Software Design, Modelling and Analysis in UML

Lecture 13: Core State Machines III

2014-12-16

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Contents & Goals

Last Lecture:
- Basic causality model
- Ether

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:**
  - System configuration
  - Transformer
  - Examples for transformer
System Configuration, Ether, Transformer
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 $(Eth, \text{ready}, \oplus, \ominus, [\cdot])$ 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.
  \[ \text{ready} : Eth \times \mathcal{D}(\mathcal{C}) \to 2^{\mathcal{D}(\mathcal{E})} \]

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

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

- A (single, global, shared, reliable) FIFO queue is an ether:
  - \( \text{Eth} = (\mathcal{D}(\mathcal{E}) \times \mathcal{D}(\mathcal{E}))^* \) e.g. \( \mathcal{E} = (\text{v}, \text{e}_1), (\text{v}, \text{f}_1), (\text{w}, \text{e}_2) \)
  - the set of all finite sequences of pairs \( (\text{u}, \text{e}) \in \mathcal{D}(\mathcal{E}) \times \mathcal{D}(\mathcal{E}) \)
  - \( \text{ready}(\text{v}, \text{e}_1) \cap \text{v} = \text{u} \text{ then } \text{ready}(\text{e}, \text{v}) = \emptyset \)
  - \( \Theta(\epsilon, \text{f}) = \epsilon \text{ empty seq.} \)
  - \( [\cdot] \): remove all \( (\text{u}, \text{e}) \) pairs from a given sequence

- One FIFO queue per active object is an ether.
- Lossy queue (\( \oplus \) becomes a relation then).
- One-place buffer.
- Priority queue.
- Multi-queues (one per sender).
- Trivial example: sink, “black hole”.
- ...
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]

The standard distinguishes, e.g., **SignalEvent** [OMG, 2007b, 450], **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. **Often seen minimal requirement**: order of sending by one object is preserved. But: we’ll later briefly discuss “discarding” of events.
Definition. Let $\mathcal{D}_0$ be a structure of the signature with signals $\mathcal{I}_0 = (\mathcal{T}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ and let $E \in \mathcal{E}_0$ be a signal.

Let $atr(E) = \{v_1, \ldots, v_n\}$. We call

$$e = (E, \{v_1 \mapsto d_1, \ldots, v_n \mapsto d_n\}),$$

or shorter (if mapping is clear from context)

$$(E, (d_1, \ldots, d_n)) \text{ or } (E, \vec{d}),$$

an event (or an instance) of signal $E$ (if type-consistent).

We use $Evs(\mathcal{E}_0, \mathcal{D}_0)$ to denote the set of all events of all signals in $\mathcal{I}_0$ wrt. $\mathcal{D}_0$.

As we always try to maximize confusion...:

- By our existing naming convention, $u \in \mathcal{D}(E)$ is also called instance of the (signal) class $E$ in system configuration $(\sigma, \varepsilon)$ if $u \in \text{dom}(\sigma)$.
- The corresponding event is then $(E, \sigma(u))$. 

\[
\begin{align*}
\mathcal{C} & \quad \langle \text{signal} \rangle \quad E \\
& \quad x : \text{int} \quad a : \text{bool} \\
\end{align*}
\]

\[
\begin{align*}
\sigma : & \quad \{ v : \text{C} \} \\
2 \exists \xi : E & \quad x = 13 \\
& \quad a = \text{true} \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{M}_i : & \quad E \quad [\text{produce} \quad \exists x \geq 0] \\
& \quad (v, 2 \exists \xi) \\
\end{align*}
\]

\[
\begin{align*}
\sigma, \xi & \quad \text{do get identity} \\
\{ 2 \exists \xi : E, x = 13, a = \text{true} \} & \quad \{ \ldots \} \\
\end{align*}
\]

\[
\begin{align*}
(\sigma, \xi) & \quad \xrightarrow{u} (\sigma', \xi') \\
\text{which object} & \quad \text{takes trans. edge} \\
\end{align*}
\]
Signals? Events...? Ether...?!

The idea is the following:

- **Signals** are types (classes).

- **Instances of signals** (in the standard sense) are kept in the system state component \( \sigma \) of system configurations \((\sigma, \varepsilon)\).

- **Identities** of signal instances are kept in the ether.

- Each signal instance is in particular an event — somehow “a recording that this signal occurred” (without caring for its identity).

- The main difference between **signal instance** and **event**:

  Events don’t have an identity.

