

# Software Design, Modelling and Analysis in UML

## Lecture 03: Object Constraint Language (OCL)

2012-10-30

Prof. Dr. Andreas Podolski, Dr. Bernd Westphal  
 Albert-Ludwigs-Universität Freiburg, Germany

### Contents & Goals

#### Last Lecture:

- Basic Object System Signature  $\mathcal{S}$  and Structure  $\mathcal{D}$
- System State  $\sigma \in \mathbb{S}_{\mathcal{D}}$   
*(Smells like they're related to class/object diagrams, officially we don't know yet...)*

#### This Lecture:

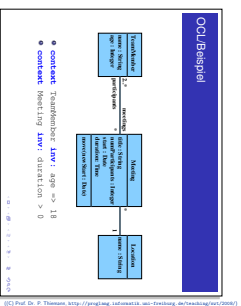
- Educational Objectives: Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Does this OCL constraint hold in this system state?
  - Can you think of a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are  $\mathcal{S}(C)$  and  $\mathcal{S}(C')$  related?
- Content:
  - OCL Syntax, OCL Semantics over system states

2/36

### What is OCL? And What is It Good For?

### What is OCL? How Does it Look Like?

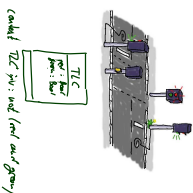
- OCL: Object Constraint Logic



4/36

### What's It Good For?

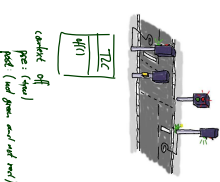
- **Most prominent:** write down requirements supposed to be satisfied by all system states. Often targeting all alive objects of a certain class.



5/36

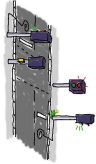
### What's It Good For?

- **Most prominent:** write down requirements supposed to be satisfied by all system states. Often targeting all alive objects of a certain class.
- **Not unknown:** write down pre/post-conditions of methods (Behavioural Features). Then evaluated over two system states.



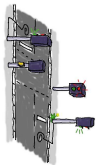
5/36

- **Most prominent:** write down requirements supposed to be satisfied by all system states. Often targeting all alive objects of a certain class.



- **Not unknown:** write down pre/post-conditions of methods (*Behavioural Features*). Then evaluated over two system states.
- **Common with State Machines:** guards in transitions.

- **Most prominent:** write down requirements supposed to be satisfied by all system states. Often targeting all alive objects of a certain class.



- **Not unknown:** write down pre/post-conditions of methods (*Behavioural Features*). Then evaluated over two system states.
- **Common with State Machines:** guards in transitions.
- **Lesser known:** provide operation bodies.

- **Metamodeling:** the UML standard is a MOF-Metamodel of UML. OCL expressions define well-formedness of UML models (cf. Lecture ~ 21).

OCL Syntax 1/4: Expressions

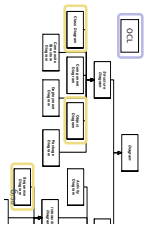
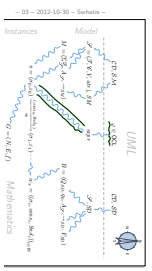
$expr ::=$	$w$	$\tau(w)$	<i>type of w</i>
	$expr_1 \rightarrow expr_2$	$T \times T \rightarrow Bool$	
	$oclIsDefined(expr_1)$	$T \times T \rightarrow Bool$	
	$oclIsTypeOf(expr_1, T_1)$	$T \times T \times T \rightarrow Bool$	<i>type of expr<sub>1</sub></i>
	$oclIsKindOf(expr_1, T_1)$	$T \times T \times T \rightarrow Bool$	<i>type of expr<sub>1</sub></i>
	$oclIsInstanceof(expr_1, T_1)$	$T \times T \times T \rightarrow Bool$	<i>type of expr<sub>1</sub></i>
	$allInstancesOf(T_1)$	$Set(T_1)$	
	$v(expr_1)$	$T \times T \rightarrow T$	
	$n_1(expr_1)$	$T \times T \rightarrow T$	
	$r_1(expr_1)$	$T \times T \rightarrow T$	
	$r_2(expr_1)$	$T \times T \rightarrow T$	

