The Object Constraint Language (OCL)

Robert B. France
Dept. of Computer Science
Colorado State University
USA
france@cs.colostate.edu
Semantics and UML models

- UML models often treated as informal descriptions
  - Useful if you use UML as a sketching language – this is not the focus of the course
  - Focus is on using as a formal language that can be used to create machine analyzable models
- UML models can be treated formally
  - Necessary if we are to use UML as a software engineering language
Defining semantics

Three key concepts

- **Syntactic domain**: Syntactic elements of the language (e.g., class symbol)

- **Semantic domain**: Elements representing meaningful concepts described by statements in the language (e.g., objects)

- **Semantic mapping**: Mapping of syntactic elements to semantic elements; the semantic elements denote the meaning of the syntactic elements that are mapped to it
Semantics of class models

- A class model characterizes a set of valid object configurations
- Syntactic domain: UML class diagram notation (e.g., class, association)
- Semantic domain: Object configurations

Example:
- A class is a set of objects
- An abstract class is the set of all objects of its concrete subclasses
- A subclass is a subset of the set of all objects of its superclass
- An association is a set of links between objects of the associated classes

A class model is a specification of valid object configurations.
Example

Class diagram

This is a valid configuration (satisfies the class diagram) and thus is in the semantic domain of the above class diagram

This is an invalid configuration and thus is not in the semantic domain of the above class diagram
How are constraints expressed in a class model?

- Association multiplicities constrain the number of elements that can participate in an association
  - Note: the multiplicity * is **not** a constraint. Why? If you can answer this then you know what it means to be constrained (or restricted)

- Attribute types restrict the type of values that can be associated with an attribute.

- Are the above enough? What if you defined an attribute age: Integer, and wanted to restrict the value to integers greater than 18?
  - You can write it in natural language but you won’t be able to mechanically reason using this information
What is OCL?

- OCL can be used
  - to describe constraints
    - A constraint is a restriction on one or more values of a model or system.
    - A constraint is an expression that evaluate to true or false
  - as a query language
    - Queries are expressions that evaluate to a value (true, false and other values)
    - Can be used to define new attributes and operations
- OCL expressions are always associated with a UML model
  - OCL expressions can be associated with any model element in UML
**Examples of constraints:**

- Duration of a flight is the same as the difference between the arrival and departure times
- The maximum number of passengers on a flight must be less than 1,001
- The origin of a flight must be different than its destination

**Examples of queries:**

- Return all the departing flights from a given airport
- Return all the flights departing from a given airport with a departure time after 4p.m.
- Derive the arrival time by adding the duration of the flight to the departure time.
Specifying Constraints - Invariants
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
- airline

flights
- * 1 airline

departing Flights
- 1 *

destination
- 1

arriving Flights
- 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:

```plaintext
<<invariant>>
duration < 4
```

<table>
<thead>
<tr>
<th>Flight</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration: Integer</td>
</tr>
</tbody>
</table>

© Robert B. France
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)
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’
Association classes

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></td>
<td></td>
<td>type : Airtype</td>
</tr>
<tr>
<td>flights</td>
<td></td>
<td></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 \textasciitilde operation

Use of the “\textasciitilde” (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

airp1

airp2

airline1

airline2

airline3

f1

f2

f3

f4

f5

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

context 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

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:

\[
\text{collection} \rightarrow \text{exists}(\text{elem} : \text{T} \mid \text{expr})
\]

\[
\text{collection} \rightarrow \text{exists}(\text{elem} \mid \text{expr})
\]

\[
\text{collection} \rightarrow \text{exists}(\text{expr})
\]
Other collection operations

- **is_empty**: true if collection has no elements
- **not_empty**: 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
- **includes_all(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.

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
Specifying Constraints: Operation Specifications
Pre- and PostCondition Example

A class named Account has an attribute balance and an operation overdraft() that returns true if the balance is less than 0 and false otherwise.

context Account::overdraft():Boolean
pre : -- none
post : result = (balance < 0)
More complex operation specifications

The operation `birthdayOccurs()` adds 1 to the customer age.

```
class Customer::birthdayOccurs()
pre : -- none
post : age = age@pre + 1
```

```
class Account::safeWithdraw(amt:Integer)
pre : balance > amt
post : balance = balance@pre - amt
```
Derived Attribute & Initial Value Example

**Defining derived attributes**

context Flight::arrivalTime:Time
derive: departTime.plus(duration)

**Defining initial attribute value**

context Flight::maxNrPassengers:Integer
init: 100

**Defining initial association end value**

context Flight::passengers:Set(Passenger)
init: Set{ }
Query operation examples

Return all the departing flights from a given airport
context Airport::departures():Set(Flight)
body: result=departingFlights

Query operation example: Return all the airports served by an airline
context Airline::served():Set(Airport)
body: result=flights.destination->asSet
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)
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

class 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(numberOfCustomers)
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

context 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

© Robert B. France
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 Summary

- 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
Conclusion

"I knew we didn't have enough blocks for this thing."