Software Design, Modelling and Analysis in UML

Lecture 12: Core State Machines II

2013-12-09

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

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

6.2.3 The Basic Causality Model [OMG, 2007b, 12]

*"'**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, [...].

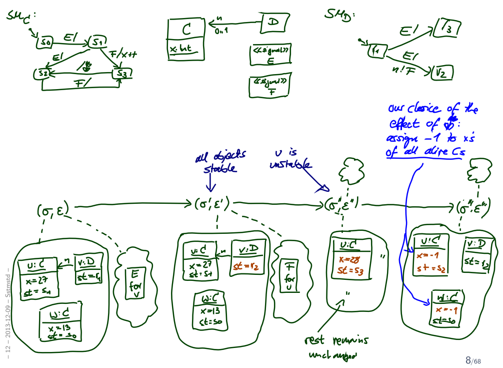
This purely 'procedural' or 'process' model can be used by itself or in conjunction with the object-oriented model of the previous example."

15.3.12 StateMachine [OMG, 2007b, 563]

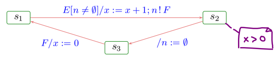
- Event occurrences are detected, dispatched, and then processed by the state machine, one at a time.
- The semantics of event occurrence processing is based on the **run-to-completion assumption**, interpreted as **run-to-completion processing**.
- **Run-to-completion processing** means that an event [...] can only be taken from the pool and dispatched if the processing of the previous [...] 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]

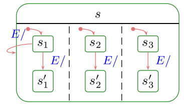
- The order of dequeuing is **not defined**, leaving open the possibility of modeling different priority-based schemes.
- Run-to-completion may be implemented in **various ways**. [...]



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.

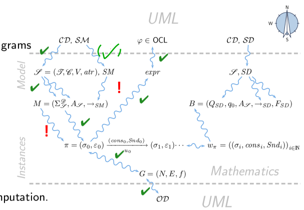


Roadmap: Chronologically

(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

Ether aka. Event Pool

Definition. Let $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ be a signature with signals and $\mathscr{D}$ a structure.

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

- a **ready** operation which yields a set of events that are ready for a given object, i.e. for an event pool ε... ..and an object identity u... ...obtain a set of signal indentities (or events)
$$ready : Eth \times \mathscr{D}(\mathscr{C}) \to 2^{\mathscr{D}(\mathscr{E})}$$
set of E instances
- a operation to **insert** an event destined for a given object, i.e.
for ε, destination id, event id — obtain a new event pool ε'
$$\oplus : Eth \times \mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E}) \to Eth$$
- a operation to **remove** an event, i.e.
ε, event id, ε'
$$\ominus : Eth \times \mathscr{D}(\mathscr{E}) \to Eth$$
- an operation to clear the ether for a given object, i.e.
$$[\,\cdot\,] : Eth \times \mathscr{D}(\mathscr{C}) \to Eth.$$

Ether: Examples

$(Eth, ready, \oplus, \ominus, [\cdot])$
$ready : Eth \times \mathscr{D}(\mathscr{C}) \to 2^{\mathscr{D}(\mathscr{E})}$

- A (single, global, shared, reliable) FIFO queue is an ether:
  - $Eth = (\mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E}))^*$
    the set of all finite sequences of pairs $(u, e) \in \mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E})$ — id of destination object, id of a signal instance
  - seq. concat. $ready((u,e).\varepsilon, v) = \begin{cases} \{(u,e)\}, & \text{if } u = v \\ \emptyset, & \text{otherwise} \end{cases}$
  - $\oplus(\varepsilon, u, e) = \varepsilon.(u,e)$
  - $\ominus((u,e).\varepsilon, f) = \begin{cases} \varepsilon, & \text{if } f = e \\ (u,e).\varepsilon, & \text{otherwise} \end{cases}$
  - $[\,\cdot\,]$: remove all (u,e) pairs from given sequence
- One FIFO queue per active object is an ether. [Rhapsody's choice]
- (Lossy queue.) (because ⊕, ready are function)
- One-place buffer.
- Priority queue.
- Multi-queues (one per sender).
- Trivial example: sink, "black hole".
- Set of events
- ...

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**. [...]

