Software Design, Modelling and Analysis in UML

Lecture 02: Semantical Model

2014-10-23

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
  - Basic Object System Signature, Structure, and System State

Why (of all things) UML?

Why (of all things) UML?

- Pre-Note:
  being a **modelling** languages doesn't mean being graphical (or: being a visual formalism [Harel]).

- [Kastens and Büning, 2008] consider as examples:
  - 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**!

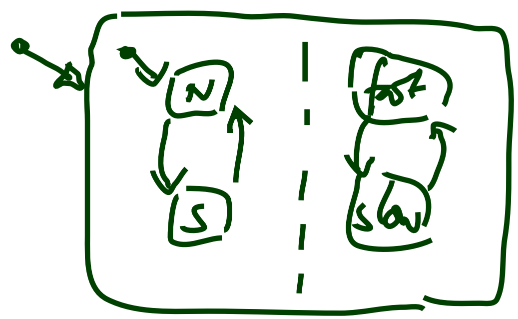
- **Beware**: you may meet people who dislike visual formalisms just for being graphical — maybe because it is easier to "trick" people with a meaningless picture than with a meaningless formula.

  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.

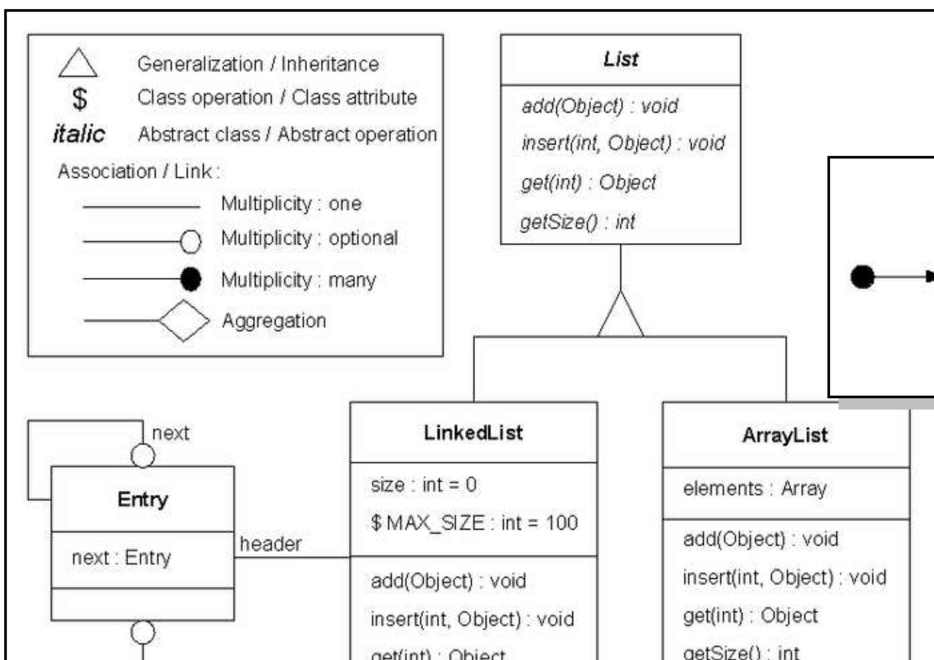
  Languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980**'s: **Statecharts** [Harel, 1987], **StateMate**™ [Harel et al., 1990]

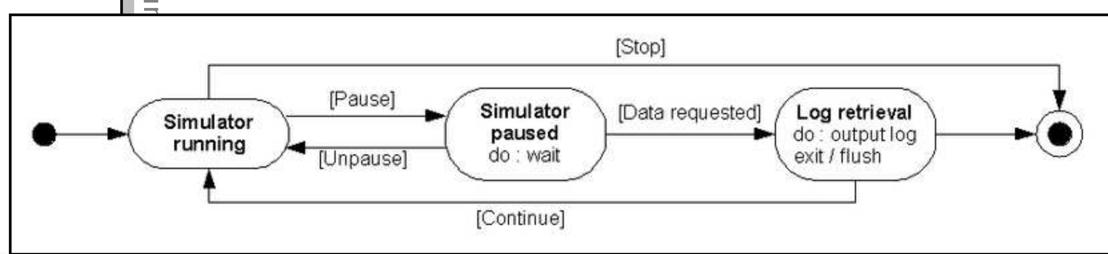


A Brief History of UML

- 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]