Where, given  $\mathcal{S} = (\mathcal{S}, \mathcal{C}, V, \text{attr})$ ,

- $W \subseteq \{self, c \mid C \in \mathcal{C}\}$  is a set of typed logical variables.
- $w$  has type  $\tau(w)$ .
- $T$  is any type from  $\mathcal{S} \cup T_B \cup T_E$ .
- $U(Set(T_0)) \mid T_0 \in T_B \cup T_E$  is the following set of basic types, in the following we use  $T_B = \{Bool, Int, String\}$ .
- $T_E = \{T_C \mid C \in \mathcal{C}\}$  is the set of object types.
- $Set(T_0)$  denotes the set of object types for  $T_0 \in T_B \cup T_E$ .
- $setOf(T_0)$  denotes the set of object types for  $T_0 \in T_B \cup T_E$ .
- $v: T \times T \rightarrow T$  is the binary function of "functioning" (cf. standard).
- $n_1: T \times T \rightarrow T$ ,  $n_2: T \times T \rightarrow T$ .
- $r_1: D_1 \in \text{attr}(C)$ ,  $r_2: D_2 \in \text{attr}(C)$ .
- $C, D \in \mathcal{C}$ .

Plan.

- **Today:** The set  $OCLExpressions(\mathcal{S})$  of OCL expressions over  $\mathcal{S}$ .
- Given an OCL expression  $expr$ , a system state  $\sigma \in \Sigma_{\mathcal{S}}$ , and a valuation of logical variables  $\beta$ , define  $\llbracket expr \rrbracket(\sigma, \beta) \in \{true, false, \perp\}$ .
- **Later:** use  $I$  to define  $\models \subseteq \Sigma_{\mathcal{S}} \times OCLExpressions(\mathcal{S})$ .



OCL Syntax: Notational Conventions for Expressions

- Each expression  $P$  may alternatively be written ("abbreviated as")  $w \rightarrow w(expr_1, expr_2, \dots, expr_n) : T_1 \times \dots \times T_n \rightarrow T$ .
- $expr_1 \dots expr_n$  if  $n$  is an object type, i.e. if  $\tau_i \in T_E$ .
- $expr_1 \rightarrow w(expr_2, \dots, expr_n)$  if  $\tau_1$  is a collection type (here: only sets), i.e. if  $\tau_1 = Set(T_0)$  for some  $T_0 \in T_B \cup T_E$ .

### OCL Syntax: Notational Conventions for Expressions

- Each expression

$wl(expr_1, expr_2, \dots, expr_n) : T_1 \times \dots \times T_n \rightarrow T$   
 may alternatively be written ("abbreviated as")  $cond$  parentheses if  $n \neq 1$

- $expr_1 \cdot wl(expr_2, \dots, expr_n)$  if  $T_1$  is an object type, i.e. if  $\tau_1 \in T_{Obj}$ .
- $expr_1 \rightarrow wl(expr_2, \dots, expr_n)$  if  $T_1$  is a collection type (here, only sets), i.e. if  $\tau_1 \in Set(\tau_0)$  for some  $\tau_0 \in T_{Obj} \cup T_{Etc}$ .

**Examples:**  $(self : \tau_0 \in W; \quad \tau_1, w : \text{Int} \in V; \quad \tau_1 : D_{Int}, \tau_2 : D_{\rightarrow} \in V)$   
 $self \cdot w()$      $self \cdot w()$      $self \cdot w()$      $self \cdot w()$   
 if we have variables:  $\tau_1$   
 $\tau_2$      $\tau_1$      $\tau_2$      $\tau_1$   
 then we can also write in OCL:  $f : self \cdot f(\tau_1)$   
 that is equivalent to  $f : self \cdot f(\tau_1)$   
 $f : self \cdot f(\tau_1)$

### OCL Syntax 2/4: Constants, Arithmetical Operators

#### For example:

```

expr ::= ...
| true | false
| expr1 {and-or-implies} expr2
| not expr1
| |l|-||l|-d|l|...
| OclUndefined
| expr1 {+-...} expr2
| expr1 {<=>} expr2

```

Generalised notation:

$expr ::= wl(expr_1, \dots, expr_n)$   
 with  $w \in \{+-\dots\}$      $e = + (expr_1, \dots, expr_n)$      $e \in \{<=>\}$      $expr_1 \{<=>\} expr_2$

$\tau_1 \times \dots \times \tau_n \rightarrow T$   
 $Bool \times Bool \rightarrow Bool$   
 $Bool \rightarrow Bool$   
 $Int \times Int \rightarrow Int$   
 $Int \times Int \rightarrow Bool$

