Software Engineering (OCL)
Object Constraint Language

Lina YE

https://www.lri.fr/~linaye/GL.html
lina.ye@centralesupelec.fr
Sequence 3, 2017-2018
1 Introduction

2 Constraints
   - Context and Self
   - Invariant
   - Pre- and Post-condition
   - Constraints on Attributes

3 Language
   - Access to characteristics
   - Types
   - Variable
   - Collections
   - OCL function

4 Example
Motivation

Why OCL

- Cannot represent all the relevant aspects of a specification (e.g., class diagram)
- Need to describe additional constraints without ambiguities
- Formal languages requires a strong mathematical background

What is OCL

- **Formal language** to express constraints, that remains easy to read and write
- Developed by IBM and standardized by OMG
- Integrated into the UML standard
Objet Constraint Language

Where to use

- Specify **invariants** on classes and types in the class model
- Describe **pre- and post-conditions** on operations and methods
- Describe **guards**
- As a navigation language
- etc.

Example

- How to represent the constraint that the age of an employee **cannot be smaller** than 18?
Pure specification language: **do not have side effect**
- evaluation of an OCL expression returns a value
- this evaluation **do not alter** the system state and is instantaneous
Objet Constraint Language

- Pure specification language: do not have side effect
  - evaluation of an OCL expression returns a value
  - this evaluation do not alter the system state and is instantaneous

- Not a programming language
  - cannot write program logic in OCL
  - cannot invoke processes or activate non-query operations
Objet Constraint Language

- Pure specification language: do not have side effect
  - evaluation of an OCL expression returns a value
  - this evaluation do not alter the system state and is instantaneous

- Not a programming language
  - cannot write program logic in OCL
  - cannot invoke processes or activate non-query operations

- Typed language: each expression has a type
  - expression must obey the type conformance rules of OCL
  - each classifier defined in a UML model represents a distinct OCL type
  - includes a set of supplementary predefined types
Context

- Each constraint must be associated to one model element
- Such an element constitutes the context of the constraint
- Syntax: keyword context

example

- class: nameClass
  context Person

- operation: nameClass::nameOperation(param1: Type1,...):TypeReturned
  context Account::getSolde(): Real

- attribute: nameClass::nameAtt: TypeAtt
  context Person::age : Integer
In an OCL expression, reserved word **self** is used to refer to the contextual instance.

If the context is **Person**, then **self** refers to an instance of **Person**.

This keyword can be **omitted** when the context is clear.

**Example**

- context Person
  - self.name
- context Person
  - name
**Invariant**

- Determine a constraint that should be always true for all instances of a type

- Syntax
  
  inv: <logic expression>

**example**

- Value of attribute nbEmployees in instances of Company must be less than or equal to 50
  
  ```
  context Company
  inv: self.nbEmployees ≤ 50
  ```

- The stock price of each company is greater than 0 (stockPrice() is a operation defined in the class Company)
  
  ```
  context Company
  inv: self.stockPrice() > 0
  ```
Pre-condition

- Constraints associated with an operation or other behavioral feature
- Constraint assumed to be true before the execution of the operation
- Syntax
  pre: <logic expression>

Example

- The age of a person who has an income must be older than or equal to 18 (income() is an operation defined in the class Person)
  context Person:: income(): Integer
  pre: self.age ≥ 18
Post-condition

- Constraints associated with an operation or other behavioral feature
- Constraint satisfied after the execution of the operation
- Keyword result denotes the value returned by the operation, whose type is the returned type
- Keyword @pre denotes the attribute value before the operation
- Syntax
  
  \[
  \text{post: <logic expression>}
  \]

**Example**

- The age of a person who has an income cannot be smaller than 18, and the income must be less than 5000

```
context Person:: income(): Integer
pre: self.age \geq 18
post: result < 5000
```
Reserved word **init** is used to represent the initial value

Possibility to precise the initial value of an attribute or an association end when the object is created

**Syntax**

```
init: expression
```

**Example**

Attribute **isMarried** in **Person** is initialized to **false**

context Person:: isMarried: Boolean
init: false
Reserved word *derive* is used to represent the derived value

Precise how to obtain the derived value of an attribute based on the value of other attributes, such a constraint should always be respected

**Syntax**

`derive: expression`

**Example**

The age of one person is obtained by subtracting their birth date from the current date

```ocl
context Person:: age: Integer
derive: currentDate-dateOfBirth
```
Access to characteristics of an object

- The access of attributes and operations of an object is specified by a dot followed by their name.