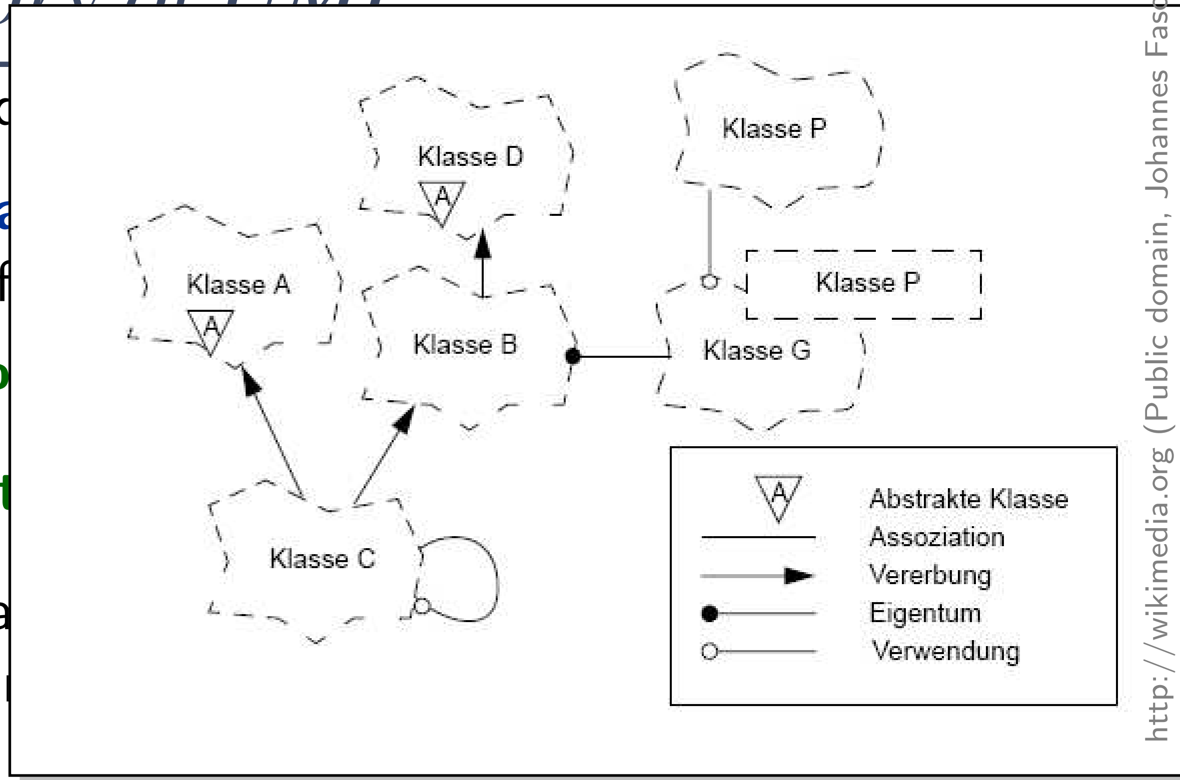


http://wikimedia.org (CC nc-sa 3.0, User:AutumnSnow)

A Brief History of UML

- Boxes/lines and ... or ages.
- 1970's, Softwa...
  — Idea: learn f... mplexity.

  Languages: Flo... grams
- Mid 1980's: St... t al., 1990]
- Early 1990's, a... gramming
  — Inflation of ...



