

# *Software Design, Modelling and Analysis in UML*

## *Lecture 11: Core State Machines I*

2013-12-04

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

Albert-Ludwigs-Universität Freiburg, Germany

- 11 - 2013-12-04 - main -

## Contents & Goals

### Last Lecture:

- Core State Machines
- UML State Machine syntax
- State machines belong to classes.

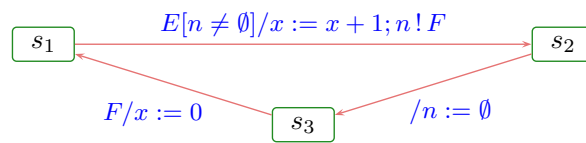
### This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.
- **Content:**
  - UML Core State Machines (first half)
  - Ether, System Configuration, Transformer
  - Run-to-completion Step
  - Putting It All Together

- 11 - 2013-12-04 - Prelim -

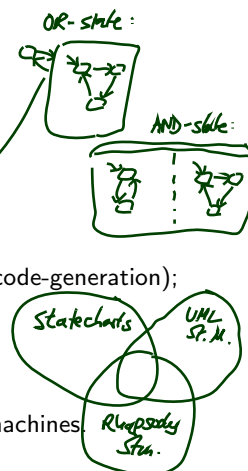
# UML State Machines

## UML State Machines



### Brief History:

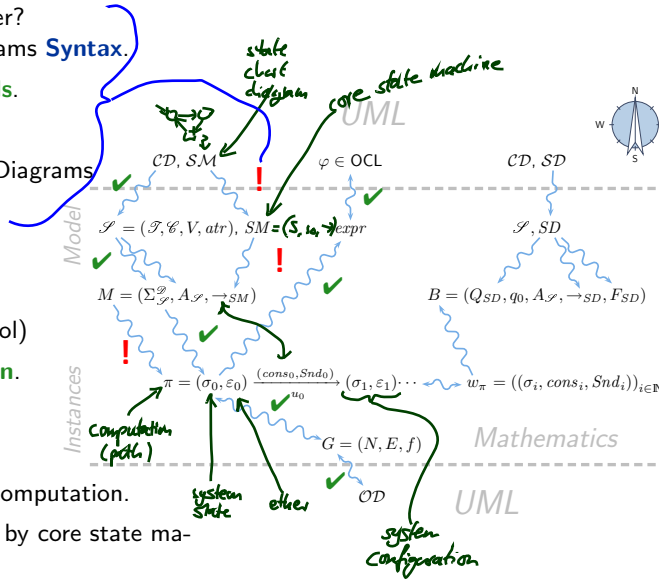
- Rooted in **Moore/Mealy machines**, Transition Systems
- [Harel, 1987]: **Statecharts** as a concise notation, introduces in particular hierarchical states.
- Manifest in tool **Statemate** [Harel et al., 1990] (simulation, code-generation); nowadays also in **Matlab/Simulink**, etc.
- From UML 1.x on: **State Machines** (in *State Chart Diagram*) (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?

*initial state*

*state*

*final state (optional)*

*transition*

*guard*

*action*

*trigger*

*basic state (no more states nested inside)*

*entry action*

*do action*

*choice*

*nested state, here OR-state*

*history connector*

*nested state (AND-state with two components)*

[Störle, 2005]

Die Zustandsübergänge von Protokoll-Zustandsautomaten verfügen über eine **Vorbereitung**, einen **Auslöser** und eine **Nachbedingung** (alle optional) – jedoch nicht über einen Effekt.

Ein **Eintrittspunkt** definiert, dass ein komplexer Zustand an einer anderen Stelle betrachtet wird, als durch den Anfangszustand definiert ist.

Reguläre Beendigung löst ein **completion**-Ereignis aus.

Ein **Region** ist ein Teil eines Zustands, der sich aus bestimmten Ereignissen auslöst.

Ein **Region** löst sich von sich aus, wenn bestimmte Ereignisse eintreffen.

