Contents & Goals

Last Lecture:
» Started to discuss “associations”, the general case.

This Lecture:
» Educational Objectives: Capabilities for following tasks/questions.
» Cont’d: Please explain this class diagram with associations.
» When is a class diagram a good class diagram?
» What are purposes of modeling guidelines? (Example?)
» Discuss the style of this class diagram.

Content:
» Treat “the rest”.
» Where do we put OCL constraints?
» Modelling guidelines, in particular for class diagram (following [Ambler, 2005])

Associations: The Rest

Recapitulation: Consider the following association:

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

- Association name \( r \) and rolenames/types \( \text{role}_i / C_i \) induce extended system states \( \lambda \).
- Multiplicity \( \mu \) is considered in OCL syntax.
- Visibility \( \xi \) / Navigability \( \nu \): well-typedness.

Now the rest:
- Multiplicity \( \mu \): we propose to view them as constraints.
- Properties \( P_i \): even more typing.
- Ownership \( o \): getting closer to pointers/references.
- Diamonds: exercise.

Visibility

Not so surprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by typing rules.

Question given:

\[
\begin{array}{c}
\text{context } C \text{ inv: self. } \text{role}.x > 0 \\
\end{array}
\]

is the following OCL expression well-typed or not (wrt. visibility):

\[
\begin{array}{c}
\text{context } C \text{ inv: self. } \text{role}.x > 0 \\
\end{array}
\]

Basically same rule as before: (analogously for other multiplicities)

\[
\begin{array}{c}
\text{context } C \text{ inv: self. } \text{role}.x > 0 \\
\end{array}
\]
Navigability is similar to visibility: expressions over non-navigable association ends \(\nu = \times\) are basically type-correct, but forbidden.

Question: given \(C_x: \text{Int}\)

D \text{inv} \[ self.role.x > 0 \]

The standards say:

- \(-\): navigation is possible
- \(>\): navigation is efficient
- \(\times\): navigation is not possible

So: In general, UML associations are different from pointers/references!

But: Pointers/references can faithfully be modelled by UML associations.

The Rest of the Rest

Recapitulation: Consider the following association:

\[ r : (\langle r_{\text{role}}: C_{\text{role}}, \mu_{\text{role}}, P_{\text{role}}, \xi_{\text{role}}, \nu_{\text{role}}, o_{\text{role}} \rangle, \ldots, \langle r_{\text{role}}: C_{\text{role}}, \mu_{\text{role}}, P_{\text{role}}, \xi_{\text{role}}, \nu_{\text{role}}, o_{\text{role}} \rangle) \]

- Association name \(r\) and rolenames/types \(r_{\text{role}}/C_{\text{role}}\)
- Multiplicity \(\mu\) is considered in OCL syntax
- Visibility / Navigability \(\nu\) / well-typedness

Now the rest:

- Multiplicity \(\mu\): we propose to view them as constraints.
- Properties \(P_{\text{role}}\): ever more typing.
- Ownership \(o_{\text{role}}\): getting closer to pointers/references.
- Diamonds: exercise.

Multiplicities as Constraints

Recall: The multiplicity of an association end is a term of the form:

\[ \mu = N | N..M | N.. \ast | \mu, \mu \ (N, M \in \mathbb{N}) \]

Proposal: View multiplicities (except \(0..1\)) as additional invariants/constraints.

\[ C \rightarrow \text{D} \]

\[ C' \rightarrow \text{D} \]

\[ \text{context} \ D \text{inv} : \text{self.role.x > 0} \]

Recall: The multiplicity of an association end is a term of the form:

\[ \mu = N | N..M | N.. \ast | \mu, \mu \ (N, M \in \mathbb{N}) \]

Proposal: View multiplicities (except \(0..1\)) as additional invariants/constraints.

\[ C \rightarrow \text{D} \]

\[ C' \rightarrow \text{D} \]

\[ \text{context} \ D \text{inv} : \text{self.role.x > 0} \]
Multiplicities as Constraints

\[ x^{Bulk}_{Pr} = \text{context } C \text{ on } (x_1 \leq \text{min} \leftrightarrow \text{size } x_1 \text{ or } \ldots \text{ or } (x_{N_k} \leq \text{min} \leftrightarrow \text{size } x_{N_k})) \]

