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
“Causality model’ is a specification of how things happen at runtime

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).

\[
E[n \neq \emptyset]/x := x + 1; n ! F
\]

\[
F/x := 0 \quad s_3 \quad /n := \emptyset
\]

\[
F/ \quad s_1 \quad \quad s_2
\]

\[
/p ! F
\]

\[
s_1 \quad \quad s_2
\]
“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.

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

\[ E[n \neq \emptyset]/x := x + 1; n! F \]

\[ F/x := 0 \]

\[ /n := \emptyset \]

\[ \langle \{E\}, \{F\} \rangle \]

\[ \langle \emptyset, \emptyset \rangle \]

\[ \langle \{E\}, \emptyset \rangle \]

\[ (\sigma_1, \varepsilon_1) \]

\[ (\sigma_2, \varepsilon_2) \]

\[ (\sigma_3, \varepsilon_3) \]

\[ (\sigma_4, \varepsilon_4) \]

\[ u_1 \]

\[ u_2 \]

\[ u_3 : E \]

\[ u_4 : F \]

\[ u_1 : C \]

\[ x = 27 \]

\[ st = s_1 \]

\[ stb = 1 \]

\[ p \]

\[ n \]

\[ u_2 : D \]

\[ x = 28 \]

\[ st = s_2 \]

\[ stb = 0 \]

\[ p \]

\[ n \]

\[ u_1 : C \]

\[ x = 28 \]

\[ st = s_3 \]

\[ stb = 1 \]

\[ p \]

\[ u_2 : D \]

\[ x = 28 \]

\[ st = s_1 \]

\[ stb = 1 \]

\[ p \]

\[ u_2 : D \]

\[ x = 28 \]

\[ st = s_1 \]

\[ stb = 1 \]

\[ p \]
Example

$SM_C:\]

\[
\begin{align*}
{s_1} & \rightarrow E[n \neq \emptyset]/x := x + 1; n! F \rightarrow s_2 \\
{s_1} & \rightarrow F/x := 0 \rightarrow s_2 \\
{s_3} & \rightarrow /n := \emptyset \rightarrow s_2
\end{align*}
\]

$SM_D$:

\[
\begin{align*}
\langle \langle \text{signal} \rangle \rangle & \rightarrow (\sigma_1, \varepsilon_1) \xrightarrow{u_1} (\sigma_2, \varepsilon_2) \\
\langle \langle \text{signal} \rangle \rangle & \rightarrow (\sigma_1, \varepsilon_1) \xrightarrow{u_1} (\sigma_3, \varepsilon_3) \\
\langle \langle \text{signal} \rangle \rangle & \rightarrow (\sigma_4, \varepsilon_4) \xrightarrow{u_2} (\sigma_5, \varepsilon_5)
\end{align*}
\]

\[
\begin{align*}
\{E\}, \{F\} & \xrightarrow{u_1} (\sigma_2, \varepsilon_2) \\
\{E\}, \{F\} & \xrightarrow{u_1} (\sigma_3, \varepsilon_3) \\
\{E\}, \{F\} & \xrightarrow{u_2} (\sigma_5, \varepsilon_5)
\end{align*}
\]

$\sigma_1 = (C, x: \text{Int})$

$\sigma_2 = (\text{D}, x: \text{Int})$

$\sigma_3 = (\text{C}, x: \text{Int})$

$\sigma_4 = (\text{D}, x: \text{Int})$

$\sigma_5 = (\text{C}, x: \text{Int})$

$\varepsilon_1 = p$

$\varepsilon_2 = 0$

$\varepsilon_3 = 0$

$\varepsilon_4 = 0$

$\varepsilon_5 = 0$

$\sigma_1^{'} = (C, x: \text{Int})$

$\sigma_2^{'} = (\text{D}, x: \text{Int})$

$\sigma_3^{'} = (\text{C}, x: \text{Int})$

$\sigma_4^{'} = (\text{D}, x: \text{Int})$

$\sigma_5^{'} = (\text{C}, x: \text{Int})$

$\varepsilon_1^{'} = 0$

$\varepsilon_2^{'} = 0$

$\varepsilon_3^{'} = 0$

$\varepsilon_4^{'} = 0$

$\varepsilon_5^{'} = 0$
Ether
The order of dequeuing is not defined, leaving open the possibility of modeling different priority-based schemes.
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.
Definition. Let $\mathcal{I} = (\mathcal{T}, \mathcal{C}, V, atr, \mathcal{E})$ be a signature with signals and $\mathcal{D}$ a structure.

