OBJECT CONSTRAINT LANGUAGE

OCL
{context Flight
  inv: type = #cargo implies airplane.type = #cargo
  inv: type = #passenger implies airplane.type = #passenger}
OCL INTRO

{context Flight
  inv:  type = #cargo implies airplane.type = #cargo
  inv:  type = #passenger implies airplane.type = #passenger}
OCL INTRO

{context Flight
  inv: type = #cargo implies airplane.type = #cargo
  inv: type = #passenger implies airplane.type = #passenger}

Always true
**OCL INTRO**

{context Flight
    inv:  **type** = #cargo \textit{implies} airplane.type = #cargo
    inv:  **type** = #passenger \textit{implies} airplane.type = #passenger}
context Flight
inv: origin <> destination
inv: origin.name = 'Amsterdam'

context Flight
inv: airline.name = 'KLM'
context Flight

inv: origin <> destination

inv: origin.name = 'Amsterdam'

context Flight

inv: airline.name = 'KLM'

inv: airline.name.toLowerCase = 'klm'
context Person inv:

if employer.name = ‘CVUT’ then
    job.type = #lecturer
else
    job.type = #programmer
endif
DEFINITION OF CONSTRAINT

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

- 1995 IBEL IBM
- Součást UML 1.1
Potřebováme jazyk pro specifikaci
Potřebujeme něco pro Objektový návrh
  ne nový jazyk
Potřebujeme něco standardizovaného
ÚČEL OCL

- Jazyk formální specifikace
  - -> implementovatelný

- Objektový návrh

- Intuitivní syntax pro OO jazyk

- Pozor není to programovací jazyk!
VÝHODY FORMÁLNÍ SPECIFIKACE

- Lepší dokumentace
  - Omezení modelu
  - Informace navíc
- Přesnost
  - Zamezí dvojznačnosti
- Komunikace bez nedorozumění
  - Komunikace mezi vývojářem a analytikem bez nedorozumění
KAM S OCL

- Specifikace invariantů tříd a typů
  - Co smí a co ne
  - Vymezení legálních hodnot
  - …
- Pre- a post-conditions pro metody
- Navigační jazyk
- Omezení/vymezení operací
- Test požadavku a specifikace
1. A person may have a mortgage only on a house she owns.
2. The start date of a mortgage is before its end date.
1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate

1. A person may have a mortgage only on a house she owns.
2. The start date of a mortgage is before its end date.
OCL SPECIFICATION OF THE CONSTRAINTS:

1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate

context Mortgage
invariant: security.owner = borrower

context Mortgage
invariant: startDate < endDate
OCL SPECIFICATION OF THE CONSTRAINTS:

1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate

context Mortgage
invariant: security.owner = borrower

context Mortgage
invariant: startDate < endDate
OCL SPECIFICATION OF THE CONSTRAINTS:

1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate
OCL SPECIFICATION OF THE CONSTRAINTS:

1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate

   context Mortgage
   invariant: security.owner = borrower

   context Mortgage
   invariant: startDate < endDate
OCL SPECIFICATION OF THE CONSTRAINTS:

1. context Mortgage
   invariant: self.security.owner = self.borrower

2. context Mortgage
   invariant: self.startDate < self.endDate

context Mortgage
invariant: security.owner = borrower

context Mortgage
invariant: startDate < endDate
MORE CONSTRAINTS EXAMPLES

- All players must be over 18.
  
  context Player invariant: self.age >= 18

- The number of guests in each room doesn’t exceed the number of beds in the room.
  
  context Room invariant: guest -> size() <= numberOfBeds

Constraint (invariant)
- Boolean OCL výraz – vyhodnotitelný do true/false

Každý constraint je vázán na specifický typ
- (class, association class, interface) v UML modelu
- Má svůj context.

Context objektu může být vytičen přes keyword ‘self’.

Context lze specifikovat přes:
- Context <context name>
- Přerušovanou čárou spojující objekt v UML modelu

Constraint může mít název za klíčovým slovem invariant.
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean

class invariant
{ points >= 0 }

<<precondition>>
i >= 0

<<postcondition>>
points = points@pre + i

<<precondition>>
points >= i and i >= 0

<<postcondition>>
result = (points=0)

<<postcondition>>
points = points@pre - i

postcondition for burn operation

precondition for burn operation
Invarianty Atributů

- Invarianty atributů:

  ```
  context Customer
  invariant age-restriction: age >= 18
  ```

  ```
  context CustomerCard
  invariant
  correctDates: validFrom.isBefore(goodThru)
  ```

  The type of `validFrom` and `goodThru` is `Date`. `isBefore(Date):Boolean` is a `Date` operation.