- Why is this useful? In particular for reflective descriptions of behaviour, we are typically not interested in the identity of a signal instance, but only whether it is an “E” or “F”, and which parameters it carries.
**System Configuration**

**Definition.** Let $S_0 = (T_0, C_0, V_0, atr_0, E)$ be a signature with signals, $D_0$ a structure of $S_0$, $(Eth, ready, \oplus, \ominus, [\cdot])$ an ether over $S_0$ and $D_0$. Furthermore assume there is one core state machine $M_C$ per class $C \in C$.

A system configuration over $S_0$, $D_0$, and $Eth$ is a pair $(\sigma, \varepsilon) \in \Sigma_D \times Eth$

where

- $S = (T_0 \cup \{S_{MC} \mid C \in C\}, C_0,$
- $V_0 \cup \{\langle stable : Bool, -, true, \emptyset \rangle\}$
- $\cup \{\langle st_C : S_{MC}, +, s_0, \emptyset \rangle \mid C \in C\}$
- $\cup \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset \rangle \mid E \in E\}$,
- $\{C \mapsto atr_0(C)$$
- \cup \{stable, st_C\} \cup \{params_E \mid E \in E\} \mid C \in C\}$, 
- $E_0)$
- $D = D_0 \cup \{S_{MC} \mapsto S(M_C) \mid C \in C\}$, and
- $\sigma(u)(r) \cap D(E) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0$. 

Additional notes:

- A new type for each class.
- If Bool & $T_0$ then add it and have $D(\text{Bool}) = \text{B}$.
- Initial state of state machine $S_{MC}$.
- Initial state of state machine $MC$.
- Set of states of state machine of class $C$.
- The only links to $S_{MC}$ instances are via params.
System Configuration: Example

\[ S_0 = (T_0, C_0, V_0, atr_0, \mathcal{C}), D_0; \quad (\sigma, \varepsilon) \in \Sigma_\mathcal{G} \times Eth \text{ where} \]

- \( S = (T_0 \cup \{S_{MC} \mid C \in \mathcal{C}\}, C_0, V_0 \cup \{\langle stable : \text{Bool}, -, \text{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\}
\]

- \( D = D_0 \cup \{S_{MC} \mapsto S(M_C) \mid C \in \mathcal{C}\}, \text{ and} \)

- \( \sigma(u)(r) \cap D(\mathcal{E}_0) = \emptyset \) for each \( u \in \text{dom}(\sigma) \) and \( r \in V_0. \)
We start with some signature with signals $\mathcal{I}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$.

A **system configuration** is a pair $(\sigma, \varepsilon)$ which comprises a system state $\sigma$ wrt. $\mathcal{I}$ (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. $\sigma(u)(st)$ 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 $\text{params}_E$. 
**Definition.**
Let $(\sigma, \varepsilon)$ be a system configuration over some $I_0$, $D_0$, $Eth$.

We call an object $u \in \text{dom}(\sigma) \cap D(C_0)$ **stable in $\sigma$** if and only if

$$\sigma(u)(\text{stable}) = \text{true}.$$
Where are we?

- **Wanted**: a labelled transition relation

\[
(\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})}_{u_x} (\sigma', \varepsilon')
\]

on system configuration, labelled with the **consumed** and **sent** events, 

\((\sigma', \varepsilon')\) being the result (or effect) of **one object** \(u_x\) taking a transition of **its** state machine from the current state machine state \(\sigma(u_x)(st_C)\).

- **Have**: system configuration \((\sigma, \varepsilon)\) comprising current state machine state and stability flag for each object, and the ether.

- **Plan**:
  
  (i) Introduce **transformer** as the semantics of action annotations. **Intuitively**, \((\sigma', \varepsilon')\) is the effect of applying the transformer of the taken transition.

  (ii) Explain how to choose transitions depending on \(\varepsilon\) and when to stop taking transitions — the **run-to-completion “algorithm”**.
Why Transformers?

- **Recall** the (simplified) syntax of transition annotations:

  \[
  \text{annot ::= [ \langle \text{event} \rangle [ \[' \langle \text{guard} \rangle \'] ] \['/\] \langle \text{action} \rangle ] ]}
  \]

- **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)
      - ...
Definition.
Let $\Sigma^D$ the set of system configurations over some $I_0$, $D_0$, $Eth$.
We call a relation

$$t \subseteq D(C) \times (\Sigma^D \times Eth) \times (\Sigma^D \times Eth)$$

a (system configuration) transformer.

