Software Design, Modelling and Analysis in UML

Lecture 5: Object Diagrams

2015-11-05

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- OCL Semantics

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does it mean that an OCL expression is satisfiable?
  - When is a set of OCL constraints said to be consistent?
  - What is an object diagram? What are object diagrams good for?
  - When is an object diagram called partial? What are partial ones good for?
  - When is an object diagram an object diagram (wrt. what)?
  - How are system states and object diagrams related?
  - Can you think of an object diagram which violates this OCL constraint?
- **Content:**
  - OCL: consistency, satisfiability
  - Object Diagrams
  - Example: Object Diagrams for Documentation

OCL Satisfaction Relation

OCL Satisfaction Relation

In the following, $\mathscr{S}$ denotes a signature and $\mathscr{D}$ a structure of $\mathscr{S}$.

Definition (Satisfaction Relation).
Let φ be an OCL constraint over $\mathscr{S}$ and $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ a system state.
We write

- $\sigma \models \varphi$ if and only if $I[\![\varphi]\!](\sigma, \emptyset) = \mathit{true}$.
- $\sigma \not\models \varphi$ if and only if $I[\![\varphi]\!](\sigma, \emptyset) = \mathit{false}$.

Note: In general we **can't** conclude from $\neg(\sigma \models \varphi)$ to $\sigma \not\models \varphi$ or vice versa.

OCL Consistency

Definition (Consistency). A set $\mathit{Inv} = \{\varphi_1, \dots, \varphi_n\}$ of OCL constraints over $\mathscr{S}$ is called consistent (or satisfiable) if and only if there exists a system state of $\mathscr{S}$ wrt. $\mathscr{D}$ which satisfies all of them, i.e. if

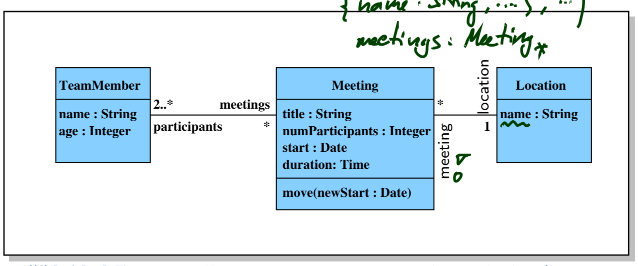
$$\exists \sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}} : \sigma \models \varphi_1 \quad \wedge \dots \wedge \quad \sigma \models \varphi_n$$

and inconsistent (or unsatisfiable) otherwise.

Example: OCL Consistent?

$\mathscr{S} = (\{String, \dots\}, \{TeamMember, \dots\}, \{name : String, \dots, meetings : Meeting_*\}, \dots)$

Location ↦ {meeting, ...}
TeamMember ↦ {meetings, ..}



((C) Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/)

- context $\mathit{Location}$ inv : name = 'Lobby' implies $\mathit{meeting}$ -> $\mathit{isEmpty}()$ consis.
- context $\mathit{Meeting}$ inv : title = 'Reception' implies $\mathit{location}$. name = 'Lobby'
- allInstances$_{\mathit{Meeting}}$ -> exists($w : \mathit{Meeting} \mid w . \mathit{title}$ = 'Reception') not consistent

Deciding OCL Consistency

- Whether a set of OCL constraints is consistent or not **is in general not as obvious** as in the made-up example.
- **Wanted**: A procedure which decides the OCL satisfiability problem.
- **Unfortunately**: in general **undecidable**.

  OCL is as expressive as first-order logic over integers.

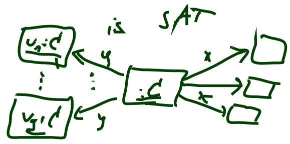
$\exists x, y \bullet x \cdot y > 27$ is SAT, $x = 3$, $y = 9$

$\mathcal{S} = (\emptyset, \{C\}, \{x : C_*, y : C_*\}, \{C \mapsto \{x, y\}\})$

$\boxed{\cancel{\mathit{allInstances}_C \to \mathit{size}() = 1}}$

$\boxed{\cancel{\text{and}}}$ $\mathit{allInstances}_C \to \mathit{exists}(c : C \mid c.x \to \mathit{size}() \cdot c.y \to \mathit{size}() > 27)$

is SAT



Deciding OCL Consistency

- Whether a set of OCL constraints is consistent or not **is in general not as obvious** as in the made-up example.
- **Wanted**: A procedure which decides the OCL satisfiability problem.
- **Unfortunately**: in general **undecidable**.

