Software Design, Modelling and Analysis in UML

Lecture 8: Class Diagrams III

2016-11-24

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
\( \mathcal{P} = (\mathcal{T}, \mathcal{C}, V, \text{attr}), SM \)

\( M = (\Sigma^\mathcal{P}, A^\mathcal{P}, \rightarrow_{SM}) \)

\( \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, S\text{nd}_0)} (\sigma_1, \varepsilon_1) \cdots \)

\( w_\pi = ((\sigma_i, \text{cons}_i, S\text{nd}_i))_{i \in \mathbb{N}} \)

\( \varphi \in \text{OCL} \)

\( \mathcal{Q} \)

\( \mathcal{G} = (N, E, f) \)
Content

- Recall: Associations
  - Overview & Plan
  - (Temporarily) Extend Signature

- From Class Diagrams to Signatures
  - What if Things are Missing?

- Association Semantics
  - Links in System States
  - Associations and OCL

- The Rest
  - Visibility, Navigability
  - Multiplicity, Properties,
  - Ownership, “Diamonds”

- Back to the Main Track
Recall: Plan & Extended Signature
Overview

- **Class diagram:**

  ![Class Diagram](image)

  **Alternative presentation:**

  ![Alternative Class Diagram](image)

- **Signature:**

  \( \mathcal{S} = (\{\text{Int}\}, \{C, D\}, \{v : \text{Int}, d : D_{\ast}, c : C_{0,1}\}, \{C \mapsto \{v, d\}, D \mapsto \{c\}\}) \)

- **Example system state:**

  \( \sigma = \{1_C \mapsto \{v \mapsto 27, d \mapsto \{5_D, 7_D\}\}, 5_D \mapsto \{c \mapsto \{1_C\}\}, 7_D \mapsto \{c \mapsto \{1_C\}\}\} \)

- **Object diagram:**

  ![Object Diagram](image)

  ![Object Diagram](image)

- **Class diagram** (with ternary association):

  ![Class Diagram with Ternary Association](image)

  **Signature:** extend again to represent

  **association** \( r \) with

  **association ends** \( a, b, \) and \( z \)

  (each with multiplicity, visibility, etc.)

- **Example system state:**

  \( (\sigma, \lambda) \)

  \( \sigma = \{1_A \mapsto \{w \mapsto 13\}, 1_B \mapsto \emptyset, 1_Z \mapsto \emptyset\} \)

  \( \lambda = \{r \mapsto \{(1_A, 1_B, 1_Z), (1_A, 1_B, 2_Z)\}\} \)

  ![Example System State](image)

- **Object diagram:** No...
So, 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, (not in UML)
- a visibility,
- a navigability,
- an ownership,
- and possibly a diamond.

Wanted: places in the signature to represent the information from the picture.
Definition. An (Extended) Object System Signature (with Associations) is a quadruple $\mathcal{S} = (\mathcal{T}, \mathcal{C}, V, atr)$ where

- ... 
- each element of $V$ is
  - either a **basic type attribute** $\langle v : T, \xi, expr_0, P_v \rangle$ with $T \in \mathcal{T}$
  - or an **association** of the form
    $$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$
    (ends with multiplicity $\mu_i$, properties $P_i$, visibility $\xi_i$, navigability $\nu_i$, ownership $o_i$, $1 \leq i \leq n$)
- ... 
- $atr : \mathcal{C} \rightarrow 2^{\{v \in V \mid v:T, T \in \mathcal{T}\}}$ maps classes to **basic type (!) attributes**.

In other words:
- only **basic type attributes** “belong” to a class (may appear in $atr(C)$),
- **associations** are not “owned” by a class (not in any $atr(C)$), but “live on their own”.

Rhapsody: assoc. names must be unique, so not $C_1 \rightarrow C_2$.
Associations in Class Diagrams
From Association Lines to Extended Signatures

\[ o_i = \begin{cases} 
1 & \text{if } C_i \\
0 & \text{if } C_i
\end{cases} \]

\[ \nu_i = \begin{cases} 
\times & \text{if } C_i \\
- & \text{if } C_i \\
> & \text{if } C_i
\end{cases} \]
Association Example

Signature:

\[ \mathcal{S} = (\{ \text{Int} \}, \{ C, D \}, \{ x: \text{Int} \rightarrow \{ x \} \}) \]
What If Things Are Missing?

Most components of associations or association end may be omitted. For instance (OMG, 2011b, 17), Section 6.4.2, proposes the following rules:

- **Name**: Use

  \[ \text{A}_n \langle C_1 \rangle \ldots \langle C_n \rangle \]

  if the name is missing.

  **Example**:

  ![Diagram](C_A_D)

  for

  ![Diagram](C_D)

- **Reading Direction**: no default.

