Software Design, Modelling and Analysis in UML

Lecture 07: Class Diagrams II

2012-11-14

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lectures:

- class diagram — except for associations; visibility within OCL type system

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Please explain this class diagram with associations.
  - Which annotations of an association arrow are semantically relevant?
  - What's a role name? What's it good for?
  - What's "multiplicity"? How did we treat them semantically?
  - What is "reading direction", "navigability", "ownership", . . . ?
  - What's the difference between "aggregation" and "composition"?
- **Content:**
  - Complete visibility
  - Study concrete syntax for "associations".
  - (**Temporarily**) extend signature, define mapping from diagram to signature.
  - Study effect on OCL.
  - Where do we put OCL constraints?

Casting in the Type System

One Possible Extension: Implicit Casts

- We **may wish** to have

$$\vdash 1 \text{ and } \textit{false} : \mathit{Bool} \qquad (*)$$

  In other words: We may wish that the type system allows to use $0, 1 : \mathit{Int}$ instead of *true* and *false* without breaking well-typedness.

- Then just have a rule:

$$(\mathit{Cast}) \quad \frac{A \vdash \mathit{expr} : \mathit{Int}}{A \vdash \mathit{expr} : \mathit{Bool}}$$

- With (Cast) (and (Int), and (Bool), and (Fun_0)), we can derive the sentence $(*)$, thus conclude well-typedness.

- **But**: that's only half of the story — the definition of the interpretation function I that we have is not prepared, it doesn't tell us what $(*)$ means...

  $I(\text{and}) : I(\text{Bool}) \times I(\text{Bool}) \to I(\text{Bool})$

Implicit Casts Cont'd

So, why isn't there an interpretation for $(1$ and *false*$)$?

- First of all, we have (syntax)

$$expr_1 \text{ and } expr_2 : Bool \times Bool \to Bool$$

- Thus,

$$I(\text{and}) : I(Bool) \times I(Bool) \to I(Bool)$$

  where $I(Bool) = \{\textit{true}, \textit{false}\} \cup \{\bot_{Bool}\}$.

- By definition,

$$I[\![1 \text{ and } \textit{false}]\!](\sigma, \beta) = I(\text{and})(\quad I[\![1]\!](\sigma, \beta), \quad I[\![\textit{false}]\!](\sigma, \beta) \quad),$$

  and **there we're stuck**.

Implicit Casts: Quickfix

- Explicitly define

$$I[\![\text{and}(expr_1, expr_2)]\!](\sigma, \beta) := \begin{cases} b_1 \wedge b_2 & \text{, if } b_1 \neq \bot_{Bool} \neq b_2 \\ \bot_{Bool} & \text{, otherwise} \end{cases}$$

  where

  - $b_1 := toBool(I[\![expr_1]\!](\sigma, \beta))$,
  - $b_2 := toBool(I[\![expr_2]\!](\sigma, \beta))$,

  and where

$$toBool : I(Int) \cup I(Bool) \to I(Bool)$$

$$x \mapsto \begin{cases} \textit{true} & \text{, if } x \in \{true\} \cup I(Int) \setminus \{0, \bot_{Int}\} \\ \textit{false} & \text{, if } x \in \{false, 0\} \\ \bot_{Bool} & \text{, otherwise} \end{cases}$$

Bottomline

- There are **wishes** for the type-system which require changes in both, the definition of I **and** the type system.
  In most cases not difficult, but tedious.

- **Note**: the extension is still a basic type system.

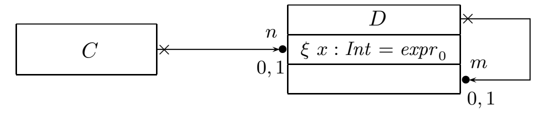
- **Note**: OCL has a far more elaborate type system which in particular addresses the relation between $Bool$ and Int (cf. [?]).

Visibility in the Type System

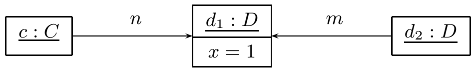
Visibility — The Intuition

$$\mathscr{S} = (\{Int\}, \{C, D\}, \{n : D_{0,1}, m : D_{0,1}, \langle x : Int, \xi, expr_0, \emptyset\rangle\}, \{C \mapsto \{n\}, D \mapsto \{x, m\}\}$$

Let's study an **Example**:



and

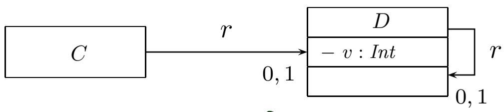


Assume $w_1 : \tau_C$ and $w_2 : \tau_D$ are logical variables. **Which** of the following **syntactically correct** (?) OCL expressions **shall** we consider to be **well-typed**?

