Software Design, Modelling and Analysis in UML

Lecture 18: Inheritance I

2012-02-01

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

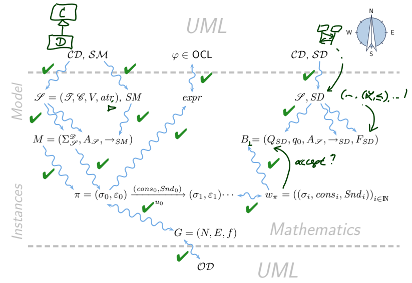
Last Lecture:

- Live Sequence Charts Semantics

This Lecture:

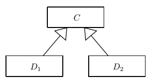
- **Educational Objectives:** Capabilities for following tasks/questions.
  - What's the Liskov Substitution Principle?
  - What is late/early binding?
  - What is the subset, what the uplink semantics of inheritance?
  - What's the effect of inheritance on LSCs, State Machines, System States?
  - What's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

Course Map

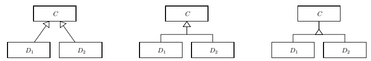


Inheritance: Syntax

Inheritance: Generalisation Relation



- **Alternative renderings:**



- **Read:**
  - C **generalises** D_1 and D_2; C is a **generalisation** of D_1 and D_2,
  - D_1 and D_2 **specialise** C; D_1 **is a** (specialisation of) C,
  - D_1 **is a** C; D_2 **is a** C.

NOT:

- **Well-formedness rule:** No **cycles** in the generalisation relation.

Abstract Syntax

Recall: a signature (with signals) is a tuple $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.

Now (finally): extend to

behav. feat.

$: \mathscr{C} \to F$

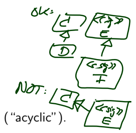
$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth, \lhd)$$

where F/mth are methods, analogously to attributes and

$$\lhd \subseteq (\mathscr{C} \times \mathscr{C}) \cup (\mathscr{E}(\mathscr{S}) \times \mathscr{E}(\mathscr{S}))$$

is a **generalisation** relation such that $C \lhd^+ C$ for **no** $C \in \mathscr{C}$ ("acyclic").

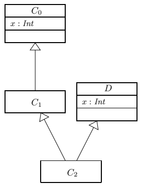
transitive closure



$C \lhd D$ reads as

- C is a generalisation of D,
- D is a specialisation of C,
- D inherits from C,
- D is a sub-class of C,
- C is a super-class of D,
- ...

Mapping Concrete to Abstract Syntax by Example



$\leadsto$

$$\mathscr{S} = \big(\{Int\},\ \{C_0, C_1, D, C_2\},\ \{C_0::x : Int,\ D::x : Int\},\ \{C_0 \mapsto \{C_0::x\},\ D \mapsto \{D::x\},\ C_1 \mapsto \emptyset,\ C_2 \mapsto \emptyset\},\ \{C_0 \lhd C_1,\ C_1 \lhd C_2,\ D \lhd C_2\} \big)$$

NOT: $atr(C_2) = \{C_0::x,\ D::x\}$

Note: we can have **multiple inheritance**.

Reflexive, Transitive Closure of Generalisation

Definition. Given classes $C_0, C_1, D \in \mathscr{C}$, we say D inherits from C_0 **via** C_1 if and only if there are $C_0^1, \dots C_0^n, C_1^1, \dots C_1^m \in \mathscr{C}$ such that

$$C_0 \lhd C_0^1 \lhd \dots C_0^n \lhd C_1 \lhd C_1^1 \lhd \dots C_1^m \lhd D.$$

We use '$\preceq$' to denote the reflexive, transitive closure of '$\lhd$'.

In the following, we assume

- that all attribute (method) names are of the form

$$C::v, \quad C \in \mathscr{C} \cup \mathscr{E} \qquad (C::f, \quad C \in \mathscr{C}),$$

- that we have $C::v \in atr(C)$ resp. $C::f \in mth(C)$ **if and only if** v (f) appears in an attribute (method) compartment of C in a class diagram.

We still want to accept "context C inv : $v < 0$", which v is meant? Later!

References

References

[Fischer and Wehrheim, 2000] Fischer, C. and Wehrheim, H. (2000). Behavioural subtyping relations for object-oriented formalisms. In Rus, T., editor, AMAST, number 1816 in Lecture Notes in Computer Science. Springer-Verlag.

[Liskov, 1988] Liskov, B. (1988). Data abstraction and hierarchy. SIGPLAN Not., 23(5):17–34.

[Liskov and Wing, 1994] Liskov, B. H. and Wing, J. M. (1994). A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems (TOPLAS), 16(6):1811–1841.

[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.

[Stahl and Völter, 2005] Stahl, T. and Völter, M. (2005). Modellgetriebene Softwareentwicklung. dpunkt.verlag, Heidelberg.