Software Design, Modelling and Analysis in UML

Lecture 11: Core State Machines I

2015-12-10

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- What makes a class diagram a good class diagram?
- Core State Machine syntax

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:**
  - UML standard: basic causality model
  - Ether
  - Transformers
  - Step, Run-to-Completion Step

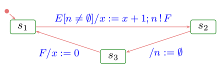
The Basic Causality Model

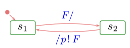
6.2.3 The Basic Causality Model (OMG, 2011b, 11)

*"**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).





6.2.3 The Basic Causality Model (OMG, 2011b, 11)

*"**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).

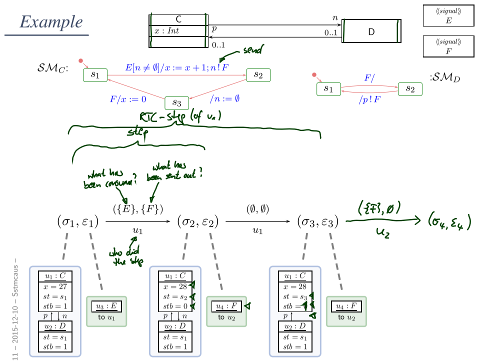
*The causality model also **subsumes** behaviors **invoking each other** and passing information to each other through arguments to parameters of the invoked behavior, [...].*

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

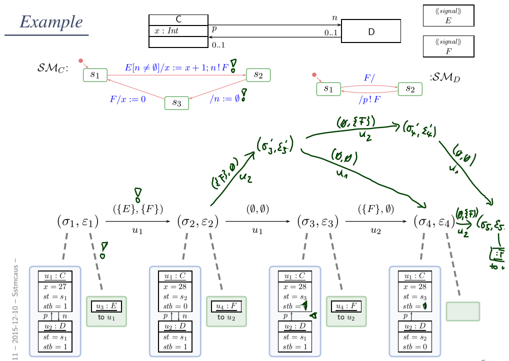
15.3.12 StateMachine (OMG, 2011b, 574)

- 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**.
- 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**. [...]

Example



Example



Ether

Recall: 15.3.12 StateMachine (OMG, 2011b, 563)

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

Ether and OMG (2011b)

The standard distinguishes (among others)

- **SignalEvent** (OMG, 2011b, 450) and **Reception** (OMG, 2011b, 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, 2011b, 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, 2011b, 450)

Our **ether** (→ in a minute) is a general representation of **many possible choices**.

Often seen minimal requirement: order of sending **by one object** is preserved.

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

for an event pool ε... – and an object identity u... → obtain a set of sig. instances ready for consumption

- a ready operation which yields a set of events (i.e., signal instances) that are ready for a given object, i.e.

$$ready : Eth \times \mathscr{D}(\mathscr{C}) \to 2^{\mathscr{D}(\mathscr{E})}$$

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

destination signal instance

$$\oplus : Eth \times \mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E}) \to Eth$$

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

signal instance

$$\ominus : Eth \times \mathscr{D}(\mathscr{E}) \to Eth$$

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

$$[\cdot] : Eth \times \mathscr{D}(\mathscr{C}) \to Eth.$$

Example: FIFO Queue

dest. id signal instance

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

- $Eth = \left(\mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E})\right)^*$ ← finite sequences   e.g. $\varepsilon = (u_1, e_3), (u_2, e_4)$

  the set of finite sequences of pairs $(u,e) \in \mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E})$
- $ready : Eth \times \mathscr{D}(\mathscr{C}) \to 2^{\mathscr{D}(\mathscr{E})}$

  $$((u_1,e).\varepsilon, u_2) \mapsto \begin{cases} \{(u_1,e)\}, & \text{if } u_1 = u_2 \\ \emptyset, & \text{otherwise (also if } \varepsilon \text{ is empty)} \end{cases}$$
- $\oplus : Eth \times \mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E}) \to Eth$

  $(\varepsilon, u, e) \mapsto \varepsilon . (u,e)$
- $\ominus : Eth \times \mathscr{D}(\mathscr{E}) \to Eth$

  $$((u,e_1).\varepsilon, e_2) \mapsto \begin{cases} \varepsilon, & \text{if } e_2 = e_1 \\ (u,e_1).\varepsilon, & \text{otherwise (also if empty)} \end{cases}$$
- $[\,\cdot\,] : Eth \times \mathscr{D}(\mathscr{C}) \to Eth$ ← dest. id u

  remove all pairs (u,e) from ε

Other Examples

- One FIFO queue per active object is an ether.

  $Eth = \mathscr{D}(\mathscr{C}) \to \left(\mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E})\right)^*$
- One-place buffer.

  $Eth = \epsilon \,\dot\cup\, (\mathscr{D}(\mathscr{C}) \times \mathscr{D}(\mathscr{E}))$
- Priority queue.
- Multi-queues (one per sender).
- Trivial example: sink, "black hole".
- Lossy queue ($\oplus$ needs to become a relation then).
- ...

System Configuration

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

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

a new type for each class

where

- $\mathscr{S} = (\mathscr{T}_0 \,\dot\cup\, \{S_{M_C} \mid C \in \underbrace{\mathscr{C}_0}_{\setminus \mathscr{E}_0}\}, \quad \mathscr{C}_0,$

  if $Bool \notin \mathscr{T}$ then add it and have $\mathscr{D}(Bool) = \{0,1\}$

  $$\begin{aligned} &V_0 \,\dot\cup\, \{\langle stable : Bool, -, true, \emptyset\rangle\} \\ &\quad \dot\cup\, \{\langle st_C : S_{M_C}, +, s_0, \emptyset\rangle \mid C \in \underbrace{\mathscr{C}_0}_{\setminus \mathscr{E}_0}\} \\ &\quad \dot\cup\, \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset\rangle \mid E \in \mathscr{E}_0\}, \\ &\{C \mapsto atr_0(C) \\ &\quad \cup \{stable, st_C\} \cup \{params_E \mid E \in \mathscr{E}_0\} \mid C \in \mathscr{C}\}, \quad \mathscr{E}_0) \end{aligned}$$
- $\mathscr{D} = \mathscr{D}_0 \,\dot\cup\, \{S_{M_C} \mapsto S(M_C) \mid C \in \mathscr{C}\}$, and ← set of states of state machine of C
- $\sigma(u)(r) \cap \mathscr{D}(\mathscr{E}_0) = \emptyset$ for each $u \in \mathrm{dom}(\sigma)$ and $r \in V_0$.

References

References

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.