Contents & Goals

Last Lectures:
- class diagram — except for associations; visibility within OCL type system

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - Please explain this class diagram with associations.
  - Which annotations of an association arrow are semantically relevant?
  - What’s a role name? What’s it good for?
  - What’s “multiplicity”? How did we treat them semantically?
  - What is “reading direction”, “navigability”, “ownership”, . . . ?
  - What’s the difference between “aggregation” and “composition”?

Content:
- Complete visibility
- Study concrete syntax for “associations”.
- (Temporarily) extend signature, define mapping from diagram to signature.
- Study effect on OCL.
- Where do we put OCL constraints?
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 : Int instead of true and false without breaking well-typedness.

- Then just have a rule:

\[ \begin{align*}
\text{(Cast)} & \quad \frac{A \vdash expr : \text{Int}}{A \vdash expr : \text{Bool}}
\end{align*} \]

- With (Cast) (and (Int), and (Bool), and (Fun0)), we can derive the sentence (\(\ast\)), 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 \(\ast\) means...
So, why isn’t there an interpretation for \(1 \text{ and } \text{false}\)?

- First of all, we have (syntax)
  \[\text{expr}_1 \text{ and } \text{expr}_2 : \text{Bool} \times \text{Bool} \rightarrow \text{Bool}\]

- Thus,
  \[I(\text{and}) : I(\text{Bool}) \times I(\text{Bool}) \rightarrow I(\text{Bool})\]
  where \(I(\text{Bool}) = \{\text{true, false}\} \cup \{\bot_{\text{Bool}}\}\).

- By definition,
  \[I[[1 \text{ and } \text{false}]](\sigma, \beta) = I(\text{and})( I[[1]](\sigma, \beta), I[[\text{false}]](\sigma, \beta) ),\]
  and there we're stuck.

Implicit Casts: Quickfix

- Explicitly define
  \[I[[\text{and}(\text{expr}_1, \text{expr}_2)]](\sigma, \beta) := \begin{cases} b_1 \wedge b_2, & \text{if } b_1 \neq \bot_{\text{Bool}} \neq b_2 \\ \bot_{\text{Bool}}, & \text{otherwise} \end{cases}\]

where

- \(b_1 := \text{toBool}(I[[\text{expr}_1]](\sigma, \beta))\),
- \(b_2 := \text{toBool}(I[[\text{expr}_2]](\sigma, \beta))\),

and where

\[\text{toBool} : I(\text{Int}) \cup I(\text{Bool}) \rightarrow I(\text{Bool})\]

\[x \mapsto \begin{cases} \text{true}, & \text{if } x \in \{\text{true}\} \cup \{1\ldots, 9\} \cup \{\bot_{\text{Int}}\} \\ \text{false}, & \text{if } x \in \{\text{false}, 0\} \\ \bot_{\text{Bool}}, & \text{otherwise} \end{cases}\]
**Bottomline**

- There are wishes for the type-system which require changes in both, the definition of \( I \) 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 \( \text{Bool} \) and \( \text{Int} \) (cf. \([?]\)).

---

**Visibility in the Type System**
Visibility — The Intuition

Let’s study an Example:

\[ \mathcal{S} = \{ \{ \text{Int} \}, \{ C, D \}, \{ n : D_{0,1}, m : D_{0,1}, (x : \text{Int}, \xi, \text{expr}_0, \emptyset) \}, \{ C \rightarrow \{ n \}, D \rightarrow \{ x, m \} \} \]

and

\[ \xi x : \text{Int} = \text{expr}_0 \]

Assume \( w_1 : \tau_C \) and \( w_2 : \tau_D \) are logical variables. Which of the following syntactically correct (?) OCL expressions shall we consider to be well-typed?

### Table: Well-typedness of OCL Expressions

<table>
<thead>
<tr>
<th>( w_1 \cdot n \cdot x = 0 )</th>
<th>public</th>
<th>private</th>
<th>protected</th>
<th>package</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>❌ later</td>
<td>not</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( w_2 \cdot m \cdot x = 0 )</th>
<th>public</th>
<th>private</th>
<th>protected</th>
<th>package</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>❌ later</td>
<td>not</td>
<td></td>
</tr>
</tbody>
</table>

Context

- **Example:** A problem?

\[ C \quad \begin{array}{ccc} \leftrightsquigarrow & r & \rhrightrightarrows \end{array} \quad D \quad \begin{array}{ccc} \leftrightsquigarrow & \downarrow & v : \text{Int} \end{array} \]

self : \( \tau_D \vdash \text{self} . r . v > 0 \) ✔

self : \( \tau_C \not\vdash \text{self} . r . v > 0 \) ❌

- 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 \( B \in \mathcal{C} \):

\[
\mathcal{A}, B \vdash \text{context} \quad \text{well typed}
\]

- In particular: prepare to treat “protected” later (when doing inheritance).
Attribute Access in Context

