Software Design, Modelling and Analysis in UML

Lecture 10: Constructive Behaviour, State Machines Overview

2011-12-14

Prof. Dr. Andreas Podelski, 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**

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

of system states.
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:

```
E[n ≠ ∅]/x := x + 1; n! F
F/x := 0
/n := ∅
```

- 06-2011-12-14 - Sbehav -}

5/75
**UML State Machines**

- 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.

**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?**

**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

\[ \mathcal{S} = (\mathcal{P}, \mathcal{E}, V, atr, \delta) \]

\[ \mathcal{E} \] a set of signals,

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

\[ (\mathcal{P}, \mathcal{E} \cup \delta, V, atr) \]

is a signature (as before).

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

![Signature Diagram]

**Core State Machine**

**Definition.**

A core state machine over signature \( \mathcal{S} = (\mathcal{P}, \mathcal{E}, V, atr, \delta) \) is a tuple

\[ M = (S, s_0, \rightarrow) \]

where

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

\[ \rightarrow \subseteq S \times (\delta \cap \{ \} ) \times \text{Expr}_{\mathcal{S}} \times \text{Act}_{\mathcal{S}} \times S \]

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 actions.
From UML to Core State Machines: By Example

UML state machine diagram $SM$:

$$\text{annot} ::= \left[ \begin{array}{c} \langle \text{event} \rangle [',', \langle \text{event} \rangle] \times [',', \langle \text{guard} \rangle] \times [',', \langle \text{action} \rangle] \end{array} \right]$$

with

- event $\in \mathcal{E}$,
- guard $\in \text{Expr}_{\mathcal{S}}$
- action $\in \text{Act}_{\mathcal{S}}$

maps to

$$M(SM) = \{ \{s_1, s_2\}, s_1 \xrightarrow{\text{event, guard, action}} s_2 \}$$

Annotations and Defaults in the Standard

Reconsider the syntax of transition annotations:

$$\text{annot} ::= \left[ \begin{array}{c} \langle \text{event} \rangle [',', \langle \text{event} \rangle] \times [',', \langle \text{guard} \rangle] \times [',', \langle \text{action} \rangle] \end{array} \right]$$

and let’s play a bit with the defaults:

$$\begin{array}{c}
\text{(empty annot.)} & \xrightarrow{} (s_1, \text{true}, \text{skip}, s_2) \\
/ & \xrightarrow{} (s_1, \text{true}, \text{skip}, s_2) \\
E / & \xrightarrow{} (s_1, E, \text{true}, \text{skip}, s_2) \\
/ \text{act} & \xrightarrow{} (s_1, \text{true}, \text{act}, s_2) \\
E / \text{act} & \xrightarrow{} (s_1, E, \text{true}, \text{act}, s_2)
\end{array}$$

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.
State-Machines belong to Classes

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

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

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

If not explicitly given, then this one:

\[
SM_0 := (\{s_0\}, s_0, (s_0, \text{true, skip}, s_0)).
\]

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


