Software Design, Modelling and Analysis in UML

Lecture 9: Class Diagrams IV

2015-12-01

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:
- Associations syntax and semantics.
- Associations in OCL syntax.

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - Compute the value of a given OCL constraint in a system state with links.
  - How did we treat “multiplicity” semantically?
  - What does “navigability”, “ownership”, . . . mean?
  - . . .

- Content:
  - Associations and OCL: semantics.
  - Associations: the rest.
Recall: Associations and OCL Syntax

Recall: OCL syntax as introduced in Lecture 3, interesting part:

\[
expr ::= \ldots | r_1(expr_1) : \tau_C \rightarrow \tau_D \quad r_1 : D_{0..1} \in atr(C) \\
| r_2(expr_1) : \tau_C \rightarrow \text{Set}(\tau_D) \quad r_2 : D \in atr(C)
\]

Now becomes

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

if there is

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

Note:
- Association name as such does not occur in OCL syntax, role names do.
- \( expr_1 \) has to denote an object of a class which “participates” in the association.
Recall:
Assume $\mathit{expr}_{1} : \tau_{C}$ for some $C \in \mathcal{C}$. Set $u_{1} \equal{} \mathcal{I}[\mathit{expr}_{1}](\sigma, \beta) \in \mathcal{D}(T_{C})$.

- $I[r_{1}(\mathit{expr}_{1})](\sigma, \beta) := \begin{cases} u, & \text{if } u_{1} \in \text{dom}(\sigma) \text{ and } \sigma(u_{1})(r_{1}) = \{u\} \\ \perp, & \text{otherwise} \end{cases}$
- $I[r_{2}(\mathit{expr}_{1})](\sigma, \beta) := \begin{cases} \sigma(u_{1})(r_{2}), & \text{if } u_{1} \in \text{dom}(\sigma) \\ \perp, & \text{otherwise} \end{cases}$

Now needed:
$I[\mathit{role}(\mathit{expr}_{1})](\sigma, \lambda, \beta)$

- We cannot simply write $\sigma(u)(\mathit{role})$.
  Recall: $\mathit{role}$ is (for the moment) not an attribute of object $u$ (not in $\mathit{atr}(C)$).
- What we have is $\lambda(\mathit{r})$ (with association name $\mathit{r}$, not with role name $\mathit{role}$).
  
  $$\langle \mathit{r} : \ldots, \langle \mathit{role} : D, \mu, \ldots \rangle, \ldots \rangle, \langle \mathit{role} : C, \ldots \rangle, \ldots \rangle$$
  
  But it yields a set of $n$-tuples, of which some relate $u$ and some instances of $D$.
- $\mathit{role}$ denotes the position of the $D$’s in the tuples constituting the value of $\mathit{r}$.

OCL and Associations: Semantics Cont’d

Assume $\mathit{expr}_{1} : \tau_{C}$ for some $C \in \mathcal{C}$. Set $u_{1} \equal{} \mathcal{I}[\mathit{expr}_{1}](\sigma, \lambda, \beta) \in \mathcal{D}(T_{C})$.

- $I[\mathit{role}(\mathit{expr}_{1})](\sigma, \lambda, \beta) := \begin{cases} u, & \text{if } u_{1} \in \text{dom}(\sigma) \text{ and } L(\mathit{role})(u_{1}, \lambda) = \{u\} \\ \perp, & \text{otherwise} \end{cases}$
- $I[\mathit{role}(\mathit{expr}_{1})](\sigma, \lambda, \beta) := \begin{cases} L(\mathit{role})(u_{1}, \lambda), & \text{if } u_{1} \in \text{dom}(\sigma) \\ \perp, & \text{otherwise} \end{cases}$

where

$$L(\mathit{role})(u, \lambda) = \{(u_{1}, \ldots, u_{n}) \in \lambda(\mathit{r}) \mid u \in \{u_{1}, \ldots, u_{n}\}\} \downarrow i$$

if

$$\langle \mathit{r} : \mathit{role}_{1}, \ldots, \mathit{role}_{i}, \ldots \rangle, \ldots \rangle, \mathit{role} = \mathit{role}_{i}$$

Given a set of $n$-tuples $A$.
$A \downarrow i$ denotes the element-wise projection onto the $i$-th component.
\[ I[\text{role(expr)1}]((\sigma, \lambda), \beta) := \begin{cases} u, & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } L(\text{role})(u_1, \lambda) = \{u\} \\ \bot, & \text{otherwise} \end{cases} \]

\[ I[\text{role(expr)2}]((\sigma, \lambda), \beta) := \begin{cases} L(\text{role})(u_1, \lambda), & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases} \]

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

Associations: The Rest
Recapitulation: Consider the following association:

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

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

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.

