

# Software Design, Modelling and Analysis in UML

## Lecture 07: A Type System for Visibility

2013-11-18

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

### Contents & Goals

- Last Lecture:**
- Representing class diagrams as (extended) signatures — for the moment
  - without associations. (see Lecture 03)
  - And: in Lecture 03, implicit assumption of well-typedness of OCL expressions.

**This Lecture:**

- Educational Objectives:** Capabilities for following tasks/questions:
  - Is this OCL expression well typed or not? Why?
  - How/in what form did we define well-typedness?
  - What is visibility good for?
- Content:**
  - Recall: type theory/static type systems,
  - Well-typedness for OCL expression,
  - Visibility as a matter of well-typedness.

### Extended Classes

- From now on, we assume that each class  $C \in \mathcal{C}$  has:
- a finite (possibly empty) set  $S_C$  of **stereotypes**,
  - a boolean flag  $a \in \{B\}$  indicating whether  $C$  is **abstract**,
  - a boolean flag  $t \in \{B\}$  indicating whether  $C$  is **active**.
- We use  $S_C$  to denote the set  $\bigcup_{C \in \mathcal{C}} S_C$  of stereotypes in  $\mathcal{C}$ .  
 (Alternatively, we could add a set  $S$  as 5th component to  $\mathcal{C}$  to provide the stereotypes (names of stereotypes) to choose from. But: too unimportant to care.)
- Convention:**
- We write  $\langle C, S_C, a, t \rangle \in \mathcal{C}$  when we want to refer to all aspects of  $C$ .
  - If the new aspects are irrelevant (for a given context), we simply write  $C \in \mathcal{C}$ ; i.e. old definitions are still valid.

### Extended Attributes

- From now on, we assume that each attribute  $v \in Y$  has (in addition to the type):
- a **visibility**  $\xi \in \{\text{public, private, protected, package}\}$
  - an **initial value expr.** given as a word from **language for initial values**, e.g. OCL expressions. (If using Java as **action language** (later) Java expressions would be fine.)
  - a finite (possibly empty) set of **properties**  $P_v$ . We define  $\mathcal{P}$  analogously to stereotypes.
- Convention:**
- We write  $(v : \tau, \xi, \text{expr}, a, P_v) \in Y$  when we want to refer to all aspects of  $v$ .
  - Write only  $v : \tau$  or  $v$  if details are irrelevant.

### Recall: From Class Boxes to Extended Signatures

### From Class Boxes to Extended Signatures

A class box  $n$  induces an (extended) signature class as follows:

where

$$V(n) := \{ (v_1 : \tau_1, \xi_1, \text{expr}_1, a_1, P_{v_1}), \dots, (v_m : \tau_m, \xi_m, \text{expr}_m, a_m, P_{v_m}) \}$$

where

- "abstract" is determined by the box:  $\text{true}$  if  $n = \boxed{C}$  or  $n = \boxed{C \ \&A}$ ,  $\text{false}$  otherwise
- "active" is determined by the frame:  $\text{true}$  if  $n = \boxed{C}$  or  $n = \boxed{C \ \&A}$ ,  $\text{false}$  otherwise

Type Theory

Recall: In lecture 03, we introduced OCL expressions with **types**, for instance

```

expr ::= w | ... logical variable w
      | true | false | Bool | ... constants
      | 0 | -1 | 1 | ... | Int | ... constants
      | expr1 + expr2 | Int × Int → Int | ... operation
      | size(expr1) | Set(τ) → Int
      | not expr | Bool → Bool
      | not w
      | size(w)
  
```

**Approach:** Derivation System, that is, a finite set of derivation rules. We then say *expr* is **well-typed** if and only if we can derive

$$A, C \vdash \text{expr} : \tau \quad (\text{read: 'expression expr has type } \tau')$$

for some OCL type  $\tau$ , i.e.  $\tau \in T_D \cup T_E \cup \{\text{Set}(t_0) \mid t_0 \in T_D \cup T_E\}, C \in \mathcal{C}$ .

