FOUNDATIONS OF THE UML

Lecture 11 :

The Object Constraint Language

Joost-Pieter Katoen

MOVES

moves.rwth-aachen.de / i2 / 192

January 21, 2008

THE OBJECT CONSTRAINT LANGUAGE

- Textual annotation to UML diagrams (such as class diagrams and statecharts)
- Strongly related to predicate, and first-order logic
- OCL constraints impose additional restrictions on the UML models

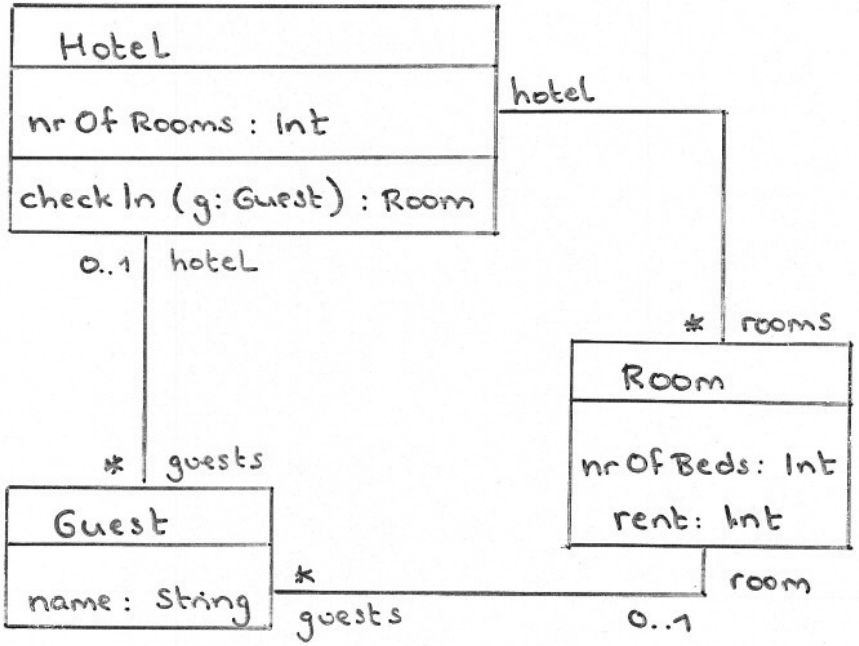
  e.g. invariants ("always $x>0$")
- Developed by Jos Warmer & Anneke Kleppe
- OCL 2.0. has been adopted by the UML
- Basic ingredients:

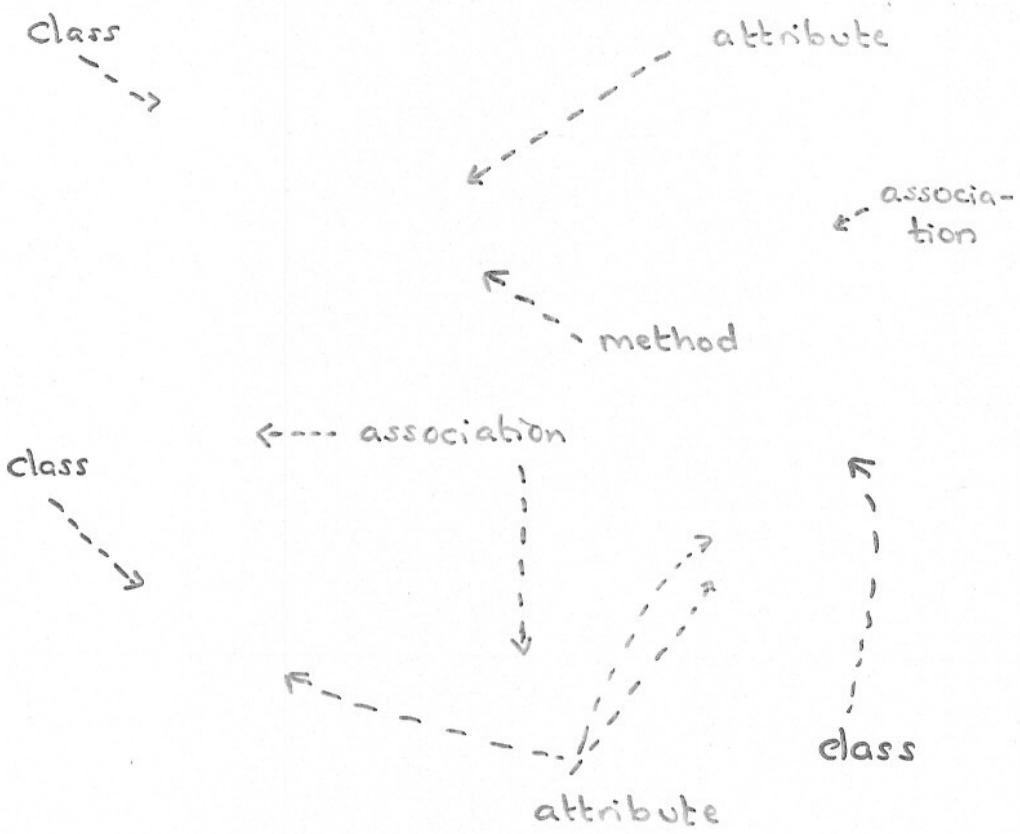
  typed variables, expressions (e.g. navigation)

