Contents & Goals

Last Lectures:
- Studied syntax of associations in 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 modelling guidelines? (Example?)
  - Discuss the style of this class diagram.

Content:
- Association semantics and effect on OCL.
- Treat “the rest”.
- Where do we put OCL constraints?
- Modelling guidelines, in particular for class diagrams (following [Ambler, 2005])
- Examples: modelling games (made-up and real-world examples)
Overview

What’s left? Named association with at least two typed ends, each having

- a role name,
- a multiplicity,
- a set of properties,
- a visibility,
- a navigability, and
- an ownership.

The Plan:

- Extend system states, introduce so-called links as instances of associations — depends on name and on type and number of ends.
- Integrate role name and multiplicity into OCL syntax/semantics.
- Extend typing rules to care for visibility and navigability.
- Consider multiplicity also as part of the constraints set Inv(CD).
- Properties: for now assume $P_v = \{\text{unique}\}$.
- Properties (in general) and ownership: later.
Recall: We consider associations of the following form:

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

Only these parts are relevant for extended system states:

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

(recall: we assume \( P_1 = P_n = \{\text{unique}\} \)).

The UML standard thinks of associations as \textit{n-ary relations}
which “\textit{live on their own}” in a system state.

That is, \textbf{links} (= association instances)
- do not belong (in general) to certain objects (in contrast to pointers, e.g.)
- are “first-class citizens” \textit{next to objects},
- are (in general) \textbf{not} directed (in contrast to pointers).
Links in System States

\[ \langle r : \langle \text{role}_1 : C_1, P_1, \ldots \rangle, \ldots, \langle \text{role}_n : C_n, P_n, \ldots \rangle \rangle \]

**Only** for the course of lectures 07/08 we change the definition of system states:

**Definition.** Let \( \mathcal{D} \) be a structure of the (extended) signature \( \mathcal{S} = (\mathcal{P}, \mathcal{E}, V, \text{atr}) \).

A system state of \( \mathcal{S} \) wrt. \( \mathcal{D} \) is a pair \((\sigma, \lambda)\) consisting of:

- a type-consistent mapping
  \[ \sigma : \mathcal{D}(\mathcal{E}) \rightarrow (\text{atr}(\mathcal{E}) \rightarrow \mathcal{D}(\mathcal{P})) \],

- a mapping \( \lambda \) which assigns each association
  \[ \langle r : \langle \text{role}_1 : C_1, \ldots \rangle, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V \]
  a relation
  \[ \lambda(r) \subseteq \mathcal{D}(C_1) \times \cdots \times \mathcal{D}(C_n) \]
  (i.e. a set of type-consistent \( n \)-tuples of identities).

**Example:** order of assoc. ends in \( \mathcal{S} \) does matter

\[
\begin{align*}
\sigma &= \{ (1, 10, 3), (2, 10, 3), (3, 10, 3), (5, 20, 3), (20, 10, 3) \} \\
\lambda &= \{ (1, 10, 3_1), (2, 5, 3_2), (3, 5, 3_3) \} \\
\text{(students may join multiple groups)} \\
\text{students may also have multiple references} \\
\end{align*}
\]

**OBJECT DIAGRAMS:**

- We will not formally define that
**Association/Link Example**

A system state of $\mathcal{S}$ (some reasonable $\mathcal{S}$) is $(\sigma, \lambda)$ with:

$\sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\}$

$\lambda = \{A \leftrightarrow C \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}$

**Extended System States and Object Diagrams**

**Legitimate question:** how do we represent system states such as

$\sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\}$

$\lambda = \{A \leftrightarrow C \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}$

as object diagram?

