The Object Constraint Language (OCL): Specifying constraints in UML models

Robert B. France
What is OCL?

• OCL is
  – a textual language to describe constraints
  – the constraint language used in UML models
    • As well as the UML meta-model

• OCL expressions are always bound to a UML model
  – OCL expressions can be bound to any model element in UML
Diagram with added invariants

```plaintext
{context Flight
  inv: type = Airtype::cargo implies airplane.type = Airtype::cargo
  inv: type = Airtype::passenger implies
    airplane.type = Airtype::passenger}
```
Different kinds of constraints

• Class invariant
  – a constraint that must always be met by all instances of the class

• Precondition of an operation
  – a constraint that must always be true BEFORE the execution of the operation

• Postcondition of an operation
  – a constraint that must always be true AFTER the execution of the operation
Example model

**Airport**
- name: String

**Flight**
- departTime: Time
- /arrivalTime: Time
- duration: Interval
- maxNrPassengers: Integer

**Passenger**
- $minAge: integer
- age: Integer
- needsAssistance: Boolean
- book(f: Flight)

**Airline**
- name: String

**CEO**
- 0..1

**book**
- 0..1
Constraint context and self

• Every OCL expression is bound to a specific context.
  – The context is often the element that the constraint is attached to

• The context may be denoted within the expression using the keyword ‘self’.
  – ‘self’ is implicit in all OCL expressions
  – Similar to `this` in C++
Notation

• Constraints may be denoted within the UML model or in a separate document.
  – the expression:
    context Flight inv: self.duration < 4
  – is identical to:
    context Flight inv: duration < 4
  – is identical to:
    Flight
      duration: Integer
      <<invariant>>
      duration < 4
Elements of an OCL expression

• In an OCL expression these elements may be used:
  – basic types: String, Boolean, Integer, Real.
  – classifiers from the UML model and their features
    • attributes, and class attributes
    • query operations, and class query operations (i.e., those operations that do not have side effects)
  – associations from the UML model
Example: OCL basic types

context Airline inv:
name.toLower = ‘klm’

context Passenger inv:
age >= ((9.6 - 3.5)* 3.1).floor implies
mature = true
Model classes and attributes

• “Normal” attributes
  context Flight inv:
  self.maxNrPassengers <= 1000

• Class attributes
  context Passenger inv:
  age >= Passenger.minAge
Example: Using query operations

context Flight inv:
self.departTime.difference
(self.arrivalTime) .equals(self.duration)

<table>
<thead>
<tr>
<th>Time</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$midnight: Time</td>
<td>nrOfDays : Integer</td>
</tr>
<tr>
<td>month : String</td>
<td>nrOfHours : Integer</td>
</tr>
<tr>
<td>day : Integer</td>
<td>nrOfMinutes : Integer</td>
</tr>
<tr>
<td>year : Integer</td>
<td>equals(i:Interval):Boolean</td>
</tr>
<tr>
<td>hour : Integer</td>
<td>$Interval(d, h, m : Integer) : Interval</td>
</tr>
<tr>
<td>minute : Integer</td>
<td></td>
</tr>
<tr>
<td>difference(t:Time):Interval</td>
<td></td>
</tr>
<tr>
<td>before(t: Time): Boolean</td>
<td></td>
</tr>
<tr>
<td>plus(d : Interval) : Time</td>
<td></td>
</tr>
</tbody>
</table>
Associations and navigations

- Every association in the model is a navigation path.
- The context of the expression is the starting point.
- Role names are used to identify the navigated association.
Example: navigations

context Flight
inv: origin <> destination
inv: origin.name = ‘Amsterdam’

context Flight
inv: airline.name = ‘KLM’
context Person inv:
if employer.name = 'Klasse Objecten' then
    job.type = JobType::trainer
else
    job.type = JobType::programmer
endif
Significance of Collections in OCL

- Most navigations return collections rather than single elements

<table>
<thead>
<tr>
<th>Flight</th>
<th>0..*</th>
<th>1</th>
<th>Airplane</th>
</tr>
</thead>
<tbody>
<tr>
<td>type : Airtype</td>
<td>flights</td>
<td>type : Airtype</td>
<td></td>
</tr>
</tbody>
</table>
Three Subtypes of Collection

• Set:
  – arrivingFlights (from the context Airport)
  – Non-ordered, unique

• Bag:
  – arrivingFlights.duration (from the context Airport)
  – Non-ordered, non-unique

• Sequence:
  – passengers (from the context Flight)
  – Ordered, non-unique
Collection operations

- OCL has a great number of predefined operations on the collection types.
- Syntax:
  - `collection->operation`

Use of the “-” (arrow) operator instead of the “.” (dot) operator
The collect operation

- The *collect* operation results in the collection of the values obtained by evaluating an expression for all elements in the collection.
The collect operation

context Airport inv:
self.arrivingFlights -> collect(airLine) -> notEmpty

departing flights
arriving flights
The collect operation syntax

- Syntax:
  collection->collect(elem : T | expr)
  collection->collect(elem | expr)
  collection->collect(expr)

