: `x.abs`; `y1.mod (y2)`

OCL/Invariants

- Expressions of type Boolean
- Interpreted in 3-valued logic (true, false, undefined)
- Arithmetic and logic expressions built with the usual operators
- Attributes of the context object directly accessible
- Alternatively through self.*attributeName*
- Other values available through **navigation**

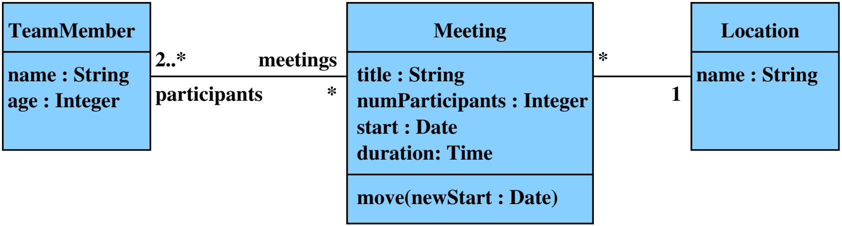
OCL/Navigation

- Task: *navigate* from *object* to associated objects
- Dot notation *object.associationEnd* yields
  - associated object (or undefined), if upper bound of multiplicity ≤ 1
  - the ordered set of associated objects, if association is $\{$ordered$\}$
  - the set of associated objects, otherwise
- Use *object.classNameOfTarget* if association end not named and target is uniquely determined

OCL/Collection Types

- Result of navigation expression has collection type
- `Collection`(t)
  Abstract type with the concrete types `Set`(t'), `OrderedSet`(t'), `Bag`(t'), and `Sequence`(t') as subtypes where t' is a subtype of t
- `Set`(t')
  Mathematical set (no duplicate elements, no order)
- `OrderedSet`(t')
  Mathematical set with ordering (no duplicate elements)
- `Bag`(t')
  Like a set, but may contain duplicates
- `Sequence`(t')
  Like a bag, but the elements are ordered

OCL/Navigation/Examples



- `context Meeting`
  - `self.location` yields the associated `Location` object
  - `self.participants` yields set of `TeamMember` objects

OCL/More Navigation

- If navigation yields object, then use
  - attribute notation
  - navigation
  - operation calls

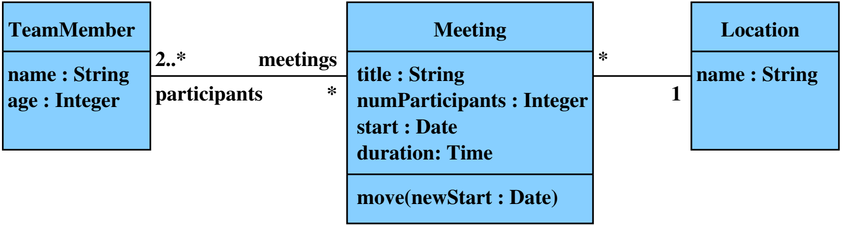
  to continue
- What if navigation yields a collection?

OCL/More Navigation

- If navigation yields object, then use
  - attribute notation
  - navigation
  - operation calls

  to continue
- What if navigation yields a collection?
- Collection operations:
  - notation *collection*->*op*(*args*)
  - example operations: `size()`, `isEmpty()`, `notEmpty()`, ...
- Single objects may also be used as collections
- Attributes, operations, and navigation of elements not directly accessible

OCL/More Navigation/Examples



- `context Meeting`
  - inv: `self.participants->size() = numParticipants`
- `context Location`
  - inv: `name="Lobby" implies meeting->isEmpty()`

OCL/Accessing Collection Elements

- Task: Continue navigation from a collection
- The `collect` operation
  - *`collection`*`->collect(` *`expression`* `)`
  - *`collection`*`->collect(` *`v`* `|` *`expression`* `)`
  - *`collection`*`->collect(` *`v`* `:` *`Type`* `|` *`expression`* `)`

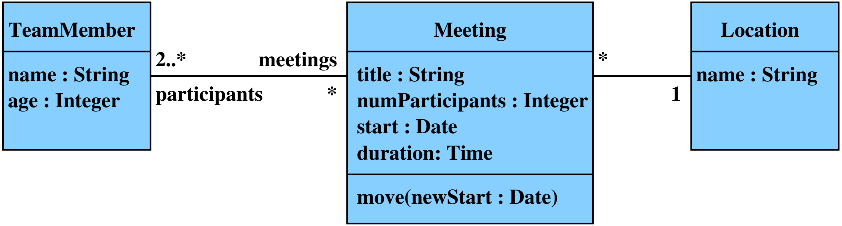
  evaluates *`expression`* for each element of *`collection`* (as context, inaccessible unless named)
- Result has same size as input *`collection`*
  - If input is a set, then result is **bag** (unordered collection with repeated elements)
  - If input is sequence or ordered set, then result is sequence.
- Change to a set using operation `->asSet()`

