Software Design, Modelling and Analysis in UML

Lecture 08: Class Diagrams III

2012-11-21

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

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 $Inv(\mathcal{CD})$.
- **Properties**: for now assume $P_v = \{\texttt{unique}\}$.
- **Properties** (in general) and **ownership**: later.

Association Semantics: The System State Aspect

Associations in General

Recall: We consider associations of the following form:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \dots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

Only these parts are relevant for extended system states:

$$\langle r : \langle role_1 : C_1, \_, P_1, \_, \_, \_ \rangle, \dots, \langle role_n : C_n, \_, P_n, \_, \_, \_ \rangle$$

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

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

$$\langle r : \langle role_1 : C_1, \_, P_1, \_, \_, \_\rangle, \dots, \langle role_n : C_n, \_, P_n, \_, \_, \_\rangle$$

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

Definition. Let $\mathscr{D}$ be a structure of the (extended) signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.

A system state of $\mathscr{S}$ wrt. $\mathscr{D}$ is a pair (σ, λ) consisting of

- a type-consistent mapping

$$\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (atr(\mathscr{C}) \nrightarrow \mathscr{D}(\mathscr{T})),$$

values for basic type attributes only

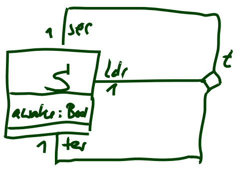
- a mapping λ which assigns each association $\langle r : \langle role_1 : C_1\rangle, \dots, \langle role_n : C_n\rangle\rangle \in V$ a relation

$$\lambda(r) \subseteq \mathscr{D}(C_1) \times \cdots \times \mathscr{D}(C_n)$$

(i.e. a set of type-consistent n-tuples of identities).

Example

order of assoc. ends in $\mathscr{S}$ does matter



$\langle t : \langle ldr : S, \dots\rangle$
$\langle sec : S, \dots\rangle$
$\langle ter : S, \dots\rangle\rangle$

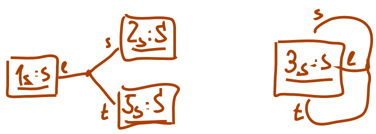
ldr sec ter

$\sigma = \{1_S \mapsto \{aw \mapsto 1\}, 2_S \mapsto \{aw \mapsto 0\}, 3_S \mapsto \{aw \mapsto 1\}, 27_S \mapsto \{aw \mapsto 1\}\}$

$\lambda = \{ t \mapsto \{ (1_S, 2_S, 3_S),$ } students may join multiple groups
$(1_S, 27_S, 3_S),$
$(2_S, 5_S, 6_S),$ } general links may also have dangling references
$(3_S, 3_S, 3_S)\}$ } one student may assume all roles (if this is not desired, then add a constraint context S inv: ldr ≠ sec and sec ≠ ter)

$\cap$ $\mathscr{D}(S) \times \mathscr{D}(S) \times \mathscr{D}(S)$

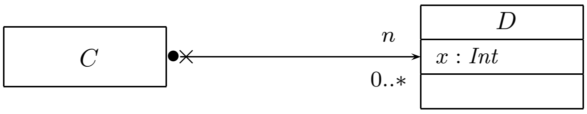
OBJECT DIAGRAMS:



↳ we would need hyperedges

WE WILL NOT FORMALLY DEFINE THAT

Association/Link Example



Signature:

by convention

$$\mathscr{S} = (\{Int\}, \{C, D\}, \{x : Int, \langle A\_C\_D : \langle c : C, 0..*, +, \{\texttt{unique}\}, \times, 1\rangle, \langle n : D, 0..*, +, \{\texttt{unique}\}, >, 0\rangle\rangle\}, \{C \mapsto \emptyset, D \mapsto \{x\}\})$$

A **system state** of $\mathscr{S}$ (some reasonable $\mathscr{D}$) is (σ, λ) 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

object 1_C is related to 3_D and 7_D by A_C_N

Extended System States and Object Diagrams

Legitimate question: how do we represent system states such as

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

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

as **object diagram**?

See 7a and 8.

