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 modeling guidelines? (Example?)
  - Discuss the style of this class diagram.

• Content:
  - Treat "the rest".
  - Where do we put OCL constraints?
  - Modeling guidelines, in particular for class diagrams (following [Ambler, 2005]).

Visibility

Notsosurprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by typing rules.

Question: given $CD^x: Int$ $\xi_{\text{role is the following OCL expression}}$ well-typed or not (wrt. visibility):

$$\text{context } C \text{ inv: self$.role$.x > 0}$$

Basically same rule as before: (analogously for other multiplicities)

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

Question: given $C_x$: Int $D_{role \times dist}$ is the following OCL expression well-typed or not (w.r.t. navigability):

```
context $D_{inv}$:
  self.role.x > 0
```

The standard says:
- $-\neg$: navigation is possible
- $\neg\triangleright$: navigation is efficient
- $\times$: navigation is not possible

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:

$$\langle r: \langle role_1: C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n: C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- Association name $r$ and rolenames/types $role_i/C_i$ 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.

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

$$\mu ::= \ast | N | N..M | N..\ast | \mu, \mu$$

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 $1 \leq i \leq 2k$, $N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{\ast\}$. 

Proposition: Consider a graph $G = (V, E)$ with a source $s$ and a sink $t$. The graph $G$ is strongly connected if and only if the set $S = \{s\} \cup \{v \in V : \text{in-degree}(v) = 0\}$ is equal to the set $T = \{t\} \cup \{v \in V : \text{out-degree}(v) = 0\}$.

Theorem: Consider a graph $G = (V, E)$ with a source $s$ and a sink $t$. The graph $G$ is strongly connected if and only if the set $S = \{s\} \cup \{v \in V : \text{in-degree}(v) = 0\}$ is equal to the set $T = \{t\} \cup \{v \in V : \text{out-degree}(v) = 0\}$.
• violated.

but: checks for array bounds everywhere...?

3: µ

\[ \mu \text{ true} = \mu \text{ true} \]

problem with our approach, we have anyway.

could be represented by a set data-structure type without fixed bounds—no

many programming languages have direct correspondences (the first corresponds

More precise, can't we just use (cf. Slide 26)
Multiplicities as Constraints...

Why Multiplicities as Constraints?

How do multiplicities affect class diagrams?

More precise, can't we just use multiplicities?

Principally acceptable — but: few programming languages/data structure libraries allow lower bounds for arrays (other than 0).

Languages/data structure libraries allow lower bounds for arrays (other than 0). How do we determine the correct bounds?

Well, if the target platform has, for instance, a fixed bound of 5, we could simply restrict the source platform to that bound.

If we have a model, we'll get a runtime error, and thereby see that the constraint is violated.

But: checks for array bounds everywhere? And don't think about constraints that way.

What about range-checked arrays with positions 0..N?

In general, properties are not well-understood.

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.

In OCL, any property and operation is either an invariant or a restriction.

Properties

Invariants

Restrictions

From now on, we will only discuss OCL.
Ownership

• role

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).

– 09 – 2012 -11-27 – Sassocrest – 15/42

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

\[
\begin{align*}
(i) & \quad C \rightarrow \rightarrow 0 \cdots 1 \times \\
(ii) & \quad C \rightarrow \rightarrow \ast \times
\end{align*}
\]

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

• Form \( (i) \) introduces 'role' \( \in \sigma \ (C) \).

• In both cases, 'role' belongs to \( \tau (C) \).

• We drop \( \lambda \) and go back to our nice \( \tau \) with \( \tau (u)(\text{'role'}) \subseteq \tau (D)(\tau (D)) \).

OCL Constraints in (Class) Diagrams

Where Shall We Put OCL Constraints?

Numerous options:

(i) Additional documents.

(ii) Notes.

(iii) Particular dedicated places.

– 09 – 2012 -11-27 – Socldia – 19/42
Where Shall We Put OCL Constraints?

Numerous options:
(i) Additional documents.
(ii) Notes.
(iii) Particular dedicated places.

