Software Design, Modelling and Analysis in UML

Lecture 09: Class Diagrams IV

2011-12-07

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lectures:

- Started to discuss "associations", 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:**
  - Treat "the rest".
  - Where do we put OCL constraints?
  - Modelling guidelines, in particular for class diagrams (following [Ambler, 2005])

Associations: The Rest

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/**Navigability** ν: well-typedness.

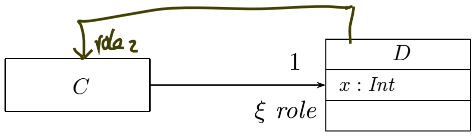
Now the rest:

- **Multiplicity** μ: we propose to view them as constraints.
- **Properties** P_i: even more typing.
- **Ownership** o: getting closer to pointers/references.
- **Diamonds**: exercise.

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):

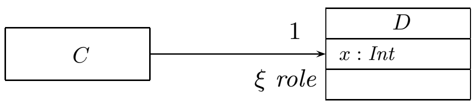
$$\text{context } C \text{ inv} : self.role.x > 0$$ well-typed always

context D inv : self.role$_2$.role.x > 0 not w.-t. if ξ = private

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):

$$\text{context } C \text{ inv} : self.role.x > 0$$

x(role (self))

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

$$(Assoc_1) \quad \frac{A, D \vdash expr_1 : \tau_C}{A, D \vdash role(expr_1) : \tau_D}, \quad \begin{array}{l} \mu = 0..1 \text{ or } \mu = 1, \\ \xi = +, \text{ or } \xi = - \text{ and } C = D \end{array}$$

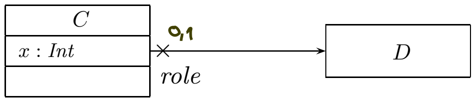
$$\langle r : \ldots \langle role : D, \mu, \_, \xi, \_, \_ \rangle, \ldots \langle role' : C, \_, \_, \_, \_, \_ \rangle, \ldots \rangle \in V$$

Navigability

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

↳ not well-typed

Question: given



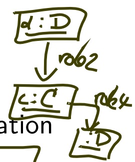
is the following OCL expression well-typed or not (wrt. navigability):

$$\text{context } D \text{ inv} : self.role.x > 0$$

Navigability

(*) class D { C* role2; };

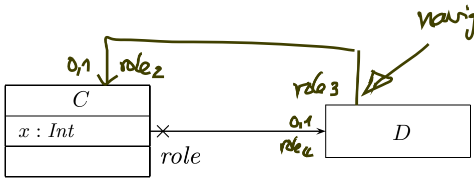
class C { int x; D* role4; };



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

navigation to here is possible from C-objects

Question: given



Task: give C++ expression which computes d given c. Difficult! In UML: a database lookup

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

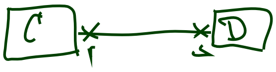
$$\text{context } D \text{ inv} : self.role.x > 0$$

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.

-  makes no sense ...?
  - in general there is no OCL expression involving r or s which is well-typed
  - for requirements, we may disregard well-typedness and write context C inv: self.s.x > 0 (artificial example)
- so, difference between '–' and '×' and '>' and '×' is in well-typedness of exprs — what about '–' and '>'?
  - in our formal, math. setting of UML models: there's no difference
  - for the implementation: define what "efficient" means and tell it to the programmers

The Rest of the Rest

Recapitulation: Consider the following association:

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

- **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/**Navigability** ν: well-typedness. ✓

Now the rest:

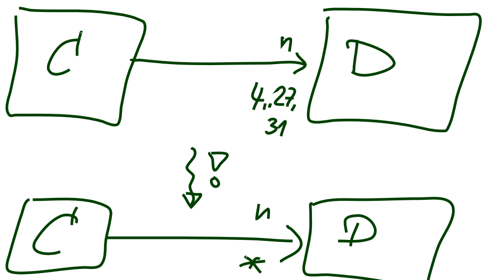
- **Multiplicity** μ: we propose to view them as constraints.
- **Properties** P_i: even more typing.
- **Ownership** o: getting closer to pointers/references.
- **Diamonds**: exercise.

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.



context C inv : $4 \leq$ size(n) ≤ 27 or size(n) = 31

size(self.n)

Multiplicities as Constraints

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

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

$\mu ::= N..M \mid \mu,\mu \quad N,M \in \mathbb{N} \cup \{*\}$

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

~~**Recall**~~ Observe: we can normalize each multiplicity μ to the form

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

e.g. 31 to 31..31

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

e.g. * to 0..*

Multiplicities as Constraints

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

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

Define $\mu^C_{\mathsf{OCL}}(role) :=$ context C inv :

$$(N_1 \leq role \text{ -> size}() \leq N_2) \text{ or } \dots \text{ or } (N_{2k-1} \leq role \text{ -> size}() \underbrace{\leq N_{2k}}_{\text{omit if } N_{2k} = *})$$

for each $\mu \neq 0..1$, $\mu \neq 1$,

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

And **define**

$$\boxed{\mu^C_{\mathsf{OCL}}(role)} := \text{context } C \text{ inv : not(oclIsUndefined}(role))$$

for each $\mu = 1$.

