Advanced Modeling with UML

- Part 1: Model Management
- Part 2: Extension Mechanisms and Profiles
- Part 3: Object Constraint Language (OCL)

Jos Warmer, Klasse Objecten
j.warmer@klasse.nl
Overview

- What are constraints
- Core OCL Concepts
- Advanced OCL Concepts
- Wrap up
Why use OCL?

What’s in it for me?

Use the Object Constraint Language
That’s why!!

Diagram:
- Flight
  - PassengerFlight
  - CargoFlight
  - flights
  - 0..*
- Airplane
- PassengerPlane
- CargoPlane
  - 0..*
  - 1
  - 1
Diagram with invariants

<table>
<thead>
<tr>
<th>Flight</th>
<th>0..*</th>
<th>1</th>
<th>Airplane</th>
</tr>
</thead>
<tbody>
<tr>
<td>type =</td>
<td></td>
<td></td>
<td>type =</td>
</tr>
<tr>
<td>enum{cargo, passenger}</td>
<td>flights</td>
<td></td>
<td>enum{cargo, passenger}</td>
</tr>
</tbody>
</table>

context Flight
inv: type = #cargo implies airplane.type = #cargo
inv: type = #passenger implies airplane.type = #passenger
Definition of constraint

• “A constraint is a restriction on one or more values of (part of) an object-oriented model or system.”
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
Constraint stereotypes

- UML defines three standard stereotypes for constraints:
  - invariant
  - precondition
  - postcondition
What is OCL?

- OCL is
  - a textual language to describe constraints
  - the constraint language of the UML
- Formal but easy to use
  - unambiguous
  - no side effects
Constraints and the UML model

- OCL expressions are always bound to a UML model
Overview

• What are constraints
• Core OCL Concepts
• Advanced OCL Concepts
• Wrap up
**Example model**

**Airport**
- name: String

**Flight**
- departTime: Time
- /arrivalTime: Time
- duration: Interval
- maxNrPassengers: Integer
- passengers: * {ordered}

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

**Airline**
- name: String

CEO
- airline: 0..1

Airline
- airline: 0..1
Constraint context and self

• Every OCL expression is bound to a specific context.
• The context may be denoted within the expression using the keyword ‘self’.
• 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:

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

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

Advanced Modeling with UML
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
  - associations from the UML model
Example: OCL basic types

context Airline inv:
name.toLowerCase = ‘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: query operations

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

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

• Navigations

class Flight

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

class Flight

inv: airline.name = ‘KLM’
context Person inv:
if employer.name = ‘Klasse Objecten’ then
  job.type = #trainer
else
  job.type = #programmer
endif
The OCL Collection types

- What are constraints
- Core OCL Concepts
  - Collections
- Advanced OCL Concepts
- Wrap up
Three subtypes to Collection

• Set:
  ■ arrivingFlights(from the context Airport)

• Bag:
  ■ arrivingFlights.duration (from the context Airport)

• Sequence:
  ■ passengers (from the context Flight)
Collection operations

- OCL has a great number of predefined operations on the collections types.

- Syntax:

  collection -> operation
The collect operation

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

- Shorthand:
  
  collection.expr

- The *collect* operation results in the collection of the values resulting evaluating *expr* for all elements in the *collection*
Example: collect operation

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

- Syntax:
  
  \[
  \text{collection-} \rightarrow \text{select} (\text{elem} : T \mid \text{expression}) \\
  \text{collection-} \rightarrow \text{select} (\text{elem} \mid \text{expression}) \\
  \text{collection-} \rightarrow \text{select} (\text{expression}) \\
  \]

- The *select* operation results in the subset of all elements for which *expression* is true
Example: collect operation

context Airport inv:
self.departingFlights->select(duration<4)->notEmpty

airp1

airp2

f1
duration = 2

f2
duration = 5

f3
duration = 3

f4
duration = 5

f5
duration = 2

airline1

airline2

airline3
The forAll operation

- Syntax:
  
  \[
  \text{collection-}\rightarrow\text{forAll} (\text{elem : T | expr}) \\
  \text{collection-}\rightarrow\text{forAll} (\text{elem | expr}) \\
  \text{collection-}\rightarrow\text{forAll} (\text{expr})
  \]

- The *forAll* operation results in true if *expr* is true for all elements of the collection
Example: forAll operation

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

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

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

```plaintext
class Airport

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

### Departing Flights
- **airp1**
  - f1: depart = 7
- **airp2**
  - f2: depart = 5
  - f3: depart = 8
  - f4: depart = 9
  - f5: depart = 8

### Arriving Flights
- **airline1**
- **airline2**
- **airline3**
Example: exists operation

context Airport inv:
self.departingFlights ->
exists(departTime.hour < 6)
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
Result in postcondition

- Example pre and postcondition

```java
class Airline {
    public Set<Airport> servedAirports() {
        // Implementation...
    }
}
```

context Airline::servedAirports() : Set(Airport)
pre :  -- none
post: result = flights.destination->asSet
Statechart: referring to states

- The operation \textit{oclInState} returns true if the object is in the specified state.

\begin{center}
\begin{tikzpicture}
  \node[draw] (open) {open};
  \node[draw, right of=open] (closed) {closed};
  \draw[->] (open) -- (closed);
\end{tikzpicture}
\end{center}

context Bottle inv:
self.oclInState(closed) implies filled = \#full
Local variables

• The Let construct defines variables local to one constraint:

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

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

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

• Example iterate:

```plaintext
class Airline inv:
flights->select(maxNrPassengers > 150)->notEmpty
```

• Is identical to:

```plaintext
class Airline inv:
flights->iterate(f : Flight; answer : Set(Flight) = Set{ } | if f.maxNrPassengers > 150 then
  answer->including(f)
else answer endif )->notEmpty
```
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
  ■ A postcondition may be strengthened
Wrap up

• What are constraints
• Core OCL Concepts
• Advanced OCL Concepts
• Wrap up
Current Developments

• Feedback from several OCL implementors handled in UML-RTF
  - e.g. the grammar has some loose ends
  - typical tool-related issues

• Development of OCL metamodel
  - currently concrete syntax only
  - will result in abstract syntax

• OCL Workshop with pUML group
  - formalization of OCL
OCL Tools

• Cybernetics
  ■ www.cybernetic.org

• University of Dresden
  ■ www-st.inf.tu-dresden.de/ocl/

• Boldsoft
  ■ www.boldsoft.com

• ICON computing
  ■ www.iconcomp.com

• Royal Dutch Navy

• Others … …
Conclusions and Tips

- **OCL invariants** allow you to
  - model more precisely
  - stay implementation independent

- **OCL pre- and postconditions** allow you to
  - specify contracts (design by contract)
  - precisely specify interfaces of components

- **OCL usage tips**
  - keep constraints simple
  - always combine natural language with OCL
  - use a tool to check your OCL