**entry** beim Betreten; **do** während des Aufenthalts.

**completion** beim Erreichen des Endzustandes einer Unter-Zustandsmaschine.

**exit** beim Verlassen.

Diese und andere Ereignisse können als Auslöser für Aktivitäten herangezogen werden.

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt. Durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.

Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte komplexe Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

Auch Zeit- und Änderungsereignisse können Zustandsübergänge auslösen:

- **after** definiert das Verstreichen eines Intervalls;
- **when** definiert einen Zustandswechsel.

Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montage- diagramm „Abfertigung“ links oben.

Der **Gedächtniszustand** sorgt dafür, dass nach dem Wieder-aufnehmen der gleiche Zustand wie vor dem Aussetzen eingenommen wird.

Der **Austrittspunkt** erlaubt es, von einem definierten inneren Zustand aus den Oberzustand zu verlassen.

Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montage- diagramm „Abfertigung“ links oben.

Ein **Region** löst sich von sich aus, wenn bestimmte Ereignisse eintreffen.

Betreten; **do** während des Aufenthalts.

Diese Ereignisse sind Auslöser für Rangezogen

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt. Durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.

Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte komplexe Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

7/71

- 11 - 2013-12-04 - Sstmsyn -

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

[Störle, 2005]

Die Zustandsübergänge von Protokoll-Zustandsautomaten verfügen über eine **Vorbereitung**, einen **Auslöser** und eine **Nachbedingung** (alle optional) – jedoch nicht über einen Effekt.

Ein **Eintrittspunkt** definiert, dass ein komplexer Zustand an einer anderen Stelle betrachtet wird, als durch den Anfangszustand definiert ist.

Reguläre Beendigung löst ein **completion**-Ereignis aus.

Ein **Region** ist ein Teil eines Zustands, der sich aus bestimmten Ereignissen auslöst.

Ein **Region** löst sich von sich aus, wenn bestimmte Ereignisse eintreffen.

**entry** beim Betreten; **do** während des Aufenthalts.

**completion** beim Erreichen des Endzustandes einer Unter-Zustandsmaschine.

**exit** beim Verlassen.

Diese Ereignisse sind Auslöser für Rangezogen

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt. Durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.

Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte komplexe Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

Auch Zeit- und Änderungsereignisse können Zustandsübergänge auslösen:

- **after** definiert das Verstreichen eines Intervalls;
- **when** definiert einen Zustandswechsel.

Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montage- diagramm „Abfertigung“ links oben.

Der **Gedächtniszustand** sorgt dafür, dass nach dem Wieder-aufnehmen der gleiche Zustand wie vor dem Aussetzen eingenommen wird.

Der **Austrittspunkt** erlaubt es, von einem definierten inneren Zustand aus den Oberzustand zu verlassen.

Zustände und zeitlicher Bezugsrahmen werden über den umgebenden Classifier definiert, hier die Werte der Ports, siehe das Montage- diagramm „Abfertigung“ links oben.

Ein **Region** löst sich von sich aus, wenn bestimmte Ereignisse eintreffen.

Betreten; **do** während des Aufenthalts.

Diese Ereignisse sind Auslöser für Rangezogen

Ein Zustand kann eine oder mehrere **Regionen** enthalten, die wiederum Zustandsautomaten enthalten können. Wenn ein Zustand mehrere Regionen enthält, werden diese in verschiedenen Abteilen angezeigt. Durch gestrichelte Linien voneinander getrennt sind. Regionen können benannt werden. Alle Regionen werden parallel zueinander abgearbeitet.

Wenn ein **Regionsendzustand** erreicht wird, wird der gesamte komplexe Zustand beendet, also auch alle parallelen Regionen.

Ein **verfeinerter Zustand** verweist auf einen Zustandsautomaten (angedeutet von dem Symbol unten links), der

**Proven approach:**

Start out simple, consider the essence, namely

- basic/leaf states
- transitions,

then extend to cover the complicated rest.

