Software Design, Modelling and Analysis in UML

Lecture 9: Class Diagrams IV

2015-12-01

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Associations syntax and semantics.
- Associations in OCL syntax.

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Compute the value of a given OCL constraint in a system state with links.
  - How did we treat "multiplicity" semantically?
  - What does "navigability", "ownership", ... mean?
  - ...
- **Content:**
  - Associations and OCL: semantics.
  - Associations: the rest.

Associations and OCL Cont'd

Recall:Associations and OCL Syntax

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

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

Now becomes

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

if there is

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

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

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.

OCL and Associations: Semantics

Recall:

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

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

Now needed:

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

- We cannot simply write $\sigma(u)(role)$.
  Recall: $role$ is (**for the moment**) not an attribute of object u (not in $atr(C)$).
- What we have is $\lambda(r)$ (with association name r, not with role name $role$!).

$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots \rangle$$

  But it yields a set of n-tuples, of which **some** relate u and 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 $u_1 := I[\![expr_1]\!]((\sigma, \lambda), \beta) \in \mathscr{D}(T_C)$.

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

where

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

if

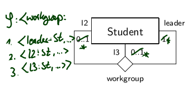
$$\langle r : \langle role_1 : \_, \_, \_, \_, \_, \_\rangle, \dots \langle role_n : \_, \_, \_, \_, \_, \_\rangle, \rangle, \quad role = role_i.$$

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

OCL and Associations Semantics: Example

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

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



$\lambda(\text{workgroup}) = \{(1_S, 2_S, 3_S), (1_S, 3_S, 5_S), (5_S, 1_S, 1_S)\}$

$5_S \in \mathrm{dom}\,\sigma$

allInstances$_{Student}$ -> Exists($s \mid s.l2 = s.l3$)

$I[\![s.l2]\!]((\sigma,\lambda),\{s \mapsto 5_S\}) = \{(1_S,3_S,5_S),(5_S,1_S,1_S)\} \downarrow 2 = \{3_S, 1_S\} = R_1$

$I[\![s.l3]\!]((\sigma,\lambda),\beta) = \{(1_S,3_S,5_S),(5_S,1_S,1_S)\} \downarrow 3 = \{5_S, 1_S\} = R_2$

$I[\![=]\!](R_1, R_2)$ = false

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 (σ, λ).
- **Multiplicity** μ is considered in OCL syntax.
- **Visibility** ξ / **Navigability** ν: well-typedness (in a minute).

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.

Navigability

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

Example: Given

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

context D 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.

Multiplicities as Constraints

Recall: Multiplicity is a term of the form $N_1..N_2, \dots, N_{2k-1}..N_{2k}$
where $N_i \le N_{i+1}$ for $1 \le i \le 2k$, $N_1, \dots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{*\}$.

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

context C inv : $(N_1 \le role$ -> size() $\le N_2)$ or $\dots$ or $(N_{2k-1} \le role$ -> size() $\le N_{2k})$ (omit if $N_{2k} = *$)

for each $\langle r : \dots, \langle role : D, \mu, \_,\_,\_,\_\rangle, \dots, \langle role' : C, \_,\_,\_,\_,\_\rangle, \dots\rangle \in V$ or

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

with $role \ne role'$, if $\mu \ne 0..1$, $\mu \ne 1..1$, and

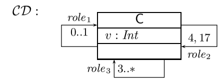
$$\mu^C_{\mathsf{OCL}}(role) := \text{context } C \text{ inv : not(oclIsUndefined}(role))$$

if $\mu = 1..1$.

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

Multiplicities as Constraints Example

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

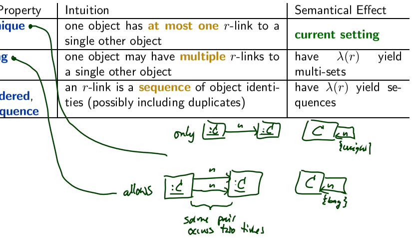


- $\mu^C_{OCL}(role_2)$ = context C inv : $4 \le role_2$ -> size ≤ 4 or $17 \le role_2$ -> size ≤ 17
- $\mu^C_{OCL}(role_3)$ = context C inv : $3 \le role_3$ -> size

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

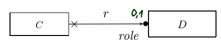
We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

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	$T_D \to Set(T_C)$
bag	$T_D \to Bag(T_C)$
ordered, sequence	$T_D \to Seq(T_C)$

For **subsets**, **redefines**, **union**, etc. see (?, 127).

Ownership



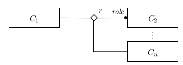
Intuitively it says:

> 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 $1..1$, 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:



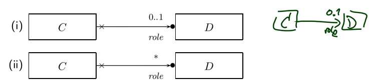
Back to the Main Track

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

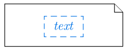
OCL Constraints in (Class) Diagrams

Where Shall We Put OCL Constraints?