A Type System for OCL

A Type System for OCL

We will give a finite set of **type rules** (a **type system**) of the form

$$\frac{\text{"premisses"} \quad \text{"side condition"} \quad \text{"name"}}{\text{conclusion}}$$

These rules will establish well-typedness statements (**type sentences**) of three different **qualities**:

- (i) Universal well-typedness:
 
$$\frac{}{\vdash \text{expr} : \tau}$$

$$\vdash 1 + 2 : \text{Int}$$
- (ii) Well-typedness in a **type environment**  $A$ : (for logical variables)
 
$$\frac{A \vdash \text{expr} : \tau}{\text{self} : \tau; C \vdash \text{self}.v : \text{Int}}$$
- (iii) Well-typedness in **type environment**  $A$  and **context**  $B$ : (for visibility)
 
$$\frac{A, B \vdash \text{expr} : \tau}{\text{self} : \tau; C, C' \vdash \text{self}.v : \text{Int}}$$

Constants and Operations

- If *expr* is a **boolean constant**, then *expr* is of type *Bool*:
 
$$(BOOL) \quad \frac{}{\vdash B : \text{Bool}} \quad B \in \{\text{true}, \text{false}\}$$
- If *expr* is an **integer constant**, then *expr* is of type *Int*:
 
$$(INT) \quad \frac{}{\vdash N : \text{Int}} \quad N \in \{0, 1, -1, \dots\}$$

• If *expr* is the application of **operation**  $\omega : \tau_1 \times \dots \times \tau_n \rightarrow \tau$  to expressions  $\text{expr}_1, \dots, \text{expr}_n$  which are of type  $\tau_1, \dots, \tau_n$ , then *expr* is of type  $\tau$ :

$$(FUN) \quad \frac{\vdash \text{expr}_1 : \tau_1 \quad \dots \quad \vdash \text{expr}_n : \tau_n \quad \omega : \tau_1 \times \dots \times \tau_n \rightarrow \tau}{\vdash \omega(\text{expr}_1, \dots, \text{expr}_n) : \tau} \quad n \geq 1, \omega \notin \text{dir}(\mathcal{O})$$

(Note: this rule also covers  $\text{isEmpty}$ ,  $\text{isEmpty}$ , and  $\text{size}$ .)

Constants and Operations Example

(BOOL)	$\frac{}{\vdash B : \text{Bool}}$	$B \in \{\text{true}, \text{false}\}$
(INT)	$\frac{}{\vdash N : \text{Int}}$	$N \in \{0, 1, -1, \dots\}$
(FUN)	$\frac{\vdash \text{expr}_1 : \tau_1 \quad \dots \quad \vdash \text{expr}_n : \tau_n \quad \omega : \tau_1 \times \dots \times \tau_n \rightarrow \tau}{\vdash \omega(\text{expr}_1, \dots, \text{expr}_n) : \tau}$	$n \geq 1, \omega \notin \text{dir}(\mathcal{O})$

Example:

- not true
 
$$(FUN) \quad \frac{\text{true} : \text{Bool} \quad \text{not} : \text{Bool} \rightarrow \text{Bool}}{\vdash \text{not true} : \text{Bool}}$$
- true + 3
 
$$(FUN) \quad \frac{\text{true} : \text{Int} \quad \text{+} : \text{Int} \times \text{Int} \rightarrow \text{Int}}{\vdash \text{true} + 3 : \text{Int}}$$

### Type Environment

- **Problem:** Whether  $w + 3$  is well-typed or not depends on the type of logical variable  $w \in W$ .
- **Approach: Type Environments**

**Definition.** A **type environment** is a (possibly empty) finite sequence of type declarations for a given set  $W$  of logical variables and types  $T$  is defined by the grammar

$$A ::= \emptyset \mid A, w : \tau$$

where  $w \in W, \tau \in T$ .

**Clear:** We use this definition for the set of OCL logical variables  $W$  and the types  $T = T_D \cup T_C \cup \{Set(\tau_0) \mid \tau_0 \in T_D \cup T_C\}$ .

### Environment Introduction and Logical Variables

- If  $expr$  is of type  $\tau$ , then it is of type  $\tau$  in any type environment:

$$(EnvIntro) \frac{}{A \vdash expr : \tau}$$

- Care for logical variables in sub-expressions of operator application:

$$(FuncApp) \frac{A \vdash expr_1 : \tau_1, \dots, A \vdash expr_n : \tau_n, \quad w : \tau_1 \times \dots \times \tau_n \rightarrow \tau, \quad n \geq 1, w \notin \text{dom}(\theta)}{A \vdash w(expr_1, \dots, expr_n) : \tau}$$

- If  $expr$  is a logical variable such that  $w : \tau$  occurs in  $A$ , then we say  $w$  is of type  $\tau$ .