- Syntax
  
  ```
  self.nameAttribute
  self.nameOperation(\textit{arg}_1, \ldots, \textit{arg}_n)
  ```

**Example**

```
context Person
self.age
self.income()
```
Navigation

- From an object, an association is navigated by a dot followed by the opposite role name.

- Value of expression depends on maximal multiplicity of the association end:
  1: value is an object ("." ) or can also be used as a set containing a single object ("→")
  ∗: value is a set of objects ("→")

- For optional associations, it is useful to check whether there is an object or not when navigating the association.
context Company

inv: self.manager.isUnemployed=false

inv: self.employee→notEmpty()
context Company
inv: self.manager.isUnemployed=false
inv: self.employee→notEmpty()

self.manager is an object of type Person
context Company
inv: self.manager.age> 40

self.manager as a set
context Company
inv: self.manager→size()=1
Example

context Person

inv: self.wife→notEmpty() implies
self.gender=Gender::Male and
self.husband→ notEmpty() implies
self.gender=Gender::Female
List of Types

Predefined types with their operators

<table>
<thead>
<tr>
<th>Types</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>and; or; xor; not; implies; if-then-else-endif;...</td>
</tr>
<tr>
<td>Integer</td>
<td>*; +; −; /; abs(); ...</td>
</tr>
<tr>
<td>Real</td>
<td>*; +; −; /; abs(); floor(); ...</td>
</tr>
<tr>
<td>String</td>
<td>concat(s: String); size(); substring(lower: Integer, upper: Integer);...</td>
</tr>
</tbody>
</table>

Implies

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P1 implies P2</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>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Operations on Types

- One person who is married must be more than 18 years old
- One person is either male or female but cannot be both
Operations on Types

- One person who is married **must be** more than 18 years old
- One person is **either** male or female but cannot be both

context Person

inv: self.isMarried implies self.age > 18
Operations on Types

- One person who is married **must be** more than 18 years old
- One person is **either** male **or** female but cannot be both

```ocl
context Person
inv: self.isMarried implies self.age > 18
```

```ocl
context Person
inv: self.gender=Gender::male xor self.gender=Gender::female
```
Create a Variable

- If the same expression is used more than one time
- For the better readability of the constraint

**Syntax**

- With keywords **let...in**
  ```plaintext
  let < variable >: TypeVar=< request > in
  < expression >
  ```

- With keyword **def**
  ```plaintext
  def: < variable >: TypeVar=< request >
  < expression >
  ```
Create a Variable

example

- An unemployed person has no job. Otherwise, he has at least one job.

context Person

inv: let numberOfJobs: Integer=let numberOfJobs: Integer=let numberOfJobs: Integer=self.job→size() in
if isUnemployed then numberOfJobs=0
else numberOfJobs > 0
endif
Create a Variable

example

- An unemployed person has no job. Otherwise, he has at least one job.
- The name of one person is the concatenation of first name and last name

context Person
inv: let numberJobs: Integer=self.job→size() in
if isUnemployed then numberJobs=0
else numberJobs > 0
endif

countext Person
def: name: String=self.firstName.concat(‘ ‘).concat(lastName)
Conversion of an association to a type within OCL

- Collection
- Association
- Role B
- cardinality
- Set
- Ordered Set
- Access to characteristics
- Types
- Variable
- Collections
- OCL function
**Operation on Collection**

**Syntax**

- For a collection, $\rightarrow$ is used to apply some operation on it: nameCollection$\rightarrow$ operation()
- Recall: the dot is used for the access of a property of an object

**Some examples**

- $\text{size()}: \text{Integer}$ (return the number of elements in the collection)
- $\text{includes(object: T): Boolean}$ (return true if object is included in the collection)
- $\text{excludes(object: T): Boolean}$ (return true if object is not in the collection)
Some examples

