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
Recapitulation: Consider the following association:

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

- **Association name** \( r \) and **role names/types** \( \text{role}_i / C_i \) induce extended system states \( \lambda \).
- **Multiplicity** \( \mu \) is considered in OCL syntax.
- **Visibility** \( \xi / \text{Navigability} \) \( \nu \): well-typedness.

Now the rest:

- **Multiplicity** \( \mu \): 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

![Diagram](image)

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

\[
\text{context } C \ \text{inv} : self.role.x > 0 \quad \text{well-typed anyway} \\
\text{context } D \ \text{inv} : self.role_2.role.x > 0 \quad \text{not well-typed} \]
Visibility

Not so surprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by **typing rules**.

**Question**: given

\[
\begin{array}{ccc}
C & \xrightarrow{1} & D \\
\xi \; role & & x : \text{Int}
\end{array}
\]

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

context \( C \) inv : self.role.x > 0

\[
x(\text{role}(\text{self}))
\]

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

\[
(Assoc_1) \quad \frac{A, D \vdash expr_1 : \tau_C}{A, D \vdash role(expr_1) : \tau_D}, \quad \mu = 0..1 \text{ or } \mu = 1,
\]

\[
\xi = +, \text{ or } \xi = - \text{ and } C = D
\]

\[
\langle r : \ldots \langle \text{role} : D, \mu, -, \xi, -, - \rangle, \ldots \langle \text{role}' : C, -, -, -, -, - \rangle, \ldots \rangle \in V
\]
Navigability

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

Question: given

\[
\begin{array}{c}
C \\
x : Int
\end{array}
\quad \xrightarrow{0..1} \quad
\begin{array}{c}
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad
\end{array}
\quad
\begin{array}{c}
D
\end{array}
\]

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

context $D$ inv : self.role.x > 0

\[\text{not well-typed}\]
Navigability is similar to visibility: expressions over non-navigable association ends \((\nu = \times)\) are **basically** type-correct, but **forbidden**.

**Question:** given

![Diagram of UML association with roles and navigability](image)

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

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

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.
\\[ CD \] makes no sense ...?

- in general there is no OCL expression involving \( x \) vs which is well-typed
- for requirements, we may disregard well-typedness and write instead \( C' \text{ inv: self.s.x > 0} \) (artificial example)

so, difference between \( \_ \) and \( 'x' \) and \( '>' \) and \( 'x' \)
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 "effective" means and tell it to the programmers
Recapitulation: Consider the following association:

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

- **Association name** \( r \) and **role names/types** \( \text{role}_i/C_i \) induce extended system states \( \lambda \).
- **Multiplicity** \( \mu \) is considered in OCL syntax.
- **Visibility** \( \xi \) and **Navigability** \( \nu \): well-typedness.

Now the rest:

- **Multiplicity** \( \mu \): we propose to view them as constraints.
- **Properties** \( P_i \): even more typing.
- **Ownership** \( o \): getting closer to pointers/references.
- **Diamonds**: exercise.
**Recall:** The multiplicity of an association end is a term of the form:

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

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

\[
\text{Context } C \text{ inv: } 4 \leq \text{size}(u) > 27 \text{ or size}(u) = 31
\]
Multiplicities as Constraints

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

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

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

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

Recall: we can normalize each multiplicity $$\mu$$ to the form

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

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

e.g. 31 to 31..31

Observe: e.g. * to 0..*
Multiplicities as Constraints

\[ \mu = N_1 \ldots N_2, \ldots, N_{2k-1} \ldots N_{2k} \]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N}, \ N_{2k} \in \mathbb{N} \cup \{\ast\} \).

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

\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \ 	ext{or} \ldots \ 	ext{or} \ (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \]

omit if \( N_{2k} = \ast \)

for each \( \mu \neq 0..1, \mu \neq 1 \),

\( \langle r : \ldots, \langle \text{role} : D, \mu, \ldots \rangle, \ldots, \langle \text{role}' : C, \ldots \rangle, \ldots \rangle \in V \) or

\( \langle r : \ldots, \langle \text{role}' : C, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots \rangle, \ldots \rangle \in V, \text{role} \neq \text{role}' \).

And define

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

for each \( \mu = 1 \).

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
Recall/Later:

\[ \mathcal{CD} = \{CD_1, \ldots, CD_n\} \]

signature \( \mathcal{S}(\mathcal{CD}) \)

invariants \( \text{Inv}(\mathcal{CD}) \)

From now on: \( \text{Inv}(\mathcal{CD}) = \{ \text{constraints occurring in notes} \} \cup \{ \mu^{\mathcal{C}}_{\text{OCL}}(\text{role}) \mid \)

\[ \langle r : \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots \rangle \in V \text{ or } \]

\[ \langle r : \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots \rangle \in V, \]

\[ \text{role} \neq \text{role}', \mu \notin \{0..1\} \}. \]
### Multiplicities as Constraints Example

**OCL**

\[ \mu_{\text{OCL}}^C(\text{role}) = \text{context } C \text{ inv :} \]

\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \]

\[
\begin{array}{c}
\text{CD :} \\
\begin{array}{c}
\text{role}_1 \\
\text{0..1} \\
\Rightarrow \\
\text{C} \\
v : \text{Int} \\
\Leftarrow \\
\text{role}_2 \\
4, 17 \\
\Leftarrow \\
\text{role}_3 \\
3..* \\
\end{array}
\end{array}
\]