Recall/Later:

\[ \text{from now on: } \text{bulk}(CP) = \{ \text{constraints occurring in now} \} \cup \{ x^{Bulk}_{Pr} \} \]

\[ (r \ldots, \text{role } D) \cup \ldots \cup \{ \text{role } \cdot \cdot \cdot, \text{role } D \} \subseteq \{ v \} \text{ or } \]

\[ (r \ldots, \text{role } D) \cup \ldots \cup \{ \text{role } \cdot \cdot \cdot, \text{role } D \} \subseteq \{ v \} \text{ role } \neq \emptyset \text{ and } \mu \neq 0.1 \}

And defines

\[ x^{Bulk}_{Pr} = \\text{context } C \text{ on } \{ \text{null} \text{ defined}(\text{role }) \} \]

for each \( p \neq \emptyset \)

Note: in set associations with \( n \geq 2 \), there is redundancy.

Multiplicities as Constraints Example

\[ x^{Bulk}_{Pr} = \text{context } C \text{ on } (x_1 \leq \text{min} \leftrightarrow \text{size } x_1 \text{ or } \ldots \text{ or } (x_{N_k} \leq \text{min} \leftrightarrow \text{size } x_{N_k})) \]

CD:

From now on: \( \text{bulk}(CP) = \{ \text{constraints occurring in now} \} \cup \{ x^{Bulk}_{Pr} \} \)

\[ (r \ldots, \text{role } D) \cup \ldots \cup \{ \text{role } \cdot \cdot \cdot, \text{role } D \} \subseteq \{ v \} \text{ or } \]

\[ (r \ldots, \text{role } D) \cup \ldots \cup \{ \text{role } \cdot \cdot \cdot, \text{role } D \} \subseteq \{ v \} \text{ role } \neq \emptyset \text{ and } \mu \neq 0.1 \}

Properties

We don’t want to cover association properties in detail, only some examples (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{single}</td>
<td>One object has at most one role in</td>
<td>Context setting</td>
</tr>
<tr>
<td>\text{many}</td>
<td>One object has at most one role in</td>
<td>Context setting</td>
</tr>
<tr>
<td>\text{size}</td>
<td>One object may have multiple roles on</td>
<td>Size ( k ) yield</td>
</tr>
<tr>
<td>\text{order}</td>
<td>One role may be a many-to-one role on</td>
<td>Multi-rels</td>
</tr>
<tr>
<td>\text{sequence}</td>
<td>One role may be a many-to-one role on</td>
<td>Multi-rels</td>
</tr>
</tbody>
</table>

Why Multiplicities as Constraints?

More precisely, can we just use types? (cf. Slide 28)

\( p, q \neq 0, 1, p \neq q \)

Many programming languages have direct correspondences (the first corresponds to type points, the second to type values) — therefore treated specially.

\( \mu = 1 \)

Could be represented by a set data structure type without bounds — no problem with our approach: we have \( p, q = 0 \text{ or } 1 \text{ any way} \)

\( p = 0, q = 1 \)

Use array of size \( k \) — if model behavior (or the implementation) adds 5th identity, but it gets a context role, and thereby we see that the constraint is violated. Principally acceptable, but checks for array bounds everywhere...

\( p = 1, q = 1 \)

Could be represented by an array of size \( 7 \) — but: the programming language data structure libraries allow bounds for arrays (other than \( 0 \)).

If we \( / \text{identity} \) and the model behavior remains same, this should be a multiplicity of \( 1 \). The implementation which does this is correct. How do we see this...

Multiplicities Never as Types...

Well, if the target platform is known and fixed, and the target platform has, for instance:

- reference types,
- range checked arrays with positions \( 0, \ldots, N \),
- set types,

then we could simply restrict the syntax of multiplicities to

\( k \geq 1 \) \{ \( 0, N \) \}

And don’t think about constraints

But use the obvious 1-to-1 mapping to types...

In general, unfortunately, we don’t know.

