

Software Design, Modelling and Analysis in UML

Lecture 02: Semantical Model

2013-10-23

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

- 02 - 2013-10-23 - main -

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

- 02 - 2013-10-23 - Prelim -

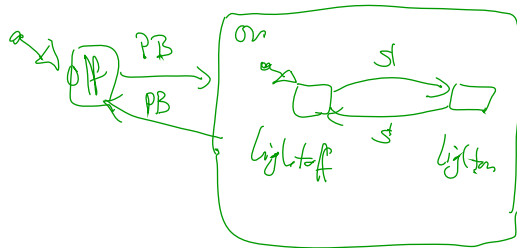
Why (of all things) UML?

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.

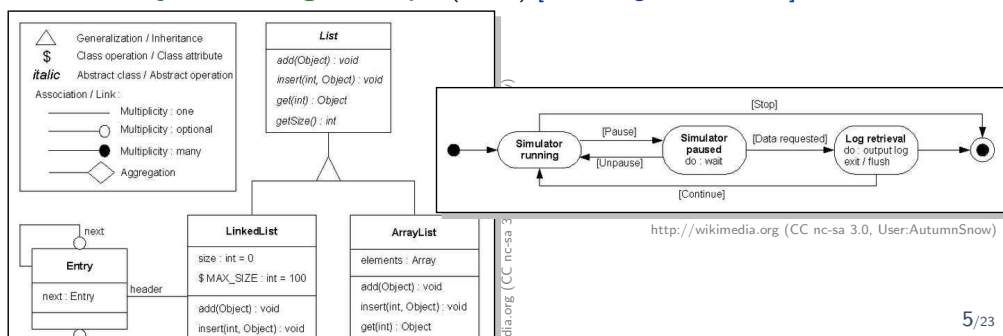
A Brief History of UML

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's, Software Crisis**TM
— Idea: learn from engineering disciplines to handle growing complexity.
Languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980's: Statecharts** [Harel, 1987], **StateMate**TM [Harel et al., 1990]



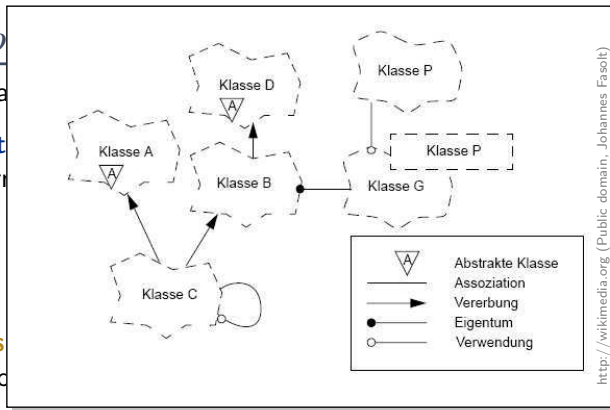
A Brief History of UML

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's, Software Crisis**TM
— Idea: learn from engineering disciplines to handle growing complexity.
Languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980's: Statecharts** [Harel, 1987], **StateMate**TM [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

- Boxes/lines a
- **1970's, Soft**
— Idea: learn
- Languages:
- Mid **1980's**:
- Early **1990's**
— Inflation c