*See 7a and 8.*
**Associations and OCL**

---

**OCL and Associations: Syntax**

**Recall:** OCL syntax as introduced in Lecture 03, interesting part:

\[
\text{expr} ::= \ldots \mid r_1(\text{expr}_1) : \tau_C \rightarrow \tau_D \quad r_1 : D_{0,1} \in \text{attr}(C) \\
\mid r_2(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad r_2 : D^* \in \text{attr}(C)
\]

Now becomes

\[
\text{expr} ::= \ldots \mid \text{role}(\text{expr}_1) : \tau_C \rightarrow \tau_D \quad \mu = 0..1 \text{ or } \mu = 1 \\
\mid \text{role}(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad \text{otherwise}
\]

\[
\text{if } (r : \ldots, (\text{role} : D, \mu, \ldots), \ldots, (\text{role}' : C, \ldots) \ldots) \in V \text{ or } \\
(r : \ldots, (\text{role}' : C, \ldots) \ldots, (\text{role} : D, \mu, \ldots) \ldots) \in V, \text{ role } \neq \text{ role}'.
\]

**Note:**
- Association name as such doesn’t occur in OCL syntax, role names do.
- \text{expr}_1 has to denote an object of a class which "participates" in the association.
**OCL and Associations Syntax: Example**

```
expr ::= ... | role(expr_i) : τC -> τD       \( \mu = 0..1 \) or \( \mu = 1 \)
| role(expr_i) : τC -> Set(τD)  otherwise
```

if

\[
\langle r : \ldots, \text{role} : D, \mu, \ldots \rangle, \ldots, \langle \text{role} : C, \ldots \rangle, \ldots \rangle \in V \quad \text{or}
\langle r : \ldots, \langle \text{role} : C, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots \rangle, \ldots \rangle \in V, \text{role \neq role'}.
\]

\*

Figure 7.21 - Binary and ternary associations [OMG, 2007b, 44].

---

### OCL and Associations: Semantics

**Recall:** (Lecture 03)

Assume \( expr_1 : \tau_C \) for some \( C \in \mathcal{C} \). Set \( u_1 := I[expr_1](\sigma, \beta) \in {}^\tau\tau_C \).

- \( I[r_1(expr_1)](\sigma, \beta) := \begin{cases} u, & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\
\bot, & \text{otherwise} \end{cases} \)

- \( I[r_2(expr_1)](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2), & \text{if } u_1 \in \text{dom}(\sigma) \\
\bot, & \text{otherwise} \end{cases} \)

**Now needed:**

\( I[\text{role}(expr_1)]((\sigma, \lambda), \beta) \)

- We cannot simply write \( \sigma(u)(\text{role}) \).
  **Recall:** \textit{role} is \textit{for the moment} not an attribute of object \( u \) (not in \( atr(C) \)).

- What we have is \( \lambda(r) \) (with \( r \), not with \( \text{role!} \)) — but it yields a set of \( n \)-tuples, of which \textit{some} relate \( u \) and other some instances of \( D \).

- \textit{role} denotes the position of the \( D \)'s in the tuples constituting the value of \( r \).
OCL and Associations: Semantics Cont’d

Assume $expr_1 : \tau_C$ for some $C \in \mathcal{C}$. Set $u_1 := I[expr_1]((\sigma, \lambda), \beta) \in \mathcal{P}(\tau_C)$.

- $I[role(expr_1)]((\sigma, \lambda), \beta) := \begin{cases} u, & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } L(role)(u_1, \lambda) = \{u\} \\ \bot, & \text{otherwise} \end{cases}$

- $I[role(expr_1)]((\sigma, \lambda), \beta) := \begin{cases} L(role)(u_1, \lambda), & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases}$

where

$L(role)(u, \lambda) = \{(u_1, \ldots, u_n) \in \lambda(r) | u \in \{u_1, \ldots, u_n\}\} \downarrow i$

if

$r : \ldots (role_1 : \ldots, \ldots, \ldots, \ldots, \ldots) \ldots (role_n : \ldots, \ldots, \ldots, \ldots, \ldots) \ldots, role = role_i$

Given a set of $n$-tuples $A$, $A \downarrow i$ denotes the element-wise projection onto the $i$-th component.