Navigability

Navigability is treated similar to visibility:

Using names of non-navigable association ends \((\nu = \times)\) are forbidden.

Example: Given

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

the following OCL expression is not well-typed wrt. navigability,

context \(D\ inv : role.x > 0\)

The standard says: navigation is...
- ’-’: possible
- ’\times’: not possible
- ’>’: efficient

So: In general, UML associations are different from pointers/references in general!
But: Pointers/references can faithfully be modelled by UML associations.
Recall: Multiplicity is a term of the form $N_1 . . . N_2 . . . N_{2k}$, 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_{\text{OCL}}(\text{role}) :=
\begin{align*}
\text{context } C \text{ inv} &\colon (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \lor \ldots \lor (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \\
\text{for each } \langle r : \ldots, \text{role} : D, \mu, \ldots \rangle, \ldots, \langle \text{role}' : C, \ldots \rangle \in V \text{ or}
\end{align*}
\quad (r : \ldots, \langle \text{role}' : C, \ldots \rangle, \ldots) \in V,
with \text{role} \neq \text{role}', \text{if } \mu \neq 0..1, \mu \neq 1..1, \text{and}
\begin{align*}
\mu_{\text{OCL}}(\text{role}) &:= \text{context } C \text{ inv} : \text{not(oclIsUndefined(\text{role}))}
\end{align*}
\text{if } \mu = 1..1.

Note: in $n$-ary associations with $n > 2$, there is redundancy.

**Multiplicities as Constraints Example**

\begin{align*}
\mu_{\text{OCL}}(\text{role}) = \text{context } C \text{ inv} : \\
(N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \lor \ldots \lor (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k})
\end{align*}

\begin{align*}
\mathcal{CD} &:\begin{array}{c|c|c}
\text{role}_1 & C & 0..1 \\
\hline
\text{role}_2 & 4, 17 & v : \text{Int} \\
\hline
\end{array}
\end{align*}

- $\mu_{\text{OCL}}(\text{role}_2) = \text{context } C \text{ inv} : \begin{cases} 4 \leq \text{role}_2 \rightarrow \text{size} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size} \leq 17 \end{cases}$
- $\mu_{\text{OCL}}(\text{role}_3) = \text{context } C \text{ inv} : \begin{cases} 3 \leq \text{role}_2 \rightarrow \text{size} \end{cases}$
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</td>
<td>current setting</td>
</tr>
<tr>
<td></td>
<td>single other object</td>
<td></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> $r$-links to</td>
<td>have $\lambda(r)$</td>
</tr>
<tr>
<td></td>
<td>a single other object</td>
<td>yield multi-sets</td>
</tr>
<tr>
<td>ordered,</td>
<td>an $r$-link is a <strong>sequence</strong> of object</td>
<td>have $\lambda(r)$</td>
</tr>
<tr>
<td>sequence</td>
<td>identities (possibly including duplicates)</td>
<td>yield sequences</td>
</tr>
</tbody>
</table>

**Property**

<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression $role(expr)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>$T_D \rightarrow Set(T_C)$</td>
</tr>
<tr>
<td>bag</td>
<td>$T_D \rightarrow Bag(T_C)$</td>
</tr>
<tr>
<td>ordered,</td>
<td>$T_D \rightarrow Seq(T_C)$</td>
</tr>
<tr>
<td>sequence</td>
<td></td>
</tr>
</tbody>
</table>

For subsets, redefines, union, etc. see (?, 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..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, 2011b, 42) for more details).

