Contents & Goals

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

This Lecture:
• Educational Objectives:
  • 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.

Is the Mapping a Function?

Question: Is \( S(C,D) \) well-defined?

There are two possible sources for problems:

(1) A class \( C \) may appear in multiple class diagrams:
   (i) \( C_v: \text{Int} \)
   (ii) \( C_v: \text{Int} \) \( \quad C_v: \text{Bool} \)

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

(2) An attribute \( v \) may appear in multiple classes with different type:
   \( C_v: \text{Bool} \)
   \( D_v: \text{Int} \)

Two approaches:
• Require unique attribute names. This requirement can easily be established (implicitly, behind the scenes) by viewing \( v \) as an abbreviation for \( C::v \) or \( D::v \) depending on the context. (\( C::v: \text{Bool} \) and \( D::v: \text{Int} \) are then unique.)
• Subtle, formalist's approach: observe that \( ⟨v: \text{Bool}, . . .⟩ \) and \( ⟨v: \text{Int}, . . .⟩ \) are different things in \( V \). We don't follow that path...
The semantics of a set of class diagrams $C_D$ is the induced signature $S(C_D)$.

The signature induces a set of system states $\Sigma_{DS}$ (given a structure $D$).

- Do we need to redefine/extend $D$?
  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 $D(T)$, would be determined by the class diagram, and not free for choice.)

- What is the effect on $\Sigma_{DS}$?
  Little. For now, we only remove abstract class instances, i.e. $\sigma: D(C) \not\rightarrow (V \not\rightarrow (D(T) \cup D(C^*) ))$ is now only called system state if and only if, for all $\langle C, S_C, 1, t \rangle \in C$, $\text{dom}(\sigma) \cap D(C) = \emptyset$.

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

Classes:
- Active: not represented in $\sigma$.
  Later: relevant for behaviour, i.e., how system states evolve over time.
- Stereotypes: in a minute.

Attributes:
- Initial value expression: not represented in $\sigma$.
  Later: provides an initial value as effect of "creation action".
- Visibility: not represented in $\sigma$.
  Later: viewed as additional typing information for well-formedness of actions; and with inheritance.

Properties:
- Such as $\text{readOnly}$, $\text{ordered}$, $\text{composite}$ (Deprecated in the standard).
  - $\text{readOnly}$— later treated similar to visibility.
  - $\text{ordered}$— not considered in our UML fragment ($\rightarrow$ sets vs. sequences).
  - $\text{composite}$— cf. lecture on associations.
New rules concerning visibility should be considered in the models of C, D.

1. Context Int: \( \{ C \} \rightarrow \{ D \} \)
2. \( m \in \{ C \rightarrow \{ D \} \) \)
3. \( m \in \{ D \rightarrow \{ C \} \) \)

The situation is the same with \( \{ \} \rightarrow \{ \} \).

\( x \in \{ C \rightarrow \{ D \} \) \)

\( x \in \{ D \rightarrow \{ C \} \) \)

\( x \in \{ \} \rightarrow \{ \} \)
Architecture has four layers:

- abstract view layer
- toolkits-specific view layer/widget
- stereotyped view layer
- groupings

Stereotypes as Labels or Tags

Visibility is a property of attributes -
- visibility.

We only evaluate (\(D_\text{m} \cdot x > 0\)) :

\[
D_\text{m} \cdot x > 0
\]

- a matter of well-typedness only
- or
- need not

In other words: given the diagram above,

- Constraints and pre/post conditions
- visibility.

Stereotypes

• Toolkits

Stereotype

Toolkits

• Application Programming Interface

• Abstract Design Elements

• Abstract View

• Core View

• Custom Views

• GUI

• MDA

Schumann et al.

Later

• Guards

• north

• n.

• x

• jump

• move

• Core View/Qt

• Visibility

- in diagrams

- contributing to typing rules / well-formedness.

- do take visibility into account.

To state "global" requirements, sometimes not

- taken into account.

- In other words: given the diagram above,

- Visibility is a property of attributes —

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It depends

- It does not

- It does not

- It does not

- It does not

- It does not
• 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


