Software Design, Modelling and Analysis in UML

Lecture 14: Hierarchical State Machines I

2016-12-22

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
• **Missing Pieces**: Create and Destroy Transformers

• **Putting It All Together** (Again)
  • Initial States
  • Consistency wrt. OCL Constraints

• **Hierarchical State Machines**
  • Overview
  • **Abstract Syntax**: States
    • pseudo-states, regions, …
  • **(Legal) System Configurations**
  • **Abstract Syntax**: Transitions
  • Enabledness of Fork/Join Transitions
    • scope, priority, maximality, …
Putting It All Together
Initial States

Recall: a labelled transition system is \((S, A, \rightarrow, S_0)\).

We have

- \(S\): system configurations \((\sigma, \varepsilon)\)
- \(\rightarrow\): labelled transition relation \((\sigma, \varepsilon) \xrightarrow{(\text{cons}, \text{Snd})_u} (\sigma', \varepsilon')\).

Wanted: initial states \(S_0\).

Proposal:
Require a (finite) set of object diagrams \(\mathcal{OD}\) as part of a UML model

\((\mathcal{CD}, \mathcal{SM}, \mathcal{OD})\).

And set

\[S_0 = \{ (\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \ \mathcal{OD} \in \mathcal{OD}, \ \varepsilon \text{ empty} \}\].

Other Approach: (used by Rhapsody tool) multiplicity of classes (plus initialisation code).
We can read that as an abbreviation for an object diagram.
The **semantics** of the **UML model**

\[ \mathcal{M} = (\mathcal{CD}, \mathcal{SM}, \mathcal{OD}) \]

where

- some classes in \( \mathcal{CD} \) are stereotyped as ‘signal’ (standard), some signals and attributes are stereotyped as ‘external’ (non-standard),
- there is a 1-to-1 relation between classes and state machines,
- \( \mathcal{OD} \) is a set of object diagrams over \( \mathcal{CD} \),

is the **transition system** \((S, A, \rightarrow, S_0)\) constructed on the previous slide(s).

The **computations of** \( \mathcal{M} \) are the computations of \((S, A, \rightarrow, S_0)\).
Let \( M = (CD, SM, OD) \) be a UML model.

We call \( M \) consistent iff, for each OCL constraint \( expr \in Inv(CD) \),

\[ \sigma \models expr \]

for each “reasonable point” \( (\sigma, \varepsilon) \) of computations of \( M \).

(Cf. tutorial for discussion of “reasonable point”.)

Note: we could define \( Inv(SM) \) similar to \( Inv(CD) \).
Last Missing Piece: Create and Destroy Transformer
abstract syntax

create(C, expr, v)

can be written as

tmp1 := new C;
tmp2 := new D;
x = tmp1.y + tmp2.z;

concrete syntax

expr.v := new C

intuitive semantics

Create an object of class C and assign it to attribute v of the object denoted by expression expr.

well-typedness

\[ \text{expr} : T_D, v \in atr(D), \]
\[ atr(C) = \{ \langle v^i : T^i, expr^0_i \rangle | 1 \leq i \leq n \} \]

semantics

... 

observables

... 

(error) conditions

\[ I [expr](\sigma, \beta) \text{ not defined.} \]
## Transformer: Create

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create(C, expr, v)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Intuitive Semantics

*Create an object of class* $C$ *and assign it to attribute* $v$ *of the object denoted by expression* $expr$.

### Well-typedness

\[
expr : T_D, \quad v \in atr(D), \quad atr(C) = \{ \langle v_1 : T_1, expr_0^1 \rangle \mid 1 \leq i \leq n \}
\]

### Semantics

...  

### Observables

...  

### (Error) Conditions

\[
I[[expr]](\sigma, \beta) \text{ not defined.}
\]

- We use an “and assign”-action for simplicity – it doesn’t add or remove expressive power, but moving creation to the expression language raises all kinds of other problems since then expressions would need to modify the system state.
- Also for simplicity: no parameters to construction (\sim parameters of constructor). Adding them is straightforward (but somewhat tedious).
How To Choose New Identities?

- **Re-use**: choose any identity that is not alive **now**, i.e. not in \( \text{dom}(\sigma) \).
  - Doesn’t depend on history.
  - May “undangle” dangling references – may happen on some platforms.

