Software Design, Modelling and Analysis in UML

Lecture 7: Class Diagrams II

2016-11-17

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

Albert-Ludwigs-Universität Freiburg, Germany

Content

Rhapsody Demo I: Class Diagrams

RECALL: SEND US YOUR POOL-ACCOUNT NAME (wegerp, NOT: mp124 (RZ))

Class Diagram Semantics Cont'd

Semantical Relevance

- The **semantics** (or meaning) of an extended object system signature $\mathscr{S}$ wrt. a structure $\mathscr{D}$ is **the set of system states** $\Sigma_{\mathscr{S}}^{\mathscr{D}}$.
- The **semantics** (or meaning) of an extended object system signature $\mathscr{S}$ is **the set of sets of system states** wrt. some structure of $\mathscr{S}$, i.e. the set

$$\{\Sigma_{\mathscr{S}}^{\mathscr{D}} \mid \mathscr{D} \text{ is structure of } \mathscr{S}\}.$$

$\mathscr{S}_1 : \langle C, \emptyset, 0, 0 \rangle \rightsquigarrow \Sigma_{\mathscr{S}_1}^{\mathscr{D}} =$
$\mathscr{S}_2 : \langle C, \emptyset, 0, 1 \rangle \rightsquigarrow \Sigma_{\mathscr{S}_2}^{\mathscr{D}}$
(only difference in $\mathscr{S}_1, \mathscr{S}_2$ is activeness of C)

Which of the following aspects is **semantically relevant**, i.e. **does contribute** to the constitution of system states?

A **class**

- has a set of **stereotypes**, ✗
- has a **name**, ✓
- ~~belongs to a **package**,~~
- can be **abstract**, ✓
- can be **active**, ✗
- has a set of **attributes**, ✓
- has a set of **operations** (later).

Each **attribute** has

- a **visibility**, ✗
- a **name**, a **type**, ✓
- a **multiplicity**, ✓ an **order**, ✗
- an **initial value**, and ✗
- a set of **properties**, (✗) such as **readOnly**, **ordered**, etc.

14/31

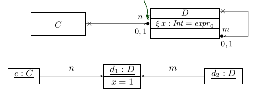
What About The Rest?

- **Classes**:
  - **Stereotypes**: Lecture 6
  - **Active**: not represented in σ.
    Later: relevant for behaviour, i.e., how system states evolve over time.
- **Attributes**:
  - **Initial value expression**: not represented in σ.
    Later: provides an initial value as effect of "creation action".
  - **Visibility**: not represented in σ.
    Later: viewed as additional **typing information** for well-formedness (- typedness) of OCL expressions and actions.
  - **Properties**: such as `readOnly`, `ordered`, `composite` (**Deprecated** in the standard.)
    - `readOnly` – can be treated **similar to visibility**.
    - `ordered` – not considered in our UML fragment ($\rightarrow$ sets vs. sequences).
    - `composite` – cf. lecture on associations.

Visibility

The Intuition by Example

$\mathscr{S} = (\{Int\}, \{C, D\}, \{n : D_{0,1}, m : D_{0,1},$
$\langle x : Int, \xi, expr_0, \emptyset\rangle\},$
$\{C \mapsto \{n\}, D \mapsto \{x, m\}\}$



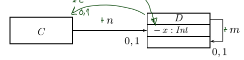
Which of the following two **syntactically correct** (?) OCL expressions **should** we consider to be **well-typed**?

	ξ = public	ξ = private	ξ = protected	ξ = package
$self_C \,.\, n \,.\, x = 0$	✔ / ✘ / ?	✔ / ✘ / ? (by class (C++, Java, …))	later	not in lect.
$self_D \,.\, m \,.\, x = 0$	✔ / ✘ / ?	✔ / ✘ / ? (by object)	later	not in lect.

Context

$\mathscr{S} = (\{Int\}, \{C, D\}, \{n : D_{0,1}, m : D_{0,1},$
$\langle x : Int, \xi, expr_0, \emptyset\rangle\},$
$\{C \mapsto \{n\}, D \mapsto \{x, m\}\}$

- **By example**:



$self_D \,.\, x > 0$ ✓

$self_D \,.\, m \,.\, x > 0$ ✓

$self_C \,.\, n \,.\, x > 0$ ✗

- That is, whether an expression involving attributes with visibility is well-typed **depends** on the class of the object which "tries to read out the value".
- Visibility is '**by class**' – **not** 'by object'.

Attribute Access in Context

Recall: attribute access in OCL Expressions, $C, D \in \mathscr{C}$.

