Contents & Goals

Last Lecture:
- Motivation: model-based development of things (houses, software) to cope with complexity, detect errors early
- Model-based (or -driven) Software Engineering
- UML Mode of the Lecture: Blueprint.

This Lecture:
- Educational Objectives: Capabilities for these tasks/questions:
  - Why is UML of the form it is?
  - Shall one feel bad if not using all diagrams during software development?
  - What is a signature, an object, a system state, etc.?
    What’s the purpose of signature, object, etc. in the course?
  - How do Basic Object System Signatures relate to UML class diagrams?

- Content:
  - Brief history of UML
  - Course map revisited
  - Basic Object System Signature, Structure, and System State
Why **(of all things)** UML?

- Note: being a **modelling** languages doesn’t mean being graphical (or: being a visual formalism [Harel]).
- For instance, [Kastens and Büning, 2008] also name:
  - Sets, Relations, Functions
  - Terms and Algebras
  - Propositional and Predicate Logic
  - Graphs
  - XML Schema, Entity Relation Diagrams, UML Class Diagrams
  - Finite Automata, Petri Nets, UML State Machines

- **Pro**: visual formalisms are found appealing and easier to **grasp**. Yet they are not necessarily easier to **write**!
- **Beware**: you may meet people who dislike visual formalisms just for being graphical — maybe because it is easier to “trick” people with a meaningless picture than with a meaningless formula.
  - More serious: it’s maybe easier to misunderstand a picture than a formula.
A Brief History of UML

- Boxes/lines and finite automata are used to visualise software for ages.

- 1970’s, **Software Crisis**™
  - Idea: learn from engineering disciplines to handle growing complexity.
  - Languages: Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams

- Mid 1980’s: **Statecharts** [Harel, 1987], **StateMate**™ [Harel et al., 1990]

- Early 1990’s, advent of **Object-Oriented**-Analysis/Design/Programming
  - Inflation of notations and methods, most prominent:
    - **Object-Modeling Technique** (OMT) [Rumbaugh et al., 1990]
A Brief History of UML

- Boxes/lines and finite automata are used to visualise software for ages.

- 1970’s, Software Crisis™ — Idea: learn from engineering disciplines to handle growing complexity.
  
  Languages: Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams

- Mid 1980’s: Statecharts [Harel, 1987], StateMate™ [Harel et al., 1990]

- Early 1990’s, advent of Object-Oriented-Analysis/Design/Programming — Inflation of notations and methods, most prominent:
  
  - Object-Modeling Technique (OMT) [Rumbaugh et al., 1990]
  - Booch Method and Notation [Booch, 1993]
  
  - Object-Oriented Software Engineering (OOSE) [Jacobson et al., 1992]

  Each “persuasion” selling books, tools, seminars...

- Late 1990’s: joint effort UML 0.x, 1.x

  Standards published by Object Management Group (OMG), “international, open membership, not-for-profit computer industry consortium”.

- Since 2005: UML 2.x
**UML Overview** [OMG, 2007b, 684]

![UML Diagram](image)

**Common Expectations on UML**

- Easily writeable, readable even by customers
- Powerful enough to bridge the gap between idea and implementation
- Means to tame complexity by separation of concerns ("views")
- Unambiguous
- Standardised, exchangeable between modelling tools
- UML standard says how to develop software
- Using UML leads to better software
- ...

**We will see...**

Seriously: After the course, you should have an own opinion on each of these claims. In how far/in what sense does it hold? Why? Why not? How can it be achieved? Which ones are really only hopes and expectations? ...?
The Plan

Recall:

- **Overall aim**: A formal language for software blueprints.
- **Approach**:
  1. Common semantical domain.
  2. UML fragments as syntax.
  3. Abstract representation of diagrams.
  4. Informal semantics: UML standard
  5. Assign meaning to diagrams
  6. Define, e.g., consistency.
### UML: Semantic Areas

Figure 6.1 - A schematic of the UML semantic areas and their dependencies

[OMG, 2007b, 11]

---

Common Semantical Domain
**Definition.** A (Basic) Object System Signature is a quadruple \( \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \) where

- \( \mathcal{T} \) is a set of (basic) types,
- \( \mathcal{C} \) is a finite set of classes,
- \( V \) is a finite set of typed attributes, i.e., each \( v \in V \) has type \( \tau \in \mathcal{T} \) or \( C_0, 1 \) or \( C^* \), \( C \in \mathcal{C} \) (written \( v : \tau \) or \( v : C_0, 1 \) or \( v : C^* \)),
- \( \text{atr} : \mathcal{C} \rightarrow 2^V \) maps each class to its set of attributes.