  constraints (e.g. invariants), iterations

OCL EXAMPLES

Consider the class diagram:





OCL EXAMPLES

- The number of guests in a room cannot exceed the room's capacity:

```
context Room
inv : guests → size ≤ nrofBeds
```

- The guests in the rooms of the hotel equals the guests in the hotel

```
context Hotel
inv : rooms.guests = guests
```

- These are invariants, i.e. they should hold in any state of the system.

- The violation of an invariant can always be shown by a finite system run that ends in a state that refutes the invariant condition

OCL EXAMPLES

- When checking in a guest g, say
  - g should not already be a guest in the hotel;
  - after checking in, the number of guests is increased by one, and should include g

```
context Hotel :: checkIn (g: Guest)
pre : not guests → includes (g)
post : guests → size =
         (guests@pre → size) + 1
       and guests → includes (g)
```

where guests@pre refers to "value" of guests at evaluating the precondition

- On each invocation of method checkIn, if the precondition holds, then on termination of checkIn, the postcondition holds

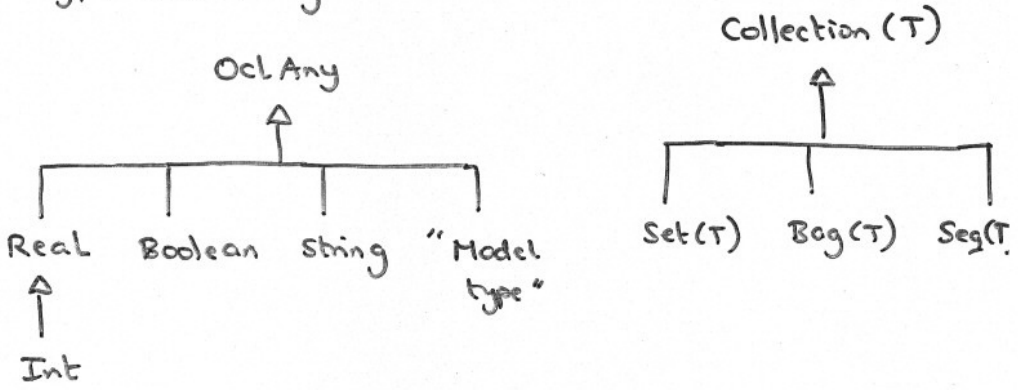
OCL TYPES

OCL is a typed language. Its types are:

- predefined types
  - basic types (e.g., Int, String, Boolean, ...)
  - collection types (e.g., Collection, Set, Bag, Sequence)
  - OclAny

  equipped with standard operations (+, −, *, and, not, union, concatenation etc.)
- model types: the types defined in the UML model (e.g., Hotel, Room, and Guest)

Type hierarchy:



Assumption: all OCL terms are type-correct

OCL Syntax

OCL constraints:

χ ::= context C inv Σ

| context C :: M($\vec{p}$) pre Σ post Σ

where C is a class

M $\in$ dom(C.meths) is a method

(C.meths: methods of class C)

$\vec{p}$ are the parameters

Σ is an OCL expression

- OCL constraints are built from OCL expressions and have type Boolean.
- Invariants have as context a class.
- Pre- and postconditions have as context a method of a class.

OCL syntax

OCL expressions:

$\Sigma ::=$ self $\mid z \mid$ result $\mid \Sigma @pre$

$\mid \Sigma.a \mid \omega(\Sigma, \ldots, \Sigma) \mid \Sigma.\omega(\Sigma, \ldots, \Sigma)$

$\mid \Sigma \rightarrow \omega(\Sigma, \ldots, \Sigma)$

$\mid \Sigma \rightarrow$ iterate $(x_1 ; x_2 := \Sigma \mid \Sigma)$

where:

- self refers to context object of class C
- z represents either
  - an attribute of the context object, or
  - a formal parameter of context metho
  - or a logical variable
- result refers to the value returned by the context method (is undefined if this method has not returned a value)
- @pre refers to the value of its operand at the time of method invocation.

Both expressions result and @pre may be used in postconditions only.

