

Software Design, Modelling and Analysis in UML

Lecture 09: Class Diagrams IV

2012-11-27

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

Contents & Goals

- **Last Lectures:**
 - Started to discuss "associations", 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:**
 - Treat "the rest".
 - Where do we put OCL constraints?
- **Modelling guidelines, in particular for class diagrams (following [Ambler, 2005])**

Associations: The Rest

The Rest

Recapitulation: Consider the following association:

$\langle r : \{role_1 : C_1, \mu_1, R_1, \xi_1, \nu_1, \sigma_1\}, \dots, \{role_n : C_n, \mu_n, R_n, \xi_n, \nu_n, \sigma_n\} \rangle$

- **Association name r and role names/types** $role_i/C_i$ induce extended system states λ .
- **Multiplicity μ_i** is considered in OCL syntax.
- **Visibility ξ_i /Navigability ν_i :** well-typedness.

Now the rest:

- **Multiplicity μ_i :** we propose to view them as constraints.
- **Properties P_i :** even more typing.
- **Ownership σ_i :** getting closer to pointers/references.
- **Diamonds:** exercise.

Visibility

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

Question: given



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

context C : inv: self.role.x > 0

Visibility

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

Question: given



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

context C : inv: self.role.x > 0

Basically same rule as before (analogously for other multiplicities)

$$\begin{aligned}
 (Assoc) \quad & \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash role_i(capt_1) : \tau_D} \quad \mu = 0, 1 \text{ or } \mu = 1, \\
 & \quad \xi = +, \text{ or } \xi = - \text{ and } C = B \\
 \langle r : \dots, \{role : D, \mu = \xi_i \cdot \nu_i\}, \dots, \{role' : C, \dots, \nu_i\}, \dots \rangle \in V
 \end{aligned}$$

Navigability

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

Question: given



is the following OCL expression well-typed or not (wrt. navigability)?
context D inv : self.role.x > 0

Navigability

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

Question: given



is the following OCL expression well-typed or not (wrt. navigability)?
context D inv : self.role.x > 0

The standard says:
 • '-': navigation is possible
 • '>': navigation is efficient
 • 'x': navigation is not possible
 by context decide what official names to get and communicate this to the designers
 So: In general, UML associations are different from pointers/references
 But: Pointers/references can faithfully be modelled by UML associations.

The Rest of the Rest

Recapitulation: Consider the following association:

$(r : (role_1 : C_1, role_2 : P_1, S_1, r_1, o_1), \dots, (role_n : C_n, role_{n+1} : P_n, S_n, r_n, o_n))$

- Association name r and role names / types
- role $_i$ / C_i induce extended system states λ_i
- Multiplicity μ_i is considered in OCL syntax.
- Visibility ε_i / Navigability ν_i : well-typedness.

Now the rest:

- Multiplicity μ_i : we propose to view them as constraints.
- Properties P_i : even more typing.
- Ownership o_i : getting closer to pointers/references.
- Diamonds: exercise.

Multiplicities as Constraints

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

$$\mu ::= * | N | N..M | N..* | \mu, \mu \quad (N, M \in \mathbb{N})$$

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

Multiplicities as Constraints

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

$$\mu ::= * | N | N..M | N..* | \mu, \mu \quad (N, M \in \mathbb{N})$$

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

Recall: we can normalize each multiplicity μ to the form:

$$N_1..N_2, \dots, N_{2k-1}..N_{2k} \quad \text{eg } 3..4, 2, \mu$$

$$3..3, 0..2, 1/11 \quad \text{eg } 3, 0, 2, \mu$$

$$N_1, \dots, N_{2k-1} \in \mathbb{N}, N_{2k} \in \mathbb{N} \cup \{*\} \quad \text{eg } 3, 3, 0..2, 1/11, 0..*$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \dots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{*\}$.

Multiplicities as Constraints

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \dots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{*\}$.

Multiplicities as Constrains

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \dots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{\infty\}$.

