Software Design, Modelling and Analysis in UML

Lecture 03: Object Constraint Language (OCL)

2013-10-28

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Basic Object System Signature $\mathscr{S}$ and Structure $\mathscr{D}$
- System State $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$

(Smells like they're related to class/object diagrams, officially we don't know yet...)

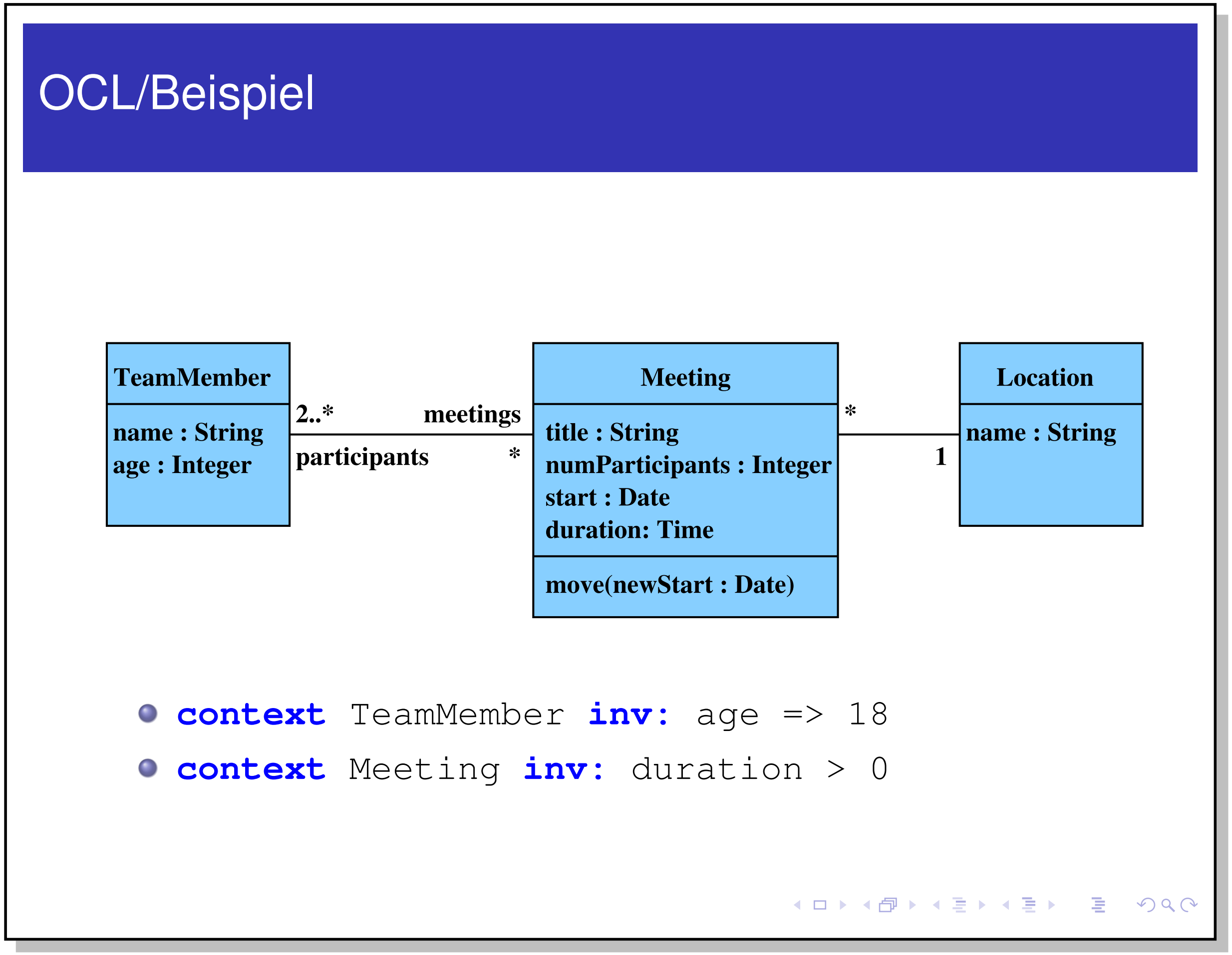
This Lecture:

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - 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 $\mathscr{D}(C)$ and τ_C related?
- **Content:**
  - OCL Syntax, OCL Semantics over system states

What is OCL? And What is It Good For?