OCL Operations

- $\Sigma.a$ attribute / parameter navigation
  - Σ is an object reference to its attribute a or to a method occurrence with a formal parameter named a
  - $\Sigma.a$ denotes the value of this attribute

examples: x. rooms. guests, hotel. rooms

- for n-ary operator ω the OCL expression

$$\omega(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$$

denotes the application ω to $(\Sigma_1, \ldots, \Sigma_n)$

examples: isEqual (g_1, g_2)

ifThenElse (b, Σ_1, Σ_2)

- $\Sigma.\omega(\Sigma_1, \ldots, \Sigma_n)$ represents an operator ω on basic types applied on $\Sigma, \Sigma_1, \ldots, \Sigma_n$.

Int, Bool, String etc.

If Σ is of collection type (set, bag, sequence, ...) then we write $\Sigma \longrightarrow \omega(\Sigma_1, \ldots, \Sigma_n)$

OCL Iterations

$\Sigma_1 \longrightarrow$ iterate $(x_1 ; x_2 = \Sigma_2 \mid \Sigma_3)$ is an expression that binds logical variables x_1, x_2

Intuitive interpretation:

- first, x_2 is initialised to Σ_2
- then Σ_3 is computed repeatedly
- and its result is assigned to x_2
- while x_1 successively takes as its value an element of the sequence Σ_1

Example:

$[1,2,3] \longrightarrow$ iterate $(x_1 ; x_2 = 0 \mid x_1 + x_2)$

computes the sum of the elements of sequence $[1,2,3]$

$x_2 := 0$, $x_1 := 1$ } $x_1 + x_2 = 1$

$x_2 := 1$, $x_1 := 2$ } $x_1 + x_2 = 3$

$x_2 := 3$, $x_1 := 3$ } $x_1 + x_2 = 6$

AN OCL DEFICIENCY

- The iterate expression is a powerful iteration mechanism on collections
- Its evaluation, however, is problematic on unordered collections (e.g., Set, Bag)

  e.g., $\{1,2,3\} \longrightarrow$ iterate $(x_1 ; x_2 = 0 \mid -x_2 + x_1)$

  the result is not well-defined.

  Depending on the order of binding the elements in $\{1,2,3\}$ to x_1, the result will be -1, 3, or 5.

- In OCL, nested collections are "automatically flattened"

  e.g., Set { Set {1,2}, Set {3,4,5}} = Set {1,2,3,4,5}

  but what about

  Sequence { Set {1,3,7}} ?

  all orderings of {1,3,7} are allowed!

OPERATIONAL MODEL

OCL semantics is defined using an operational model of an object-based system.

Let:

VNAME is a countable set of variable names

MNAME is a countable set of method names (ranged over by M)

CNAME is a countable set of class names (ranged over by C)

τ ($\in$ TYPE) ::= void | nat | bool | τ list | C ref | C.M ref

- void is the unit type with trivial value ()
- τ list denotes the type of lists of τ with elements [] (empty list) and h::w (list with head h and tail w)

  notation $[h_1, h_2, \ldots, h_n]$
- C ref = type of objects of class C
- C.M ref = type of method occurrences of M of C

OPERATIONAL MODEL

Partial functions:

- VDECL : VNAME $\rightharpoonup$ TYPE

  maps variable names onto types
- MDECL : MNAME $\rightharpoonup$ VDECL (formal parameters) $\times$ TYPE (return type)
- CDECL : CNAME $\rightharpoonup$ VDECL (attributes) $\times$ MDECL (methods)

Notation: let D $\in$ CDECL. For C $\in$ dom(D)

let C.attrs ($\in$ VDECL) are its attributes

C.meths ($\in$ MDECL) are its methods

For method M of class C:

M.fpars ($\in$ VDECL) are its formal parameters

M.retty ($\in$ TYPE) is its return type

Thus: C.meths(M) = (M.fpars, M.retty)

EXAMPLE

OPERATIONAL MODEL

As operational model we use automata* of the form $(Conf, \longrightarrow, I)$ where:

- $Conf$ is a set of configurations
- $\longrightarrow \subseteq Conf \times Conf$, a transition relation
- $I \subseteq Conf$ a set of initial states with $I \neq \emptyset$

Intuition: a configuration denotes the state of the UML model (e.g., current objects, current method calls, state of each object + methods) and $\longrightarrow$ models the evolution of the system, such that: if an active method occurrence becomes inactive then it has a well-defined return value (i.e, not $\perp$).

* also called Kripke structures

OBJECTS AND EVENTS

References to objects and events will be used as data values.

Events correspond to method occurrences, i.e., invocations of a given method of a given object.