  OCL is as expressive as first-order logic over integers.

- **And now**? Options: Cabot and Clarisó (2008)
  - Constrain OCL, use a **less rich** fragment of OCL.
  - Revert to **finite domains** — basic types vs. number of objects.

OCL Critique

OCL Critique

- **Concrete Syntax / Features**

  "The syntax of OCL has been criticized – e.g., by the authors of Catalysis [...] – for being hard to read and write.

  - OCL's expressions are stacked in the style of Smalltalk, which makes it hard to see the scope of quantified variables.
  - Navigations are applied to atoms and not sets of atoms, although there is a collect operation that maps a function over a set.
  - Attributes, [...], are partial functions in OCL, and result in expressions with undefined value." Jackson (2002)

OCL Critique

- **Expressive Power**:
  "Pure OCL expressions only compute primitive recursive functions, but not recursive functions in general." Cengarle and Knapp (2001)

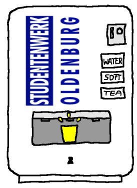
  - **Evolution over Time**: "finally $\mathit{self}.x > 0$"
    Proposals for fixes e.g. Flake and Müller (2003). (Or: sequence diagrams.)

  - **Real-Time**: "Objects respond within 10s"
    Proposals for fixes e.g. Cengarle and Knapp (2002)

  - **Reachability**: "After insert operation, node shall be reachable."
    Fix: add transitive closure.

What Is OCL Good For?

What's It Good For?



- **Most prominent**:

  Formalise **requirements** supposed to be satisfied by all system states.

  Example: "the choice panels of a VM should be consistent"

  $$\text{context } VM \text{ inv : } \{\text{true}, \text{false}\} \text{ -> exists}(b \mid cp \text{ -> forAll}(c \mid c.wen = b))$$

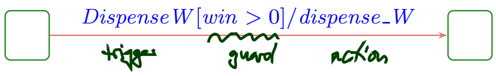
- **Not unknown**:

  Formalise **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states (before/after executing the method).

  Example: "the dispense water method should decrement win"

  $$\begin{aligned} \text{context } DD :: dispense\_W \quad & \text{pre} : win > 0 \\ & \text{post} : win = win@\,\text{pre} - 1 \end{aligned}$$

- **Common with State Machines**: **Guards** in transitions.

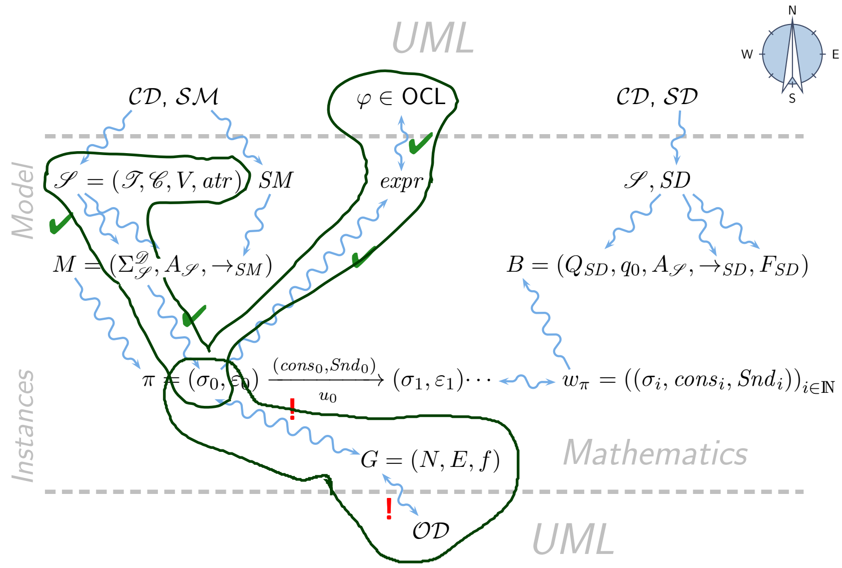


- **Lesser known**: Specify **operation bodies**.