What is OCL? How Does it Look Like?

- **OCL**: Object Constraint Logic.

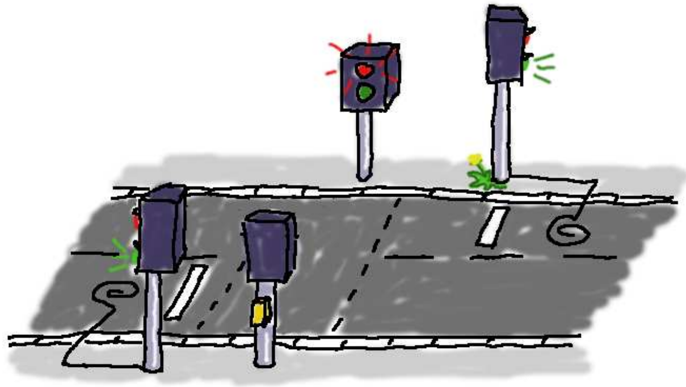


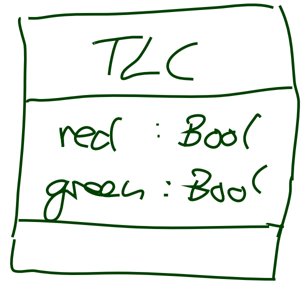
((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/)

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.



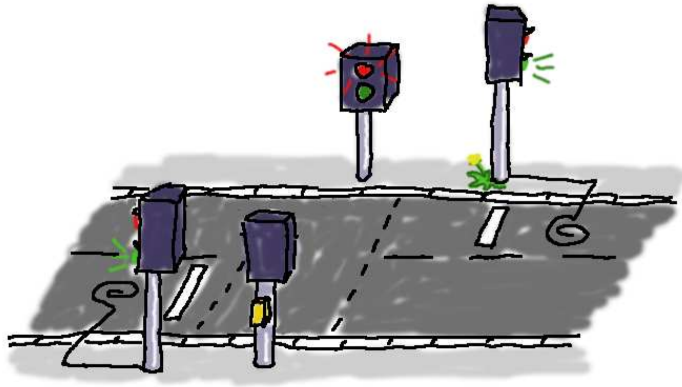


context TLC inv : not (red and green)

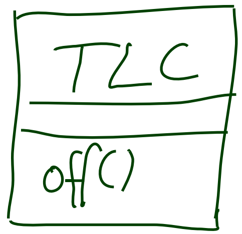
What's It Good For?

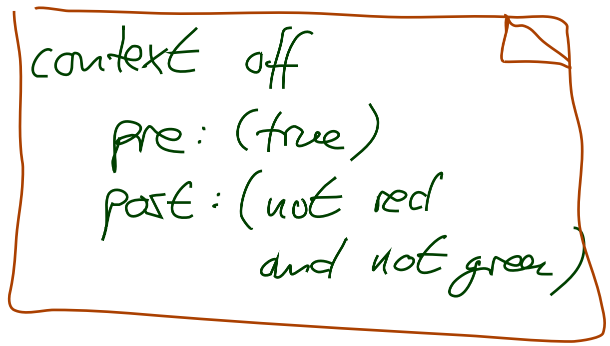
- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

  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.

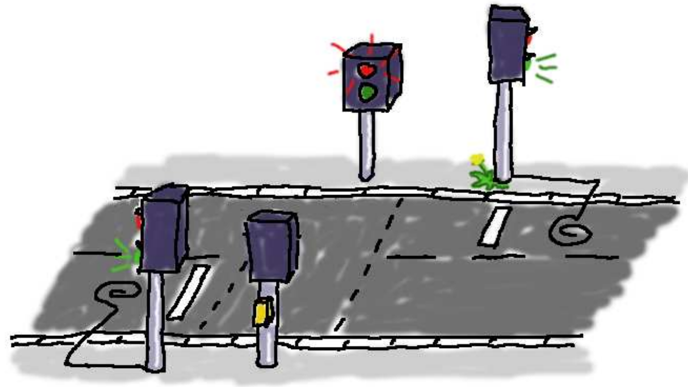




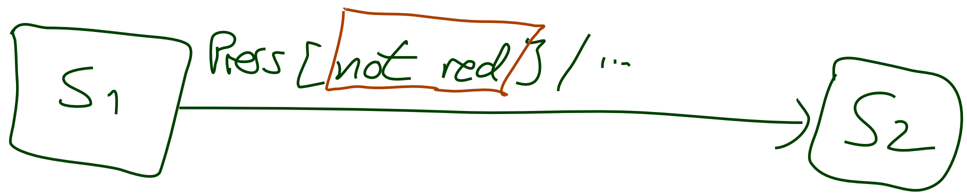
What's It Good For?

- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

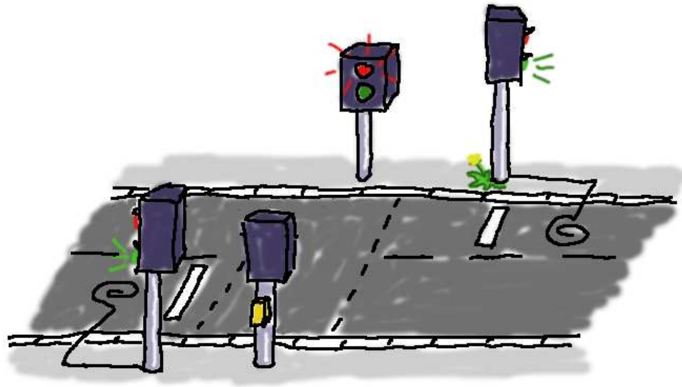
  Often targeting all alive objects of a certain class.



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



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.
- **Common with State Machines**:
  guards in transitions.
- **Lesser known**:
  provide **operation bodies**.

- **Metamodeling**: the UML standard is a MOF-Model of UML.
  OCL expressions define well-formedness of UML models (cf. Lecture $\sim$ 21).

Plan.

$I : OCLExpr(\mathscr{S}) \times \Sigma_\mathscr{S}^\mathscr{D} \times B \longrightarrow \{true, false, \perp\}$

