Software Design, Modelling and Analysis in UML

Lecture 9: Class Diagrams IV

2016-11-29

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Content

- Associations: The Rest
  - Visibility, Navigability, Properties,
  - Ownership, “Diamonds”,
  - Multiplicity

- Back to the Main Track

- OCL in (Class) Diagrams

- What makes a class diagram a good class diagram?
  - Web-Shop Examples
  - The Elements of UML 2.0 Style
  - Example: Game Architecture
Associations: The Rest

Multiplicities

\[\begin{array}{c}
\text{context } C \text{ inv: role}_3 \rightarrow \text{size} \geq 3
\end{array}\]
Recall: Multiplicity is a term of the form \( N_1 \ldots N_{2k} \) where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \). Define \( \mu_{\text{C}O\text{L}}(\text{role}) := \) context \( C \text{inv} : (N_1 \leq \text{role} \rightarrow \text{size}() \leq N_2) \) or \( (N_{2k-1} \leq \text{role} \rightarrow \text{size}() \leq N_{2k}) \) or omit if \( N_{2k} = \ast \) for each \((r : \ldots, \langle \text{role} : D, \mu, \_\ldots, \_\rangle, \ldots) \in V \) or \((r : \ldots, \langle \text{role}' : D, \mu, \_\ldots, \_\rangle, \ldots) \in V \), with \( \text{role} \neq \text{role}' \), if \( \mu \neq 0..1, \mu \neq 1..1 \), and

\[ \mu_{\text{C}O\text{L}}(\text{role}) := \text{context } C \text{inv} : \neg (\text{oclIsUndefined}(\text{role})) \]

if \( \mu = 1..1 \).

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
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) \[ C \xleftarrow{0.1 \text{ role}} D \]

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

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

- Form (i) introduces \( \text{role} : \mathcal{D}_{0.1} \), and form (ii) introduces \( \text{role} : \mathcal{D} \), in the set of attributes \( V \).
- In both cases, \( \text{role} \in \text{atr}(C) \).
- We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(\text{role}) \subseteq \mathcal{R}(D) \).
Where Shall We Put OCL Constraints?

Three options:

(o) Separate document.
(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 in Notes: Conventions**

stands for

![Diagram](image)

**Where Shall We Put OCL Constraints?**

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

For simplicity, we view the above as an abbreviation for

![Diagram](image)
**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 $Inv(\mathcal{CD})$ as the set $\{\varphi_1, \ldots, \varphi_n\}$ 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.

\[
Inv(\mathcal{CD}) = \bigcup_{\mathcal{CD} \in \mathcal{CD}} Inv(\mathcal{CD}) \cup \{\mu_{\text{OCL}}(\text{role}) | \{r : \ldots, (\text{role} : D, \mu, \_ \_ \_ \_ \_ \_), \ldots, (\text{role}' : D, \mu, \_ \_ \_ \_ \_ \_), \ldots\} \in V \text{ or } \{r : \ldots, (\text{role}' : C, \_ \_ \_ \_ \_ \_), \ldots, (\text{role} : D, \mu, \_ \_ \_ \_ \_ \_), \ldots\} \in V\}.
\]

- **Analogously:** $Inv(\cdot)$ 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

\[
[KD] := (\mathcal{S}(\mathcal{CD}), Inv(\mathcal{CD})).
\]

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

In pictures:
**Pragmatics**

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

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

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $\text{Inv}(\mathcal{C}D)$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $\text{Inv}(\mathcal{C}D)$ 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 Web-Shop Class Diagrams
A Closer Look

![Diagram]

\[ V = \{ n : < BackEnd, BackEnd, t, r >, \}

\]
A Closer Look

[Diagram image of a class diagram with relationships between classes FrontEnd, Backend, Session, and Item. The classes have attributes and methods.]
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 \( D \) and asked to conduct task \( T \).

- Can you do \( T \) with \( D \)?
  (semantics sufficiently clear? all necessary information available? …)

- Does doing \( T \) with \( D \) cost you more nerves/time/money/… than it should?”

In other words:

- the things most relevant for task \( T \), do they stand out in \( D \)?
- the things less relevant for task \( T \), do they disturb in \( D \)?

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) farther

- Implementation/A
  - close to target platform
    \((C_0 \ldots 1 \text{ is easy for Java. } C_\ast \text{ comes at a cost – other way round for RDB})\)

- 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)

(As noted, “Exceptions prove the rule.”)

- **2.1 Readability**
  - 1.–3. Support Readability of Lines
  - 4. Apply Consistently Sized Symbols
  - 9. Minimize the Number of Bubbles
  - 10. Include White-Space in Diagrams
  - 13. Provide a Notational Legend

- **2.2 Simplicity**
  - 14. Show Only What You Have to Show
  - 15. Prefer Well-Known Notation over Exotic Notation
  - 16. Large vs. Small Diagrams
  - 18. Content First, Appearance Second
General Diagramming Guidelines Ambler (2005)

- **2.2 Simplicity**
  - 14. Show Only What You Have to Show
  - 15. Prefer Well-Known Notation over Exotic Notation
  - 16. Large vs. Small Diagrams
  - 18. Content First, Appearance Second

- **2.3 Naming**
  - 20. Set and (23. Consistently) Follow Effective Naming Conventions

- **2.4 General**
  - 24. Indicate Unknowns with Question-Marks
  - 25. Consider Applying Color to Your Diagram
  - 26. Apply Color Sparingly

Class Diagram Guidelines Ambler (2005)

- **5.1 General Guidelines**
  - 88. Indicate Visibility Only on Design Models (in contrast to analysis models)

- **5.2 Class Style Guidelines**
  - 96. Prefer Complete Singular Nouns for Class Names
  - 97. Name Operations with Strong Verbs
  - 99. Do Not Model Scaffolding Code [Except for Exceptions]
    
    *e.g.* `get/set` methods
5.2 Class Style Guidelines

- 103. Never Show Classes with Just Two Compartments
- 104. Label Uncommon Class Compartments
- 105. Include an Ellipsis (...) at the End of an Incomplete List
- 107. List Operations/Attributes in Order of Decreasing Visibility

5.3 Relationships

- 112. Model Relationships Horizontally
- 115. Model a Dependency When the Relationship is Transitory
- 117. Always Indicate the Multiplicity
- 118. Avoid Multiplicity "*"
- 119. Replace Relationship Lines with Attribute Types
Class Diagram Guidelines Ambler (2005)

5.4 Associations

- 127. Indicate Role Names When Multiple Associations Between Two Classes Exist
- 129. Make Associations Bidirectional Only When Collaboration Occurs in Both Directions
- 131. Avoid Indicating Non-Navigability
- 133. Question Multiplicities Involving Minimums and Maximums

5.6 Aggregation and Composition

- exercises

Tell Them What You’ve Told Them...

- Associations:
  - view multiplicities as shorthand for constraints.
- OCL constraints can be added to a class diagram in notes or at dedicated places.
- The semantics of a class diagram is its (extended) signature, and a set of (explicit and implicit) OCL constraints.
- Class Diagrams can be “drawn” well or not so well.
- A diagram is a good diagram if it serves its purpose.
- Purposes (for class diagrams):
  - Documentation of the top-level architecture.
  - Documentation of the structural design decisions.
  - Details can go into comments in the code.
- Ambler (2005): The Elements of UML 2.0 Style.
References

