Contents & Goals

Last Lectures:
- completed class diagrams... except for visibility and associations

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 is “multiplicity”? How did we treat them semantically?
  - What is “reading direction”, “navigability”, “ownership”,...?
  - What’s the difference between “aggregation” and “composition”?

- Content:
  - Study concrete syntax for “associations”.
  - (Temporarily) extend signature, define mapping from diagram to signature.
  - Study effect on OCL.
  - Btw.: where do we put OCL constraints?
The Intuition by Example

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

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>
<thead>
<tr>
<th>( \xi ) of ( x )</th>
<th>public</th>
<th>private</th>
<th>protected</th>
<th>package</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 \cdot n \cdot x = 0 )</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>later</td>
</tr>
<tr>
<td>( \times(n \cdot w_1) )</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>not</td>
</tr>
<tr>
<td>( w_2 \cdot m \cdot x = 0 )</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>later</td>
</tr>
<tr>
<td>( \times(m \cdot w_2) )</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>not</td>
</tr>
</tbody>
</table>
\[ S = (\{\text{Int}\}, \{C, D\}, \{r : D_{0.1}, \langle v : \text{Int}, \xi, \ast, \emptyset \rangle \}, \{C \mapsto \{r\}, D \mapsto \{v, r\}\}) \]

**Example:**

\[
\begin{array}{c}
\text{C} \\
\downarrow \quad r \\
0, 1 \\
\end{array}
\quad \begin{array}{c}
\text{D} \\
\downarrow \quad v : \text{Int} \\
\downarrow \quad r \\
0, 1 \\
\end{array}
\]

- \( \text{self}_D \cdot v > 0 \) *
- \( \text{self}_D \cdot r \cdot v > 0 \) *
- \( \text{self}_C \cdot r \cdot v > 0 \) ❌

That is, whether an expression involving attributes with visibility is well-typed depends on the class of objects for which it is evaluated.

**Attribute Access in Context**

**Recall:** attribute access in OCL Expressions, \( C, D \in \mathcal{C} \).

\[
\begin{align*}
\text{v(expr}_1) &: \tau_C \rightarrow \tau(v) \quad \text{v : } \tau(v) \in \text{atr}(C), \tau(v) \in \mathcal{S}, \\
\text{r}_1(expr}_1) &: \tau_C \rightarrow \tau_D \quad \text{r}_1 : D_{0.1} \in \text{atr}(C), \\
\text{r}_2(expr}_1) &: \tau_C \rightarrow \text{Set}(\tau_D) \quad \text{r}_2 : D_\ast \in \text{atr}(C),
\end{align*}
\]

**New rules:**

\[
\begin{align*}
\text{v(w)} &: \tau_C \rightarrow \tau(v) \quad \langle v : \tau, \xi, \text{expr}_0, P_\xi \rangle \in \text{atr}(C), \\
\text{r}_1(w) &: \tau_C \rightarrow \tau_D \quad \langle r_1 : D_{0.1}, \xi, \text{expr}_0, P_\xi \rangle \in \text{atr}(C), \\
\text{r}_2(w) &: \tau_C \rightarrow \text{Set}(\tau_D) \quad \langle r_1 : D_\ast, \xi, \text{expr}_0, P_\xi \rangle \in \text{atr}(C), \\
\text{v(expr}_1(w)) &: \tau_{C_2} \rightarrow \tau(v) \quad \langle v : \tau, \xi, \text{expr}_0, P_\xi \rangle \in \text{atr}(C), \text{expr}_1(w) : \tau_{C_2}, w : \tau_{C_1}, \text{ and } C_1 = C_2 \text{ or } \xi = +, \\
\text{r}_1(expr}_1(w)) &: \tau_{C_2} \rightarrow \tau_D \quad \langle v : D_{0.1}, \xi, \text{expr}_0, P_\xi \rangle \in \text{atr}(C), \text{expr}_1(w) : \tau_{C_2}, w : \tau_{C_1}, \text{ and } C_1 = C_2 \text{ or } \xi = +,
\end{align*}
\]
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 diagram above, is it useful to state the following invariant (even though $x$ is private in $D$)

context $C$ inv : $n \cdot x > 0$ ?

It depends.

(cf. [OMG, 2006], 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 (= do take visibility into account).
  Any so-called action language typically takes visibility into account.

References