OCL/Accessing Collection Elements

- Task: Continue navigation from a collection
- The `collect` operation
  - *collection*`->collect(` *expression* `)`
  - *collection*`->collect(` *v* `|` *expression* `)`
  - *collection*`->collect(` *v* `:` *Type* `|` *expression* `)`

  evaluates *expression* for each element of *collection* (as context, inaccessible unless named)
- Result has same size as input *collection*
  - If input is a set, then result is **bag** (unordered collection with repeated elements)
  - If input is sequence or ordered set, then result is sequence.
- Change to a set using operation `->asSet()`
- Shorthands
  - *col.attribute* for *col*`->collect(`*attribute*`)`
  - *col.op* (*args*) for *col*`->collect(`*op* (*args*)`)`

OCL/Accessing Collection Elements



- context TeamMember
  - inv: meetings.start = meetings.start->asSet()->asBag()

OCL/Iterator Expressions

- Task:
  - Examine a collection
  - Define a subcollection
- Tool: the `iterate` expression
  $source$`->iterate(`it`;` res `=` $init$ `|` $expr$`)`
- Value:

  $(\mathtt{Set}\ \{\})\mathtt{->iterate}$
  $\quad(it\ ;\ res = init\ |\ expr)$
  $\quad= init$

  $(\mathtt{Set}\ (\{\mathrm{x1}\} \cup M))\mathtt{->iterate}$
  $\quad(it\ ;\ res = init\ |\ expr)$
  $\quad= (\mathtt{Set}\ M)\mathtt{->iterate}$
  $\qquad(\ it$
  $\qquad;\ res = expr[it = \mathrm{x1},\ res = init]$
  $\qquad|\ expr)$

OCL/Iterator Expressions/Predefined

`exists` : there is one element that makes *body* true

```
source->exists(it|body) =
source->iterate(it;r=false|r or body)
```

`forAll` : all elements make *body* true

```
source->forAll(it|body) =
source->iterate(it;r=true|r and body)
```

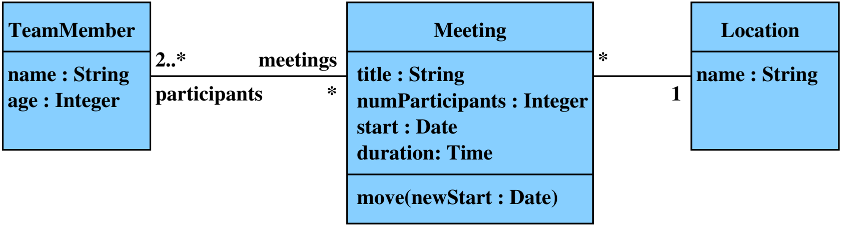
`select` : subset where *body* is true

```
source->select(it|body) =
source->iterate(it;r=Set{}|
                        if body
                        then r->including(it)
                        else r
                        endif)
```

OCL/Iterator Expressions/Predefined/2

- Shorthand with implicit variable binding: `source->select(body)`
- Further iterator expressions
  - On Collection: closure, exists, forAll, `isUnique`, `any`, `one`, collect
  - On Set, Bag, Sequence, OrderedSet: select, `reject`, `collectNested`, `sortedBy`

OCL/Iterator Expressions/Examples




```
context TeamMember
inv: meetings->forAll (m1
       | meetings->forAll (m2
       | m1<>m2 implies disjoint (m1, m2)))
def: disjoint (m1 : Meeting, m2 : Meeting) : Boolean =
     (m1.start + m1.duration <= m2.start) or
     (m2.start + m2.duration <= m1.start)
```

- def: extends `TeamMember` by `<<OclHelper>>` operation

OCL/OclAny, OclVoid, OclInvalid, Model Elements

- OclAny is supertype of the UML model types and all primitive types (**not** of collection types)
- OclInvalid is subtype of every type
  - single instance invalid
  - any operation applied to invalid yields invalid (except `oclIsUndefined()`)
- OclVoid contains an additional error value, null
- OclModelElement enumeration with a literal for each element in the UML model
- OclType enumeration with a literal for each classifier in the UML model
- OclState enumeration with a literal for each state in the UML model