- **Fresh**: choose any identity that has not been alive **ever**, i.e. not in \( \text{dom}(\sigma) \) and any predecessor in current run.
  - Depends on history.
  - Dangling references remain dangling – could mask “dirty” effects of platform.
abstract syntax | concrete syntax
---|---
create$(C, expr, v)$

intuitive semantics

*Create an object of class* $C$ *and assign it to attribute* $v$ *of the object denoted by expression* $expr$.

well-typedness

$$expr : T_D, v \in atr(D),$$
$$atr(C) = \{ \langle v_i : T_1, expr^0_i \rangle \mid 1 \leq i \leq n \}$$

semantics

$$(\langle \sigma, \varepsilon \rangle, (\sigma', \varepsilon')) \in t_{create}(C, expr, v)[u_x] \iff$$

$$\sigma' = \sigma[u_0 \mapsto \sigma(u_0)[v \mapsto u]] \cup \{ u \mapsto \{ v_i \mapsto d_i \mid 1 \leq i \leq n \} \},$$
$$\varepsilon' = [u](\varepsilon); \quad u \in D(C) \fresh, \text{i.e.} \quad u \notin \text{dom}(\sigma);$$
$$u_0 = I[expr](\sigma, u_x); \quad d_i = I[expr^0_i](\sigma, \emptyset) \text{ if } expr^0_i \neq \mathbb{W} \text{ and arbitrary value from } D(T_i) \text{ otherwise.}$$

observables

$\text{Obs}_{create}[u_x] = \{ (*, u) \}$

(error) conditions

$$(\text{error}) \quad I[expr_i^0](\sigma, u_x) \text{ not defined.}$$
Create Transformer Example

\( S_M_D: \)

\(/n := \text{new } C\)

\( s_1 \rightarrow s_2 \)

\[\text{create}(C, expr, v)\]

\( t_{\text{create}(C, expr, v)}[u_x](\sigma, \varepsilon) = \ldots\)

\( 1_C : C \)

\( \sigma:\)

\( d : D \)

\( n = \emptyset \)

\( st_D = s_1 \)

\( \text{stable} = 0 \)

\( \varepsilon:\)

\( \text{init\' \ sketch\ of } S_{M_C} \)

\( :\sigma' \)

\( :\varepsilon' \)
<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>destroy(expr)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Intuitive Semantics**

*Destroy the object denoted by expression* `expr`.

**Well-Typedness**

`expr : T_C, C \in \mathcal{C}`

**Semantics**

... 

**Observables**

\[ \text{Obs}_{\text{destroy}}[u_x] = \{(u_x, \bot, (+, \emptyset), u)\} \]

**(Error) Conditions**

\[ I[expr](\sigma, \beta) \text{ not defined.} \]
What to Do With the Remaining Objects?

Assume object $u_0$ is destroyed…

- object $u_1$ may still refer to it via association $r$:
  - allow dangling references?
  - or remove $u_0$ from $\sigma(u_1)(r)$?

- object $u_0$ may have been the last one linking to object $u_2$:
  - leave $u_2$ alone?
  - or remove $u_2$ also? (garbage collection)

- Plus: (temporal extensions of) OCL may have dangling references.

**Our choice**: Dangling references and no garbage collection!

This is in line with “expect the worst”, because there are target platforms which don’t provide garbage collection – and models shall (in general) be correct without assumptions on target platform.

**But**: the more “dirty” effects we see in the model, the more expensive it often is to analyse. Valid proposal for simple analysis: monotone frame semantics, no destruction at all.
### Transformer: Destroy

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>destroy(expr)</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Intuitive Semantics

*Destroy the object denoted by expression* $expr$.

#### Well-Typedness

$expr : T_C, C \in \mathcal{C}$

#### Semantics

$$t_{\text{destroy}}(expr)[u_x](\sigma, \varepsilon) = \{ (\sigma', \varepsilon') \} \quad \varepsilon' = [u] \cup \varepsilon$$

where $\sigma' = \sigma|_{\text{dom}(\sigma)\setminus\{u\}}$ with $u = I[expr](\sigma, u_x)$.

