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**

\[ \sigma_0, \sigma_1, \ldots \in \Sigma^\omega \]

of system states.
\( CD, SM \)  

\( \mathcal{I} = (\mathcal{F},\mathcal{C},V,\text{atr}), SM \)

\( (\Sigma_{\mathcal{I}}, A_{\mathcal{I}}, \rightarrow_{SM}) = M \)

\( \varphi \in \text{OCL} \)

\( CD, SD \)

\( B = (Q_{SD}, q_0, A_{\mathcal{I}}, \rightarrow_{SD}, F_{SD}) \)

\( (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \xrightarrow{(\text{cons}_1, \text{Snd}_1)} \ldots \)

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

\( O D \)
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
- \textbf{[Harel, 1987]}: Statecharts as a concise notation, introduces in particular hierarchical states.
- Manifest in tool Statemate \textbf{[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.

\textbf{Note}: there is a common core, but each dialect interprets some constructs subtly different \textbf{[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**

\[ E[n \neq \emptyset]/x := x + 1; n! F \]

\[ F/x := 0 \]

\[ /n := \emptyset \]

**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?


Ein Zustand löst von sich aus bestimmte Ereignisse aus:
- entry beim Betreten;
- do während des Aufenthaltes;
- completion beim Erreichen des Endzustandes einer Unter-Zustandsmaschine
- exit beim Verlassen.

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


Wenn ein Regionsendzustand erreicht wird, wird der gesamte komplexer Zustand beendet, also auch alle parallelen Regionen. Ein verfeinerner 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.
Definition. A tuple

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

is called signature (with signals) if and only if

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

is a signature (as before).

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

Alternative (maybe even better) definition:

$$\exists(\mathcal{I}) = \{<C, S, a, t> \in \mathcal{I} \mid \text{signal} \in S_t\}$$
Definition.
A core state machine over signature $\mathcal{S} = (T, \mathcal{C}, V, \text{atr}, \mathcal{E})$ is a tuple

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

where

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

$$\rightarrow \subseteq S \times (\mathcal{E} \cup \{\_\}) \times \text{Expr}_\mathcal{S} \times \text{Act}_\mathcal{S} \times S$$

is a labelled transition relation.

We assume a set $\text{Expr}_\mathcal{S}$ of boolean expressions over $\mathcal{S}$ (for instance OCL, may be something else) and a set $\text{Act}_\mathcal{S}$ of \textbf{actions}.
From UML to Core State Machines: By Example

UML state machine diagram \( SM \):

\[
\text{annot ::= } \left[ \langle \text{event} \rangle \cdot \langle \text{event} \rangle \ast \right] \left[ \lceil \langle \text{guard} \rangle \rceil \right] \left[ \lfloor \langle \text{action} \rangle \rfloor \right]
\]

with

- \( \text{event} \in \mathcal{E} \),
- \( \text{guard} \in \text{Expr}_\mathcal{F} \)
- \( \text{action} \in \text{Act}_\mathcal{F} \)

maps to

\[
M(SM) = (\{s_1, s_2\}, s_0, (s_1, \text{event}, \text{guard}, \text{action}, s_2))
\]
Reconsider the syntax of transition annotations:

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

and let’s play a bit with the defaults:

\[
\begin{align*}
(\text{empty annot} & : ) & \mapsto (s_1, \text{true}, \text{skip}, s_2) \\
/ & \mapsto (s_1, [\cdot \text{true}], \text{true}, \text{skip}, s_2) \\
E / & \mapsto (s_1, E, \text{true}, \text{skip}, s_2) \\
/ \text{act} & \mapsto (s_1, \text{true}, \text{act}, s_2) \\
E / \text{act} & \mapsto (s_1, E, \text{true}, \text{act}, s_2) \\
E[e] / \text{act} & \mapsto (s_1, E, e, \text{act}, s_2)
\end{align*}
\]

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 \(E \in E\)
- \(E(v : \tau)\) — similar, but \(v\) is a local variable, scope is the transition

\[\text{(we don’t discuss these)}\]

\[\text{we view as an abbrev.} \quad E_{[g] / a}\]

\[E_{[g] / a}\]

\[E_{[g] / a}\]
\[ S = \left( \{ \text{Int} \}, \{ (\text{Int}, 0, 0), (E, \text{signal}, 0), \right. \\
\left. (\text{F}, \text{signal}, 0, 0) \}, \{ x: \text{Int}, y: \text{Int}, n: \text{Int} \}, \\
\{ c \mapsto \{ x, n \}, E \mapsto y \} \right) \]

\[ \mathcal{E}(S) = \{ E, \text{F} \} \]

\[ M = ( \{ s_1, s_2, s_3, s_4, s_5 \}, \\
\{ (s_1, \text{false}, \text{skip}, s_3), \\
(s_1, E \text{ not odd(int(x)))}, \\
(\text{F}, s_2), \\
\ldots \} \) \]
State-Machines belong to Classes

- In the following, we assume that a UML models consists of a set \( \mathcal{CD} \) of class diagrams and a set \( \mathcal{SM} \) of state chart diagrams (each comprising one state machine \( SM \)).

- Furthermore, we assume each that each state machine \( SM \in \mathcal{SM} \) is associated with a class \( C_{SM} \in \mathcal{C}(\mathcal{I}). \)

- For simplicity, we even assume a bijection, i.e. we assume that each class \( C \in \mathcal{C}(\mathcal{I}) \) has a state machine \( SM_C \) and that its class \( C_{SM_C} \) is \( C \).

  If not explicitly given, then this one:

  \[
  SM_0 := ([s_0], s_0, (s_0, \_ , true, skip, s_0)).
  \]

  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 \( SM_C \).

**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


