Software Design, Modelling and Analysis in UML

Lecture 11: Core State Machines I

2016-12-08

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Content

- Recall: Basic Causality Model
- Event Pool
  - insert, remove, clear, ready.
- System Configuration
  - implicit attributes:
    - stable, st, and friends.
  - system state plus event pool
- Actions
  - simple action language.
  - transformer: effects of actions.
**Roadmap: Chronologically**

**Syntax:**
(i) UML State Machine Diagrams.
(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.
(xii) Later: Hierarchical state machines.

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.[...]

**15.3.12 StateMachine (OMG, 2011b, 574)**
Example

\[ C_x : Int \]
\[ D_s \]
\[ s_1 \rightarrow E[n \neq 0]; x := x + 1; F \]
\[ F_x := 0 \]
\[ n := 0 \]
\[ s_2 \rightarrow F \]
\[ p_1 F \]

SM_D:

\( \{E, F\} \)

\( \{F\} \)

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 $\mathcal{S} = (\mathcal{T}, \mathcal{C}, V, atr, \mathcal{E})$ 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 (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:

- $Eth = (\mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}))^*$

  the set of finite sequences of pairs $(u, e) \in \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E})$

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

  $\{ \varepsilon, u_2 \} \mapsto \begin{cases} \{(u_2, e)\}, & \varepsilon = (u_2, e), e' \\ \emptyset, & \text{otherwise} \end{cases}$

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

  $(\varepsilon, u, e) \mapsto \varepsilon, (u, e)$

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

  $(\varepsilon, e) \mapsto \begin{cases} \varepsilon', & \varepsilon = (u, e), e', u \in \mathcal{D}(\mathcal{C}) \\ \emptyset, & \text{otherwise} \end{cases}$

- $[\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \to Eth$

  $[\cdot](\varepsilon, u) : \text{remove all } (u, e) \text{ elements from the queue } \varepsilon, \forall e \in \mathcal{D}(\mathcal{C})$
Other Examples

- One FIFO queue per active object is an ether.
  \[ \mathcal{E}_\lambda = \mathcal{D}(C) \rightarrow (\mathcal{D}(e) \times \mathcal{D}(e))^* \]
- One-place buffer.
  \[ \mathcal{E}_\lambda = \varepsilon \cup (\mathcal{D}(e) \times \mathcal{D}(e)) \]
- Priority queue.
  ...
- Multi-queues (one per sender).
  ...
- Trivial example: sink, “black hole”.
  ...
- Lossy queue (⊕ needs to become a relation then).
  ...

System Configuration
Definition. Let $\mathcal{A}_0 = (\mathcal{F}_0, \mathcal{G}_0, V_0, \text{atr}_0, E_0)$ be a signature with signals, $\mathcal{D}_0$ a structure of $\mathcal{A}_0$, $(\text{Eth}, \text{ready}, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{A}_0$ and $\mathcal{D}_0$.

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

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

\[ S = (T_0 \cup \{S_{M_C} \mid C \in \mathcal{C}_0\}, \mathcal{C}_0, V_0 \cup \{\langle \text{stable}, \text{Bool}\rangle, -\text{true}, \emptyset\} \cup \{\langle \text{params}_E, E_0, 1, +, 0, \emptyset \rangle \mid E \in E_0\},\}

\[ \sigma(u)(r) \cap \mathcal{D}(E_0) = \emptyset \text{ for each } u \in \text{dom}(\sigma) \text{ and } r \in V_0.\]

System Configuration: Example

\[ \mathcal{A}_0 = (\mathcal{F}_0, \mathcal{G}_0, V_0, \text{atr}_0, E_0), \mathcal{D}_0: \ (\sigma, \varepsilon) \in \Sigma_{\mathcal{D}_0} \times \text{Eth} \text{ where} \]

\[ \mathcal{T} = (\mathcal{F}_0 \cup \{S_{M_C} \mid C \in \mathcal{C}_0\}, \mathcal{C}_0, V_0 \cup \{\langle \text{stable}, \text{Bool}\rangle, -\text{true}, \emptyset\} \cup \{\langle \text{params}_E, E_0, 1, +, 0, \emptyset \rangle \mid E \in E_0\},\}

\[ \mathcal{D} = \mathcal{D}_0 \cup \{S_{M_{C}}, \mathcal{D}(S_{M_{C}}) \mid C \in \mathcal{C}_0\}, \text{ and} \]

\[ \sigma(u)(r) \cap \mathcal{D}(E_0) = \emptyset \text{ for each } u \in \text{dom}(\sigma) \text{ and } r \in V_0.\]
System Configuration Step-by-Step

- We start with some signature with signals $\mathcal{R}_0 = (\mathcal{R}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$.