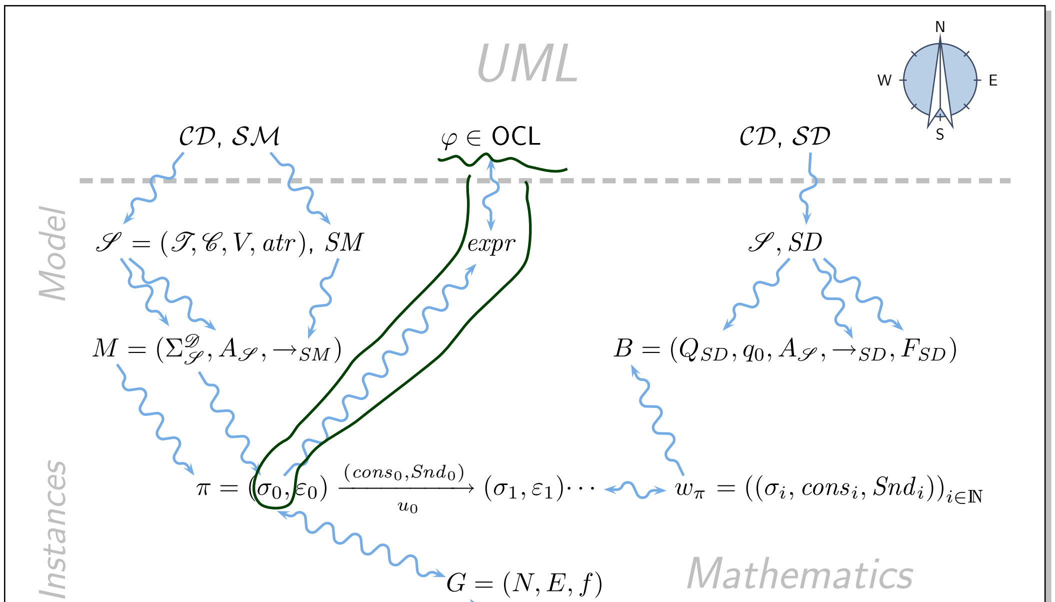
$I(expr, \sigma, \beta)$

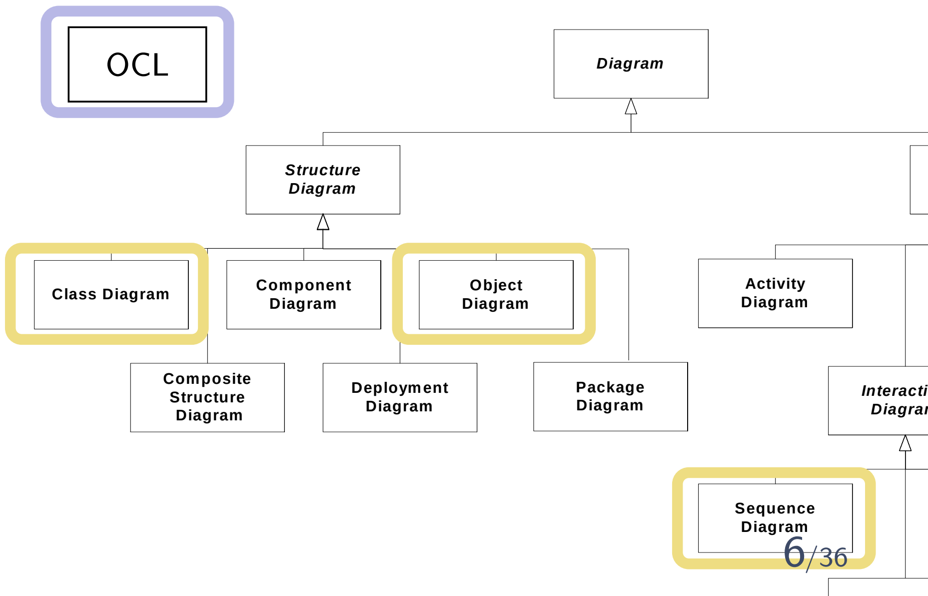
valuations of W

- **Today**:
  - The set $OCLExpressions(\mathscr{S})$ of OCL expressions over $\mathscr{S}$.
  - Given an OCL expression $expr$, a system state $\sigma \in \Sigma_\mathscr{S}^\mathscr{D}$, and a valuation of logical variables β, define

$$I[\![expr]\!](\sigma, \beta) \in \{\mathit{true}, \mathit{false}, \perp\}.$$

- **Later**: use I to define $\models \; \subseteq \Sigma_\mathscr{S}^\mathscr{D} \times OCLExpressions(\mathscr{S})$.





(Core) OCL Syntax [OMG, 2006]

OCL Syntax 1/4: Expressions

$$
\begin{array}{lll}
expr ::= & & \\
\quad w & & : \tau(w) \\
\quad \mid \; expr_1 =_\tau expr_2 & & : \tau \times \tau \to Bool \\
\quad \mid \; \mathsf{oclIsUndefined}_\tau(expr_1) & & : \tau \to Bool \\
\quad \mid \; \{expr_1, \dots, expr_n\} & & : \tau \times \dots \times \tau \to Set(\tau) \\
\quad \mid \; \mathsf{isEmpty}(expr_1) & & : Set(\tau) \to Bool \\
\quad \mid \; \mathsf{size}(expr_1) & & : Set(\tau) \to Int \\
\quad \mid \; \mathsf{allInstances}_C & & : Set(\tau_C) \\
\quad \mid \; v(expr_1) & & : \tau_C \to \tau(v) \\
\quad \mid \; r_1(expr_1) & & : \tau_C \to \tau_D \\
\quad \mid \; r_2(expr_1) & & : \tau_C \to Set(\tau_D)
\end{array}
$$

set comprehension operator

type of $expr_1$

type of $expr_2$

type of $expr_1 =_\tau expr_2$

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$,

- $W \supseteq \{self_C \mid C \in \mathscr{C}\}$ is a set of typed logical variables, w has type $\tau(w)$
- $\tau(self_C) = \tau_C$
- τ is any type from $\mathscr{T} \cup T_B \cup T_{\mathscr{C}}$ $\cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\}$
  - T_B is a set of basic types, in the following we use $T_B = \{Bool, Int, String\}$
  - $T_{\mathscr{C}} = \{\tau_C \mid C \in \mathscr{C}\}$ is the set of object types,
  - $Set(\tau_0)$ denotes the set-of-τ_0 type for $\tau_0 \in T_B \cup T_{\mathscr{C}}$ (sufficient because of "flattening" (cf. standard))
- $v : \tau(v) \in atr(C)$, $\tau(v) \in \mathscr{T}$,
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,
- $C, D \in \mathscr{C}$.