**Note:** Inspired by OCL 2.0 standard [OMG, 2006], Annex A.

---

**Basic Object System Signature Example**

\( \mathcal{S} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \) where

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

**Example:**

\( \mathcal{S}_0 = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_0, 1, n : C^*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \)
Basic Object System Signature

Another Example

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

- (basic) types \( \mathcal{T} \) and classes \( \mathcal{C} \), (both finite),
- typed attributes \( V, \tau \) from \( \mathcal{T} \) or \( C_{0,1} \) or \( C_{\ast} \), \( C \in \mathcal{C} \),
- \( atr : \mathcal{C} \rightarrow 2^V \) mapping classes to attributes.

Example:

\[ \mathcal{P} = (\{ X, Y, z \}, \{ F(T), p \}, \{ X, f \}, \{ X \rightarrow X \}) \]

Basic Object System Structure

**Definition.** A Basic Object System Structure of

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

is a domain function \( \mathcal{D} \) which assigns to each type a domain, i.e.

- \( \tau \in \mathcal{T} \) is mapped to \( \mathcal{D}(\tau) \),
- \( C \in \mathcal{C} \) is mapped to an infinite set \( \mathcal{D}(C) \) of (object) identities.

Note: Object identities only have the "\( = \)" operation; object identities of different classes are disjoint, i.e.

\[ \forall C, D \in \mathcal{C} : C \neq D \rightarrow \mathcal{D}(C) \cap \mathcal{D}(D) = \emptyset. \]

- \( C_{\ast} \) and \( C_{0,1} \) for \( C \in \mathcal{C} \) are mapped to \( 2^{\mathcal{D}(C)} \).

We use \( \mathcal{D}(\mathcal{C}) \) to denote \( \bigcup_{C \in \mathcal{C}} \mathcal{D}(C) \); analogously \( \mathcal{D}(C_{\ast}) \).

**Note:** We identify objects and object identities, because both uniquely determine each other (cf. OCL 2.0 standard).
**Basic Object System Structure Example**

**Wanted:** a structure for signature

\[ \mathcal{S}_0 = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]

Recall: by definition, seek a \( \mathcal{D} \) which maps
- \( \tau \in \mathcal{C} \) to some \( \mathcal{D}(\tau) \),
- \( c \in \mathcal{C} \) to some identities \( \mathcal{D}(C) \) (infinite, disjoint for different classes),
- \( C_* \) and \( C_{0,1} \) for \( C \in \mathcal{C} \) to \( \mathcal{D}(0) = 2^{\mathcal{D}(C)} \).

We have

\[
\begin{align*}
\mathcal{D}(\text{Int}) &= \mathbb{Z} \\
\mathcal{D}(C) &= \mathbb{N}^I \times \{C\}^1 = \{\lambda \ell, 2_{C_{0,1}}, 3_{C_*}\} \\
\mathcal{D}(D) &= \mathbb{N}^I \times \{D\}^1 = \{\lambda \ell, 2_{D_{0,1}}, 3_{D_*}\} \\
\mathcal{D}(C_{0,1}) &= \mathcal{D}(C_*) = 2^{\mathcal{D}(C)} \\
\mathcal{D}(D_{0,1}) &= \mathcal{D}(D_*) = 2^{\mathcal{D}(D)}
\end{align*}
\]

**System State**

**Definition.** Let \( \mathcal{D} \) be a structure of \( \mathcal{S} = (\mathcal{F}, \mathcal{C}, \mathcal{V}, \text{atr}) \). A system state of \( \mathcal{S} \) wrt \( \mathcal{D} \) is a **type-consistent** mapping

\[
\sigma : \mathcal{D}(\mathcal{C}) \mapsto (\mathcal{V} \mapsto (\mathcal{D}(\mathcal{F}) \cup \mathcal{D}(\mathcal{C}))).
\]

