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)
Association Semantics
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 $\text{Inv}(CD)$.

- Properties: for now assume $P_v = \{\text{unique}\}$.

- Properties (in general) and ownership: later.
Association Semantics: The System State Aspect
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:

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

(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)

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

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

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

**Definition.** Let \( \mathcal{D} \) be a structure of the (extended) signature \( \mathcal{I} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \).

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

- a type-consistent mapping
  \[
  \sigma : \mathcal{D}(\mathcal{C}) \leftrightarrow (\text{atr}(\mathcal{C}) \leftrightarrow \mathcal{D}(\mathcal{T})),
  \]

- a mapping \( \lambda \) which assigns each association
  \[
  \langle r : \langle \text{role}_1 : C_1 \rangle, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V
  \]
  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 S does matter

\[ \sigma = \{1_s \rightarrow s_1 \omega 1_f, 2_s \rightarrow s_2 \omega 0_8, 3_s \rightarrow s_3 \omega 0_9, 2t_5 \rightarrow s_5 \omega 0_{25}7\}\]

\[ \lambda = \{t \rightarrow \{1_s, 2_s, 3_s\}, \{1_s, 2_s, 3_s\}, \{2_s, 5_s, 6_s\}\} \]  

students may join multiple groups

\[ \text{ Dist}(S) = \{2_s, 5_s, 6_s\} \]  

general links may also have changing references

\[ \text{ Dist}(S) = \{3_s, 3_s, 3_s\} \]  

one student may assume all roles (if this is not desired, then add a constraint context S inv: \(1_s \neq \text{sec} \) and \(5_s \neq \text{sec} \))

OBJECT DIAGRAM:

![Object Diagram]

So we would need hyperedges

WE WILL NOT FORMALLY DEFINE THAT
**Association/Link Example**

**Signature:**

\[ \mathcal{S} = \{\text{Int}, \{C, D\}, \{x : \text{Int}, \langle A \_ C \_ D : \langle c : C, 0..*, +, \{\text{unique}\}, \times, 1\rangle, \langle n : D, 0..*, +, \{\text{unique}\}, >, 0\rangle\rangle, \{C \mapsto \emptyset, D \mapsto \{x\}\}\} \]

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

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

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

This case can be represented by an object diagram.
**Legitimate question:** how do we represent system states such as

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

\[
\lambda = \{A \_C \_D \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}
\]

as **object diagram**?

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

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

```
expr ::= \ldots \mid r_1(expr_1) : \tau_C \to \tau_D \quad r_1 : D_{0,1} \in atr(C)
      \mid r_2(expr_1) : \tau_C \to \text{Set}(\tau_D) \quad r_2 : D_* \in atr(C)
```

Now becomes

```
expr ::= \ldots \mid \text{role}(expr_1) : \tau_C \to \tau_D \quad \mu = 0..1 \text{ or } \mu = 1
      \mid \text{role}(expr_1) : \tau_C \to \text{Set}(\tau_D) \quad \text{otherwise}
```

if

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

Note:

- Association name as such doesn’t occur in OCL syntax, role names do.
- \( expr_1 \) has to denote an object of a class which “participates” in the association.
OCL and Associations Syntax: Example

\[ expr ::= \ldots \mid role(expr_1) : \tau_C \rightarrow \tau_D \quad \mu = 0..1 \text{ or } \mu = 1 \]
\[ \mid role(expr_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad \text{otherwise} \]

\[
\begin{align*}
\text{if} & \quad \langle r : \ldots, \langle \text{role} : D, \mu, \_ , \_ , \_ \rangle, \ldots, \langle \text{role} : C, \_ , \_ , \_ , \_ \rangle, \ldots \rangle \in V \text{ or } \langle r : \ldots, \langle \text{role} : C, \_ , \_ , \_ , \_ \rangle, \ldots, \langle \text{role} : D, \mu, \_ , \_ , \_ \rangle, \ldots \rangle \in V, \text{ role } \neq \text{ role}'.
\end{align*}
\]

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 $u_1 := I[expr_1](\sigma, \beta) \in \mathcal{D}(\tau_C)$.