7/71

- 11 - 2013-12-04 - Sstmsyn -

## Signature With Signals

**Definition.** A tuple

$$\mathcal{S} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E}), \mathcal{C} \supseteq \mathcal{E} \text{ a set of signals,}$$

is called **signature (with signals)** if and only if

$$(\mathcal{I}, \mathcal{C} \setminus \mathcal{E}, V, atr)$$

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  $\mathcal{S} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E})$  is a tuple

$$\mathcal{S}M = (S, s_0, \rightarrow)$$

where

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

$$\rightarrow \subseteq S \times (\mathcal{E} \dot{\cup} \{-\}) \times \underbrace{Expr_{\mathcal{S}}}_{\text{guard}} \times \underbrace{Act_{\mathcal{S}}}_{\text{action}} \times S$$

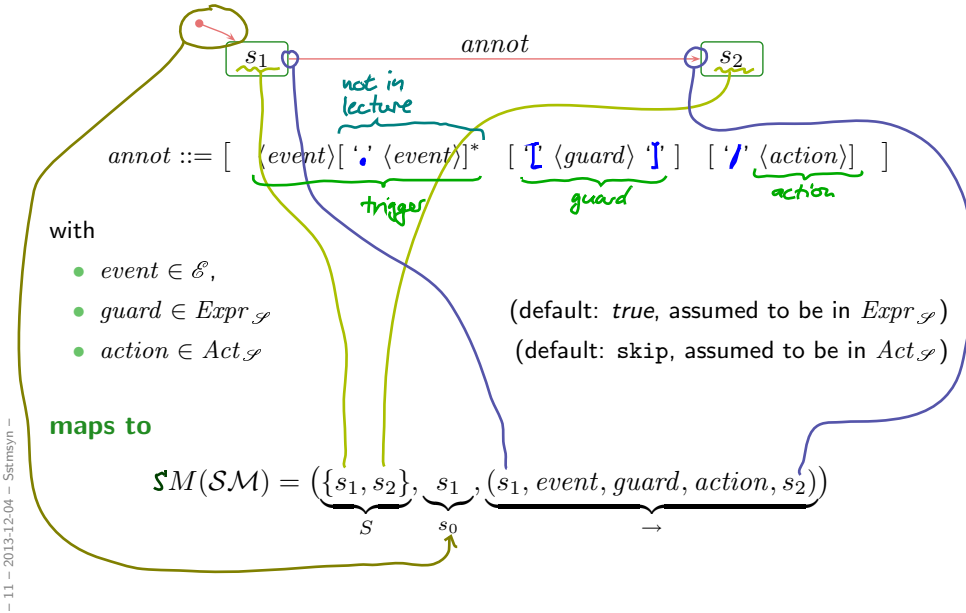
Handwritten annotations: "source state" points to the first  $S$ ; "set of signals" points to  $\mathcal{E}$ ; "disjoint union" points to  $\mathcal{E} \dot{\cup} \{-\}$ ; "trigger" points to  $\mathcal{E}$ ; "guard" points to  $Expr_{\mathcal{S}}$ ; "action" points to  $Act_{\mathcal{S}}$ ; "destination state" points to the final  $S$ .

is a labelled transition relation.

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

# From UML to Core State Machines: By Example

UML state machine diagram  $SM$ :



## Annotations and Defaults in the Standard

**Reconsider** the syntax of transition annotations:

$annot ::= [ \langle event \rangle [ \cdot \langle event \rangle ]^* [ [ \langle guard \rangle ] ] [ [ \langle action \rangle ] ] ]$

and let's play a bit with the defaults:

the empty annotation	$\rightarrow$	$\rightsquigarrow$	$-, true, skip$
	$/$	$\rightsquigarrow$	$-, \text{---}, skip$
$E \in \mathcal{E}$	$E /$	$\rightsquigarrow$	$E, true, skip$
$act \in Act_{\mathcal{F}}$	$E / act$	$\rightsquigarrow$	$-, true, act$
$expr \in Expr_{\mathcal{F}}$	$E / [expr]$	$\rightsquigarrow$	$E, true, act$
	$[expr]$	$\rightsquigarrow$	$-, expr, skip$

