Software Design, Modelling and Analysis in UML

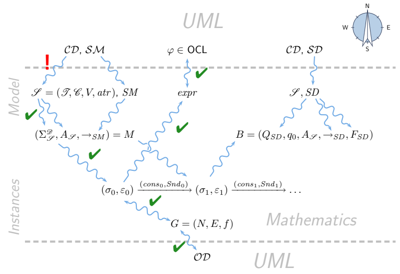
Lecture 05: Class Diagrams I

2011-11-15

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

Albert-Ludwigs-Universität Freiburg, Germany

Course Map



Contents & Goals

Last Lecture:

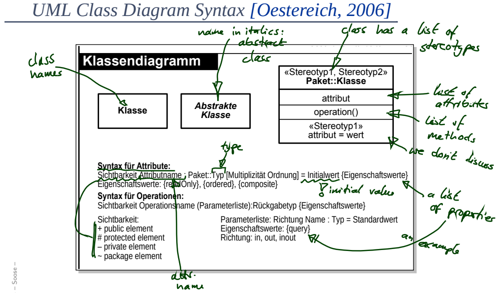
- OCL Semantics
- Object Diagrams

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a class diagram?
  - For what purposes are class diagrams useful?
  - Could you please map this class diagram to a signature?
  - Could you please map this signature to a class diagram?
- **Content:**
  - Study UML syntax.
  - Prepare (extend) definition of signature.
  - Map class diagram to (extended) signature.
  - Stereotypes – for documentation.

UML Class Diagrams: Stocktaking

UML Class Diagram Syntax [Oestereich, 2006]



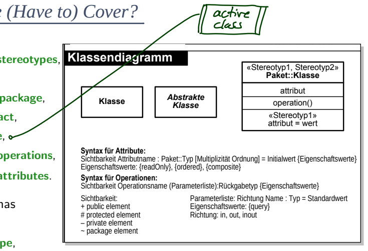
What Do We (Have to) Cover?

A **class**

- has a set of **stereotypes**,
- has a **name**,
- NOT ||| belongs to a **package**,
- can be **abstract**,
- can be **active**,
- LATER || has a set of **operations**,
- has a set of **attributes**.

Each **attribute** has

- a **visibility**,
- a **name**, a **type**,
- NOT ||| a **multiplicity**, an **order**,
- an **initial value**, and
- a set of **properties**, such as **readOnly**, or query **ordered**, etc.



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

Extended Signature

Recall: Signature

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

- (basic) types $\mathscr{T}$ and classes $\mathscr{C}$, (both finite),
- typed attributes V, τ from $\mathscr{T}$ or $C_{0,1}$ or C_*, $C \in \mathscr{C}$,
- $atr : \mathscr{C} \to 2^V$ mapping classes to attributes.

Too abstract to represent class diagram, e.g. no "place" to put class **stereotypes** or attribute **visibility**.

So: **Extend** definition for classes and attributes: Just as attributes already have types, we will assume that

- classes have (among other things) **stereotypes** and
- attributes have (in addition to a type and other things) a **visibility**.

Extended Classes

From now on, we assume that each class $C \in \mathscr{C}$ has:

- a finite (possibly empty) set S_C of **stereotypes**,
- a boolean flag $a \in \mathbb{B}$ indicating whether C is **abstract**,
- a boolean flag $t \in \mathbb{B}$ indicating whether C is **active**.

We use $S_\mathscr{C}$ to denote the set $\bigcup_{C \in \mathscr{C}} S_C$ of stereotypes in $\mathscr{S}$.

(Alternatively, we could add a set St as 5-th component to $\mathscr{S}$ to provides the stereotypes (names of stereotypes) to choose from. But: too unimportant to care.)

Convention:

- We write
$$\langle C, S_C, a, t \rangle \in \mathscr{C}$$
when we want to refer to all aspects of C.
- If the new aspects are irrelevant (for a given context), we simply write $C \in \mathscr{C}$ i.e. old definitions are still valid.