Two options:
(i) Notes.
(ii) Particular dedicated places.

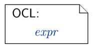
(i) **Notes**:

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

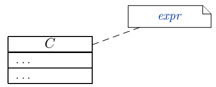


text can principally be **everything**, in particular **comments** and **constraints**.

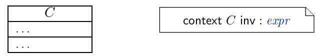
Sometimes, content is **explicitly classified** for clarity:



OCL in Notes: Conventions

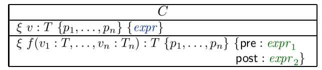


stands for

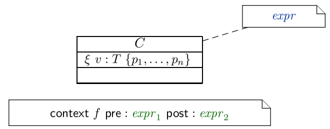


Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behavioural features: later)



For simplicity, we view the above as an abbreviation for



Invariants of a Class Diagram

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

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

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

- **As usual**: consider all invariants in all notes in any class diagram — plus implicit multiplicity-induced invariants.

$$\mathit{Inv}(\mathscr{CD}) = \bigcup_{\mathcal{CD} \in \mathscr{CD}} \mathit{Inv}(\mathcal{CD}) \cup \{\mu^C_{\mathsf{OCL}}(role) \mid \langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots \rangle \in V \text{ or } \langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots \rangle \in V\}.$$

- **Analogously**: $\mathit{Inv}(\cdot)$ for any kind of diagram (like **state machine diagrams**).

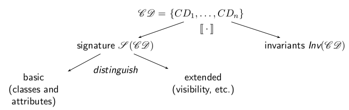
Semantics of a Class Diagram

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

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

Given a structure $\mathscr{D}$ of $\mathscr{S}$ (and thus of $\mathscr{CD}$), the class diagrams describe the system states $\Sigma^{\mathscr{D}}_{\mathscr{S}}$, of which **some** may satisfy $\mathit{Inv}(\mathscr{CD})$.

In pictures:



Pragmatics

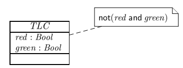
Recall: a UML **model** is an image or pre-image of a software system.

A set of class diagrams $\mathscr{CD}$ describes the **structure** of system states.
Together with the invariants $\mathit{Inv}(\mathscr{CD})$ it can be used to state:

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

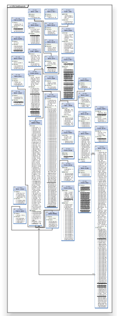
Example: highly abstract model of traffic lights controller.

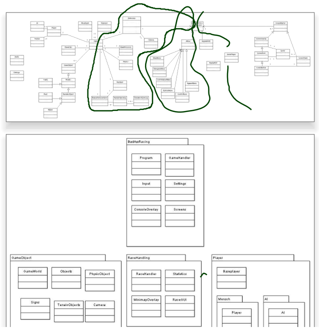


Design Guidelines for (Class) Diagram

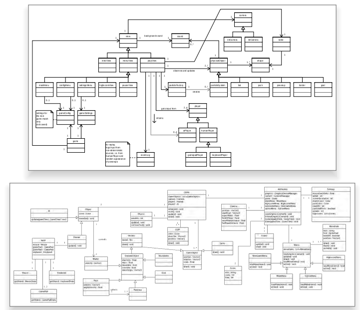
(partly following Ambler (2005))

Some Example Class Diagrams





Some More Example Class Diagrams



So: what makes a class diagram a **good class diagram**?

Main and General Modelling Guideline

Be good to your audience.

"Imagine you're given **your** diagram $\mathcal{D}$ and asked to conduct task $\mathcal{T}$.

- Can you do $\mathcal{T}$ with $\mathcal{D}$?
  (semantics sufficiently clear? all necessary information available? ...)
- Does doing $\mathcal{T}$ with $\mathcal{D}$ cost you more nerves/time/money/... than it should?"
  (syntactical well-formedness? readability? intention of deviations from standard syntax clear? reasonable selection of information? layout? ...)

In other words:

- the things **most relevant** for task $\mathcal{T}$, do they **stand out** in $\mathcal{D}$?
- the things **less relevant** for task $\mathcal{T}$, do they **disturb** in $\mathcal{D}$?

Main and General Quality Criterion

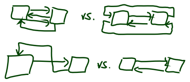
- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

Examples for purposes and points and rules-of-thumb:

- **Analysis/Design**
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer
- **Implementation/A**
  - close to target platform
    ($C_{0,1}$ is easy for Java, C_* comes at a cost — other way round for RDB)
- **Implementation/B**
  - complete, executable
- **Documentation**
  - Right level of abstraction: "if you've only one diagram to spend, illustrate the concepts, the architecture, the difficult part"
  - The more detailed the documentation, the higher the probability for regression
    "outdated/wrong documentation is worse than none"

General Diagramming Guidelines Ambler (2005)

(Note: "Exceptions prove the rule.")



- **2.1 Readability**
  - 1.–3. Support Readability of Lines

⋮

References

References

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

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.