Associations and OCL

OCL and Associations: Syntax

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

$$\begin{array}{llll} expr ::= \dots & \mid r_1(expr_1) & : \tau_C \to \tau_D & r_1 : D_{0,1} \in atr(C) \\ & \mid r_2(expr_1) & : \tau_C \to Set(\tau_D) & r_2 : D_* \in atr(C) \end{array}$$

Now becomes

$$\begin{array}{llll} expr ::= \dots & \mid role(expr_1) & : \tau_C \to \tau_D & \mu = 0..1 \text{ or } \mu = 1 \\ & \mid role(expr_1) & : \tau_C \to Set(\tau_D) & \text{otherwise} \end{array}$$

if

$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots \rangle \in V \text{ or}$$

$$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots \rangle \in V, role \neq role'.$$

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

$$\begin{array}{llll} expr ::= \dots & \mid role(expr_1) & : \tau_C \to \tau_D & \mu = 0..1 \text{ or } \mu = 1 \\ & \mid role(expr_1) & : \tau_C \to Set(\tau_D) & \text{otherwise} \end{array}$$

if

$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots\rangle \in V \text{ or}$$

$$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots\rangle \in V, role \neq role'.$$

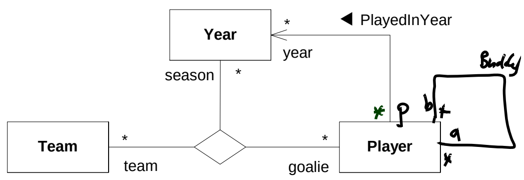


Figure 7.21 - Binary and ternary associations [OMG, 2007b, 44].

- context Player inv: size(years(self)) > 0 — Ok
- context Player inv: size(p (self)) > 0 — NOT ok
- context Player inv: size (season(self)) > 0 — Ok
- context Player inv: size(a(self)) > 0 — Ok

OCL and Associations: Semantics

Recall: (Lecture 03)

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

- $I[\![r_1(expr_1)]\!](\sigma, \beta) := \begin{cases} u & \text{, if } u_1 \in \mathrm{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\ \bot & \text{, otherwise} \end{cases}$
- $I[\![r_2(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \bot & \text{, otherwise} \end{cases}$

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 $atr(C)$).
- 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 $C \in \mathscr{C}$. Set $u_1 := I[\![expr_1]\!]((\sigma, \lambda), \beta) \in \mathscr{D}(\tau_C)$.

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

  "look up role cont. u_1 in λ"

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

where

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

(select rows where u occurs; projection on the i-th component element-wise)

if

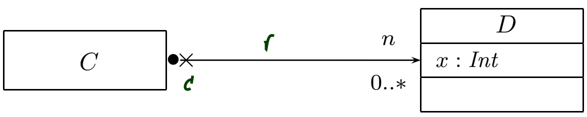
$$\langle r : \dots \langle role_1 : \_, \_, \_, \_, \_, \_\rangle, \dots \langle role_n : \_, \_, \_, \_, \_, \_\rangle, \dots \rangle, role = role_i.$$

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

OCL and Associations Example

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

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



$\langle r : \langle c : C, \dots \rangle, \langle n : D, \dots \rangle \rangle$

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

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

$I[\![self\ .\ n]\!]((\sigma, \lambda), \{self \mapsto 1_C\}) = I[\![n(self)]\!]((\sigma, \lambda), \beta)$

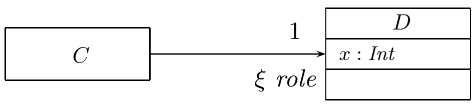
$= L(n)(\beta(self), \lambda) = L(n)(1_C, \lambda) = \{(1_C, 3_D), (1_C, 7_D)\} \downarrow 2 = \{3_D, 7_D\}$

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



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

$$\text{context } C \text{ inv} : \mathit{self}.\mathit{role}.x > 0$$ NOT if ξ = private

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

$$(\mathit{Assoc}_1) \quad \frac{A, B \vdash \mathit{expr}_1 : \tau_C}{A, B \vdash \mathit{role}(\mathit{expr}_1) : \tau_D}, \quad \begin{array}{l}\mu = 0..1 \text{ or } \mu = 1, \\ \xi = +, \text{ or } \xi = - \text{ and } C = B\end{array}$$

$$\langle r : \ldots \langle \mathit{role} : D, \mu, \_, \xi, \_, \_\rangle, \ldots \langle \mathit{role}' : C, \_, \_, \_, \_, \_\rangle, \ldots\rangle \in V$$

Navigability

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

Question: given

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

$$\text{context } D \text{ inv} : \mathit{self}.\mathit{role}.x > 0$$ NOT well-typed

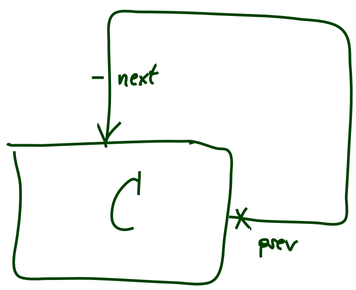
The standard says:

- '$-$': navigation is possible
- '$\times$': 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:



The Rest

Recapitulation: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1\rangle, \dots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n\rangle\rangle$$

- Association name r and role names/types $role_i/C_i$ induce extended system states λ.
- Multiplicity μ is considered in OCL syntax.
- Visibility ξ and navigability ν give rise to well-typedness rules.