- **Metamodeling**: the UML standard is a MOF-model of UML.
  OCL expressions define well-formedness of UML models (cf. Lecture ~ 21).

Where Are We?

You Are Here.



Object Diagrams

Recall: Graph

Definition. A node-labelled graph is a triple

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

consisting of

- vertexes N,
- edges E,
- node labeling $f : N \to X$, where X is some label domain,

Object Diagrams

Definition. Let $\mathscr{D}$ be a structure of signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$ and $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$ a system state.

Then any node-labelled graph $G = (N, E, f)$ where

- nodes are identities (not necessarily alive), i.e. $N \subset \mathscr{D}(\mathscr{C})$ finite,
- edges correspond to "links" of objects, i.e.

$$E \subseteq N \times \underbrace{\{v : T \in V \mid T \in \{C_{0,1}, C_* \mid C \in \mathscr{C}\}\}}_{=: V_{0,1;*}} \times N,$$

$$\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r),$$

- nodes ~~objects~~ are labelled with attribute valuations, and non-alive identities with "X", i.e.

$$X = \{\mathsf{X}\} \,\dot{\cup}\, (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cancel{\cup \mathscr{D}(\mathscr{C}_*)}))$$

$$\forall\, u \in N \cap \mathrm{dom}(\sigma) : f(u) \subseteq \sigma(u)$$

$$\forall\, u \in N \setminus \mathrm{dom}(\sigma) : f(u) = \{\mathsf{X}\}$$

is called object diagram of σ.

Object Diagram: Examples

- $N \subset \mathscr{D}(\mathscr{C})$ finite • $E \subset N \times V_{0,1;*} \times N$ • $\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$,
- $f : N \to X$ • $X = \{\mathsf{X}\} \,\dot\cup\, (V \nrightarrow (\mathscr{D}(\mathscr{T}) \boxed{\cancel{\cup \mathscr{D}(\mathscr{C}_*)}}))$ • $f(u) \subseteq \sigma(u)$ / $f(u) = \{\mathsf{X}\}$ if $u \notin \mathrm{dom}(\sigma)$

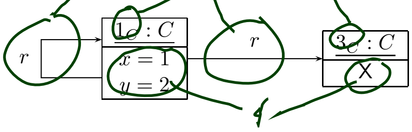
$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{x, y, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

$$\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{1_C, 3_C\}\}\}$$

- $G = (N, E, f)$ with
  - nodes $N = \{1_C, 3_C\}$
  - edges $E = \{(1_C, r, 1_C), (1_C, r, 3_C)\}$,
  - node labelling $f = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2\}, 3_C \mapsto \mathsf{X}\}$ since $3_C \notin$ dom σ

  is an object diagram of σ.
- Yes, and...? G can equivalently (!) be **represented** graphically as follows:

E N

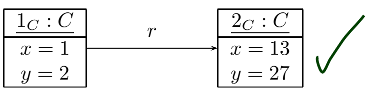


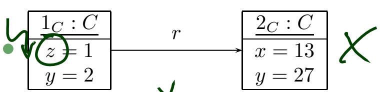
Object Diagram: More Examples?

