Contents & Goals

Last Lecture:
- Representing class diagrams as (extended) signatures — for the moment without associations: later.

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - Could you please map this class diagram to a signature?
  - What if things are missing?
  - Could you please map this signature to a class diagram?
  - What is the semantics of ‘abstract’?
  - What is visibility good for?

Content:
- Map class diagram to (extended) signature cont’d.
- Stereotypes — for documentation.
- Visibility as an extension of well-typedness.
Recall

Example Cont’d

\[
\langle \langle S_1, \ldots, S_k \rangle \rangle_{C} \xrightarrow{\xi_1 v_1 : T_1 = \text{expr}_1 \{P_{1,1}, \ldots, P_{1,m_1}\}} \ldots \xrightarrow{\xi_\ell v_\ell : T_\ell = \text{expr}_\ell \{P_{\ell,1}, \ldots, P_{\ell,m_\ell}\}} C(n) := \langle C, \{S_1, \ldots, S_k\}, a(n), t(n) \rangle
\]

\[
V(n) := \{v_1 : T_1, \xi_1, \text{expr}_1 \{P_{1,1}, \ldots, P_{1,m_1}\} ; \ldots ; (v_\ell : T_\ell, \xi_\ell, \text{expr}_\ell \{P_{\ell,1}, \ldots, P_{\ell,m_\ell}\})} \}
\]

\[
\alpha(n) := \{C \mapsto \{v_1, \ldots, v_\ell\}\}
\]

\[
\langle \langle S_{\text{Domain}}, +, \emptyset, \text{ordered} \rangle \rangle_{\text{CD}} \xrightarrow{D_2} \langle \langle S_{\text{Domain}}, +, \emptyset, \{\text{ordered}\} \rangle \rangle
\]
Is the Mapping a Function?

**Question:** Is $\mathcal{F}(\mathcal{C \mathcal{D}})$ well-defined?

There are two possible sources for problems:

1. A class $C$ may appear in multiple class diagrams:

   (i) $\mathcal{C}_1$:
   
   $\mathcal{C}_2$:
   
   Simply *forbid* the case (ii) — easy syntactical check on diagram.

2. An attribute $v$ may appear in multiple classes with different type:

   $\mathcal{C}$:
   
   $\mathcal{D}$:

   **Two approaches:**
   
   - Require *unique* attribute names.
     This requirement can easily be established (implicitly, behind the scenes) by viewing $v$ as an abbreviation for $C::v$ or $D::v$
     depending on the context. ($C::v::Bool$ and $D::v::Int$ are then unique.)
   - Subtle, formalist’s approach: observe that
     
     $(v::Bool, \ldots)$ and $(v::Int, \ldots)$ are different things in $V$. We don’t follow that path...
The semantics of a set of class diagrams $\mathcal{CD}$ is the induced signature $\mathcal{S}(\mathcal{CD})$.

The signature induces a set of system states $\Sigma^\mathcal{S}$ (given a structure $\mathcal{D}$).

- Do we need to redefine/extend $\mathcal{D}$? No.

(Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type $T$, i.e. the set $\mathcal{D}(T)$, would be determined by the class diagram, and not free for choice.)
Semantics

The semantics of a set of class diagrams $\mathcal{C} \mathcal{D}$ is the induced signature $\mathcal{S}(\mathcal{C} \mathcal{D})$.

The signature induces a set of system states $\Sigma^\mathcal{D}$ (given a structure $\mathcal{D}$).

- Do we need to redefine/extend $\mathcal{D}$? No.
  (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type $T$, i.e. the set $\mathcal{D}(T)$, would be determined by the class diagram, and not free for choice.)

- What is the effect on $\Sigma^\mathcal{D}$? Little.
  For now, we only remove abstract class instances, i.e.

  $$\sigma : \mathcal{D}(\mathcal{C}) \rightarrow (V \rightarrow (\mathcal{D}(\mathcal{I}) \cup \mathcal{D}(\mathcal{C}_{\ast})))$$

  is now only called system state if and only if, for all $\langle C, S_C, 1, t \rangle \in \mathcal{C}$,

  $$\text{dom}(\sigma) \cap \mathcal{D}(C) = \emptyset.$$

  With $a = 0$ as default “abstractness”, the earlier definitions apply directly. (We’ll revisit this when discussing inheritance.)

What About The Rest?

- Classes:
  - Active: not represented in $\sigma$.
    Later: relevant for behaviour, i.e., how system states evolve over time.
  - Stereotypes: in a minute.

