Software Design, Modelling and Analysis in UML

Lecture 1: Introduction

2011-10-25

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

This Lecture:

- **Educational Objectives:** After this lecture you should
  - be able to explain the term *model*.
  - know the idea (and hopes and promises) of *model-based* SW development.
  - be able to explain how *UML* fits into this general picture.
  - know what we’ll do in the course, and why.
  - thus be able to decide whether you want to stay with us...

- **Content:**
  - Analogy: Model-based/-driven development by construction engineers.
  - Software engineers: “me too” – Model-based/-driven Software Engineering.
  - UML Mode of the Lecture: Blueprint.
  - Contents of the course
  - Formalia
Modelling

Disclaimer

- The following slides may raise thoughts such as:
  - “everybody knows this”,
  - “completely obvious”,
  - “trivial”,
  - “clear”,
  - “irrelevant”,
  - “oversimplified”
  - …

Which is true, in some sense,

- but: “everybody” is a strong claim, and I want to be sure that this holds for the audience from now on.
  In other words: that we’re talking about the same things.
An Analogy: The House-Building Problem (Oversimplified)

**Given** a set of **Requirements**, such as:
- The house shall fit on the given piece of land.
- Each room shall have a door, the doors shall open.
- The given furniture shall fit into the living room.
- The bathroom shall have a window.
- The cost shall be in budget.

**Wanted**: a house which satisfies the requirements.

Now, strictly speaking, a house is a **complex system**:
- Consists of a huge number of bricks.
- Consists of subsystems, such as windows.
- Water pipes and wirings have to be in place.
- Doors have to open consistently.
- Floors depend on each other (load-bearing walls).
- 
  How do construction engineers **handle** this complexity...?

**Approach: Floorplan**

1. **Requirements**
   - Shall fit on given piece of land.
   - Each room shall have a door.
   - Furniture shall fit into living room.
   - Bathroom shall have a window.
   - Cost shall be in budget.

2. **Design**

3. **System**

**Observation**: Floorplan **abstracts** from, e.g., . . .
- kind, number, and placement of bricks,
- subsystem details (e.g., window style),
- water pipes/wiring, and
### Approach: Floorplan

#### 1. Requirements

- Shall fit on given piece of land.
- Each room shall have a door.
- Furniture shall fit into living room.
- Bathroom shall have a window.
- Cost shall be in budget.

#### 2. Design

![Floorplan](http://wikimedia.org (CC nc-sa 3.0, Ottoklages))

#### 3. System

![System](http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82))

**Observation:** Floorplan preserves, e.g.,

- house and room extensions (to scale),
- presence/absence of windows and doors,
- placement of subsystems (such as windows).

---

**Floorplan as an Abstraction**

- Floorplan $F$ denotes a set $\gamma(F)$ of houses (concretisations of $F$), which differ, e.g. in colour of bricks, or making of windows.
- Floorplan $F$ represents house $H$ according to abstraction $\alpha$.
- By adding information to $F$ (such as making of windows), we can narrow down $\gamma(F)$.
What is it good for? Build by Plan.

- As said before, the floorplan abstraction \( \alpha \) preserves some properties. For instance, we have:
  Room \( R \) has window in \( H \) if and only if \( R \)-representation in \( \alpha(H) \) has window.

- And we have the general rule:
  If a house \( H' \) is (or: will have been) built according to plan \( F \), and if plan \( F \) has property \( \phi \), and if \( \alpha/\gamma \) preserve this property, then \( H' \) has (or: will have) property \( \phi \).

- So we can answer some questions about \( H \) before even building it, e.g.:
  - Bathroom shall have a window.
  - Shall fit on given piece of land.
  - Each room shall have a door.
  - Furniture shall fit into living room.
  - Cost shall be in budget.

- And: it’s typically easier (and cheaper) to correct errors in the plan, rather than in the finished house.

“Silver Bullet” or Can Anything Go Wrong...?

- If the requirements are already contradictory (or inconsistent), then there is no sense in drawing a plan.

Example:

- The house shall fit on the given piece of land.
- The given furniture shall fit into the living room.

What if the land is 10m narrow and the couch is 11m \( \times \) 11m?
**Good for Anything Else? Documentation.**

- **Given:** a house.
- **Wanted:** a concise description for potential buyers.
- **Approach:** draw a floorplan.

