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
Recapitulation: Consider the following association:

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

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

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.
Visibility

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

**Question**: given

\[
\begin{array}{c}
\text{context } C \\
\text{inv : } self.role.x > 0
\end{array}
\]

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

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

**Question:** given

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ξ role</td>
</tr>
</tbody>
</table>
```

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)

\[(\text{Assoc}_1) \quad \frac{A, B \vdash \text{expr}_1 : \tau_C}{A, B \vdash \text{role}(\text{expr}_1) : \tau_D}, \quad \mu = 0..1 \text{ or } \mu = 1, \quad \xi = +, \text{ or } \xi = - \text{ and } C = B
\]

\[
\langle r : \ldots \langle \text{role} : D, \mu, -, \xi, -, - \rangle, \ldots \langle \text{role}' : C, -, -, -, -, \ldots \rangle, \ldots \rangle \in V
\]
Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are basically type-correct, but forbidden.

**Question**: given

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

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

\[
\text{context } D \text{ inv : } self\text{.role.x} > 0
\]
Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are basically type-correct, but forbidden.

**Question:** given

```
<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x : Int</td>
</tr>
<tr>
<td>role</td>
</tr>
</tbody>
</table>
```

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 not possible
- `'>'`: navigation is efficient

So: In general, UML associations are different from pointers/references!

But: Pointers/references can faithfully be modelled by UML associations.
Recapitulation: Consider the following association:

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

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

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

\[ \mu ::= \ast \mid N \mid N..M \mid N..\ast \mid \mu, \mu \]  
\[(N, M \in \text{IN})\]

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:

\[ \mu ::= * \mid N \mid N.M \mid N.* \mid \mu, \mu \quad (N, M \in \mathbb{N}) \]

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

**Recall:** we can normalize each multiplicity \( \mu \) to the form

\[ N_1..N_2, \ldots, N_{2k-1}..N_{2k} \]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N} \), \( N_{2k} \in \mathbb{N} \cup \{*\} \).
Multiplicities as Constraints

\[ \mu = N_1\ldots N_2, \ldots, N_{2k-1}\ldots N_{2k} \]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k-1} \in \mathbb{N} \), \( N_{2k} \in \mathbb{N} \cup \{\ast\} \).
Multiplicities as Constraints

\[ \mu = N_1 \ldots N_2, \ldots, N_{2k-1} \ldots N_{2k} \]

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 \( \mu_{\text{OCL}}^C(\text{role}) := \text{context } C \text{ inv :} \)

\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq \text{role} \rightarrow \text{size()} \leq N_{2k}) \]

omit if \( N_{2k} = \ast \)

for each \( \mu \neq 0 \ldots 1, \mu \neq 1 \),

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

\[ \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}' \).

And define

\[ \mu_{\text{OCL}}^C(\text{role}) := \text{context } C \text{ inv : } \text{not(oclIsUndefined(\text{role}))} \]

for each \( \mu = 1 \).

Note: in \( n \)-ary associations with \( n > 2 \), there is redundancy.
\[ \mu_{\text{OCL}}^C(\text{role}) = \text{context } C \text{ inv:} \]
\[ (N_1 \leq \text{role} \rightarrow \text{size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2^{k-1}} \leq \text{role} \rightarrow \text{size()} \leq N_{2^k}) \]

\[ \text{Inv}(CD) = \]
\[ \{ \text{context } D \text{ inv: } 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17 \} \]
\[ \text{equivalent to } \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17 \]
\[ \cup \{ \text{context } C \text{ inv: } 3 \leq \text{role}_3 \rightarrow \text{size()} \} \]
\[ \cup \{ \text{context } C \text{ inv: not } \text{ac1 is Undefined (role4)} \} \]
Why Multiplicities as Constraints?

More precise, can’t we just use \textbf{types}? (cf. Slide [26])

- $\mu = 0..1$, $\mu = 1$: 
Why Multiplicities as Constraints?

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

- $\mu = 0..1$, $\mu = 1$: many programming languages have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$: 
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 $\mu_{OCL} = \text{true}$ anyway.
- $\mu = 0..3$:
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 $\mu_{OCL} = true$ 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...?
- $\mu = 5..7$: 
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 $\mu_{OCL} = true$ 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...?
Well, if the **target platform** is known and fixed, and the target platform has, for instance,

- reference types,
- range-checked arrays with positions $0, \ldots, N$,
- set types,

then we could simply **restrict** the syntax of multiplicities to

$$\mu ::= 1 \mid 0..N \mid *$$

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

In general, **unfortunately**, we don’t know.
Recall/Later:

\[ CD = \{ CD_1, \ldots, CD_n \} \]

signature \( \mathcal{S}(CD) \)

invariants \( Inv(CD) \)

basic (classes and attributes)

extended (visibility)

\[ \text{distinguish} \]

\[ \text{infinisimally in class diagram contributes to effects...} \]

\[ \text{From now on: } Inv(CD) = \{ \text{constraints occurring in notes} \} \cup \{ \mu^C_{OCL}(\text{role}) \mid \]

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

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

\[ \text{role } \neq \text{role}', \mu \notin \{0..1\} \} \]
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><strong>unique</strong></td>
<td>one object has <em>at most one</em> ( r )-link to a single other object</td>
<td><em>current setting</em></td>
</tr>
<tr>
<td><strong>bag</strong></td>
<td>one object may have <em>multiple</em> ( r )-links to a single other object</td>
<td>have ( \lambda(r) ) yield multi-sets</td>
</tr>
<tr>
<td><strong>ordered, sequence</strong></td>
<td>an ( r )-link is a <em>sequence</em> of object identities (possibly including duplicates)</td>
<td>have ( \lambda(r) ) yield sequences</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
## Properties

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><strong>unique</strong></td>
<td>one object has <em>at most one</em> $r$-link to a single other object</td>
<td><em>current setting</em></td>
</tr>
<tr>
<td><strong>bag</strong></td>
<td>one object may have <strong>multiple</strong> $r$-links to a single other object</td>
<td>have $\lambda(r)$ yield multi-sets</td>
</tr>
<tr>
<td><strong>ordered, sequence</strong></td>
<td>an $r$-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have $\lambda(r)$ yield sequences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression $\text{role}(\text{expr})$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unique</strong></td>
<td>$\tau_D \rightarrow \text{Set}(\tau_C)$</td>
</tr>
<tr>
<td><strong>bag</strong></td>
<td>$\tau_D \rightarrow \text{Bag}(\tau_C)$</td>
</tr>
<tr>
<td><strong>ordered, sequence</strong></td>
<td>$\tau_D \rightarrow \text{Seq}(\tau_C)$</td>
</tr>
</tbody>
</table>

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].
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 \( \sigma \)).
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, 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 $\lambda$), 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, 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 beauty”. For the remainder of the course, we should look for something simpler...

**Proposal:**

- **from now on**, we only use associations of the form

  ![Diagram](image)

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

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

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

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