$$\mathcal{S} = (\{Integ\}, \{C, D\}, \{x : Integ, y : Integ, p : C_*, q : D_{0,1}\}, \{C \mapsto \{x, y, p, q\}, D \mapsto \{y, q\}\})$$

$$W = \{self_C : \tau_C, self_D : \tau_D\} \cup \{a, b, c : Int\} \cup \{n, m : \tau_C, e : \tau_D\} \cup \{N, M : set(\tau_C)\}$$

- $self_C$
- a
- n
- $oclIsUndefined(a)$
- $\{n, m\} : Set(\tau_C)$
- $size(N) : Int$
- $allInstances_C$

- $p(n) : Set(\tau_C)$

  $n : \tau_C$, $p : C_* \in atr(C)$
- $q(n) : \tau_D$

  $n : \tau_C$

  $q : D_{0,1} \in atr(C)$
- $q(\underbrace{q(n)}_{expr_1}) : \tau_D$

  $q : D_{0,1} \in atr(D)$

- $x(\cancel{p}\, n) : Integ$

  $n : \tau_C$, $x : Integ \in atr(C)$, $Integ \in \mathcal{T}$
- $x(a)$? NO (type mismatch)
- $a(n)$? YES ! NO !! ? ||||||||

  NO, a is not an attribute of C
- $x(p)$? NO, p is an attribute and not a logical variable

OCL Syntax: Notational Conventions for Expressions

- Each expression

$$\omega(expr_1, expr_2, \dots, expr_n) : \tau_1 \times \dots \times \tau_n \to \tau$$

may alternatively be written ("abbreviated as")

- $expr_1 \,.\, \omega(expr_2, \dots, expr_n)$ if τ_1 is an **object type**, i.e. if $\tau_1 \in T_{\mathscr{C}}$.
- $expr_1 \to \omega(expr_2, \dots, expr_n)$ if τ_1 is a **collection type** (here: only sets), i.e. if $\tau_1 = Set(\tau_0)$ for some $\tau_0 \in T_B \cup T_{\mathscr{C}}$.

- **Examples:** $(self : \tau_C \in W; \quad v, w : Int \in V; \quad r_1 : D_{0,1}, r_2 : D_* \in V)$
  - $self \,.\, v$ [expr1] [ω]   v(self)
  - $self \,.\, r_1 \,.\, w$ [expr] [ω]   w(self.r1) [expr] [ω]   attributes ∈W   w(r1(self))
  - $self \,.\, r_2 \to$ isEmpty   [expr] [ω]   isEmpty(self.r2)   isEmpty(r2(self))

- self -> v   NO, self not of collection type

OCL Syntax 2/4: Constants, Arithmetical Operators

For example:

$$
\begin{aligned}
expr ::= \; & \ldots \\
& \mid \mathsf{true}, \mathsf{false} && : Bool \\
& \mid expr_1 \; \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\} \; expr_2 && : Bool \times Bool \to Bool \\
& \mid \mathsf{not} \; expr_1 && : Bool \to Bool \\
& \mid 0, -1, 1, -2, 2, \ldots && : Int \\
& \mid \mathsf{OclUndefined}_{\tau} && : \tau \\
& \mid expr_1 \; \{+, -, \ldots\} \; expr_2 && : Int \times Int \to Int \\
& \mid expr_1 \; \{<, \leq, \ldots\} \; expr_2 && : Int \times Int \to Bool
\end{aligned}
$$

Generalised notation:

$$expr ::= \omega(expr_1, \ldots, expr_n) \qquad : \tau_1 \times \cdots \times \tau_n \to \tau$$

with $\omega \in \{+, -, \ldots\}$

e.g.
and(expr1, expr2)

OCL Syntax 3/4: Iterate

$\mathscr{S} = (\emptyset, \{C, D\}, \{p : C_*, q : D_*\}, \{C \mapsto \{p, q\}\})$

$\mathit{self.p} \to \mathit{iterate}(\mathit{iter};\ \mathit{res} : \mathit{Int} = 0;\ \mathit{res} + \mathit{size}(\mathit{iter.q}))$

$$expr ::= \cdots \mid expr_1\text{->iterate}(w_1 : \tau_1 \,;\, w_2 : \tau_2 = expr_2 \mid expr_3)$$

