Software Design, Modelling and Analysis in UML

Lecture ~~09~~ 10: Constructive Behaviour, State Machines Overview

2011-12-14

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

Albert-Ludwigs-Universität Freiburg, Germany

– 09 – 2011-12-14 – main –

Contents & Goals

Last Lecture:

- Completed discussion of modelling **structure**.

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - ~~Discuss the style of this class diagram.~~
  - What's the difference between reflective and constructive descriptions of behaviour?
  - What's the purpose of a behavioural model?
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
- **Content:**
  - Purposes of Behavioural Models
  - Constructive vs. Reflective
  - UML Core State Machines (first half)

– 09 – 2011-12-14 – Sprelim –

Modelling Behaviour

Stocktaking...

Have: Means to model the **structure** of the system.

- Class diagrams graphically, concisely describe sets of system states.
- OCL expressions logically state constraints/invariants on system states.

Want: Means to model **behaviour** of the system.

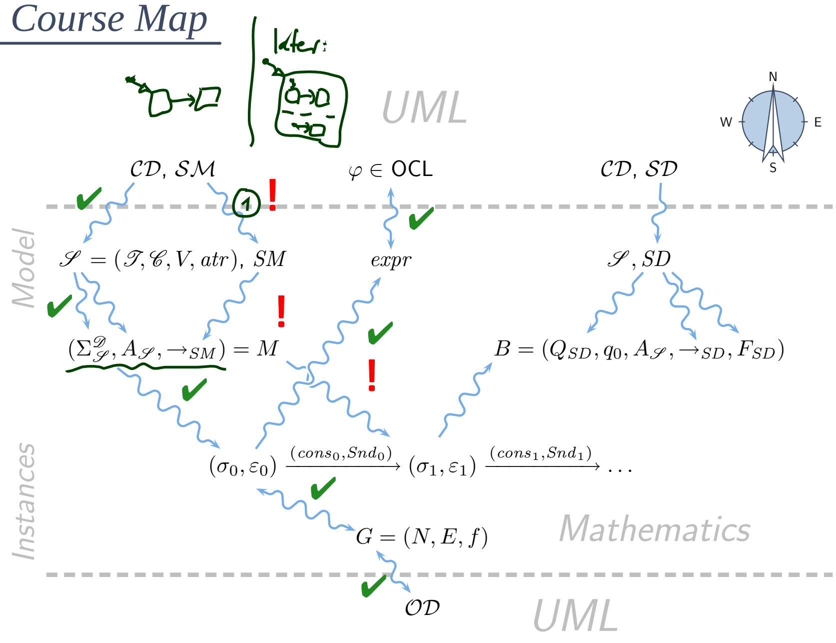
- Means to describe how system states **evolve over time**, that is, to describe sets of **sequences**

$$\sigma_0, \sigma_1, \dots \in \Sigma^\omega$$

  of system states.

not real-time, discrete time

Course Map



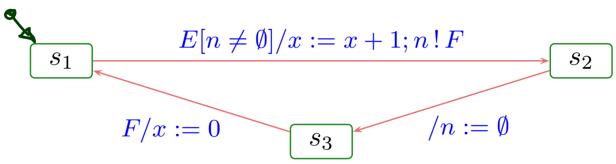
Constructive UML

UML provides two visual formalisms for constructive description of behaviours:

- **Activity Diagrams**
- **State-Machine Diagrams**

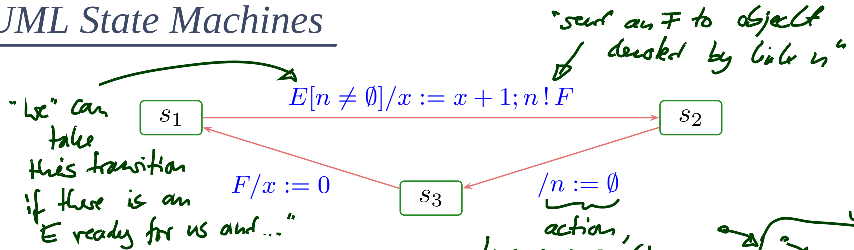
We (exemplary) focus on State-Machines because

- somehow "practice proven" (in different flavours),
- prevalent in embedded systems community,
- indicated useful by [Dobing and Parsons, 2006] survey, and
- Activity Diagram's intuition changed (between UML 1.x and 2.x) from transition-system-like to petri-net-like...
- Example state machine:



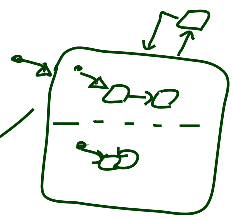
UML State Machines: Overview

UML State Machines



Brief History:

- Rooted in **Moore/Mealy machines**, Transition Systems
- [Harel, 1987]: **Statecharts** as a concise notation, introduces in particular hierarchical states.



