Software Modeling and Verification
Lehrstuhl für Informatik 2
RWTH Aachen University

Prof. Dr. Ir. J.-P. Katoen
C. Kern

Foundations of the UML

Winter Term 07/08

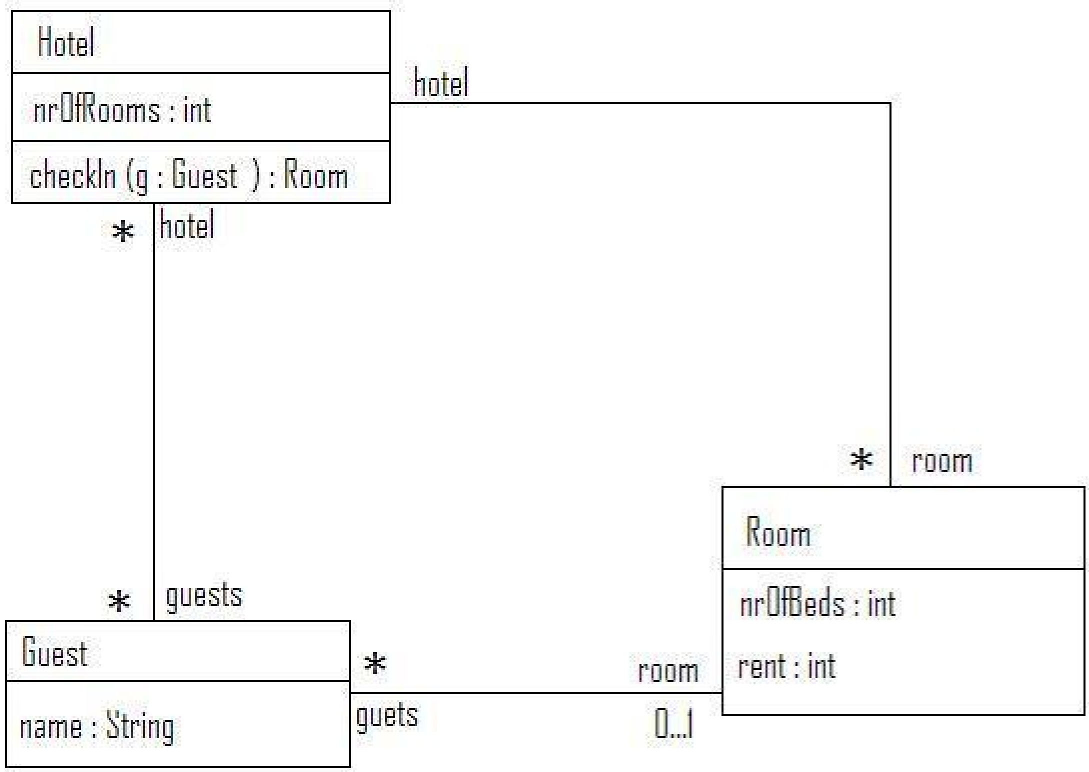
– Lecture number 12: The Object Constraint Language –

(Date (January 21, 2008))
summarized by *Hussein Hamid Baagil* and *Önder Babur*

The Object Constrain Language

Introduction

- OCL is used for textual annotation in UML
  (such as class diagrams and statecharts)
- Is strongly related to predicate, and first-order *logic*
- OCL constraints impose additional restrictions on UML models
  e.g. invariants ("‘always x > 0"’)
- Developed by Jos Warmer and 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 example

for the example above:

- **class**: Hotel, Room, Guest
- **attribute**: nrOfRooms, nrOfBeds, rent, name
- **methode** : checkIn(g:Guest)
- Between the classes exit some association

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

context Room

inv: guests $\rightarrow$ size $\leq$ nrOfBeds

- The guest 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 state that refuses the invariant condition

- 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:Guests)
  pre: not guests $\rightarrow$ includes(g)
  post: guests $\rightarrow$ size = (guests@pre $\rightarrow$ size)+1
  and guests $\rightarrow$ includes(g)

  where guests@pre refers to the "‘value"’ of guests at evaluating the precondition. Or another way to say, it refers to guests value at the time of method invocation

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

OCL Types

OCL is a typed language, it consists of **predefined** types and **model** types.

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

  these predefined types could be equipped with standard operation (+, -, *, and, not, union, concatenation, etc.)