ξ of x:	public	private	protected	package
$w_1 \,.\, n \,.\, x = 0$	✔ ← ✘ ?	✔ I ✘ III ← ? -	later	not
		privateness is by class, and not by object		
$w_2 \,.\, m \,.\, x = 0$	✔ ← ✘ ?	✔ ← ✘ |||| ?	later	not

Context

- **Example**: A problem?



D-object D-object

$$self : \tau_D \vdash self \,.\, r \,.\, v > 0 \quad ✓$$

$$self : \tau_C \nvdash self \,.\, r \,.\, v > 0 \quad ✗$$

- That is, whether an expression involving attributes with visibility is well-typed **depends** on the class of objects for which it is evaluated.
- **Therefore**: well-typedness in type environment A and **context** $B \in \mathscr{C}$:

$$A, B \vdash expr : \tau$$

order doesn't matter

- In particular: prepare to treat "protected" later (when doing inheritance).

Attribute Access in Context

- If $expr$ is of type τ in a type environment, then it is in **any context**:

B,

$$(Context\cancel{Intro}) \quad \frac{A \vdash expr : \tau}{A, \boxed{B} \vdash expr : \tau}$$

Drop

- **Accessing attribute** v of a C-object via logical variable w is well-typed if
  - ~~v is public, or w is of type τ_B~~

$$(Attr_1) \quad \frac{A \vdash w : \tau_B}{A, B \vdash v(w) : \tau}, \quad \langle v : \tau, \xi, expr_0, P_{\mathscr{C}} \rangle \in atr(B)$$

- **Accessing attribute** v of a C-object of via expression $expr_1$ is well-typed **in context** B if
  - v is public, **or** $expr_1$ denotes an object of class B:

$$(Attr_2) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash v(expr_1) : \tau}, \quad \begin{array}{l} \langle v : \tau, \xi, expr_0, P_{\mathscr{C}} \rangle \in atr(C), \\ \xi = +, \text{ or } C = B \end{array}$$

- Acessing $C_{0,1}$- or C_*-typed attributes: similar.

Context in Operator Application

- Operator Application:

$$(Fun_2) \quad \frac{A, B \vdash expr_1 : \tau_1 \ \ldots \ A, B \vdash expr_n : \tau_n}{A, B \vdash \omega(expr_1, \ldots, expr_n) : \tau}, \quad \begin{array}{l} \omega : \tau_1 \times \cdots \times \tau_n \to \tau, \\ n \geq 1, \ \omega \notin atr(\mathscr{C}) \end{array}$$

- Iterate:

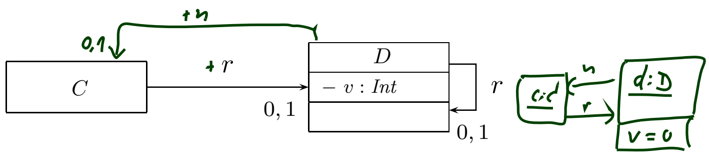
$$(Iter_1) \quad \frac{A, B \vdash expr_1 : Set(\tau_1) \quad A, B \vdash expr_2 : \tau_2 \quad A', B \vdash expr_3 : \tau_2}{A, B \vdash expr_1\text{->iterate}(w_1 : \tau_1 \ ; \ w_2 : \tau_2 = expr_2 \mid expr_3) : \tau_2}$$

where $A' = A \oplus (w_1 : \tau_1) \oplus (w_2 : \tau_2)$

Attribute Access in Context Example

$$(Context\cancel{Intro})\ \text{Drop} \quad \frac{B,\ A \vdash expr : \tau}{A, \cancel{B} \vdash expr : \tau}$$

$$(Attr_2) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash v(expr_1) : \tau}, \quad \langle v : \tau, \xi, expr_0, P_{\mathscr{C}} \rangle \in atr(C),\ \xi = +, \text{ or } \xi = - \text{ and } C = B$$



