UML in Action
A Two-Layered Interpretation for Testing

Bernhard K. Aichernig

Joint work with
Harald Brandl, Elisabeth Jöbstl, Willibald Krenn

Institute for Software Technology
Graz University of Technology
Austria

UML&FM, Nov 16, 2010
Research Context

▶ **FP7 STREP**: Model-based Generation of Tests for Dependable Embedded Systems

▶ 10 Partners:
  - A.I.T (Austria), SP (Sweden)
  - Budapest Univ. of Techn. (Hungary), TU Graz (Austria), Oxford Univ./ETH (UK, Switzerland)
  - Ford (Germany), Prolan (Hungary), Prover (Sweden), Re:Lab (Italy), Thales Rail Signalling Solutions (Austria)

Model-Based Mutation Testing

- UML State Charts: hierarchical, parallel states, constraints over quantors etc. (Papyrus)
- Object-Oriented Action Systems provide formal semantics
- Test-objective based on fault models: mutations in model
- Idea: generate test cases to prevent modeled faults in SUT
- Translator joint work with AIT
- TCG also for Hybrid Systems
Car Alarm System: Interface
Car Alarm System: Behaviour

### Alarm System State Machine

- **OpenAndUnlocked**
  - Open
  - Close
  - Unlock

- **ClosedAndUnlocked**
  - Unlock
  - Lock

- **OpenAndLocked**
  - Close
  - Open

- **ClosedAndLocked**
  - 20

- **Armed**
  - Unlock
  - Show Armed /entry
  - Show Unarmed /exit

- **SilentAndOpen**
  - Close

- **Alarm**
  - Activate Alarms /entry
  - Deactivate Alarms /exit
  - FlashAndSound
    - Flash
    - 30 / Deactivate Sound
  - 300

- **Unlock**
Action Systems Semantics

Action System:

\[
\begin{align*}
\textbf{var} & \quad v : T := \textit{init} \\
\textbf{methods} & \quad M_1; \ldots; M_n \\
\textbf{actions} & \quad A_1 = g_1 \rightarrow v := e_1; \\
& \quad \ldots; \\
& \quad A_m = g_m \rightarrow M_i(e_i); \\
\textbf{do} & \quad A_1 \\
& \quad \quad \quad \Box \\
& \quad \quad A_2; A_3 \\
& \quad \quad \quad \quad \text{//} \\
& \quad \quad A_m \\
\textbf{od} & \quad : M_i
\end{align*}
\]

Motivation:

- Well-suited for embedded systems modeling (Event-B)
- Action view maps naturally to LTS testing theories
- Solid foundation:
  - precise semantics
  - refinement
- Compositional modeling
- Many extensions available:
  - object-orientation
  - hybrid systems
### UML State Transition as Action

1. `transition_Armed_to_Alarm =`
2. `requires`
3. `state = Armed and`
4. `events <> [nil] and`
5. `(hd events)[0] = Open`
6. `:`
7. `state := Alarm`
8. `end;`
9. `...`
10. `dequeue =`
11. `requires`
12. `events <> [nil]`
13. `:`
14. `events := tl self.events`
15. `end`
UML Entry Actions

Protocol layer in do-od-block serves to model entry actions:

```
1 do
2  ( transition_Armed_to_Alarm;
3      call_ShowUnarmed;
4      call_ActivateAlarm;
5      call_AcousticAlarm_SetOn )
6  []
7  transition_OpenAndLocked_to_ClosedAndLocked
8  []
9  /* .. other transitions .. */
10 )
11  // dequeue()
12 od
```
LTS Semantics of Action Systems

Mapping to CADP toolbox:
- model simplification
- model checking
- scenario-based TCG in TGV

IOLTS testing theories
76 Alarm System Mutations, e.g.
Mutation-Based Test Case Generation

Action System Model

\[ IOLTS^S \]

For every mutant \( ioco \), there is a discriminating test case.

\[ IOLTS^M \]

\[ \text{ioconf} \ldots \text{input-output conformance} \]
Mutation-Based Test Case Generation

Action System Model

Ioconf ... input-output conformance
Mutation-Based Test Case Generation

Action System Model

\[ IOLTS^S \]

for every mutant

\[ IOLTS^M \]

discriminating test case

ioconf ... input-output conformance
Mutation-Based Test Case Generation

Action System Model

`IOLTS^S` discriminating test case

for every mutant

`IOLTS^M`

ioconf ... input-output conformance
Tool Chain

Figure: Test Case Generation Tool Chain.

Figure: The computation steps of Ulysses.
### Results: Generated Test Cases

<table>
<thead>
<tr>
<th></th>
<th>Mutation (Ulysses)</th>
<th>Scenarios (TGV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML Mutants [#]</td>
<td>76</td>
<td>-</td>
</tr>
<tr>
<td>Max. Depth</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>Gen. TCs [#]</td>
<td>63</td>
<td>9</td>
</tr>
<tr>
<td>Duplicates [#]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unique [#]</td>
<td>59</td>
<td>9</td>
</tr>
<tr>
<td>Gen. Time [min]</td>
<td>23</td>
<td>-</td>
</tr>
</tbody>
</table>
Testing a Java Implementation

- Implemented the Car Alarm System in Java
- Created 72 mutants with \( \mu \text{java} \)
- Selected 38 relevant mutants

<table>
<thead>
<tr>
<th>State</th>
<th>Mutants</th>
<th>Equiv.</th>
<th>Pairwise Equiv.</th>
<th>Different Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetState</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Close</td>
<td>16</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Open</td>
<td>16</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Lock</td>
<td>12</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Unlock</td>
<td>20</td>
<td>2</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Constr.</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>72</td>
<td>8</td>
<td>26</td>
<td>38</td>
</tr>
</tbody>
</table>
Surviving Implementation Mutants:

<table>
<thead>
<tr>
<th></th>
<th>Mutation-Based TCG</th>
<th>TGV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetState</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Close</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Open</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Lock</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Unlock</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Constr.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>1</td>
<td>13</td>
</tr>
</tbody>
</table>

Detection Rate

- Mutation-Based TCG: 97%
- TGV: 66%
Conclusions

- Action System Semantic for UML
  - precise, compositional background models
  - concurrency!
  - straightforward IOLTS interpretation

- Tools
  - **CADP**: scenario-based test case generation
  - **Ulysses**: mutation-based test case generation

- We need both!
  - Industry wants scenarios & requirements traceability
  - Scenarios fast, but ad-hoc
  - Mutations slow, but safer

- By the way: without UML our tool is 50% faster
- Ongoing: tool improvement, semantic mutations