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

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

Now needed:

$I[role(expr_1)]((\sigma, \lambda), \beta)$

- We cannot simply write $\sigma(u)(role)$.

  Recall: role is (for the moment) not an attribute of object $u$ (not in $\text{attr}(C)$).

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

- 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 $u_1 := I[expr_1]((\sigma, \lambda), \beta) \in \mathcal{D}(\tau_C)$.

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

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

where

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

if

$\langle r : \ldots \langle role_1 : \_ \_ \_ \_ \_ \_ \rangle, \ldots \langle role_n : \_ \_ \_ \_ \_ \_ \rangle, \ldots \rangle, role = role_i \rangle$.

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

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

\[ L(role)(u, \lambda) = \{(u_1, \ldots, u_n) \in \lambda(r) \mid u \in \{u_1, \ldots, u_n\}\} \downarrow i \]

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

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

\[ I[self \cdot n]((\sigma, \lambda), \{self \mapsto 1_C\}) = I[\n(x \mapsto \{self \mapsto 1_C\}(\sigma, \lambda), \beta)]_{\text{self} \mapsto 1_C} = \]

\[ = \bigcup(n)(\beta(x \mapsto \text{self}), \lambda) = \bigcup(n)(1_C, \lambda) = \{(1_C, 3_D), (1_C, 7_D)\} \downarrow_2 = \{(3_D, 7_D)\} \]
Associations: The Rest
Visibility

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

**Question:** given

\[
\begin{array}{c}
C \xrightarrow{1} \xi \text{role} \\
\end{array}
\]

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

context \(C\) inv : \(self.role.x > 0\)  **not if \(\xi = \text{private}\)**

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

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

\[
\langle r : \ldots \langle role : D, \mu, -, \xi, -, - \rangle, \ldots \langle role' : C, -, -, -, -, \rangle, \ldots \rangle \in V
\]
Navigability is similar to visibility: expressions over non-navigable association ends \((\nu = \times)\) are basically type-correct, but forbidden.

**Question:** given

![Diagram](image.png)

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

```ocl
class 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.
Visibility and Navigability:
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.
**Multiplicities as Constraints**

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

\[ \mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \] \hspace{1cm} (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 \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k} \in \mathbb{N} \), \( N_{2k} \in \mathbb{N} \cup \{*\} \).

**Define**

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

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

for each

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

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

**Note:** in \( n \)-ary associations with \( n > 2 \), there is redundancy.
Recall:

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

signature \( \mathcal{I}(\mathcal{CD}) \)

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

 basic
(clases and attributes)

distinguish

extended
(visibility)

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

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

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

\[ \text{role} \neq \text{role}', \mu \notin \{0, 1\} \} \].
\[ \mu_{\text{OCL}} = \text{context } C \text{ inv :} \]
\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ and } \ldots \text{ and } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \]

\[ CD : \]

\[ \text{Inv}(CD) = \]
\[ \bullet \{ \text{context } C \text{ inv : } 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17\} \]
\[ = \{ \text{context } C \text{ inv : } \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17\} \]
\[ \bullet \cup \{ \text{context } C \text{ inv : } 3 \leq \text{role}_3 \rightarrow \text{size()}\} \]
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 $\mu_{OCL} = \text{true}$ 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...?
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.
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 single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> $r$-links to a single other object</td>
<td>have $\lambda(r)$ yield multi-sets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an $r$-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have $\lambda(r)$ yield 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>$\tau_D \rightarrow Set(\tau_C)$</td>
</tr>
<tr>
<td>bag</td>
<td>$\tau_D \rightarrow Bag(\tau_C)$</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>$\tau_D \rightarrow Seq(\tau_C)$</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:**

![Diagram](image-url)
Back to the Main Track
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) $C \times\ {0..1} \rightarrow role \rightarrow D$

(ii) $C \times\ {\ast} \rightarrow role \rightarrow D$

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

- Form (i) introduces $role: C_{0,1}$, and form (ii) introduces $role: C_\ast$ in $V$.

- In both cases, $role \in atr(C')$.

- We drop $\lambda$ and go back to our nice $\sigma$ with $\sigma(u)(role) \subseteq \mathcal{P}(D)$. 
References
References