Example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{self : \tau_D \in A}{A \vdash self : \tau_D}(Var)}{D, self : \tau_D \vdash n(self) : \tau_C}(Attr_2)}{D, self : \tau_D \vdash v(n(self)) : \tau_D}(Attr_2)}{D, self : \tau_D \vdash v(r(n(self))) : Int}(Attr_2) \qquad \cancel{\dfrac{\dfrac{\dfrac{}{\vdash 0}(Int)}{A \vdash 0}(EnvIntro)}{D, self : \tau_D \vdash 0}(ContextIntro)}}{\cancel{D,}\ \underbrace{self : \tau_D}_{=: A} \vdash self.n.r.v > 0}$$

$A \vdash v(r(n(self))) : Int$

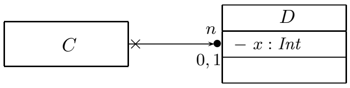
$>(v(r(n(self))), 0)$

The Semantics of Visibility

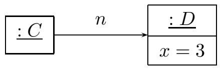
- **Observation**:
  - Whether an expression **does** or **does not** respect visibility is a matter of well-typedness **only**.
  - We only evaluate (= apply I to) **well-typed** expressions.

$\longrightarrow$ We **need not** adjust the interpretation function I to support visibility.

What is Visibility Good For?



- Visibility is a property of attributes — is it useful to consider it in OCL?



- In other words: given the picture above, **is it useful** to state the following invariant (even though x is private in D)

$$\text{context } C \text{ inv} : n.x > 0 \text{ ?}$$

- **It depends**. (cf. [?], Sect. 12 and 9.2.2)
  - **Constraints and pre/post conditions**:
    - Visibility is **sometimes** not taken into account. To state "global" requirements, it may be adequate to have a "global view", be able to look into all objects.
    - But: visibility supports "narrow interfaces", "information hiding", and similar good design practices. To be more robust against changes, try to state requirements only in the terms which are visible to a class.

    Rule-of-thumb: if attributes are important to state requirements on design models, leave them public or provide get-methods (later).
  - **Guards** and **operation bodies**:
    If in doubt, **yes** (= do take visibility into account).

    Any so-called **action language** typically takes visibility into account.

Recapitulation

Recapitulation

Class Diagrams $\mathscr{CD}$

$\Big\downarrow$ **induces**

extended (!) signature $\mathscr{S}(\mathscr{CD})$

$\Big\downarrow$ **gives rise to**

Basic Type System

$true + 3$

$\xrightarrow{\text{well typed}} I_1[\![true + 3]\!]$, using toInt

$\xrightarrow{\text{abbrev.}} toInt(true) + 3$

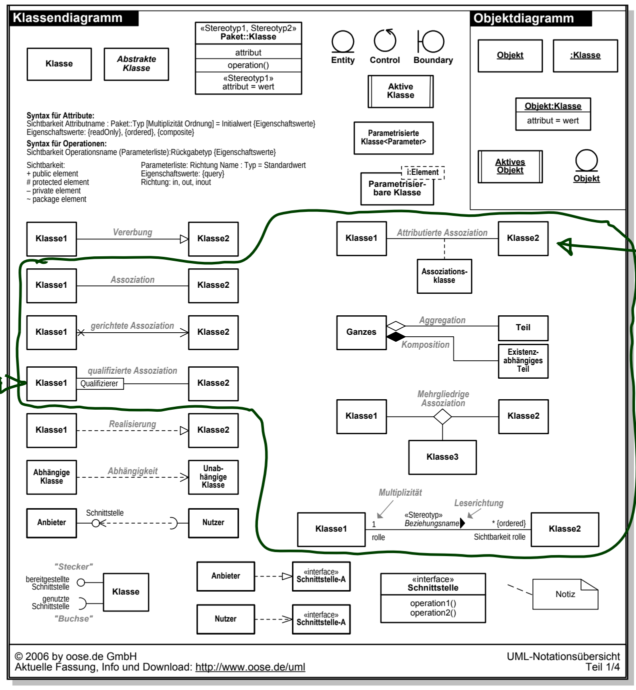
$\xrightarrow{\text{well typed}} I_2[\![+]\!](I[\![toInt]\!](I[\![true]\!]), I[\![3]\!])$

- We extended the type system for
  - **casts** (requires change of I) and
  - **visibility** (no change of I).
- **Later**: **navigability** of associations.

Good: well-typedness is decidable for these type-systems. That is, we can have automatic tools that check, whether OCL expressions in a model are well-typed.