- $v(expr_1) \quad : \tau_C \to \tau(v)$ — $v : T \in atr(C), T \in \mathscr{T}$,
- $r_1(expr_1) \quad : \tau_C \to \tau_D$ — $r_1 : D_{0,1} \in atr(C)$,
- $r_2(expr_1) \quad : \tau_C \to Set(\tau_D)$ — $r_2 : D_* \in atr(C)$,

New rules for well-typedness **considering visibility**:

- $v(w) \quad : \tau_C \to T \qquad w : \tau_C, \quad v : T \in atr(C), T \in \mathscr{T}$
- $r_1(w) \quad : \tau_C \to \tau_D \qquad w : \tau_C, \quad r_1 : D_{0,1} \in atr(C)$
- $r_2(w) \quad : \tau_C \to Set(\tau_D) \qquad w : \tau_C, \quad r_1 : D_* \in atr(C)$

- $v(expr_1(w)) \quad : \tau_C \to T \qquad \langle v : T, \xi, expr_0, P\rangle \in atr(C), T \in \mathscr{T}$, $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C$, or $\xi = +$
- $r_1(expr_1(w)) \quad : \tau_C \to \tau_D \qquad \langle r_1 : D_{0,1}, \xi, expr_0, P\rangle \in atr(C)$, $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C$, or $\xi = +$
- $r_2(expr_1(w)) \quad : \tau_C \to Set(\tau_D) \qquad \langle r_2 : D_*, \xi, expr_0, P\rangle \in atr(C)$, $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C$, or $\xi = +$

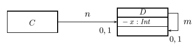
Example

(i) $v(w) \quad : \tau_C \to T \qquad w : \tau_C, \quad v : T \in atr(C), T \in \mathscr{T}$

(ii) $r_1(w) \quad : \tau_C \to \tau_D \qquad w : \tau_C, \quad r_1 : D_{0,1} \in atr(C)$

(iii) $v(expr_1(w)) \quad : \tau_C \to T \qquad \langle v : T, \xi, expr_0, P\rangle \in atr(C), T \in \mathscr{T}$, $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C$, or $\xi = +$

(iv) $r_1(expr_1(w)) \quad : \tau_C \to \tau_D \qquad \langle r_1 : D_{0,1}, \xi, expr_0, P\rangle \in atr(C)$, $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C$, or $\xi = +$



- $self_D \,.\, x > 0$ ⇝ $x(self_D) > 0$ OK, by (i)
- $self_D \,.\, m \,.\, x > 0$ ⇝ $x(m(self_D)) > 0$ OK, by (iii)
- $self_C \,.\, n \,.\, x > 0$ ⇝ $x(n(self_C)) > 0$ NOT OK

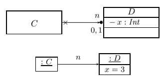
The Semantics of Visibility

- **Observation**:
  - Whether an expression **does** or **does not** respect visibility is **a matter of well-typedness only**.
  - We only evaluate (= apply I to) **well-typed** expressions.

→ We **need not** adjust the interpretation function I to support visibility.

Just decide: should we take visibility into account yes / no, and check well-typedness by the new / old rules.

What is Visibility Good For?



- Visibility is a property of attributes – is it useful to consider it in OCL?
- In other words: given the diagram above, **is it useful** to state the following invariant (even though x is private in D)

context C inv : $n.x > 0$?

It depends. (cf. OMG (2006), Sect. 12 and 9.2.2)

- **Constraints and pre/post conditions**:
  - Visibility is **sometimes not** taken into account. To state "global" requirements, it may be adequate to have a "global view", i.e. be able to "look into" all objects.
  - But: visibility supports "narrow interfaces", "information hiding", and similar **good design practices**. To be more robust against changes, try to state requirements only in the terms which are visible to a class.

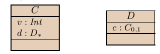
  Rule-of-thumb: if attributes are important to state requirements on design models, leave them public or provide get-methods (later).
- **Guards** and **operation bodies**:
  - If in doubt, **yes** (= do take visibility into account).

  Any so-called **action language** typically takes visibility into account.

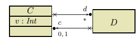
Associations

Overview

- **Class diagram**:



Alternative presentation:



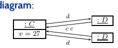
- **Signature**:

$\mathscr{S} = (\{Int\}, \{C, D\}, \{v : Int, d : D_*, c : C_{0,1}\}, \{C \mapsto \{v, d\}, D \mapsto \{c\}\})$

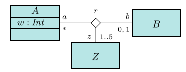
- **Example system state**:

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

- **Object diagram**:

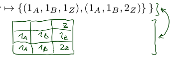


- **Class diagram** (with ternary association):



- **Signature**: extend again to represent
  - **association** r with
    - **association ends** a, b, and z (each with multiplicity, visibility, etc.)
- **Example system state**: (σ, λ)

$\sigma = \{1_A \mapsto \{w \mapsto 13\}, 1_B \mapsto \emptyset, 1_Z \mapsto \emptyset\}$
$\lambda = \{ r \mapsto \{(1_A, 1_B, 1_Z), (1_A, 1_B, 2_Z)\} \}$



- **Object diagram**: No...

Plan

(i) Study association **syntax**.

(ii) Extend **signature** accordingly.

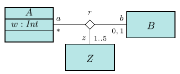
(iii) Define (σ, λ) **system states** with
- **objects** in σ (instances of classes),
- **links** in λ (instances of associations).

(iv) Change **syntax** of OCL to refer to **association ends**.

(v) Adjust **interpretation** I accordingly.

(vi) ... go back to the special case of $C_{0,1}$ and C_* attributes.

