Contents & Goals

Last Lecture:
- State machine syntax
- Core state machines

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:
  - The basic causality model
  - Ether
  - System Configuration, Transformer
  - Examples for transformer
  - Run-to-completion Step
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 itself or in conjunction with the object-oriented model of the previous example.”
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 [...] 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**.
15.3.12 StateMachine [OMG, 2007b, 563]

- 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. [...]


our choice of the effect of \( x \):
assign \(-1\) to \( x \) of all donc \( C \)

all objects stable

\( \sigma \) is unstable

rest remains unchanged
And?

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.
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.
System Configuration, Ether, Transformer
**Definition.** Let $\mathcal{S} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E})$ be a signature with signals and $\mathcal{D}$ a structure.

We call a tuple $(Eth, ready, \oplus, \ominus, [\cdot])$ an **ether** over $\mathcal{S}$ and $\mathcal{D}$ if and only if it provides

- a **ready** operation which yields a set of events that are ready for a given object, i.e.
  
  $$
  \text{ready} : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow 2^{\mathcal{D}(\mathcal{E})}
  $$

- a operation to **insert** an event destined for a given object, i.e.

  $$
  \oplus : Eth \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \rightarrow Eth
  $$

- a operation to **remove** an event, i.e.

  $$
  \ominus : Eth \times \mathcal{D}(\mathcal{E}) \rightarrow Eth
  $$

- an operation to **clear** the ether for a given object, i.e.

  $$
  [\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow Eth.
  $$
Ether: Examples

- A (single, global, shared, reliable) FIFO queue is an ether:
  - \( Eth = (D(C) \times D(E))^* \)
  - \( \text{ready} \{ (u,e) : E, v \} : \begin{cases} \{ (u,e) \}, & \text{if } u = v \\ \emptyset, & \text{otherwise} \end{cases} \)
  - \( \oplus (e_1, v, e) = e \cdot (u, e) \)
  - \( \ominus ((u,e), e, f) = \begin{cases} \{ (u,e) \}, & \text{if } f = e \\ \{ (u,e), e, \}, & \text{otherwise} \end{cases} \)
  - \([ \cdot ]): \text{remove all (u,e) pairs from given sequence}\)

- One FIFO queue per active object is an ether. [Rhapsody’s choice]

  (Lossy queue.) (because \( \ominus, \text{ready} \) are function)

- One-place buffer.

- Priority queue.

- Multi-queues (one per sender).

- Trivial example: sink, “black hole”.

- Set of events
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. [...]
Ether and [OMG, 2007b]

The standard distinguishes (among others)

- **SignalEvent** [OMG, 2007b, 450] and **Reception** [OMG, 2007b, 447].

On **SignalEvents**, it says

_A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition._ [OMG, 2007b, 449]

[...]

**Semantic Variation Points**

_The means by which requests are transported to their target depend on the type of requesting action, the target, the properties of the communication medium, and numerous other factors._

_In some cases, this is instantaneous and completely reliable while in others it may involve transmission delays of variable duration, loss of requests, reordering, or duplication._

_(See also the discussion on page 421.)_ [OMG, 2007b, 450]

Our _ether_ is a general representation of the possible choices. (_X needs relation_)

**Often seen minimal requirement**: order of sending _by one object_ is preserved.

But: we’ll later briefly discuss “discarding” of events.
System Configuration

Definition. Let \( \mathcal{S}_0 = (\mathcal{S}, \emptyset, \mathcal{S}_0, V_0, \mathcal{V}_0) \) be a signature with signals, \( \mathcal{V}_0, \mathcal{S}_0, \mathcal{V}_0 \) a structure of \( \mathcal{S}_0 \), (\( \mathcal{S}_0, \mathcal{V}_0, \mathcal{S}_0, \mathcal{V}_0 \)) an ether over \( \mathcal{S}_0 \) and \( \mathcal{V}_0 \).

A system configuration over \( \mathcal{S}_0, \mathcal{V}_0, \) and \( \mathcal{V}_0 \) is a pair

\[
\mathfrak{S} = (\mathcal{S}_0 \cup \mathcal{S}_M \cup \mathcal{V}_0, \mathcal{V}_0, \mathcal{V}_0),
\]

where

\[
\mathcal{S}_0 = (\mathcal{S}_0 \cup \mathcal{S}_M \cup \mathcal{V}_0 \cup \mathcal{V}_0),
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]

\[
\mathcal{V}_0,
\]
$y_0 = (\{\text{Int}, \{c, e_3\}, \{a, x\}, \{c' \mapsto x, e \mapsto e_3\}, \{e_3\}\})$

$y = (\{\text{Int}, s_{MC}\}, \{c, e\})$

$\sigma:\ C$

$E$

$C$

$t_1$

$t_2$

$S_{MC}$

$E_1$

$E_0$

$\delta:\ C'$

$x = 22$

$\text{stable}_C = \text{true}$

$s_C = s_1$

$e_1:\ E$

$a = 10$
We start with some signature with signals $\mathcal{S}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, \text{atr}_0, \mathcal{E})$.

A system configuration is a pair $(\sigma, \varepsilon)$ which comprises a system state $\sigma$ wrt. $\mathcal{I}$ (not wrt. $\mathcal{I}_0$).

Such a system state $\sigma$ wrt. $\mathcal{I}$ provides, for each object $u \in \text{dom}(\sigma)$,

- values for the explicit attributes in $V_0$,
- values for a number of implicit attributes, namely
  - a stability flag, i.e. $\sigma(u)(\text{stable})$ is a boolean value,
  - a current (state machine) state, i.e. $\sigma(u)(\text{st})$ denotes one of the states of core state machine $M_C$,
  - a temporary association to access event parameters for each class, i.e. $\sigma(u)(\text{params}_E)$ is defined for each $E \in \mathcal{E}$.

For convenience require: there is no link to an event except for $\text{params}_E$. 
References
References