http://wikimedia.org (Public domain, Johannes Fasolt)

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

A Brief History of UML

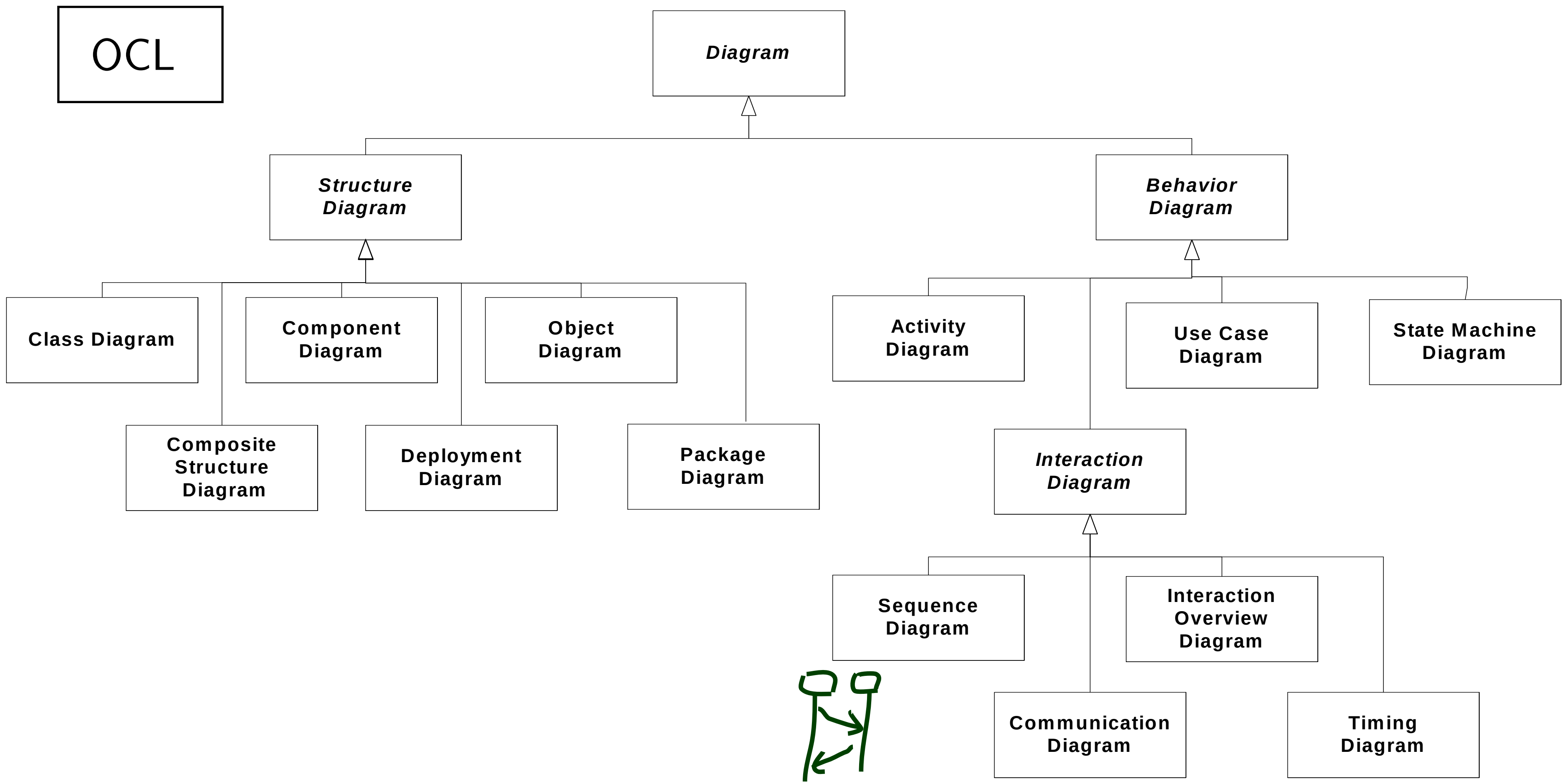
- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™
  — Idea: learn from engineering disciplines to handle growing complexity.