```
[ text ]
```

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

Sometimes, content is explicitly classified for clarity:

```
OCL:

expr
```
OCL in Notes: Conventions

stands for

\[
\text{context } C \text{ inv : } expr
\]
(ii) **Particular dedicated places** in class diagrams:  

\[
    C
\]

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

\[
    \xi \; f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau \{ p_1, \ldots, p_n \} \{ pre : expr_1 \}
    \quad \text{post : expr}_2 \}
\]

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

\[
\begin{align*}
C & \quad \xi v : \tau \{p_1, \ldots, p_n\} \{\text{expr}\} \\
\xi f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau \{p_1, \ldots, p_n\} \{\text{pre} : \text{expr}_1, \\
& \quad \text{post} : \text{expr}_2\}
\end{align*}
\]

For simplicity, we view the above as an abbreviation for

\[
\begin{align*}
C & \quad \xi v : \tau \{p_1, \ldots, p_n\} \\
\text{context } f \quad \text{pre} : \text{expr}_1 \quad \text{post} : \text{expr}_2
\end{align*}
\]
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 $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints occurring in notes in $CD$ — after unfolding all abbreviations (cf. next slides).
Let $\mathcal{CD}$ be a class diagram.

As we (now) are able to recognise OCL constraints when we see them, we can define

$$\text{Inv}(\mathcal{CD})$$

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

As usual: $\text{Inv}(\mathcal{D}) := \bigcup_{\mathcal{CD} \in \mathcal{D}} \text{Inv}(\mathcal{CD})$.

Principally clear: $\text{Inv}(\cdot)$ for any kind of diagram.
If $\mathcal{CD}$ consists of only $CD$ with the single class $C$, then

- $Inv(\mathcal{CD}) = Inv(CD) = \{\text{correct } C \text{ inv: } v > 3\}$
**Semantics of a Class Diagram**

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

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

Given a structure \( \mathcal{D} \) of \( \mathcal{S} \) (and thus of \( \mathcal{C}D \)), the class diagrams describe the system states \( \Sigma^\mathcal{D} \). Of those, some satisfy \( \mathcal{Inv}(\mathcal{C}D) \) and some don’t.
We call a system state \( \sigma \in \Sigma^\mathcal{D} \) consistent if and only if \( \sigma \models \mathcal{Inv}(\mathcal{C}D) \).

**In pictures:**

\[
\mathcal{C}D = \{ CD_1, \ldots, CD_n \}
\]

\[
\mathcal{S}(\mathcal{C}D) \rightarrow \mathcal{Inv}(\mathcal{C}D)
\]

\[
\mathcal{S}(\mathcal{C}D) \rightarrow \text{distinguish (classes and attributes)}
\]

\[
\mathcal{S}(\mathcal{C}D) \rightarrow \text{extend (visibility)}
\]

\[
\mathcal{S}(\mathcal{C}D) \rightarrow \sigma \in \Sigma^\mathcal{D}
\]

\[
\sigma \models \mathcal{Inv}(\mathcal{C}D)
\]
Recall: a UML model is an image or pre-image of a software system.

A set of class diagrams $\mathcal{CD}$ with invariants $\text{Inv}(\mathcal{CD})$ 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 $\text{Inv}(\mathcal{CD})$.

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

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

A set of class diagrams $\mathcal{CD}$ with invariants $\mathit{Inv}(\mathcal{CD})$ 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 $\mathit{Inv}(\mathcal{CD})$.

- **Post-image:** Dear user/maintainer, in the existing system, only system states which satisfy $\mathit{Inv}(\mathcal{CD})$ 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.
## Constraints vs. Types

### Find the 10 differences:

<table>
<thead>
<tr>
<th>$C$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x : \text{Int} \ {x = 3 \lor x &gt; 17}$</td>
<td>$x : T$</td>
</tr>
</tbody>
</table>

$\mathcal{D}(T) = \{3\} \cup \{n \in \mathbb{N} \mid n > 17\}$