ages.
 complexity.
 aims
 l., 1990]
 amming

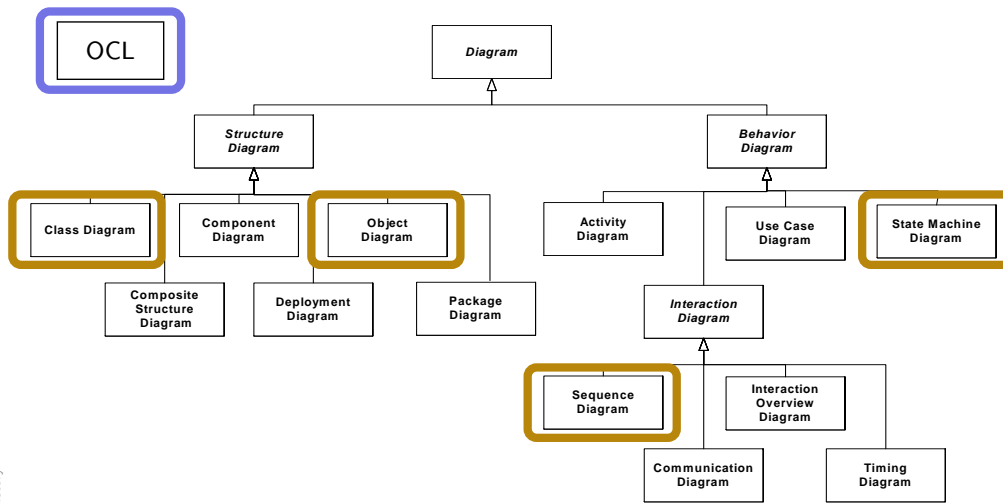
- **Object-Modeling Technique (OMT)** [Rumbaugh et al., 1990]
- **Booch Method and Notation** [Booch, 1993]

A Brief History of UML

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's, Software Crisis**TM
— Idea: learn from engineering disciplines to handle growing complexity.
Languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980's**: **Statecharts** [Harel, 1987], **StateMate**TM [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**



- 02 - 2013-10-23 - Shistory -

Figure A.5 - The taxonomy of structure and behavior diagram

[Dobing and Parsons, 2006]

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...

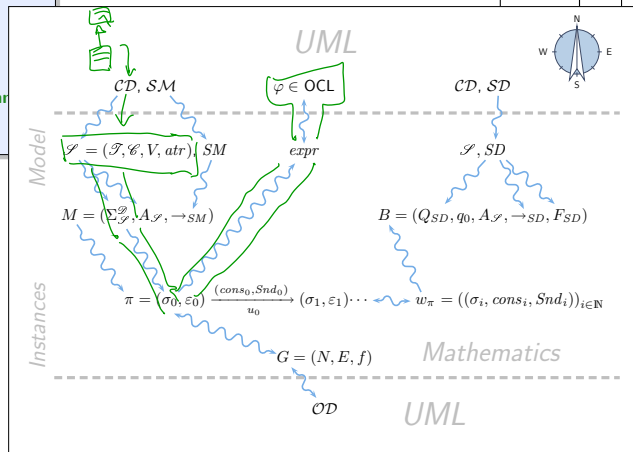
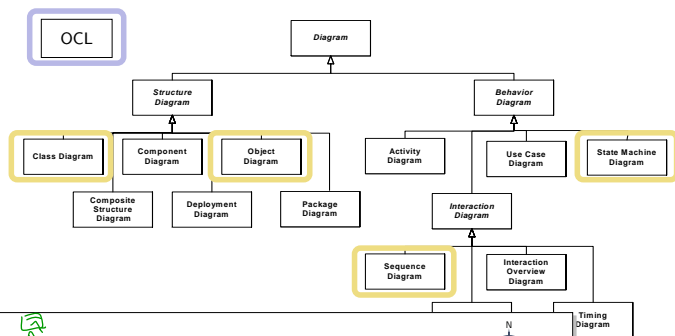
Seriously: After the course, you should have an own opinion on each of these claims. In how far/in what sense does it hold? Why? Why not? How can it be achieved? Which ones are really only hopes and expectations? ... ?