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

  **Example**:

  ![Diagram](C_D_c_d)

  for

  ![Diagram](C_D)

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

![Diagram](C_D_itsC_itsD)

for

![Diagram](C_D)
What If Things Are Missing?

- **Multiplicity:** 1
  
  In my opinion, it's safer to assume 0..1 or ∗ (for 0..∗) if there are no fixed, written, agreed conventions ("expect the worst").

- **Properties:** ∅

  **Visibility:** public

- **Navigability and Ownership:** not so easy. (OMG, 2011b, 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 ∗’s:** Navigation and its absence are made completely explicit.
  - **Suppress all arrows and ∗’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 your convention and leave out the *’s.
Associations: Semantics
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 n-ary relations which "live on their own" in a system state.

That is, links (= association instances)

- do not belong (in general) to certain objects (in contrast to pointers, e.g.)
- are "first-class citizens" next to objects,
- are (in general) not directed (in contrast to pointers).
Only for the course of lectures 8 / 9 we change the definition of system states:

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

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

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

- a mapping $\lambda$ which maps each association
  $$\langle r : \langle \text{role}_1 : C_1 \rangle, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V$$
  to 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).
Association / Link Example

Signature:

\[ S = \{ \{ \text{Int} \}, \{ A, B, Z \}, \{ w: \text{Int} \}, \{ r \} \} \]

System state:

\[ \sigma = \{ 1A \leftarrow \{ w \rightarrow 2A \}, 2A \leftarrow \{ w \rightarrow 1B \}, 4B \leftarrow \emptyset, 3B \leftarrow \emptyset, 7B \leftarrow \emptyset, 8B \leftarrow \emptyset, 3A \leftarrow \emptyset \} \]

\[ \lambda = \{ r \mapsto \{ (7A, 42, 3B), (1A, 42, 7B), (1A, 42, 5B), (2A, 42, 3B), (42, 3B, 2A), (7A, 42) \} \} \]

\[ (42, 3B, 2A) \]

\[ (7A, 42) \]
Associations and OCL
**OCL and Associations: Syntax**

**Recall:** OCL syntax as introduced in Lecture 3, interesting part:

\[
expr ::= \ldots \\
| r_1(expr_1) : \tau_C \rightarrow \tau_D \\
| r_2(expr_1) : \tau_C \rightarrow \text{Set}(\tau_D)
\]

\[
\begin{align*}
  r_1 &: D_{0,1} \in \text{attr}(C) \\
  r_2 &: D_{\ast} \in \text{attr}(C)
\end{align*}
\]

Now becomes

\[
expr ::= \ldots \\
| \text{role}(expr_1) : \tau_C \rightarrow \tau_D \\
| \text{role}(expr_1) : \tau_C \rightarrow \text{Set}(\tau_D)
\]

\[
\begin{align*}
  \mu &= 0..1 \text{ or } \mu = 1..1 \\
  \text{otherwise}
\end{align*}
\]

if there is

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

**Note:**

- Association name as such **does not occur** in OCL syntax, role names do.

- \( expr_1 \) has to denote an object of a class which “participates” in the association.
Recall:

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

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

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

Now needed:

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

- We cannot simply write \( \sigma(u)(\text{role}) \).

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

- What we have is \( \lambda(r) \) (with association name \( r \), not with role name \( \text{role}! \)).

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

  But it yields a set of \( n \)-tuples, of which some relate \( u \) and some instances of \( D \).

- \( \text{role} \) denotes the position of the \( D \)'s in the tuples constituting the value of \( r \).
Assume $expr_1 : \tau_C$ for some $C \in \mathcal{C}$. Set $u_1 := I[expr_1]((\sigma, \lambda), \beta) \in \mathcal{D}(T_C)$.

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

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

where

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

$\langle r : \langle \text{role}_1 : _, _, _, _, _, \rangle, \ldots \langle \text{role}_n : _, _, _, _, _, \rangle \rangle,$  \hspace{1cm} $\text{role} = \text{role}_i$.

