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]
    - Booch Method and Notation [Booch, 1993]
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]

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? . . . ?

---

**Course Map Revisited**
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

<table>
<thead>
<tr>
<th>Activities</th>
<th>State Machines</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

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

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
  - \( C_0, C_1, or C^* \), where \( C \in \mathcal{C} \) (written \( v : \tau \) or \( v : C_0, C_1, \) or \( v : C^* \)),
- \( atr : \mathcal{C} \to 2^V \) maps each class to its set of attributes.

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

\[ S = (\mathcal{F}, \mathcal{C}, V, atr) \]

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

Example:

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

Basic Object System Signature Another Example

\[ S = (\mathcal{F}, \mathcal{C}, V, atr) \]

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

Example:

\[ S_1 = (\{\text{Int}, \text{Comp}\}, \{C, D\}, \{x : \text{Int}, p : C_{0,1}, n : C^*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]
**Basic Object System Structure**

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

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

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

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

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

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

- \( C_0 \) and \( C_0, 1 \) for \( C \in \mathcal{C} \) are mapped to \( 2^{\mathcal{S}(C)} \).

We use \( \mathcal{S}(\mathcal{C}) \) to denote \( \bigcup_{C \in \mathcal{C}} \mathcal{S}(C) \); analogously \( \mathcal{S}(\mathcal{C}_*) \).

**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

\[ \mathcal{S}_0 = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}, p : C_0, 1, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{x\}\}) \]

Recall: by definition, seek a \( \mathcal{S} \) which maps

- \( \tau \in \mathcal{P} \) to some \( \mathcal{S}(\tau) \),
- \( c \in \mathcal{C} \) to some identities \( \mathcal{S}(C) \) (infinite, disjoint for different classes),
- \( C_* \) and \( C_0, 1 \) for \( C \in \mathcal{C} \) to \( \mathcal{S}(C_0, 1) = \mathcal{S}(C_*) = 2^{\mathcal{S}(C)} \).

\[
\begin{align*}
\mathcal{S}(\text{Int}) &= \mathbb{Z} \quad \text{(could also be \{-100, \ldots, 100\})} \\
\mathcal{S}(C) &= \mathbb{N}^* \times \{\text{C}\} \ni \{\text{C}, 2\text{C}, 3\text{C}, \ldots\} \\
\mathcal{S}(D) &= \mathbb{N}^* \times \{\text{D}\} \ni \{1\text{D}, 2\text{D}, 3\text{D}, \ldots\} \\
\mathcal{S}(C_0, 1) = \mathcal{S}(C_*) &= 2^{\mathcal{S}(C)} \\
\mathcal{S}(D_0, 1) = \mathcal{S}(D_*) &= 2^{\mathcal{S}(D)}
\end{align*}
\]
**System State**

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

$$\sigma : \mathcal{D}(\mathcal{C}) \rightarrow (V \rightarrow (\mathcal{D}(\mathcal{F}) \cup \mathcal{D}(\mathcal{C}_{*}))).$$

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 $\Sigma_{\mathcal{D}}^{\mathcal{S}}$ to denote the set of all system states of $\mathcal{S}$ wrt. $\mathcal{D}$.

---

**System State Example**

**Signature, Structure:**

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

$\mathcal{D}(\text{Int}) = \mathbb{Z}$, $\mathcal{D}(C) = \{1_{C}, 2_{C}, 3_{C}, ...\}$, $\mathcal{D}(D) = \{1_{D}, 2_{D}, 3_{D}, ...\}$

**Wanted:** $\sigma : \mathcal{D}(\mathcal{C}) \rightarrow (V \rightarrow (\mathcal{D}(\mathcal{F}) \cup \mathcal{D}(\mathcal{C}_{*})))$ such that

- $\text{dom}(\sigma(u)) = \text{atr}(C)$,
- $(\sigma(u))(v) \in \mathcal{D}(\tau)$ if $v : \tau, \tau \in \mathcal{F}$,
- $(\sigma(u))(v) \in \mathcal{D}(D_{*})$ if $v : D_{*}$ with $D \in \mathcal{C}$.
System State Example

Signature, Structure:

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

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

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

- \( \text{dom}(\sigma(u)) = \text{atr}(C) \),
- \( \sigma(u)(v) \in \mathcal{P}(\tau) \) if \( v : \tau, \tau \in \mathcal{F} \),
- \( \sigma(u)(v) \in \mathcal{P}(C_s) \) if \( v : D_s \) 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

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

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

You Are Here.
$\mathcal{C} = \langle \mathbb{N}, E, f \rangle$

$\psi \in \text{OCL expr}$

$\mathcal{B} = \langle \mathbb{Q}_\mathcal{B}, q_0, \mathbb{A}_\mathcal{B}, \rightarrow, \mathbb{F}_\mathcal{B} \rangle$
References