**Not clear to me:**

---

**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) \[ C \xrightarrow{0.1 \text{ role}} D \]
(ii) \[ C \xrightarrow{\ast \text{ role}} D \]

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

• Form (i) introduces \( \text{role} : C_{0,1} \), and form (ii) introduces \( \text{role} : C \ast \) in \( V \).
• In both cases, \( \text{role} \in \text{atr}(C) \).
• 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?

**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
```

---

OCL in Notes: Conventions

```
C
...
...
expr
```

stands for

```
context C inv : expr
```

---
Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behavioural features: later)

\[
\xi v : T \{p_1, \ldots, p_n\} \{expr\}
\]

\[
\xi f(v_1 : T, \ldots, v_n : T) : T \{p_1, \ldots, p_n\} \{pre : expr_1, post : expr_2\}
\]

For simplicity, we view the above as an abbreviation for

\[
\xi v : T \{p_1, \ldots, p_n\}
\]

\[
\text{context } f \text{ pre : expr}_1 \text{ post : expr}_2
\]

**Invariants of a Class Diagram**

- Let \(\mathcal{CD}\) be a class diagram.
- We are (now) able to recognise OCL constraints when we see them, so define
  \[
  \text{Inv}(\mathcal{CD})
  \]
  as the set \(\{\phi_1, \ldots, \phi_n\}\) of OCL constraints **occurring** in notes in \(\mathcal{CD}\) — after **unfolding** all **graphical** abbreviations (cf. previous slides).
- **As usual**: consider all invariants in all notes in any class diagram — plus implicit multiplicity-induced invariants.
  \[
  \text{Inv}(\mathcal{D}) = \bigcup_{\mathcal{CD} \in \mathcal{D}} \text{Inv}(\mathcal{CD}) \cup \{\mu_{\text{OCL}}(\text{role}) \mid \langle r : \ldots, \langle \text{role} : D, \mu, \ldots, \rangle, \ldots, \langle \text{role} : C, \ldots, \rangle, \ldots \rangle \in V \text{ or } \langle r : \ldots, \langle \text{role} : D, \mu, \ldots, \rangle, \ldots, \langle \text{role} : C, \ldots, \rangle, \ldots \rangle \in V\}.
  \]
- **Analogously**: \(\text{Inv}(\cdot)\) for any kind of diagram (like **state machine diagrams**).
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

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

Given a structure $\mathcal{S}$ of $\mathcal{CD}$ (and thus of $\mathcal{C}$), the class diagrams describe the system states $\Sigma$, of which some may satisfy $\mathcal{Inv}(\mathcal{CD})$.

In pictures:

Pragmatics

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

A set of class diagrams $\mathcal{CD}$ describes the structure of system states. Together with the invariants $\mathcal{Inv}(\mathcal{CD})$ it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $\mathcal{Inv}(\mathcal{CD})$.

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

(partly following Ambler (2005))

Some Example Class Diagrams
So: what makes a class diagram a **good class diagram**?
Main and General Modelling Guideline

Be good to your audience.

“Imagine you’re given your diagram $\mathcal{D}$ and asked to conduct task $\mathcal{T}$.

- Can you do $\mathcal{T}$ with $\mathcal{D}$?
  (semantics sufficiently clear? all necessary information available? ...)

- Does doing $\mathcal{T}$ with $\mathcal{D}$ cost you more nerves/time/money/... than it should?”

In other words:

- the things most relevant for task $\mathcal{T}$, do they stand out in $\mathcal{D}$?
- the things less relevant for task $\mathcal{T}$, do they disturb in $\mathcal{D}$?

Main and General Quality Criterion

- Q: When is a (class) diagram a good diagram?
- A: If it serves its purpose/makes its point.

Examples for purposes and points and rules-of-thumb:

- Analysis/Design
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer

- Implementation/A
  - close to target platform
    ($C_{0.1}$ is easy for Java, $C_*$ comes at a cost — other way round for RDB)

- Implementation/B
  - complete, executable

- Documentation
  - Right level of abstraction: “if you’ve only one diagram to spend, illustrate the concepts, the architecture, the difficult part”
  - The more detailed the documentation, the higher the probability for regression “outdated/wrong documentation is worse than none”
General Diagramming Guidelines Ambler (2005)

(Note: “Exceptions prove the rule.”)

- 2.1 Readability
  - 1.–3. Support Readability of Lines

References
References