That is, for each \( u \in \mathcal{D}(C), C \in \mathcal{C}, \) if \( u \in \text{dom}(\sigma) \)

- \( \text{dom}(\sigma(u)) = \text{atr}(C) \)
- \( \sigma(u)(v) \in \mathcal{D}(\tau) \) if \( v : \tau, \tau \in \mathcal{F} \)
- \( \sigma(u)(v) \in \mathcal{D}(D*) \) if \( v : D_{0,1} \) or \( v : D_* \) with \( D \in \mathcal{C} \)

We call \( u \in \mathcal{D}(\mathcal{C}) \) **alive** in \( \sigma \) if and only if \( u \in \text{dom}(\sigma) \).

We use \( \Sigma_{\mathcal{S}}^{\mathcal{D}} \) to denote the set of all system states of \( \mathcal{S} \) wrt \( \mathcal{D} \).
**System State Example**

**Signature, Structure:**

\[ \mathcal{S}_0 = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_{0.1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]

\[ \mathcal{P}(\text{Int}) = \mathbb{Z}, \quad \mathcal{P}(C) = \{1_C, 2_C, 3_C, \ldots\}, \quad \mathcal{P}(D) = \{1_D, 2_D, 3_D, \ldots\} \]

**Wanted:** \( \sigma : \mathcal{P}(\mathcal{C}) \rightarrow (\mathcal{V} \rightarrow (\mathcal{P}(\mathcal{F}) \cup \mathcal{P}(\mathcal{E}))) \) such that

- \( \text{dom}(\sigma(u)) = \text{atr}(C) \),
- \( \sigma(u)(v) \in \mathcal{P}(\tau) \) if \( v : \tau, \tau \in \mathcal{F} \),
- \( \sigma(u)(v) \in \mathcal{P}(C_{\ast}) \) if \( v : D_{\ast} \) with \( D \in \mathcal{E} \).

One way to read out:
- Object \( v \) has a \( p \)-link to \( 1_C \) (i.e., to itself)
- Object \( 1_C \) refers to objects \( 5C, 6C \) no link in

**Concrete, explicit:**

\[ \sigma = \{ 1_C \mapsto \{ p \mapsto 0, n \mapsto \{5C, 6C\} \}, 5C \mapsto \{ p \mapsto 0, n \mapsto \{5C\} \}, 1_D \mapsto \{x \mapsto 23\} \}. \]

**Alternative:** symbolic system state

\[ \sigma = \{ 1_C \mapsto \{ p \mapsto 0, n \mapsto \{c_1\} \}, 5C \mapsto \{ p \mapsto 0, n \mapsto \{c_2\} \}, 1_D \mapsto \{x \mapsto 23\} \} \]

assuming \( 5C \in \mathcal{P}(C), d \in \mathcal{P}(D), c_1 \neq c_2 \).
You Are Here.

Course Map

$\mathcal{F} = (\mathcal{F}, \mathcal{E}, V, \text{stmt})$, $\mathcal{SM}$

$M = (\Sigma, A, \to_{\mathcal{SM}})$

$\pi = (\sigma_0, \epsilon_0) \xrightarrow{\text{cons}_i, \text{Snd}_i} (\sigma_1, \epsilon_1) \cdots$

$\varphi \in \text{OCL}$

$\mathcal{F}, \mathcal{SD}$

$B = (Q_{\mathcal{SD}}, q_0, A, \to_{\mathcal{SD}}, F_{\mathcal{SD}})$

$\varphi \in \text{OCL}$

$\mathcal{CD}, \mathcal{SM}$

$\mathcal{CD}, \mathcal{SD}$

$G = (N, E, f)$

$\pi = (\sigma_0, \epsilon_0) \xrightarrow{\text{cons}_i, \text{Snd}_i} (\sigma_1, \epsilon_1) \cdots$

$\mathcal{OD}$

$\mathcal{OD}$

Mathematics

UML

UML
References


