

# Software Design, Modelling and Analysis in UML

## Lecture 9: Class Diagrams IV

2016-11-29

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

### Content

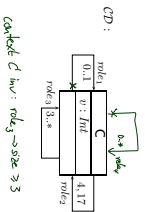
- Associations: The Rest
- Visibility, Navigability, Properties, Ownership: Diamonds?
- Multiplicity
- Back to the Main Track
- OCL in (Class) Diagrams
- What makes a class diagram a good class diagram?
- Web-Shop Examples
- The Elements of UML 2.0 Style
- Example: Game Architecture

2/18

### Associations: The Rest

3/18

### Multiplicities



4/18

### Multiplicities as Constraints

**Recall: Multiplicity** is a term of the form  $N_1..N_2, \dots, N_{M-1}..N_M$  where  $N_i \leq M_i$  for  $1 \leq i \leq 2k$ ,  $M_1, \dots, M_{k-1} \in \mathbb{N}$ ,  $N_M \in \mathbb{N} \cup \{\ast\}$

**Define**  $f_{GC}(m_k) :=$  context  $C \text{ inv: } (N_1 \leq m_k \rightarrow \text{size}() \leq N_2) \text{ or } \dots \text{ or } (N_{M-1} \leq m_k \rightarrow \text{size}() \leq N_M)$

for each  $\{r : \dots, \{role : D, m_1, \dots, \dots, \{role' : \dots, \dots, \dots\} \in V \text{ or } \dots\} \in V$

$\{r : \dots, \{role' : \dots, \dots, \dots, \dots, \dots, \dots, \dots\} \in V$

with  $m_k \neq m_k'$ , if  $m_1 \neq 0, 1, m_1 \neq 1, 1$  and

$f_{GC}(m_k) :=$  context  $C \text{ inv: not}(\text{oclIsUnderfined}(m_k))$

if  $m = 1, 1$ .

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

5/18

### Multiplicities as Constraints Example

$f_{GC}(m_k) =$  context  $C \text{ inv: } (N_1 \leq m_k \rightarrow \text{size}() \leq N_2) \text{ or } \dots \text{ or } (N_{M-1} \leq m_k \rightarrow \text{size}() \leq N_M)$

$f_{GC}(m_k) =$  context  $C \text{ inv: not}(\text{oclIsUnderfined}(m_k))$

CD:

Calculate C Inv: 3 <= reqs -> size() (0..17 <= reqs -> size() <= 17)

- Calculate C Inv: 4 <= reqs -> size() 4 or 17 <= reqs -> size() 17

6/18

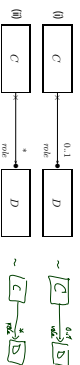
Back to the Main Track

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



OCL Constrains in (Class) Diagrams

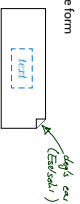
Where Shall We Put OCL Constrains?

Three options:

- (i) Separate document
- (ii) Notes
- (iii) Particular dedicated places

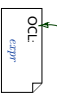
(i) Notes:

A UML note is a picture of the form



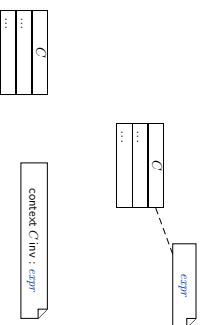
text can principally be **everything**, in particular comments and constrains.

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



OCL in Notes: Conventions

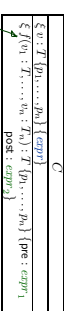
stands for



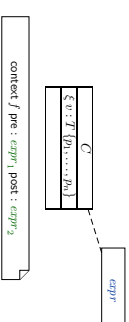
Where Shall We Put OCL Constrains?

(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  $CD$  be a class diagram.
- We are (now) able to recognise OCL constraints when we see them, so define  $Inv(CD)$  as the set  $\{\varphi_1, \dots, \varphi_n\}$  of OCL constraints occurring in nodes in  $CD$  – after unfolding all graphical abbreviations (cf. previous slides).
- As usual, consider all invariants in all nodes in any class diagram – plus implicit multiplicity-induced invariants.

$$Inv(\mathcal{C}) = \bigcup_{CD \in \mathcal{C}} Inv(CD) \cup \text{multiplicity-induced invariants}$$

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

13/38

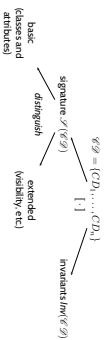
## Semantics of a Class Diagram

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

$$[[\mathcal{C}]] := (\mathcal{S}(\mathcal{C}), Inv(\mathcal{C})).$$

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

In pictures:



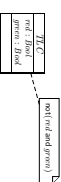
14/38

## Pragmatics

- Recall: a UML model is an image or pre-image of a software system.
- A set of class diagrams  $\mathcal{C}$  describes the structure of system states. Together with the invariants  $Inv(\mathcal{C})$  it can be used to state:
  - Pre-image: Dear programmer, please provide an implementation which uses only system states that satisfy  $Inv(\mathcal{C})$ .
  - Post-image: Dear user/maintainer: in the existing system, only system states which satisfy  $Inv(\mathcal{C})$  are used.