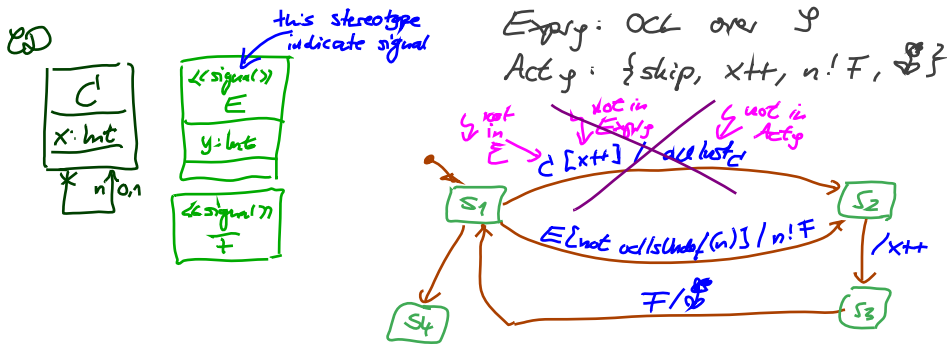
*trigger guard action* (handwritten labels pointing to the trigger, guard, and action parts of the syntax)

**In the standard**, the syntax is even more elaborate:

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

*special keyword to access event attributes* (handwritten note pointing to  $E(x)$ )

*in Rhapsody:*  
 $E(x) \dots$   
 $\rightsquigarrow E/x := param \rightarrow x$



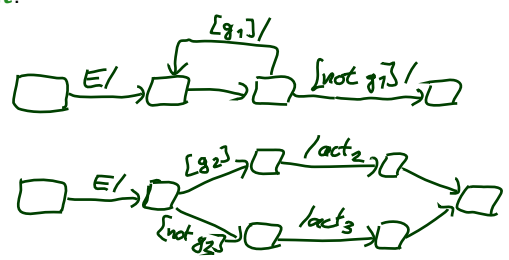
# UML

$S = (\{ int \}, \{ C, E, F \}, \{ x: int, y: int, n: C_{0,1} \}, \{ C \mapsto x, n \}, \{ E \mapsto S_2, F \mapsto \emptyset \}, \{ E, F \})$   
 UML  
 MATHE

$SM = (\{ S_1, S_2, S_3, S_4 \}, S_1, \{ (S_1, E, not ok / skip / n!F, S_2), (S_2, -, true, x++, S_3), (S_1, -, true, skip, S_4), (S_3, F, true, \emptyset, S_1) \})$

## What is that useful for?

- No Event:



- No annotation:

see above

## State-Machines belong to Classes

- In the following, we assume that a UML models consists of a set  $\mathcal{C}\mathcal{D}$  of class diagrams and a set  $\mathcal{SM}$  of **state chart diagrams** (each comprising one **state machines**  $SM$ ).
- Furthermore, we assume that **each state machine**  $SM \in \mathcal{SM}$  is **associated** with a **class**  $C_{SM} \in \mathcal{C}(\mathcal{S})$ .
- For simplicity, we even assume a bijection, i.e. we assume that each class  $C \in \mathcal{C}(\mathcal{S})$  has a state machine  $SM_C$  and that its class  $C_{SM_C}$  is  $C$ .  
If not explicitly given, then this one:

$$SM_0 := (\{s_0\}, s_0, \emptyset).$$

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

- **Intuition 1:**  $SM_C$  describes the behaviour of **the instances** of class  $C$ .
- **Intuition 2:** Each instance of  $C$  executes  $SM_C$  with own "program counter".

**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.)

13/71

## 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.
- [Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.
- [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.
- [Störrle, 2005] Störrle, H. (2005). *UML 2 für Studenten*. Pearson Studium.