Associations: Syntax

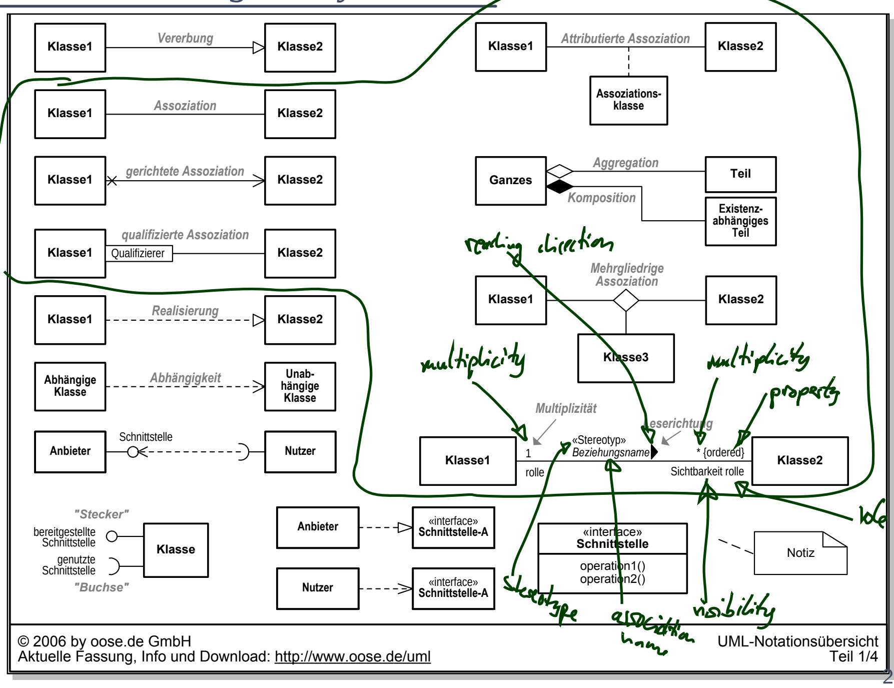
UML Class Diagram Syntax [?]



NOT in lecture

NOT in lecture

UML Class Diagram Syntax [?]



UML Class Diagram Syntax [?, 61;43]

ownership

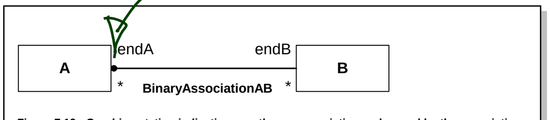


Figure 7.19 - Graphic notation indicating exactly one association end owned by the association

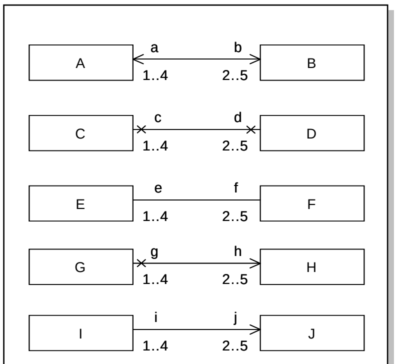


Figure 7.23 - Examples of navigable ends

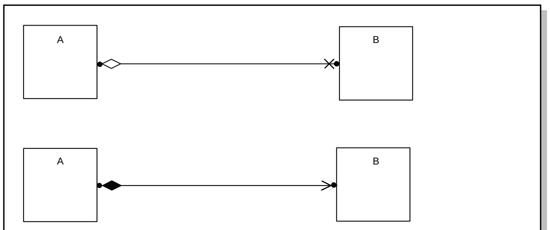


Figure 7.20 - Combining line path graphics

What Do We (Have to) Cover?

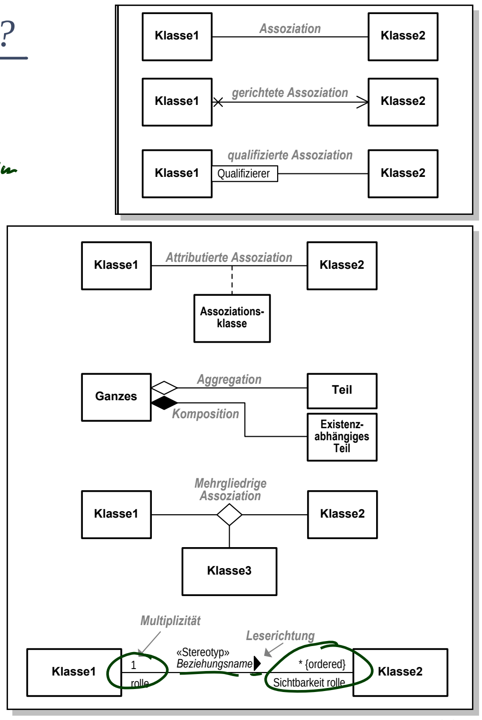
An **association** has

- a **name**,
- a **reading direction**, and just a hint to the reader of the diagram
- at least two **ends**.

