Software Design, Modelling and Analysis in UML

Lecture 21: Model-based Software Development

2017-02-06

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Content

- Live Sequence Charts
  - Semantics
    - Full LSCs
      - Existential and Universal
      - Pre-Charts
      - Forbidden Scenarios
    - LSCs and Tests

- Model-Based-/Driven Software Engineering
  - Model Element Coverage of test cases
  - Model-based Testing
Full LSCs

A full LSC $\mathcal{L} = \left( (L, \preceq, \sim), \mathcal{I}, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta \right), ac_0, am, \Theta_{\mathcal{L}} \right)$ consists of

- **body** $\left( (L, \preceq, \sim), \mathcal{I}, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta \right)$,
- **activation condition** $ac_0 \in \text{Expr}_{\mathcal{L}}$,
- **strictness flag** $\text{strict}$ (if $\text{false}$, $\mathcal{L}$ is called **permissive**)
- **activation mode** $am \in \{\text{initial, invariant}\}$,
- **chart mode** $\text{existential} (\Theta_{\mathcal{L}} = \text{cold})$ or $\text{universal} (\Theta_{\mathcal{L}} = \text{hot})$.

Concrete syntax:
A full LSC \( \mathcal{L} = ((L, \preceq, \sim), I, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta), ac_0, \text{am}, \theta) \) consists of

- **body** \((L, \preceq, \sim), I, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta)\).
- **activation condition** \(ac_0 \in \text{Expr}_R\).
- **strictness flag** \(strict\) (if false, \(\mathcal{L}\) is called **permissive**).
- **activation mode** \(am \in \{\text{initial}, \text{invariant}\}\).
- **chart mode** **existential** \((\Theta \mathcal{L} = \text{cold})\) or **universal** \((\Theta \mathcal{L} = \text{hot})\).

A set of words \(W \subseteq (\text{Expr}_B \rightarrow B)^w\) is **accepted** by \(\mathcal{L}\) if and only if

\[
\begin{array}{ll}
\Theta & \text{am = initial} \\
\exists \beta \exists w \in W \cdot w^0 \models_\beta ac \land \neg \psi_{\text{exit}}(C_0) \\
& \land w^0 \models_\beta \psi_{\text{prog}}(0, C_0) \land w \models L_B(B(\mathcal{L})) \\
\Theta & \text{am = invariant} \\
\exists \beta \exists w \in W \forall k \in \mathbb{N}_0 \cdot w^k \models_\beta ac \land \neg \psi_{\text{exit}}(C_0) \\
& \land w^k \models_\beta \psi_{\text{prog}}(0, C_0) \land w / k + 1 \in L_B(B(\mathcal{L}))
\end{array}
\]

where \(C_0\) is the minimal (or instance heads) cut.

---

**Full LSC Semantics: Example**
Existential LSC Example: Buy A Softdrink
Existential LSC Example: Get Change

Live Sequence Charts — Precharts
A full LSC $L = (PC, MC, ac_0, am, \Theta_L)$ actually consist of

- **pre-chart** $PC = ((L_P, \preceq_P, \sim_P), TP, \alpha, MSg_P, Cond_P, LcInv_P, \Theta_P)$ (possibly empty).
- **main-chart** $MC = ((L_M, \preceq_M, \sim_M), TM, \alpha, MSg_M, Cond_M, LcInv_M, \Theta_M)$ (non-empty).
- **activation condition** $ac_0 : \text{Bool} \in \text{Expr}_L$.
- **strictness flag** $\text{strict}$ (otherwise called **permissive**).
- **activation mode** $am \in \{\text{initial, invariant}\}$.
- **chart mode** $\text{existential}$ ($\Theta_L = \text{cold}$) or $\text{universal}$ ($\Theta_L = \text{hot}$).

### Pre-Charts Semantics

{..Weakened text due to complexity and visual nature.}
Universal LSC: Example

LSC: buy water
AC: true
AM: invariant I: strict

User CoinValidator CP: ChoicePanel Dispenser

C50 WATER

¬(C50 ∨ E1 ∨ pSOFT ∨ pTEA ∨ pFILLUP)

User CoinValidator CP: ChoicePanel Dispenser

¬(C50 ∨ E1 ∨ pSOFT ∨ pTEA ∨ pFILLUP)

User CoinValidator CP: ChoicePanel Dispenser

¬(C50 ∨ E1 ∨ pSOFT ∨ pTEA ∨ pFILLUP)

User CoinValidator CP: ChoicePanel Dispenser

¬(C50 ∨ E1 ∨ pSOFT ∨ pTEA ∨ pFILLUP)
Forbidden Scenario Example: Don’t Give Two Drinks

LSC: only one drink
AC: true
AM: invariant I: permissive

User | Vend. Ma.
---|---
E1
pSOFT
SOFT
SOFT

false

Note: Sequence Diagrams and (Acceptance) Test

- **Existential LSCs** may hint at test-cases for the acceptance test!
  (as well as (positive) scenarios in general, like use-cases)

- **Universal LSCs** (and negative/anti-scenarios) in general need exhaustive analysis!
  (Because they require that the software never ever exhibits the unwanted behaviour.)
UML

\[ \mathcal{F} = (\mathcal{F}, \mathcal{E}, V, \text{attr}), \text{SM} \]

\[ M = (\Sigma_{\mathcal{T}}, A_{\mathcal{T}}, \rightarrow_{\text{SM}}) \]

\[ \varphi \in \text{OCL} \]

\[ \mathcal{F}, \text{expr} \]

\[ \mathcal{F}, \text{SD} \]

\[ B = (Q_{\text{SD}}, q_0, A_{\mathcal{T}}, \rightarrow_{\text{SD}}, F_{\text{SD}}) \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} (\sigma_1, \varepsilon_1) \cdots \xrightarrow{w_w} ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \]

\[ \mathcal{G} = (N, E, f) \]