- $x = 4$ 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).
Constraints vs. Types

Find the 10 differences:

<table>
<thead>
<tr>
<th>(C)</th>
<th>(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x: Int {x = 3 \lor x &gt; 17})</td>
<td>(x: T)</td>
</tr>
</tbody>
</table>

\[\mathcal{D}(T) = \{3\} \cup \{n \in \mathbb{N} \mid n > 17\}\]

- \(x = 4\) 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, 2005])

Be careful whose advice you buy, but, be patient with those who supply it.

Baz Luhrmann/Mary Schmich
Be good to your audience.
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?”

Main and General Modelling Guideline  (admittedly: 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?”

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

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

- **Implementation/A**
Main and General Quality Criterion  \(\text{(again: trivial and obvious)}\)

- **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) farther

- **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) 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**
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) farer

- **Implementation/A**
  - close to target platform
    
    
    \[ C_{0,1} \text{ is easy for Java, } C_\ast \text{ comes at a cost} \]
    
- **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
2.1 Readability

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

10. Include White-Space in Diagrams
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
2.2 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
5.1 General Guidelines

- 88. Indicate Visibility Only on Design Models (in contrast to analysis models)
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
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 *(or have good defaults)*
  
  - 118. Avoid Multiplicity “*”
  
  - 119. Replace Relationship Lines with Attribute Types
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**

(it depends, often is want to be)

- 133. Question Multiplicities Involving Minimums and Maximums
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 sunscreen.

Baz Luhrmann/Mary Schmich
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.

<table>
<thead>
<tr>
<th>Degrees of freedom:</th>
<th>Exemplary choice: 2D-Tron</th>
</tr>
</thead>
<tbody>
<tr>
<td>• simulation vs. arcade</td>
<td>arcade</td>
</tr>
<tr>
<td>• platform (SDK or not, open or proprietary, hardware capabilities...)</td>
<td>open</td>
</tr>
<tr>
<td>• graphics (3D, 2D, ...)</td>
<td>2D</td>
</tr>
<tr>
<td>• number of players, AI</td>
<td>min. 2, AI open</td>
</tr>
<tr>
<td>• controller</td>
<td>open (later determined by platform)</td>
</tr>
<tr>
<td>• game experience</td>
<td>minimal: main menu and game</td>
</tr>
</tbody>
</table>
Modelling Structure: 2D-Tron

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

```
<table>
<thead>
<tr>
<th>External inputs</th>
<th>Game Logic</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard, Joystick, ...</td>
<td>player scores, interface inputs/engine</td>
<td>Graphics (from ASCII to bitmap; native or via API)</td>
</tr>
<tr>
<td>(Physics) Engine</td>
<td>physical objects, collision notification</td>
<td>Sound, ...</td>
</tr>
</tbody>
</table>
```

update notify?
Modelling Structure: 2D-Tron

- Joystick?
- Keyboard?
- Control
  - Player
    - colour
    - score
    - direction
    - speed
- AI?
- Segment
  - x0, y0
  - x1, y1
  - colour
- Engine
  - areawidth
  - areaheight
- Tron
- Gameplay
  - update
  - notify
  - world
- Render
  - OpenGL?
  - aalib?

Conventions:
- default ξ is 1
References
References