**Inv(CD) =**

- \{\text{context } C \text{ inv : } 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17\} \\
  = \{\text{context } C \text{ inv : } \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17\}
- \cup \{\text{context } C \text{ inv : } 3 \leq \text{role}_3 \rightarrow \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 $\mu_{OCL} = true$ 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...?
Well, if the **target platform** is known and fixed, and the target platform has, for instance,
- reference types,
- range-checked arrays with positions 0, ..., \( N \),
- 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.
We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unique</strong></td>
<td>one object has <strong>at most one</strong> ( r )-link to a single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td><strong>bag</strong></td>
<td>one object may have <strong>multiple</strong> ( r )-links to a single other object</td>
<td>have ( \lambda(r) ) yield multi-sets</td>
</tr>
<tr>
<td><strong>ordered, sequence</strong></td>
<td>an ( r )-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have ( \lambda(r) ) yield sequences</td>
</tr>
</tbody>
</table>

So not 

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\]

but

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C} \\
\text{D}
\end{array}
\]
We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>one object has <strong>at most one</strong> ( r )-link to a single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> ( r )-links to a single other object</td>
<td>have ( \lambda(r) ) yield multi-sets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an ( r )-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have ( \lambda(r) ) yield sequences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression ( \text{role}(expr) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>( \tau_D \rightarrow \text{Set}(\tau_C) )</td>
</tr>
<tr>
<td>bag</td>
<td>( \tau_D \rightarrow \text{Bag}(\tau_C) )</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>( \tau_D \rightarrow \text{Seq}(\tau_C) )</td>
</tr>
</tbody>
</table>

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].
Ownership

![Diagram of ownership concept]

Intuitively it says:

Association \( r \) is **not a “thing on its own”** (i.e. provided by \( \lambda \)),
but association end ‘\( role \)’ is **owned** by \( C \) (!).
(That is, it’s stored inside \( C \) object and provided by \( \sigma \)).

**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:**

![Diagram of not clear to me concept]
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

  
  ![Diagram](image)

  (And we may omit the non-navigability and ownership symbols.)

- Form (i) introduces $\text{role} : C_{0,1}$, and form (ii) introduces $\text{role} : C_*$ in $V$.

- In both cases, $\text{role} \in \text{attr}(C')$.

- We drop $\lambda$ and go back to our nice $\sigma$ with $\sigma(u)(\text{role}) \subseteq \mathcal{P}(D)$. 
OCL Constraints in (Class) Diagrams
Where Shall We Put OCL Constraints?

Numerous options:
(i) Additional documents.
(ii) Notes.
(iii) Particular dedicated places.

Notes:
A UML note is a picture of the form

`text`

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

Sometimes, content is explicitly classified for clarity:
OCL in Notes: Conventions

stands for

context $C$ inv : $expr$
(ii) **Particular dedicated places** in class diagrams:  

\[
C \\
\xi \; v : \tau \{p_1, \ldots, p_n\} \{expr\} \\
\xi \; f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau \{p_1, \ldots, p_n\} \{pre : expr_1 \quad post : expr_2\}
\]

For simplicity, we view the above as an abbreviation for

\[
C \\
\xi \; v : \tau \{p_1, \ldots, p_n\} \\
\text{context } f \quad \text{pre} : expr_1 \quad \text{post} : expr_2
\]
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

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

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

- As usual: $\text{Inv}(\mathcal{D}) := \bigcup_{\mathcal{CD} \in \mathcal{D}} \text{Inv}(\mathcal{CD})$.
- **Principally clear:** $\text{Inv}(\cdot)$ for any kind of diagram.
If \( \mathcal{CD} \) consists of only \( CD \) with the single class \( C \), then

- \( Inv(\mathcal{CD}) = Inv(CD) = \emptyset \)
**Semantics of a Class Diagram**

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

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

$$[[\mathcal{CD}]] := \langle \mathcal{I}(\mathcal{CD}), \text{Inv}(\mathcal{CD}) \rangle.$$ 

Given a structure $\mathcal{D}$ of $\mathcal{I}$ (and thus of $\mathcal{CD}$), the class diagrams describe the system states $\Sigma^\mathcal{D}$. Of those, some satisfy $\text{Inv}(\mathcal{CD})$ and some don’t.

We call a system state $\sigma \in \Sigma^\mathcal{D}$ consistent if and only if $\sigma \models \text{Inv}(\mathcal{CD})$.

**In pictures:**

\[
\mathcal{CD} = \{CD_1, \ldots, CD_n\} \\
\text{signature } \mathcal{I}(\mathcal{CD}) \\
\text{distinguish} \\
\text{basic} \\
\text{(classes and attributes)} \\
\text{induce} \\
(\sigma \in) \Sigma^\mathcal{D} \\
\text{invariants } \text{Inv}(\mathcal{CD}) \\
\text{extended} \\
\text{(visibility)}
\]
**Pragmatics**

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

A set of class diagrams $\mathcal{CD}$ with invariants $\text{Inv}(\mathcal{CD})$ 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 $\text{Inv}(\mathcal{CD})$.

- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $\text{Inv}(\mathcal{CD})$ 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.

```
<table>
<thead>
<tr>
<th>TLCtl</th>
</tr>
</thead>
<tbody>
<tr>
<td>red : Boolean</td>
</tr>
<tr>
<td>green : Boolean</td>
</tr>
</tbody>
</table>
```

$\text{not(red and green)}$
Constraints vs. Types

Find the 10 differences:

<table>
<thead>
<tr>
<th>$C$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x : \text{Int} {x = 3 \lor x &gt; 17}$</td>
<td>$\mathcal{D}(T) = {3} \cup {n \in \mathbb{N} \mid n &gt; 17}$</td>
</tr>
</tbody>
</table>

- $x = 4$ 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 $\mathcal{D}(T)$ (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

