Recall: Core State Machine

Definition. A core state machine over signature $S = (T, C, V, atr, E)$ 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,
- $\rightarrow \subseteq S \times (E \cup \{\}\) trigger \times Expr S \times Act S \times S$ is a labelled transition relation.

We assume a set $Expr S$ of boolean expressions (may be OCL, may be something else) and a set $Act S$ of actions over $S$.

From UML to Core State Machines: By Example

UML state machine diagram

```
SM1
s1
annot
annot::= [⟨event⟩]['.']⟨event⟩∗['.']['] ['/']⟨action⟩] [']'

with
- event ∈ E
- guard ∈ Expr S (default: true, assumed to be in Expr S)
- action ∈ Act S (default: skip, assumed to be in Act S)
```

Maps to $M(SM1) = (\{s1, s2\}, s1, (s1, event, guard, action, s2))$
What is that useful for?

- 12 – 2014-12-09 – Sstmsyn –

7

• No Event

• No annotation

State-Machines belong to Classes

In the following, we assume that a UML models consists of a set $C_D$ of class diagrams and a set $S_M$ of state chart diagrams (each comprising one state machine $SM$).

Furthermore, we assume that each state machine $SM \in S_M$ is associated with a class $C_{SM} \in C$.

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

If not explicitly given, then this one:

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

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.)

The Basic Causality Model

6.2.3 The Basic Causality Model

"Causality model" is a specification of how things happen at run time [ ... ].

The causality model is quite straightforward:

• Objects respond to messages that are generated by objects executing communication actions.

• When these messages arrive, the receiving objects eventually respond by executing the behavior that is matched to that message.

• The dispatching method by which a particular behavior is associated with a given message depends on the higher-level formalism used and is not defined in the UML specification (i.e., it is a semantic variation point).

The causality model also subsumes behaviors invoking each other and passing information to each other through arguments to parameters of the invoked behavior, [ ... ]. This purely 'procedural' or 'process' model can be used by itselfor in conjunction with the object-oriented model of the previous example.

6.2.3 The Basic Causality Model

- 12 – 2014-12-09 – Sstmstd –

10

\[ \text{Int}_{n_0..1} \times \]

15.3.12 StateMachine

• Event occurrences are detected, dispatch, and then processed by the state-machine, one at a time.

• The semantics of event occurrence processing is based on the run-to-completion assumption, interpreted as run-to-completion processing.

• Run-to-completion processing means that an event [ ... ] can only be taken from the pool and dispatched if the processing of the previous [ ... ] is fully completed.

• The processing of a single event occurrence by a state machine is known as a run-to-completion step.

• Before commencing on a run-to-completion step, a state machine is in a stable state configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.

• The same conditions apply after the run-to-completion step is completed.

• Thus, an event occurrence will never be processed [ ... ] in some intermediate and inconsistent situation.

• [IOW,] The run-to-completion step is the passage between two state configurations of the state machine.

• The run-to-completion assumption simplifies the transition function of the StM, since concurrency conflicts are avoided during the processing of event, allowing the StM to safely complete its run-to-completion step.

The Basic Causality Model
The order of dequeuing is not defined, leaving open the possibility of modeling different priority-based schemes.

Run-to-completion may be implemented in various ways. [...]

We have to formally define what event occurrence is.

We have to define where events are stored — what the event pool is.

We have to explain how transitions are chosen — “matching”.

We have to explain what the effect of actions is — on state and event pool.

We have to decide on the granularity — micro-steps, steps, run-to-completion steps (aka. super-steps)?

We have to formally define a notion of stability and RTC-step completion.

And then: hierarchical state machines.

(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.
An (single, global, shared, reliable) FIFO queue is an ether:

\[ \text{Eth} \cdot \text{ready} \oplus \text{⊖} \cdot \text{·} \]

One FIFO queue per active object is an ether.

Lossy queue (\(\oplus\) becomes a relation then).

One-place buffer.

Priority queue.

Multi-queues (one per sender).

Trivial example: sink, “black hole”.

15.3.12 StateMachine

The order of dequeuing is \textbf{not defined}, leaving open the possibility of modeling different priority-based schemes.

Run-to-completion may be implemented in various ways.