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



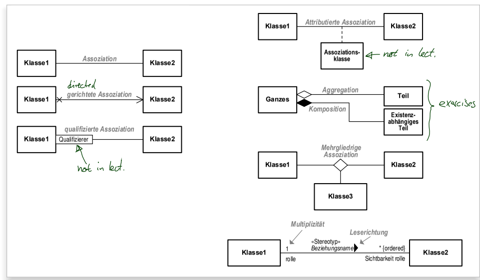
- **Signature**: extend again to represent
  - **association** r with
    - **association ends** a, b, and z (each with multiplicity, visibility, etc.)
- **Example system state**:

$\sigma = \{1_A \mapsto \{w \mapsto 13\}, 1_B \mapsto \emptyset, 1_Z \mapsto \emptyset\}$
$\lambda = \{ r \mapsto \{(1_A, 1_B, 1_Z), (1_A, 1_B, 2_Z)\} \}$

- **Object diagram**: No...

Associations: Syntax

UML Association Syntax Oesterreich (2006)



More Association Syntax (OMG, 2011b, 61;43)

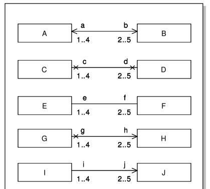


Figure 7.23 - Examples of navigable ends

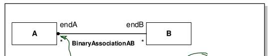


Figure 7.19 - Graphic notation indicating exactly one association end owned by the association

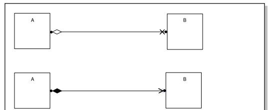


Figure 7.20 - Combining line path graphics

So, What Do We (Have to) Cover?

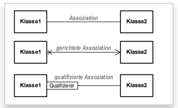
An **association** has

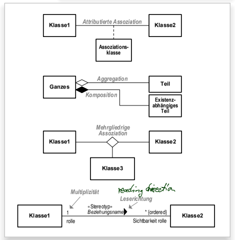
- a **name**,
- a **reading direction**, and
- at least two **ends**.

Each **end** has

- a **role name**,
- a **multiplicity**,
- a set of **properties**, such as **unique**, **ordered**, etc.
- a **qualifier**, (not in lect.)
- a **visibility**,
- a **navigability**,
- an **ownership**,
- and possibly a **diamond**.

Wanted: places in the signature to represent the information from the picture.





(Temporarily) Extend Signature: Associations

Only for the course of Lectures 7 – 9 we assume that each element in V is

- **either** a **basic type attribute** $\langle v : T, \xi, expr_0, P_v \rangle$ with $T \in \mathscr{T}$ (as before),
- **or** an **association** of the form

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

- $n \geq 2$ (at least two ends),
- $r, role_i$ are just **names**, $\quad C_i \in \mathscr{C}, 1 \leq i \leq n$,
- the **multiplicity** μ_i is an expression of the form

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

0..1 ✓
10..27, 80..81 ✓
3..3 ✓
0..* ✓

- P_i is a set of **properties** (as before),
- $\xi \in \{+, -, \#, \sim\}$ (as before),
- $\nu_i \in \{\times, -, >\}$ is the **navigability**,
- $o_i \in \mathbb{B}$ is the **ownership**.

- N for $N..N$,
- $*$ for $0..*$ (use with care!)

(Temporarily) Extend Signature: Associations

Only for the course of Lectures 7 – 9 we assume that each element in V is

- **either** a **basic type attribute** $\langle v : T, \xi, expr_0, P_v \rangle$ with $T \in \mathscr{T}$ (as before),
- **or** an **association** of the form

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

- $n \geq 2$ (at least two ends),
- $r, role_i$ are just **names**, $\quad C_i \in \mathscr{C}, 1 \leq i \leq n$,
- the **multiplicity** μ_i is an expression of the form

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

- P_i is a set of **properties** (as before),
- $\xi \in \{+, -, \#, \sim\}$ (as before),
- $\nu_i \in \{\times, -, >\}$ is the **navigability**,
- $o_i \in \mathbb{B}$ is the **ownership**.

Multiplicity abbreviations:

- N for $N..N$, e.g. 3 for 3..3
- $*$ for $0..*$ (use with care!)

Temporarily (Lecture 7 – 9) Extended Signature

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

- ...
- each element of V is
  - **either** a **basic type attribute** $\langle v : T, \xi, expr_0, P_v \rangle$ with $T \in \mathscr{T}$
  - **or** an **association** of the form

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

(ends with multiplicity μ_i, properties P_i, visibility ξ_i, navigability ν_i, ownership o_i, $1 \leq i \leq n$)

- ...
- $atr : \mathscr{C} \to 2^{\{v \in V \mid v:T, T \in \mathscr{T}\}}$ 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"**.

Tell Them What You've Told Them...

- Class Diagrams in the **Rhapsody** Tool
- **Visibility** of attributes contributes to the well-typedness of (among others) OCL expressions.
  - Well-typedness depends on the **context**.
  - We only interpret (= apply I to) **well-typed** OCL constraints.
  - Sometimes we **consider** visibility, sometimes we don't.
- **Associations** can have any number (≥ 2) of **Association Ends**.

References

References

Oestereich, B. (2006). *Analyse und Design mit UML 2.1, 8. Auflage*. Oldenbourg, 8. edition.

OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

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.