- The class on which the invariant must be put is the invariant `context`.
- For the above example, this means that the expression is an invariant of the `Customer` class.
Invarianty : Navigace přes asociace – role

Navigation over associations is used to refer to Associated objects, starting from the context object:

context CustomerCard

\[\text{invariant: } \text{owner}.\text{age} \geq 18\]

\[\text{owner} \rightarrow \text{a Customer instance.}\]
\[\text{owner}.\text{age} \rightarrow \text{an Integer.}\]

Note: This is not the “right” context for this constraint!
If the role name is missing – use the class name at the other end of the association, starting with a lowercase letter.
Preferred: Always give role names.
Invarianty : Navigace přes asociace – roles

context CustomerCard
invariant printedName:

\[
\text{printedName} = \text{owner.title}.\text{concat}(\ ' ').\text{concat}(\text{owner.name})
\]

\[
\text{printedName} \rightarrow \text{a String.}
\]

\[
\text{owner} \rightarrow \text{a Customer instance.}
\]

\[
\text{owner.title} \rightarrow \text{a String.}
\]

\[
\text{owner.name} \rightarrow \text{a String.}
\]

String is a recognized OCL type.
concat is a String operation, with the signature
\[
\text{concat(String)} : \text{String.}
\]
Navigation from a class through an association class uses the association class name to obtain all tuples of an object:

“The cards of the memberships of a customer are only the customer’s cards”:

```
context Customer
invariant correctCard:
  cards->includesAll(Membership.card)
```

This is exactly the same as the previous constraint:

“The owner of the card of a membership must be the customer in the membership”:

```
context Membership
invariant correctCard: card.owner = customer
```

The Membership correctCard constraint is better!
Invariants using Navigation through Associations with “Many” Multiplicity

Navigation over associations roles with multiplicity greater than 1 yields a **Collection** type. Operations on collections are accessed using an arrow “->”, followed by the operation name.

“A customer card belongs only to a membership of its owner”:

```ocl
class CustomerCard

context CustomerCard
invariant correctCard:
  owner.Membership->includes(membership)

owner \rightarrow a Customer instance.
owner.Membership \rightarrow a set of Membership instances.
membership \rightarrow a Membership instance.
includes is an operation of the OCL Collection type.
```
NAVIGATING TO COLLECTIONS

If we want to use this as a set we have to do the following:

```
account.transaction -> asSet
```
“The partners of a loyalty program have at least one delivered service”:  

```plaintext
context LoyaltyProgram
invariant minServices:
partners.deliveredservices->size() >= 1
```

“The number of a customer’s programs is equal to that of his/her valid cards”:  

```plaintext
context Customer
invariant sizesAgree:
Programs->size() = cards->select(valid=true)->size()
```
“When a loyalty program does not offer the possibility to earn or burn points, the members of the loyalty program do not have loyalty accounts. That is, the loyalty accounts associated with the Memberships must be empty”:

context LoyaltyProgram
invariant noAccounts:
   partners.deliveredservices->
   forall(pointsEarned = 0 and pointsBurned = 0)
   implies Membership.account->isEmpty()

and, or, not, implies, xor are logical connectives.
THE OCL COLLECTION TYPES

- **Collection** is a predefined OCL type
  - Operations are defined for collections
  - They never change the original
- **Three different collections:**
  - **Set** (no duplicates)
  - **Bag** (duplicates allowed)
  - **Sequence** (ordered Bag)
- With collections type, an OCL expression either states a fact about all objects in the collection or states a fact about the collection itself, e.g. the size of the collection.
- **Syntax:**
  - `collection->operation`
COLLECTION OPERATIONS

<collection> → size ( )
    → isEmpty ( )
    → notEmpty ( )
    → sum ( )
    → count ( object )
    → includes ( object )
    → includesAll ( collection )
COLLECTIONS CONT.

