Software Design, Modelling and Analysis in UML

Lecture 02: Semantical Model

2012-10-24

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

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 Bünning, 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]

![Diagram of UML overview](image-url)
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

![Diagram of UML semantic areas and their dependencies]

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

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

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

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

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

where

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

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

where

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

Example:

\[ S = \left( \{\text{Int}, \text{Float}\}, \{C, D\}, \{x: \text{Int}, y: \text{Float}, z: \text{Float}\}, \{C \mapsto \{x\}, D \mapsto \{y, z\}\} \right) \]

Q: What is the attribute \( x: \text{Int} \)?
A: Renamer consistently.
**Basic Object System Structure**

**Definition.** A Basic Object System Structure of
\[
S = (\mathcal{T}, \mathcal{C}, \mathcal{V}, \text{atr})
\]
is a domain function \( D \) which assigns to each type a domain, i.e.
- \( \tau \in \mathcal{T} \) is mapped to \( D(\tau) \),
- \( C \in \mathcal{C} \) 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.
\[ \forall C, D \in \mathcal{C} : C \neq D \rightarrow D(C) \cap D(D) = \emptyset. \]
- \( C_\ast \) and \( C_{0,1} \) for \( C \in \mathcal{C} \) are mapped to \( 2^{D(C)} \).

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

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

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

\[
\begin{align*}
D(\text{Int}) &= \mathbb{Z} \\
D(C) &= \mathbb{N}^* \times \{c\} \cong \{1_c, 2_c, 3_c, \ldots\} \\
D(D) &= \mathbb{N}^* \times \{d\} \cong \{1_d, 2_d, 3_d, \ldots\} \\
D(C_{0,1}) &= D(C_\ast) = 2^{D(C)} \\
D(D_{0,1}) &= D(D_\ast) = 2^{D(D)}
\end{align*}
\]
Definition. Let $\mathcal{I}$ be a structure of $\mathcal{I} = (\mathcal{C}, \mathcal{E}, V, \text{atr})$. A system state of $\mathcal{I}$ wrt. $\mathcal{I}$ is a type-consistent mapping $\sigma : \mathcal{I}(\mathcal{C}) \not\rightarrow (V \not\rightarrow (\mathcal{I}(\mathcal{I}) \cup \mathcal{I}(\mathcal{E}))).$

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

- $\text{dom}(\sigma(u)) = \text{atr}(C)$
- $\left\{\sigma(u)(v) \in \mathcal{I}(\tau) \text{ if } v : \tau, \tau \in \mathcal{I}\right\}$
- $\left\{\sigma(u)(v) \in \mathcal{I}(D_*) \text{ if } v : D_0 \text{ or } v : D_* \text{ with } D \in \mathcal{C}\right\}$

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

We use $\Sigma_D^\mathcal{I}$ to denote the set of all system states of $\mathcal{I}$ wrt. $\mathcal{I}$.

---

System State Example

**Signature, Structure:**

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

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

**Wanted:** $\sigma : \mathcal{I}(\mathcal{C}) \not\rightarrow (V \not\rightarrow (\mathcal{I}(\mathcal{I}) \cup \mathcal{I}(\mathcal{E})))$ such that

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

**Signature, Structure:**

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

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

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

- \(\text{dom}(\sigma(u)) = \text{atr}(C)\),
- \(\sigma(u)(v) \in \mathcal{D}(\tau)\) if \(v: \tau, \tau \in \mathcal{F}\),
- \(\sigma(u)(v) \in \mathcal{D}(C_\ast)\) if \(v: D_\ast\) 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 \(c_1, c_2 \in \mathcal{D}(C), d \in \mathcal{D}(D), c_1 \neq c_2.\)
\( \varphi \in \text{OCL} \)

\( \mathcal{M} = (\Sigma, A_{\varphi}, \rightarrow_{SM}) \)

\( \pi = (\sigma_0, \epsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \epsilon_1) \cdots \quad w_{\pi} = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \)

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

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

\( \text{References} \)
References