  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]



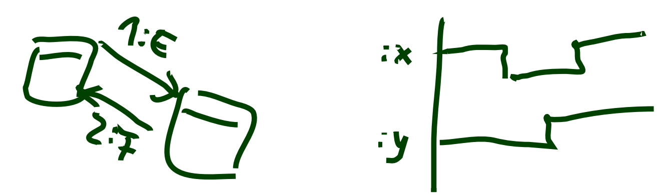


Figure A.5 - The taxonomy of structure and behavior diagram

UML Overview [OMG, 2007b, 684]

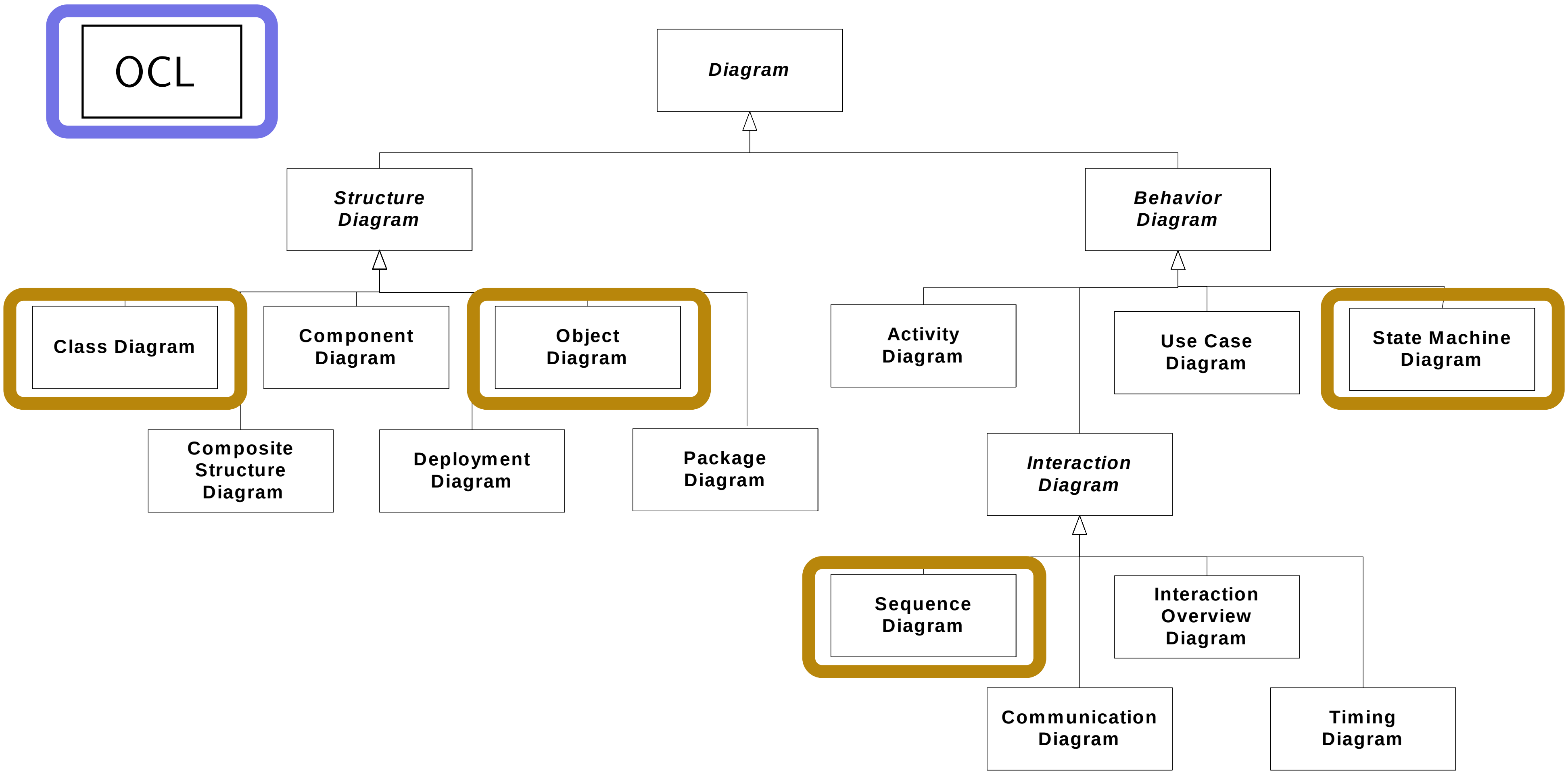


Figure A.5 - The taxonomy of structure and behavior diagram

[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? ...?

Course Map Revisited

The Plan

Recall:

- **Overall aim**:
  a formal language
  for software blueprints.
- **Approach**:

  (i) Common semantical domain.

  (ii) UML fragments as **syntax**.

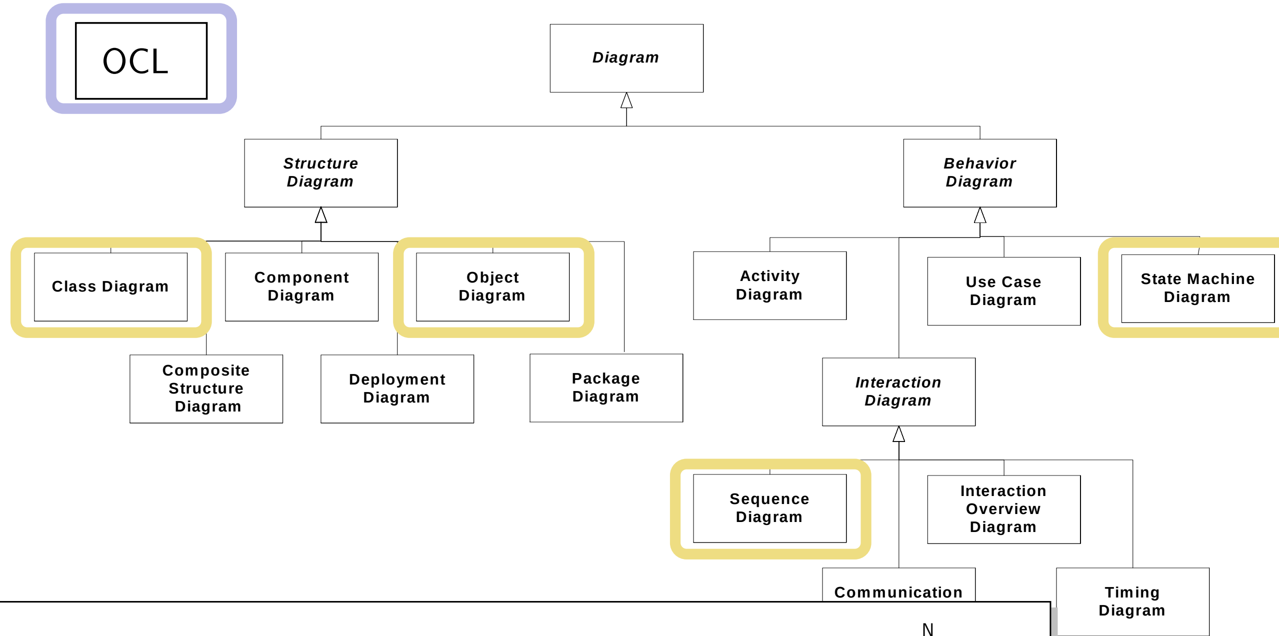
  (iii) Abstract **representation of diagrams**.