### OCL Syntax 3/4: Iterate

$expr ::= \dots | expr_1 \rightarrow Iterate(w : \tau_1, w_2 : \tau_2 = expr_2 | expr_3)$   
 or with a little renaming:  
 $expr ::= \dots | expr_1 \rightarrow Iterate(Iter : \tau_1, result : \tau_2 = expr_2 | expr_3)$

where

- $expr_1$  is of a collection type (here: a set  $Set(\tau_0)$  for some  $\tau_0$ ).
- $Iter \in W$  is called **Iterator**, gets type  $\tau_1$  (if  $\tau_1$  is omitted,  $\tau_0$  is assumed as type of  $Iter$ )
- $result \in W$  is called **result variable**, gets type  $\tau_2$ .
- $expr_2$  in an expression of type  $\tau_2$  giving the **initial value** for  $result$ .
- (OclUndefined if omitted)
- $expr_3$  is an expression of type  $\tau_2$
- $expr_3$  in which in particular  $Iter$  and  $result$  may appear.

### Iterate: Inuitive Semantics (Formally: later)

```

expr ::= expr_1 -> Iterate(Iter : \tau_1;
    result : \tau_2 = expr_2 | expr_3)

```

$Set(\tau_0) \text{ obj} = \langle expr_1 \rangle;$   
 $\tau_1 \text{ iter};$   
 $\tau_2 \text{ result} = \langle expr_2 \rangle;$   
 while ( $!obj.empty()$ ) do  
 $\text{iter} = obj.pop();$   
 $\text{result} = \langle expr_3 \rangle;$   
 od

eg.  $result + obj$

**Note:** In our (simplified) setting, we always have  $expr_1 : Set(\tau_1)$  and  $\tau_0 = \tau_1$ . In the type hierarchy of full OCL with inheritance and oclAny, they may be different and still type consistent.

### Abbreviations on Top of Iterate

```

expr ::= expr_1 -> Iterate(w1 : \tau_1;
    w2 : \tau_2 = expr_2 | expr_3)

```

$expr_1 \rightarrow Iterate(w1 : \tau_1, w2 : \tau_2 = expr_2 | expr_3)$   
 is an abbreviation for

$expr_1 \rightarrow Iterate(w1 : \tau_1, w2 : \tau_2 = true | w2 \leftarrow expr_2, w2 \leftarrow expr_3)$   
 (To ensure confusion, we may again omit all kinds of things, cf. [OMG, 2006])

Similar:  $expr_1 \rightarrow Exist(w : \tau_1 | expr_3)$

### OCL Syntax 4/4: Context

$context ::= context \ w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv: } expr$   
 where  $w \in W$  and  $\tau \in T_{Etc}$ ,  $1 \leq i \leq n$ ,  $n \geq 0$ .

$context \ w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv: } expr$   
 is an abbreviation for

$allInstances_{C_1} \rightarrow forall(w_1 : \tau_1 | \dots | forall(w_n : \tau_n | \dots | expr))$

Context: More Notational Conventions

- For context  $self : TC$  inv :  $expr$  we may alternatively write ("abbreviate as") context  $TC$  inv :  $expr$
- Within the later abbreviation, we may omit the "self" in  $expr$ , i.e. for  $self\ x$  and  $self\ r$  we may alternatively write ("abbreviate as")  $x$  and  $r$

Examples (from lecture)

Handwritten notes on the diagram:
 

- context Transformation:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$

Handwritten notes:
 

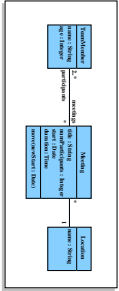
- context:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$

Examples (from lecture "Softwaretechnik 2008")

Handwritten notes on the diagram:
 

- context:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$
- context:  $inv : age > 18$

Example (from lecture "Softwaretechnik 2008")



- context Meeting inv :  $inv : age > 18$
- context Meeting inv :  $inv : age > 18$
- context Meeting inv :  $inv : age > 18$
- context Meeting inv :  $inv : age > 18$

"Not Interesting"

- Enumeration types
- Type hierarchy
- Complete list of arithmetical operators
- The two other collection types Bag and Sequence
- Caching
- Runtime type information
- Pre/post conditions (maybe later... when we officially know what an operation is)
- ...