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

- Such a system state $\sigma$ wrt. $\mathcal{R}$ 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 $\text{params}^E$.

Stability

Definition. Let $(\sigma, \varepsilon)$ be a system configuration over some $\mathcal{R}_0, \mathcal{R}_0, \mathcal{E}_0$. We call an object $u \in \text{dom}(\sigma) \cap \mathcal{D}(\mathcal{C}_0)$ stable in $\sigma$ if and only if

$$\sigma(u)(\text{stable}) = 1$$

And unstable otherwise.
Where are we?

$$S\mathcal{M}_C :$$

\[
\begin{align*}
\sigma_1, \varepsilon_1 & \xrightarrow{\{E\}, \{F\}} \sigma_2, \varepsilon_2, \\
\sigma_3, \varepsilon_3 & \xrightarrow{\{F\}} \sigma_4, \varepsilon_4.
\end{align*}
\]

$$S\mathcal{M}_D :$$

$$\psi(x) \mapsto 1 + 1 \cdot \psi(x).$$

Transformer
Recall

- The (simplified) syntax of transition annotations:
  \[\text{annot} ::= [\langle \text{event} \rangle \left[ \top \langle \text{guard} \rangle \right] \left[ \top / \langle \text{action} \rangle \right]]\]

- **Clear**: \(\langle \text{event} \rangle\) is from \(\mathcal{E}\) of the corresponding signature.
- **But**: What are \(\langle \text{guard} \rangle\) and \(\langle \text{action} \rangle\)?
- UML can be viewed as being parameterized in expression language (providing \(\langle \text{guard} \rangle\)) and action language (providing \(\langle \text{action} \rangle\)).
- **Examples**:
  - Expression Language:
    - OCL
      - Java, C++, … expressions
    - ...
  - Action Language:
    - UML Action Semantics, "Executable UML"
    - Java, C++, … statements (plus some event send action)
    - ...

Needed: Semantics

\[
\text{OCL}: \quad I_{\text{Expr}}(\text{guard}):
\begin{cases}
1, & \text{if } I_{\text{Expr}}(\text{guard}) = \top \\
0, & \text{if } I_{\text{Expr}}(\text{guard}) = \bot \\
\text{undefined, otherwise}
\end{cases}
\]

In the following, we assume that we’re given
- an expression language \(\mathcal{Expr}\) for guards, and
- an action language \(\mathcal{Act}\) for actions,

and that we’re given
- a **semantics** for boolean expressions in form of a partial function
  \[
  \text{I}_{\mathcal{Expr}}(\cdot, \cdot) : \mathcal{Expr} \times \Sigma_{\mathcal{D}} \times \mathcal{D}(\mathcal{G}) \rightarrow \mathbb{B}
  \]
  which evaluates expressions in a given system configuration,

  **Assuming** \(I\) to be partial is a way to treat “undefined” during runtime. *If* \(I\) is not defined (for instance because of dangling-reference navigation or division-by-zero), we want to go to a designated “error” system configuration.

- a **transformer** for each action: for each \(\text{act} \in \mathcal{Act}\), we assume to have
  \[
  t_{\text{act}} \subseteq \mathcal{D}(\mathcal{G}) \times (\Sigma_{\mathcal{D}} \times \mathcal{Eth}) \times (\Sigma_{\mathcal{D}} \times \mathcal{Eth})
  \]
**Transformer**

**Definition.**
Let $\Sigma_\mathcal{D}S$ the set of system configurations over some $\mathcal{S}_0, \mathcal{D}_0, Eth$. We call a relation $t \subseteq (\mathcal{D}(\mathcal{E})) \times (\Sigma_\mathcal{D} \times Eth) \times (\Sigma_\mathcal{D} \times Eth)$ a (system configuration) transformer.

**Example:**
- $t[u_x](\sigma, \varepsilon) \subseteq \Sigma_\mathcal{D} \times Eth$ is
  - the set (!) of the system configurations
  - which may result from object $u_x$
  - executing transformer $t$.
- $t_{\text{skip}}[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}$
- $t_{\text{create}}[u_x](\sigma, \varepsilon) :$ add a previously non-alive object to $\sigma$ \{id, create, close, \}

**Observations**

- In the following, we assume that
  - each application of a transformer $t$
  - to some system configuration $(\sigma, \varepsilon)$
  - for object $u_x$
  is associated with a set of observations

$$Obs_t[u_x](\sigma, \varepsilon) \in 2^{(\mathcal{D}(\mathcal{E}) \cup \{*,+\}) \times \mathcal{D}(\mathcal{E})}.$$  

- An observation

$$(u_e, u_{dst}) \in Obs_t[u_x](\sigma, \varepsilon)$$

represents the information that,
a side effect of object $u_x$ executing $t$ in system configuration $(\sigma, \varepsilon)$,
the event $u_e$ has been sent to $u_{dst}$.