Ether and [OMG, 2007b]

"receiving takes place"
more conceptional; for us: dispatch / discard

The standard distinguishes (among others)

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

On **SignalEvents**, it says

for us: event

> *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**
> = messages
> *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. (⌜⌟* needs relations)

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 $\mathscr{S}_0 = (\mathscr{T}_0, \mathscr{C}_0, V_0, atr_0, \mathscr{E})$ be a signature with signals, $\mathscr{D}_0$ a structure of $\mathscr{S}_0$, $(Eth, ready, \oplus, \ominus, [\,\cdot\,])$ an ether over $\mathscr{S}_0$ and $\mathscr{D}_0$. Furthermore assume there is one core state machine M_C per class $C \in \mathscr{C}$.

A system configuration over $\mathscr{S}_0$, $\mathscr{D}_0$, and Eth is a pair

a type name for each state machine

$$(\sigma, \varepsilon) \in \Sigma_{\mathscr{S}}^{\mathscr{D}} \times Eth$$

where

if Bool ∉ $\mathscr{T}_0$, then add it and have $\mathscr{D}(Bool) = \mathbb{B}$

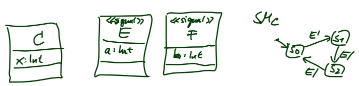
initial state of M_C

each object can refer to the signal instances in order to access event attributes

the only links to events are via the params

- $\mathscr{S} = (\mathscr{T}_0 \,\dot\cup\, \{S_{M_C} \mid C \in \mathscr{C}\}, \quad \mathscr{C}_0,$
  $V_0 \,\dot\cup\, \{\langle stable : Bool, -, \mathsf{true}, \emptyset\rangle\}$
  $\dot\cup\, \{\langle st_C : S_{M_C}, +, s_0, \emptyset\rangle \mid C \in \mathscr{C}\}$
  $\dot\cup\, \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset\rangle \mid E \in \mathscr{E}_0\},$
  $\{C \mapsto atr_0(C)$
  $\cup \{stable, st_C\} \cup \{params_E \mid E \in \mathscr{E}_0\} \mid C \in \mathscr{C}\}, \quad \mathscr{E}_0)$
- $\mathscr{D} = \mathscr{D}_0 \,\dot\cup\, \{S_{M_C} \mapsto S(M_C) \mid C \in \mathscr{C}\}$, and

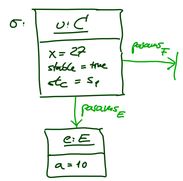
  set of states of state machine M_C of C
- $\sigma(u)(r) \cap \mathscr{D}(\mathscr{E}_0) = \emptyset$ for each $u \in \mathrm{dom}(\sigma)$ and $r \in V_0$.



$\mathscr{S}_0 = (\{Int\}, \{C, E\}, \{a, x\}, \{C \mapsto \{x\}, E \mapsto \{a\}\}, \{E\})$

$\mathscr{S} = (\{Int, S_{M_C}\}, \{C, E\},$
$\{a, x, stable_C : Bool, st_C : S_{M_C}\} \cup \{params_E : E_{0,1}, params_F : F_{0,1}\}$
$\{C \mapsto \{x, stable_C, st_C\} \cup \{params_E, params_F\}$
$E \mapsto \{a\}\},$
$\{E\}),$

$\mathscr{D}(S_{M_C}) = \{s_0, s_1, s_2\}$



System Configuration Step-by-Step

- We start with some signature with signals $\mathscr{S}_0 = (\mathscr{T}_0, \mathscr{C}_0, V_0, atr_0, \mathscr{E})$.
- A **system configuration** is a pair (σ, ε) which comprises a system state σ wrt. $\mathscr{S}$ (not wrt. $\mathscr{S}_0$).
- Such a **system state** σ wrt. $\mathscr{S}$ provides, for each object $u \in \mathrm{dom}(\sigma)$,
  - values for the **explicit attributes** in V_0,
  - values for a number of **implicit attributes**, namely
    - a **stability flag**, i.e. $\sigma(u)(stable)$ is a boolean value,
    - a **current (state machine) state**, i.e. $\sigma(u)(st)$ denotes one of the states of core state machine M_C,
    - a temporary association to access **event parameters** for each class, i.e. $\sigma(u)(params_E)$ is defined for each $E \in \mathscr{E}$.
- For convenience require: there is **no link to an event** except for $params_E$.

References

References

[Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.