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?
Casting in the Type System
One Possible Extension: Implicit Casts

- We **may wish** to have

  \[ \vdash 1 \text{ and } false : Bool \]  

  **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:

  \[
  (\text{Cast}) \quad \quad \frac{A \vdash expr : Int}{A \vdash expr : Bool}
  \]

- With (Cast) (and (Int), and (Bool), and (Fun\textsubscript{0})), 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...

\[ I(\text{and}) : I(\text{Int}) \times I(\text{Int}) \to I(\text{Bool}) \]
So, why isn’t there an interpretation for \((1 \text{ and } false)\)?

- First of all, we have (syntax)
  \[
  expr_1 \text{ and } expr_2 : \text{Bool} \times \text{Bool} \to \text{Bool}
  \]

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

- By definition,
  \[
  I[1 \text{ and } false](\sigma, \beta) = I(\text{and})(I[1](\sigma, \beta), I[false](\sigma, \beta))
  \]
  and \textbf{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

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

\[ x \mapsto \begin{cases} \text{true}, & \text{if } x \in \{\text{true}\} \cup I(\text{Int}) \setminus \{0, \bot_{\text{Int}}\} \\ \text{false}, & \text{if } x \in \{\text{false}, 0\} \\ \bot_{\text{Bool}}, & \text{otherwise} \end{cases} \]
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 $Bool$ and $Int$ (cf. [?]).
Visibility in the Type System
Visibility — The Intuition

Let’s study an Example:

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

and

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>✗️</td>
</tr>
<tr>
<td>( w_2 \cdot m \cdot x = 0 )</td>
<td>✔️</td>
<td>✗️</td>
<td>✔️</td>
<td>✗️</td>
</tr>
</tbody>
</table>

- privateness is by class, and not by object

• **Example:** A problem?

\[
\begin{align*}
C & \xrightarrow{r} D \\
& 0, 1 \\
& \begin{array}{c}
\text{D.object} \\
\text{D.get}
\end{array} \\
& \begin{array}{c}
\text{self} : \tau_D \vdash \text{self} \ . \ r \ . \ v > 0 \quad \checkmark \\
\text{self} : \tau_C \nvdash \text{self} \ . \ r \ . \ v > 0 \quad \times
\end{array}
\end{align*}
\]

• 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 \( A \) and context \( B \in \mathcal{C} \):

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

\( \text{order doesn't matter} \)

• 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:

$$\begin{array}{c}
A \vdash expr : τ \\
\hline
A \Box B \vdash expr : τ
\end{array}$$

\[\text{(ContextIntro)}\]

• Accessing attribute $v$ of a $C$-object via logical variable $w$ is well-typed if
  - $v$ is public, or $w$ is of type $τ_B$

\[\begin{array}{c}
A \vdash w : τ_B \\
\hline
A, B \vdash v(w) : τ
\end{array}\]

\[\text{(Attr}_1)\]

• 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 $B$:

\[\begin{array}{c}
A, B \vdash expr_1 : τ_C \\
\hline
A, B \vdash v(expr_1) : τ
\end{array}\]

\[\text{(Attr}_2)\]

- Acessing $C_{0,1}$- or $C_*$-typed attributes: similar.
Context in Operator Application

- Operator Application:

\[
A, B \vdash expr_1 : \tau_1 \ldots A, B \vdash expr_n : \tau_n \\
\vdash A, B \vdash \omega(expr_1, \ldots, expr_n) : \tau \quad \omega : \tau_1 \times \ldots \times \tau_n \rightarrow \tau, \\
n \geq 1, \omega \notin \operatorname{atr}(C)
\]

- Iterate:

\[
A, B \vdash expr_1 : \operatorname{Set}(\tau_1) \\
A', B \vdash expr_2 : \tau_2 \\
A', B \vdash expr_3 : \tau_2 \\
\vdash A, B \vdash expr_1 \rightarrow \operatorname{iterate}(\omega_1 : \tau_1; w_2 : \tau_2 = expr_2 | expr_3) : \tau_2
\]

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

\[
\begin{align*}
\text{(Context rule)} & \quad \frac{B, A \vdash expr : \tau}{A \Downarrow B \vdash expr : \tau} \\
\text{(Attr rule)} & \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash v(expr_1) : \tau}, \quad \langle v : \tau, \xi, expr_0, P_\circ \rangle \in atr(C), \quad \xi = +, \text{ or } \xi = - \text{ and } C = B
\end{align*}
\]

Example:

\[
\begin{align*}
A & \vdash \text{self} : \tau_0 \\
A & \vdash \text{self.n.r.} v > 0
\end{align*}
\]
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 picture 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. [?], 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.
Recapitulation
Recapitulation

Class Diagrams $CD$

\[
\text{induces}
\]

extended (!) signature $\mathcal{S}(CD)$

\[
gives \text{rise to}
\]

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
UML Class Diagram Syntax

Klassendiagramm

- Klasse
- Abstrakte Klasse
- <Stereotyp1, Stereotyp2>
  Paket::Klasse
  attribut
  operation()()
  <Stereotyp1>
  attribut = wert