- `count(object: T): Integer` (return the number of object)
- `isEmpty(): Boolean` (return true if the collection is empty)
- `notEmpty(): Boolean` (return true if the collection is not empty)
- `includesAll(c: Collection(T)): Boolean` (return true if the collection contains all elements of c)
- `excludesAll(c: Collection(T)): Boolean` (return true if the collection does not contain any element of c)
- `sum(): T` (return the sum of all elements in the collection)
- `union(set: Set(T)): Set(T)` (return the union of self with set)
Operation on Collection

Some examples

- any(exp: OclExpression): Type (return any element in self validating exp)
- =(set: Set(T)): Boolean (return true if self and set contain exactly the same elements)
- including(object: T): Set(T) (return a collection that contains all elements of self plus object)
Operation on Collection

- One company has at least one employee

- context Company
  inv: self.employee → size() > 0
  inv: self.employee → notEmpty()
Operation on Collection

- One company has at least one employee
- The manager of a company is also an employee

context Company
inv: self.employee \rightarrow size() > 0
inv: self.employee \rightarrow notEmpty()

context Company
inv: self.employee \rightarrow includes(self.manager)
Operation on Elements

**Syntax**

- `collection → operation (expression)`
- `collection → operation (v | expression-with-v)`
- `collection → operation (v: Type | expression-with-v)`

The expression is applied on **each element** of the collection.
**Operation on Elements**

- **select**: generate a sub-collection that contains only the elements **satisfying** expression

Each company must have at least one employee that is more than 50 years old

- **context Company**:
  
  inv: self.employee → **select**(p: Person | p.age > 50) → notEmpty()
Operation on Elements

- **select**: generate a sub-collection that contains only the elements **satisfying** expression
- **reject**: generate a sub-collection that contains only the elements that **does not satisfy** expression

Each company must have at least one employee that is more than 50 years old

- **context** Company:
  - **inv**: self.employee → **select** (p: Person | p.age > 50) → notEmpty()
- **context** Company:
  - **inv**: self.employee → **reject** (p: Person | p.age <= 50) → notEmpty()
Operation on Elements

- **forAll**: return true if the expression is true for each element

Each company must have at least one employee that is more than 50 years old

- context Company:
  - inv: self.employee → not (forAll(p: Person | p.age ≤ 50))
Operation on Elements

- **forAll**: return true if the expression is true for each element
- **exists**: return true if the expression is true for at least one element

Each company must have at least one employee that is more than 50 years old

- **context Company**:
  - inv: self.employee → not (forAll(p: Person | p.age≤ 50))
- **context Company**:
  - inv: self.employee → exists(p: Person | p.age> 50)
The employer of an employee that participates a team project is the organisation that possesses this project.

- context Person:
  - inv: (self.employer→size()=1 and self.team→size()=1)
  - implies self.employer=self.team.project.organisation
Operation on Elements

- **collect**: create a new collection, for which each element is the result of the expression

It is required to obtain the set of birthday dates for all employees

- context Company
  self.employee → **collect** (p: Person | p.birthdayDate)

Shorthand for **collect**

- context Company
  self.employee.birthdayDate
Iterate Operation

- **iterate**: calculate an accumulator whose value is built up during the iteration of the collection.
- Collection → iterate (e: Type; acc: Type=initial expression | expression with e and acc)

The sum of ages of all children for a person

- context Person
  self.children → iterate (p: Person; acc: Integer=0 | acc=acc+p.age)
Re-typing or Casting

- With `ooclAsType(T2)`, one re-types an object `o` of type `T1` into another type `T2`
- Let type `Super` be a super type of type `Sub`
- Allows one to use a property of an object defined on a subtype of the currently known type of the object

```
context Super
inv: selfoclAsType(Sub).p (accesses the p property defined in Sub, valid when actual type of self is Sub, otherwise, invalid)
```

- Can be used to access a property of a superclass

```
context Sub
inv: selfoclAsType(Super).p (accesses the p property defined in Super)
```
Other functions

- `oclIsTypeOf(t: Type):` return true if the type of `self` and `t` are the same
- `oclIsKindOf(t: Type):` return true if `t` is a direct/indirect (supertype) type of `self`
- `oclIsNew:` used in a post-condition, return true if the object has been created during the operation

Example
context Person
inv: self.oclIsTypeOf(Person) –true
inv: self.oclIsTypeOf(Company) –false
Class features