<collection> → select ( e:T | <b.e.>)
→ reject ( e:T | <b.e.>)
→ collect ( e:T | <v.e.>)
→ forAll ( e:T* | <b.e.>)
→ exists ( e:T | <b.e.>)
→ iterate ( e:T₁; r:T₂ = <v.e.> | <v.e.>)

b.e. stands for: boolean expression
v.e. stands for: value expression
### CHANGING THE CONTEXT

<table>
<thead>
<tr>
<th>Customer</th>
<th>StoreCard</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>printName: String</td>
</tr>
<tr>
<td>title: String</td>
<td>points: Integer</td>
</tr>
<tr>
<td>golduser: Boolean</td>
<td>earn(p: Integer)</td>
</tr>
<tr>
<td>age(): Integer</td>
<td></td>
</tr>
</tbody>
</table>

**Invariant:*** 

```plaintext
printName = owner.title.concat(owner.name)
```

**Context StoreCard**

**Context Customer**

```plaintext
cards ➔ forAll ( printName = owner.title.concat(owner.name) )
```
EXAMPLE UML DIAGRAM

- **Student**
  - name: String
  - 0..* taken_by
  - submitted_by

- **Module**
  - code: String
  - credit: Integer
  - 1..* takes

- **Assessment**
  - weight: Integer
  - 1..*

- **Exam**
  - hours: Integer

- **Coursework**
  - date: String

- Subtypes:
  - 0..* for_module
  - 1..* set_work

- Generalization:
  - Exam and Coursework
CONTRAINTS

a) Modules can be taken iff they have more than seven students registered

b) The assessments for a module must total 100%

c) Students must register for 120 credits each year

d) Students must take at least 90 credits of CS modules each year

e) All modules must have at least one assessment worth over 50%

f) Students can only have assessments for modules which they are taking
CONSTRAINT (A)

Modules can be taken iff they have more than seven students registered

Note: when should such a constraint be imposed?

cor text *Module*

invariant: \( \text{taken}_\text{by} \rightarrow \text{size} > 7 \)
**CONSTRAINT (B)**

The assessments for a module must total 100%

**context Module**

**invariant:**

\[
\text{set\_work\_weight} \rightarrow \text{sum}(\ ) = 100
\]
CONSTRAINT (C)

Students must register for 120 credits each year

category Student

invariant: \( \text{takes.credit} \rightarrow \text{sum}( ) = 120 \)
CONSTRAINT (D)

Students must take at least 90 credits of CS modules each year

class Student

context Student

invariant:

takes →

\[ \text{select}(\text{code}.\text{substring}(1,2) = \text{‘CS’}).\text{credit} \rightarrow \text{sum}( ) \geq 90 \]
CONSTRAINT (E)

All modules must have at least one assessment worth over 50%

context Module

invariant: set_work \rightarrow \exists (weight > 50)
CONSTRAINT (F)

Students can only have assessments for modules which they are taking

certainty **Student**

**invariant:** \( \text{takes} \rightarrow \text{includesAll} (\text{submits} . \text{for}_\text{module}) \)
A constraint is a restriction on one or more values of (part of) an object model/system.

Constraints come in different forms:

- invariant
  - constraint on a class or type that must always hold
- pre-condition
  - constraint that must hold before the execution of an op.
- post-condition
  - constraint that must hold after the execution of an op.
- guard
  - constraint on the transition from one state to another.

We studied only class constraints (invariants).
FURTHER RESOURCES FOR OCL

- The Object Constraint Language
  - ISBN 0-201-37940-6
- OCL home page
  - [www.kLASse.nl/ocl/index.htm](http://www.kLASse.nl/ocl/index.htm)
FURTHER RESOURCES FOR OCL

- The Amsterdam Manifesto on OCL
  - In Object Modeling with the OCL (LNCS2263) p115-149
- The Object Constraint Language, Precise Modeling with UML 2nd
- Response to the UML 2.0 OCL RfP (ad/2000-09-03) Revised Submission, Version 1.6 January 6, 2003
- Some Shortcomings of OCL, the Object Constraint Language of UML
  - Mandana Vaziri and Daniel Jackson, 1999
  - http://www.klasse.nl/english/uml/ UML CENTER
- Informal formality? The Object Constraint Language and its application in the UML metamodel
  - Anneke Kleppe, Jos Warmer, Steve Cook
- A Practical Application of the Object Constraint Language OCL
  - Kjetil Moage
- The UML's Object Constraint Language: OCL Specifying Components, JAOO Tutorial – September 2000
  - Jos Warmer & Anneke Kleppe