- Manifest in tool **Statemate** [Harel et al., 1990] (simulation, code-generation); nowadays also in **Matlab/Simulink**, etc.
- From UML 1.x on: **State Machines** (not the official name, but understood: UML-Statecharts)
- Late 1990's: tool **Rhapsody** with code-generation for state machines.

Note: there is a common core, but each dialect interprets some constructs subtly different [Crane and Dingel, 2007]. *(Would be too easy otherwise. . .)*

Roadmap: Chronologically

(i) What do we (have to) cover?
UML State Machine Diagrams **Syntax**.

(ii) Def.: Signature with **signals**.

(iii) Def.: **Core state machine**.

(iv) Map UML State Machine Diagrams to core state machines.

Semantics:
The Basic Causality Model

(v) Def.: **Ether** (aka. event pool)

(vi) Def.: **System configuration**.

(vii) Def.: **Event**.

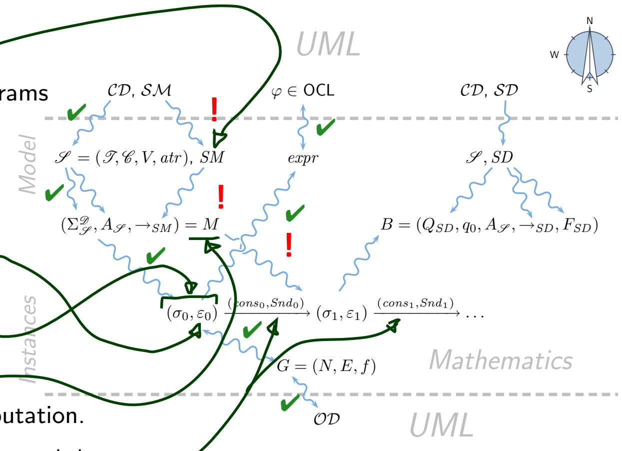
(viii) Def.: **Transformer**.

(ix) Def.: **Transition system**, computation.

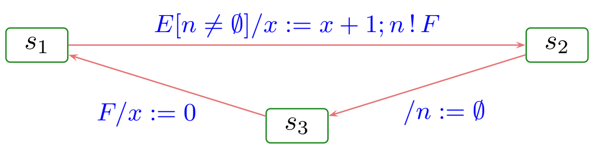
(x) Transition relation induced by core state machine.

(xi) Def.: **step**, **run-to-completion step**.

(xii) Later: Hierarchical state machines.



UML State Machines



Brief History:

- Rooted in **Moore/Mealy machines**, Transition Systems
- [Harel, 1987]: **Statecharts** as a concise notation, introduces in particular hierarchical states.
- Manifest in tool **Statemate** [Harel et al., 1990] (simulation, code-generation); nowadays also in **Matlab/Simulink**, etc.
- From UML 1.x on: **State Machines** (not the official name, but understood: UML-Statecharts)
- Late 1990's: tool **Rhapsody** with code-generation for state machines.

Note: there is a common core, but each dialect interprets some constructs subtly different [Crane and Dingel, 2007]. *(Would be too easy otherwise. . .)*

Roadmap: Chronologically

(i) What do we (have to) cover?
UML State Machine Diagrams **Syntax**.

(ii) Def.: Signature with **signals**.

(iii) Def.: **Core state machine**.

(iv) Map UML State Machine Diagrams to core state machines.

Semantics:
The Basic Causality Model

(v) Def.: **Ether** (aka. event pool)

(vi) Def.: **System configuration**.

(vii) Def.: **Event**.