- Features of a **class**, not of its instances
- Either **predefined** or **user-defined**

- **Predefined:** `allInstances` holds on all types and returns the set of class instances
- There are at most 100 persons
  - context `Person`
  - inv: `Person.allInstances() → size() ≤ 100`
- A user-defined feature `averageAge` of class `Person`
  - context `Person`
  - `Person.averageAge=(Person.allInstances() → collect (age) → sum()) / (Person.allInstances()→ size())`
Example 1

Diagram of a hotel and room relationship:
- Hotel
  - address: String
  - floorMin: int
  - floorMax: int
  - income(): float
- Room
  - floor: int
  - number: int
  - nbBeds: int
  - price: float
  - repaint (c: Color)
- Person
  - name: String
  - age: int
- BathRoom
  - floor: int
  - number: int
- Relationship
  - Hotel: 1..* bosses Person
  - Person: * clients Hotel
  - Room: 0..1 BathRoom
  - BathRoom: 0..1 Room
OCL for example_1

A hotel never has a floor 13, because of superstition.
A hotel never has a floor 13, because of superstition.

- context Room
  inv: self.floor <> 13

- context BathRoom
  inv: self.floor <> 13
OCL for example 1

The number of clients for each room must be smaller or equal to the number of beds in the rented room. The children under 4 are not “taken into account” in this calculation rule (condition : maximum of one child under 4 per room).
OCL for example 1

The number of clients for each room must be smaller or equal to the number of beds in the rented room. The children under 4 are not “taken into account” in this calculation rule (condition: maximum of one child under 4 per room).

- context Room
  inv: clients → size() ≤ nbBeds or
  (clients → size() = nbBeds + 1 and clients → exists(p: Person | p.age < 4))
OCL for example _1

Each floor owns at least one room except for floor 13.
Each floor owns at least one room except for floor 13.

- context Hotel
  
  inv: Sequence{floorMin, ..., floorMax} → forall (f: Integer | f <> 13 implies self.room → select(r: Room | r.floor=f) → notEmpty())
Rooms are on the first to the last floor.
OCL for example 1

Rooms are on the first to the last floor.

- context Hotel
  inv: self.room → forall (r: Room | r.floor ≤ self.floorMax and r.floor ≥ self.floorMin)
OCL for example 1

A room can be repainted when it is not occupied. Once repainted, the cost of a room is 10% more.
A room can be repainted when it is not occupied. Once repainted, the cost of a room is 10% more.

- context Room::repaint(c: Color)
  pre: clients → isEmpty()
  post: price=price@pre*1.1
The hotel income is equal to the sum of prices for all rented rooms.
The hotel income is equal to the sum of prices for all rented rooms.

- context Hotel::income():Real
  post: result=self.room \rightarrow \text{select}(r: \text{Room} \mid r.\text{clients} \rightarrow \text{notEmpty}()) \rightarrow \text{collect}(r: \text{Room} \mid r.\text{price}) \rightarrow \text{sum}()
The salary of an employee cannot be greater than the salary of his/her supervisor.

context Employee

inv: self.supervisor → notEmpty() implies self.salary < self.supervisor.salary
OCL for example 2

- The salary of an employee cannot be greater than the salary of his/her supervisor
  context Employee
  inv: self.supervisor → notEmpty() implies
  self.salary < self.supervisor.salary
- The condition notEmpty must be tested since the multiplicity of the role is not mandatory
OCL for example 2

- The SSN of employees is an identifier (or a key) context Employee
  inv: Employee.allInstances() \rightarrow \text{forall}(e1, e2 \mid e1 \not<=> e2 \implies e1.\text{SSN} \not<=> e2.\text{SSN})
OCL for example 2

- The SSN of employees is an identifier (or a key) context Employee
  inv: Employee.allInstances() → forall(e1, e2 | e1 <> e2 implies e1.SSN <> e2.SSN)
- The name and relationship of dependents is a partial identifier: they are unique among all dependents of an employee context Employee
  inv: self.dependents → notEmpty() implies self.dependents → forall(e1, e2 | e1 <> e2 implies e1.name <> e2.name or e1.relationship <> e2.relationship)
An employee cannot supervise him/herself
context Employee
inv: self.subordinates → excludes(self)