- model type : these are types whic are defined in the UML moderl (in our example, Hotel, Room, and Guests)

OCL Syntax

OCL constraints are defined as:

$$\chi ::= \underline{\textbf{context}}\ \textbf{C}\ \underline{\textbf{inv}}\ \zeta$$
$$\quad |\ \underline{\textbf{context}}\ \textbf{C :: M}(\vec{p})\ \underline{\textbf{pre}}\ \zeta\ \underline{\textbf{post}}\ \zeta$$

C is a class
M $\in$ dom(C.meths), a method of class C
$\vec{p}$ are the parameters of method M
ζ is an OCL expression

- Each OCL constraint is built from OCL expression (ζ) and have type boolean
- Invariants have as context a class
- Pre- and postconditions have as context a method of a class and a class

OCL expressions are defined as:

$$
\begin{aligned}
\zeta ::= &\ \text{self} \mid \text{z} \mid \text{result} \mid \zeta@\text{pre} \\
&\mid \zeta \cdot \text{a} \mid \omega(\zeta,\ldots,\zeta) \mid \zeta \cdot \omega(\zeta,\ldots,\zeta) \\
&\mid \zeta \rightarrow \omega(\zeta,\ldots,\zeta) \\
&\mid \zeta \rightarrow \text{iterate } (x_1; x_2 := \zeta \mid \zeta)
\end{aligned}
$$

self refers to context object of class C
z represents either
- an attribute of the context object, or
- a formal parameter of context method
- 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 opperand at the time of method invocation

Both expression result and @pre may only be used in postconditions

OCL Operations

- $\zeta \cdot a$ an attribute or parameter navigation
  - ζ is an object refers to its attribute a or to *a* method occurence with a formal parameter *a*
  - $\zeta \cdot a$ returns the value of the attribute

  example: *x.rooms.guests* returns the value of the number of guests in rooms at hotel *x*
- for n-ary operator ω the OCL expression
  $\omega(\zeta_1,\zeta_2,...,\zeta_n)$
  returns the value of application/funktion ω on $(\zeta_1,\zeta_2,...,\zeta_n)$
  example: isEqual(g_1,g_2) return the boolean value of application/funktion *isEqul* on g_1 and g_2
- the notation $\zeta \cdot \omega(\zeta_1,\zeta_2,...,\zeta_n)$ represent operator ω on basic types (Int, Boolean, etc.). If ζ is of collection type (set, bag, etc.) then we use the notation $\zeta \rightarrow \omega(\zeta_1,\zeta_2,...,\zeta_n)$

OCL Iteration

$\zeta_1 \rightarrow$ iterate (x_1 ; $x_2 = \zeta_2$ | ζ_3)

a) x_2 will be initialised to ζ_2

b) x_1 takes as its value the first element from ζ_1

c) ζ_3 is computed and its result is assigned to x_2

d) x_1 will successively takes as its value next element of the sequence ζ_1

Example: $[1, 2, 3] \rightarrow$ iterate (x_1 ; $x_2 = 0$ | $x_1 + x_2$)
compute the sum of the elments of the list $[1, 2, 3]$

first iteration

- $x_2 := 0$
- $x_1 := 1$
- $x_2 := x_1 + x_2$

second iteration

- $x_2 := 1$
- $x_1 := 2$
- $x_2 := x_1 + x_2$

third iteration

- $x_2 := 3$
- $x_1 := 3$
- $x_2 := x_1 + x_2$

finaly $x_2 := 6$

An OCL Deficiency

- The iterate expression is a powerful iteration mechanism on ordered collection
- Thus its evaluation is problematic on unordered collections, like set and bag
  e.g., $\{1, 2, 3\} \rightarrow$ iterate (x_1 ; $x_2 = 0$ | $-x_2 + x_1$)
  $\Rightarrow$ the result is not well-defined, depending on the order binding the elements
- In OCL, nested collections are "‘automatically flattened"’ e.g, Set $\{Set\,\{1, 2\}$, Set$\{3, 4, 5\}$ = Set $\{1, 2, 3, 4, 5\}$
  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 M)
> CNAME is a countable set of class names (ranged over C)
> T ($\in$ TYPE) ::== void | nat | bool | T list | C ref | C.M ref

- void is the unit type with trivial value ()

- T list denotes the type of lists of T with elements [] (empty list) and h::w (list with head h and tail w)
  notation [h1, h2, ... h3]

- C ref = type of objects of class C

- C.M ref = type of method occurrency of M of C

Partial functions:

- VDECL : VNAME $\rightarrow$ TYPE
  maps variable names onto types
- MDECL : MNAME $\rightarrow$ VDECL $\times$ TYPE, VDECL are formal parameters, TYPE is return type

- CNAME : CNAME $\rightarrow$ VDECL $\times$ MDECL, VDECL are attributes, MDECL are methods

