### What is OCL?

- **OCL** = object constraint language
- standard query language of UML 2
- specify expressions and constraints in
  - object-oriented models
  - object modeling artifacts
- specification edited by OMG:
  [http://www.omg.org/spec/OCL/2.2/](http://www.omg.org/spec/OCL/2.2/)

### OCL/Expressions and Constraints

#### Expressions
- initial values, derived values
- parameter values
- body of operation (no side effects ⇒ limited to queries)
- of type: Real, Integer, String, Boolean, 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
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 acht ...'
- Collection types: 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, toLowerCase, concat(), substring()
- ... and many more

context TeamMember inv: age > 0
context Meeting inv: duration > 0
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′), Bag(t′), and Sequence(t′) as subtypes where t′ is a subtype of t
- Set(t′)
  Mathematical set (no duplicate elements, no order)
- Bag(t′)
  Like a set, but may contain duplicates
- Sequence(t′)
  Like a bag, but the elements are ordered
▶ context Meeting
  ▶ self.location yields the associated Location object
  ▶ self.participants yields set of TeamMember objects

▶ If navigation yields object, then use
  ▶ attribute notation
  ▶ navigation
  ▶ operation calls
to continue

▶ What if navigation yields a collection?

▶ Collection operations:
  ▶ notation \( collection\rightarrow op(args) \)
  ▶ example operations: size(), isEmpty(), notEmpty(), ...

▶ Single objects may also be used as collections
▶ Attributes, operations, and navigation of elements not directly accessible

▶ 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, optionally named)

- Result is bag (unordered collection with repeated elements); same size as original collection
- 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, optionally named)

- Result is bag (unordered collection with repeated elements); same size as original collection
- Change to a set using operation ->asSet()
- Shorthands
  - col.attribute for col->collect(attribute)
  - col.op (args) for col->collect(op (args))

OCL/Iterator Expressions

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

- Value:

  (Set {})->iterate
  (it ; res = init | expr)
  = init

  (Set ({x1} ∪ M))->iterate
  (it ; res = init | expr)
  = (Set M)->iterate
  ( it
  ; res = expr[it = x1, res = init]
  | expr)

context TeamMember

- inv: meetings.start = meetings.start->asSet()->asBag()
OCL/Iterator Expressions/Predefined

exists : there is one element that makes \textit{body} true

\[
\text{source}\rightarrow\text{exists}(\text{it} | \text{body}) = \text{source}\rightarrow\text{iterate}(\text{it}; \text{r=false} | \text{r or body})
\]

forAll : all elements make \textit{body} true

\[
\text{source}\rightarrow\text{forAll}(\text{it} | \text{body}) = \text{source}\rightarrow\text{iterate}(\text{it}; \text{r=true} | \text{r and body})
\]

select : subset where \textit{body} is true

\[
\text{source}\rightarrow\text{select}(\text{it} | \text{body}) = \text{source}\rightarrow\text{iterate}(\text{it}; \text{r=Set} | \text{if body then r->including(it) else r endif})
\]

Shorthand with implicit variable binding: \text{source}\rightarrow\text{select(\text{body})}

Further iterator expressions

- On Collection: exists, forAll, isUnique, any, one, collect
- On Set, Bag, Sequence: select, reject, collectNested, sortedBy

OCL/OclAny, OclVoid, Model Elements

- \textit{OclAny} is supertype of the UML model types and all primitive types (not of collection types)
- \textit{OclVoid} is subtype of every type
- single instance \textit{OclUndefined}
- any operation applied to \textit{OclUndefined} yields \textit{OclUndefined} (except \textit{oclIsUndefined()})
- \textit{OclModelElement} enumeration with a literal for each element in the UML model
- \textit{OclType} enumeration with a literal for each classifier in the UML model
- \textit{OclState} enumeration with a literal for each state in the UML model

def: extends TeamMember by <<OclHelper>> operation
OCL/Operations on OclAny

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

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

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

OCL/Operations on OclAny/Examples

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

```
context Location

<table>
<thead>
<tr>
<th>TeamMember</th>
<th>.Meetings</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String, age : Integer</td>
<td>2..* meetings</td>
</tr>
<tr>
<td>participants</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>title : String, numParticipants : Integer, start : Date, duration : Time</td>
</tr>
<tr>
<td>move(newStart : Date)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String</td>
</tr>
</tbody>
</table>
```

PT (Univ. Freiburg) Software Engineering SS 2011 21 / 28
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

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)