**Distinguish:**
- Sometimes the plan $F$ is first, and the realisation $H \in \gamma(F)$ comes later.
- Sometimes the realisation $H$ is first, and the “plan” $F = \alpha(H)$ comes later.

**What’s the Essence?**

**Definition.** [Folk] A model is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

**Definition.** [Glinz, 2008, 425]
A model is a concrete or mental image (Abbildung) of something or a concrete or mental archetype (Vorbild) for something.

Three properties are constituent:

(i) the image attribute (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,

(ii) the reduction attribute (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,

(iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.
Software System (Very Abstract View)

We see software $M$ as a transition system.

- It has a (possibly infinite) set of states $S$, (structure)
- an initial state $s_0$, and
- a (possibly $L$-labelled) transition relation

\[ \rightarrow \subseteq S \times L \times S. \]  

(behaviour)

Software may have infinite and finite runs, i.e. sequences of consecutive states.

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]

(initial, $s_i \in \mathcal{S}$, $(s_i, s_{i+1}) \in \mathcal{R}$ for all $i \in \mathbb{N}$)
Software System (Very Abstract View)

We see software $M$ as a transition system.
- It has a (possibly infinite) set of states $S$, $(\text{structure})$
- an initial state $s_0$, and
- a (possibly $L$-labelled) transition relation
  
  $$\rightarrow \subseteq S \times L \times S.$$  
  $(\text{behaviour})$

Software may have infinite and finite runs, i.e. sequences of consecutive states.

The software engineering problem:
- Given: informal requirements $\varphi$.
- Desired: correct software, i.e. software $M$ such that $M$ satisfies $\varphi$.

Two prominent obstacles:
- Getting $\varphi$ formal in order to reason about $\varphi$ and $M$, e.g. prove $M$ correct.
- $M$ typically too large to “write it down” at once.

Model-Driven Software Engineering
Model-Driven Software Engineering with UML

Class Diagram  

Sequence Diagram  

requirements model

requirements/constraints

design

system model

Implementation

Class Diagram

Class Diagram'

Class Diagram''

Sequence Diagram'

State Machine

State Machine'

generate/program

elicit

refine

refine

refine

OMG, 2007a, 135
Model-Driven Software Engineering with UML

[Diagram of UML diagrams showing the process of elicitation, refinement, and generation, with arrows connecting Class Diagrams, Sequence Diagrams, State Machines, and Implementation to indicate the flow of requirements, model, constraints, design, and system model.]
Model-Driven Software Engineering with UML

- **Class Diagram**: Elicit requirements model
- **Sequence Diagram**: Elicit requirements/constraints
- **State Machine**: Design
- **State Machine**: System model

---

DialToone

- Dialing
- Active
- Idle
- Busy
- Connected
- Context
- Dial digit(n) [valid]
- Dial digit(n) [invalid]
- Get dial tone
- Play busy tone
- Play ringing tone
- Enable speech
- Disconnect
- Active entry
- Abort
- Terminate

---

[OMG, 2007b, 567]
**Needed: A Modelling Language for SW-Engineering**

- What would be a “from scratch” approach?
  (i) Define a **formal** language to define requirements and designs.
  (ii) Equip it with a **formal** semantics.
  (iii) Define consistency/satisfaction relation in terms of semantics.

- The approach in this course:
  (i) Introduce a common semantical domain — what is a very abstract mathematical characterisation of **object based** transitions **systems**?
    *Why? Because in the end SW-Engineering is about the creation of (object based) transitions systems and Modeling is about describing them.*
  (ii) Take (a fragment of) the visual formal language **UML** as **syntax**.
  (iii) Introduce an abstract mathematical **representation of diagrams**.
    *Why? Because it is easier to handle than “pictures”; it abstracts from details such as graphical layout (which don’t contribute to the semantics — note: in floor plans it does).*
  (iv) Study the **UML** standard documents for the **informal semantics**.
  (v) Define a mapping from (abstract representations of) diagrams to the semantical domain: assign meaning to diagrams.
  (vi) Define (in terms of the meaning) when a diagram is, e.g., **consistent**.

