Software Design, Modelling and Analysis in UML

Lecture 06: Class Diagrams I

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
UML Model Instances

\[ \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, atr), \mathcal{SM} \]

\[ M = (\Sigma^\mathcal{D}_\mathcal{S}, A_\mathcal{S}, \rightarrow_{\mathcal{SM}}) \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(cons_0, Snd_0)} u_0 \xrightarrow{(\sigma_1, \varepsilon_1)} \ldots \]

\[ w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}} \]

\[ G = (N, E, f) \]

\[ \mathcal{O} \mathcal{D} \]

\[ \frac{1}{2} \mathcal{C} \mathcal{D} \]

\[ x = 2 \]

\[ j = 1.0 \]

\[ \rho \]

\[ \frac{2}{c} : \mathcal{C} \]

\[ \rho = \emptyset \]
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:
  - Final notes on object diagrams.
  - Study UML syntax.
  - Prepare (extend) definition of signature.
  - Map class diagram to (extended) signature.
  - Stereotypes – for documentation.
The Other Way Round
If we only have a picture as below, we typically assume that it’s meant to be an object diagram wrt. some signature and structure.

In the example, we can conclude that the author is referring to some signature \( \mathcal{S} = (\mathcal{I}, \mathcal{C}, \mathcal{V}, \text{atr}) \) with at least

- \( \{C, D\} \subseteq \mathcal{C} \) (either fine, don’t know)
- \( \mathcal{T} \in \mathcal{I} \)
- \( \{x : C^*, p : C^*, z : \mathcal{T}\} \subseteq \mathcal{V} \)
- \( \{x\} \subseteq \text{atr}(C) \)
- \( \{p, z\} \subseteq \text{atr}(D) \)

and a structure with

- \( 0 \in \mathcal{D}(T) \)
- \( \{u_1, u_2\} \subseteq \mathcal{D}(C) \)
- \( \{u_3\} \subseteq \mathcal{D}(D) \)
Example: Object Diagrams for Documentation
Example: Data Structure [Schumann et al., 2008]

```
BaseNode
+parent: BaseNode*
+prevSibling: BaseNode*
+nextSibling: BaseNode*
+firstChild: BaseNode*
+lastChild: BaseNode*

Node
+data:T
+Node(data: T)

Iterator
+operator++()
+operator--()
+operator*()

Forest
+appendTopLevel(data: T)
+appendChild(parentIt: Iterator&, data: T)
+remove(it: Iterator&)
+depth(it: Iterator&)
+end()
+begin()
+empty()
+size()
```
Example: Illustrative Object Diagram [Schumann et al., 2008]
UML Class Diagrams: Stocktaking
UML Class Diagram Syntax [Oestereich, 2006]

Syntax für Attribute:
Sichtbarkeit Attributname : Paket::Typ [Multiplizität Ordnung] = Initialwert {Eigenschaftswerte}
Eigenschaftswerte: {readOnly}, {ordered}, {composite}

Syntax für Operationen:
Sichtbarkeit Operationsname (Parameterliste):Rückgabetyp {Eigenschaftswerte}
Parameterliste: Richtung Name : Typ = Standardwert
Eigenschaftswerte: {query}
Richtung: in, out, inout
What Do We (Have to) Cover?

A class

- has a set of **stereotypes**,
- has a **name**,  
- belongs to a **package**,  
- can be **abstract**,  
- can be **active**, e.g. ![Class]
- has a set of **operations**,  
- has a set of **attributes**.

Each **attribute** has

- a **visibility**,  
- a **name**, a **type**,  
- a **multiplicity**, an **order**  
- an **initial value**, and  
- a set of **properties**, such as **readOnly**, **ordered**, etc.

**Wanted**: places in the signature to represent the information from the picture.
Extended Signature
Recall: Signature

\[ \mathcal{I} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \] where

- (basic) types \( \mathcal{T} \) and classes \( \mathcal{C} \), (both finite),
- typed attributes \( V, \tau \) from \( \mathcal{T} \) or \( C_0,1 \) or \( \mathcal{C}_* \), \( C \in \mathcal{C} \),
- \( \text{atr} : \mathcal{C} \rightarrow 2^V \) 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 $C \in \mathcal{C}$ has:

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

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

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

Convention:

- We write

  \[
  \langle C, S_C, a, t \rangle \in \mathcal{C}
  \]

  when we want to refer to all aspects of $C$.

- If the new aspects are irrelevant (for a given context), we simply write $C \in \mathcal{C}$ i.e. old definitions are still valid.
Extended Attributes

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

**Convention:**
- We write \( \langle v : \tau, \xi, expr_0, P_v \rangle \in V \) when we want to refer to all aspects of \( v \).
- Write only \( v : \tau \) or \( v \) if details are irrelevant.
Note:
All definitions we have up to now *principally still apply* as they are stated in terms of, e.g., $C \in \mathbb{C}$ — 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
A class box $n$ induces an (extended) signature class as follows:

$$n: \langle \langle S_1, \ldots, S_k \rangle \rangle$$

where

- "abstract" is determined by the font:
  $$a(n) = \begin{cases} 
  true, & \text{if } n = \boxed{C} \text{ or } n = \boxed{C \{A\}} \\
  false, & \text{otherwise}
  \end{cases}$$

- "active" is determined by the frame:
  $$t(n) = \begin{cases} 
  true, & \text{if } n = \boxed{C} \text{ or } n = \boxed{C} \\
  false, & \text{otherwise}
  \end{cases}$$

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

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

$$\text{attr}(n) := \{ C \mapsto \{v_1, \ldots, v_\ell\} \}$$
\[ y = \left\{ \{ \text{Int}, \text{Float}, \text{Bool} \} \right. \]
\[ , \left\{ \langle C, \varnothing, \text{true}, \text{false} \rangle, \langle D, \text{false}, \text{false}, \text{true} \rangle \right\} \]
\[ , \left\{ \langle x: \text{Int}, +, 0, \varnothing \rangle, \langle y: \text{Float}, +, \varnothing, \varnothing \rangle, \langle z: \text{Bool}, +, \varnothing, \varnothing \rangle, \langle y: \text{Bool}, -, \varnothing, \varnothing \rangle \right\} \]