$$(Var) \frac{w : \tau \in A}{A \vdash w : \tau}$$

### Type Environment Example

$$(EnvIntro) \frac{}{A \vdash expr : \tau}$$

$$(FuncApp) \frac{A \vdash expr_1 : \tau_1, \dots, A \vdash expr_n : \tau_n, \quad w : \tau_1 \times \dots \times \tau_n \rightarrow \tau, \quad n \geq 1, w \notin \text{dom}(\theta)}{A \vdash w(expr_1, \dots, expr_n) : \tau}$$

$$(Var) \frac{}{A \vdash w : \tau}$$

- Example:  $w + 3, A = w : Int$



### All Instances and Attributes in Type Environment

- If  $expr$  refers to all instances of class  $C$ , then it is of type  $Set(\tau_C)$ .

$$(AllInst) \frac{}{\text{allinstances}_C : Set(\tau_C)}$$

- If  $expr$  is an attribute access of an attribute of type  $\tau$  for an object of  $C$  as denoted by  $expr_1$ , then the premise is that  $expr_1$  is of type  $\tau_C$ :

$$(Attr0) \frac{A \vdash expr_1 : \tau_C, \quad \tau : \tau_C \in \text{attr}(C), \quad \tau \in \mathcal{D}}{A \vdash expr_1.\tau : \tau}$$

$$(Attr0_1) \frac{A \vdash expr_1 : \tau_C, \quad \tau_1 : D_{1,1} \in \text{attr}(C)}{A \vdash \tau_1(expr_1) : \tau_D}$$

$$(Attr0_2) \frac{A \vdash expr_1 : \tau_C, \quad \tau_2 : D_2 \in \text{attr}(C)}{A \vdash \tau_2(expr_1) : Set(\tau_D)}$$

### Attributes in Type Environment Example

$$(Attr0) \frac{}{A \vdash expr_1 : \tau_C}, \quad \tau : \tau_C \in \text{attr}(C), \quad \tau \in \mathcal{D}$$

$$(Attr0_1) \frac{}{A \vdash expr_1 : \tau_C}, \quad \tau_1 : D_{1,1} \in \text{attr}(C)$$

$$(Attr0_2) \frac{}{A \vdash expr_1 : \tau_C}, \quad \tau_2 : D_2 \in \text{attr}(C)$$



- $self : \tau_C \vdash self.x : Int$  well-typed by  $(Attr0), (Env)$
- $self : \tau_C \vdash self.y : Int$  well-typed by  $(Attr0), (Env)$
- $self : \tau_C \vdash self.z : Int$  well-typed by  $(Attr0_1), (Env)$
- $self : \tau_C \vdash self.x.z : Int$  not well-typed, get stuck when applying  $(Attr0_1)$
- $self : \tau_C \vdash self.x.y : Int$  well-typed by  $(Attr0_2), (Env), (Env)$

### Iterate

- If  $expr$  is an iterate expression, then
- the iterator variable has to be type consistent with the base set, and
- initial and update expressions have to be consistent with the result variable.

$$(Iter) \frac{A \vdash expr_1 : Set(\tau), \quad A \vdash expr_2 : \tau_2, \quad A \vdash expr_3 : \tau_2}{A \vdash expr_1 \rightarrow \text{iterate}(var : \Delta : \tau_0 : \tau_2 = expr_2 | expr_3) : \tau_2}$$

where  $A' = A \oplus (var : \tau_0) \oplus (var_2 : \tau_2)$

variable typing of the code up in  $A'$

$(var : \tau_0, var_2 : \tau_2)$  under another scope

code scope

$all(x) \rightarrow all(x) \cdot (1 \rightarrow 1)$

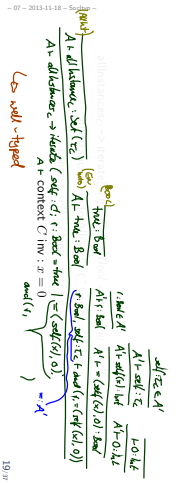
$(1 \rightarrow 1)$  under scope

### Henke Example

(AllEnv)	$\frac{}{F \text{ allEnvEncr} : \text{Set}(\alpha)}$	(All)	$\frac{A \vdash \text{expr}_1 : \tau_1}{A \vdash \text{encr}_1 : \tau}$
(Let)	$\frac{A \vdash \text{expr}_1 : \text{Set}(C) \quad A \vdash \text{expr}_2 : \tau \quad A \vdash \text{expr}_3 : \tau_2}{A \vdash \text{encr} \rightarrow \text{SetEnv}(\text{env} : \tau_1 \oplus \text{env}_2 : \tau_2) \quad A \vdash \text{encr}_3 : \tau_2}$		