---

**Course Map**

```
Model
  \( (\mathcal{F}, \mathcal{E}, V, \mathcal{A}, \mathcal{X}) \), SM
.expr \( A_{\mathcal{X}, \rightarrow_{SM}} = M \)

G = (N, E, f)

\( B = (Q_{SD}, q_0, A_{\mathcal{X}, \rightarrow_{SD}}, F_{SD}) \)
```

---

*Mathematics*
**Consequences of the Pragmatic Attribute**

Recall [Glinz, 2008, 425]:

> [...] (iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.

**Examples for context/purpose:**

- **Floorplan as sketch:**

- **Floorplan as blueprint:**

- **Floorplan as program:**

![Floorplan as sketch](image1)

![Floorplan as blueprint](image2)

![Floorplan as program](image3)
Actually, the last slide is inspired by Martin Fowler, who puts it like this:

“[…] people differ about what should be in the UML because there are differing fundamental views about what the UML should be.

I came up with three primary classifications for thinking about the UML: UmlAsSketch, UmlAsBlueprint, and UmlAsProgrammingLanguage. ([… ] S. Mellor independently came up with the same classifications.)

So when someone else’s view of the UML seems rather different to yours, it may be because they use a different UmlMode to you.”

Claim:

• And this not only applies to UML as a language (what should be in it?)
• but at least as well to individual UML models.

<table>
<thead>
<tr>
<th>Sketch</th>
<th>Blueprint</th>
<th>ProgrammingLanguage</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this UmlMode developers use the UML to help communicate some aspects of a system. […]</td>
<td>In forward engineering the idea is that blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up. That design should be sufficiently complete that all design decisions are laid out and the programming should follow as a pretty straightforward activity that requires little thought. […]</td>
<td>If you can detail the UML enough, and provide semantics for everything you need in software, you can make the UML be your programming language. Tools can take the UML diagrams you draw and compile them into executable code. The promise of this is that UML is a higher level language and thus more productive than current programming languages. The question, of course, is whether this promise is true. I don’t believe that graphical programming will succeed just because it’s graphical. […]</td>
</tr>
<tr>
<td>Sketches are also useful in documents, in which case the focus is communication rather than completeness. […]</td>
<td>The tools used for sketching are lightweight drawing tools and often people aren’t too particular about keeping to every strict rule of the UML. Most UML diagrams shown in books, such as mine, are sketches. Their emphasis is on selective communication rather than complete specification. Hence my sound-bite “comprehensiveness is the enemy of comprehensibility”</td>
<td>Forward engineering tools support diagram drawing and back it up with a repository to hold the information. […]</td>
</tr>
</tbody>
</table>
The "mode" fitting the lecture best is **AsBlueprint**.

The purpose of the lecture’s formal semantics is:
- to be precise to **avoid misunderstandings**.
- to allow formal **analysis of consistency/implication**
  on the **design level** — find errors early.

while being consistent with the (informal semantics) from the standard [?, OMG, 2007b] as far as possible.
UML-Mode of the Lecture: As Blueprint

- The "mode" fitting the lecture best is AsBlueprint.

- The purpose of the lecture’s formal semantics is:
  - to be precise to avoid misunderstandings.
  - to allow formal analysis of consistency/implication on the design level — find errors early.
  - while being consistent with the (informal semantics) from the standard [?, OMG, 2007b] as far as possible.

- Being precise also helps for mode AsSketch:
  - it should be easier to “fill in” missing parts or resolve inconsistencies.

- Lecture serves as a starting point to define your semantics for your context/purpose (maybe obtaining a Domain Specific Language).

- Lecture could be worked out into mode AsProgrammingLanguage.

Course Overview
Table of Contents

- Motivation and Overview (VL 01)
- Semantical Domain (VL 02)
- OCL (VL 03)
- Object Diagrams (VL 04)
- Modelling Structure: Class Diagrams (VL 05–08)
- Modelling Behaviour
  - Constructive: State Machines (VL 09–16)
  - Reflective: Live Sequence Charts (VL 17–19)
- Inheritance (VL 20–21)
- Meta-Modeling (VL 22)
- Putting it all together: MDA, MDSE (VL 23)

Course Path: Over Map

- Motivation
- Semantical Domain
- OCL
- Object Diagrams
- Class Diagrams
- State Machines
- Live Sequence Charts
- Real-Time
- Components
- Inheritance
- Meta-Modeling
- MDA, MDSE
Table of Non-Contents

**Everything else**, including

- **Development Process**
  UML is only the language for artefacts. But: we’ll discuss exemplarily, where in an abstract development process which means could be used.