OCL and Associations Example

$I[role(expr_1)]((\sigma, \lambda), \beta) := \begin{cases} L(role)(u_1, \lambda), & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases}$

$L(role)(u, \lambda) = \{(u_1, \ldots, u_n) \in \lambda(r) | u \in \{u_1, \ldots, u_n\}\} \downarrow i$

$\sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\}$

$\lambda = \{A, C, D \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}$

$I[\text{self}.n][(\sigma, \lambda), \{\text{self} \mapsto 1_C\}] = I[I[n(\text{self})]][(\sigma, \lambda), \beta] \downarrow \{1_C\} = L(n)(\beta, \lambda) \star L(\text{self})(1_C, \lambda) \star \{(q, 3_d), (q, 7_d)\} \downarrow \{q, 3_d, 7_d\}$
Visibility

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

Question: given

\[
\begin{array}{c}
\text{context } C \text{ inv : self.role.x > 0 } \quad \text{\textbf{not if } } \xi = \text{private}
\end{array}
\]

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

\[
\begin{array}{c}
\text{context } C \text{ inv : self.role.x > 0 } \quad \text{\textbf{not if } } \xi = \text{private}
\end{array}
\]

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

\[
(Assoc_1) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash \text{role}(expr_1) : \tau_D}, \quad \mu = 0..1 \text{ or } \mu = 1, \quad \xi = +, \text{ or } \xi = - \text{ and } C = B
\]

\[
\langle r : \ldots \langle \text{role : } D, \mu, \xi, \_ \_ \_ \_ \rangle, \ldots \langle \text{role}' : C, \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \rangle, \ldots \rangle \in V
\]
Navigability

Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are basically type-correct, but forbidden.

**Question:** given

\[
\begin{array}{c}
  C \\
  x : \text{Int} \\
  \text{role} \\
  \end{array} \rightarrow \begin{array}{c}
  D \\
  \end{array}
\]

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

```ocl
context D inv : self.role.x > 0
```

The standard says:

- '−': navigation is possible
- '×': navigation is not possible
- '>): navigation is efficient

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

**But:** Pointers/references can faithfully be modelled by UML associations.
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 role names/types \( \text{role}_i, C_i \) induce extended system states \( \lambda \).
- Multiplicity \( \mu \) is considered in OCL syntax.
- Visibility \( \xi \) and navigability \( \nu \) give rise to well-typedness rules.

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.

### Multic和平ties as Constraints

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

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

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

**Recall:** we can normalize each multiplicity to the form

\[N_1..N_2, \ldots, N_{2k-1}..N_{2k}\]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k} \in \mathbb{N}, \quad N_{2k} \in \mathbb{N} \cup \{\ast\} \).

**Define**

\[
\mu_{\text{OCL}} = \text{context } C \text{ inv :}
\]

\[
(N_1 \leq \text{role} \rightarrow \text{size}()) \leq N_2) \quad \text{and} \quad \ldots \quad \text{and} \quad (N_{2k-1} \leq \text{role} \rightarrow \text{size}()) \leq N_{2k})
\]

for each

\[
\langle r : \ldots, \langle \text{role} : D, \mu, \ldots, \rangle, \ldots, \langle \text{role}' : C, \ldots, \rangle, \ldots \rangle \in V \text{ or}
\]

\[
\langle r : \ldots, \langle \text{role}' : C, \ldots, \rangle, \ldots, \langle \text{role} : D, \mu, \ldots, \rangle, \ldots \rangle \in V, \text{role} \neq \text{role}'.
\]

**Note:** in \( n \)-ary associations with \( n > 2 \), there is redundancy.
**Multiplicities as Constraints of Class Diagram**

**Recall:**

\[ \mathcal{D} = \{ CD_1, \ldots, CD_n \} \]

- \( \mathcal{I}(\mathcal{D}) \)
- \( \text{signature} \)
- \( \text{basic} \)
- \( \text{extended} \)
- \( \text{(classes and attributes)} \)
- \( \text{(visibility)} \)
- \( \text{invariants} \)

