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
- Visibility
  - Intuition
  - Context, OCL with Visibility
  - What is Visibility Good For?
- Associations
  - Overview & Plan
  - (Temporarily) Extend Signature
  - From Diagrams to Signatures
    - What if Things are Missing?
RECALL: SEND US YOUR POOL-ACCOUNT NAME

\( \text{unyield, NOT: mp124 (R2)} \)
Class Diagram Semantics Cont’d
Semantical Relevance

- The **semantics** (or meaning) of an extended object system signature $S$ wrt. a structure $D$ is the set of system states $\Sigma_D$.

- The **semantics** (or meaning) of an extended object system signature $S$ is the set of sets of system states wrt. some structure of $S$, i.e. the set

$$\{ \Sigma_D \mid D \text{ is structure of } S \}.$$

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**, \(\times\)
- has a **name**, \(\checkmark\)
- belongs to a **package**,
- can be **abstract**, \(\checkmark\)
- can be **active**, \(\times\)
- has a set of **attributes**, \(\checkmark\)
- has a set of **operations** (later).

Each **attribute** has
- a **visibility**, \(\times\)
- a **name**, a **type**, \(\checkmark\)
- a **multiplicity**, an **order**, \(\times\)
- an **initial value**, and \(\times\)
- a set of **properties**, \(\times\) such as **readOnly**, **ordered**, etc.
What About The Rest?

- **Classes:**
  - **Stereotypes:** Lecture 6
  - **Active:** not represented in $\sigma$.
    - *Later:* relevant for behaviour, i.e., how system states evolve over time.

- **Attributes:**
  - **Initial value expression:** not represented in $\sigma$.
    - *Later:* provides an initial value as effect of "creation action".
  - **Visibility:** not represented in $\sigma$.
    - *Later:* viewed as additional *typing information* for well-formedness 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 (→ sets vs. sequences).
    - `composite` – cf. lecture on associations.
Visibility
Which of the following two syntactically correct (?) OCL expressions should we consider to be well-typed?

<table>
<thead>
<tr>
<th></th>
<th>public</th>
<th>private</th>
<th>protected</th>
<th>package</th>
</tr>
</thead>
<tbody>
<tr>
<td>(self_C \cdot n \cdot x = 0)</td>
<td>✔</td>
<td>✗</td>
<td>later</td>
<td>by class (C++, Java, ..)</td>
</tr>
<tr>
<td>(self_D \cdot m \cdot x = 0)</td>
<td>✔</td>
<td>✗</td>
<td>later</td>
<td>by object</td>
</tr>
</tbody>
</table>
Context

- By example:

\[ \mathcal{S} = (\{ \text{Int} \}, \{ C, D \}, \{ n : D_{0,1}, m : D_{0,1}, \langle x : \text{Int}, \xi, \text{expr}_0, \emptyset \rangle \}, \{ C \mapsto \{ n \}, D \mapsto \{ x, m \} \} ) \]

\[ C \xrightarrow{+n} D \]

\[ self_D \cdot x > 0 \quad \checkmark \]

\[ self_D \cdot m \cdot x > 0 \quad \checkmark \]

\[ self_C \cdot n \cdot x > 0 \quad \times \]

- 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’.
**Recall**: attribute access in OCL Expressions, $C, D \in \mathcal{C}$.

- $v(expr_1) : \tau_C \rightarrow \tau(v)$
- $r_1(expr_1) : \tau_C \rightarrow \tau_D$
- $r_2(expr_1) : \tau_C \rightarrow Set(\tau_D)$

- $v : T \in atr(C), T \in \mathcal{T}$,
- $r_1 : D_{0,1} \in atr(C)$,
- $r_2 : D_* \in atr(C)$,

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

- $v(w) : \tau_C \rightarrow T$
- $r_1(w) : \tau_C \rightarrow \tau_D$
- $r_2(w) : \tau_C \rightarrow Set(\tau_D)$

- $v(expr_1(w)) : \tau_C \rightarrow T$
- $r_1(expr_1(w)) : \tau_C \rightarrow \tau_D$
- $r_2(expr_1(w)) : \tau_C \rightarrow Set(\tau_D)$

- $w : \tau_C, v : T \in atr(C), T \in \mathcal{T}$
- $w : \tau_C, r_1 : D_{0,1} \in atr(C)$
- $w : \tau_C, r_2 : D_* \in atr(C)$

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

(i) \( v(w) : \tau_C \rightarrow T \)  \( w : \tau_C, \ v : T \in \text{atr}(C), T \in \mathcal{T} \)

(ii) \( r_1(w) : \tau_C \rightarrow \tau_D \)  \( w : \tau_C, \ r_1 : D_{0,1} \in \text{atr}(C) \)