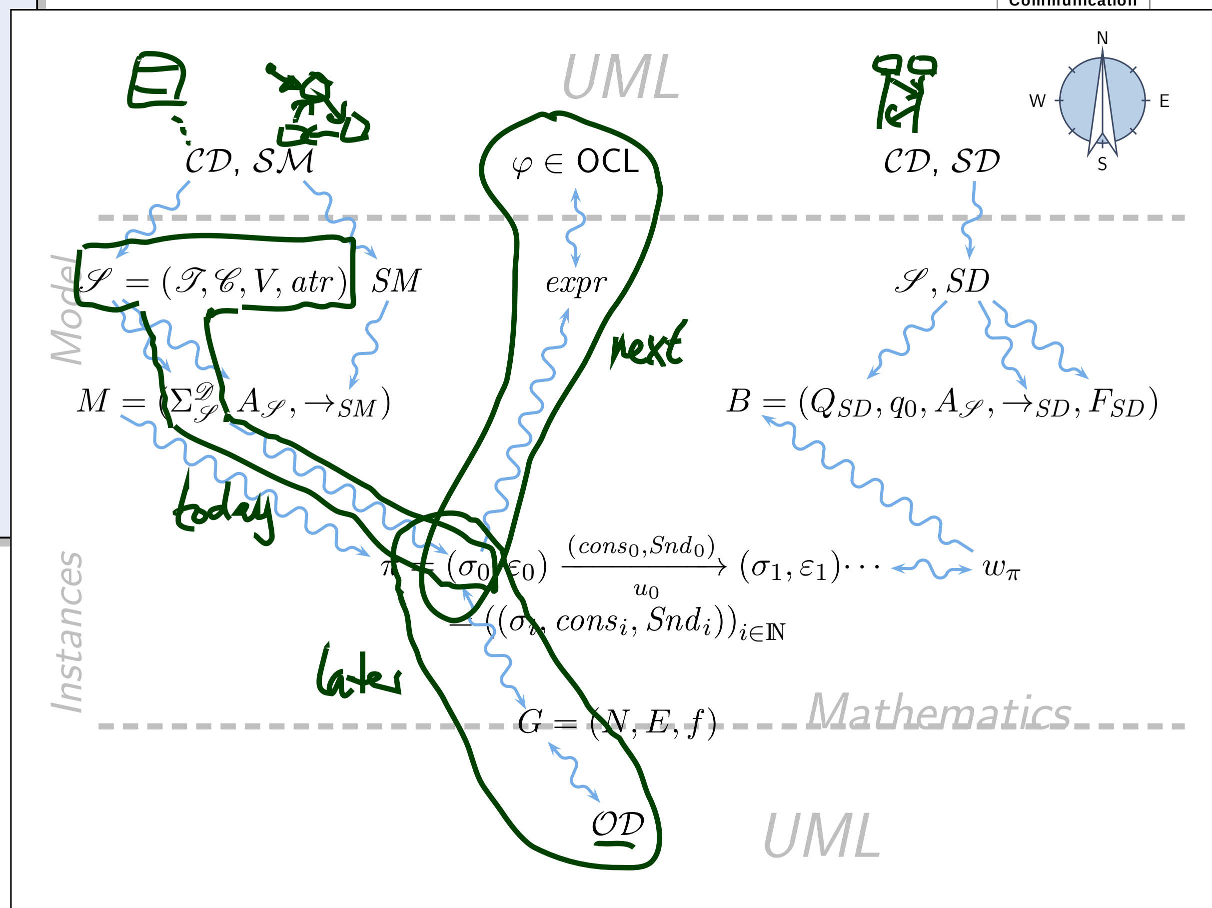
  (iv) **Informal semantics**: UML standard

  (v) **assign meaning to diagrams**.

  (vi) Define, e.g., **consistency**.