Languages/datastructure libraries allow lower bounds for arrays (other than 0) — but: few programming...
Properties

We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>an object has at most one ( r )-link to a single other object</td>
<td>( n(\sigma(u)) = 1 ) for all ( u )</td>
</tr>
<tr>
<td>bag</td>
<td>an object may have multiple ( r )-links to a single other object</td>
<td>( n(\sigma(u)) &gt; 1 ) for all ( u )</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an ( r )-link is a sequence of object identifiers (possibly including duplicates)</td>
<td>( \lambda(\sigma(u)) ) yield sequence of identifiers</td>
</tr>
</tbody>
</table>

Not clear to me:

For subsets, redefinition, union, etc. see [OMG, 2007a, 127].

Ownership

Intuitively it says:

Association \( r \) is not a “thing on its own” (i.e. provided by \( \lambda \)), but association end ‘role’ is owned by \( C \)!

(That is, it’s stored inside \( C \) object and provided by \( \sigma \)).

So, if multiplicity of role is 0..1 or 1, then the picture above is very close to concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

Back to the Main Track

Recall: on some earlier slides we said, the extension of the signature is only to study associations in “full beauty.”

For the remainder of the course, we should look for something simpler...

Proposal:

1. From now on, we only use associations of the form

   (i) \( C \text{ • } 0..1 \times \text{•} \)

   (ii) \( C \text{ • } \ast \times \text{•} \)

   (And we may omit the non-navigability and ownership symbols.)

   - Form (i) introduces role \( C_{0..1} \) and form (ii) introduces role \( C_{\ast} \) in \( Y \).
   - In both cases, role \( C \) is \( \sigma(u)(C) \).
   - We drop \( r \) and go back to our role \( r \) with \( n(\sigma(u)(r)) \subseteq \sigma(u)(D) \).

OCL Constraints in (Class) Diagrams

Where Shall We Put OCL Constraints?

Numerous options:

1. Additional documents.
2. Notes.
3. Particular dedicated places.

Notes:

A UML note is a picture of the form

![Example note](image)

that can principably be everything, in particular comments and constraints.

Sometimes, content is explicitly classified for clarity:

![OCL expression](image)
Where Shall We Put OCL Constraints?

(ii) Particular dedicated places in class diagrams: (below: notation: brace)

\[
\xi_v^\tau \{ p_1, \ldots, p_n \} \{ expr \}
\]

\[
\xi_f(v_1: \tau, \ldots, v_n: \tau_n): \tau \{ p_1, \ldots, p_n \} \{ pre: expr_1, post: expr_2 \}
\]

For simplicity, we view the above as an abbreviation for

\[
\xi_v^\tau \{ p_1, \ldots, p_n \} \{ pre: expr_1, post: expr_2 \}
\]

Pragmatics

Recall: a UML model is an image or pre-image of a software system.

A set of class diagrams \( \mathcal{CD} \) with invariants \( \text{Inv}(\mathcal{CD}) \) describes the structure of system states.

Together with the invariants it can be used to state:

- Pre-image: Dear programmer, please provide an implementation which uses only system states that satisfy \( \text{Inv}(\mathcal{CD}) \).
- Post-image: Dear user/maintainer, in the existing system, only system states which satisfy \( \text{Inv}(\mathcal{CD}) \) are used.

[The exact meaning of "use" will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system states by calling methods or firing transitions in state machines.]

Example: highly abstract model of traffic lights controller.

\[
\text{TrafficLight} \sim \begin{cases}
\text{Red} & \text{if } T = \text{Red} \\
\text{Green} & \text{if } T = \text{Green} \\
\text{Not (Red and Green)} & \text{otherwise}
\end{cases}
\]
Constraints vs. Types

Find the 10 differences:

\[
\begin{array}{l}
\text{C}_x: \quad \text{Int} \\
\quad x = 3 \lor x > 17
\end{array}
\quad \text{vs.}
\quad \begin{array}{l}
\text{C}_x: \quad \text{T} / \text{BW}(T) = \{ 3 \} \\
\quad \cup \{ n \in \mathbb{N} | n > 17 \}
\end{array}
\]

- \( x = 4 \) is well-typed in the left context,
- a system state satisfying \( x = 4 \) violates the constraints of the diagram.
- \( x = 4 \) is not even well-typed in the right context,
- there cannot be a system state with \( \sigma(u)(x) = 4 \) because \( \sigma(u)(x) \) is supposed to be in \( \text{T} / \text{BW}(T) \) (by definition of system state).

Rule of thumb:
- If something "feels like" a type (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a requirement or restriction of an otherwise useful type, then make it a constraint.

References