**Special cases:** creation ('*') / destruction ('+').
A Simple Action Language

In the following we use

\[ \text{Act}_\mathcal{S} = \{ \text{skip} \} \]

\[ \cup \{ \text{update}(expr_1, v, expr_2) \mid \text{expr}_1, \text{expr}_2 \in \text{Expr}_\mathcal{S}, v \in \text{atr} \} \]

\[ \cup \{ \text{send}(E(expr_1, \ldots, expr_n), expr_{\text{dst}}) \mid \text{expr}_1, \text{expr}_{\text{dst}} \in \text{Expr}_\mathcal{S}, E \in \mathcal{E} \} \]

\[ \cup \{ \text{create}(C, expr, v) \mid C \in \mathcal{C}, \text{expr} \in \text{Expr}_\mathcal{S}, v \in V \} \]

\[ \cup \{ \text{destroy}(expr) \mid \text{expr} \in \text{Expr}_\mathcal{S} \} \]

and OCL expressions over \( \mathcal{S} \) (with partial interpretation) as \( \text{Expr}_\mathcal{S} \).

Transformer Examples: Presentation

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td></td>
</tr>
<tr>
<td>intuitive semantics</td>
<td>...</td>
</tr>
<tr>
<td>well-typedness</td>
<td>...</td>
</tr>
<tr>
<td>semantics</td>
<td></td>
</tr>
<tr>
<td>[ (\sigma, \varepsilon), (\sigma', \varepsilon') \in t_{op}[u_x] \text{ iff } \ldots ]</td>
<td></td>
</tr>
<tr>
<td>[ t_{op}[u_x](\sigma, \varepsilon) = { (\sigma', \varepsilon') \mid \text{ where } \ldots } ]</td>
<td></td>
</tr>
<tr>
<td>observables</td>
<td>[ \text{Obs}_{op}[u_x] = { \ldots } ]</td>
</tr>
<tr>
<td>(error) conditions</td>
<td>Not defined if ...</td>
</tr>
</tbody>
</table>
### Transformer: Skip

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>skip</td>
</tr>
</tbody>
</table>

**intuitive semantics**

*do nothing*

**well-typedness**

/. .

**semantics**

\[ t_{\text{skip}}[u_2](\sigma, \varepsilon) = \{ (\sigma, \varepsilon) \} \]

**observables**

\[ \text{Obs}_{\text{skip}}[u_2](\sigma, \varepsilon) = \emptyset \]

**(error) conditions**

Not defined if \( I[\text{expr}_2](\sigma, u_2) \) or \( I[\text{expr}_2](\sigma, u_2) \) not defined.

### Transformer: Update

**abstract syntax**

\[ \text{update}(\text{expr}_1, v, \text{expr}_2) \]

**concrete syntax**

\[ \text{expr}_1 := \text{expr}_2 \]

**intuitive semantics**

*Update attribute* \( v \) *in the object denoted by* \( \text{expr}_1 \) *to the value denoted by* \( \text{expr}_2 \).

**well-typedness**

\( \text{expr}_1 : T_C \) *and* \( v : T \in \text{atr}(C) ; \) \( \text{expr}_2 : T ; \)

\( \text{expr}_1, \text{expr}_2 \) *obey visibility and navigability*

**semantics**

\[ t_{\text{update}}(\text{expr}_1, v, \text{expr}_2)[u_2](\sigma, \varepsilon) = \{ (\sigma', \varepsilon) \} \]

where \( \sigma' = \sigma[u \mapsto \sigma(u)[v \mapsto I[\text{expr}_2](\sigma, u_2)]] \) with

\[ u = I[\text{expr}_1](\sigma, u_2) \]

**observables**

\[ \text{Obs}_{\text{update}}(\text{expr}_1, v, \text{expr}_2)[u_2] = \emptyset \]

**(error) conditions**

Not defined if \( I[\text{expr}_2](\sigma, u_2) \) or \( I[\text{expr}_2](\sigma, u_2) \) not defined.
Update Transformer Example

\[ S.M.C: \quad \begin{array}{c}
  \text{expr}_1: x = 4 \\
  \text{expr}_2: y = 0
  \end{array} \]

\[ \text{update}(\text{expr}_1, v, \text{expr}_2)[u](\sigma, \epsilon) = (\sigma' = \sigma[u \mapsto \sigma(u) \mapsto \{ I[\text{expr}_1](\sigma, u) \}], u = I[\text{expr}_1](\sigma, u)) \]

Transformer: Send

**abstract syntax**

\[ \text{send}(E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{\text{dat}}) \]

**concrete syntax**

\[ \text{send}(E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{\text{dat}}) \]

**intuitive semantics**

Object \( u_x : C \) sends event \( E \) to object \( \text{expr}_{\text{dat}} \), i.e. create a fresh signal instance, fill in its attributes, and place it in the ether.