OCL
CD
OD
SM
LSC





UML: Semantic Areas

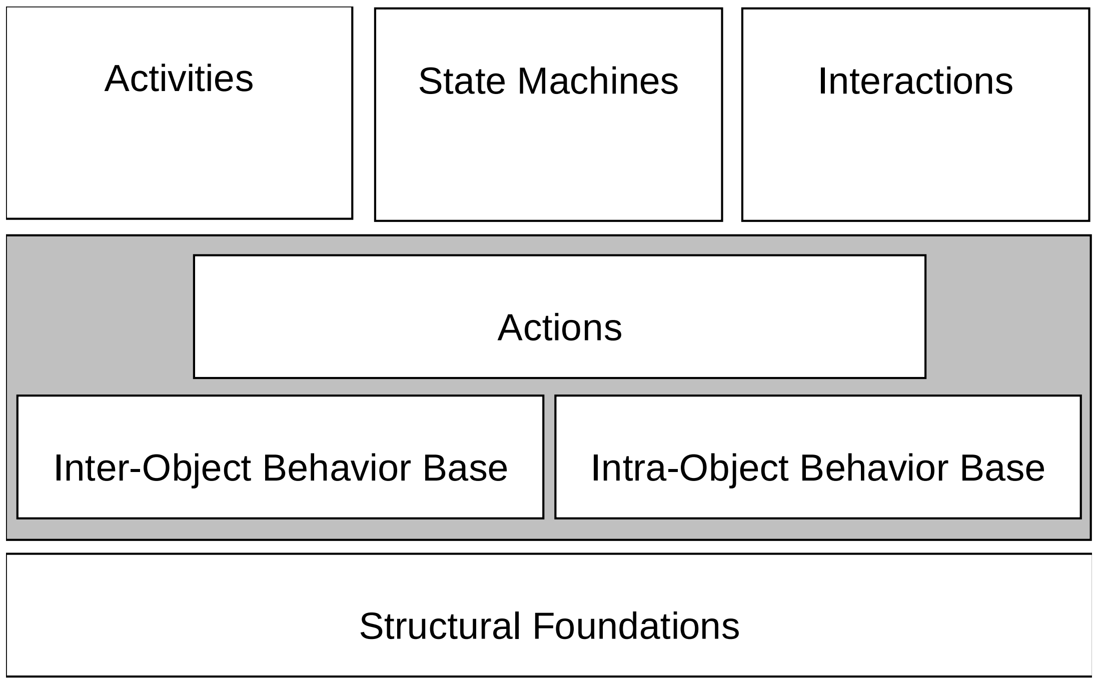


Figure 6.1 - A schematic of the UML semantic areas and their dependencies

[OMG, 2007b, 11]

Common Semantical Domain

Basic Object System Signature

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

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

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

$C_{0,1}$ or: $C_{\triangle}$ or

C_* $C_{\square}$ $\boxed{C}$

where

- $\mathscr{T}$ is a set of (basic) types,
- $\mathscr{C}$ is a finite set of classes,
- V is a finite set of typed attributes, i.e., each $v \in V$ has type
  - $\tau \in \mathscr{T}$ or
  - $C_{0,1}$ or C_*, where $C \in \mathscr{C}$

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

total function

powerset of V

Basic Object System Signature Example

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

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

Example:

$$\mathscr{S}_0 = (\{\overset{\text{types}}{Int}\}, \{C, \overset{\text{classes}}{D}\}, \{\underbrace{\overset{\text{attributes}}{x : Int}}_{\substack{\text{attribute}\\ x \text{ has type}\\ Int\ (\in \mathscr{T})}}, p : C_{0,1}, \underbrace{n : C_*}_{\substack{\text{attribute}\\ n \text{ has type}\\ C_*}}\}, \{\underbrace{\overset{\text{atr}}{C} \mapsto \{p, n\}}_{\substack{atr(C)\\ = \{p,n\}}}, \underbrace{D \overset{\text{maps to}}{\mapsto} \{x\}}_{\substack{atr(D)\\ = \{x\}}}\})$$

Basic Object System Signature Another Example

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

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

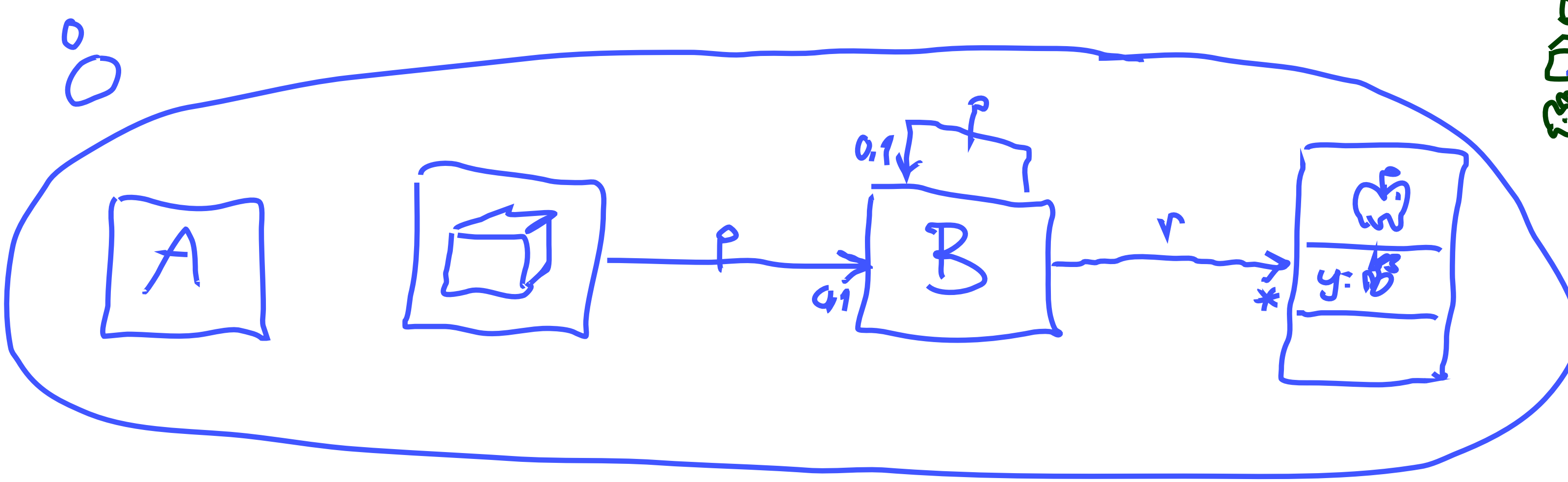
no, not in $\mathscr{T}$ or derived

no, not by choice of derivation rules in this course

empty set

Example:

$\mathscr{S}_1 = (\{🌷\}, \{A, B, ◻, 🍎\}, \{y : 🌷,\ p : B_{0,1},\ q : \cancel{◻}\ ◻_{0,1},\ \cancel{x}\ r : 🍎_*\}, \{A \mapsto \emptyset,\ B \mapsto \{p, r\},\ ◻ \mapsto \{p\},\ 🍎 \mapsto \{y\}\})$



Basic Object System Structure

Definition. A Basic Object System Structure of $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$ is a domain function $\mathscr{D}$ which assigns to each type a domain, i.e.

