Contents & Goals

Last Lecture:
• Live Sequence Charts Semantics

This Lecture:
• Educational Objectives: Capabilities for following tasks/questions.
  • What’s the Liskov Substitution Principle?
  • What is late/early binding?
  • What is the subset, what the uplink semantics of inheritance?
  • What’s the effect of inheritance on LSCs, State Machines, System States?
  • What is the idea of Meta-Modelling?
• Content:
  • Inheritance in UML: concrete syntax
  • Liskov Substitution Principle — desired semantics
  • Two approaches to obtain desired semantics

Inheritance: Syntax

Inheritance: Generalisation Relation

Abstract Syntax

Recall: a signature (with signals) is a tuple $\mathcal{S} = (\Sigma, \mathcal{C}, \Delta)$. Now (finally): extend to

$\mathcal{S} = (\Sigma, \mathcal{C}, \Delta, \mathcal{F}, \text{mth}, \triangleright)$

where $\text{mth}$ are methods, analogously to attributes and

$\triangleright \subseteq (\mathcal{C} \times \mathcal{C}) \cup (\mathcal{X} \times \mathcal{X})$

is a generalisation relation such that $C \triangleright D$ for all $C \in \mathcal{C}$ ("arising from") $D$. $C \triangleright D$ reads as

• $C$ is a generalisation of $D$, $D$ is a specialisation of $C$, $D$ is a subclass of $C$, $C$ is a super-class of $D$, $\ldots$

Well-formedness rule: No cycles in the generalisation relation.
Mapping Concrete to Abstract Syntax by Example

$$\mathcal{F} = \left\{ \{C_0, C_1, D\}, \{C_0, C_2, D\} \right\}$$

D

C_0

C_1

C_2

Note: we can have multiple inheritance.

Reflexive, Transitive Closure of Generalisation

Definition. Given classes \(C_0, C_1, D \in \mathcal{B}\), we say \(D\) inherits from \(C_0\) via \(C_1\) if and only if there are \(C_{1^0}, \ldots, C_{1^n}, C_{1^1}, \ldots, C_{1^m} \in \mathcal{B}\) such that

\[
C_0 \triangleright C_{1^0} \triangleright \ldots \triangleright C_{1^n} \triangleright C_{1^1} \triangleright \ldots \triangleright C_{1^m} \triangleright D.
\]

We use ‘\(\triangleright\)’ to denote the reflexive, transitive closure of ‘\(\triangleright\)’.

In the following, we assume

- that all attribute (method) names are of the form
  \(C::v, C::f \in \mathcal{A}(C)\) resp. \(C::f \in \mathcal{M}(C)\) if and only if \(v\) (\(f\)) appears in an attribute (method) compartment of \(C\) in a class diagram.

- that we have \(C::v \in \mathcal{A}(C)\) resp. \(C::f \in \mathcal{M}(C)\) if and only if \(v\) (\(f\)) appears as an attribute (method) compartment of \(C\) in a class diagram.

We will want to accept "context \(C\) over \(v < f\)" if and only if \(v\) is meaningful. Later!

References