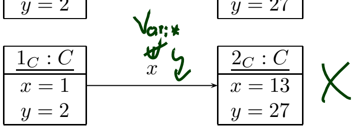
- $N \subset \mathscr{D}(\mathscr{C})$ finite • $E \subset N \times V_{0,1;*} \times N$ • $\forall (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$,
- $f : N \to X$ • $X = \{\mathsf{X}\} \,\dot\cup\, (V \nrightarrow (\mathscr{D}(\mathscr{T}) \boxed{\cancel{\cup \mathscr{D}(\mathscr{C}_*)}}))$ • $f(u) \subseteq \sigma(u)$ / $f(u) = \{\mathsf{X}\}$ if $u \notin \mathrm{dom}(\sigma)$

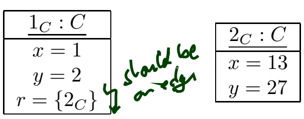
$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{x, y, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

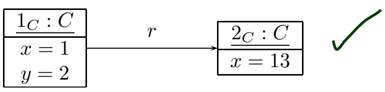
$$\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{2_C\}\}, \quad 2_C \mapsto \{x \mapsto 13, y \mapsto 27, r \mapsto \emptyset\}\},$$

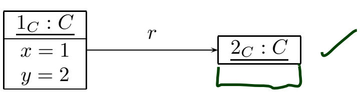
- 

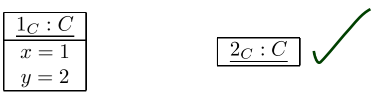
- 

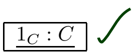
- 

- 

- 

- 

- 

- 

- ✓

Complete vs. Partial Object Diagram

Definition. Let $G = (N, E, f)$ be an object diagram of system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$.
We call G complete wrt. σ if and only if

- G is object complete, i.e.
  - G consists of all alive and "linked" non-alive objects, i.e.

$$N = \mathrm{dom}(\sigma) \cup \{u \mid \exists\, u_1 \in \mathscr{D}(\mathscr{C}), r \in V_{0,1;*} \bullet u \in \sigma(u_1)(r)\}$$

- G is attribute complete, i.e.
  - G comprises all "links" between objects, i.e. if and only if $u_2 \in \sigma(u_1)(r)$ for some $u_1, u_2 \in \mathscr{D}(\mathscr{C})$ and $r \in V$, then $(u_1, r, u_2) \in E$, and
  - each node is labelled with the values of all $\mathscr{T}$-typed attributes, i.e. for each $u \in \mathrm{dom}(\sigma)$,

function restriction

$$f(u) = \sigma(u)|_{V_{\mathscr{T}}}$$

where $V_{\mathscr{T}} := \{v : T \in V \mid T \in \mathscr{T}\}$.

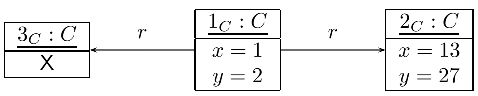
Otherwise we call G partial.

Complete vs. Partial: Examples

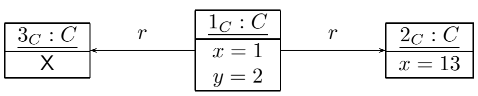
- $N \subset \mathscr{D}(\mathscr{C})$ finite
- $E \subset N \times V_{0,1;*} \times N$
- $\forall\, (u_1, r, u_2) \in E : u_1 \in \mathrm{dom}(\sigma) \wedge u_2 \in \sigma(u_1)(r)$,
- $f : N \to X$
- $X = \{\mathsf{X}\} \,\dot\cup\, (V \nrightarrow (\mathscr{D}(\mathscr{T}) \boxed{\cancel{\cup\, \mathscr{D}(\mathscr{C})}}))$
- $f(u) \subseteq \sigma(u)$ / $f(u) = \{\mathsf{X}\}$ if $u \notin \mathrm{dom}(\sigma)$

$$\mathscr{S} = (\{Int\}, \{C\}, \{x : Int, y : Int, r : C_*\}, \{C \mapsto \{v_1, v_2, r\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

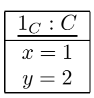
$$\sigma = \{1_C \mapsto \{x \mapsto 1, y \mapsto 2, r \mapsto \{2_C, 3_C\}\}, \quad 2_C \mapsto \{x \mapsto 13, y \mapsto 27, r \mapsto \emptyset\}\},$$

- 

✓ complete

- 

partial (y missing in 2_C)

- 

partial

Complete/Partial is Relative

- Each (consistent) object diagram G represents a set of system states, namely

$$G^{-1} := \{\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}} \mid G \text{ is an object diagram of } \sigma\}$$

- How many?