Syntax für Attribute:
Sichtbarkeit: Attributname : Paket::Typ [Multiplizität Ordnung] = Initialwert [Eigenschaftswerte]
Eigenschaftswerte: (readOnly), [ordered], [composite]

Syntax für Operationen:
Sichtbarkeit: Parameterliste: Richtung Name : Typ = Standardwert [Eigenschaftswerte]
Richtung: in, out, inout

Objektdiagramm

- Objekt
- :Klasse
- Aktive Klasse
- Boundary
- Control
- Entity

- Parametrierte Klasse
  Parameterliste: Richtung Name : Typ = Standardwert [Eigenschaftswerte]

- Aktives Objekt
UML Class Diagram Syntax

- **Vererbung**: Klasse1 -> Klasse2
- **Assoziation**: Klasse1 -> Klasse2
- **gerichtete Assoziation**: Klasse1 -> Klasse2
- **qualifizierte Assoziation**: Klasse1 -> Klasse2
- **Realisierung**: Klasse1 -> Klasse2
- **Abhängigkeit**: Klasse1 -> Klasse2
- **Abhängige Klasse**: Klasse1
- **Unabhängige Klasse**: Klasse2
- **Schnittstelle**: Anbieter -> Nutzer
- **“Stecker”**
- **bereitgestellte Schnittstelle**: genutzte Schnittstelle
- **“Buchse”**
- **operation1()**
- **operation2()**
- **Stereotyp**
- **Beziehungname**
- **Multiplizität**
- **Existenzabhängiges Teil**
- **Komposition**
- **Aggregation**
- **Multiplizität**
- **Leserichtung**
- **Sichtbarkeit rolle**
- **multyplicity**
- **role name**
- **property**
- **stereotype**
- **association**
- **realization**
- **generalization**
- **implementation**
- **aggregation**
- **composition**
- **multiplicity**
- **visibility**
UML Class Diagram Syntax [?, 61;43]

**Figure 7.19** - Graphic notation indicating exactly one association end owned by the association

**Figure 7.20** - Combining line path graphics

**Figure 7.23** - Examples of navigable ends
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 check)
- • a visibility,
- • a navigability,
- • an ownership,
- • and possibly a diamond. (exercises)

Wanted: places in the signature to represent the information from the picture.
Only for the course of Lectures 07/08 we assume that each attribute in \( V \)

- either is \( \langle v : \tau, \xi, \text{expr}_0, P_v \rangle \) with \( \tau \in \mathcal{T} \) (as before),
- or is an association of the form

\[
\langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \\
\vdots \\
\langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle
\]

where

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

\[
\mu ::= * \mid N \mid N..M \mid N..* \mid \mu_1 \mu_2 
\]

- \( P_i \) is a set of properties (as before),
- \( \xi \in \{+, -, \#, \sim\} \) (as before),
- \( \nu_i \in \{\times, -, >\} \) is the navigability,
- \( o_i \in \mathcal{B} \) is the ownership.
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 $\tau \in \mathcal{T}$ (as before),
- or is an association of the form

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots \rangle$$

**Alternative syntax** for multiplicities:

$$\mu ::= N..M | N..* | \mu, \mu$$

and define $*$ and $N$ as abbreviations. E.g. $N..$ $1..*$

**Note:** $N$ could abbreviate $0..N$, $1..N$, or $N..N$. We use last one.

- $r, role_i$ are just names, $C_i \in \psi$, $1 \leq i \leq n$,
- the multiplicity $\mu_i$ is an expression of the form

$$\mu ::= * | N | N..M | N..* | \mu, \mu$$

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

Also only for the course of this 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

$$(\mathcal{I}, \mathcal{C}, V, \text{atr})$$

a **signature (extended for associations)** if

$$\text{atr} : \mathcal{C} \to 2\{v \in V | v : \tau, \tau \in \mathcal{I}\}.$$
From Association Lines to Extended Signatures

- Stereotypes for associations:
  - easy to read
  - reading direction not represented

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle \rangle$$

$$\langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

$$_i \mu_1 = \begin{cases} 1 & , \text{if } C_i \\ 0 & , \text{if } C_i \end{cases}$$

$$_i \nu_1 = \begin{cases} \times & , \text{if } C_i \\ - & , \text{if } C_i \\ > & , \text{if } C_i \end{cases}$$
Association Example

Signature:

\[ \mathcal{I} = \left( \{ \text{Int} \}, \{ c, D \}, \{ \text{x: Int} \}, \langle c: c, 0..*, \{ \text{unique} \}, -, x, + \rangle, \langle n: D, 0..*, \{ \text{unique} \}, +, >, 0 \rangle \right), \]

\{ \text{CH} \} \{ \text{x} \} \leftarrow \text{only basic type attributes here!} \]
What If Things Are Missing?

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

- **Name**: Use
  \[ A_\langle C_1 \rangle \cdots \langle C_n \rangle \]
  if the name is missing.

  **Example**:

  ![Diagram](https://example.com/diagram.png)

- **Reading Direction**: no default.

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

  **Example**:

  ![Diagram](https://example.com/diagram.png)

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

  ![Diagram](https://example.com/diagram.png)
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