Contents & Goals

Last Lecture:
- OCL Semantics
- Object Diagrams

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What is a class diagram?
  - For what purpose 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.

What Do We (Have to) Cover?

A class
- has a set of stereotropes,
- has a name,
- belongs to a package,
- can be abstract,
- can be active,
- 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

\[ \mathcal{F} = (\mathcal{F}, \mathcal{V}, \tau, \text{at}) \]

- \( \mathcal{F} \) are types \( \mathcal{F} \) and classes \( \mathcal{V} \) (both finite).
- typed attributes \( \tau \) from \( \mathcal{F} \) or \( \mathcal{F}_0 \) or \( \mathcal{V} \) \( \mathcal{V} \)
- \( \text{at} : \mathcal{V} \to 2^\mathcal{F} \) mapping classes to attributes.

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

So: **Extended** 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 \( C \in \mathcal{V} \) has:

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

We use \( S_C \) to denote the set \( \bigcup_{C \in \mathcal{V}} S_C \) of stereotypes in \( \mathcal{F} \).

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

**Convention:**

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

**Extended Attributes**

- From now on, we assume that each attribute \( v \in \mathcal{V} \) has:
  - (in addition to the type):
  - a visibility

\[ \xi \in \{ \text{public}, \text{private}, \text{protected}, \text{package} \} \]

- an initial value \( \text{expr}_0 \), 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 \( P_v \) analogously to stereotypes.

**Convention:**

- We write \( \langle v: \tau, \xi, \text{expr}_0, P_v \rangle \) 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., \( C \in \mathcal{V} \) — 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.
### From Class Diagrams to Extended Signatures

- We view a class diagram $CD$ as a graph with nodes $\{C_1, \ldots, C_n\}$ (each "class rectangle" is a node).
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t, a\}\}$ (where $\ell, m$ are domains)
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t\}\}$
  - $\psi(C_i) = \{\ell, m \mapsto \{v\}\}$

- In a UML model, we can have finitely many class diagrams, $\mathcal{S} = \{C_1, \ldots, C_n\}$, which induce the following signature:
  $$\mathcal{S}(\psi) = \left\{ \left( x, \left\{ \psi(C_1)(x), \ldots, \psi(C_n)(x) \right\} \right) \right\}.$$  

### What If Things Are Missing?

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

### Mapping UML CDs to Extended Signatures

A class box induces an extended signature class as follows:

- $\xi(a) = \{v = \mathbf{false}, x = \mathbf{false}, t = \mathbf{false}, a = \mathbf{true}\}$
- $\xi(a) = \{v = \mathbf{false}, x = \mathbf{false}, t = \mathbf{false}, a = \mathbf{false}\}$

### From Class Diagrams to Extended Signatures

- We view a class diagram $CD$ as a graph with nodes $\{C_1, \ldots, C_n\}$ (each "class rectangle" is a node).
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t, a\}\}$ (where $\ell, m$ are domains)
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t\}\}$
  - $\psi(C_i) = \{\ell, m \mapsto \{v\}\}$

- In a UML model, we can have finitely many class diagrams, $\mathcal{S} = \{C_1, \ldots, C_n\}$, which induce the following signature:
  $$\mathcal{S}(\psi) = \left\{ \left( x, \left\{ \psi(C_1)(x), \ldots, \psi(C_n)(x) \right\} \right) \right\}.$$  

### Is the Mapping a Function?

- Is $\mathcal{S}(\psi)$ well-defined?
- Two possible sources for problems:
  1. A class $C$ may appear in multiple class diagrams:
  - $\mathcal{S}(\psi) = \left( x, \left\{ \psi(C_1)(x), \ldots, \psi(C_n)(x) \right\} \right)$
  - Simply forbid the case (i) — easy syntactical check on diagram.

### What If Things Are Missing?

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

### Mapping UML CDs to Extended Signatures

A class box induces an extended signature class as follows:

- $\xi(a) = \{v = \mathbf{false}, x = \mathbf{false}, t = \mathbf{false}, a = \mathbf{true}\}$
- $\xi(a) = \{v = \mathbf{false}, x = \mathbf{false}, t = \mathbf{false}, a = \mathbf{false}\}$

### From Class Diagrams to Extended Signatures

- We view a class diagram $CD$ as a graph with nodes $\{C_1, \ldots, C_n\}$ (each "class rectangle" is a node).
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t, a\}\}$ (where $\ell, m$ are domains)
  - $\psi(C_i) = \{\ell, m \mapsto \{v, t\}\}$
  - $\psi(C_i) = \{\ell, m \mapsto \{v\}\}$