$G: \boxed{\underline{1_C : C}}$ $\quad |G^{-1}|$ $= 0$ – , $= 1$ / , > 1 | , > 100 | , > 1000000

Infinitely many!

- Each finite system state has **exactly one complete** object diagram.
- A finite system state can have **many partial** object diagrams.

- **Observation**:

  If somebody **tells us** for a given (consistent) object diagram G
  - that it is **meant to be complete**, and
  - if it is not inherently incomplete (e.g. missing attribute values),

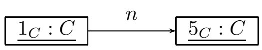
  then it uniquely denotes **the** corresponding system state, denoted by $\sigma(G)$.

  Therefore we can use complete object diagrams **exchangeably** with system states.

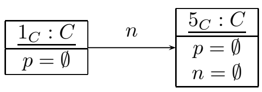
Non-Standard Notation

- $\mathscr{S} = (\{Int\}, \{C\}, \{n, p : C_*\}, \{C \mapsto \{n, p\}\})$.

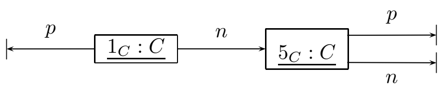
- Instead of



we want to write



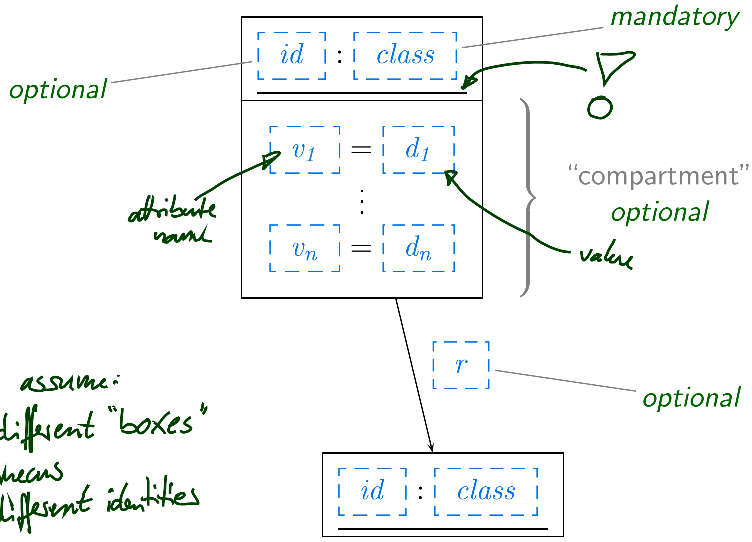
or



to **explicitly** indicate that attribute $p : C_*$ has value $\emptyset$ (also for $p : C_{0,1}$).

UML Object Diagrams

UML Notation for Object Diagrams



we assume:
different "boxes"
means
different identities

e.g. :C → :C

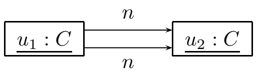
Discussion

We slightly deviate from the standard (for reasons):

- We **allow** to show non-alive objects.
  - Allows us to represent "dangling references",
    i.e. references to objects which are not alive in the current system state.

- We **introduce** a graphical representation of $\emptyset$ values.
  - Easier to distinguish partial and complete object diagrams.

- In the course, $C_{0,1}$ and C_*-typed attributes only have **sets** as values.
  UML also considers multisets, that is, they can have

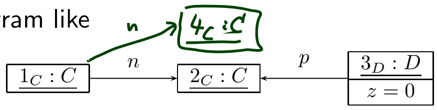
  

  This is **not** an object diagram in the sense of **our definition**
  because of the requirement on the edges E.
  Extension is straightforward but tedious.