- 02 - 2013-10-23 - Shistory -

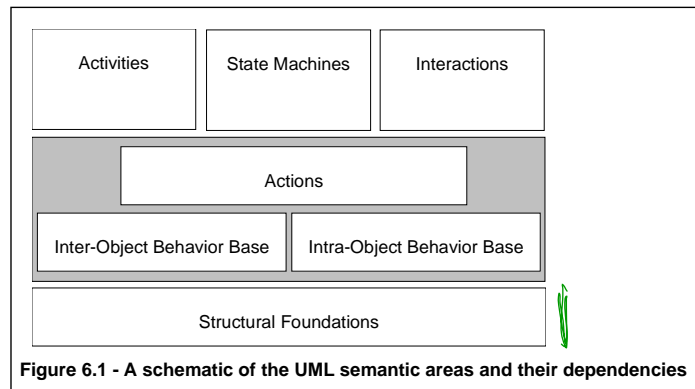
Course Map Revisited

The Plan

- Recall:
- **Overall aim:** a formal language for software blueprints.
 - **Approach:**
 - (i) Common semantical domain.
 - (ii) UML fragments as **syntax**.
 - (iii) Abstract **representation of diagrams**.
 - (iv) **Informal semantics:** UML standard
 - (v) **assign meaning to diagram**
 - (vi) Define, e.g., **consistency**.



UML: Semantic Areas



[OMG, 2007b, 11]

Common Semantical Domain

Basic Object System Signature

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

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

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

for each class $D \in \mathcal{C}$
there are two different
types:

$D_{0,1}$ D_* $\alpha:$ $\alpha:$

D_x D_{\emptyset} \triangle \square

in other words:

$$typeof : V \rightarrow \mathcal{T} \cup \{D_{0,1}, D_x, D_{\emptyset}\}$$

total function powerset of V

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

Basic Object System Signature Example

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

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

Example:

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

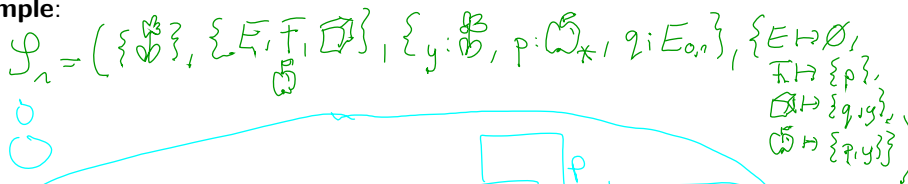
basic types classes attributes
 ↓ ↓ ↓
 attribute x has type Int
 $atr(C) = \{p, n\}$ $atr(D) = \{x\}$
 maps to

Basic Object System Signature Another Example

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

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

Example:



-02 - 2013-10-23 - Semdom -

14/23

Basic Object System Structure

Definition. A Basic Object System Structure of

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

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

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

Note: Object identities only have the "=" operation; object identities of different classes are disjoint, i.e. $\forall C, D \in \mathcal{C} : C \neq D \rightarrow \mathcal{D}(C) \cap \mathcal{D}(D) = \emptyset$.

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

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

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

-02 - 2013-10-23 - Semdom -

15/23

Basic Object System Structure Example

Wanted: a structure for signature

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

Recall: by definition, seek a \mathcal{D} which maps

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

alternative
choice:

$$\begin{aligned} \mathcal{D}(Int) &= \mathbb{Z} \\ \mathcal{D}(C) &= \mathbb{N}^+ \times \{C\} = \{1_C, 2_C, 3_C, \dots\} \\ \mathcal{D}(D) &= \mathbb{N}^+ \times \{D\} = \{1_D, 2_D, 3_D, \dots\} \\ \mathcal{D}(C_{0,1}) = \mathcal{D}(C_*) &= 2^{\mathbb{N}^+ \times \{C\}} \\ \mathcal{D}(D_{0,1}) = \mathcal{D}(D_*) &= 2^{\mathbb{N}^+ \times \{D\}} \end{aligned}$$

$\mathcal{D}_2(Int) = \{1, 2, \dots, 127\}$
 $\mathcal{D}_2(C) = \{1, 3, 5, \dots\}$
 $\mathcal{D}_2(D) = \{2, 4, 6, \dots\}$
 $2^{\mathcal{D}(C)}$
 $2^{\mathcal{D}(D)}$

System State

Definition. Let \mathcal{D} be a structure of $\mathcal{S} = (\mathcal{T}, \mathcal{C}, V, atr)$.

