Course Map
Recall: Plan & Extended Signature
Overview

• Class diagram:

```
 C
  v : Int
  d : D

 D
  e : C
```

Alternative presentation:

```
 C
  v : Int
  d

 D
```

• Signature:

\[ \mathcal{S} = \mathcal{S} = (\{ \text{Int} \}, \{ \text{C, D} \}, \{ \text{v, d} : \text{D} \rightarrow \{ \text{c} \} \}) \]

• Example system state:

\[ \sigma = (1 \text{C} \rightarrow \{ v \rightarrow 27, d \rightarrow \{ 5 \text{D}, 7 \text{D} \} \}, \text{5D} \rightarrow \{ e \rightarrow \{ 1 \text{C} \}, 7 \text{D} \rightarrow \{ e \rightarrow \{ 1 \text{C} \} \} \}) \]

• Object diagram:

```
 C
  v : Int
  d

 D
```

• Class diagram (with ternary association):

```
 A
  w : Int

 B
  z

 Z
```

• Signature: extend again to represent association \( r \) with

- association ends \( a, b, \) and \( z \)
  (each with multiplicity, visibility, etc.)

• Example system state:

\[ \sigma = \{ 1 \text{A} \rightarrow \{ w \rightarrow 13 \}, 1 \text{B} \rightarrow \{ 0, 1 \text{Z} \rightarrow \{ 0 \} \} \}

\[ \lambda = \{ r \rightarrow \{ 1 \text{A}, 1 \text{B}, 1 \text{Z} \}, 1 \text{A}, 1 \text{B}, 2 \text{Z} \} \}

• 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, \( \text{qualifier} \in \{ u, o, l, d \} \)
- a visibility,
- a navigability,
- an ownership,
- and possibly a diamond.

Wanted: places in the signature to represent the information from the picture.
**Temporarily (Lecture 7 – 9) Extended Signature**

**Definition.** An (Extended) Object System Signature (with Associations) is a quadruple \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, V, atr) \) where

- ... each element of \( V \) is
  - either a basic type attribute \( (v : T, \xi, expr_1, P_v) \) with \( T \in \mathcal{F} \)
  - or an association of the form
    \[ \begin{align*}
    r : (\text{role}_1 : C_1, \mu_1, P_1, \xi_1, o_1, \nu_1), \\
    & \vdots \\
    r : (\text{role}_n : C_n, \mu_n, P_n, \xi_n, o_n, \nu_n)).
    \end{align*} \]

(ends with multiplicity \( \mu_n \), properties \( P_n \), visibility \( \xi_n \), navigability \( \nu_n \), ownership \( o_n, 1 \leq i \leq n \))

- ... \( atr : \mathcal{C} \to 2^{\{v \in V \mid v.T \in \mathcal{F}\}} \) 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".

**Associations in Class Diagrams**
From Association Lines to Extended Signatures

Association Example

Signature:

\[ \mathcal{S} = \{ \{ \text{Int} \}, \{ C, D \}, \{ \times, \div, +, o \}, r \} \]

\[ \langle r : \langle n : D, \ast, \{ \text{unique} \}, +, 0 \rangle, 0 \rangle \]

\[ \langle c : C, 0, \{ \text{unique} \}, -, x, > \rangle \]

\[ \{ C \mapsto r, \rangle \mapsto \{ x \} \} \]

\[ D \mapsto \{ x \} \]
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 \( A_{C_1 \ldots C_n} \) if the name is missing.

  **Example**:
  
  \[
  \begin{array}{c}
  C \quad \text{A}_{C \ldots D} \quad D \\
  \end{array}
  \]
  
  \[
  \begin{array}{c}
  C \quad D \\
  \end{array}
  \]

- **Reading Direction**: no default.

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

  **Example**:
  
  \[
  \begin{array}{c}
  C \quad e \quad d \quad D \\
  \end{array}
  \]
  
  \[
  \begin{array}{c}
  C \quad D \\
  \end{array}
  \]

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

  \[
  \begin{array}{c}
  C \quad \text{itsC} \quad \text{itsD} \quad D \\
  \end{array}
  \]
  
  \[
  \begin{array}{c}
  C \quad D \\
  \end{array}
  \]

- **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
Associations in General

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).

Links in System States

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 \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, V, \text{atr}) \).

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

- a type-consistent mapping (as before)
  \[ \sigma : \mathcal{D}(\mathcal{C}) \leftrightarrow (\text{atr}(\mathcal{C}) \leftrightarrow \mathcal{D}(\mathcal{F})), \]
- a mapping \( \lambda \) which maps each association
  \[ \langle r : \langle \text{role}_1 : C_1, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V \text{ to a relation} \]
  \[ \lambda(r) \subseteq \mathcal{D}(C_1) \times \cdots \times \mathcal{D}(C_n) \]
  (i.e. a set of type-consistent \( n \)-tuples of identities).
**Association / Link Example**

### Signature:

\[ \mathcal{S} = \{ \{ a \in \text{Int} \}, \{ A, Z, B \}, \{ w : \text{Int} \}, \langle r : \langle a : A, 0 \leq a \leq x, \{ w \in \text{Real} \} \rangle, 0 \leq \rangle \} \]

### System state:

\[ \gamma = \{ \langle 1, \{ a = 2 \} \rangle, \langle 2, \{ a = 3 \} \rangle, \langle 3, \{ a \in \{ 1 \} \} \rangle, \langle 4, \{ a \in \{ 1 \} \} \rangle \} \]

\[ \gamma' = \{ \langle 2, 3 \rangle, \langle 4, 5 \rangle \} \]

\[ \gamma'' = \{ \langle 4, 3 \rangle, \langle 2, 5 \rangle \} \]

### Associations and OCL
OCL and Associations: Syntax

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

\[
\begin{align*}
\text{expr} ::= \ldots & | r_1(\text{expr}_1) : \tau_C \rightarrow \tau_D & r_1 : D_{\mathit{a_1}} \in \text{atr}(C) \\
& | r_2(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) & r_2 : D_n \in \text{atr}(C)
\end{align*}
\]

Now becomes

\[
\begin{align*}
\text{expr} ::= \ldots & | \text{role} (\text{expr}_1) : \tau_C \rightarrow \tau_D & \mu = 0..1 \text{ or } \mu = 1..1 \\
& | \text{role} (\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) & \text{otherwise}
\end{align*}
\]

if there is

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

Note:

- Association name as such does not 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: Semantics

Recall: Assume \text{expr} : \tau_C for some \( C \in \mathcal{C} \). Set \( u_1 := I[\text{expr}_1](\sigma, \beta) \in \mathcal{P}(\tau_C) \).

\[
\begin{align*}
\cdot I[r_1(\text{expr}_1)](\sigma, \beta) := \begin{cases} 
  u & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\
  \bot & \text{otherwise}
\end{cases} \\
\cdot I[r_2(\text{expr}_1)](\sigma, \beta) := \begin{cases} 
  \sigma(u_1)(r_2) & \text{if } u_1 \in \text{dom}(\sigma) \\
  \bot & \text{otherwise}
\end{cases}
\end{align*}
\]

Now needed:

\[
I[\text{role}(\text{expr}_1)]((\sigma, \lambda), \beta)
\]

- 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}!).
  \[
  \langle r : \ldots, (\text{role} : D, \mu, \ldots), \ldots, (\text{role}' : C, \ldots), \ldots \rangle
  \]
  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 \( u_1 := I[expr_1]((\sigma, \lambda), \beta) \in \mathcal{P}(\Theta_C) \).

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

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

where

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

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**

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

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

\[
\text{allInstances}_{\text{Student}} \rightarrow \exists (s \mid s.l2 = s.l3)
\]
Recapitulation: Consider the following association:

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

- **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** \( \alpha \): 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

![Diagram](image)

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

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

The standard says: navigation is...

- \( - \) : possible
- \( \times \) : 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.

Multiplicities as Constraints

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 \( \mu^{\mathcal{C}}_{\text{OCL}}(\text{role}) := \)

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

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

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

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

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

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
**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 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 multi-sets</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>

Property OCL Typing of expression $\text{role} (\text{expr})$

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

For subsets, redefines, union, etc. see (7, 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

  (i) \[ C \rightarrow_{0.1} \] \[ D \]  

  (ii) \[ C \rightarrow \ast \] \[ D \]

(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 \( \sigma(u)(\text{role}) \subseteq \mathcal{P}(D) \).

**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