- Attributes:
  - Initial value expression: not represented in $\sigma$.
    Later: provides an initial value as effect of “creation action”.
  - Visibility: not represented in $\sigma$.
    Later: viewed as additional typing information for well-formedness of actions; and with inheritance.
  - Properties: such as readOnly, ordered, composite (Deprecated in the standard.)
    - readOnly — later treated similar to visibility.
    - ordered — not considered in our UML fragment (→ sets vs. sequences).
    - composite — cf. lecture on associations.
Rhapsody Demo I

RECALL: SEND ME YOUR POOL-ACCOUNT NAME

Visibility
Assume $w_1 : \tau_C$ and $w_2 : \tau_D$ are logical variables.

Which of the following syntactically correct (? OCL expressions should 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>later</td>
<td>not</td>
</tr>
<tr>
<td>$w_2 \cdot m \cdot x = 0$</td>
<td></td>
<td></td>
<td>later</td>
<td>not</td>
</tr>
</tbody>
</table>
The Intuition by Example

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

\[ C : C \]

\[ D : D \]

\[ x : \text{Int} \]

\[ 0 \]

\[ 1 \]

\[ n \]

\[ m \]

\[ \xi \]

\[ \text{expr}_0 \]

\[ \emptyset \]

\[ C \mapsto \{n\} \]

\[ D \mapsto \{x, m\} \]

Assume \( w_1 : \tau_C \) and \( w_2 : \tau_D \) are logical variables.

**Which of the following syntactically correct (?) OCL expressions should 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>later</td>
<td>not</td>
</tr>
<tr>
<td>( w_2 \cdot m \cdot x = 0 )</td>
<td>✔</td>
<td>✗</td>
<td>later</td>
<td>not</td>
</tr>
</tbody>
</table>

**Context**

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

- **By example:**

\[ C \]

\[ D \]

\[ x : \text{Int} \]

\[ 0 \]

\[ 1 \]

\[ n \]

\[ m \]

\[ \xi \]

\[ \text{expr}_0 \]

\[ \emptyset \]

\[ C \mapsto \{n\} \]

\[ D \mapsto \{x, m\} \]

\[ \text{self}_D \cdot x > 0 \]

✔

\[ \text{self}_D \cdot m \cdot x > 0 \]

✔

\[ \text{self}_C \cdot n \cdot x > 0 \]

✗
\[ \mathcal{F} = \{ \{ \text{Int} \}, \{ C, D \}, \{ n : D_{0,1}, m : D_{0,1}, x : \text{Int}, \xi, \text{expr}_0, \emptyset \}, \{ C \mapsto \{ n \}, D \mapsto \{ x, m \} \} \]

- By example:

\[ C \quad \begin{array}{c}
\begin{array}{c}
\text{D} \\
0,1
\end{array}
\end{array} \]

\[ n \]

\[ m \]

\[ \text{self}_{D \cdot x} > 0 \quad \checkmark \]

\[ \text{self}_{D \cdot m \cdot x} > 0 \quad \checkmark \]

\[ \text{self}_{C \cdot n \cdot x} > 0 \quad \times \]

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

Visibility is ‘by class’ — not ‘by object’.

**Attribute Access in Context**

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

\[ v(\text{expr}_1) \quad : \quad \tau_C \rightarrow \tau_T \]

\[ r_1(\text{expr}_1) \quad : \quad \tau_C \rightarrow \tau_D \]

\[ r_2(\text{expr}_1) \quad : \quad \tau_C \rightarrow \text{Set}(\tau_D) \]

\[ v(\text{expr}_2) \quad : \quad \tau_C \rightarrow T \]

\[ r_1(\text{expr}_1) \quad : \quad \tau_C \rightarrow \tau_D \quad w : \tau_C, \quad r_1 : D_{0,1} \in \text{atr}(C), \]

\[ r_2(\text{expr}_1) \quad : \quad \tau_C \rightarrow \text{Set}(\tau_D) \quad r_2 : D_* \in \text{atr}(C), \]

New rules for well-typedness considering visibility:

- \( v(w) \quad : \quad \tau_C \rightarrow T \)

\[ w : \tau_C, \quad v : T \in \text{atr}(C), T \in \mathcal{F} \]

- \( r_1(w) \quad : \quad \tau_C \rightarrow \tau_D \)

\[ w : \tau_C, \quad r_1 : D_{0,1} \in \text{atr}(C) \]

- \( r_2(w) \quad : \quad \tau_C \rightarrow \text{Set}(\tau_D) \)

\[ w : \tau_C, \quad r_1 : D_* \in \text{atr}(C) \]

