Software Design, Modelling and Analysis in UML

Lecture 9: Class Diagrams IV

2016-11-29

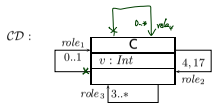
Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Content

Associations: The Rest

Multiplicities



context C inv : $role_3$ -> size ≥ 3

Multiplicities as Constraints

Recall: Multiplicity is a term of the form $\underbrace{N_1..N_2}, \ldots, \underbrace{N_{2k-1}..N_{2k}}$
where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{*\}$.

Define $\mu^C_{\mathsf{OCL}}(role) :=$

context C inv : $(N_1 \leq role$ -> size() $\leq N_2)$ or $\ldots$ or $(N_{2k-1} \leq role$ -> size() $\underbrace{\leq N_{2k}}_{\text{omit if } N_{2k} = *})$

for each $\langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_\rangle, \ldots\rangle \in V$ or

$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V,$$

with $role \neq role'$, if $\mu \neq 0..1$, $\mu \neq 1..1$, and

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

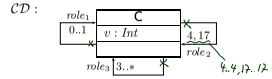
if $\mu = 1..1$.

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

Multiplicities as Constraints Example

$\mu^C_{\mathsf{OCL}}(role) =$ context C inv :
$(N_1 \leq role$ -> size() $\leq N_2)$ or $\ldots$ or $(N_{2k-1} \leq role$ -> size() $\leq N_{2k})$

$\mu^C_{\mathsf{OCL}}(role) =$ context C inv : not(oclIsUndefined($role$))



- context C inv : $3 \leq role_3$->size() ~~4 or 17 ≤ role2 -> size() ≤ 17~~
  ~~= [context C inv : role3 -> size() ≥ 3 or role2 -> size() <= 17]~~
- context C inv : $4 \leq role_2$->size() ≤ 4 or $17 \leq role_2$->size() ≤ 17

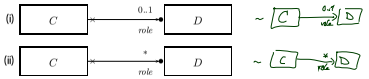
Back to the Main Track

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



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

- Form (i) introduces $role : \mathfrak{D}_{0..1}$, and form (ii) introduces $role : \mathfrak{D}_*$, in the set of attributes V.
- In both cases, $role \in atr(C)$.
- We drop λ and go back to our nice σ with $\sigma(u)(role) \subseteq \mathscr{P}(D)$.

OCL Constraints in (Class) Diagrams

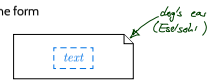
Where Shall We Put OCL Constraints?

Three options:

(o) Separate document.
(i) Notes.
(ii) Particular dedicated places.

(i) **Notes**:

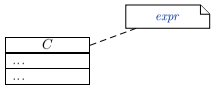
A UML **note** is a picture of the form



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

Sometimes, content is **explicitly classified** for clarity:

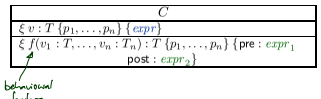
OCL in Notes: Conventions



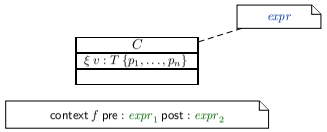
stands for

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



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.

$$Inv(\mathscr{CD}) = \bigcup_{\mathcal{CD} \in \mathscr{CD}} Inv(\mathcal{CD}) \cup$$

only applies in the general setting for associations

$$\{\mu^C_{\mathsf{OCL}}(role) \mid \langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots\rangle \in V \text{ or } \langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V\}.$$

- Analogously: $Inv(\cdot)$ for any kind of diagram (like state machine diagrams).

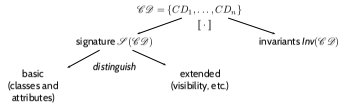
Semantics of a Class Diagram

Definition. Let $\mathscr{CD}$ be a set of class diagrams.
We say, the semantics of $\mathscr{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathscr{CD}$, denoted

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

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

In pictures:



Pragmatics

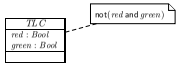
Recall: a UML model is an image or pre-image of a software system.

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

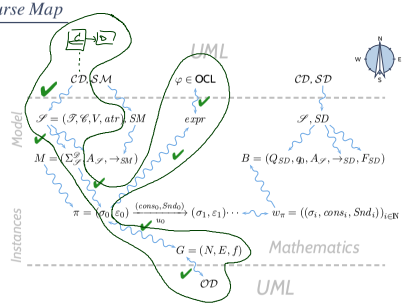
- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $Inv(\mathscr{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $Inv(\mathscr{CD})$ are used.

(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.



