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'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 Signatures, Structure, and System State

Why (of all things) UML?

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

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
  - Influenza of notations and methods, most prominent:
    - Object-Oriented Technique (OMT) [Rumbaugh et al., 1990]

- 1970’s, Soft
  - Med. fail
  - Language:
    - Booch Method and Notations [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., 1991]
  - 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 0.0, 1.x — Standardised, exchangeable between modelling tools.
- Since 2005: UML 2.x

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 standards say 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.
Is it true? Or not at all? Why? Why not? How can it be achieved?
Which ones are really only hopes and expectations? ...

UML: Semantic Areas

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

The Plan

- Overall aim: a formal language for software blueprints.
- Approach:
  - Common semantical domain
  - UML fragments as syntax
  - Abstract representation of diagrams
  - Internal semantics of UML members
  - Usage meaning for diagrams
  - Unified, consistent...

UML Overview

Common Expectations on UML
Basic Object System Signature

Definition. A (Basic) Object System Signature is a quadruple 
\[ \mathcal{S} = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mu) \]
where:
- \( \mathcal{F} \) is a set of (basic) types,
- \( \mathcal{V} \) is a finite set of classes,
- \( \mathcal{E} \) is a finite set of typed attributes, i.e., each \( \mathcal{E} \) has type \( r \in \mathcal{F} \) or
- \( C \in \mathcal{V} \), where \( C \in \mathcal{V} \) (written \( v \) or \( C \), \( v \)),
- \( \mu : \mathcal{E} \rightarrow \{ 0 \} \) maps each class to its set of attributes.

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

Basic Object System Signature Example

Example: \( \mathcal{S} = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mu) \) where:
- \( \mathcal{F} = \{ \text{int}, \text{float}, \text{string} \} \),
- \( \mathcal{V} = \{ C_1, C_2, C_3 \} \),
- \( \mathcal{E} = \{ \text{age}, \text{weight}, \text{height} \} \),
- \( \mu : \text{age} \rightarrow \{ 0 \} \) maps class to attributes.

Basic Object System Structure

Definition. A Basic Object System Structure of
\[ \mathcal{S} = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mu) \]
is a function \( \mathcal{S} \) which assigns to each type a domain, i.e.,
- \( r \in \mathcal{F} \) is mapped to \( \mathcal{D}(r) \).
- \( C \in \mathcal{V} \) is mapped to an infinite set \( \mathcal{D}(C) \) of object identities.

Object identities only have the “\( \tau \)” operation, object identities of different classes are disjoint, i.e.,
\[ \forall C, D \in \mathcal{V} : C \neq D \rightarrow \mathcal{D}(C) \cap \mathcal{D}(D) = \emptyset \] .

We use \( \mathcal{D}(C) \) to denote \( \bigcup_{i=0}^{\infty} \mathcal{D}(C_i) \) analogously \( \mathcal{D}(E) \).

Note: We identify objects and object identities, because both uniquely determine each other (cf. OCL 2.0 standard).

Basic Object System Structure Example

Example: \( \mathcal{S} = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mu) \) where:
- \( \mathcal{F} = \{ \text{int}, \text{float}, \text{string} \} \),
- \( \mathcal{V} = \{ C_1, C_2, C_3 \} \),
- \( \mathcal{E} = \{ \text{age}, \text{weight}, \text{height} \} \),
- \( \mu : \text{age} \rightarrow \{ 0 \} \) maps class to attributes.
Cons.

foot soak 

A common technique in computing is to map a set of system states to a set of system states. This can be done by 

\[
\sigma \mapsto \emptyset
\]

Foot soak is the process of soaking the foot in water. This can be done by 

\[
\sigma \mapsto \emptyset
\]

A common technique in computing is to map a set of system states to a set of system states. This can be done by 

\[
\sigma \mapsto \emptyset
\]
References