Define $\text{isect}(v, l) := \text{context } C \text{ inv} :$

$$(N_i \leq \text{size}(v) \rightarrow \text{size}(l) \leq N_i) \text{ or } \dots \text{ or } (N_{2k-1} \leq \text{size}(v) \rightarrow \text{size}(l) \leq N_{2k})$$

for each $h \neq 0, 1$, $h \neq 1$,

$$\{r : \dots, \text{isect}(v, l) : C, h, \dots, \text{isect}(v, l) : D, h, \dots, \text{isect}(v, l) : V, \text{size}(v) \neq \text{size}(l)\} \in V \text{ or}$$

$$\{r : \dots, \text{isect}(v, l) : C, h, \dots, \text{isect}(v, l) : D, h, \dots, \text{isect}(v, l) : V, \text{size}(v) \neq \text{size}(l)\} \in V; \text{size}(v) \neq \text{size}(l).$$

And define

$$\text{isect}(v, l) := \text{context } C \text{ inv} : \text{not}(\text{isect}(\text{undefined}(v), l))$$

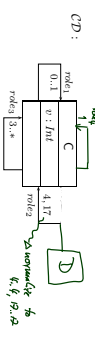
for each $h = 1$.

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



Multiplicities as Constrains Example

$\text{isect}(v, l) := \text{context } C \text{ inv} :$
 $(N_1 \leq \text{size}(v) \rightarrow \text{size}(l) \leq N_1) \text{ or } \dots \text{ or } (N_{2k-1} \leq \text{size}(v) \rightarrow \text{size}(l) \leq N_{2k})$



$\text{inv}(C, D) :=$

$$\{ \text{context } D \text{ inv} : h \in \text{size}(v) \rightarrow \text{size}(l) \leq h \text{ or } h \in \text{size}(v) \rightarrow \text{size}(l) \in h \}$$

$$\cup \{ \text{context } C \text{ inv} : 3 \leq \text{size}(v) \rightarrow \text{size}(l) \}$$

$$\cup \{ \text{context } C \text{ inv} : \text{not}(\text{isect}(\text{undefined}(v), l)) \}$$



Why Multiplicities as Constrains?

More precise, can't we just use types? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$: many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:

Why Multiplicities as Constrains?

More precise, can't we just use types? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$: many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$: could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\text{isect} = \text{true}$ anyway.
- $\mu = 0..3$:

Why Multiplicities as Constrains?

More precise, can't we just use types? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:

Why Multiplicities as Constrains?

More precise, can't we just use types? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$: many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$: could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\text{isect} = \text{true}$ anyway.
- $\mu = 0..3$: 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...?
- $\mu = 3..7$:

Why Multiplicities as Constraints?

More precise, can't we just use types? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$: many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
 - $\mu = *$: could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\text{mult} = \text{true}$ anyway.
 - $\mu = 0..4$: 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 $0, \dots, N$,
 - set types,
- then we could simply restrict the syntax of multiplicities to

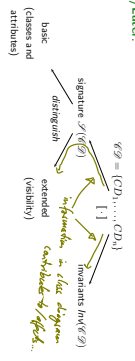
$$\mu ::= [0..N] *$$

and don't think about constraints (but use the obvious 1-to-1 mapping to types)...

In general, **unfortunately**, we don't know.

Multiplicities as Constraints of Class Diagram

Recall/Later:

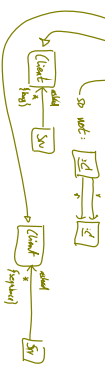


From now on: $\text{Inv}(C,D) = \{\text{constraints occurring in notes}\} \cup \{\text{OCL}(role) \mid \{r : \dots, (role : D, \mu = \dots), \dots, (role : C, \dots) \} \in V \text{ or } \{r : \dots, (role : C, \dots), \dots, (role : D, \mu = \dots), \dots\} \in V, role \neq role', \mu \notin \{0..1\}\}$

Properties

We don't want to cover association properties in detail, only some observations (assume binary associations):

Property	Intuition	Semantical Effect
unique	one object has at most one r -link to a single other object	current setting
bag	one object may have multiple r -links to a single other object	have $\lambda(r)$ yield multi-sets
ordered sequence	an r -link is a sequence of object identities (possibly including duplicates)	have $\lambda(r)$ yield sequences



Properties

We don't want to cover association properties in detail, only some observations (assume binary associations):

