Contents & Goals

Last Lecture:
• Core State Machines
• UML State Machines 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:
  - Ether, System Configuration, Transformer
  - Run-to-completion Step
  - Putting It All Together

Recall: UML State Machines

Core State Machine
Definition.

A core state machine, over signature \( CB \), 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,
- \( \rightarrow \subseteq S \times (BV \cup \{\} \times Expr / CB \times \text{guard} \times \text{Act / CB} \times \text{action} \times S) \) is a labelled transition relation.

We assume a set \( \text{Expr / CB} \) of boolean expressions over \( \text{CB} \) (for instance OCL, may be something else) and a set \( \text{Act / CB} \) of actions.

From UML to Core State Machines: By Example

UML state machine diagram SM:

```
annot ::= [⟨event⟩] [']⟨event⟩']∗ [⟨guard⟩] [']⟨action⟩']
```

with

- \( \text{event} \in \text{CB} \),
- \( \text{guard} \in \text{Expr / CB} \) (default: \( \text{true} \), assumed to be in \( \text{Expr / CB} \))
- \( \text{action} \in \text{Act / CB} \) (default: \( \text{skip} \), assumed to be in \( \text{Act / CB} \))

mapsto \( M(\text{SM}) = (\{s_1, s_2\} \times S, s_1 \times s_0, (s_1, \text{event}, \text{guard}, \text{action}, s_2)) \times \rightarrow) \)
6.2.3 The Basic Causality Model

[OMG, 2007b, 12]

"Causality model" is a specification of how things happen at runtime...

- 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 argument to parameters of the invoked behavior.
- This purely 'procedural' or 'process' model can be used by itself or in conjunction with the object-oriented model of the previous example.

15.3.12 StateMachine

[OMG, 2007b, 563]

- Event occurrences are detected, dispatched, 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 event 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 State Machine, as concurrency conflicts are avoided during the processing of events, allowing the State Machine to safely complete its run-to-completion step.

And?

\[
\begin{align*}
& s_1 \quad s_2 \quad s_3 \\
& s'_{1} \quad s'_{2} \quad s'_{3}
\end{align*}
\]

\[
E/ \quad E/ \quad E/
\]

- We have to formally define what an 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 an notion of stability and RTC-step completion.
- And then: hierarchical state machines.
But we'll later briefly discuss the "discarding" of events.

Oftenseen, minimal requiremengether reordering, or duplication. One-place buffer.

In some cases, this is instantaneous and completely reliable, while in others medium, and numerous other factors. Lossy queue.

One FIFO queue per active object is ether.

The means by which requests are transported to their target depend on the medium, and numerous other factors. "Discarding" of events.

The standard distinguishes (among others)

- Run-to-completion
- Non-running
- Complete

**Mathematics**

- \( S \)
- \( m \)
- \( v \)
- \( w \)
- \( x \)
- \( y \)
- \( z \)
- \( \sigma \)
- \( \pi \)
- \( \rho \)
- \( \delta \)
- \( \theta \)
- \( \lambda \)
- \( \mu \)
- \( \nu \)
- \( \xi \)
- \( \omicron \)
- \( \Psi \)

**Roadmap: Chronologically**

- Start with the overview of the standard
- Follow with the discussion of the core concepts
- Discuss the various aspects of the standard in detail