- Shorthand:
  collection.expr

- Shorthand often trips people up. Be Careful!
The select operation

The *select* operation results in the subset of all elements for which a boolean expression is true

countext Airport inv:
self.departingFlights->select(duration<4)->notEmpty
The select operation syntax

- Syntax:
  
  collection->select(elem : T | expression)
  collection->select(elem | expression)
  collection->select(expression)
The forAll operation

• The forAll operation results in true if a given expression is true for all elements of the collection
Example: forAll operation

class Airport

context Airport inv:
self.departingFlights->forAll(departTime.hour>6)
The `forAll` operation syntax

- Syntax:
  - `collection->forAll(elem : T | expr)`
  - `collection->forAll(elem | expr)`
  - `collection->forAll(expr)`
The exists operation

• The *exists* operation results in true if there is at least one element in the collection for which a given expression is true.
Example: exists operation

context Airport inv:
self.departingFlights->exists(departTime.hour<6)
The exists operation syntax

• Syntax:
  collection->exists(elem : T | expr)
  collection->exists(elem | expr)
  collection->exists(expr)
Other collection operations

- **isEmpty**: true if collection has no elements
- **notEmpty**: true if collection has at least one element
- **size**: number of elements in collection
- **count(elem)**: number of occurrences of elem in collection
- **includes(elem)**: true if elem is in collection
- **excludes(elem)**: true if elem is not in collection
- **includesAll(coll)**: true if all elements of coll are in collection
Local variables

• The *let* construct defines variables local to one constraint:

  Let var : Type = <expression1> in <expression2>

• Example:

  context Airport inv:

  Let supportedAirlines : Set (Airline) =
    self.arrivingFlights -> collect(airLine) in
    (supportedAirlines ->notEmpty) and
    (supportedAirlines ->size < 500)
Iterate

- The *iterate* operation for collections is the most generic and complex building block.

```python
collection->iterate(elem : Type;
    answer : Type = <value> | 
    <expression-with-elem-and-answer>)
```
Iterate example

- **Example iterate:**
  
  context Airline inv:
  flights->select(maxNrPassengers > 150)->notEmpty

- **Is identical to:**
  
  context Airline inv:
  flights->iterate (f : Flight;
  answer : Set(Flight) = Set{ } |
  if f.maxNrPassengers > 150 then
    answer->including(f)
  else
    answer endif )->notEmpty
An Example: Royal and Loyal Model

Taken from “The Object Constraint Language” by Warmer and Kleppe
Defining initial values & derived attributes

context LoyaltyAccount::points
init: 0

context CustomerCard::valid
init: true

context CustomerCard::printedName
Derive: owner.title.concat(‘ ’).concat(owner.name)
**context** LoyaltyProgram

**inv**: partners.deliveredServices -> size() >= 1

**context** LoyaltyProgram

**inv**: partners.deliveredServices ->

forAll(pointsEarned = 0 and pointsBurned = 0)

implies Membership.account -> isEmpty()

*A note on the collect operation*

partners -> collect(numberIOfCustomers)

can also be written as

partners.numberOfCustomers
context Customer
inv: programs -> size() = cards -> select (valid = true) -> size()

context ProgramPartner
inv: numberOfCustomers = programs.participants -> asSet() -> size()
context LoyaltyProgram::getServices
(pp:ProgramPartner:Set(Service)
body: if partners -> includes(pp)
then pp.deliveredServices
else Set{}
endif
Defining new attributes and operations

context LoyaltyAccount
def: turnover :
Real = transactions.amount -> sum()
//Attributes introduced in this manner are always derived attributes

custom LoyaltyProgram
def: getServicesByLevel(levelName:String): Set(Service)
= levels -> select (name = levelName).availableServices ->asSet()
Specifying Operations

**context** LoyaltyAccount::isEmpty():Boolean
**pre:** true
**post:** result = (points = 0)

**context** Customer::birthdayHappens()
**post:** age = age@pre + 1

**context** LoyaltyProgram::enroll(c:Customer)
**pre:** c.name <> ‘ ’
**post:** participants @pre -> including(c)

**context** Service::upgradePointsEarned(amount: Integer)
**post:** calcPoints() = calcPoints@pre() + amount
Inheritance of constraints

• Guiding principle Liskov’s Substitution Principle (LSP):
  – “Whenever an instance of a class is expected, one can always substitute an instance of any of its subclasses.”
Inheritance of constraints

• Consequences of LSP for invariants:
  – An invariant is always inherited by each subclass.
  – Subclasses may strengthen the invariant.

• Consequences of LSP for preconditions and postconditions:
  – A precondition may be weakened (contravariance)
  – A postcondition may be strengthened (covariance)
OCL Tips

• OCL invariants allow you to
  – model more precisely
  – remain implementation independent

• OCL pre- and post-conditions allow you to
  – specify contracts (design by contract)
  – specify interfaces of components more precisely

• OCL usage tips
  – keep constraints simple
  – always give natural language comments for OCL expressions
  – use a tool to check your OCL