where  $A' = A \oplus (\text{env} : \tau_1) \oplus (\text{env}_2 : \tau_2)$

Example:  $\mathcal{C} = (\text{Int}), (C; \text{Int}), (C \mapsto (x))$



### One Possible Extension: Implicit Casts

- We may wish to have  $\vdash 1 \text{ and } \text{false} : \text{Bool}$  (\*)
- In other words:** We may wish that the type system allows to use  $0, 1 : \text{Int}$  instead of  $\text{true}$  and  $\text{false}$  without breaking well-typedness.
- Then just have a rule:  $\frac{}{\text{Cast}} \frac{A \vdash \text{expr} : \text{Int}}{A \vdash \text{expr} : \text{Bool}}$
- With (Cast) (and (Int), and (Bool), and (Fun)), we can derive the sentence (\*), thus conclude well-typedness.
- But:** that's only half of the story — the definition of the interpretation function  $I$  that we have is not prepared, it doesn't tell us what (\*) means...

### First Recapitulation

- $I$  only defined for well-typed expressions.
- What can hinder something, which looks like a well-typed OCL expression, from being a well-typed OCL expression...?
- Raw syntax error:  $\mathcal{C} = (\text{Int}, D); (C; \text{Int}, n); (C \mapsto (n), D \mapsto (x))$  context  $C; \text{false}$  *"raw" missing*
- Subtle syntax error (depends on signature)  $\text{enc}$  in  $\mathcal{C}$  context  $C; \text{Int}; \beta = 0$
- Type error:  $\text{context self} : C; \text{Int}; \text{self}; n = \text{self}; n; x$  *"no" missing*

### Implicit Casts Cont'd

- So, why isn't there an interpretation for (1 and false)?
- First of all, we have (syntax)  $\text{expr}_1$  and  $\text{expr}_2 : \text{Bool} \times \text{Bool} \rightarrow \text{Bool}$
- Thus,  $I(\text{and}) : I(\text{Bool}) \times I(\text{Bool}) \rightarrow I(\text{Bool})$  where  $I(\text{Bool}) = \{\text{true}, \text{false}\} \cup \{\perp_{\text{Bool}}\}$ .
- By definition,  $I[\text{and}][\alpha, \beta] = I(\text{and})( I[\alpha], I[\beta] )$ , and there we're stuck

### Casting in the Type System

- Explicitly define  $I[\text{and}](\text{expr}_1, \text{expr}_2)[\alpha, \beta] = \begin{cases} b_1 \wedge b_2 & , \text{ if } b_1 \neq \perp_{\text{Bool}} \neq b_2 \\ \perp_{\text{Bool}} & , \text{ otherwise} \end{cases}$
- where  $b_1 := I[\text{bool}][\text{expr}_1][\alpha, \beta]$ ,  $b_2 := I[\text{bool}][\text{expr}_2][\alpha, \beta]$ , and where  $I[\text{bool}] : I(\text{Int}) \cup I(\text{Bool}) \rightarrow I(\text{Bool})$
- $$x \mapsto \begin{cases} \text{true} & , \text{ if } x \in \{\text{true}\} \cup \{I(1)\} \cup \{I(0)\} \\ \text{false} & , \text{ if } x \in \{\text{false}\} \\ \perp_{\text{Bool}} & , \text{ otherwise} \end{cases}$$

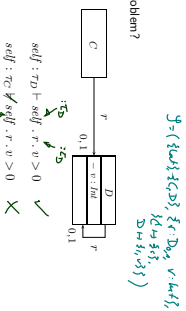
Bottomline

- There are **wishes** for the type-system which require changes in both the definition of  $\tau$  and the type system. In most cases not difficult, but tedious.
- **Note:** the extension is still a basic type system.
- **Note:** OCL has a far more elaborate type system which in particular addresses the relation between  $Dom$  and  $Int$  (cf. [OCL'S 2009]).

Visibility in the Type System

Context

- **Example:** A problem?



- That is, whether an expression involving attributes with visibility is well-typed **depends** on the class of objects for which it is evaluated.
- **Therefore:** well-typedness in type environment  $\mathcal{A}$  and context  $B \in \mathcal{E}$ :  
 $A, B \vdash expr : \tau$
- In particular: prepare to treat "protected" later (when doing inheritance).