#### Observables

$$\text{Obs}_{\text{destroy}}(expr)[u_x] = \{ (+, u) \}$$

#### (Error) Conditions

$I[expr](\sigma, u_x)$ not defined.
\[ \text{destroy}(\text{expr}) \]
\[ t_{\text{destroy}(\text{expr})[u_x]}(\sigma, \varepsilon) = \ldots \]
Hierarchical State-Machines
UML distinguishes the following kinds of states:

- **simple state**
  - entry
  - do
  - exit
  - $E_1$/act
  - ...
  - $E_n$/act

- **final state**

- **composite state**
  - OR
  - AND

- **pseudo-state**
  - initial
  - (shallow) history
  - deep history
  - fork/join

- **submachine state**
  - junction, choice
  - entry point
  - exit point
  - terminate

- **terminate**
Blessing or Curse...?
Plan:

States / Syntax:
• What is the abstract syntax of a diagram?

States / Semantics:
• what is the type of the implicit $st$ attribute?
• what are legal system configurations?

Transitions / Syntax:
• what are legal / well-formed transitions?

Transitions / Semantics:
• when is a legal transition enabled?
• which effects do transitions have?

For example: From $s_1, s_5$,
• what may happen on $E$?
• what may happen on $E, F$?
• can $E, G$ kill the object?
• ...

\[true\]
So far:

\[(S, s_0, \rightarrow), \quad s_0 \in S, \quad \rightarrow \subseteq S \times (\mathcal{E} \cup \{\_\}) \times \text{Expr}_\mathcal{G} \times \text{Act}_\mathcal{G} \times S\]
Representing All Kinds of States

- So far:

\[(S, s_0, \rightarrow), \quad s_0 \in S, \quad \rightarrow \subseteq S \times (\mathcal{E} \cup \{\_\}) \times \text{Expr}_\mathcal{G} \times \text{Act}_\mathcal{G} \times S\]

- From now on: (hierarchical) state machines

\[(S, \text{kind}, \text{region}, \rightarrow, \psi, \text{annot})\]

where

- \(S \supseteq \{\text{top}\}\) is a finite set of states \((\text{new: top})\),
- \(\text{kind}: S \rightarrow \{\text{st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term}\}\) is a function which labels states with their \text{kind} \((\text{new})\),
- \(\text{region}: S \rightarrow 2^{2^S}\) is a function which characterises the \text{regions} of a state \((\text{new})\),
- \(\rightarrow\) is a set of transitions \((\text{changed})\),
- \(\psi: (\rightarrow) \rightarrow 2^S \times 2^S\) is an \text{incidence function} \((\text{new})\), and
- \(\text{annot}: (\rightarrow) \rightarrow (\mathcal{E} \cup \{\_\}) \times \text{Expr}_\mathcal{G} \times \text{Act}_\mathcal{G}\) provides an annotation for each transition \((\text{new})\).

\(s_0\) is then redundant – replaced by proper state (!) of kind ‘init’.)
## Well-Formedness: Regions

<table>
<thead>
<tr>
<th></th>
<th>$\in S$</th>
<th>$\text{kind}$</th>
<th>$\text{region} \subseteq 2^S, S_i \subseteq S$</th>
<th>$\text{child} \subseteq S$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>final state</strong></td>
<td>$s$</td>
<td>$\text{fin}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>pseudo-state</strong></td>
<td>$s$</td>
<td>$\text{init, ...}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>simple state</strong></td>
<td>$s$</td>
<td>$\text{st}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td><strong>composite state</strong></td>
<td>$s$</td>
<td>$\text{st}$</td>
<td>${S_1, \ldots, S_n}, n \geq 1$</td>
<td>$S_1 \cup \cdots \cup S_n$</td>
</tr>
<tr>
<td><strong>implicit top state</strong></td>
<td>$\text{top}$</td>
<td>$\text{st}$</td>
<td>${S_1}$</td>
<td>$S_1$</td>
</tr>
</tbody>
</table>

- Final and pseudo states **must not comprise** regions.
- States $s \in S$ with $\text{kind}(s) = \text{st}$ **may comprise** regions.

Naming conventions can be defined based on regions:

- No region: simple state.
- One region: OR-state.
- Two or more regions: AND-state.