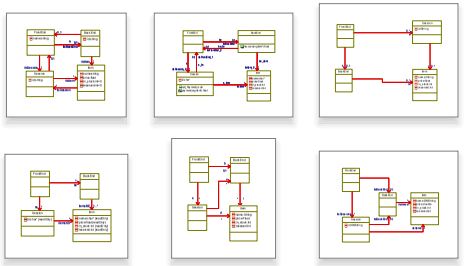
Course Map



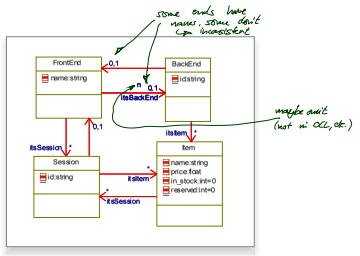
Design Guidelines for (Class) Diagram

(partly following Ambler (2005))

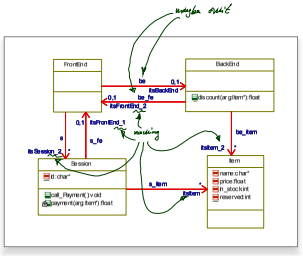
Some Web-Shop Class Diagrams



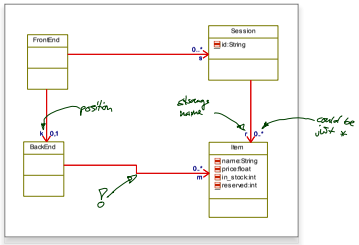
A Closer Look



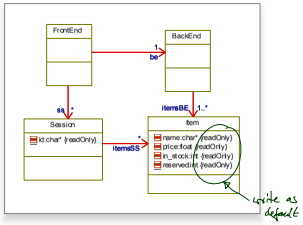
A Closer Look



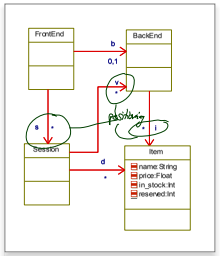
A Closer Look



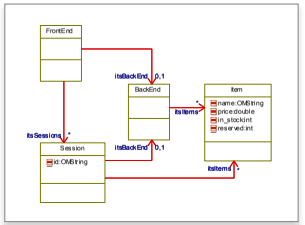
A Closer Look



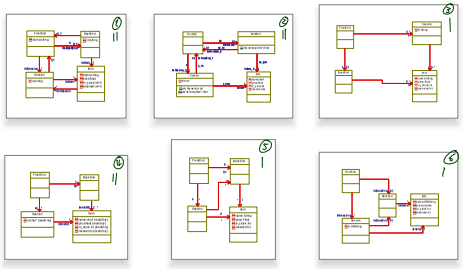
A Closer Look



A Closer Look



Some Web-Shop 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?"
  (syntactical well-formedness? readability? intention of deviations from standard syntax clear? reasonable selection of information? layout? ...)

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.

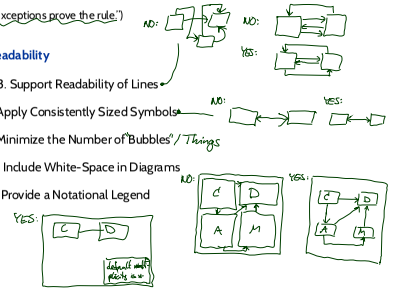
Examples for purposes and points and rules-of-thumb:

- **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)

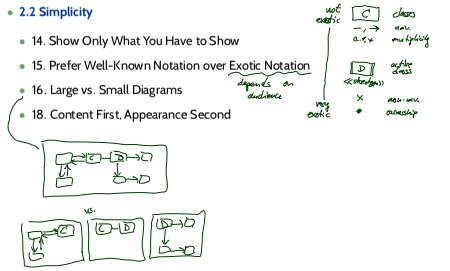
(Note: "Exceptions prove the rule.")

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



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



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)

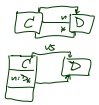
- **5.1 General Guidelines**
  - 88. Indicate Visibility Only on Design Models (in contrast to analysis models)
- **5.2 Class Style Guidelines**
  - 96. Prefer Complete Singular Nouns for Class Names
  - 97. Name Operations with Strong Verbs
  - 99. Do Not Model Scaffolding Code [Except for Exceptions]
    e.g. get/set methods

Class Diagram Guidelines Ambler (2005)

- **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
    (from + to −)

Class Diagram Guidelines Ambler (2005)

- **5.3 Relationships**
  - 112. Model Relationships Horizontally
  - 115. Model a Dependency When the Relationship is Transitory
  - 117. Always Indicate the Multiplicity
    (or know good defaults)
  - 118. Avoid Multiplicity "*"
  - 119. Replace Relationship Lines with Attribute Types
    (to have fewer lines)



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** (if depends, often → is meant to be ↔)
  - 133. Question Multiplicities Involving Minimums and Maximums
    e.g. 3..10
- **5.6 Aggregation and Composition**
  - → exercises

Tell Them What You've Told Them...

- **Associtions**:
  - 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

Ambler, S. W. (2005). *The Elements of UML 2.0 Style*. Cambridge University Press.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.