Example

$\langle C, S_C, a, t \rangle$

C $\rightsquigarrow \langle C, \emptyset, 0, 0 \rangle$ not abstract not active

«strange» D $\rightsquigarrow \langle D, \{strange\}, 0, 1 \rangle$

«signal, ✿» E {A} $\rightsquigarrow \langle E, \{signal, ✿\}, 1, 0 \rangle$

Extended Attributes

- From now on, we assume that each attribute $v \in V$ has (in addition to the type):
  - a **visibility**
$$\xi \in \{\underbrace{\text{public}}_{:=+}, \underbrace{\text{private}}_{:=-}, \underbrace{\text{protected}}_{:=\#}, \underbrace{\text{package}}_{:=\sim}\}$$
  - an **initial value** $expr_0$ given as a word from **language for initial values**, e.g. OCL expresions.
    (If using Java as **action language** (later) Java expressions would be fine.)
  - a finite (possibly empty) set of **properties** P_v.
    We define $P_\mathscr{C}$ analogously to stereotypes.

Convention:

- We write $\langle v : \tau, \overset{\text{visibility}}{\xi}, expr_0, P_v \rangle \in V$ when we want to refer to all aspects of v.
- Write only $v : \tau$ or v if details are irrelevant.

And?

- **Note**:
  All definitions we have up to now **principally still apply** as they are stated in terms of, e.g., $C \in \mathscr{C}$ — which still has a meaning with the extended view.

  For instance, system states and object diagrams remain mostly unchanged.
- **The other way round: most** of the newly added aspects **don't contribute** to the constitution of system states or object diagrams.
- Then what **are** they useful for...?
- First of all, to represent class diagrams.
- And then we'll see.

Mapping UML CDs to Extended Signatures

From Class Boxes to Extended Signatures

A class box n **induces** an (extended) signature class as follows:

n:

$$\langle\langle S_1, \dots, S_k \rangle\rangle$$
$$C$$
$$\xi_1\ v_1 : \tau_1 = v_{0,1}\ \{P_{1,1}, \dots, P_{1,m_1}\}$$
$$\vdots$$
$$\xi_\ell\ v_\ell : \tau_\ell = v_{0,\ell}\ \{P_{\ell,1}, \dots, P_{\ell,m_\ell}\}$$

$$\mathscr{C}(n) := \langle C, \{S_1, \dots, S_k\}, a(n), t(n) \rangle$$

$$V(n) := \{\langle v_1 : \tau_1, \xi_1, v_{0,1}, \{P_{1,1}, \dots, P_{1,m_1}\}\rangle, \dots, \langle v_\ell : \tau_\ell, \xi_\ell, v_{0,\ell}, \{P_{\ell,1}, \dots, P_{\ell,m_\ell}\}\rangle\}$$

$$atr(n) := \{C \mapsto \{v_1, \dots, v_\ell\}\}$$

where

- "abstract" is determined by the font:

$$a(n) = \begin{cases} \textit{true} & \text{, if } n = \boxed{C} \text{ or } n = \boxed{C\ \{A\}} \\ \textit{false} & \text{, otherwise} \end{cases}$$

- "active" is determined by the frame:

$$t(n) = \begin{cases} \textit{true} & \text{, if } n = \boxed{\mathsf{C}} \text{ or } n = \boxed{\mathsf{C}} \\ \textit{false} & \text{, otherwise} \end{cases}$$

What If Things Are Missing?

C
$v : Int$

- For instance, what about the box above?
- v has **no visibility**, **no initial value**, and (strictly speaking) **no properties**.

It depends.

- What does the standard say? [OMG, 2007a, 121]

  "**Presentation Options.**
  The type, visibility, default, multiplicity, property string may be suppressed from being displayed, even if there are values in the model."

  - **Visibility**: There is no "no visibility" — an attribute **has** a visibility in the (extended) signature.
    Some (and we) assume **public** as default, but conventions may vary.
  - **Initial value**: some assume it **given by domain** (such as "leftmost value", but what is "leftmost" of $\mathbb{Z}$?).
    Some (and we) understand **non-deterministic initialisation**.
  - **Properties**: probably safe to assume $\emptyset$ if not given at all.