Each **end** has

- a **role name**,
- a **multiplicity**,
- a set of **properties**, such as **unique**, **ordered**, etc.
- a **qualifier**, (we will not treat)
- a **visibility**,
- a **navigability**,
- an **ownership**,
- and possibly a **diamond**. (exercises)

Wanted: places in the signature to represent the information from the picture.



(Temporarily) Extend Signature: Associations

Only for the course of Lectures 07/08 we assume that each attribute in V

- **either** is $\langle v : \tau, \xi, expr_0, P_v \rangle$ with $\tau \in \mathscr{T}$ (as before),
- **or** is an **association** of the form

$$\begin{aligned} \langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \\ \vdots \\ \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle \end{aligned}$$

  where
  - $n \geq 2$ (at least two ends),
  - r, $role_i$ are just **names**, $C_i \in \mathscr{C}$, $1 \leq i \leq n$,
  - the **multiplicity** μ_i is an expression of the form

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu_1, \mu_2 \qquad (N, M \in \mathbb{N})$$

  - P_i is a set of **properties** (as before),
  - $\xi \in \{+, -, \#, \sim\}$ (as before),
  - $\nu_i \in \{\times, -, >\}$ is the **navigability**,
  - $o_i \in \mathbb{B}$ is the **ownership**.

– 07 – 2012-11-14 – Sassocsyn –

(Temporarily) Extend Signature: Associations

Only for the course of Lectures 07/08 we assume that each attribute in V

- **either** is $\langle v : \tau, \xi, expr_0, P_v \rangle$ with $\tau \in \mathscr{T}$ (as before),
- **or** is an **association** of the form

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle,$$

Alternative syntax for multiplicities:

N..N e.g.

$$\mu ::= N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N} \cup \{*\})$$

0..* e.g. or 1..*

and define $*$ and N as abbreviations.

Note: N could abbreviate $0..N$, $1..N$, or $N..N$. We use last one.

  - r, $role_i$ are just **names**, $C_i \in \mathscr{C}$, $1 \leq i \leq n$,
  - the **multiplicity** μ_i is an expression of the form

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

  - P_i is a set of **properties** (as before),
  - $\xi \in \{+, -, \#, \sim\}$ (as before),
  - $\nu_i \in \{\times, -, >\}$ is the **navigability**,
  - $o_i \in \mathbb{B}$ is the **ownership**.

– 07 – 2012-11-14 – Sassocsyn –

(Temporarily) Extend Signature: Basic Type Attributes

Also only for the course of this lectures 07/08

- we only consider **basic type attributes** to "belong" to a class (to appear in $atr(C)$),
- **associations** are not "owned" by a particular class (do not appear in $atr(C)$), but live on their own.