- **How to come up with a good design**
  UML is only the language to write down designs. But: we’ll have a couple of examples.

- **Requirements Management**
  Versioning, Propagation of Changes

- **Every little bit and piece of UML**
  Boring. Instead we learn how to read the standard.

- **Object Oriented Programming**
  Interesting: inheritance is one of the last lectures.
Formalia

Formalia: Event

- **Lecturer:** Dr. Bernd Westphal
- **Support:** Evis Plaku
- **Homepage:**
  
  http://swt.informatik.uni-freiburg.de/teaching/
  
  winter-term-2011-2012/sd

- **Questions:**
  - **“online”:**
    (i) ask immediately or in the break
  - **“offline”:**
    (i) try to solve yourself
    (ii) discuss with colleagues
    (iii) Exercises: contact tutor by mail (cf. homepage)
    Rest: contact lecturer by mail (cf. homepage)
    or just drop by: Building 52, Room 00-020
**Formalia: Dates/Times, Break**

- **Location:**
  - Tuesday, Wednesday: here (bldg. 106, room 00-007)

- **Schedule:**
  - **Week \( N \),** Wednesday, 12–14 lecture (exercise sheet \( K \) online)
  - **Week \( N + 1 \),** Tuesday, 12–14 lecture
    - Wednesday, 12–14 lecture
  - **Week \( N + 2 \),** Monday, 9:00 (exercises \( K \) early submission)
    - Tuesday, 12:00 (exercises \( K \) late submission)
    - 12–14 tutorial

  With a prefix of lectures, see homepage for details.

- **Break:**
  - Unless a majority objects now,
    - we’ll have a **15 min. break** in the middle of each event from now on.

**Formalia: Lectures**

- **Course language:** English
  - (slides/writing, presentation, questions/discussions)

- **Presentation:**
  - half slides/half on-screen hand-writing — for reasons

- **Script/Media:**
  - slides with annotations on homepage, 2-up for printing, typically soon after the lecture
  - recording on eLectures portal with max. 1 week delay (link on homepage)

- **Interaction:**
  - absence often moaned but it takes two,
  - so please ask/comment immediately.
Formalia: Exercises and Tutorials

- **Schedule/Submission:**
  - **hand-out** on Wednesday after lecture,
  - **early turn in** on following Monday by 9:00 local time
  - **regular turn in** on following Tuesday by 12:00 local time
  - should work in groups of **approx. 3**, clearly give **names** on submission
  - please submit **electronically** by Mail to B. Westphal (cf. homepage), **paper submissions** are **tolerated**

- **Rating system:** “most complicated rating system ever”
  - **Admission points** (good-will rating, upper bound)
    - (“reasonable proposal given student’s knowledge before tutorial”)
  - **Exam-like points** (evil rating, lower bound)
    - (“reasonable proposal given student’s knowledge after tutorial”)

  10% **bonus** for **early** submission.

- **Tutorial:** Plenary.
  - Together develop **one** good proposal,
    - starting from discussion of the early submissions (anonymous).

Formalia: Exam

- **Exam Admission:**
  
  Achieving **50%** of the regular **admission points** in total is **sufficient** for admission to exam.

  Typically, 20 regular admission points per exercise sheet.

- **Exam Form:**
  - oral for BSc and on special demand,
  - **written** for everybody else (if sufficiently many candidates remain).

  Scores from the exercises **do not** contribute to the final grade.
Formalia: Evaluation

• **Mid-term Evaluation:**
  - We will have a mid-term evaluation (early December, roughly 1/3 of the course’s time).
  - If you decide to leave the course earlier you may want to do us a favour and tell us the reasons – by participating in the mid-term evaluation (will be announced on homepage).
  - Note: we’re **always** interested in comments/hints/proposals/wishes/… concerning form or content.

Feel free to approach us (tutors, me) in any form. **We don’t bite.**

---

**Literature**

Literature: UML

- OMG: Unified Modeling Language Specification, Superstructure, 2.1.2
- OMG: Object Constraint Language Specification, 2.0
  All three: http://www.omg.org (cf. hyperlinks on course homepage)

Literature: Modelling

  http://www.springerlink.com/content/0170-6012
Questions?

References