From Class Diagrams to Extended Signatures

- We view a **class diagram** $\mathcal{CD}$ as a graph with nodes $\{n_1, \dots, n_N\}$ (each "class rectangle" is a node).
  or class boxes
  - $\mathscr{C}(\mathcal{CD}) := \bigcup_{i=1}^N \{\mathscr{C}(n_i)\} = \{\mathscr{C}(n) \mid n \in \{n_1, \dots, n_N\}\}$
  - $V(\mathcal{CD}) := \bigcup_{i=1}^N V(n_i)$
  - $atr(\mathcal{CD}) := \bigcup_{i=1}^N atr(n_i)$
- In a **UML model**, we can have **finitely many** class diagrams,

$$\mathscr{C}\mathscr{D} = \{\mathcal{CD}_1, \dots, \mathcal{CD}_k\},$$

which **induce** the following signature:

$$\mathscr{S}(\mathscr{C}\mathscr{D}) = \left(\mathscr{T}, \bigcup_{i=1}^k \mathscr{C}(\mathcal{CD}_i), \bigcup_{i=1}^k V(\mathcal{CD}_i), \bigcup_{i=1}^k atr(\mathcal{CD}_i)\right).$$

(Assuming $\mathscr{T}$ given. In "reality", we can introduce types in class diagrams, the class diagram then contributes to $\mathscr{T}$.)

Is the Mapping a Function?

- Is $\mathscr{S}(\mathscr{C}\mathscr{D})$ **well-defined**?

Two possible **sources for problems**:

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

(i)

$\mathcal{CD}_1$

C
$v : Int$

$\mathcal{CD}_2$

C
$w : Int$

(ii)

$\mathcal{CD}_1$

C
$v : Int$

$\mathcal{CD}_2$

C
$v : Bool$

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

Is the Mapping a Function?

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

C
$v : Bool$

D
$v : Int$

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 unique.)
- Subtle, formalist's approach: observe that

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

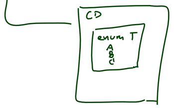
  are **different things** in V. But we don't follow that path...

Class Diagram Semantics

Semantics

- The semantics of a set of **class diagrams** $\mathscr{CD}$ first of all is the induced (extended) **signature** $\mathscr{S}(\mathscr{CD})$.
- The **signature** gives rise to a set of **system states** 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 τ, i.e. the set $\mathscr{D}(\tau)$, would be determined by the class diagram, and not free for choice.)



$\leadsto \mathscr{S}(\{T\}, \ldots)$

$\mathscr{D}(T) = \{A, B, C\}$

Semantics

- The semantics of a set of **class diagrams** $\mathscr{CD}$ first of all is the induced (extended) **signature** $\mathscr{S}(\mathscr{CD})$.
- The **signature** gives rise to a set of **system states** 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 τ, i.e. the set $\mathscr{D}(\tau)$, would be determined by the class diagram, and not free for choice.)

- What is the effect on $\Sigma_{\mathscr{S}}^{\mathscr{D}}$? **Little.** alive objects attr. valuations

  For now, we only **remove** abstract class instances, i.e.

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

  abstract

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

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

  "no identity of $\mathscr{D}(C)$ ever in $\mathrm{dom}(\sigma)$ if C is abstract"

  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**: 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 system transformers; and with inheritance.
  - **Properties**: such as `readOnly`, `ordered`, `composite`
    (**Deprecated** in the standard.)
    - `readOnly` — **later** treated similar to visibility.
    - `ordered` — too fine for our representation.
    - `composite` — cf. lecture on associations.

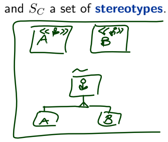
Stereotypes

Stereotypes as Labels or Tags

- So, a class is

  $$\langle C, S_C, a, t\rangle$$

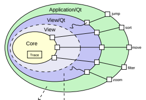
  with a the abstractness flag, t activeness flag, and S_C a set of **stereotypes**.
- What are Stereotypes?
  - **Not** represented in system states.
  - **Not** contributing to typing rules.
    (cf. **later** lecture on type theory for UML)

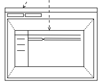


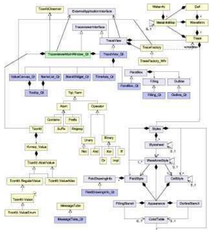
- [Oestereich, 2006]:
  View stereotypes as (additional) "**labelling**" ("tags") or as "**grouping**".