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

- An observation $(u_{src}, u_e, (E, \vec{d}), u_{dst}) \in Obs_t[u_x](\sigma, \varepsilon)$ represents the information that, as a “side effect” of $u_x$ executing $t$, an event (!) $(E, \vec{d})$ has been sent from $u_{src}$ to $u_{dst}$.

**Special cases:** creation/destruction.
Transformers as Abstract Actions!

In the following, we assume that we’re given

- an expression language $Expr$ for guards, and
- an action language $Act$ for actions,

and that we’re given

- a semantics for boolean expressions in form of a partial function

\[ I[\cdot](\cdot, \cdot) : Expr \rightarrow (\Sigma_F \times D(C) \rightarrow \mathbb{B}) \]

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 $act \in Act$, we assume to have

\[ t_{act} \subseteq D(C) \times (\Sigma_F \times Eth) \times (\Sigma_F \times Eth) \]
We can make the assumptions from the previous slide because **instances exist**:

- for OCL, we have the OCL semantics from Lecture 03. Simply remove the pre-images which map to “⊥”.
- for Java, the operational semantics of the SWT lecture uniquely defines transformers for sequences of Java statements.

We distinguish the following kinds of transformers:

- **skip**: do nothing — recall: this is the default action
- **send**: modifies $\varepsilon$ — interesting, because state machines are built around sending/consuming events
- **create/destroy**: modify domain of $\sigma$ — not specific to state machines, but let’s discuss them here as we’re at it
- **update**: modify own or other objects’ local state — boring
A Simple Action Language

In the following we use

\[
\text{Act}_y := \{ \text{skip} \}
\]

\[
\cup \{ \text{update}(\text{exp}_1, v, \text{exp}_2) \mid \text{exp}_1, \text{exp}_2 \in \text{OCLExp}, v \in V \}
\]

\[
\cup \{ \text{send}(\text{exp}_1, E, \text{exp}_2) \mid \text{exp}_1, \text{exp}_2 \in \text{OCLExp}, E \in E \}
\]

\[
\cup \{ \text{create}(C, \text{exp}_1, v) \mid C \in C \setminus E, \text{exp}_1 \in \text{OCLExp}, v \in V \}
\]

\[
\cup \{ \text{destroy}(\text{exp}) \mid \text{exp} \in \text{OCLExp} \}
\]

\[
\text{Exp}_y : \text{OCL expressions}
\]

if \((\text{new } C \neq \text{NULL})\) ...

\[
v := \text{new } C;
\]

if \((v \neq \text{NULL})\) ...
### Transformer Examples: Presentation

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{op} )</td>
<td>( \text{op} )</td>
</tr>
</tbody>
</table>

**Intuitive Semantics**

... 

**Well-typedness**

... 

**Semantics**

\[
((σ, ε), (σ', ε')) \in t_{\text{op}}[u_x] \text{ iff } ...
\]

or

\[
t_{\text{op}}[u_x](σ, ε) = \{(σ', ε')\} \text{ where } ...
\]

**Observables**

\[
\text{Obs}_{\text{op}}[u_x] = \{\ldots\}, \text{ not a relation, depends on choice}
\]

**Error Conditions**

Not defined if ...
## Transformer: Skip

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

### Intuitive Semantics

*do nothing*

### Well-typedness

./.

### Semantics

\[
t[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}
\]

### Observables

\[
Obs_{\text{skip}}[u_x](\sigma, \varepsilon) = \emptyset
\]

### (Error) Conditions

...
# Transformer: Update

**abstract syntax**

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

**concrete syntax**

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

**intuitive semantics**

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

**well-typedness**

\[
\text{expr}_1 : \tau_C \quad \text{and} \quad v : \tau \in \text{atr}(C) ; \quad \text{expr}_2 : \tau ;
\]

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

**semantics**

\[
\begin{align*}
\text{t}_{\text{update}}(\text{expr}_1, v, \text{expr}_2)[u_x](\sigma, \varepsilon) &= \{ (\sigma', \varepsilon) \} \\
\text{where} \quad \sigma' &= \sigma[u \mapsto \sigma(u)[v \mapsto I[\text{expr}_2](\sigma, v_x)]] \quad \text{with} \\
& \quad u = I[\text{expr}_1](\sigma, v_x) \quad \text{(object denoted by expr1 (refrigerator to } v_x)\text{)}
\end{align*}
\]

**observables**

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

**error conditions**

Not defined if \(I[\text{expr}_1](\sigma, v_x)\) or \(I[\text{expr}_2](\sigma, v_x)\) not defined.

---

- Change local state of object \(u\)
References