Property	Intuition	Semantical Effect
unique	one object has at most one r -link to a single other object	current setting
bag	one object may have multiple r -links to a single other object	have $\lambda(r)$ yield multi-sets
ordered sequence	an r -link is a sequence of object identities (possibly including duplicates)	have $\lambda(r)$ yield sequences

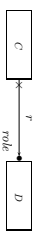
Property	OCL Typing of expression $\text{role}(expr)$
unique	$TD \rightarrow \text{Set}(TC)$
bag	$TD \rightarrow \text{Bag}(TC)$
ordered sequence	$TD \rightarrow \text{Seq}(TC)$

For subsets, redlines, union, etc see [OMG, 2007a, 127]

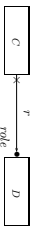
Ownership

Intuitively it says:

Association r is **not** a "thing on its own" (ie. provided by λ), but association end $role'$ is owned by $C()$. (That is, it's stored inside C object and provided by σ)



Ownership

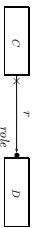


Intuitively it says:

Association r is not a "thing on its own" (i.e. provided by λ), but association end role^r is owned by C (i). (That is, it's stored inside C object and provided by σ .)

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).

Ownership

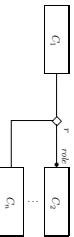


Intuitively it says:

Association r is not a "thing on its own" (i.e. provided by λ), but association end role^r is owned by C (i). (That is, it's stored inside C object and provided by σ .)

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:



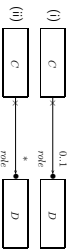
Back to the Main Track

Back to the main track:

Recall: on some earlier slides we said, the extension of the signature is **only** to study associations in "full heavy". For the remainder of the course, we should look for something simpler...

Proposal:

- From now on, we only use associations of the form



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

- Form (i) introduces $\text{role} : C_{0..1}$ and form (ii) introduces $\text{role} : C_{\lambda}$ in V .

- In both cases, $\text{role} \in \text{atr}(C)$

- We drop λ and go back to our nice σ with $\sigma(\lambda)(\text{role}) \subseteq \mathcal{O}(D)$.

OCL Constraints in (Class) Diagrams

Where Shall We Put OCL Constraints?

Numerous options:

(i) Additional documents.

(ii) Notes.

(iii) Particular dedicated places.

Where Shall We Put OCL Constraints?

Numerous options:

- (i) Additional documents;
- (ii) Notes;
- (iii) Particular dedicated places;
- (i) **Notes:**
A UML note is a picture of the form



Exclude, (English does not)

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

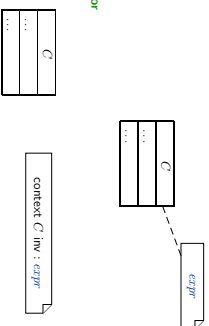
Sometimes, content is explicitly classified for clarity:



19.e

OCL in Notes: Conventions

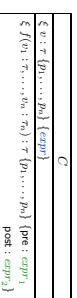
stands for



20.e

Where Shall We Put OCL Constraints?

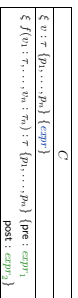
- (ii) Particular dedicated places in class diagrams: (behav. feature: later)



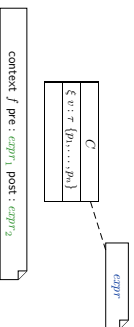
21.e

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



21.e

Invariants of a Class Diagram

- Let CD be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

$$Inv(CD)$$

as the set $\{\varphi_1, \dots, \varphi_n\}$ of OCL constraints occurring in notes in CD — after unfolding all abbreviations (cf. next slides)

22.e

Invariants of a Class Diagram

- Let CD be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

$$Inv(CD)$$

as the set $\{\varphi_1, \dots, \varphi_n\}$ of OCL constraints occurring in notes in CD — after unfolding all abbreviations (cf. next slides).

- As usual: $Inv(\mathcal{C}) := \bigcup_{CD \in \mathcal{C}} Inv(CD)$, + *implicit constraint from abbreviations (cf. below)*
- Principally clear: $Inv(\cdot)$ for any kind of diagram.