Now the rest:

- Multiplicity μ: 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:

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N})$$

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

Recall: we can normalize each multiplicity to the form

$$N_1..N_2, \dots, N_{2k-1}..N_{2k}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \dots, N_{2k} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

Define

$\mu_{\mathsf{OCL}} =$ context C inv :

$$(N_1 \leq role \text{ -> size}() \leq N_2) \text{ and } \dots \text{ and } (N_{2k-1} \leq role \text{ -> size}() \leq N_{2k})$$

for each

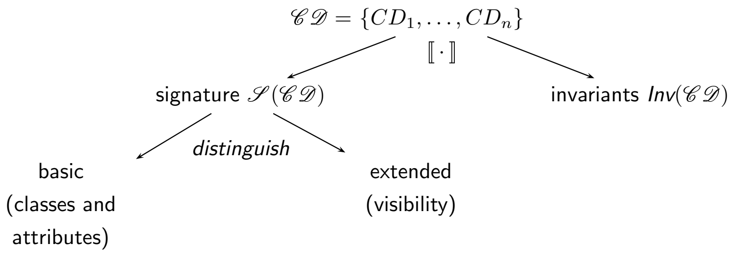
$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots\rangle \in V \text{ or}$$

$$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots\rangle \in V, role \neq role'.$$

Note: in n-ary associations with $n > 2$, there is redundancy.

Multiplicities as Constraints of Class Diagram

Recall:



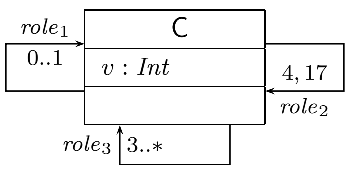
From now on: $Inv(\mathscr{CD}) = \{\text{constraints occurring in notes}\} \cup \{\mu_{\text{OCL}} \mid$

$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots \rangle \in V \text{ or}$$
$$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots \rangle \in V,$$
$$role \neq role', \mu \notin \{0..1, 1\}\}.$$

Multiplicities as Constraints Example

$\mu_{\text{OCL}} =$ context C inv :
$(N_1 \leq role$ -> size() $\leq N_2)$ and $\dots$ and $(N_{2k-1} \leq role$ -> size() $\leq N_{2k})$

$\mathcal{CD}$:



$Inv(\mathcal{CD}) =$

- $\{$context C inv : $4 \leq role_2$ -> size() ≤ 4 or $17 \leq role_2$ -> size() $\leq 17\}$
  $= \{$context C inv : $role_2$ -> size() $= 4$ or $role_2$ -> size() $= 17\}$
- $\cup \{$context C inv : $3 \leq role_3$ -> 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 $\mu_{\mathsf{OCL}} = \mathit{true}$ 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

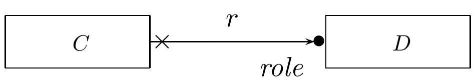
We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

Property	Intuition	Semantical Effect
unique	one object has **at most one** r-link to a single other object	**current setting**
bag	one object may have **multiple** r-links to a single other object	have $\lambda(r)$ yield multi-sets
ordered, **sequence**	an r-link is a **sequence** of object identities (possibly including duplicates)	have $\lambda(r)$ yield sequences

Property	OCL Typing of expression $role(expr)$
unique	$\tau_D \to Set(\tau_C)$
bag	$\tau_D \to Bag(\tau_C)$
ordered, **sequence**	$\tau_D \to Seq(\tau_C)$

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

Ownership



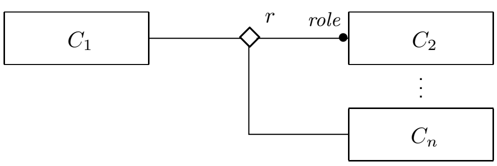
Intuitively it says:

Association r is **not a "thing on its own"** (i.e. provided by λ),
but association end '$role$' is **owned** by C (!).
(That is, it's stored inside C object and provided by σ).

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:



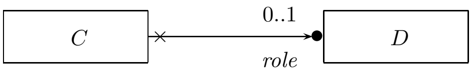
Back to the Main Track

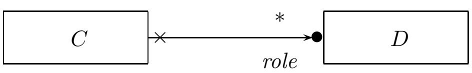
Back to the main track:

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) 

(ii) 

(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, $role \in atr(C)$.
- We drop λ and go back to our nice σ with $\sigma(u)(role) \subseteq \mathscr{D}(D)$.

References

References

[Ambler, 2005] Ambler, S. W. (2005). *The Elements of UML 2.0 Style*. Cambridge University Press.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.