Notes:

AUML 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 – 09 – 2012 -11-27 – Socldia – 19/42

OCL in Notes: Conventions

For simplicity, we view the above as an abbreviation for

\[
\xi v: \tau \{ p_1, \ldots, p_n \} \{ \text{expr} \} \\
\xi f(v_1: \tau, \ldots, v_n: \tau): \tau \{ p_1, \ldots, p_n \} \{ \text{pre: expr}_1, \text{post: expr}_2 \}
\]

InvariantsofaClassDiagram

• Let \( CD \) be a class diagram.

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

\[ \text{Inv}(CD) = \{ \phi_1, \ldots, \phi_n \} \]

of OCL constraints occurring in notes in \( CD \) — after unfolding all abbreviations (cf. next slides).

• As usual:

\[ \text{Inv}(\cdot) := \bigcup_{CD \in \text{CD}} \text{Inv}(CD) \]

• Principally clear: \( \text{Inv}(\cdot) \) for any kind of diagram.
Find the 10 differences:

\[
\text{Pragmatics:}
\]

**Rule-of-thumb:**

If something is a correspondence in the application domain, then make it a type.

- **Example:**
  - Suppose a structure describesthe
    desirability
  - \( \text{CD} \) isnot
    superfluous
  - \( \text{Inv} \) isnot
    mandatory
  - \( \text{S} \) isnot
    bijective

Right in the domain:

\[
\text{Def}\text{inition.}
\]

Wesay, the

\[
\text{CD} \text{is}
\]

an image or pre-image ofa softwaresystem.

\[
\text{Post-image}
\]

\[
\text{Pre-image}
\]

\[
\text{In pictures}
\]

\[
\text{Constraint vs. Types}
\]

\[
\text{Pragmatics}
\]

\[
\text{Invariants in ClassDiagramExample}
\]

\[
\text{Constraint vs. Types}
\]
Design Guidelines for (Class) Diagram

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

Baz Luhrmann/Mary Schmich

(partly following [Ambler, 2005])

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?
  (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 $D$?

• the things less relevant for $T$, do they disturb in $D$?

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

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 ($C_0$, $C_1$ is easy for Java, $C^*$ comes at a cost—other way around 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”
2.1 Readability

1. Support Readability of Lines
2. 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

2.3 Naming

20. Set and (23. Consistently) Follow Effective Naming Conventions
2.2 Simplicity
- Show Only What You Have to Show
- Prefer Well-Known Notation over Exotic Notation
- Large vs. Small Diagrams
- Content First, Appearance Second

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

2.4 General
- Indicate Unknowns with Question-Marks
- Consider Applying Color to Your Diagram
- Apply Color Sparingly

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

5.2 Class Style Guidelines
- Prefer Complete Singular Nouns for Class Names
- Name Operations with Strong Verbs
- Do Not Model Scaffolding Code (except for exceptions)
- Never Show Classes with Just Two Compartments
- Label Uncommon Class Compartments
- Include an Ellipsis (...) at the End of an Incomplete List
- List Operations/Attributes in Order of Decreasing Visibility

5.3 Relationships
- Model Relationships Horizontally
- Model a Dependency When the Relationship is Transitory
- Always Indicate the Multiplicity
- Avoid "Multiplicity" "\ast"
- Replace Relationship Lines with Attribute Types

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

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

5.6 Aggregation and Composition

Exercises

---

Example: Modelling Games

Task: Game Development

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

Exemplary choice: 2D - Tron

- Simulation vs. Arcade
- Platform open
- 2D
- Min. 2, AI open
- Controller open (later determined by platform)
- Only game, no menus

In many domains, there are canonical architectures – and an adept reader tries to see/find/match this!

For games:

Main External inputs

- Keyboard
- Joystick
- ...

Game Logic

- Players scores
- Interface inputs/engine (Physics) Engine
- Physical objects
- Collision notification

Output

- Graphics (from ASCII to bitmap; native or via API)
- Sound
- ...

---