A **system state** of \mathcal{S} wrt. \mathcal{D} is a **type-consistent** mapping

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

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

- $\text{dom}(\sigma(u)) = atr(C)$ ||
- $(\sigma(u))(v) \in \mathcal{D}(\tau)$ if $v : \tau, \tau \in \mathcal{T}$
- $(\sigma(u))(v) \in \mathcal{D}(D_*)$ if $v : D_{0,1}$ or $v : D_*$ with $D \in \mathcal{C}$

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

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

System State Example

Signature, Structure:

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

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

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

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

- $\sigma_1 = \emptyset$ ← empty function
- $\sigma_2 = \{ 1_C \mapsto \{ p \mapsto \{ 1_D \}, n \mapsto \{ 5_C, 6_C \} \}, 2_C \mapsto \{ x \mapsto 3 \} \}$
- one way to read out:
= object 1_C has a p-link to 1_C (i.e. to itself)
= object 1_C refers to objects $5_C, 6_C$ via link n
- With \mathcal{D}_2 :
- $\sigma_3 = \{ 5 \mapsto \{ p \mapsto \{ 17 \}, n \mapsto \emptyset \} \}$

System State Example

Signature, Structure:

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

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

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

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

Concrete, explicit:

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

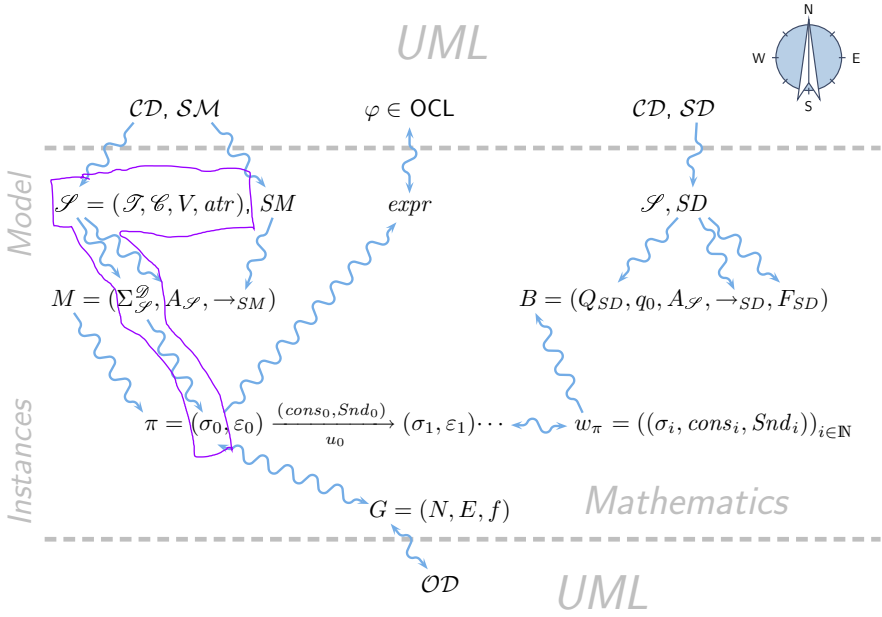
Alternative: symbolic system state

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

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

You Are Here.

Course Map



References

References

- [Booch, 1993] Booch, G. (1993). *Object-oriented Analysis and Design with Applications*. Prentice-Hall.
- [Dobing and Parsons, 2006] Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, 49(5):109–114.
- [Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.
- [Harel et al., 1990] Harel, D., Lachover, H., et al. (1990). Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414.
- [Jacobson et al., 1992] Jacobson, I., Christerson, M., and Jonsson, P. (1992). *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley.
- [Kastens and Büning, 2008] Kastens, U. and Büning, H. K. (2008). *Modellierung, Grundlagen und Formale Methoden*. Carl Hanser Verlag München, 2nd edition.
- [OMG, 2006] OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.
- [OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.
- [OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.
- [Rumbaugh et al., 1990] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenzen, W. (1990). *Object-Oriented Modeling and Design*. Prentice Hall.