Attribute Access in Context

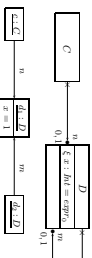
- If  $expr$  is of type  $\tau$  in a type environment, then it is in **any context**:  
 $(Context) \frac{A, B \vdash expr : \tau}{A, B \vdash expr : \tau}$
- **Accessing attribute  $v$  of a  $C$ -object via logical variable  $w$  is well-typed if**
  - $w$  is of type  $\tau_B$
  - $v$  is public, or  $expr_1$  denotes an object of class  $B$ .

$(Attr_1) \frac{A, B \vdash w : \tau_B \quad (v : \tau, \xi, expr_0, R_0) \in attr(B)}{A, B \vdash (w.v) : \tau}$   
 $(Attr_2) \frac{A, B \vdash expr_1 : \tau_C \quad (v : \tau, \xi, expr_0, R_0) \in attr(C) \quad \xi = +_v \text{ or } C = B}{A, B \vdash (expr_1.v) : \tau}$

Visibility — The Intuition

$\mathcal{D} = \{(w) \vdash (C, D), (v : D_{w_1}, m : D_{w_2}, (e : int, \xi, expr_0, \theta)), (C \rightarrow (v) \vdash (D \rightarrow (x, m)))\}$

Let's study an Example:



Assume  $w_1, \xi_1, x_1 = 0$  and  $w_2, \xi_2$  are logical variables. Which of the following syntactically correct (?) OCL expressions shall we consider to be well-typed?

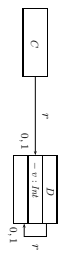
$\xi$ of $x$ :	public	private	protected	package
$w_1, m, x = 0$	✓	✗	✗	✗
$w_2, m, x = 0$	✓	✗	✗	✗
$x.(m(x.v)) > 0$	?	✗	✗	✗

Context in Operator Application

- Operator Application:  
 $(App_2) \frac{A, B \vdash expr_1 : \tau_1 \dots A, B \vdash expr_n : \tau_n \quad w : \tau_1 \times \dots \times \tau_n \rightarrow \tau \quad n \geq 1, w \notin attr(\mathcal{E})}{A, B \vdash w(expr_1, \dots, expr_n) : \tau}$
- Iterate:  
 $(Iter_1) \frac{A, B \vdash expr_1 : Set(C) \quad \mathcal{A}', B \vdash expr_2 : \tau_2 \quad \mathcal{A}', B \vdash expr_3 : \tau_3}{A, B \vdash expr_1 \rightarrow Iterate(w : \tau_1 \times w_2 : \tau_2 = expr_2 \mid expr_3) : \tau_3}$   
 where  $\mathcal{A}' = \mathcal{A} \oplus (w_1 : \tau_1) \oplus (w_2 : \tau_2)$ .

## Attribute Access in Context Example

(Context **Exp**)  
 $\lambda A.B \rightarrow expr : \tau$   
 (Attr)  
 $\lambda A.B \rightarrow expr_1 : \tau_C$  ( $\forall \tau : \tau \in \text{expr}_0, P_0 \in \text{Attr}(C)$ )  
 $\lambda A.B \rightarrow v(\text{expr}_1) : \tau$  ( $\forall \tau : \tau \in \text{expr}_0, P_0 \in \text{Attr}(C)$ ,  
 $\xi = \tau_0, \eta = \xi$  and  $\xi = B$ )



Example:

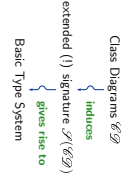
$self : \tau_C \quad \vdash \text{self} . r . v > 0$

## Recapitulation

## The Semantics of Visibility

- **Observation:**
- Whether an expression **does** or **does not** respect visibility is a matter of well-typedness **only**.
- We only evaluate (= apply  $I$  to) **well-typed** expressions.
- We need **not** adjust the interpretation function  $I$  to support visibility.

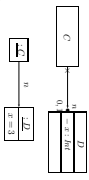
## Recapitulation



- We extended the type system for
  - casts (requires change of  $I$ ) and **new** *object* *statics*
  - **visibility** (no change of  $I$ ).
  - **Later:** **navigability** of associations.
- Good: well-typedness is decidable for these type-systems. That is, we can have automatic tools that check, whether OCL expressions in a model are well-typed.

## What is Visibility Good For?

- Visibility is a property of attributes — is it useful to consider it in OCL?
- In other words: given the picture above, is it **useful** to state the following invariant (even though  $x$  is private in  $D$ )?



## 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.