Software Design, Modelling and Analysis in UML

Lecture 09: Class Diagrams III

2013-11-25

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lectures:
- Studied syntax and semantics of associations in 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:
  - Effect of association semantics on OCL.
  - Treat “the rest”.
  - Where do we put OCL constraints?
  - Modelling guidelines, in particular for class diagrams (following [Ambler, 2005])
  - Examples: modelling games (made-up and real-world examples)
Links in System States

\[ \langle r : \langle \text{role}_1 : C_1, P_1, \ldots \rangle, \ldots, \langle \text{role}_n : C_n, P_n, \ldots \rangle \rangle \]

Only for the course of lectures 2008/09 we change the definition of system states:

**Definition.** Let \( \mathcal{D} \) be a structure of the (extended) signature \( \mathcal{S} = (\mathcal{R}, \mathcal{C}, V, \text{atr}) \).

A system state of \( \mathcal{S} \) wrt. \( \mathcal{D} \) is a pair \( (\sigma, \lambda) \) consisting of:

- a type-consistent mapping
  \[ \sigma : \mathcal{D}(\mathcal{C}) \rightarrow (\text{atr}(\mathcal{C}) \rightarrow \mathcal{P}(\mathcal{D})) \],
- a mapping \( \lambda \) which assigns each association
  \[ \langle r : \langle \text{role}_1 : C_1 \rangle, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V \] a relation
  \[ \lambda(r) \subseteq \mathcal{D}(C_1) \times \cdots \times \mathcal{D}(C_n) \]
  (i.e. a set of type-consistent \( n \)-tuples of identities).

**Example**

\[ T = \{ t_1 \rightarrow \{ \text{math}\}, Z_3 \rightarrow \{ \text{math}\}, Z_5 \rightarrow \{ \text{math}\}, Z_7 \rightarrow \{ \text{math}\}, Z_9 \rightarrow \{ \text{math}\} \} \]

\[ \lambda = \{ t_1 \rightarrow \{ (13, 25, 32) \}, (15, 23, 32), (23, 52, 63) \} \]

Students may join multiple groups

\[ \lambda = \{ t_1 \rightarrow \{ (13, 25, 32) \}, (15, 23, 32), (23, 52, 63), (34, 35, 36) \} \]

Lecturers also have dangling references

One student may assume all roles

(Add a constraint if this is not desired)

**Object diagrams:**

\[ \text{We will not formally define that} \]
**Association/Link Example**

\[ C \xrightarrow{n} D \]

**Signature:**
\[ \mathcal{P} = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}\},\]
\[ (A,C,D) : \langle c : C, 0..*, +, \{\text{unique}\}, x, 1 \rangle),\]
\[ \langle n : D, 0..*, +, \{\text{unique}\}, >, 0 \rangle),\]
\[ \{C \mapsto \emptyset, D \mapsto \{x\}\}).\]

A system state of \( \mathcal{P} \) (some reasonable \( \mathcal{P} \)) is \((\sigma, \lambda)\) with:
\[ \sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\}\]
\[ \lambda = \{A,C,D \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}\]

\[ \text{Object } c \text{ is related to } 3_D \text{ and } 7_D \text{ by } A-C-N.\]

**Associations and OCL**
OCL and Associations: Syntax

Recall: OCL syntax as introduced in Lecture 03, interesting part:

\[
\text{expr ::= ... | } r_1(\text{expr}_1) : \tau_C \rightarrow \tau_D \quad r_1 : D_{0,1} \in \text{atr}(C) \\
| r_2(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad r_2 : D_e \in \text{atr}(C)
\]

Now becomes

\[
\text{expr ::= ... | } \text{role}(\text{expr}_1) : \tau_C \rightarrow \tau_D \\
| \text{role}(\text{expr}_1) : \tau_C \rightarrow \text{Set}(\tau_D) \\
\text{if } \langle r : \ldots, (\text{role} : D, \mu, \ldots), \ldots, (\text{role} : C, \ldots) \ldots \rangle \in V \text{ or } \\
\langle r : \ldots, (\text{role} : C, \ldots), \ldots, (\text{role} : D, \mu, \ldots) \ldots \rangle \in V, \text{role} \neq \text{role}'.
\]

Note:
- Association name as such doesn’t occur in OCL syntax, role names do.
- \text{expr}_1 has to denote an object of a class which “participates” in the association.
OCL and Associations Syntax: Example

\[
expr ::= \ldots \mid role(expr_1) : \tau_C \rightarrow \tau_D \quad \mu = 0..1 \text{ or } \mu = 1
\]

\[
\mid role(expr_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad \text{otherwise}
\]

if

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

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

\[
\text{Figure 7.21 - Binary and ternary associations} \quad [\text{OMG, 2007b, 44}].
\]