- If `expr` is of type `τ` in a type environment, then it is in any context:

\[
\frac{A \vdash \text{expr} : \tau}{A, B \vdash \text{expr} : \tau}
\]

Accessing attribute `v` of a `C`-object via logical variable `w` is well-typed if
- `v` is public, or `w` is of type `τ`:

\[
A \vdash w : \tau_B
\]

\[
A, B \vdash v(w) : \tau
\]

Accessing attribute `v` of a `C`-object via expression `expr_1` is well-typed in context `B` if
- `v` is public, or `expr_1` denotes an object of class `B`:

\[
A, B \vdash \text{expr}_1 : \tau_C
\]

\[
A, B, v(\text{expr}_1) : \tau, \langle v : \tau, \xi, \text{expr}_0, P \rangle \in \text{atr}(C), \xi = +, \text{or } C = B
\]

- Accessing `C_{0,1}`- or `C_\ast`-typed attributes: similar.

Context in Operator Application

- Operator Application:

\[
A, B \vdash \text{expr}_1 : \tau_1 \ldots A, B \vdash \text{expr}_n : \tau_n
\]

\[
A, B \vdash \omega(\text{expr}_1, \ldots, \text{expr}_n) : \tau
\]

\[
n \geq 1, \omega \notin \text{atr}(\emptyset)
\]

- Iterate:

\[
A, B \vdash \text{expr}_1 : \text{Set}(\tau_1) \quad A, B \vdash \text{expr}_2 : \tau_2 \quad A, B \vdash \text{expr}_3 : \tau_2
\]

\[
A, B \vdash \text{expr}_1 \rightarrow\text{iterate}(w_1 : \tau_1, w_2 : \tau_2 = \text{expr}_2 | \text{expr}_3) : \tau_2
\]

where `A' = A \oplus (w_1 : \tau_1) \oplus (w_2 : \tau_2)`
Attribute Access in Context Example

\[
\frac{\text{Expr: } \tau}{\text{A, B} \vdash \text{expr : } \tau}
\]

Example:

\[
\frac{\text{Attr 1}}{\text{A, B} \vdash v(\text{expr}_1) : \tau}
\]

\[
\frac{\text{v : } \tau, \xi, \text{expr}_0, P_v}{\langle \text{v : } \tau, \xi, \text{expr}_0, P_v \rangle \in \text{atr}(C), \xi = +, \text{or } \xi = - \text{ and } C = B}
\]

\[
\frac{\text{v : } \text{Int}}{\text{0, 1} \vdash \text{v : } \text{Int}}
\]

\[
\frac{\text{0, 1}}{\text{0, 1} \vdash \text{0, 1}}
\]

\[
\frac{\text{self : } \tau}{\text{self : } \tau}
\]

\[
\frac{\text{self : } \tau}{\text{self : } \tau}
\]

\[
\frac{\text{self : } \tau}{\text{self : } \tau}
\]

\[
\frac{\text{0, 1}}{\text{0, 1} \vdash \text{0, 1}}
\]

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.
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 $D$)

$$\text{context } C \, \text{inv: } n \cdot x > 0 ?$$

It depends. (cf. [?], Sect. 12 and 9.2.2)

Constraints and pre/post conditions:

Visibility is sometimes not taken into account. To state "global" requirements, it may be adequate to have a "global view", be able to look into all objects.

But: visibility supports "narrow interfaces", "information hiding", and similar good design practices. To be more robust against changes, try to state requirements only in the terms which are visible to a class.

Rule-of-thumb: if attributes are important to state requirements on design models, leave them public or provide get-methods (later).

Guards and operation bodies:

If in doubt, yes ($= \text{do take visibility into account}$).

Any so-called action language typically takes visibility into account.

Recapitulation
Recapitulation

Class Diagrams $\mathcal{C} \mathcal{P}$

\[ \downarrow \text{induces} \]

extended (!) signature $\mathcal{I}(\mathcal{C} \mathcal{P})$

\[ \downarrow \text{gives rise to} \]

Basic Type System

- We extended the type system for
  - casts (requires change of $I$) and
  - 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.

Associations: Syntax
What Do We (Have to) Cover?

An association has

- a name,
- a reading direction, and
- at least two ends.

Each end has

- a role name,
- a multiplicity,
- a set of properties, such as unique, ordered, etc.
- a qualifier, (we will not test)
- a visibility,
- a navigability,
- an ownership,
- and possibly a diamond (exercise)

Wanted: places in the signature to represent the information from the picture.
(Temporarily) Extend Signature: Associations

Only for the course of Lectures 07/08 we assume that each attribute in \( V \)

- **either** is \( \langle v : \tau, \xi, expr_0, P_v \rangle \) with \( \tau \in \mathcal{F} \) (as before),
- **or** is an association of the form

\[
\{ r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \\
\cdots \\
\langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \}\]

where

- \( n \geq 2 \) (at least two ends),
- \( r, role_i \) are just names, \( C_i \in \mathcal{C}, \ 1 \leq i \leq n \),
- the multiplicity \( \mu_i \) is an expression of the form

\[
\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \quad (N, M \in \mathbb{N})
\]

- \( P_i \) is a set of properties (as before),
- \( \xi \in \{+, -, #, \sim\} \) (as before),
- \( \nu_i \in \{\times, -, >\} \) is the navigability,
- \( o_i \in B \) is the ownership.