**well-typedness**

\( E \in \mathcal{E}; \ \text{atr}(E) = \{ v_1 : T_1, \ldots, v_n : T_n \}; \ \text{expr}_i : T_i, 1 \leq i \leq n; \ \text{expr}_{\text{dat}} : T_D, C, D \in \mathcal{C} \setminus \mathcal{E}; \)

all expressions obey visibility and navigability in \( C \)

**semantics**

\( (\sigma', \epsilon') \in t_{\text{send}}(E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{\text{dat}})[u_x](\sigma, \epsilon) \)

if \( \sigma' = \sigma \cup \{ u \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \}; \ \epsilon' = \epsilon \oplus (u_{\text{dat}}, u); \)

if \( u_{\text{dat}} = I[\text{expr}_{\text{dat}}](\sigma, u_x) \in \text{dom}(\sigma); \ d_i = I[\text{expr}_i](\sigma, u_x) \) for \( 1 \leq i \leq n; \)

\( u \in \mathcal{E}(E) \) a fresh identity, i.e. \( u \notin \text{dom}(\sigma) \),

and where \( (\sigma', \epsilon') = (\sigma, \epsilon) \) if \( u_{\text{dat}} \notin \text{dom}(\sigma) \).

**observables**

\[ \text{Obs}_{\text{send}}[u_x] = \{ (u_x, u_{\text{dat}}) \} \]

**error conditions**

\( I[\text{expr}](\sigma, u_x) \) not defined for any \( \text{expr} \in \{ \text{expr}_{\text{dat}}, \text{expr}_1, \ldots, \text{expr}_n \} \)
Send Transformer Example

**SM_C:**

\[
\text{Send}(\text{expr}_{src}, E(\text{expr}_1, \ldots, \text{expr}_n), \text{expr}_{dst})[u_x](\sigma, \epsilon) \ni (\sigma', \epsilon') \iff \epsilon' = \epsilon \oplus (u_{\text{dist}}, u);
\]

\[
\sigma' = \sigma \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\}; u_{\text{dist}} = I[expr_{\text{dist}}](\sigma, u_x) \in \text{dom}(\sigma);
\]

\[
d_i = I[expr_i](\sigma, u_x), 1 \leq i \leq n; u \in \mathcal{D}(E) \text{ a fresh identity};
\]

\[
s_1 \xrightarrow{\text{Send}} s_2
\]

\[
\text{Obs}(t_2 \circ t_1)[u_x](\sigma, \epsilon) = \text{Obs}_{t_1}[u_x](\sigma, \epsilon) \cup \text{Obs}_{t_2}[u_x](t_1(\sigma, \epsilon)).
\]

**Sequential Composition of Transformers**

- **Sequential composition** \( t_1 \circ t_2 \) of transformers \( t_1 \) and \( t_2 \) is canonically defined as

\[
(t_2 \circ t_1)[u_x](\sigma, \epsilon) = t_2[u_x](t_1[u_x](\sigma, \epsilon))
\]

with observation

\[
\text{Obs}_{t_2 \circ t_1}[u_x](\sigma, \epsilon) = \text{Obs}_{t_1}[u_x](\sigma, \epsilon) \cup \text{Obs}_{t_2}[u_x](t_1(\sigma, \epsilon)).
\]

- **Clear:** not defined if one the two intermediate "micro steps" is not defined.
Observation: our transformers are in principle the denotational semantics of the actions/action sequences. The trivial case, to be precise.

Note: with the previous examples, we can capture
- empty statements, skips,
- assignments,
- conditionals (by normalisation and auxiliary variables),
- create/destroy (later).

but not possibly diverging loops.

Our (Simple) Approach: if the action language is, e.g. Java, then (syntactically) forbid loops and calls of recursive functions.

Other Approach: use full blown denotational semantics.

No show-stopper, because loops in the action annotation can be converted into transition cycles in the state machine.

Tell Them What You’ve Told Them...

- A ether is an abstract representation of different possible “event pools” like
  - FIFO queues (shared, or per sender),
  - One-place buffers,
  - ...

- A system configuration consists of
  - an event pool (pending messages),
  - a system state over a signature with implicit attributes for
    - current state,
    - stability,
    - etc.

- Transitions are labelled with actions, the effect of actions is explained by transformers, transformers may modify system state and ether.
References