- \( v(\text{expr}_1(w)) \quad : \quad \tau_C \rightarrow T \)

\[ \langle v : T, \xi, \text{expr}_0, P \rangle \in \text{atr}(C), T \in \mathcal{F}, \]

\[ \langle v_1(\text{expr}_1) \rangle \quad : \quad \tau_C, \quad w : \tau_{C_1} \text{ and } C_1 = C, \text{ or } \xi = + \]

- \( r_1(\text{expr}_1(w)) \quad : \quad \tau_C \rightarrow \tau_D \)

\[ \langle r_1 : D_{0,1}, \xi, \text{expr}_0, P \rangle \in \text{atr}(C), \]

\[ \langle r_1(\text{expr}_1) \rangle \quad : \quad \tau_C, \quad w : \tau_{C_1} \text{ and } C_1 = C, \text{ or } \xi = + \]

- \( r_2(\text{expr}_1(w)) \quad : \quad \tau_C \rightarrow \text{Set}(\tau_D) \)

\[ \langle r_2 : D_*, \xi, \text{expr}_0, P \rangle \in \text{atr}(C), \]

\[ \langle r_2(\text{expr}_1) \rangle \quad : \quad \tau_C, \quad w : \tau_{C_1} \text{ and } C_1 = C, \text{ or } \xi = + \]
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.

  \[ \rightarrow \text{We need not adjust the interpretation function } I \text{ to support visibility.} \]

  Just decide: should we take visibility into account yes / no, and check well-typedness by the new / old rules.

\[ \begin{array}{c}
\vdash C \rightarrow T \\
\vdash r_1(w) \rightarrow \tau_D \\
\vdash v(expr_1(w)) \rightarrow T \\
\vdash r_1(expr_1(w)) \rightarrow \tau_D
\end{array} \]

- Example

  \[ \begin{array}{c}
  \vdash \text{self}_D \cdot x > 0 \rightarrow x(\text{self}_D) > 0 \text{ ok, by (i)} \\
  \vdash \text{self}_D \cdot m \cdot x > 0 \rightarrow x(\text{self}_D) > 0 \text{ ok, by (ii)} \\
  \vdash \text{self}_C \cdot n \cdot x > 0 \rightarrow x(\text{self}_C) > 0 \text{ ok, by (iii)} \\
  \end{array} \]

\[ \begin{array}{c}
\vdash \text{self}_D \cdot x > 0 \rightarrow x(\text{self}_D) > 0 \text{ ok, by (i)} \\
\vdash \text{self}_D \cdot m \cdot x > 0 \rightarrow x(\text{self}_D) > 0 \text{ ok, by (ii)} \\
\vdash \text{self}_C \cdot n \cdot x > 0 \rightarrow x(\text{self}_C) > 0 \text{ ok, by (iii)} \\
\end{array} \]
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)

\[ \text{context } C \text{ inv } : n.x > 0 \]

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", i.e. 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.

Stereotypes
Stereotypes as Labels or Tags

- **What are Stereotypes?**
  - Not represented in system states.
  - Not contributing to typing rules / well-formedness.

- **Oestereich (2006):**
  View stereotypes as (additional) “labelling” ("tags") or as “grouping”.

- Useful for documentation and model-driven development, e.g. code-generation:
  - **Documentation**: e.g. layers of an architecture.
    Sometimes, packages (cf. OMG (2011a,b)) are sufficient and “right”.
  - **Model Driven Architecture (MDA)**: later.

---

**Example: Stereotypes for Documentation**

- **Example**: Timing Diagram Viewer
  Schumann et al. (2008)

- Architecture has four layers:
  - core, data layer
  - abstract view layer
  - toolkit-specific view layer/widget
  - application using widget

  Stereotype “=” layer “=” colour.
Other Examples

- Use stereotypes ‘Team\textsubscript{1}’, ‘Team\textsubscript{2}’, ‘Team\textsubscript{3}’ and assign stereotype Team\textsubscript{i} to class C if Team\textsubscript{i} is responsible for class C.

- Use stereotypes to label classes with licensing information (e.g., LGPL vs. proprietary).

- Use stereotypes ‘Server\textsubscript{A}’, ‘Server\textsubscript{B}’ to indicate where objects should be stored.

- Use stereotypes to label classes with states in the development process like "under development", "submitted for testing", "accepted".

- etc. etc.

**Necessary:** a common idea of what each stereotype stands for.

(To be defined / agreed on by the team, not the job of the UML consortium.)

References
References