Let $C \in CNAME$ and $M \in MNAME$:

$$OID^C = \{C\} \times \mathbb{N}$$

numbered instances of the class C

$$EVT^{C,M} = OID^C \times \{M\} \times \mathbb{N}$$

numbered instances of method M with explicit associated to object executing M.

$$OID = \bigcup_C OID^C$$

set of object ids

$$EVT = \bigcup_C \bigcup_M EVT^{C,M}$$

set of events

Thus $o \in OID$ is a pair (C, n)

"the n-th object instance of class C"

$e \in EVT$ is a triple $((C,n), M, k)$

"k-th invocation of method M, executed by (C,n)"

EXAMPLE

Consider the Hotel class diagram

Example instances of class Hotel:

(Hotel, 1) , (Hotel, 2) , (Hotel, 27) , ...

Example instances of class Guest:

(Guest, 231) , (Guest, 0) , ...

Example events related to method CheckIn:

((Hotel, 1), checkIn, 1)
((Hotel, 1), checkIn, 2)
((Hotel, 27), checkIn, 1)
.....

} different execution of checkIn by same object

VALUES

Data types of operational model:

$$\tau ::= \text{void} \mid \text{nat} \mid \text{bool} \mid \tau\ \text{list} \mid C\ \text{ref} \mid C.M\ \text{ref}$$

The universe of values $VAL = \bigcup_{\tau} VAL^{\tau}$

VAL^{τ}, set of values for type τ, is defined by:

$$VAL^{\text{void}} = \{()\}$$

$$VAL^{\text{nat}} = \mathbb{N}$$

$$VAL^{\text{bool}} = \{ff, tt\}$$

$VAL^{\text{char}} = \{a, b, c, \ldots z\}$

$$VAL^{\tau\ \text{list}} = \{[]\} \cup \{h::w \mid h \in VAL^{\tau}, w \in VAL^{\tau\ \text{list}}\}$$

$$VAL^{C\ \text{ref}} = \{\text{null}\} \cup OID^{C}$$

$$VAL^{C.M\ \text{ref}} = EVT^{C,M}$$

Data types are equipped with standard operation:

e.g.:

$$+ : VAL^{\text{nat}} \times VAL^{\text{nat}} \longrightarrow VAL^{\text{nat}}$$

$$\text{sort} : VAL^{\tau\ \text{list}} \longrightarrow VAL^{\tau\ \text{list}}$$

$$\text{flat} : VAL^{\tau\ \text{list}\ \tau\ \text{list}} \longrightarrow VAL^{\tau\ \text{list}}$$

↳ flattens nested lists

STRICTNESS

$\perp \notin VAL$ denotes the "undefined" value

let $VAL_{\perp} = VAL \cup \{\perp\}$

All operations are extended to $VAL_{\perp}$ such that the interpretation is strict, i.e., if one of the operands is strict, the entire expression equals $\perp$.

For example:

$$\perp :: w = \perp$$

$$h :: \perp = \perp$$

$$3 + \perp = \perp$$

$$\text{sort}\ \perp = \perp$$

etc.

$$\perp \vee tt = \perp$$

CONFIGURATIONS

A configuration := current objects + current method invocations + object states + method invociations state.

Formally, a configuration is a tuple (O, E, σ, τ) with:

$O \subseteq OID$ currently alive objects

$E \subseteq EVT$ currently alive method call:

$\sigma : O \to VNAME \to VAL$

$\tau : E \to (VNAME \to VAL) \times VAL_\perp$

For each $o \in O$, $\sigma(o)$ is local state of object o

$\sigma(o) = \ell$ with $o \in OID^C \to dom(\ell) = dom(C.attrs)$ and $\ell(a) \in VAL^{C.attrs(a)}$ for each $a \in dom(\ell)$

σ is extended point-wise to lists of objects, i.e.,

$\sigma([])(a) = []$

$\sigma(h:w)(a) = \sigma(h)(a) :: \sigma(w)(a)$

CONFIGURATIONS

$\tau : E \to (VNAME \to VAL) \times VAL_\perp$

- E: event (= method invocation)
- $(VNAME \to VAL)$: valuation of formal parameters of invoked method
- $VAL_\perp$: return value of method

if $\tau(e) = (\ell, v)$ for $e \in EVT^{C,M}$ then:

- $dom(\ell) = dom(M.fpars)$
- $\ell(p) \in VAL^{M.fpars(p)}$ for $p \in dom(\ell)$
- $v \in VAL_\perp^{M.retty}$

A method invociation has terminated in the current configuration if it is deallocated in the next state. On termination, the method has a well-defined value (i.e., different from $\perp$)

if: $(O, E, \sigma, \tau) \to (O', E', \sigma', \tau')$ then

$e \in E \setminus E' \to \exists v \in VAL.\ \tau(e) = (\ell, v)$

EXAMPLE