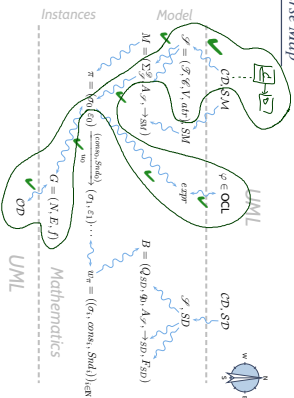
(The exact meaning of "use" will become clear when we study behavior – initially, the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines)

**Example:** highly abstract model of traffic lights controller.



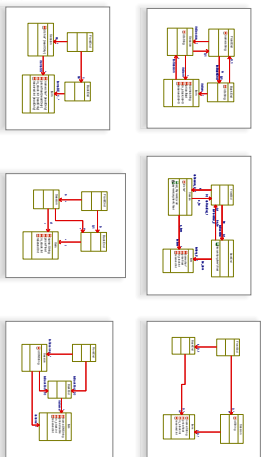
15/38

## Course Map



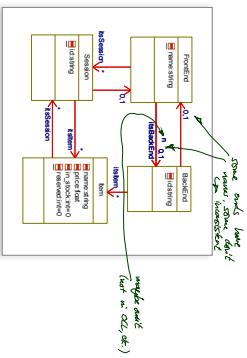
16/38

## Design Guidelines for (Class) Diagram (partly following Ambler (2005))



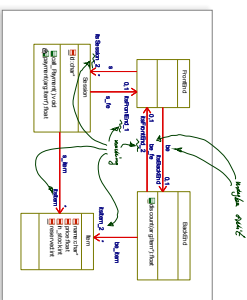
18/38

A Closer Look



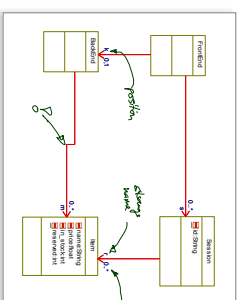
19/38

A Closer Look



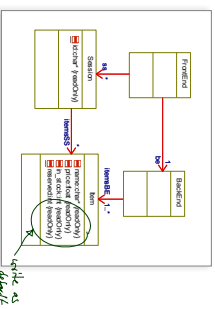
19/38

A Closer Look



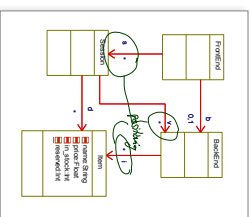
19/38

A Closer Look



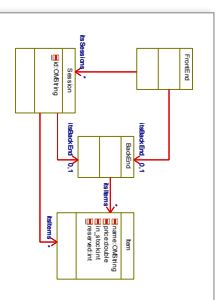
19/38

A Closer Look

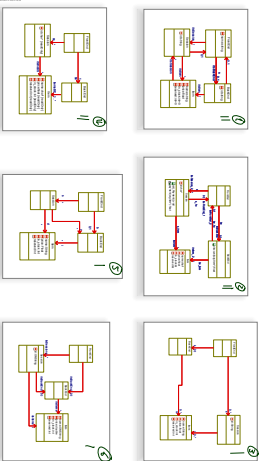


19/38

A Closer Look



19/38



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**
  - readable, no contradictions
  - abstract, focused, admitting degrees of freedom to (more detailed) design
  - platform independent – as far as possible but not (artificially) free
- **Implementation /A**
  - clear to target platform
  - (C++) is easy for Java, C, comes at a cost – other way round for Java!
- **Implementation /B**
  - complete, executable
- **Documentation**
  - right level of abstraction, "if you're 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"

So, what makes a class diagram a good class diagram?

General Diagramming Guidelines Ambler (2005)

- (Note: "Exaggerates purpose here!")
- **21 Readability**
    - 1-3 Support Readability of Lines
    - 4 Apply Consistently Sized Symbols
    - 9 Minimize the Number of Bubbles / Tags
    - 10 Include White-Space in Diagrams
    - 13 Provide a Notational Legend
- 

Be good to your audience.

- "Imagine you're given your diagram D and asked to conduct task T. (semantics sufficiently clear? all necessary information available?)"
- Can you do T with D?
- Does doing T with D cost you more nerves/time/money? ... than it should?" (syntactical well-formedness? readability? invention of deviations from standard syntax clear? reasonable selection of information? layout? ...)

General Diagramming Guidelines Ambler (2005)

- **22 Simplicity**
    - 14 Show Only What You Have to Show
    - 15 Prefer Well-Known Notation over Exotic Notation
    - 16 Larger 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

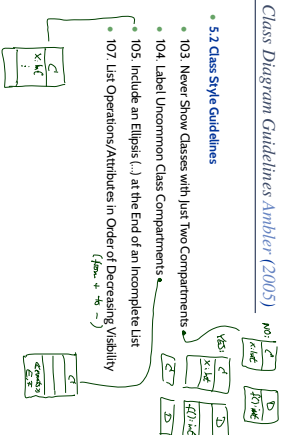
25/18

## 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 Scalloping Code [Except for Exceptions]
    - eg: `getFact` `makeNode`

26/18

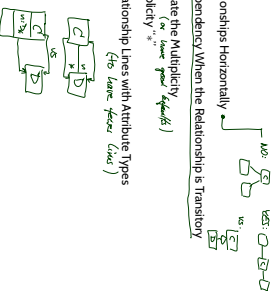
## Class Diagram Guidelines Ambler (2005)



27/18

## 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
  - 118. Avoid Multiplicity "..." (for same "power" class)
  - 119. Replace Relationship Lines with Attribute Types



28/18

## 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 (is *avoided*, *only* `class` → `class`)
  - 133. **Question** Multiplicities Involving Minimals and Maximals
    - eg: 3..10
- 5.6 Aggregation and Composition
  - `o` - exercises



29/18

## 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**

36/18

References

---

*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.