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

  Note: Object identities only have the "$=$" operation;
  object identities of different classes are disjoint, i.e. $\forall\, C, D \in \mathscr{C} : C \neq D \rightarrow \mathscr{D}(C) \cap \mathscr{D}(D) = \emptyset$.
- C_* **and** $C_{0,1}$ for $C \in \mathscr{C}$ are mapped to $2^{\mathscr{D}(C)}$.

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

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

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

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

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

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

e.g. $\{2_D, 27_D\} \in \mathscr{D}(D_*)$

$\mathscr{D}_2$:

$= \{-127, \dots, 128\}$

$= \{1, 3, 5, 7, \dots\}$

$= \{2, 4, 6, 8, \dots\}$

e.g. $\{2, 4, 6\}$

$$\mathcal{S}_1 = \Big(\{ \ \}, \{A, B, \ , \ \}, \{ y : \ , \ p : B_{0,1}, \ q : \ _{0,1}, \ r : \ _{*} \}, \{ A \mapsto \emptyset, \ B \mapsto \{p, r\}, \ \ \mapsto \{ \ \}, \ \ \mapsto \{y\} \} \Big)$$

$\mathcal{D}(\ ) = \{a, b, c, d\}$ [could also be {rose, tulip, lily, jasmine}]

$\mathcal{D}(A) = \{A, AA, AAA, \ldots\}$

$\mathcal{D}(B) = \{B, BB, BBB, \ldots\}$

$\mathcal{D}(\ ) = \{1, 2, 3, \ldots\}$

$\mathcal{D}(\ ) = \{1, 2, 3, \ldots\}$

$\mathcal{D}(A_*) = 2^{\mathcal{D}(A)}$ e.g. $\{AA\} \in \mathcal{D}(A_*)$

the set of all object identities defined by $\mathscr{D}$

partial function from V to types' domains/values

partial function

Definition. Let $\mathscr{D}$ be a structure of $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$.
A system state of $\mathscr{S}$ wrt. $\mathscr{D}$ is a **type-consistent** mapping

set of attributes in $\mathscr{S}$

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

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

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

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

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

System State Example

Signature, **Structure**:

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

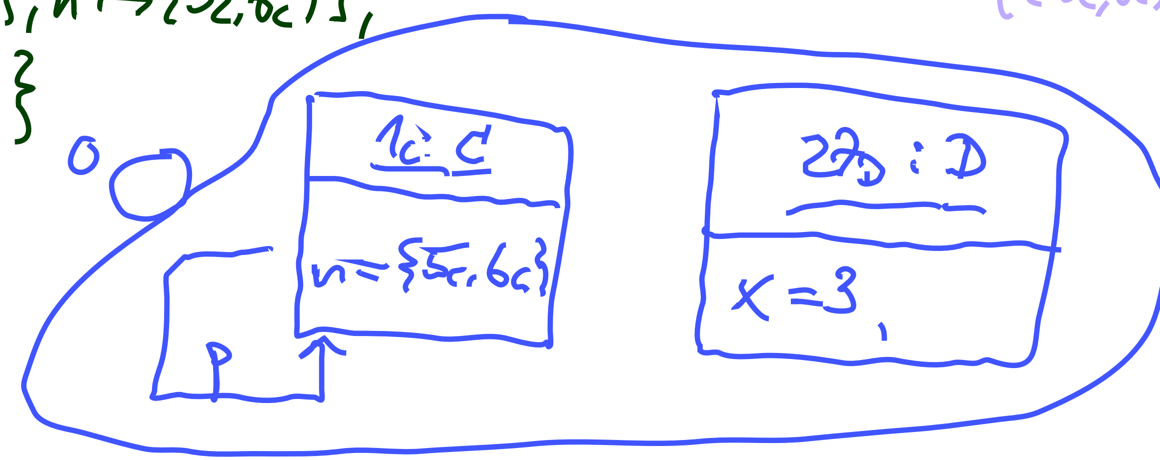
$$\mathscr{D}(Int) = \mathbb{Z}, \quad \mathscr{D}(C) = \{1_C, 2_C, 3_C, ...\}, \quad \mathscr{D}(D) = \{1_D, 2_D, 3_D, ...\}$$

Wanted: $\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$ such that for all $u \in dom(\sigma)$