- Each state (except for $\text{top}$) **must** lie in exactly one region.

- **Note:** The region function induces a child function.
- **Note:** Diagramming tools (like Rhapsody) can ensure well-formedness.
### From UML to Hierarchical State Machine: By Example

\[(S, \text{kind}, \text{region}, \rightarrow, \psi, \text{annot})\]

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>(\in S)</th>
<th>kind</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple state</td>
<td><img src="simple_state.png" alt="" /></td>
<td>(s)</td>
<td>(s)</td>
<td>(st)</td>
</tr>
<tr>
<td>final state</td>
<td><img src="final_state.png" alt="" /></td>
<td>(q)</td>
<td>(q)</td>
<td>(fin)</td>
</tr>
<tr>
<td>composite state</td>
<td><img src="composite_state.png" alt="" /></td>
<td>(s)</td>
<td>(s)</td>
<td>(st)</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td></td>
<td></td>
<td></td>
<td>{ {s_1, s_2, s_3} }</td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td></td>
<td></td>
<td></td>
<td>{ {s_1, s'_1}, {s_2, s'_2}, {s_3, s'_3} }</td>
</tr>
<tr>
<td>submachine state</td>
<td>(later)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pseudo-state</td>
<td><img src="pseudo_state.png" alt="" /></td>
<td>(q)</td>
<td>(q)</td>
<td>(init)</td>
</tr>
</tbody>
</table>

\((s, \text{kind}(s))\) for short
... denotes \((S, \text{kind}, \text{region}, \to, \psi, \text{annot})\) with

- \(S = \{\text{top}, s_1, s, s_2\}\)
- \(\text{kind} = \{\text{top} \mapsto \text{st}, s_1 \mapsto \text{init}, s \mapsto \text{st}, s_2 \mapsto \text{fin}\}\)
- \(\text{or } (S, \text{kind}) = \{(\text{top}, \text{st}), (s_1, \text{init}), (s, \text{st}), (s_2, \text{fin})\}\)
- \(\text{region} = \{\text{top} \mapsto \{\{s_1, s, s_2\}\}, s_1 \mapsto \emptyset, s \mapsto \emptyset, s_2 \mapsto \emptyset\}\)
- \(\to, \psi, \text{annot}: \text{in a minute.}\)
**Plan:**

**States / Syntax:**
- What is the abstract syntax of a diagram? ✓

**States / Semantics:**
- what is the type of the implicit $st$ attribute?
- what are legal system configurations?

**Transitions / Syntax:**
- what are legal well-formed transitions?

**Transitions / Semantics:**
- when is a legal transition enabled?
- which effects do transitions have?

For example: From $s_1$, $s_5$,
- what may happen on $E$?
- what may happen on $E$, $F$?
- can $E$, $G$ kill the object?
- ...

For example: From $s_1$, $s_5$, $s_7$, $s_8$,
Semantics: State Configuration

- The type of (implicit attribute) \( st \) is from now on a set of states, i.e. \( \mathcal{D}(S_{MC}) = 2^S \)

- A set \( S_1 \subseteq S \) is called (legal) state configuration if and only if
  - \( \top \in S_1 \), and
  - for each region \( R \) of a state in \( S_1 \), exactly one (non pseudo-state) element of \( R \) is in \( S_1 \), i.e.
    \[
    \forall s \in S_1 \forall R \in \text{region}(s) \bullet |\{s \in R | \text{kind}(s) \in \{st, fin\}\} \cap S_1| = 1.
    \]

- **Examples:**

  \[
  \begin{align*}
  S_0 &= \{s_0\} \times \\
  S_2 &= \{s_0, \top\} \checkmark \\
  S_3 &= \{s_1, \top\} \times \\
  S_4 &= \{s_1, \top, s_3, s_4\} \times \\
  S_5 &= \{s_1, \top, s_4\} \checkmark \\
  S_6 &= \{s_5, \top, s_6, s_9\} \times \\
  S_7 &= \{s_5, \top, s_6, s_9, s_{10}\} \checkmark \\
  S_8 &= \{s_6, s_0, s_1, s_2\} \times
  \end{align*}
  \]
Recall

Plan:

States / Syntax:
- What is the abstract syntax of a diagram?