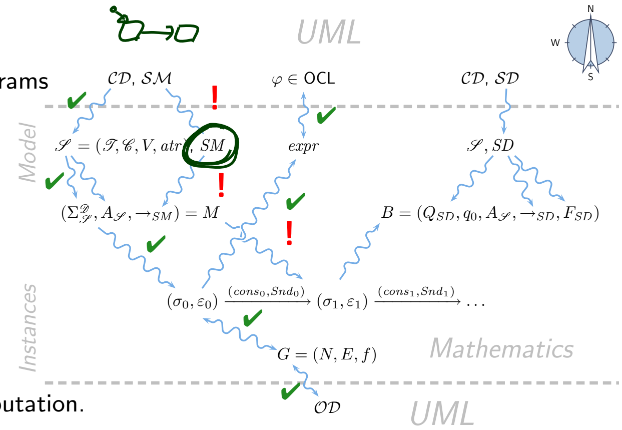
(viii) Def.: **Transformer**.

(ix) Def.: **Transition system**, computation.

(x) Transition relation induced by core state machine.

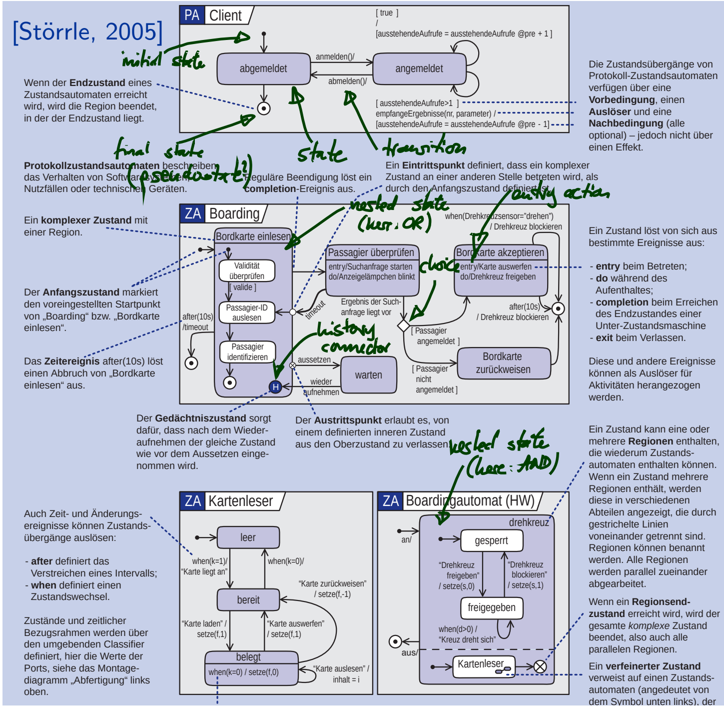
(xi) Def.: **step**, **run-to-completion step**.

(xii) Later: Hierarchical state machines.

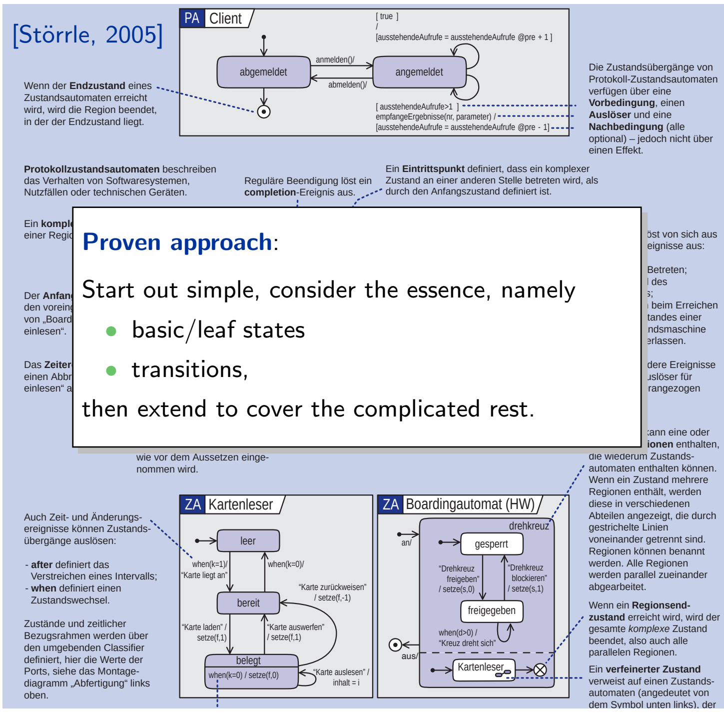


UML State Machines: Syntax

UML State-Machines: What do we have to cover?



UML State-Machines: What do we have to cover?