  Useful for documentation and MDA.
  - **Documentation**: e.g. layers of an architecture.
    Sometimes, packages (cf. the standard) are sufficient and "right".
  - **Model Driven Architecture** (MDA): **later**.

Example: Stereotypes for Documentation







- Example: Timing Diagram Viewer [Schumann et al., 2008]
- Architecture of four layers:
  - core, data layer
  - abstract view layer
  - toolkit-specific view layer/widget
  - application using widget
- Stereotype "=" layer "=" colour

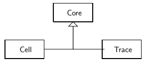
Stereotypes as Inheritance

- Another view (due to whom?): distinguish
  - **Technical Inheritance**

    If the **target platform**, such as the programming language for the implementation of the blueprint, is object-oriented, assume a 1-on-1 relation between inheritance in the model and on the target platform.
  - **Conceptual Inheritance**

    Only meaningful with a **common idea** of what stereotypes stand for. For instance, one could label each class with the team that is responsible for realising it. Or with licensing information (e.g., LGPL and proprietary).

    Or one could have labels understood by code generators (cf. lecture on MDSE).
- **Confusing**:
  - Inheritance is often referred to as the "is a"-relation.
    Sharing a stereotype also expresses "being something".
  - We can always (ab-)use UML-inheritance for the conceptual case, e.g.



Excursus: Type Theory (cf. Thiemann, 2008)

Type Theory

Recall: In lecture 03, we introduced OCL expressions with **types**, for instance:

$expr ::=$	w	$: \tau$	. . . logical variable w
	$\mid$ true $\mid$ false	$: Bool$	. . . constants
	$\mid 0 \mid -1 \mid 1 \mid \dots$	$: Int$	. . . constants
	$\mid expr_1 + expr_2$	$: Int \times Int \to Int$	. . . operation
	$\mid$ size$(expr_1)$	$: Set(\tau) \to Int$	

Wanted: A procedure to tell **well-typed**, such as $(w : Bool)$ ✓

not w — $: Bool \to Bool$

from **not well-typed**, such as,

size(w). $: Bool$ ↳ ⚡

$: Set(\tau) \to Int$

Type Theory

Recall: In lecture 03, we introduced OCL expressions with **types**, for instance:

$expr ::=$	w	$: \tau$	. . . logical variable w
	$\mid$ true $\mid$ false	$: Bool$	. . . constants
	$\mid 0 \mid -1 \mid 1 \mid \dots$	$: Int$	. . . constants
	$\mid expr_1 + expr_2$	$: Int \times Int \to Int$	. . . operation
	$\mid$ size$(expr_1)$	$: Set(\tau) \to Int$	

Wanted: A procedure to tell **well-typed**, such as $(w : Bool)$

not w

from **not well-typed**, such as,

size(w).

Approach: Derivation System, that is, a finite set of derivation rules.
We then say $expr$ **is well-typed** if and only if we can derive

$$A, C \vdash expr : \tau \qquad \text{(**read**: "expression } expr \text{ has type } \tau\text{")}$$

for some OCL type τ, i.e. $\tau \in T_B \cup T_{\mathscr{C}} \cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup T_{\mathscr{C}}\}$, $C \in \mathscr{C}$.

A Type System for OCL

A Type System for OCL

We will give a finite set of **type rules** (a **type system**) of the form

$$(\text{"name"})\ \frac{\text{"premises"}}{\text{"conclusion"}}\ \text{"side condition"}$$

A Type System for OCL

We will give a finite set of **type rules** (a **type system**) of the form

$$(\text{"name"})\ \frac{\text{"premises"}}{\text{"conclusion"}}\ \text{"side condition"}$$

These rules will establish well-typedness statements (**type sentences**) of three different "**qualities**":

(i) Universal well-typedness:

$$\vdash expr : \tau$$
$$\vdash 1 + 2 : Int$$

(ii) Well-typedness in a **type environment** A: (for logical variables)

$$A \vdash expr : \tau$$
$$self : \tau_C \vdash self.v : Int$$

(iii) Well-typedness in type environment A and **context** D: (for visibility)

$$A, D \vdash expr : \tau$$
$$self : \tau_C, C \vdash self\,.\,r\,.\,v : Int$$

References

References

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

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Schumann et al., 2008] 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.