Model-Based/-Driven Software Engineering
Model-Driven Software Engineering

- Idea
  - Structure
    - Declarative Behaviour
    - Constructive Behaviour
  - Declarative Behaviour
  - Constructive Behaviour
- Structure
  - requirements model
    - requirements / constraints
      - design
        - system model
  - refine
  - generate / program
- Implementation

Model-Driven Software Engineering with UML

- Idea
  - Class Diagram
    - Sequence Diagram
      - requirements model
        - requirements / constraints
          - design
            - system model
    - refine
  - refine
  - generate / program
- Implementation
Model-Based Testing
Recall: Test Case

Definition. A test case $T$ is a pair $(In, Soll)$ consisting of
- a description $In$ of sets of finite input sequences,
- a description $Soll$ of expected outcomes,
and an interpretation $[$ of these descriptions.

A test execution $\pi$, i.e., $((\pi^0, \ldots, \pi^n) \downarrow \Sigma_w) \in In$ for some $n \in \mathbb{N}_0$, is called
- successful (or positive)
  if it discovered an error, i.e., if $\pi \not\in [Soll]$.
  (Alternative: test item $S$ failed to pass test; confusing: "test failed")
- unsuccessful (or negative)
  if it did not discover an error, i.e., if $\pi \in [Soll]$.
  (Alternative: test item $S$ passed test; okay: "test passed")

Glass-Box Testing: Coverage

- Coverage is a property of test cases and test suites.
- Execution $\pi$ of test case $T$ achieves $p$\% statement coverage if and only if
  $$p = \text{cov}_{stm}(\pi) := \frac{\sum_{i \in \mathbb{N}_0 \text{stm}(\pi_i)}}{|\text{Stm}_S|} \neq 0.$$  
  Test case $T$ achieves $p$\% statement coverage if and only if $p = \min_{\pi \text{ execution of } T} \text{cov}_{stm}(\pi)$.
- Execution $\pi$ of $T$ achieves $p$\% branch coverage if and only if
  $$p = \text{cov}_{cnd}(\pi) := \frac{\sum_{i \in \mathbb{N}_0 \text{cnd}(\pi_i)}}{|\text{Cnd}_S|} \neq 0.$$  
  Test case $T$ achieves $p$\% branch coverage if and only if $p = \min_{\pi \text{ execution of } T} \text{cov}_{cnd}(\pi)$.
- Define: $p = 100$ for empty program.
- Statement/branch coverage canonically extends to test suite $T = \{T_1, \ldots, T_n\}$, e.g. given executions $\pi_1, \ldots, \pi_n$, $T$ achieves
  $$p = \frac{\sum_{1 \leq i \leq n, \pi_i \in \mathbb{N}_0 \text{stm}(\pi_i)}}{|\text{Stm}_S|}, |\text{Stm}_S| \neq 0, \text{ statement coverage.}$$
**Coverage Example**

```c
int f(int x, int y, int z)
{
    i1: if (x > 100 ∧ y > 10)
        z = z + 2;
    else
        s2: z = z/2;
    s2: if (x > 500 ∨ y > 50)
        z = z*2;
    s2:
}
```

- **Requirement**: \{true\} f \{true\} (no abnormal termination), i.e. \( S\ell = \Sigma^* \cup \Sigma^w \).

### Test Suite Coverage

<table>
<thead>
<tr>
<th>( I_n )</th>
<th>( x, y, z )</th>
<th>( i_1/f )</th>
<th>( i_2/f )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( i_3/f )</th>
<th>( i_4/f )</th>
<th>( c_1 )</th>
<th>( c_2 )</th>
<th>( c_3 )</th>
<th>( c_4 )</th>
<th>%</th>
<th>%</th>
<th>( i_2/% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>501, 11, 0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>501, 0, 0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>0, 0, 0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>0, 51, 0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

**Model-Element Coverage**

- **100 % Element coverage** of \( C \)'s state machine:
  - a set of test cases (e.g. Sequence Diagrams) such that
  - when conducting these test cases
    - each state of \( C \) is reached at least once.
    - each transition of \( C \) is taken at least once.

In general: **State coverage of a set of test cases**

- number-of-states reached / number-of-states in state machine.
Excursion: Automatic Test Generation

Model-based Testing

- Given a set of test cases passing for the model,
- and an implementation of the model (maybe hand-written).

- Execute the test cases on the implementation (or the final system).
  This may need an appropriate interpretation. For example, if the test case says
  - send “C50” to the CoinValidator,
  - rather insert a 50 Cent coin into the vending machine.

- If the vending machine does not behave according to the test,
  - then there’s something wrong (wrong test conduction, wrong implementation, etc.).

- If the vending machine does behave according to the test,
  - then we know that this scenario works – not more.
Vocabulary

- **Software-in-the-loop:**
The final implementation is examined using a separate computer to simulate other system components.

- **Hardware-in-the-loop:**
The final implementation is running on (prototype) hardware which is connected by its standard input/output interface (e.g., CAN-bus) to a separate computer which simulates other system components.

References