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

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

For methods M of class C:

M.fpars ($\in$ VDECL) are its formal parameters
C.retty ($\in$ TYPE) is its return type

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

Example :

$$Hotel.attrs(v) = \begin{cases} nat & \text{if v = nrOfRooms} \\ Roomlist & \text{if v = rooms} \\ Guestlist & \text{if v = guests} \\ \perp & \text{otherwise. (means undefined)} \end{cases}$$

$$Room.attrs(v) = \begin{cases} Hotel & \text{if v = Hotel} \\ Guest & \text{if v = guests} \\ \perp & \text{otherwise.} \end{cases}$$

$$checkIn.fpars(v) = \begin{cases} Guests & \text{if v = g} \\ \perp & \text{otherwise.} \end{cases}$$

$$Hotel.meths(m) = \begin{cases} (checkIn.fpars, void) & \text{if m = checkIn} \\ \perp & \text{otherwise.} \end{cases}$$

Note that $\perp$ is a special value, where $\perp \vee$ true = $\perp$.

As operational model, we use <u>automata</u> (also called Kripke structures) of the form (Conf, $\rightarrow$, I) where:

- Conf is a set of <u>configurations</u>
- $\rightarrow \subseteq$ Conf $\times$ Conf, a <u>transition relation</u>
- I $\subseteq$ Conf, a set of initial states with I $\neq \emptyset$

<u>Intuition</u> : a configuration denotes the state of the UML model (e.g, current objects, current method calls, state of each object + methods) and $\rightarrow$ 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 $\perp$).

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$^{C} = \{C\} \times$ IN (numbered instances of the class C)
EVT$^{C,M} = OID^{C} \times$ {M} IN $\times$ (numbered instances of method M with explicit asociated to object executing M.)
OID = $\cup_C OID^C IN(setofobjectids)$
EVT = $\cup_C \cup_M EVT^{C,M} IN(setofevents)$

Thus o $\in$ 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:
Example instances of class Guest:

((Hotel,1), checkIn, 1)
((Hotel,1), checkIn, 2) $\rightarrow$ different execution of checkIn by same object
((Hotel,27), checkIn, 1)
...

Values

Data types of operational model:

T ::= void | nat | bool | T list | C ref |C.M ref

The universe of values VAL = $\cup_T VAL^T$
$VAL^T, set of values for type T, is defined by$:

VAL$^{void} = \{()\}$
VAL$^{nat} = IN$
VAL$^{bool} = \{(ff, tt)\}$
VAL$^{char} = \{(a, b, c, ..., z)\}$
VAL$^{Tlist} = \{[]\}\cup$ {h::w | h $\in$ VAL^T, $w\in$ VAL$^{Tlist}\}$
VAL$^{Cref} = \{null\}\cup$ OID^C
VAL$^{C.Mref} = \{EVT^{C,M}\}$

Data types are equipped with standard operation. eg:

+ : VAL$^{nat}\times$ VAL$^{nat}\rightarrow$ VAL^{nat}
sort : VAL$^{Tlist}\rightarrow$ VAL^{Tlist}
flat : VAL$^{Tlist(5mm)Tlist}\rightarrow$ VAL$^{Tlist}(flattens nested lists)$

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 :: $\perp$ = $\perp$
ζ + $\perp$ = $\perp$
sort $\perp$ = $\perp$
$\perp\vee$ tt = $\perp$
etc.

Configurations

A configuration := current objects + current method invocations + object states + method invocation state.
Formally, a configuration is a tuple (O, E, σ, γ) with:

$O \subseteq OID$
$E \subseteq EVT$
$\sigma : O \to VNAME \to VAL$
$\gamma : E \to (VNAME \to VAL) \times VAL$

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

$\sigma(o) = \ell$ with o $\in$ OID^C,$\to$ dom(ℓ) = dom(C.attrs) and $\ell(a) \in$ VAL$^{C.attrs(a)} for(5mm)each(5mm)a \in$ dom(ℓ).

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

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

$\gamma : E \to (VNAME \to VAL) \times VAL\bot$
event(method invocation) $\to$ valuations of formal parameters of invoked method $\times$ return value of method

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

dom(ℓ) = dom(M.fpars)

$\ell(p) \in$ VAL$^{M.fpars(p)} for(5mm)p \in$ dom(ℓ)

$v \in VAL_T^{M.retty}$

A method invocation 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 $\bot$).

If $(O, E, \sigma, \gamma) \to (O', E', \sigma', \gamma')$ then
$e \in E \setminus E' \to \exists\, v \in VAL.\ \gamma(e) = (\ell, v)$.