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

Multiplicities as Constraints of Class Diagram

Recall/Later:

$\mathscr{C}\mathscr{D} = \{CD_1, \dots, CD_n\}$

$[\![\cdot]\!]$

signature $\mathscr{S}(\mathscr{C}\mathscr{D})$ — invariants *Inv*($\mathscr{C}\mathscr{D}$)

distinguish

basic (classes and attributes) — extended (visibility)

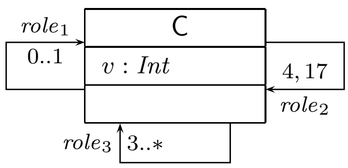
From now on: *Inv*($\mathscr{C}\mathscr{D}$) $= \{$constraints occurring in notes$\} \cup \{\mu^C_{\mathsf{OCL}}(role) \mid$

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

Multiplicities as Constraints Example

$$\mu^C_{\mathsf{OCL}}(role) = \text{context } C \text{ inv : } (N_1 \le role \text{ -> size()} \le N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \le role \text{ -> size()} \le N_{2k})$$

$\mathcal{CD}$:



$Inv(\mathcal{CD}) =$

- $\{\text{context } C \text{ inv : } 4 \le role_2 \text{ -> size()} \le 4 \text{ or } 17 \le role_2 \text{ -> size()} \le 17\}$
  $= \{\text{context } C \text{ inv : } role_2 \text{ -> size()} = 4 \text{ or } role_2 \text{ -> size()} = 17\}$
- $\cup \{\text{context } C \text{ inv : } 3 \le role_3 \text{ -> size()}\}$

Why Multiplicities as Constraints?

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  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 $0, \dots, N$,
- set types,

then we could simply **restrict** the syntax of multiplicities to

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

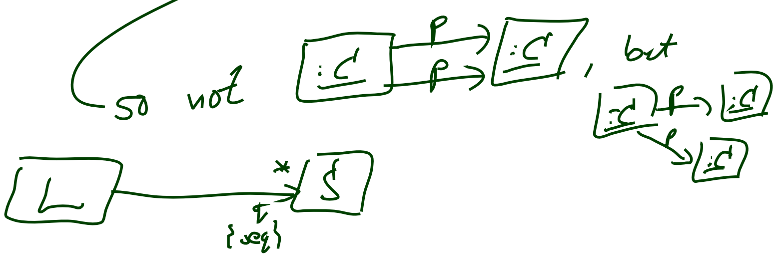
and don't think about constraints
(but use the obvious 1-to-1 mapping to types)...

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



Properties

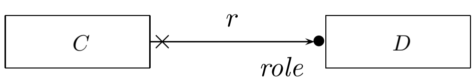
We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

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



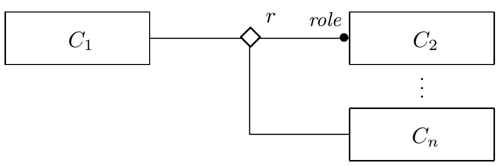
Intuitively it says:

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:



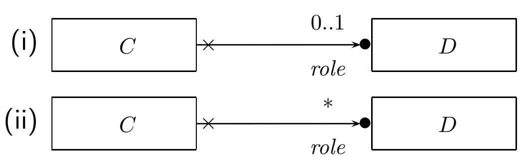
Back to the Main Track

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



  (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 $\sigma(u)(role) \subseteq \mathscr{D}(D)$.

OCL Constraints in (Class) Diagrams

Where Shall We Put OCL Constraints?

Numerous options:

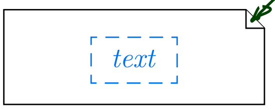
(i) Additional documents.

(ii) Notes.

(iii) Particular dedicated places.

(ii) **Notes**:

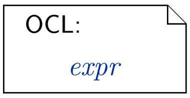
A UML **note** is a picture of the form



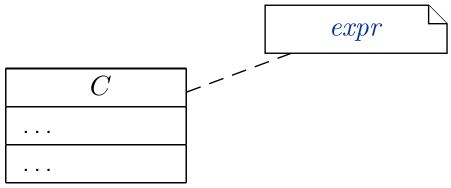
Eselsohr,
English: dog ears

text can principally be **everything**, in particular **comments** and **constraints**.

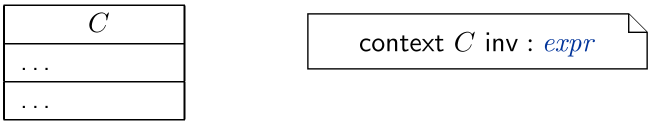
Sometimes, content is **explicitly classified** for clarity:



OCL in Notes: Conventions

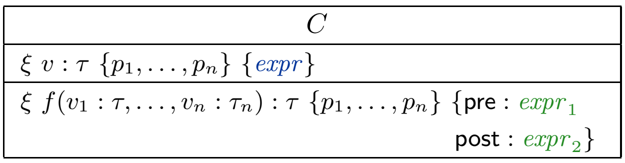


stands for

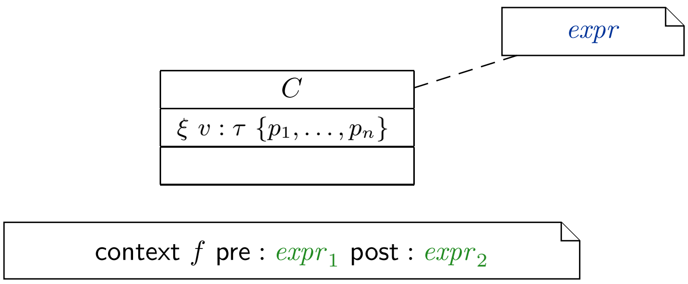


Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)



For simplicity, we view the above as an abbreviation for



Invariants of a Class Diagram

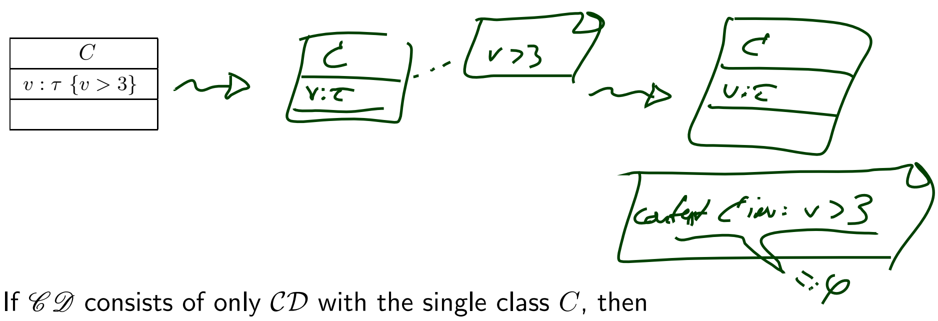
- Let $\mathcal{CD}$ be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

$$Inv(\mathcal{CD})$$

  as the set $\{\varphi_1, \dots, \varphi_n\}$ of OCL constraints **occurring** in notes in $\mathcal{CD}$ — after **unfolding** all abbreviations (cf. next slides).

- As usual: $Inv(\mathscr{CD}) := \bigcup_{\mathcal{CD} \in \mathscr{CD}} Inv(\mathcal{CD})$.
- **Principally clear**: $Inv(\cdot)$ for any kind of diagram.

Invariant in Class Diagram Example



If $\mathscr{CD}$ consists of only $\mathcal{CD}$ with the single class C, then

- $Inv(\mathscr{CD}) = Inv(\mathcal{CD}) = \{\varphi\}$

Semantics of a Class Diagram

Definition. Let $\mathscr{CD}$ be a set of class diagrams.

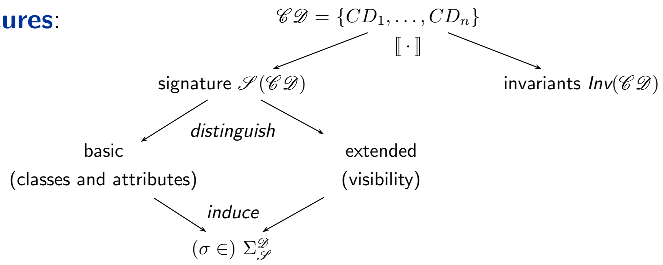
We say, the semantics of $\mathscr{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathscr{CD}$, denoted

$$[\![\mathscr{CD}]\!] := \langle \mathscr{S}(\mathscr{CD}), Inv(\mathscr{CD}) \rangle.$$

Given a structure $\mathscr{D}$ of $\mathscr{S}$ (and thus of $\mathscr{CD}$), the class diagrams describe the system states $\Sigma_{\mathscr{S}}^{\mathscr{D}}$. Of those, **some** satisfy $Inv(\mathscr{CD})$ and some don't.

We call a system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ consistent if and only if $\sigma \models Inv(\mathscr{CD})$.

In pictures:



Pragmatics

Recall: a UML **model** is an image or pre-image of a software system.

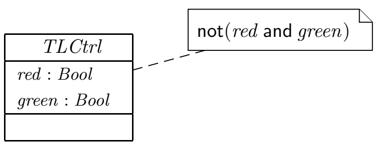
A set of class diagrams $\mathscr{CD}$ with invariants $Inv(\mathscr{CD})$ describes the **structure** of system states.

Together with the invariants it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $Inv(\mathscr{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $Inv(\mathscr{CD})$ are used.

(The exact meaning of "use" will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

Example: highly abstract model of traffic lights controller.



Constraints vs. Types

Find the 10 differences:

C
$x : \mathit{Int}\ \{x = 3 \lor x > 17\}$

C
$x : T$

$$\mathscr{D}(T) = \{3\} \cup \{n \in \mathbb{N} \mid n > 17\}$$

- $x = 4$ is well-typed in the left context, a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context, there cannot be a system state with $\sigma(u)(x) = 4$ because $\sigma(u)(x)$ is supposed to be in $\mathscr{D}(T)$ (by definition of system state).

Rule-of-thumb:

- If something **"feels like" a type** (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a **requirement** or restriction of an otherwise useful type, then make it a constraint.

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.