The Other Way Round

From Object Diagram to Signature / Structure

- If we **only** have a diagram like



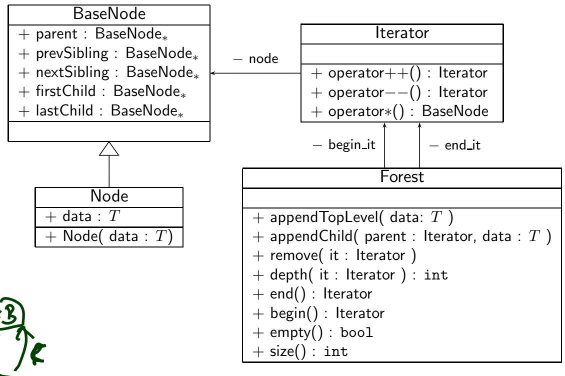
  we typically assume that it is **meant to be**
  an object diagram wrt. **some signature** and **structure**.

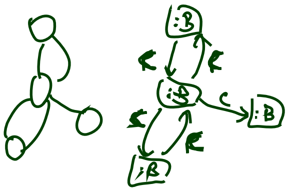
- In the example, we conclude that the author is referring to **some** signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$ with at least
  - $\{C, D\} \subseteq \mathscr{C}$
  - $T \in \mathscr{T}$
  - $\{n : C_*, p : C_{0,1}, z : T\} \subseteq V$
  - $\{z\} \subseteq atr(D)$
  - $\{p, n\} \subseteq atr(C)$

  and a structure $\mathscr{D}$ with
  - $\{1_C, 4_C, 2_C\} \subseteq \mathscr{D}(C)$
  - $3_D \in \mathscr{D}(D)$
  - $0 \in \mathscr{D}(T)$

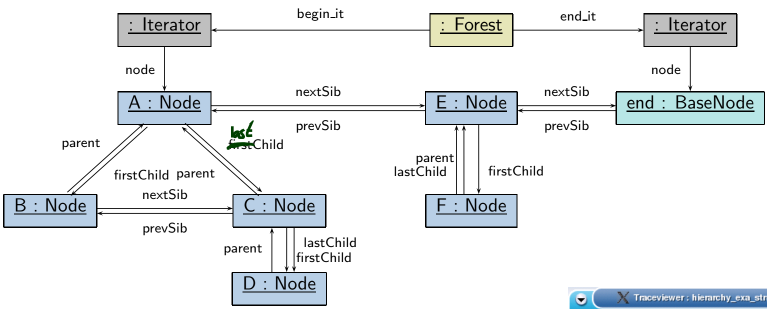
Example: Object Diagrams for Documentation

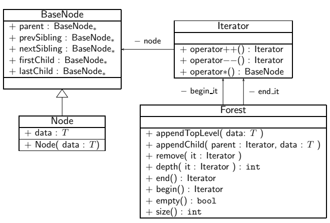
Example: Data Structure (Schumann et al., 2008)

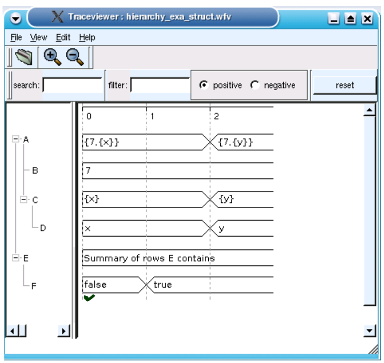




Example: Illustrative Object Diagram (Schumann et al., 2008)







References

References

Cabot, J. and Clarisó, R. (2008). UML-OCL verification in practice. In Chaudron, M. R. V., editor, *MoDELS Workshops*, volume 5421 of *Lecture Notes in Computer Science*. Springer.

Cengarle, M. V. and Knapp, A. (2001). On the expressive power of pure OCL. Technical Report 0101, Institut für Informatik, Ludwig-Maximilians-Universität München.

Cengarle, M. V. and Knapp, A. (2002). Towards OCL/RT. In Eriksson, L.-H. and Lindsay, P. A., editors, *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 390–409. Springer-Verlag.

Flake, S. and Müller, W. (2003). Formal semantics of static and temporal state-oriented OCL constraints. *Software and Systems Modeling*, 2(3):164–186.

Jackson, D. (2002). Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.

Schumann, M., Steinke, J., Deck, A., and Westphal, B. (2008). Traceviewer technical documentation, version 1.0. Technical report, Carl von Ossietzky Universität Oldenburg und OFFIS.