

## Software Design, Modelling and Analysis in UML

### Lecture 10: Constructive Behaviour, State Machines Overview

2012-11-28

Prof. Dr. Andreas Podolski, Dr. Bernd Westphal  
Albert-Ludwigs-Universität Freiburg, Germany

## 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)

## 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**  
 $\sigma_0, \sigma_1, \dots \in \Sigma^*$   
*not including, just reaching states*
- of system states.

### What Can Be Purposes of Behavioural Models?

(We will discuss this in more detail in Lecture 22)

**Example:** Pre-Image  
(the UML model is supposed to be the blue-print for a software system).

A description of behaviour could serve the following purposes:

- Require** Behaviour.
  - "This sequence of inserting money and requesting and getting water must be possible."
  - (Otherwise the software for the vending machine is completely broken.)
- Allow** Behaviour.
  - "After inserting money and choosing a drink the drink is dispensed (if in stock)." (if the implementation insists on taking the money first, that's a fair choice.)
- Forbid** Behaviour.
  - "This sequence of getting both, a water and all money back, must not be possible." (Otherwise the software is broken.)

### What Can Be Purposes of Behavioural Models?

(We will discuss this in more detail in Lecture 22)

**Example:** Pre-Image  
(the UML model is supposed to be the blue-print for a software system)

A description of behaviour could serve the following purposes:

- Require** Behaviour.
    - "System definitely does this"
    - "This sequence of inserting money and requesting and getting water must be possible."
    - (Otherwise, the software for the vending machine is completely broken.)
  - Allow** Behaviour.
    - "System does subset of this"
    - "After inserting money and choosing a drink the drink is dispensed (if in stock)." (if the implementation insists on taking the money first, that's a fair choice.)
  - Forbid** Behaviour.
    - "System never does this"
    - "This sequence of getting both, a water and all money back, must not be possible." (Otherwise the software is broken.)
- Note:** the latter two are trivially satisfied by doing nothing...

[Harel, 1997] propose to distinguish constructive and reflective descriptions:

- "A language is **constructive** if it contributes to the dynamic semantics of the model. That is, its constructs contain information needed in executing the model or in translating it into executable code."
- A constructive description tells **how** things are computed (which can then be desired or undesired).

- "Other languages are **reflective** or **assertive**, and can be used by the system modeller to capture parts of the thinking that go into building the model – behavior included – to derive and present views of the model, statically or during execution, or to set constraints on behavior in preparation for verification."
- A reflective description tells **what** shall or shall not be computed.

**Note:** No sharp boundaries!

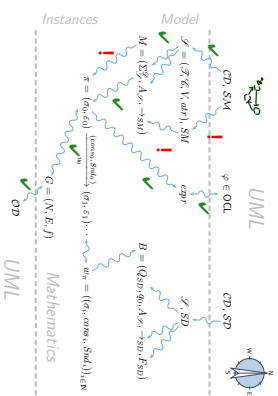
UML provides two visual formalisms for constructive description of behaviours:

- Activity Diagrams
- State-Machine Diagrams

We (exemplary) focus on State-Machines because

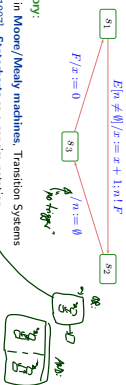
- somewhat "practice proven" (in different flavours),
- prevalent in embedded systems community,
- indicated useful by [Dehning and Parsons, 2006] survey, and
- Activity Diagram's intuition changed (between UML 1.x and 2.x) from transition-system-like to petri-net-like...

Example state machine:



UML State Machines: Overview

UML State Machines



Brief History:

- Rooted in Moore/Meady machines, Transition Systems
- Harel, 1987<sup>2</sup>: Statecharts as a concise notation, introduces in particular hierarchical states.
- Manifest in tool **StateSpace**, Harel et al., 1990 (simulation, code-generation); nowadays also in **Modelb/Simulink**, etc.
- From UML 1.x on: **State-Machines** (i.e. **State-Machine Diagrams**) (not the official name, but understood: UML Statecharts)
- Late 1990's: tool **RUNGED** with code-generation for state machines.

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

Readmap: Chronologically

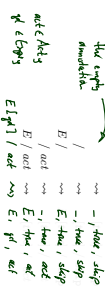
- (i) What do we (have to) cover? UML State Machine Diagrams **Syntax**
  - (ii) Def.: Signature with signals.
  - (iii) Def.: Core state machine.
  - (iv) Map UML State Machine Diagrams to core state machines.
- Semantics:**
- (v) Def.: Either (aka. event pool)
  - (vi) Def.: System configuration.
  - (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.



Reconsider the syntax of transition annotations:

$annot ::= [ (event) [ : (event) ] ] [ : (guard) ] [ : [ / (action) ] ]$

and let's play a bit with the defaults:



In the standard, the syntax is even more elaborate:

- $E(x)$  — when examining  $E$  in object  $u$ , attribute  $x$  of  $u$  is assumed the corresponding attribute of  $E$ .
- $E(x; ?)$  — similar, but  $x$  is a local variable scope is the transition.

References

State-Machines belong to Classes, An Example by Object

- In the following, we assume that a UML model consists of a set  $\mathcal{C}$  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  $C_{SM} \in \mathcal{C}$
- For simplicity, we even assume a bijection, i.e. we assume that each class  $C \in \mathcal{C}$  has a state machine  $SM_C$  and that its class  $C_{SM_C}$  is  $C$ . If not explicitly given, then this one:

$SM_C = (\{s_0\}, \{s_0\}, \{s_0\}, \{s_0\})$

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 class  $C$  executes  $SM_C$  and *not* in *their program*.

**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

[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.

[Dobling and Parsons, 2006] Dobling, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, 49(3):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 Grunberg, 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):51–62.

[Harel et al., 1990] Harel, D., Lachover, H., et al. (1990). Statestate: 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.