22.e

Invariant in Class Diagram Example

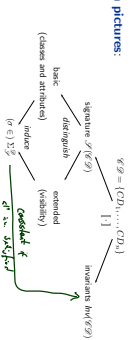


If $\mathcal{C}\mathcal{D}$ consists of only CD with the single class C , then

$$\text{Inv}(\mathcal{C}\mathcal{D}) = \text{Inv}(CD) = \{ \text{constraint } \{ \text{inv}: v > 3 \} \}$$

Semantics of a Class Diagram

Definition. Let $\mathcal{C}\mathcal{D}$ be a set of class diagrams. We say, the semantics of $\mathcal{C}\mathcal{D}$ is the signature it induces and the set of OCL constraints occurring in $\mathcal{C}\mathcal{D}$, denoted $[\mathcal{C}\mathcal{D}] = (\mathcal{S}(\mathcal{C}\mathcal{D}), \text{Inv}(\mathcal{C}\mathcal{D}))$.
 Given a structure σ of \mathcal{S} (and thus of $\mathcal{C}\mathcal{D}$), the class diagrams describe the system states $\Sigma_{\mathcal{C}\mathcal{D}}$. Of those, some satisfy $\text{Inv}(\mathcal{C}\mathcal{D})$ and some don't. We call a system state $\sigma \in \Sigma_{\mathcal{C}\mathcal{D}}$ consistent if and only if $\sigma \models \text{Inv}(\mathcal{C}\mathcal{D})$.



Pragmatics

Recall: a UML model is an image or pre-image of a software system. A set of class diagrams $\mathcal{C}\mathcal{D}$ with invariants $\text{Inv}(\mathcal{C}\mathcal{D})$ describes the **structure** of system states. Together with the invariants it can be used to state:

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

(The exact meaning of 'used' will become clear when we study behaviour — in particular, the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

Pragmatics

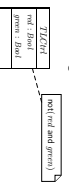
Recall: a UML model is an image or pre-image of a software system.

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

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

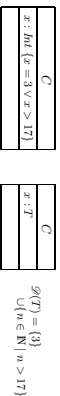
(The exact meaning of 'used' will become clear when we study behaviour — in particular, the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

Example: highly abstract model of traffic lights controller.



Constraints vs. Types

Find the 10 differences:



- $x = 4$ is well-typed in the left context, a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context, there cannot be a system state with $\sigma(U)(x) = 4$ because $\sigma(U)(x)$ is supposed to be in $\mathcal{D}(T)$ (by definition of system state).

Pragmatics

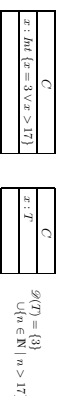
Recall: a UML model is an image or pre-image of a software system. A set of class diagrams $\mathcal{C}\mathcal{D}$ with invariants $\text{Inv}(\mathcal{C}\mathcal{D})$ describes the **structure** of system states. Together with the invariants it can be used to state:

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

(The exact meaning of 'used' will become clear when we study behaviour — in particular, the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

Constraints vs. Types

Find the 10 differences:



- $x = 4$ is well-typed in the left context, a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context, there cannot be a system state with $\sigma(U)(x) = 4$ because $\sigma(U)(x)$ is supposed to be in $\mathcal{D}(T)$ (by definition of system state).

Rule-of-thumb:

- If something "feels like" a **type** (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a **requirement** or restriction of an otherwise useful type, then make it a constraint.

Design Guidelines for (Class) Diagram

(partly following Ambler 2003)

*Be careful whose advice you buy, but,
be patient with those who supply it.
Bret Lehnman/King Smith*

- 09 - 2012-11-27 - main -

27/4

Main and General Modelling Guideline (ambler's: trivial and obvious)

Be good to your audience.

- 09 - 2012-11-27 - Seminars -

28/4

Main and General Modelling Guideline (ambler's: trivial and obvious)

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

- 09 - 2012-11-27 - Seminars -

28/4

Main and General Modelling Guideline (ambler's: trivial and obvious)

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

In other words:

- the things **most relevant** for T , do they **stand out** in D ? *if yes, good*
- the things **less relevant** for T , do they **disturb** in D ? *if yes, bad*

- 09 - 2012-11-27 - Seminars -

28/4

Main and General Quality Criterion (ogden: trivial and obvious)

- **Q:** When is a (class) diagram a good diagram?

