Outline

Introduction to OCL
  • History and Background
  • Types in OCL
  • Referring to UML Model
  • Context of OCL Expression
  • Examples
  • Tool (USE)

OCL – Object Constraint Language
OCL - Object Constraint Language

- OCL is about objects
- OCL is a language
- OCL is not about constraints

OCL is the
UML expression / query language

UML related to OCL

- UML is a visual language:
  - Class and object diagrams
  - Use case diagram
  - Sequence and collaboration diagrams
  - Deployment and component diagrams
  - Statechart and activity diagrams
  - UML 2.0 ➔ even more models and details

- OCL is a textual language:
  - Used as an add-on language for UML
History of OCL

- Influenced by Z
- Language for Business Modelling
  - Was defined by IBM for insurance applications
- IBM/ObjecTime OMG submission
- Part of UML standard
- Extended and formally defined for UML 2.0

OCL Characteristics

- Ease-of-use
  - Readable and writeable by wide audience
- Pure expression language
  - Side-effect free
- Specification language
  - Declarative and implementation independent
- Object modelling language
  - Expressiveness in terms of UML object models
- Formality
  - Unambiguous
Object Constraint Language

- The OCL is a language of typed expressions
- An OCL expression is valid if it is written according to the rules (formal grammar) of OCL
- A constraint is a valid OCL expression of type Boolean
- A constraint is a restriction on one or more values of (part of) an object-oriented model or system.

Example for OCL

```
self.transaction -> forall(t:Transaction | t.value > 100)
```

```
Account 1
  0..*
  Transaction

Super-Saver Account

self.balance > 0
```
**Design by contract (Meyer – Eiffel)**

```plaintext
put (element: T, key: STRING) is
  -- insert element with given key
  require
    count < capacity
  do
    .. insertion algorithm ...
  ensure
    count <= capacity;
    item (key) = element;
    count = old count + 1
  end --put
```

<table>
<thead>
<tr>
<th>Obligations</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td><strong>Contractor</strong></td>
</tr>
<tr>
<td>Call put only on a</td>
<td>Insert x so that it</td>
</tr>
<tr>
<td>non-full table</td>
<td>may be retrieved through key</td>
</tr>
<tr>
<td>Get modified table in which</td>
<td>No need to deal with the case in which the</td>
</tr>
<tr>
<td>x is associated with key</td>
<td>table is full before insertion</td>
</tr>
</tbody>
</table>

**OCL Syntax**

- Each expression is evaluated in the context of an object
  ```plaintext
  context TypeName
  ```

- The context can be named
  ```plaintext
  context c : TypeName
  ```
OCL Syntax

- You refer to the object using the keyword “self”
  
  ```
  self.numberOfEmployees > 50
  ```

- Or you use the context name if you are working in a named context
  
  ```
  c.numberOfEmployees > 50
  ```

OCL Syntax

- The expression can specify an invariant of the object
  
  ```
  context Company inv:
  ```

- Or it can specify pre- and post-conditions
  
  ```
  context T_name::op_name(par1: T1,..): Ret
  pre: par1 > ...
  post: result = ...
  ```
OCL Syntax

- You can set also a package context

```ocl
to do

package Package::SubPackage
```

Types in OCL

The types in OCL are as follows:

- **Predefined types**
  - Basic types - `Integer`, `Real`, `String` and `Boolean`
  - Collection types - `Collection`, `Set`, `Bag`, `Sequence`
- **Meta types**
  - `OclAny`, `OclExpression`, `OclType`
- **User-defined model types**
  - All classes, types and interfaces from UML diagrams (UML Classifiers).
OCL Operations

- Boolean
  - and or xor not implies if-then-else
- Integer
  - * + - / abs()
- Real
  - * + - / floor()
- String
  - toUpper() concat()

Collection Types

- Set
  - A mathematical set, elements are unique and not ordered
- Bag
  - A set, but may contain duplicates
- Sequence
  - A bag with elements ordered
- OrderedSet (New in next revision)
  - A set with elements ordered
Collection Types

Collection
- size() : Integer
- sizeEmpty() : Boolean
- isEmpty() : Boolean
- sum() : Integer
- count(object : OclAny) : Integer
- includes(object : OclAny) : Boolean
- excludes(object : OclAny) : Boolean
- including(object : T) : Collection
- excluding(object : T) : Collection
- select(expr : OclExpression) : T (Set, ...)
- reject(expr : OclExpression) : T (Set, ...)
- collect(expr : OclExpression) : Bag(T)
- exists(expr : OclExpression) : Boolean
- forAll(expr : OclExpression) : Boolean
- one(expr : OclExpression) : Boolean
- any(expr : OclExpression) : Boolean
- asSet() : Set(T)
- asBag() : Bag(T)
- asSequence() : Sequence(T)
- asUnique() : Unique(T)
- sortedBy(expr : OclExpression) : Sequence(T)
- includesAll(c2 : Collection(T)) : Boolean
- excludesAll(c2 : Collection(T)) : Boolean

Sequence
- append() : T
- append() : T
- at() : T
- subSequence() : Sequence(T)
- first() : T
- last() : T
- asSet() : Set(T)
- asBag() : Bag(T)

Set
- union() : Set(T)
- intersection() : Set(T)
- asSequence() : Sequence(T)
- asBag() : Bag(T)

Bag
- union() : Bag(T)
- intersection() : Bag(T)
- asSequence() : Sequence(T)
- asSet() : Set(T)
Collection Types

**Sequence**
- `append(object : T) : Sequence(T)`
- `deque(object : T) : Sequence(T)`
- `at(i : Integer) : T`
- `subSequence(lower : Integer, upper : Integer) : Sequence(T)`
- `first() : T`
- `last() : T`
- `asSet() : Set(T)`
- `asBag() : Bag(T)`

**Set**
- `union(s2 : Set(T)) : Set(T)`
- `intersection(s2 : Set(T)) : Set(T)`
- `asSequence() : Sequence(T)`
- `asBag() : Bag(T)`

**Bag**
- `union(b2 : Bag(T)) : Bag(T)`
- `intersection(b : Bag(T)) : Bag(T)`
- `asSequence() : Sequence(T)`
- `asSet() : Set(T)`
Predefined OCL Types

OclType:

- All types defined in a UML model, or pre-defined within OCL, have a type. This type is an instance of the OCL type called OclType.

- **type**: instance of OclType
  - type.name : String
  - type.attributes : Set(String)
  - type.associationEnds : Set(String)
  - type.operations : Set(String)
  - type.supertypes : Set(OclType)
  - type.allSupertypes : Set(OclType)
  - type.allInstances : Set(type)

OclAny:

Within the OCL context, the type OclAny is the supertype of all types in the model and the basic predefined OCL type. The predefined OCL Collection types are not subtypes of OclAny.

- **object**: instance of OclAny
  - object=(object2:OclAny)
  - object<>(object2:OclAny) : Boolean
  - objectoclType : OclType
  - object.oclIsKindOf(type:OclType) : Boolean
  - object.oclAsType(type : OclType) : type
  - object.oclInState(state : OclState) : Boolean
  - object.oclIsNew() : Boolean
Predefined features on all objects (OclAny)

- Type of an object
  \texttt{oclType : OclType}
  Feature \texttt{oclType} results in type of an object
- Direct type
  \texttt{oclIsTypeOf(t:OclType):Boolean}
- Direct or super type
  \texttt{oclIsKindOf(t:OclType):Boolean}

Examples

- **context** Person **inv:**
  \texttt{self.oclType}
  \texttt{results in Person}

- **context** Person **inv:**
  \texttt{self.oclIsTypeOf(Person)--true}
  \texttt{self.oclIsTypeOf(Company)--false}
Let Expression

- Allows local reuse

```plaintext
context Person inv:
  let income : Integer = self.job.salary->sum()
  let hasTitle(t : String) : Boolean =
      self.job->exists(title = t)
in
  if isUnemployed then
    self.income < 100
  else
    self.income >= 100 and self.hasTitle('manager')
endif
```

Definition Constraint

- Allows global reuse

```plaintext
context Person def:
  let income : Integer = self.job.salary->sum()
  let hasTitle(t : String) : Boolean =
      self.job->exists(title = t)
```
OCL-Type Hierarchy

- **OCL-Typierarchie**

---

**UML Model Predefined OCL Types**

- OclAny
- OclExpression
- OclState
- Enumeration
- Boolean
- String
- Real
- Integer
- Collection
- Set
- Bag
- Sequence

---

Referring to UML Model Info

- Many properties expressed in UML models can be referred to from OCL:
  - attributes
  - operations
  - navigation are derived from associations
  - class attributes and operations
  - states

- Referring to UML Model Info
OCL Example

context Person inv:
  self.wife->notEmpty() implies self.wife.age >= 18 and
  self.husband->notEmpty() implies self.husband.age >= 18

Married people are of age >= 18

context Company inv:
  self.employee->size() <= 50

A company has at most 50 employees

Referring to UML Model Info (Attributes, Methods)

- Class1.Attribute
  refers to the values of the attributes of the class
- Class1.Method()
  refers to the results of Method() of the class
- Assuming Attribute is of type integer one can specify the following constraint for objects:
  Context Class1
  Inv: self.Attribute <= 100
Referring to UML Model Info (Associations)

• **context** Class1 **inv:**
  - self.Class2 is a sequence of objects
  - self.Class2.Class3 is a bag of objects

• **context** Class2 **inv:**
  - self.Class3 is a set of objects

• **context** Class1 **inv:**
  - self.Class2 is a set of objects
  - self.Class2.Class3 is a bag of objects

Referring to UML Model Info

• **context** Class1 **inv:**
  - self.Class2 is an object
  - self.Class2.Class3 is an object

• **context** Class2 **inv:**
  - self.Class3 is an object

• **context** Class1 **inv:**
  - self.Class2 is a set of objects
  - self.Class2.Class3 is a bag of objects
The context of an OCL expression

context Software Engineer inv:
self.Project->size() <= 10

context Project inv:
selfSOFTWARE ENGINEER <= 100

Can you express the same information by multiplicity?

Example in Context of Customer

Context Customer inv:
self.age>=0 and self.age< 150
self.age <18 implies self.salary=0
self.age <18 implies self.title=""
The Implication Operation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A implies B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Example in Context of University

context University inv:
self.member -> includes( self.president )
self.member -> includesAll(self.professor)
Example in Context of Account

**Transaction**

<table>
<thead>
<tr>
<th>Account</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value : Real</td>
</tr>
<tr>
<td></td>
<td>Source : Account</td>
</tr>
<tr>
<td></td>
<td>Destination : Account</td>
</tr>
</tbody>
</table>

**context** Account **inv:**

self.transaction -> select( value > 500 )

**context** Account **inv:**

self.transaction -> reject( not(value > 500 ))

**context** Account **inv:**

self.transaction->collect(value) -> sum()

**context** Account **inv:**

self.transaction.value -> sum()

---

Example in Context of Class

**Constraint**

**context** Class1 **inv:**

Class1.allInstances

-> select(oclType = Class1) -> isEmpty

specifies Class1 as an abstract Class.
The 4-Level Meta-Architecture

M3  Meta-Meta-Meta-Data or Meta-Meta-Model Level

M2  Meta-Meta-Data or Meta-Model Level

M1  Meta-Data or Model Level

M0  Instance Level

Relationship to UML

- The UML Metamodel is a M2 model
- Your model using UML is a M1 model
- The instantiations of your UML classes (UML Objects) are at M0
- The (M3) MOF Model is used to define the UML Metamodel
UML is defined using UML

- UML is defined in a subset of UML which is consistent with MOF

- All elements are defined using
  - Class diagrams
  - OCL constraints
  - English language informal information
UML Metamodel

UML Core Package
UML Core: Classifiers

UML Relationships
UML Extension Mechanism

UML Actions
**UML - OCL**

- **UML is specified by UML and OCL**
  - e.g. `GeneralizableElement`

  2.5.3.20 `GeneralizableElement` (page 2-60 v.1.4 Sept. 2001)

  1. A root cannot have any Generalizations.
     ```
     self.isRoot implies self.generalization->isEmpty
     ```

  2. No `GeneralizableElement` can have a parent
     Generalization to an element that is a leaf.
     ```
     self.parent->forall(s | not s.isLeaf)
     ```
UML - OCL

UML is specified by UML and OCL
e.g. Interface

2.5.3.23 Interface (page 2-61 v.1.4 Sept. 2001)

[1] An Interface can only contain Operations.
   self.allFeatures->forAll(f | f.oclIsKindOf(Operation) or f.oclIsKindOf(Reception))
   • allFeatures =
     self.feature->union(self.parent.oclAsType(Classifier).allFeatures)

   self.allContents->isEmpty
   • allContents =
     self.contents->union( self.parent.allContents->select(e | e.elementOwnership.visibility = #public or
e.elementOwnership.visibility = #protected))
   • contents = self.ownedElement -> union(self.namespace, contents)

[3] All Features defined in an Interface are public.
   self.allFeatures->forAll ( f | f.visibility = #public )

Tool Support

USE (UML - based Spezification Environment)
UML - OCL

Peter Forbrig

association WorksOn between
EmployeeTrole employee, ProjectTrole project
end

context Project tmo BudgetWithinDepartmentBudget:
{self.budget <= self.department.budget}
context Employee inv MoreProjectsHigherSalary:
    Employee.allinstances->forall(e1, e2 : Employee | ((e1.project->size > e2.project->size)
        implies (e1.salary > e2.salary)))
State of the system can be saved

Script of executed commands
Script can be executed

Layout can be saved too
Change of Multiplicity

from "*" to "1..*"

association WorksOn between
  Employee[1..*];
  Project[1..*];
end

Multiplicity constraint violation in association "WorksOn"
  Object "Martin" is connected to 0 object of class "Project"
  but the multiplicity is specified as "1..*"
Evaluate OCL expression

Enter OCL expression:
```ocl
cs.project->select(p | p.employee->exists(e | e.name = 'Peter'))
->collect(c | c.name)
```

Result:
```plaintext
Set([oo2], Project)
```
UML - OCL

Peter Forbrig

Evaluate OCL expression

Enter OCL expression:

```
cs.project->select(c|c.employee->exists(e|e.name='Peter')
   ->collect(c.employee.name))
```

Result:

```
Bag(p2): Bag(String)
```

---

Evaluate OCL expression

Enter OCL expression:

```
Project.allInstances->forall(p1,p2|p1<=p2 implies p1.name=p2.name)
```

Result:

```
true: Boolean
```
Evaluate OCL expression

Enter OCL expression:

Project allInstances = select (p | p.budget >= 0)

Result:

Set1 : Set[Project]
Evaluate OCL expression

Enter OCL expression

Project all instances -> forAll p1, p2 | p1 <> p2 implies
  p1.employee -> intersection(p2.employees) -> size = 0

Result
true : Boolean