OCL and Associations: Semantics

Recall: (Lecture 03)

Assume \( expr_1 : \tau_C \) for some \( C \in \mathcal{C} \). Set \( u_1 := I[expr_1](\sigma, \beta) \in \mathcal{P}(\tau_C) \).

\[
\begin{align*}
&I[r_1(expr_1)](\sigma, \beta) := \begin{cases} 
&u, \text{ if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\
&\bot, \text{ otherwise}
\end{cases} \subseteq \mathcal{P}(\tau_D) \\
&I[r_2(expr_1)](\sigma, \beta) := \begin{cases} 
&\sigma(u_1)(r_2), \text{ if } u_1 \in \text{dom}(\sigma) \\
&\bot, \text{ otherwise}
\end{cases}
\end{align*}
\]

Now needed:

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

- We cannot simply write \( \sigma(u)(\text{role}) \).
  Recall: \( \text{role} \) is (for the moment) not an attribute of object \( u \) (not in \( \text{atr}(C) \)).
- What we have is \( \lambda(r) \) (with \( r \), not with \( \text{role}! \)) — but it yields a set of \( n \)-tuples, of which some relate \( u \) and other some instances of \( D \).
- \( \text{role} \) 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 $C \in \mathcal{C}$. Set $u_1 := I[expr_1]((\sigma, \lambda), \beta) \in \mathcal{P}(\tau_C)$.

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

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

where

\[
L(\text{role})(u, \lambda) = \{(u_1, \ldots, u_n) \in \lambda(r) \mid u \in \{u_1, \ldots, u_n\}\} \downarrow i
\]

if

\[
\langle r : \ldots \langle \text{role}_1 : \ldots \rangle, \ldots \langle \text{role}_n : \ldots \rangle, \ldots \rangle, \text{role} = \text{role}_i.
\]

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

### OCL and Associations Example

\[
\begin{align*}
I[role(expr_1)]((\sigma, \lambda), \beta) &:= \begin{cases} L(\text{role})(u_1, \lambda) \text{, if } u_1 \in \text{dom}(\sigma) \\ \bot \text{, otherwise} \end{cases} \\
L(\text{role})(u, \lambda) &:= \{(u_1, \ldots, u_n) \in \lambda(r) \mid u \in \{u_1, \ldots, u_n\}\} \downarrow i
\end{align*}
\]

\[
\sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\} \cup \{2_C \mapsto \emptyset\}
\]

\[
\lambda = \{\{x \mapsto 1\} \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\} \cup \{2_C \mapsto \emptyset\}
\]

\[
I[\text{self} \cdot n]((\sigma, \lambda), \{\text{self} \mapsto 1_C\}) = \{3_D, 7_D\} \cup \{L(\text{role})(1_C, \lambda)\}
\]

\[
= \{\{(1_C, 3_D), (1_C, 7_D)\}\} \cup \{3_D, 7_D\}
\]

\[
\{3_D, 7_D\} \downarrow i
\]
\[ \lambda \cdot \overrightarrow{\epsilon} \mapsto \{ (a_1, b_1, c), (a_2, b_2, c), (a_3, b_3, c), (a_4, b_4, c) \} \]

\[ L (\epsilon) (a_1) = \{ (a_1, b_1, c), (a_4, b_4, c) \} \downarrow = \Phi_b \cup \Phi_d \]

\[ L (\epsilon) (a_2) = \Phi_b \cup \Phi_d \]

\[ L (\epsilon) (a_3) = \Phi_b \cup \Phi_d \]

**Associations: The Rest**
Visibility

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

**Question:** given

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

\[\xi \text{ role} \]

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

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

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

\[
(Assoc_1) \quad A, B \vdash expr_1 : \tau_C, \mu = 0..1 \text{ or } \mu = 1, \xi = +, \text{ or } \xi = - \text{ and } C = B
\]

\[
(r : \ldots (\text{role} : D, \mu, \xi, \ldots), \ldots (\text{role} : C, \ldots, \ldots) ) \in V
\]

Navigability

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

**Question:** given

\[
\begin{array}{c}
\text{C} \\
\text{x: Int}
\end{array} \quad \begin{array}{c}
\text{D} \\
\text{x: Int}
\end{array}
\]

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

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

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.
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
- \( \text{role}_i \) induce extended system states \( \lambda \).
- Multiplicity \( \mu \) is considered in OCL syntax.
- Visibility \( \xi \) and navigability \( \nu \) give rise to well-typedness rules.

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.