- 09 - 2012-11-27 - Seminars -

29/4

Main and General Quality Criterion (ogden: trivial and obvious)

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

- 09 - 2012-11-27 - Seminars -

29/4

Main and General Quality Criterion (again: trivial and obvious)

- **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**

Main and General Quality Criterion (again: trivial and obvious)

- **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) fair
- **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**

Main and General Quality Criterion (again: trivial and obvious)

- **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) fair
- **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**

Main and General Quality Criterion (again: trivial and obvious)

- **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) fair
- **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**

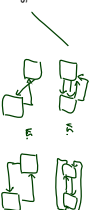
Main and General Quality Criterion (again: trivial and obvious)

- **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) fair
- **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 “undated/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



General Diagramming Guidelines [Amber, 2005]

(Note: "Exceptions prove the rule.")

2.1 Readability

- 1-3 Support Readability of Lines
- 4 Apply Consistently Sized Symbols



General Diagramming Guidelines [Amber, 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

General Diagramming Guidelines [Amber, 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?
- 10 Include White Space in Diagrams

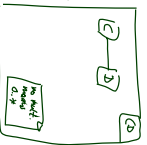


General Diagramming Guidelines [Amber, 2005]

(Note: "Exceptions prove the rule.")

2.1 Readability

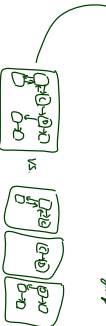
- 1-3 Support Readability of Lines
- 4 Apply Consistently Sized Symbols
- 9 Minimize the Number of Bubbles
- 10 Include White-Space in Diagrams
- 13 Provide a Notational Legend



General Diagramming Guidelines [Amber, 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



— *avoid in general in needs analysis*

— *avoiding 5/2*

— *avoiding 5/2*

— *avoiding 5/2*

General Diagramming Guidelines [Amber, 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

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

31.4

Class Diagram Guidelines [Ambler, 2005]

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

32.4

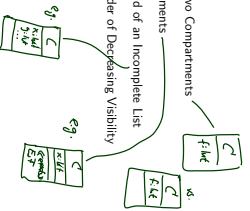
Class Diagram Guidelines [Ambler, 2005]

- 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]

32.4

Class Diagram Guidelines [Ambler, 2005]

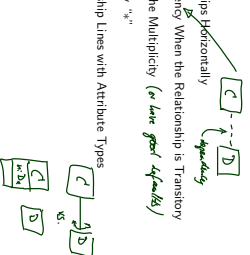
- 5.2 Class Style Guidelines
 - 103. Never Show Classes with Just Two Compartments
 - 104. Label Uncommon Class Compartments
 - 105. Include an Ellipsis (...) at the End of an Incomplete List
 - 107. List Operations/Attributes in Order of Decreasing Visibility



33.4

Class Diagram Guidelines [Ambler, 2005]

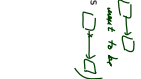
- 5.3 Relationships
 - 112. Model Relationships Horizontally
 - 115. Model a Dependency When the Relationship is Transitory
 - 117. Always Indicate the Multiplicity (or *late goal: asymmetry*)
 - 118. Avoid Multiplicity "x"
 - 119. Replace Relationship Lines with Attribute Types



34.4

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, role, C must be D*)
 - 133. Question Multiplicities Involving Minimums and Maximums (if *min=1, max=1*)



35.4

Class Diagram Guidelines [Amber, 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

[...] But trust me on the screenshot.
Byc: Lahnmann/Mery/Schlich

Example: Modelling Games

Task: Game Development

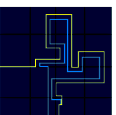
Task: develop a video game. Genre: Racing. Rest: open, i.e.

- Degrees of freedom:
- simulation vs. arcade
 - platform (SDK or not, open or proprietary, hardware capabilities...)
 - graphics (3D, 2D, ...)
 - number of players, AI
 - controller
 - game experience

Task: Game Development

Task: develop a video game. Genre: Racing. Rest: open, i.e.

- Degrees of freedom:
- simulation vs. arcade
 - platform (SDK or not, open or proprietary, hardware capabilities...)
 - graphics (3D, 2D, ...)
 - number of players, AI
 - controller
 - game experience



Modelling Structure: 2D-Tron

- In many domains, there are canonical architectures – and adept readers try to see/find/match this!
- For games:

