Software Design, Modelling and Analysis in UML

Lecture 7: Class Diagrams II

2015-11-17

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Representing class diagrams as (extended) signatures — for the moment without associations: later.

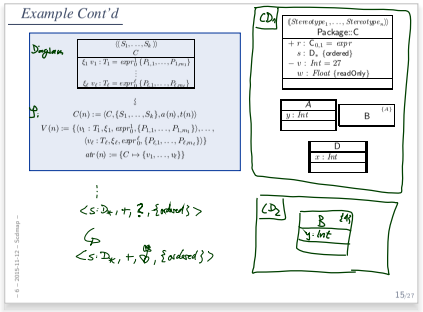
This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Could you please map this class diagram to a signature?
  - What if things are missing?
  - Could you please map this signature to a class diagram?
  - What is the semantics of 'abstract'?
  - What is visibility good for?
- **Content:**
  - Map class diagram to (extended) signature cont'd.
  - Stereotypes – for documentation.
  - Visibility as an extension of well-typedness.

Mapping UML CDs to Extended Signatures

Recall

Example Cont'd



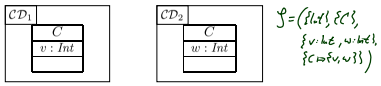
Is the Mapping a Function?

Question: Is $\mathscr{S}(\mathscr{CD})$ **well-defined**?

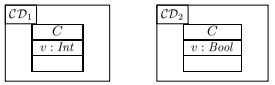
There are two possible **sources for problems**:

(1) A **class** C may appear in **multiple** class **diagrams**:

(i)



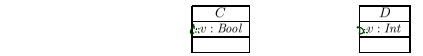
(ii)



Simply **forbid** the case (ii) — easy syntactical check on diagram.

Is the Mapping a Function?

(2) An **attribute** v may appear in **multiple classes** with different type:



Two approaches:

- Require **unique** attribute names.
  This requirement can easily be established (implicitly, behind the scenes) by viewing v as an abbreviation for

  $$C{::}v \quad \text{or} \quad D{::}v$$

  depending on the context. ($C{::}v : Bool$ and $D{::}v : Int$ are then unique.)
- Subtle, formalist's approach: observe that

  $$\langle v : Bool, \ldots \rangle \quad \text{and} \quad \langle v : Int, \ldots \rangle$$

  are **different things** in V. We don't follow that path...

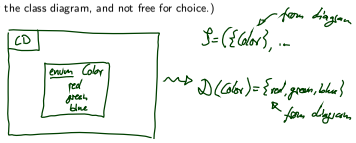
Class Diagram Semantics

Semantics

The semantics of a set of **class diagrams** $\mathscr{CD}$ is the induced **signature** $\mathscr{S}(\mathscr{CD})$.

The **signature** induces a set of **system states** $\Sigma^{\mathscr{D}}_{\mathscr{S}}$ (given a **structure** $\mathscr{D}$).

- Do we need to redefine/extend $\mathscr{D}$? No.
  (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type T, i.e. the set $\mathscr{D}(T)$, would be determined by the class diagram, and not free for choice.)



Semantics

The semantics of a set of **class diagrams** $\mathscr{CD}$ is the induced **signature** $\mathscr{S}(\mathscr{CD})$.

The **signature** induces a set of **system states** $\Sigma^{\mathscr{D}}_{\mathscr{S}}$ (given a **structure** $\mathscr{D}$).

- Do we need to redefine/extend $\mathscr{D}$? **No.**
  (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type T, i.e. the set $\mathscr{D}(T)$, would be determined by the class diagram, and not free for choice.)
- What is the effect on $\Sigma^{\mathscr{D}}_{\mathscr{S}}$? **Little.**
  For now, we only **remove** abstract class instances, i.e.

$$\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_a)))$$

  (abstract)

  is now **only** called **system state** if and only if, for all $\langle C, S_C, 1, t\rangle \in \mathscr{C}$,

$$\underbrace{\mathrm{dom}(\sigma) \cap \mathscr{D}(C) = \emptyset}.$$

  With $a = 0$ as default "abstractness", the earlier definitions apply directly.
  (We'll revisit this when discussing inheritance.)

What About The Rest?

- **Classes**:
  - **Active**: not represented in σ.
    Later: relevant for behaviour, i.e., how system states evolve over time.
  - **Stereotypes**: in a minute.
- **Attributes**:
  - **Initial value expression**: not represented in σ.
    Later: provides an initial value as effect of "creation action".
  - **Visibility**: not represented in σ.
    Later: viewed as additional **typing information** for well-formedness of actions; and with inheritance.
  - **Properties**: such as `readOnly`, `ordered`, `composite` (Deprecated in the standard.)
    - `readOnly` — later treated similar to visibility.
    - `ordered` — not considered in our UML fragment (→ sets vs. sequences).
    - `composite` — cf. lecture on associations.

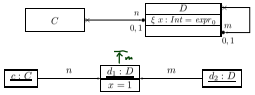
Rhapsody Demo I

RECALL: SEND ME YOUR POOL-ACCOUNT NAME
(meyerp, NOT: ab124, xh702 (R2))

Visibility

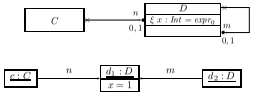
The Intuition by Example

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



The Intuition by Example

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



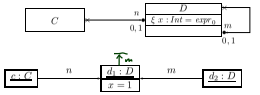
Assume $\underbrace{w_1 : \tau_C}$ and $\underbrace{w_2 : \tau_D}$ are logical variables.

Which of the following **syntactically correct** (?) OCL expressions **should** we consider to be **well-typed**?

ξ of x:	public	private	protected	package
$w_1 \,.\, n \,.\, x = 0$			later	not
$w_2 \,.\, m \,.\, x = 0$			later	not

The Intuition by Example

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



Assume $\underbrace{w_1 : \tau_C}$ and $\underbrace{w_2 : \tau_D}$ are logical variables.

Which of the following **syntactically correct** (?) OCL expressions **should** we consider to be **well-typed**?

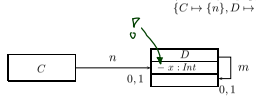
ξ of x:	public	private	protected	package
$w_1 \,.\, n \,.\, x = 0$	✓ Int ✗ ξ ? ξ	✓ Int ✗ Int ? ξ	later	not
$w_2 \,.\, m \,.\, x = 0$	✓ Int ✗ ?	✓ ✗ Int ! ⇐ by object ?	later	not

by class (OCL, C++, Java, ~)

Context

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

- **By example**:



$\underbrace{self_D}_{:\tau_D} . x > 0$ ✓

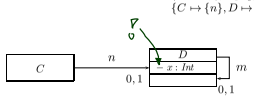
$\underbrace{self_D . m}_{:\tau_D} . x > 0$ ✓

$\underbrace{self_C . n}_{:\tau_C} . x > 0$ ✗

Context

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

- **By example**:



$\underbrace{self_D}_{:\tau_D} . x > 0$ ✓

$\underbrace{self_D . m}_{:\tau_D} . x > 0$ ✓

$\underbrace{self_C . n}_{:\tau_C} . x > 0$ ✗

- That is, whether an expression involving attributes with visibility is well-typed **depends** on the class of objects for which it is evaluated.
- Visibility is '**by class**' — **not** 'by object'.

Attribute Access in Context

Recall: attribute access in OCL Expressions, $C, D \in \mathscr{C}$.

$v(expr_1)$	$: \tau_C \to$ ~~Set~~ T	• $v : T \in atr(C)$, $T \in \mathscr{T}$,
$r_1(expr_1)$	$: \tau_C \to \tau_D$	• $r_1 : D_{0,1} \in atr(C)$,
$r_2(expr_1)$	$: \tau_C \to Set(\tau_D)$	• $r_2 : D_* \in atr(C)$,

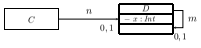
New rules for well-typedness **considering visibility**:

- $v(w)$ $: \tau_C \to T$ $\quad w : \tau_C, \quad v : T \in atr(C), T \in \mathscr{T}$
- $r_1(w)$ $: \tau_C \to \tau_D$ $\quad w : \tau_C, \quad r_1 : D_{0,1} \in atr(C)$
- $r_2(w)$ $: \tau_C \to Set(\tau_D)$ $\quad w : \tau_C, \quad r_1 : D_* \in atr(C)$

- $v(expr_1(w))$ $: \tau_C \to T$ $\quad \langle v : T, \xi, expr_0, P \rangle \in atr(C), T \in \mathscr{T},$
  $\underbrace{expr_1(w)}_{: \tau_C}$, $\underbrace{w : \tau_{C_1} \text{ and } C_1 = C,}$ or $\underbrace{\xi = +}$
- $r_1(expr_1(w))$ $: \tau_C \to \tau_D$ $\quad \langle r_1 : D_{0,1}, \xi, expr_0, P \rangle \in atr(C),$
  $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C,$ or $\xi = +$
- $r_2(expr_1(w))$ $: \tau_C \to Set(\tau_D)$ $\quad \langle r_2 : D_*, \xi, expr_0, P \rangle \in atr(C),$
  $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C,$ or $\xi = +$

Example

(i) $v(w)$	$: \tau_C \to T$	$w : \tau_C, \quad v : T \in atr(C), T \in \mathscr{T}$
(ii) $r_1(w)$	$: \tau_C \to \tau_D$	$w : \tau_C, \quad r_1 : D_{0,1} \in atr(C)$
(iii) $v(expr_1(w))$	$: \tau_C \to T$	$\langle v : T, \xi, expr_0.P \rangle \in atr(C), T \in \mathscr{T},$ $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C, \quad$ or $\xi = +$
(iv) $r_1(expr_1(w))$	$: \tau_C \to \tau_D$	$\langle r_1 : D_{0,1}, \xi, expr_0.P \rangle \in atr(C),$ $expr_1(w) : \tau_C, \quad w : \tau_{C_1}$ and $C_1 = C, \quad$ or $\xi = +$



- $self_D \,.\, x > 0$
- $self_D \,.\, m \,.\, x > 0$
- $self_C \,.\, n \,.\, x > 0$

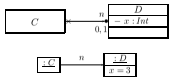
The Semantics of Visibility

- **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.

→ We **need not** adjust the interpretation function I to support visibility.

Just decide: should we take visibility into account yes / no, and check well-typedness by the new / old rules.

What is Visibility Good For?



- Visibility is a property of attributes — is it useful to consider it in OCL?
- In other words: given the diagram above, **is it useful** to state the following invariant (even though x is private in D)

context C inv : $n.x > 0$?

It depends. (cf. OMG (2006), 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", i.e. 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.

Stereotypes

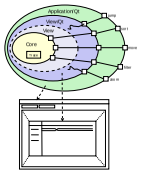
Stereotypes as Labels or Tags

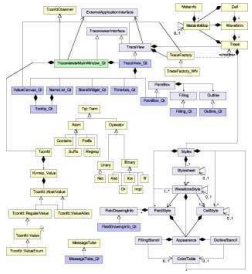
- What are Stereotypes?
  - **Not** represented in system states.
  - **Not** contributing to typing rules / well-formedness.

- Oestereich (2006):
  View stereotypes as (additional) "**labelling**" ("tags") or as "**grouping**".
- Useful for documentation and model-driven development, e.g. code-generation:
  - **Documentation**: e.g. layers of an architecture.

    Sometimes, packages (cf. OMG (2011a,b)) are sufficient and "right".
  - **Model Driven Architecture** (MDA): **later**.

Example: Stereotypes for Documentation





- **Example**: Timing Diagram Viewer Schumann et al. (2008)
- Architecture has four layers:
  - core, data layer
  - abstract view layer
  - toolkit-specific view layer/widget
  - application using widget

Stereotype "=" layer "=" colour.

Other Examples

- Use stereotypes 'Team$_1$', 'Team$_2$', 'Team$_3$' and assign stereotype Team$_i$ to class C if Team$_i$ is responsible for class C.
- Use stereotypes to label classes with licensing information (e.g., LGPL vs. proprietary).
- Use stereotypes 'Server$_A$', 'Server$_B$' to indicate where objects should be stored.
- Use stereotypes to label classes with states in the development process like "under development", "submitted for testing", "accepted".
- etc. etc.

Necessary: a **common idea** of what each stereotype stands for.

(To be defined / agreed on by the team, not the job of the UML consortium.)

References

References

Oestereich, B. (2006). *Analyse und Design mit UML 2.1, 8. Auflage*. Oldenbourg, 8. edition.

OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.

Schumann, M., Steinke, J., Deck, A., and Westphal, B. (2008). Traceviewer technical documentation, version 1.0. Technical report, Carl von Ossietzky Universität Oldenburg und OFFIS.