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
  - Basic Object System Signature, Structure, and System State
Why (of all things) UML?

• Pre-Note: being a modelling languages doesn’t mean being graphical (or: being a visual formalism [Harel]).

• [Kastens and Büning, 2008] consider as examples:
  • 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]

---

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
Figure A.5 - The taxonomy of structure and behavior diagram
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? ...?
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]
Basic Object System Signature

Definition. A (Basic) Object System Signature is a quadruple

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

where

- \( T \) is a set of (basic) types,
- \( C \) 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 \( C_{\ast} \), where \( C \in \mathcal{C} \)
  (written \( v : \tau \) or \( v : C_{0,1} \) or \( v : C_{\ast} \)),
- \( atr : \mathcal{C} \rightarrow 2^V \) maps each class to its set of attributes.

Note: Inspired by OCL 2.0 standard [OMG, 2006], Annex A.
Basic Object System Signature Example

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

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

Example:

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

Definition. A Basic Object System Structure of \( S = (T, C, V, atr) \) is a domain function \( D \) which assigns to each type a domain, i.e.

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

Note: Object identities only have the “=” operation; object identities of different classes are disjoint, i.e. \( \forall C, D \in C : C \neq D \) \( \Rightarrow \) \( D(C) \cap D(D) = \emptyset \).

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

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

**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

\[ S_0 = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]

Recall: by definition, seek a \( D \) which maps

- \( \tau \in T \) to some \( D(\tau) \),
- \( c \in C \) to some identities \( D(C) \) (infinite, disjoint for different classes),
- \( C_* \) and \( C_{0,1} \) for \( C \in C \) to \( D(C_{0,1}) = D(C_*) = 2^{D(C)} \).

\[
\begin{align*}
D(\text{Int}) &= \mathbb{Z} \\
D(C) &= \mathbb{N}^\times \{d \} \Leftrightarrow \{f, g, \ldots\} \\
D(D) &= \mathbb{N}^\times \{x \} \Leftrightarrow \{f, g, \ldots\} \\
D(C_{0,1}) &= D(C_*) = 2^{D(C)} \\
D(D_{0,1}) &= D(D_* ) = 2^{D(D)}
\end{align*}
\]
Definition. Let $\mathcal{P}$ be a structure of $\mathcal{I} = (\tau, \mathcal{C}, V, \text{atr})$. A system state of $\mathcal{I}$ wrt. $\mathcal{P}$ is a type-consistent mapping

$$\sigma : \mathcal{P}(\mathcal{C}) \rightarrow (V \rightarrow (\mathcal{P}(\tau) \cup \mathcal{P}(\mathcal{D}_u)))$$

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

- $\text{dom}(\sigma(u)) = \text{atr}(C)$
- $\sigma(u)(v) \in \mathcal{P}(\tau)$ if $v : \tau, \tau \in \tau$
- $\sigma(u)(v) \in \mathcal{P}(D_u)$ if $v : D_0, 1$ or $v : D_u$ with $D \in \mathcal{C}$

We call $u \in \mathcal{P}(\mathcal{C})$ alive in $\sigma$ if and only if $u \in \text{dom}(\sigma)$. We use $\Sigma^\mathcal{P}_\mathcal{I}$ to denote the set of all system states of $\mathcal{I}$ wrt. $\mathcal{P}$. 

\[\mathcal{P}(A) = \{A, A, A, A, A\} \]
\[\mathcal{P}(B) = \{B, B, B, B\} \]
\[\mathcal{P}(\sigma) = \{1, 2, 3, \ldots\} \]
\[\mathcal{P}(5) = \{1, 2, 3, \ldots\} \]

\[\mathcal{P}(\mathcal{A}_u) = 2^{\mathcal{D}(\mathcal{A}_u)} \text{ e.g. } \{A, A\} \in \mathcal{D}(\mathcal{A}_u)\]
System State Example

**Signature, Structure:**

\[ S_0 = (\{\text{Int}\}, \{C, D\}, \{x: \text{Int}, p: C_{0,1}, n: C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]

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

**Wanted:** \( \sigma : \mathcal{D}(C) \rightarrow (V \rightarrow (\mathcal{D}(\mathcal{T}) \cup \mathcal{D}(C_*))) \) such that for all \( v \in \text{dom}(\sigma) \):

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

**Concrete, explicit:**

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

**Alternative:** **symbolic** system state

\[ \sigma = \{c_1 \mapsto \{p \mapsto \emptyset, n \mapsto \{c_2\}\}, c_2 \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, d \mapsto \{x \mapsto 23\}\} \]
You Are Here.

Course Map

\[ CD, SM \]

\[ \mathcal{F} = (\mathcal{I}, \mathcal{E}, V, \text{attr}) \]

\[ SM \]

\[ \varphi \in \text{OCL} \]

\[ expr \]

\[ CD, SD \]

\[ M = \Sigma_{\mathcal{F}, A_{\mathcal{F}}, \rightarrow_{SM}} \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \ldots \]

\[ B = (Q_{SD}, q_0, A_{\mathcal{F}}, \rightarrow_{SD}, F_{SD}) \]

\[ w_\pi = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \]

\[ G = (N, E, f) \]

\[ \mathcal{D} \]

\[ \mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{f}) \]

\[ \mathcal{UML} \]

Mathematics
References