$\mathit{self} : \tau_C$

or, with a little renaming,

$$expr ::= \cdots \mid expr_1\text{->iterate}(iter : \tau_1;\ result : \tau_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some τ_0),
- $iter \in W$ is called **iterator**, gets type τ_1
  (if τ_1 is omitted, τ_0 is assumed as type of $iter$)
- $result \in W$ is called **result variable**, gets type τ_2,
- $expr_2$ in an expression of type τ_2 giving the **initial value** for $result$,
  ('OclUndefined' if omitted)
- $expr_3$ is an expression of type τ_2
  in which in particular $iter$ and $result$ may appear.

Iterate: Intuitive Semantics (Formally: later)

$$
\begin{aligned}
expr ::= \ & expr_1\texttt{->iterate}(iter : \tau_1; \\
& \qquad result : \tau_2 = expr_2 \mid expr_3)
\end{aligned}
$$

pseudo code

$$
\begin{aligned}
& Set(\tau_0)\ hlp = \langle expr_1 \rangle; \\
& \tau_1\ iter; \\
& \tau_2\ result = \langle expr_2 \rangle; \\
& while\ (!hlp.empty())\ do \\
& \qquad iter = hlp.pop(); \\
& \qquad result = \langle expr_3 \rangle; \\
& od
\end{aligned}
$$

pick and remove one element

e.g. result + size(iter.g)

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

$$expr ::= expr_1\text{->iterate}(w_1 : \tau_1;\quad w_2 : \tau_2 = expr_2 \mid expr_3)$$

- $$expr_1\text{->forAll}(w : \tau_1 \mid expr_3)$$
  is an abbreviation for
  $$expr_1\text{->iterate}(w : \tau_1;\ w_1 : Bool = true \mid w_1\ and\ expr_3).$$
  (To ensure confusion, we may again omit all kinds of things, cf. [OMG, 2006]).
- Similar: $$expr_1\text{->Exists}(w : \tau_1 \mid expr_3)$$

OCL Syntax 4/4: Context

$$context ::= \texttt{context}\ \underbrace{w_1 (: \tau_1)}, \dots, \underbrace{w_n (: \tau_n)}\ \texttt{inv :}\ expr$$

where $w \in W$ and $\tau_i \in T_{\mathscr{C}}$, $1 \leq i \leq n$, $n \geq 0$.

τ_{C_1}

context $w_1 : C_1, \dots, w_n : C_n$ inv : $expr$

is an **abbreviation** for

allInstances$_{C_1}$ -> forAll($w_1 : C_1$ |

$\dots$

allInstances$_{C_n}$ -> forAll($w_n : C_n$ |

$expr$

)

$\dots$

)

– 03 – 2013-10-28 – Socl syn –

Context: More Notational Conventions

- For

$$\text{context } self : \tau_C \text{ inv} : expr$$

we may alternatively write ("abbreviate as")

$$\text{context } \tau_C \text{ inv} : expr$$

e.g. context C_1 inv : expr

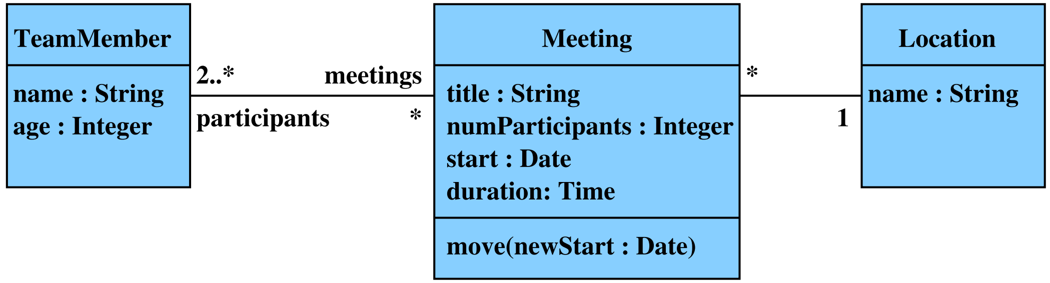
- **Within** the latter abbreviation, we may omit the "$self$" in $expr$, i.e. for