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

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

- a operation to **insert** an event for a given object, i.e.
  \[
  \oplus : Eth \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \to Eth
  \]

- a operation to **remove** an event, i.e.
  \[
  \ominus : Eth \times \mathcal{D}(\mathcal{E}) \to Eth
  \]

- an operation to **clear** the ether for a given object, i.e.
  \[
  [\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \to Eth.
  \]
Example: FIFO Queue

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

- \( Eth = (D(C) \times D(E))^* \)
  - e.g. \( \varepsilon = (u_1, e_3), (u_2, e_4) \)
  - the set of finite sequences of pairs \( (u, e) \in D(C) \times D(E) \)

- ready : \( Eth \times D(C) \to 2^{D(E)} \)
  - \( (\varepsilon, u, e) \mapsto \begin{cases} \{ (u, e) \} & \text{if } u_1 = u_2 \\ \emptyset & \text{otherwise} \end{cases} \) (also if \( \varepsilon \) is empty)

- \( \oplus : Eth \times D(C) \times D(E) \to Eth \)
  - \( (\varepsilon, u, e) \mapsto \varepsilon \cdot (u, e) \)

- \( \ominus : Eth \times D(E) \to Eth \)
  - \( (\varepsilon, e) \mapsto \varepsilon \ominus (u, e) \)
  - \( \varepsilon \ominus (u, e) = \begin{cases} \varepsilon & \text{if } e_2 = e_1 \\ \varepsilon \ominus (u, e), \varepsilon & \text{otherwise} (\text{also if empty}) \end{cases} \)

- \( [\cdot] : Eth \times D(C) \to Eth \)
  - remove all pairs \( (u, e) \) from \( \varepsilon \)
Other Examples

- One FIFO queue per active object is an ether.
  \[ E_{\mathcal{H}} = \mathcal{O}(\mathcal{E}) \rightarrow (\mathcal{O}(\mathcal{E}) \times \mathcal{O}(\mathcal{E}))^* \]

- One-place buffer.
  \[ E_{\mathcal{H}} = \varepsilon \cup (\mathcal{O}(\mathcal{E}) \times \mathcal{O}(\mathcal{E})) \]

- Priority queue.

- Multi-queues (one per sender).

- Trivial example: sink, “black hole”.

- Lossy queue (⊕ needs to become a relation then).

- . . .
System Configuration
**Definition.** Let $\mathcal{S}_0 = (\mathcal{T}_0, C_0, V_0, atr_0, \mathcal{E})$ be a signature with signals, $\mathcal{D}_0$ a structure of $\mathcal{S}_0$, $(Eth, ready, \oplus, \ominus, \lfloor \cdot \rfloor)$ an ether over $\mathcal{S}_0$ and $\mathcal{D}_0$.

Furthermore assume there is one core state machine $M_C$ per class $C \in \mathcal{C}$.

A **system configuration** over $\mathcal{S}_0$, $\mathcal{D}_0$, and $Eth$ is a pair $(\sigma, \varepsilon) \in \Sigma_{\mathcal{D}} \times Eth$ where

- $\mathcal{S} = (\mathcal{T}_0 \cup \{S_{MC} \mid C \in \mathcal{C}\}, C_0, V_0 \cup \{\langle stable : Bool, -, true, \emptyset \rangle\} \cup \{\langle st_C : S_{MC}, +, s_0, \emptyset \rangle \mid C \in \mathcal{C}\} \cup \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset \rangle \mid E \in \mathcal{E}_0\},

- $C \mapsto atr_0(C)$

- $\cup \{stable, st_C\} \cup \{params_E \mid E \in \mathcal{E}_0 \mid C \in \mathcal{C}\}, \mathcal{E}_0$)

- $\mathcal{D} = \mathcal{D}_0 \cup \{S_{MC} \mapsto S(M_C) \mid C \in \mathcal{C}\}$, and

- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0$. 
References
References