**From now on:** \( \text{Inv}(\mathcal{D}) = \{ \text{constraints occurring in notes} \} \cup \{ \mu_{\text{OCL}} | \)

\[ \langle r : \ldots, (\text{role} : D, \mu, \ldots), \ldots, (\text{role} : C, \ldots) \rangle \in V \text{ or } \]
\[ \langle r : \ldots, (\text{role} : C, \ldots), \ldots, (\text{role} : D, \mu, \ldots) \rangle \in V, \]
\[ \text{role} \neq \text{role}', \mu \notin \{0, 1\} \}. \]

---

**Multiplicities as Constraints Example**

\( \mu_{\text{OCL}} = \text{context } C \text{ inv} : \)

\( (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ and } \ldots \text{ and } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \)

\( \mathcal{CD} : \)

\( \begin{array}{c}
\text{role}_1 \\
0.1 \\
\text{v : Int}
\end{array} \quad 4, 17 \\
\quad \begin{array}{c}
\text{role}_2 \\
3..* \\
\text{role}_3
\end{array} \)

\( \text{Inv}(\mathcal{CD}) = \)

- \( \{ \text{context } C \text{ inv : } 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17 \} \)
- \( \{ \text{context } C \text{ inv : } \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17 \} \)
- \( \cup \{ \text{context } C \text{ inv : } 3 \leq \text{role}_3 \rightarrow \text{size()} \} \)
Why Multiplicities as Constraints?

More precise, can’t we just use types? (cf. Slide 36)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — this is why we excluded them.

- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\text{OCL}} = \text{true}$ anyway.

- $\mu = 0..3$:
  use array of size 4 — if model behaviour (or the implementation) adds 5th identity, we’ll get a runtime error, and thereby see that the constraint is violated. Principally acceptable, but: checks for array bounds everywhere...?

- $\mu = 5..7$:
  could be represented by an array of size 7 — but: few programming languages/data structure libraries allow lower bounds for arrays (other than 0). If we have 5 identities and the model behaviour removes one, this should be a violation of the constraints imposed by the model.
  The implementation which does this removal is wrong. 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

$$\mu ::= 1 \mid 0..N \mid *$$

and don’t think about constraints
(but use the obvious 1-to-1 mapping to types)... 

In general, unfortunately, we don’t know.
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>one object has <strong>at most one</strong> ( r )-link to a</td>
<td>current setting</td>
</tr>
<tr>
<td></td>
<td>single other object</td>
<td></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> ( r )-links to a</td>
<td>have ( \lambda(r) ) yield</td>
</tr>
<tr>
<td></td>
<td>single other object</td>
<td>multi-sets</td>
</tr>
<tr>
<td>ordered,</td>
<td>an ( r )-link is a <strong>sequence</strong> of object identities</td>
<td>have ( \lambda(r) ) yield</td>
</tr>
<tr>
<td>sequence</td>
<td>(possibly including duplicates)</td>
<td>sequences</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression ( role(expr) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>( \tau_D \rightarrow \text{Set}(\tau_C) )</td>
</tr>
<tr>
<td>bag</td>
<td>( \tau_D \rightarrow \text{Bag}(\tau_C) )</td>
</tr>
<tr>
<td>ordered,</td>
<td>( \tau_D \rightarrow \text{Seq}(\tau_C) )</td>
</tr>
<tr>
<td>sequence</td>
<td></td>
</tr>
</tbody>
</table>

For **subsets**, **redefines**, **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).

Not clear to me:
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:
- from now on, we only use associations of the form

(i) \( C \) \( \overset{0..1}{\text{role}} \rightarrow \) \( D \)

(ii) \( C \) \( \overset{*}{\text{role}} \rightarrow \) \( D \)

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

- Form (i) introduces role : \( C_{0,1} \), and form (ii) introduces role : \( C_* \) in \( V \).
- In both cases, role \( \in \text{atr}(C) \).
- We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(\text{role}) \subseteq \mathcal{P}(D) \).
References