$$self.v \quad \text{and} \quad self.r$$

we may alternatively write ("abbreviate as")

$$v \quad \text{and} \quad r$$

Examples (from lecture



- `context TeamMember inv: age => 18`
- `context Meeting inv: duration > 0`

context self : TM inv: age ≥ 18

—— " —— self . age ≥ 18

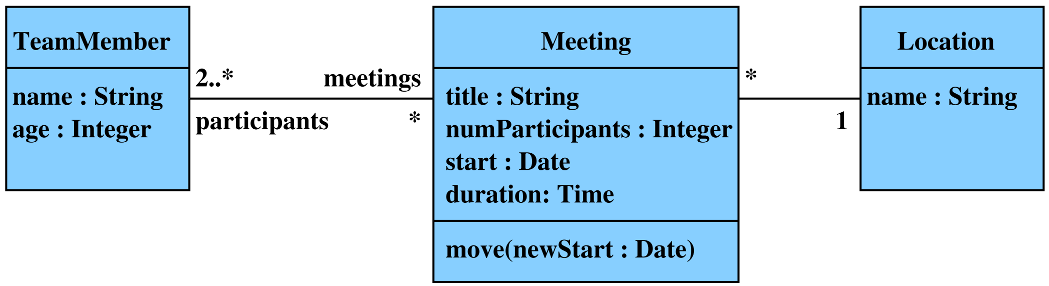
—— " —— age (self) ≥ 18

—— " —— ≥ (age (self), 18)

↳ all Instances$_{TM}$ -> forAll (self : TM | ≥ (age (self), 18))

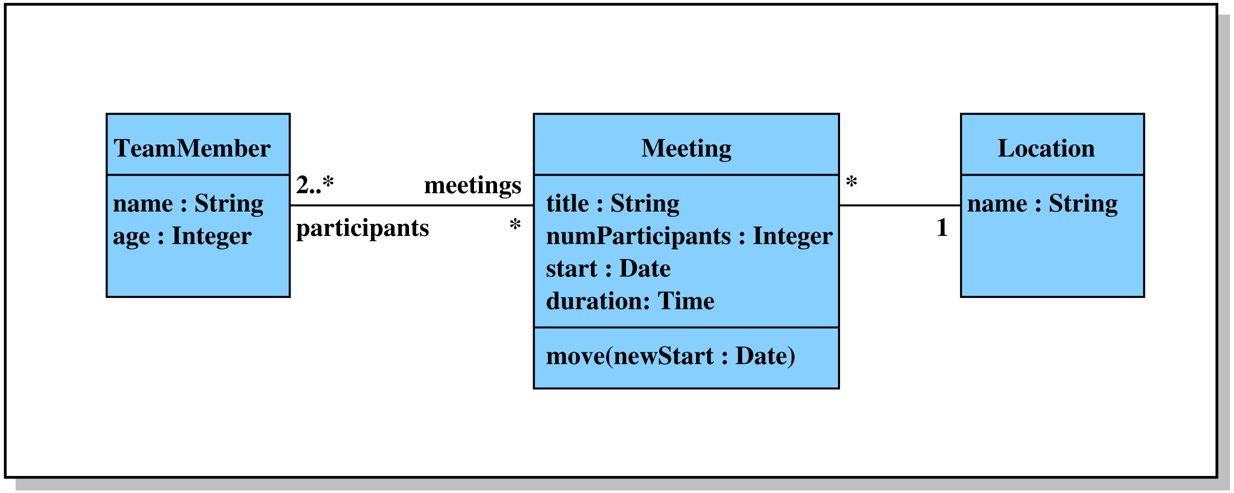
Examples (from lecture “Softwaretechnik 2008”)

OCL/Mehr Navigation/Beispiele



- `context Meeting`
  - `inv: self.participants->size() = numParticipants`
- `context Location`
  - `inv: name="Lobby" implies meeting->isEmpty()`

Example (from lecture "Softwaretechnik 2008")



- context $Meeting$ inv :

$$participants \text{ -> } \mathsf{iterate}(i : TeamMember; n : Int = 0 \mid n + i\,.\,age)$$

$$/ participants \text{ -> } \mathsf{size}() > 25$$

“Not Interesting”

Among others:

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

References

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.

[Warmer and Kleppe, 1999] Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language*. Addison-Wesley.