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

\[
I[\text{role(expr}_1\text{)}]((\sigma, \lambda), \beta) := \begin{cases} 
  u, & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } L(\text{role})(u_1, \lambda) = \{u\} \\
  \bot, & \text{otherwise}
\end{cases}
\]

\[
I[\text{role(expr}_1\text{)}]((\sigma, \lambda), \beta) := \begin{cases} 
  \{L(\text{role})(u_1, \lambda)\}, & \text{if } u_1 \in \text{dom}(\sigma) \\
  \bot, & \text{otherwise}
\end{cases}
\]

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

\[
\lambda(\text{workgroup}) = \{(1_s, 2_s, 3_s), (1_s, 3_s, 4_s), (5_s, 1_s, 1_s)\}
\]

\[
\text{allInstances}_{\text{Student} \rightarrow} \exists (s \mid s.l2 = s.l3)
\]

\[
\text{Exists}(s \mid s.l2 = s.l3)
\]

\[
\begin{array}{c}
\begin{aligned}
\text{leader} & \quad \text{Student} \\
0.1 & \quad 1 \\
\text{workgroup} & \quad l2 \\
0.1 & \quad l3 \\
\end{aligned}
\end{array}
\]

\[
\begin{align*}
\text{student} = & \{s_1, s_2, s_3, s_4, s_5\} \\
\text{leader} = & \{l2, l3\} \\
\lambda(\text{workgroup}) = & \{(1, 2, 3), (1, 3, 4), (5, 1, 1)\}
\end{align*}
\]
Associations: The Rest
Recapitulation: Consider the following association:

\[
\langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, \sigma_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, \sigma_n \rangle \rangle
\]

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

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

Navigability is treated similar to visibility:
Using names of non-navigable association ends \( (\nu = \times) \) are forbidden.

**Example:** Given

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

the following OCL expression is **not well-typed** wrt. navigability,

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

**The standard says:** navigation is...

- '−': ...possible
- '×': ...not possible
- '>': ...efficient

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

But: Pointers / references can faithfully be modelled by UML associations.
Recall: Multiplicity is a term of the form \( N_1..N_2, \ldots, N_{2k-1}..N_{2k} \) where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N}, \ N_{2k} \in \mathbb{N} \cup \{\ast\} \).

Define \( \mu_{OCL}^C(\text{role}) := \)

\[
\text{context } C \ \text{inv} : (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \\
\text{omitted if } N_{2k} = \ast
\]

for each \( \langle r : \ldots, \langle \text{role} : D, \mu, _, _, _, _ \rangle, \ldots, \langle \text{role}' : C, _, _, _, _ \rangle, \ldots \rangle \in V \) or

\( \langle r : \ldots, \langle \text{role}' : C, _, _, _, _ \rangle, \ldots, \langle \text{role} : D, \mu, _, _, _, _ \rangle, \ldots \rangle \in V, \)

with \( \text{role} \neq \text{role}' \), if \( \mu \neq 0..1, \mu \neq 1..1 \), and

\[\mu_{OCL}^C(\text{role}) := \text{context } C \ \text{inv} : \text{not(oclIsUndefined(\text{role}))}\]

if \( \mu = 1..1 \).

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

\[
\mu^C_{\text{OCL}}(\text{role}) = \text{context } C \text{ inv }:\]

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

\[CD :\]

0..1

\[\rightarrow\]

\[\text{v : Int}\]

\[\rightarrow\]

4, 17

\[\text{role}_2 \]

\[\rightarrow\]

3..*

\[\text{role}_3 \]

\[\rightarrow\]

C

- \{ context \ C \ inv : 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17 \}
- \{ context \ C \ inv : \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17 \}
- \cup \{ context \ C \ inv : 3 \leq \text{role}_3 \rightarrow \text{size()} \}
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 at most one $r$-link to a single other object</td>
<td>current setting</td>
</tr>
<tr>
<td>bag</td>
<td>one object may have multiple $r$-links to a single other object</td>
<td>have $\lambda(r)$ yield multisets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an $r$-link is a sequence 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>$T_D \to Set(T_C)$</td>
</tr>
<tr>
<td>bag</td>
<td>$T_D \to Bag(T_C)$</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>$T_D \to Seq(T_C)$</td>
</tr>
</tbody>
</table>

For subsets, redefines, union, etc. see (?, 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..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, 2011b, 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

\[
\begin{align*}
(i) & \quad C \times_{\text{role}} \{0..1\} D \\
(ii) & \quad C \times_{\text{role}} \{\ast\} D 
\end{align*}
\]

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

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

- In both cases, \(\text{role} \in \text{atr}(C)\).

- We drop \(\lambda\) and go back to our nice \(\sigma\) with \(\sigma(u)(\text{role}) \subseteq \mathcal{D}(D)\).
Tell Them What You’ve Told Them…

- From class diagrams with (general) associations, we obtain extended signatures.

- Links (instances of associations) “live on their own” in the \( \lambda \) in extended system states \((\sigma, \lambda)\).

- OCL considers role names, the semantics is (more or less) straightforward.

The Rest:

- navigability is treated like visibility,
- view multiplicities as shorthand for constraints,
- properties, ownership, “diamonds”: exist

Back to the main track:

For simplicity, let’s restrict the following discussion to \( C_{0,1} \) and \( C^* \) as before (now viewed as abbreviations for particular associations).
References
References