Proven approach:

Start out simple, consider the essence, namely

- basic/leaf states
- transitions,

then extend to cover the complicated rest.

Signature With Signals

Definition. A tuple

$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E}), \qquad \mathscr{E} \text{ a set of signals,}$$

is called signature (with signals) if and only if

$$(\mathscr{T}, \mathscr{C} \cup \mathscr{E}, V, atr)$$

is a signature (as before).

Note: Thus conceptually, **a signal is a class** and can have attributes of plain type and associations.

Core State Machine

Definition.
A core state machine over signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ is a tuple

$$M = (S, s_0, \to)$$

where

- S is a non-empty, finite set of **(basic) states**,
- $s_0 \in S$ is an **initial state**,
- and

$$\to \subseteq S \times \underbrace{(\mathscr{E} \,\dot{\cup}\, \{\_\})}_{\text{trigger}} \times \underbrace{Expr_{\mathscr{S}}}_{\text{guard}} \times \underbrace{Act_{\mathscr{S}}}_{\text{action}} \times S$$

  is a labelled transition relation.

We assume a set $Expr_{\mathscr{S}}$ of boolean expressions over $\mathscr{S}$ (for instance OCL, may be something else) and a set $Act_{\mathscr{S}}$ of **actions**.

From UML to Core State Machines: By Example

UML state machine diagram $\mathcal{SM}$:

$annot ::= [\langle event \rangle [\text{'.'} \langle event \rangle]^*] [\text{'['} \langle guard \rangle \text{']'}] [\text{'/'} \langle action \rangle]$

(trigger) (guard) (action)

with

- $event \in \mathscr{E}$, (default: _ if no trigger)
- $guard \in Expr_{\mathscr{S}}$ (default: *true*, assumed to be in $Expr_{\mathscr{S}}$)
- $action \in Act_{\mathscr{S}}$ (default: skip, assumed to be in $Act_{\mathscr{S}}$)

maps to

$$M(\mathcal{SM}) = (\underbrace{\{s_1, s_2\}}_{S}, \underbrace{s_1}_{s_0}, \underbrace{(s_1, event, guard, action, s_2)}_{\rightarrow})$$

(initial state)

Annotations and Defaults in the Standard

Reconsider the syntax of transition annotations:

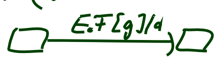
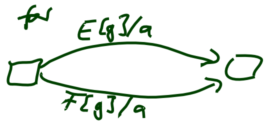
$annot ::= [[\langle event \rangle [\text{'.'} \langle event \rangle]^*] [\text{'['} \langle guard \rangle \text{']'}] [\text{'/'} [\langle action \rangle]]]$

and let's play a bit with the defaults:

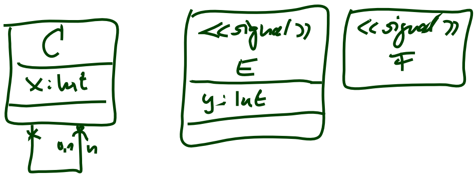
(empty annot:)	$\rightsquigarrow$	$(s_1, \_, true, skip, s_2)$
/	$\rightsquigarrow$	$(s_1, \_, true, skip, s_2)$
E /	$\rightsquigarrow$	$(s_1, E, true, skip, s_2)$
/ act	$\rightsquigarrow$	$(s_1, \_, true, act, s_2)$
E / act	$\rightsquigarrow$	$(s_1, E, true, act, s_2)$
E[e]/act	$\rightsquigarrow$	(s_1, E, e, act, s_2)

In the standard, the syntax is even more elaborate: (we don't discuss those)

- $E(v)$ — when consuming E in object u, attribute v of u is assigned the corresponding attribute of $E \in \mathscr{E}$
- $E(v : \tau)$ — similar, but v is a local variable, scope is the transition

-  we view as an abbrev. for 

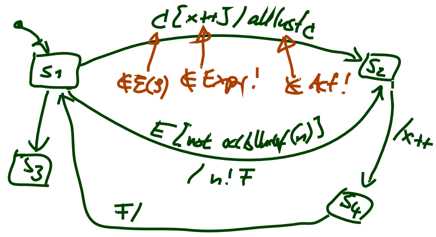
CD:



$Expr_{\mathcal{S}}$: OCL over $\mathcal{S}$

$Act_{\mathcal{S}}$: { skip, x++, n!F }

UML

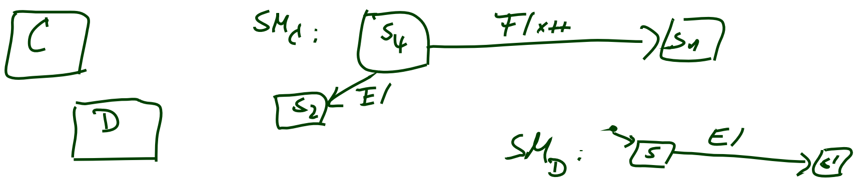


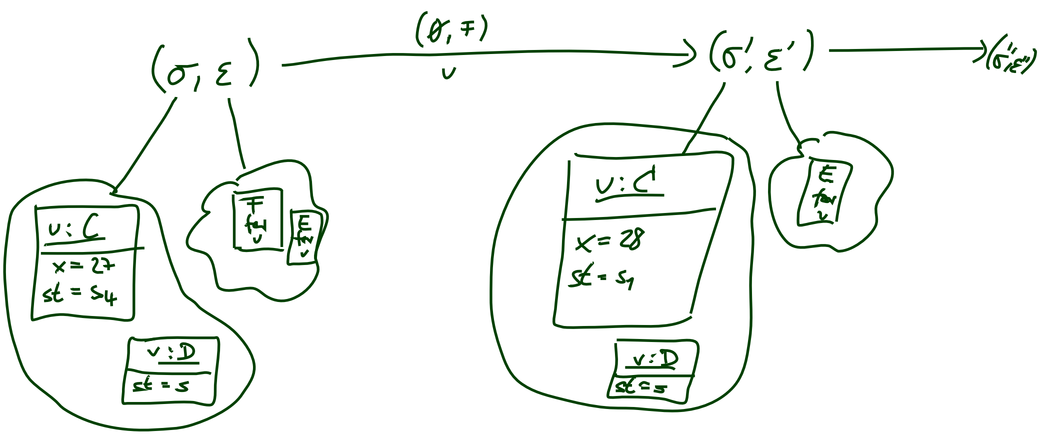
$\mathcal{S} = \big(\{Int\}, \{\langle C,\emptyset,0,0\rangle, \langle E,\{signal\},0,0\rangle, \langle F,\{signal\},0,0\rangle\}, \{x:Int, y:Int, n:C_{0,1}\}, \{C \mapsto \{x,n\}, E \mapsto \{y\}\}\big)$

$\mathcal{E}(\mathcal{S}) = \{E, F\}$

$M = (\{s_1, s_2, s_3, s_4\},$
$s_1,$
$\{(s_1, -, true, skip, s_3),$
$(s_1, E, not\ oclIsUndef(n), n!F, s_2),$
$\dots\})$

MATH





State-Machines belong to Classes

- In the following, we assume that a UML models consists of a set $\mathscr{CD}$ of class diagrams and a set $\mathscr{SM}$ of **state chart diagrams** (each comprising one **state machines** $\mathcal{SM}$).
- Furthermore, we assume each that each state machine $\mathcal{SM} \in \mathscr{SM}$ is **associated with a class** $C_{\mathcal{SM}} \in \mathscr{C}(\mathscr{S})$.
- For simplicity, we even assume a bijection, i.e. we assume that each class $C \in \mathscr{C}(\mathscr{S})$ has a state machine $\mathcal{SM}_C$ and that its class $C_{\mathcal{SM}_C}$ is C.

  If not explicitly given, then this one:

$$\mathcal{SM}_0 := (\{s_0\}, s_0, (s_0, \_, \mathit{true}, \texttt{skip}, s_0)).$$

  We'll see later that, semantically, this choice does no harm.

- **Intuition 1**: $\mathcal{SM}_C$ describes the behaviour of **the instances** of class C.
  Intuition 2: Each instance of class C executes $\mathcal{SM}_C$.

Note: we don't consider **multiple state machines** per class.
Because later (when we have AND-states) we'll see that this case can be viewed as a single state machine with as many AND-states.

References

References

[Crane and Dingel, 2007] Crane, M. L. and Dingel, J. (2007). UML vs. classical vs. rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6(4):415–435.

[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, 1997] Harel, D. (1997). Some thoughts on statecharts, 13 years later. In Grumberg, O., editor, *CAV*, volume 1254 of *LNCS*, pages 226–231. Springer-Verlag.

[Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.

[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.

[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.