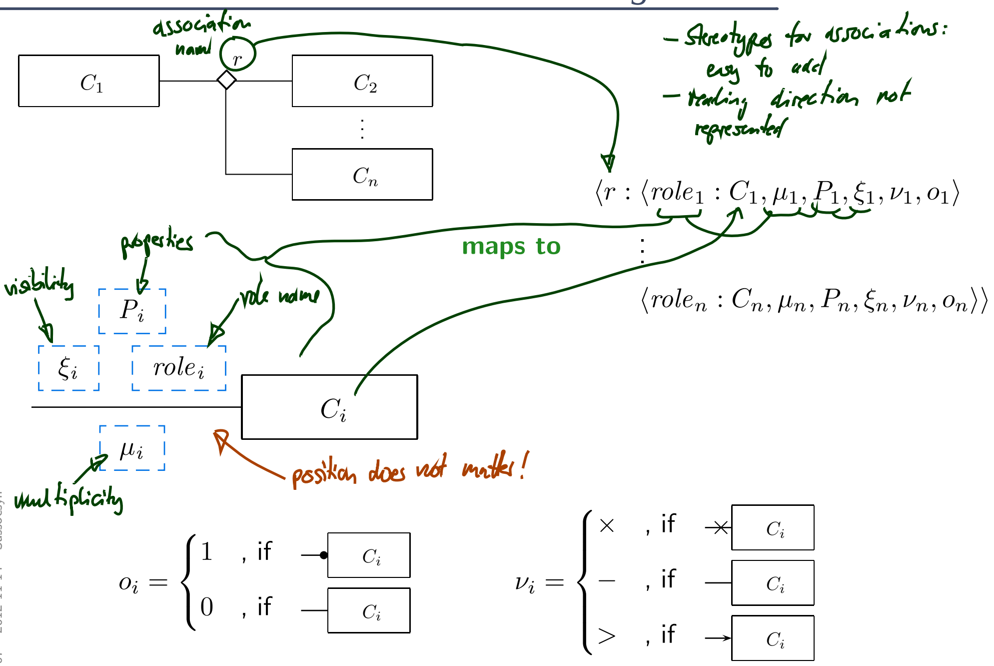
Formally: we only call

$$(\mathscr{T}, \mathscr{C}, V, atr)$$

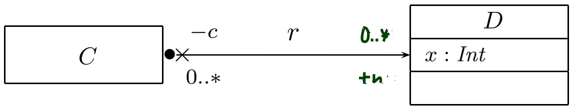
a **signature (extended for associations)** if

$$atr : \mathscr{C} \to 2^{\{v \in V \mid v:\tau, \tau \in \mathscr{T}\}}.$$

From Association Lines to Extended Signatures



Association Example



Signature:

always use unique here – later

$$\mathscr{S} = (\{Int\}, \{C, D\}, \{x : Int, \langle r : \langle c : C, 0..*, \{unique\}, -, \times, 1\rangle, \langle n : D, 0..*, \{unique\}, +, >, 0\rangle\rangle\}, \{C \mapsto \{\}, D \mapsto \{x\}\})$$

only basic type attributes here!

What If Things Are Missing?

Most components of associations or association end may be omitted.
For instance [?, 17], Section 6.4.2, proposes the following rules:

- **Name**: Use

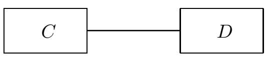
  $$A\_\langle C_1\rangle\_\cdots\_\langle C_n\rangle$$

  if the name is missing.

  Example:

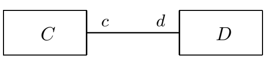
  

  for

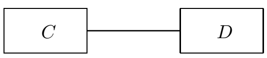
  

- **Reading Direction**: no default.
- **Role Name**: use the class name at that end in lower-case letters

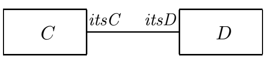
  Example:

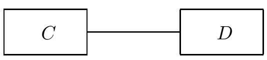
  for

  Other convention: (used e.g. by modelling tool Rhapsody)

  for

What If Things Are Missing?

- **Multiplicity**: 1
  In my opinion, it's safer to assume $0..1$ or $*$ 0..* if there are no fixed, written, agreed conventions ("expect the worst").
- **Properties**: $\emptyset$ (here: {unique})
- **Visibility**: public
- **Navigability and Ownership**: not so easy. [?, 43]

"Various options may be chosen for showing navigation arrows on a diagram.

In practice, it is often convenient to suppress some of the arrows and crosses and just show exceptional situations:

- *Show all arrows and x's. Navigation and its absence are made completely explicit.*
- *Suppress all arrows and x's. No inference can be drawn about navigation. This is similar to any situation in which information is suppressed from a view.*
- *Suppress arrows for associations with navigability in both directions, and show arrows only for associations with one- way navigability.*

  In this case, the two-way navigability cannot be distinguished from situations where there is no navigation at all; however, the latter case occurs rarely in practice."

Wait, If Omitting Things...

- ...**is causing so much trouble** (e.g. leading to misunderstanding), why does the standard say "**In practice, it is often convenient**..."?

  Is it a good idea to trade **convenience** for **precision/unambiguity**?

  It depends.
  - Convenience as such is a legitimate goal.
  - In UML-As-Sketch mode, precision "doesn't matter", so convenience (for writer) can even be a primary goal.
  - In UML-As-Blueprint mode, **precision** is the **primary goal**. And misunderstandings are in most cases annoying.

    But: (even in UML-As-Blueprint mode)
    If all associations in your model have multiplicity $*$,
    then it's probably a good idea not to write all these $*$'s.
    So: tell the reader about it and leave out the $*$'s.

References