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?
Rhapsody Demo I: Class Diagrams

RECALL: SEND US YOUR POOL-ACCOUNT NAME

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 \( S_D \).
- 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 \( \{ S_D \mid D \text{ is structure of } S \} \).

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

- Each attribute has
  - a visibility, \( \surd \)
  - a name, a type, \( \surd \)
  - a multiplicity, an order, \( \surd \)
  - an initial value, and \( \surd \)
  - a set of properties, \( \{ \surd \} \)
    - 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

The Intuition by Example

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

<table>
<thead>
<tr>
<th>self_C.n.x = 0</th>
<th>self_D.m.x = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \xi = \text{public} )</td>
<td>( \xi = \text{private} )</td>
</tr>
<tr>
<td>✔ later</td>
<td>✔ later</td>
</tr>
<tr>
<td>✔ later</td>
<td>✔ later</td>
</tr>
</tbody>
</table>

\( \xi = \text{protected} \)

by class (C++, Java, ...)

\( \xi = \text{package} \)

by object
Context

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

- By example:

\[ D \rightarrow \text{Int} \]

\[ S = \{\} \]

\[ n \in D_{0,1}, m \in D_{0,1}, x \in \text{Int} \]

\[ \langle x \rangle = \text{expr}_0, \emptyset \]

\[ \{C \mapsto \{n\}, D \mapsto \{x, m\}\} \]

- 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, \( C, D \in \mathcal{S} \).

- \( v(\text{expr}_1) : \tau_C \rightarrow \tau(v) \)
- \( r_1(\text{expr}_1) : \tau_C \rightarrow \tau_D \)
- \( r_2(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) \)

New rules for well-typedness considering visibility:

- \( v(w) : \tau_C \rightarrow T \) \( w : \tau_C, v : T \in \text{atr}(C), T \in \mathcal{S} \)
- \( r_1(w) : \tau_C \rightarrow \tau_D \) \( w : \tau_C, r_1 : D_{0,1} \in \text{atr}(C) \)
- \( r_2(w) : \tau_C \rightarrow \text{Set}(\tau_D) \) \( w : \tau_C, r_2 : D_e \in \text{atr}(C) \)

\[ \langle T, \xi, \text{expr}_0, P \rangle \in \text{atr}(C), T \in \mathcal{S} \]

\[ \text{expr}_1(w) : \tau_C \]

\[ \{r_1 : D_{0,1}, \xi, \text{expr}_0, P \} \in \text{atr}(C), \]

\[ \text{expr}_1(w) : \tau_C \]

\[ \{r_2 : D_e, \xi, \text{expr}_0, P \} \in \text{atr}(C), \]

\[ w : \tau_C \]

\[ \text{and} \quad C_1 = C, \quad \text{or} \quad \xi = + \]

\[ \{r_1 : D_{0,1}, \xi, \text{expr}_0, P \} \in \text{atr}(C), \]

\[ \text{expr}_1(w) : \tau_C \]

\[ \{r_2 : D_e, \xi, \text{expr}_0, P \} \in \text{atr}(C), \]

\[ w : \tau_C \]

\[ \text{and} \quad C_1 = C, \quad \text{or} \quad \xi = + \]
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.

- **Example**

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

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

  (iii) $v(\text{expr}_1(w)) : \tau_C \rightarrow T$
  \begin{align*}
  &\begin{cases}
  T(\text{expr}_1(w)) : \tau_C, \\
  r_1 : D_{0,1}, \xi, \text{expr}_0, P_l \in \text{atr}(C), T \in \mathcal{F},
  \end{cases}
  \end{align*}

  (iv) $r_1(\text{expr}_1(w)) : \tau_C \rightarrow \tau_D$
  \begin{align*}
  &\begin{cases}
  T(\text{expr}_1(w)) : \tau_C, \\
  w : \tau_C, and C_1 = C, \ or \ \xi = +
  \end{cases}
  \end{align*}

- **self**

  $\text{D}.x > 0 \Rightarrow x(\text{self}_D) > 0$  
  OK, by (i)

  $\text{D}.m.x > 0 \Rightarrow x(\text{self}_D) > 0$  
  OK, by (iii)

  $\text{C}.n.x > 0 \Rightarrow x(\text{self}_C) > 0$  
  NOT OK
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 $D$)

$$\text{context } C \text{ inv : } n.x > 0 ?$$

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  D
 v: Int  w: Int
 d: D  c: C
```

Alternative presentation:

```
 C  D
 v: Int  w: Int
d: D  c: C
```

- Signature:

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

- 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  D
 v: D  w: Int
d: D  c: C
```

- Class diagram (with ternary association):

```
 A  B
 w: Int  z: \mathbb{N}
 r: C
```

- 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...

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.
 Associations: Syntax

UML Association Syntax Oestreich (2006)
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 java)
- 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 $(v : T, \xi, \text{expr}_0, P_v)$ with $T \in \mathcal{T}$ (as before),
- or an association of the form

$$\langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- $n \geq 2$ (at least two ends),
- $r, \text{role}_i$ 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.

Multiplicity abbreviations:

- $N$ for $N..N$
- $*$ for $0..*$ (use with care!)
**Definition.** An (Extended) Object System Signature (with Associations) is a quadruple \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, V, \text{atr}) \) where

- ... 
- each element of \( V \) is
  - either a basic type attribute \( \langle v : T, \xi, \text{expr}_0, P_v \rangle \) with \( T \in \mathcal{F} \)
  - or an association of the form
    \( \langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle \)
    (ends with multiplicity \( \mu_i \), properties \( P_i \), visibility \( \xi_i \), navigability \( \nu_i \), ownership \( o_i \), \( 1 \leq i \leq n \))
- ...
- \( \text{atr} : \mathcal{C} \rightarrow 2^\{v \in V \mid v : T, T \in \mathcal{F} \} \) 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