OCL/Operations on OclAny

- `= (obj : OclAny) : Boolean`
- `<> (obj : OclAny) : Boolean`
- `oclIsNew() : Boolean`
- `oclIsUndefined() : Boolean`
- `oclAsType(typeName : OclType) : T`
- `oclIsTypeOf(typeName : OclType) : Boolean`
- `oclIsKindOf(typeName : OclType) : Boolean`
- `oclIsInState(stateName : OclState) : Boolean`
- `allInstances() : Set(T)` must be applied to a classifier with finitely many instances
- = and <> also available on OclModelElement, OclType, and OclState

OCL/Operations on OclAny/KindOf vs TypeOf

Suppose that Student is a subclass of Person and that Course is a separate, unrelated class

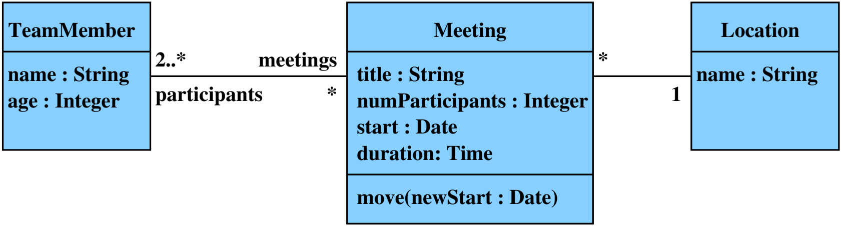
```
context Student inv:
oclIsKindOf (Person)        -- true
oclIsTypeOf (Person)        -- false
oclIsKindOf (Student)       -- true
oclIsTypeOf (Student)       -- true
oclIsKindOf (Course)        -- false
```

OCL/Operations on OclAny/oclAsType

`obj.oclAsType (type: OclType) : type`

- analogous to explicit type cast in Java
- `obj`'s static type becomes `type`
- the expression evaluates to the object denoted by `obj` if `obj.oclIsKindOf(type : OclType)` is true,
- the expression is undefined otherwise.

OCL/Operations on OclAny/Examples




```
context Meeting inv:
  title = "general assembly" implies
    numParticipants = TeamMember.allInstances()->size()
```

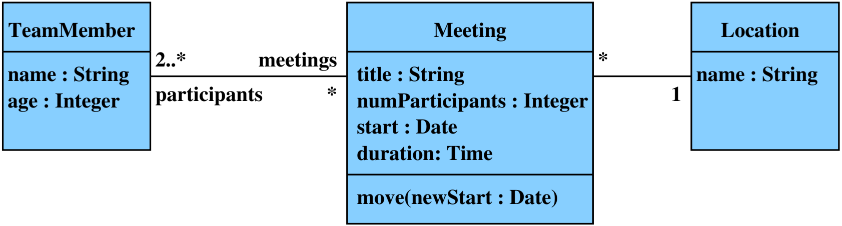
OCL/Pre- and Postconditions

Specification of operations by

context *Type::operation(param1 : Type1, ...): ReturnType*
pre *parameterOk*: `param1 > self.prop1`
post *resultOk* : `result = param1 - self.prop1@pre`

- pre precondition with optional name *parameterOk*
- post postcondition with optional name *resultOk*
- `self` receiver object of the operation
- `result` return value of the operation
- `@pre` accesses the value **before** executing the operation
- body: *expression* defines the result value of the operation
- pre, post, body are optional

OCL/Pre- and Postconditions/Examples




```
context Meeting::move (newStart : Date)
pre: Meeting.allInstances()->forAll (m |
        m<>self implies
          disjoint(m, newStart, self.duration))
post: self.start = newStart
```

OCL/Pre- and Postconditions/Examples/2

```
context Meeting::joinMeeting (t : TeamMember)
pre:  not (participants->includes(t))
post: participants->includes(t) and
       participants->includesAll (participants@pre)
```

OCL/Summary

- OCL is the UML-endorsed way of expressing invariants and other logical formulae on UML diagrams
- Used for specifying constraints that cannot (easily) be expressed by the diagrams
- Makes precise the intuitive meaning of the diagrams
- Facilitates
  - generation of simulations and tests
  - consistency checks
  - code generation, *e.g.*, MDA tools (model driven architecture)