Multiplicities as Constraints

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

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

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

Recall: we can normalize each multiplicity to the form

\[ \mu \leq N_1N_2\ldotsN_{2k-1}N_{2k} \]

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

Define

\[ \mu_{\text{OCL}} = \text{context } C \text{ inv: } \]

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

for each

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

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

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
**Multiplicities as Constraints of Class Diagram**

Recall:

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

signature \( \mathcal{I}(\mathcal{D}) \)

\[ \text{invariants } \text{inv}(\mathcal{D}) \]

basic (classes and attributes)

distinguish

extended (visibility)

From now on:

\[ \text{inv}(\mathcal{D}) = \{ \text{constraints occurring in notes} \} \cup \{ \mu_{\text{OCL}} \} \]

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

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

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

\[ \mu_{OCL} = \text{context } C \text{ inv :} \]
\[ (N_1 \leq \text{role} \rightarrow \text{size}()) \leq N_2) \text{ and } \ldots \text{ and } (N_{2k-1} \leq \text{role} \rightarrow \text{size}()) \leq N_{2k}) \]

\[ CD : \]

\[
\begin{array}{c|c|c}
\text{role}_1 & C & 0.1 \\
\hline
\text{role}_2 & v : Int & 4.17 \\
\hline
\text{role}_3 & 3..* \\
\end{array}
\]

\[ \text{Inv}(CD) = \]
\[ \{ \text{context } C \text{ inv : } 4 \leq \text{role}_2 \rightarrow \text{size}()) \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size}()) \leq 17 \} \]
\[ \cup \{ \text{context } C \text{ inv : } \text{role}_2 \rightarrow \text{size}()) = 4 \text{ or } \text{role}_2 \rightarrow \text{size}()) = 17 \} \]

**Why Multiplicities as Constraints?**

More precise, can’t we just use **types**? (cf. Slide 29)

- \( \mu = 0..1, \mu = 1 \):
  - many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — this is why we excluded them.

- \( \mu = * \):
  - could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have \( \mu_{OCL} = \text{true} \) anyway.

- \( \mu = 0..\mathbb{N} \):
  - 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...?
**Multiplicities Never as Types...?**

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.

---

**Properties**

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>current setting</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>

---

\[\text{igraph diagram showing relationships between objects} \]

“related triple”
**Properties**

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 at most one ( r )-link to a single other object</td>
<td>current setting</td>
</tr>
<tr>
<td>bag</td>
<td>one object may have multiple ( 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 sequence 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 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**

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:**

- [Image of diagram showing ownership concept]
**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

(i) \[
\begin{array}{c}
C \\
\end{array} 
\begin{array}{c}
\text{0..1} \\
\text{role} \\
\end{array} 
\begin{array}{c}
\rightarrow \\
\rightarrow \\
D \\
\end{array}
\]

(ii) \[
\begin{array}{c}
C \\
\end{array} 
\begin{array}{c}
* \\
\text{role} \\
\end{array} 
\begin{array}{c}
\rightarrow \\
\rightarrow \\
D \\
\end{array}
\]

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

- Form (i) introduces `role : C_0,1` and form (ii) introduces `role : C_\ast` in `V`.
- In both cases, `role \in atr(C)`.
- We drop `\lambda` and go back to our nice `\sigma` with `\sigma(u)(\text{role}) \subseteq \mathcal{P}(D)`.
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 }
```

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

Sometimes, content is explicitly classified for clarity:

```
OCL:
   expr
```
Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)

For simplicity, we view the above as an abbreviation for
**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.

---

**Invariant in Class Diagram Example**

- \( \begin{array}{c|c}
  C \\
  v: \tau \{ v \geq 3 \}
\end{array} \)

If $\mathcal{D}$ consists of only $\mathcal{CD}$ with the single class $C$, then

- \( \text{Inv}(\mathcal{D}) = \text{Inv}(\mathcal{CD}) = \)
Definition. Let $\mathcal{C}D$ be a set of class diagrams. We say, the semantics of $\mathcal{C}D$ is the signature it induces and the set of OCL constraints occurring in $\mathcal{C}D$, denoted

$$[\mathcal{C}D] := (\mathcal{I}(\mathcal{C}D), \text{Inv}(\mathcal{C}D)).$$

Given a structure $\mathcal{D}$ of $\mathcal{I}$ (and thus of $\mathcal{C}D$), the class diagrams describe the system states $\Sigma_{\mathcal{D}}$, of which some may satisfy $\text{Inv}(\mathcal{C}D)$.

In pictures:

$$\mathcal{C}D = \{CD_1, \ldots, CD_n\}$$

- Signature $\mathcal{I}(\mathcal{C}D)$
- Invariants $\text{Inv}(\mathcal{C}D)$
- Basic (classes and attributes)
- Extended (visibility)

References
References