States / Semantics:
- what is the type of the implicit *st* attribute?
- what are legal system configurations?

Transitions / Syntax:
- what are legal well-formed transitions?

Transitions / Semantics:
- when is a legal transition enabled?
- which effects do transitions have?

For example: From \(s_1, s_5\),
- what may happen on \(E\)?
- what may happen on \(E, F\)?
- can \(E, G\) kill the object?
- ...

Diagram:
- \(s_1\) to \(E/\)
- \(s_2\)
- \(s_3\) to \(F/\)
- \(s_4\) to \(F/\)
- \(s_5\) to \(E/\)
- \(s_6\) to \(G/\)
- \([true]/\) to \(F/\)
- \(s_7\)
For simplicity, we consider transitions with (possibly) multiple sources and targets, i.e.

\[ \psi : (\to) \to (2^S \setminus \emptyset) \times (2^S \setminus \emptyset) \]

For instance,

\[ \begin{array}{c}
    s_1 \\
    s_2 \\
    s_3 \\
    tr[gd]/act \\
    s_4 \\
    s_5 \\
    s_6
\end{array} \]

translates to

\[ (S, \text{kind}, \text{region}, \{t_1\}, \{t_1 \mapsto (\{s_2, s_3\}, \{s_5, s_6\})\}, \{t_1 \mapsto (\text{tr}, \text{gd}, \text{act})\}) \]

Naming convention: \( \psi(t) = (\text{source}(t), \text{target}(t)) \).
Orthogonal States

- Two states $s_1, s_2 \in S$ are called orthogonal, denoted $s_1 \perp s_2$, if and only if
- they “live” in different regions of one AND-state, i.e.

$$\exists s, \text{region}(s) = \{S_1, \ldots, S_n\}, 1 \leq i \neq j \leq n : s_1 \in \text{child}(S_i) \land s_2 \in \text{child}(S_j),$$
A hierarchical state-machine \((S, \text{kind}, \text{region}, \rightarrow, \psi, \text{annot})\) is called **well-formed** if and only if for all transitions \(t \in \rightarrow\),

- source (and destination) states are pairwise orthogonal, i.e.
  \[ \forall s, s' \in \text{source}(t) (\in \text{target}(t)) \bullet s \perp s', \]
- the top state is neither source nor destination, i.e.
  \[ \text{top} \notin \text{source}(t) \cup \text{source}(t). \]

**Recall:** final states are not sources of transitions.

**Example:**

![Diagram](image-url)
### Plan

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
</table>
| Simple state        | ![Diagram](https://via.placeholder.com/150) | - Entry/act<sub>1</sub>  
- Do/act<sub>do</sub>  
- Exit/act<sub>exit</sub>  
- E<sub>1</sub>/act<sub>E<sub>1</sub></sub>  
- ...  
- E<sub>n</sub>/act<sub>E<sub>n</sub></sub> |
| Final state         | ![Diagram](https://via.placeholder.com/150) | ![Diagram](https://via.placeholder.com/150) |
| Composite state     | ![Diagram](https://via.placeholder.com/150) | ![Diagram](https://via.placeholder.com/150) |
| OR                  | ![Diagram](https://via.placeholder.com/150) | ![Diagram](https://via.placeholder.com/150) |
| AND                 | ![Diagram](https://via.placeholder.com/150) | ![Diagram](https://via.placeholder.com/150) |

- Transitions involving non-pseudo states.
- Initial pseudostate, final state.
- Entry/do/exit actions, internal transitions.
- History and other pseudostates, the rest.
Tell Them What You’ve Told Them…

- For the **Create Action**, we have two main choices:
  - **re-use** identities (“nasty semantics”),
  - **use fresh** identities (“clean semantics”, depends on history).

Similar for **Destroy**.

- **Hierarchical State Machines** introduce **Regions**.
  - Thereby, **states** can lie within **states** as **children**.
  - The implicit variable $st$ becomes set-valued.

- **Transitions** may now have
  - **multiple** source states, **multiple** destination states,
  - but need to adhere to **well-formedness conditions**.

- **Enabledness** of a set (!) of transitions
  is **a bit tricky to define** (→ scope, priority, maximality).

- **Steps** are a proper generalisation of core state machines.
References
References