(iii) \[ v(expr_1(w)) : \tau_C \rightarrow T \]

\[ \forall (\omega_1 \cdots (\omega_n (w)) \cdots) \]

\( expr_1(w) : \tau_C, \ w : \tau_{C_1} \text{ and } C_1 = C, \text{ or } \xi = + \)

(iv) \[ r_1(expr_1(w)) : \tau_C \rightarrow \tau_D \]

\[ \langle r_1 : D_{0,1}, \xi, expr_0, P \rangle \in \text{atr}(C), \]

\( expr_1(w) : \tau_C, \ w : \tau_{C_1} \text{ and } C_1 = C, \text{ or } \xi = + \)

- \( self_D \cdot x > 0 \text{ OK, by (i)} \)

- \( self_D \cdot m \cdot x > 0 \text{ OK, by (iii)} \)

- \( self_C \cdot n \cdot x > 0 \text{ NOT OK} \)
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 \( D \))

\[
\text{context } C \ 	ext{inv : } n.x > 0 ?
\]

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:

  \[ C \]
  \[ v : \text{Int} \]
  \[ d : D^\ast \]

  \[ D \]
  \[ c : C_{0,1} \]

  Alternative presentation:

  \[ C \]
  \[ v : \text{Int} \]
  \[ d \]
  \[ c \]
  \[ 0,1 \]

  \[ D \]

- Signature:

  \[ \mathcal{S} = (\{\text{Int}\}, \{C, D\}, \{v : \text{Int}, d : D^\ast, c : C_{0,1}\}, \{C \mapsto \{v, d\}, D \mapsto \{c\}\}) \]

- 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:

  \[ C \]
  \[ v = 27 \]
  \[ d \]
  \[ c \]
  \[ D \]

- Class diagram (with ternary association):

  \[ A \]
  \[ w : \text{Int} \]
  \[ a \]
  \[ r \]
  \[ b \]

  \[ Z \]
  \[ z \]
  \[ 1..5 \]
  \[ 0,1 \]

  \[ D \]

  \[ C \]

- Signature: extend again to represent

  - association \( r \) with

    - association ends \( a, b, \) and \( z \)

      (each with multiplicity, visibility, etc.)

- Example system state:

  \( (\sigma, \lambda) \)

  \[ \sigma = \{1_A \mapsto \{w \mapsto 13\}, 1_B \mapsto \emptyset, 1_Z \mapsto \emptyset\} \]

  \[ \lambda = \{r \mapsto \{(1_A, 1_B, 1_Z), (1_A, 1_B, 2_Z)\}\} \]

- Object diagram:

  \[ \vdash \]

  \[ \tau \]
  \[ \gamma \]
  \[ \alpha \]
  \[ \beta \]
  \[ \delta \]
  \[ \gamma \]
  \[ \alpha \]
  \[ \beta \]
  \[ \delta \]

- Object diagram: No…
Plan

(i) Study association syntax.

(ii) Extend signature accordingly.

(iii) Define \((\sigma, \lambda)\) system states with

- **objects** in \(\sigma\)
  (instances of classes),
- **links** in \(\lambda\)
  (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
More Association Syntax (OMG, 2011b, 61;43)

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

Figure 7.20 - Combining line path graphics

Figure 7.23 - Examples of navigable ends
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 $T \in \mathcal{T}$ (as before),
- or an association of the form

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

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

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

- $P_i$ 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.

- $N$ for $N..N$,
- $*$ for $0..*$ (use with care!)
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 \( T \in \mathcal{T} \) (as before),
- or an **association** of the form

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

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

\[
\mu ::= N..M | N..* | \mu, \mu \quad (N, M \in \mathbb{N})
\]

- \( P_i \) 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!)
Definition. An (Extended) Object System Signature (with Associations) is a quadruple \( \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \) where

- ... 

- each element of \( V \) is
  - either a basic type attribute \( \langle v : T, \xi, \text{expr}_0, P_v \rangle \) with \( T \in \mathcal{T} \)
  - or an association of the form
    \[
    \langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \\
    \vdots \\
    \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle 
    \]
    (ends with multiplicity \( \mu_i \), properties \( P_i \), visibility \( \xi_i \), navigability \( \nu_i \), ownership \( o_i \), \( 1 \leq i \leq n \))

- ...

- \( \text{atr} : \mathcal{C} \rightarrow 2\{v \in V \mid v : T, T \in \mathcal{T} \} \) maps classes to basic type (!) attributes.

In other words:

- only basic type attributes “belong” to a class (may appear in \( \text{atr}(C) \)),
- associations are not “owned” by a class (not in any \( \text{atr}(C) \)), 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 ($\geq 2$) of Association Ends.
References
References