OCL Semantics [OMG, 2006]

OCL Syntax / OCL Expressions		Where given $\mathcal{E} = (\mathcal{Z} \subseteq \mathcal{T}, \text{rel})$ .
expr ::=		
$\text{true} \mid \text{false}$	$\text{true} \mid \text{false}$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{true}$ is true and $\text{false}$ is false.
$\text{not}(expr)$	$\text{not}(expr)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{not}(expr)$ is the set of objects in $\mathcal{Z}$ that do not satisfy $expr$ .
$\text{and}(expr_1, expr_2)$	$\text{and}(expr_1, expr_2)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{and}(expr_1, expr_2)$ is the set of objects in $\mathcal{Z}$ that satisfy both $expr_1$ and $expr_2$ .
$\text{or}(expr_1, expr_2)$	$\text{or}(expr_1, expr_2)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{or}(expr_1, expr_2)$ is the set of objects in $\mathcal{Z}$ that satisfy either $expr_1$ or $expr_2$ .
$\text{xor}(expr_1, expr_2)$	$\text{xor}(expr_1, expr_2)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{xor}(expr_1, expr_2)$ is the set of objects in $\mathcal{Z}$ that satisfy exactly one of $expr_1$ or $expr_2$ .
$\text{implies}(expr_1, expr_2)$	$\text{implies}(expr_1, expr_2)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{implies}(expr_1, expr_2)$ is the set of objects in $\mathcal{Z}$ that satisfy $expr_1$ implies $expr_2$ .
$\text{exists}(x : \text{Type}, expr)$	$\text{exists}(x : \text{Type}, expr)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{exists}(x : \text{Type}, expr)$ is the set of objects in $\mathcal{Z}$ that satisfy $expr$ for some object $x$ of type $\text{Type}$ .
$\text{forall}(x : \text{Type}, expr)$	$\text{forall}(x : \text{Type}, expr)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{forall}(x : \text{Type}, expr)$ is the set of objects in $\mathcal{Z}$ that satisfy $expr$ for all objects $x$ of type $\text{Type}$ .
$\text{let}(x : \text{Type} \mid expr_1, expr_2)$	$\text{let}(x : \text{Type} \mid expr_1, expr_2)$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{let}(x : \text{Type} \mid expr_1, expr_2)$ is the set of objects in $\mathcal{Z}$ that satisfy $expr_2$ under the assumption that $expr_1$ is true.
$\text{if}(expr) \text{ then } expr_1 \text{ else } expr_2$	$\text{if}(expr) \text{ then } expr_1 \text{ else } expr_2$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{if}(expr) \text{ then } expr_1 \text{ else } expr_2$ is the set of objects in $\mathcal{Z}$ that satisfy $expr_1$ if $expr$ is true and $expr_2$ if $expr$ is false.
$\text{if}(expr) \text{ then } expr_1$	$\text{if}(expr) \text{ then } expr_1$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{if}(expr) \text{ then } expr_1$ is the set of objects in $\mathcal{Z}$ that satisfy $expr_1$ if $expr$ is true.
$\text{if}(expr) \text{ then } expr_1 \text{ else } \text{false}$	$\text{if}(expr) \text{ then } expr_1 \text{ else } \text{false}$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{if}(expr) \text{ then } expr_1 \text{ else } \text{false}$ is the set of objects in $\mathcal{Z}$ that satisfy $expr_1$ if $expr$ is true.
$\text{if}(expr) \text{ then } \text{false}$	$\text{if}(expr) \text{ then } \text{false}$	If $\mathcal{Z}(\text{rel})$ is a set of typed objects, then $\text{if}(expr) \text{ then } \text{false}$ is the set of objects in $\mathcal{Z}$ that do not satisfy $expr$ .

Given an OCL expression  $expr$ , a system state  $\sigma \in \Sigma_{\mathcal{Z}}^{\mathcal{E}}$ , and a valuation of logical variables  $\beta$ , denote

$$\llbracket expr \rrbracket(\sigma, \beta) : \text{OCL Expressions}(\mathcal{Z}) \times \Sigma_{\mathcal{Z}}^{\mathcal{E}} \times (V \rightarrow \{\mathcal{Z} \cup \mathcal{T}_D, UT_D\}) \rightarrow \{\text{Bool}\}$$

such that

$$\llbracket expr \rrbracket(\sigma, \beta) \in \{\text{true}, \text{false}, \text{undef}\}.$$

### References

- [OMG, 2006] OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.
- [OMG, 2007a] OMG (2007a). Unified modeling Language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.
- [OMG, 2007b] OMG (2007b). Unified modeling Language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.
- [Warner and Kleppe, 1999] Warner, J. and Kleppe, A. (1999). *The Object Constraint Language*. Addison-Wesley.