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

- $\sigma_1 = \emptyset$ ← empty function
- $\sigma_2 = \{1_C \mapsto \{p \mapsto \{1_C\}, n \mapsto \{5_C, 6_C\}\}, 27_D \mapsto \{x \mapsto 3\}\}$

$$\sigma_2(1_C)(v) = \begin{cases} \{1_C\}, & \text{if } v = p \\ \{5_C, 6_C\}, & \text{if } v = n \end{cases}$$



System State Example

Signature, **Structure**:

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

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

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

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

- **Concrete, explicit**:

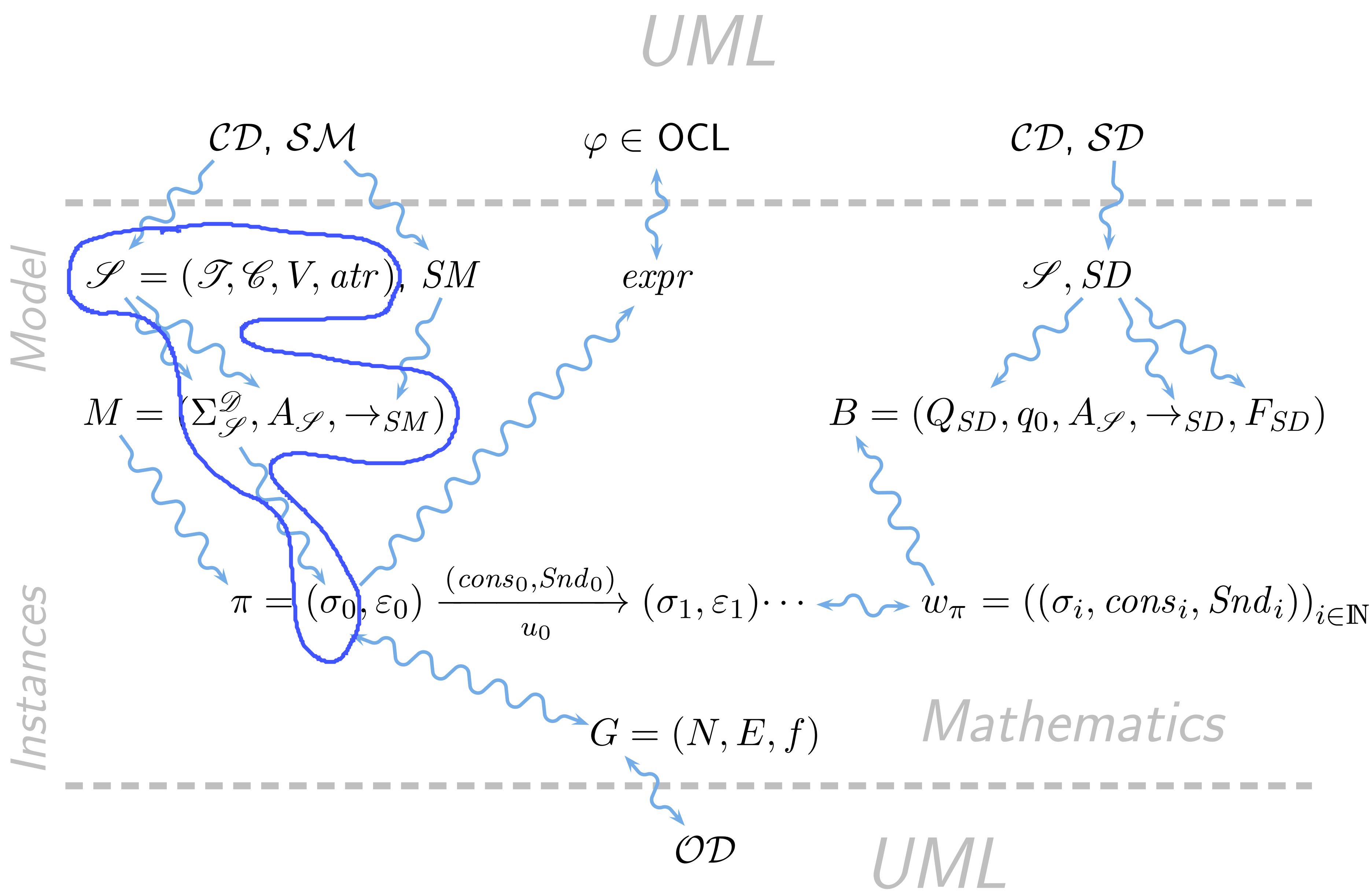
$$\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{x \mapsto 23\}\}.$$

- **Alternative**: **symbolic** system state

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

You Are Here.

Course Map


UML
$\mathcal{CD}$, $\mathcal{SM}$
$\varphi \in$ OCL
$\mathcal{CD}$, $\mathcal{SD}$
Model
$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$, SM
$expr$
$\mathscr{S}$, SD
$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \to_{SM})$
$B = (Q_{SD}, q_0, A_{\mathscr{S}}, \to_{SD}, F_{SD})$
Instances
$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$
$w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$
$G = (N, E, f)$
Mathematics
$\mathcal{OD}$
UML

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 Lorensen, W. (1990). *Object-Oriented Modeling and Design*. Prentice Hall.