- In a UML model, we can have finitely many class diagrams, $\mathcal{S} = \{C_1, \ldots, C_n\}$, which induce the following signature:
  $$\mathcal{S}(\psi) = \left\{ \left( x, \left\{ \psi(C_1)(x), \ldots, \psi(C_n)(x) \right\} \right) \right\}.$$  

### Is the Mapping a Function?

- Is $\mathcal{S}(\psi)$ well-defined?
- Two possible sources for problems:
  1. A class $C$ may appear in multiple class diagrams:
  - $\mathcal{S}(\psi) = \left( x, \left\{ \psi(C_1)(x), \ldots, \psi(C_n)(x) \right\} \right)$
  - Simply forbid the case (i) — easy syntactical check on diagram.
Semantics

• The semantics of a set of class diagrams \( \mathcal{V}^{\mathcal{S}} \) first of all is the induced (extended) signature \( \mathcal{J}(\mathcal{V}^{\mathcal{S}}) \).

• The signature gives rise to a set of system states given a structure \( \mathcal{S} \).

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

• What is the effect on \( \Sigma^{\mathcal{S}} \)?

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

\[
\sigma : \mathcal{S}(C) \not\rightarrow \mathcal{S}(V) \not\rightarrow \left( \mathcal{S}(V) \cup \mathcal{S}(C)^* \right)
\]

is now only called system state if and only if, for all \( \langle C, S, C, t \rangle \in \mathcal{V}^{\mathcal{S}} \),

\[
dom(\sigma) \cap \mathcal{S}(C) = \emptyset.
\]

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

Stereotypes as Labels or Tags

• So, a class is \( \langle C, A, a, t \rangle \) with \( a \) the abstraction flag, \( j \) the creation flag, and \( A \) 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)

[Oestreich, 2006]: View stereotypes as "adjectives" or "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.
**Stereotypes for Documentation**

- Example: Timing Diagram Viewer
- Architecture of four layers:
  - core, data layer
  - abstract view layer
  - toolkit-specific view layer/widget
  - application using widget

**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 making it. Or with licensing information (e.g., LGPL and proprietary).

**Confusing:**

- Inheritance is often referred to as the "isa"-relation. Sharing a stereotype also expresses "beingsomething".
- We can always (ab-)use UML-inheritance for the conceptual case, e.g.

**Type Theory**

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

\[ \text{expr ::= } w : \tau \quad \text{logical variable } w \]
\[ \text{true} \mid \text{false} : \text{Bool} \quad \text{constants} \]
\[ [0] \mid [1] \mid \ldots : \text{Int} \quad \text{constants} \]
\[ \text{size}(\text{expr}) : \text{Set}(\tau) \rightarrow \text{Int} \quad \text{operation} \]

Wanted: A procedure to tell well-typed, such as \((w : \text{Bool})\)

\[ \text{not}(w) \]

from not well-typed, such as:

\[ \text{size}(\text{not}(w)) \]

**Excursus: Type Theory (cf. Thiemann, 2008)**

- A Type System for OCL

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

\[ \text{expr ::= } w : \tau \quad \text{logical variable } w \]
\[ \text{true} \mid \text{false} : \text{Bool} \quad \text{constants} \]
\[ [0] \mid [1] \mid \ldots : \text{Int} \quad \text{constants} \]
\[ \text{size}(\text{expr}) : \text{Set}(\tau) \rightarrow \text{Int} \quad \text{operation} \]

Wanted: A procedure to tell well-typed, such as \((w : \text{Bool})\)

\[ \text{not}(w) \]

from not well-typed, such as:

\[ \text{size}(\text{not}(w)) \]

**Approach:** Derivation System, that is, a finite set of derivation rules.

We then say \(\text{expr}\) is well-typed if and only if we can derive

\[ A, C \vdash \text{expr} : \tau \quad \text{(read: "expression expr has type } \tau\text{")} \]

for some OCL type \(\tau\), i.e., \(\tau \in T + \cup_{\tau \in T} \cup \{\text{Set}(\tau)\} \cup \{\gamma \in T \cup \cup_{\tau \in T} \} \cup W\).
A Type System for OCL

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

```
("name") "premises" "conclusion" "side condition"
```

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

(i) Universal well-typedness:

\[ \Gamma \vdash \text{expr} : \tau \]
\[ \Gamma \vdash 1 + 2 : \text{Int} \]

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

\[ A \vdash \text{expr} : \tau \]
\[ \text{self} : \tau \vdash \text{self} \cdot v : \text{Int} \]

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

\[ A, D \vdash \text{expr} : \tau \]
\[ \text{self} : \eta_2, \diamondsuit \vdash \text{self} \cdot v : \text{Int} \]

References