Alternative syntax for multiplicities:

\[
\mu ::= N..M \mid N..* \mid \mu, \mu \quad (N, M \in \mathbb{N} \cup \{\ast\})
\]

and define \( * \) and \( N \) as abbreviations. \( 0..* \) or \( 1..* \)

Note: \( N \) could abbreviate \( 0..N, 1..N \), or \( N..N \). We use last one.

- \( r, role_i \) are just names, \( C_i \in \mathcal{C}, \ 1 \leq i \leq n \),
- the multiplicity \( \mu_i \) is an expression of the form

\[
\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \quad (N, M \in \mathbb{N})
\]

- \( P_i \) is a set of properties (as before),
- \( \xi \in \{+, -, #, \sim\} \) (as before),
- \( \nu_i \in \{\times, -, >\} \) is the navigability,
- \( o_i \in B \) is the ownership.
(Temporarily) Extend Signature: Basic Type Attributes

Also only for the course of lectures:

- we only consider basic type attributes to “belong” to a class (to appear in \( \text{atr}(C) \)),

- associations are not “owned” by a particular class (do not appear in \( \text{atr}(C) \)), but live on their own.

Formally: we only call

\[ (\mathcal{F}, \mathcal{C}, V, \text{atr}) \]

a signature (extended for associations) if

\[ \text{atr} : \mathcal{C} \to 2^{\{ v \in V | \forall r \in \mathcal{F} \}}. \]

From Association Lines to Extended Signatures

\[ o_i = \begin{cases} 1, & \text{if } C_i \\ 0, & \text{if } C_i \end{cases} \quad \nu_i = \begin{cases} x, & \text{if } x \\ -, & \text{if } C_i \\ >, & \text{if } C_i \end{cases} \]
**Association Example**

![Association Diagram]

**Signature:**

\[ S = \{ \{ \text{Int} \}, \{ C, D \}, \{ x : \text{Int} \}, \langle r : \{ C, 0..n, \{ \text{unique} \}, -r, x, r \rangle, \langle \text{D}, 0..n, \{ \text{unique} \}, +r, 0 \rangle \}, \{ \{ C \}, \{ D \} \} \]  

**What If Things Are Missing?**

Most components of associations or association end may be omitted. For instance [7, 17], Section 6.4.2, proposes the following rules:

- **Name:** Use \( A(C_1) \ldots A(C_n) \) if the name is missing.

  **Example:**
  
  ![Example Diagram]

  for \( C \rightarrow D \)

- **Reading Direction:** no default.

- **Role Name:** use the class name at that end in lower-case letters

  **Example:**
  
  ![Example Diagram]

  for \( C \rightarrow D \)

**Other convention:** (used e.g. by modelling tool Rhapsody)

- **Example:**
  
  ![Example Diagram]

  for \( C \rightarrow D \)
What If Things Are Missing?

- **Multiplicity:** 1
  In my opinion, it’s safer to assume 0..1 or ∗ if there are no fixed, written, agreed conventions (“expect the worst”).
- **Properties:** ∅
- **Visibility:** public
- **Navigability and Ownership:** not so easy. [?, 43]

“Various options may be chosen for showing navigation arrows on a diagram. In practice, it is often convenient to suppress some of the arrows and crosses and just show exceptional situations:

- Show all arrows and x’s. Navigation and its absence are made completely explicit.
- Suppress all arrows and x’s. No inference can be drawn about navigation. This is similar to any situation in which information is suppressed from a view.
- Suppress arrows for associations with navigability in both directions, and show arrows only for associations with one-way navigability.

In this case, the two-way navigability cannot be distinguished from situations where there is no navigation at all; however, the latter case occurs rarely in practice.”

Wait, If Omitting Things...

- ...is causing so much trouble (e.g. leading to misunderstanding), why does the standard say “In practice, it is often convenient...”? Is it a good idea to trade convenience for precision/unambiguity?

It depends.

- Convenience as such is a legitimate goal.
- In UML-As-Sketch mode, precision “doesn’t matter”, so convenience (for writer) can even be a primary goal.
- In UML-As-Blueprint mode, precision is the primary goal. And misunderstandings are in most cases annoying.

But: (even in UML-As-Blueprint mode)
If all associations in your model have multiplicity ∗, then it’s probably a good idea not to write all these ∗’s.
So: tell the reader about it and leave out the ∗’s.
References