UML Checker – A Toolset for Conformance Testing against UML Sequence Diagrams

https://blogs.fe.up.pt/sdbt/

João Pascoal Faria, FEUP/INESC TEC, jpf@fe.up.pt
(with Ana Paiva, Mário Castro, Zuhanli Yang, Tamara Krasnova, and Bruno Lima)
MAPI, 03 December 2014
Motivation

- The development of detailed UML design models of software intensive systems for documentation only has several problems
  - is time consuming
  - the models are often wrong (no static analysis, compilation and testing)
  - the models soon become outdated and are not maintained

- MDD approaches aim at avoiding such problems by generating executable applications from models

- However, in many cases, the level of detail of the behavioral models needed to generate complete applications may be too high or only effective for specific domains
For situations where developing full behavioral models is not practical, we propose a lightweight approach:

- continue to develop structural models from which parts of the application can be generated (e.g., class skeletons)
- develop partial behavioral models, not sufficient for app generation, but adequate for test generation

Partial behavior spec = Test spec

This is also more in line with the agile values

(value more) Working software over comprehensive documentation

To demonstrate the approach we developed a tool that generates executable tests from parameterized sequence diagrams acting also as specifications of test scenarios
Approach: hybrid MDE (2)

5. Complete production code (method bodies)

1a. Model app struct
1b. Model app behavior

2. Check model consistency & completeness (UML Checker)

3a. Generate production code skeletons (EA)
3b. Generate test code (TestGenerator)

UML Class Diagrams
UML Sequence Diagrams

Production code (Java)
Test code (JUnit3)

Standard libraries (Java)
Tracing library (AspectJ)

Traces execution of methods & constructors
Uses
Tests

New test library

5. Complete production code (method bodies)

6. Execute tests and see them pass
4. Execute tests and see them fail

New EA plugin
New EA plugin
New EA plugin
Test ready sequence diagrams

Actor (client app or user) 

- Things in the system
- Things not yet implemented
- Example values for parameters

Behavioral Model/Spec

Generated Test Code

- (Driver) Generate inputs as in spec and check responses against spec
- (Monitor) Trace execution and check against spec
- (Stub) Generate the responses as in spec
- Exercise the scenario for each example
Tool user interface
Tool architecture (v3)

*EPN=Extended Petri Nets
Live demonstration

- Example UML model
- Test generation
- Test code generated
- Test execution and reporting (including coverage information)
- Bug fixing
- Stubs
- Loose conformance
- Combined fragments
- User interaction testing
- Model consistency and completeness checking
Key features and benefits (1)

Feature

- Support the modeling & automatic testing of
  - External interactions with users (UI)
  - External interactions with client applications (API)
  - Internal interactions among objects in the program

Benefits

- Covers 4 design views (w/ structural model)
- Assures higher conformance with spec
- Improves fault localization
- Accelerates test phase

<table>
<thead>
<tr>
<th>Dynamic</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext.</td>
<td>Ext.</td>
</tr>
<tr>
<td>Sequence diagrams (external interactions)</td>
<td>Class diagrams (public/external interfaces)</td>
</tr>
<tr>
<td>Int.</td>
<td>Int.</td>
</tr>
<tr>
<td>Sequence diagrams (internal interactions)</td>
<td>Class diagrams (private/internal interfaces)</td>
</tr>
</tbody>
</table>
## Key features and benefits (2)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameterization</td>
<td>Keep behavioral specs as generic as desired</td>
</tr>
<tr>
<td>Combined fragments (alt, opt, loop, par)</td>
<td>Keep behavioral specs as simple as desired (focus on relevant interactions)</td>
</tr>
<tr>
<td>Loose conformance checking</td>
<td>Verifiable completeness criteria</td>
</tr>
<tr>
<td></td>
<td>Higher quality assurance</td>
</tr>
<tr>
<td></td>
<td>Iterative implementation &amp; testing</td>
</tr>
<tr>
<td>Automatic checking of model consistency &amp; completeness</td>
<td>Independence of external components</td>
</tr>
</tbody>
</table>

- Additional or intermediate calls are allowed in implementation.
- "Stubs" inject the specified response messages for things marked as not yet implemented.
## Related work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction parameters</td>
<td>Partly&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Partly&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Partly&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Keyword-based UI testing</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partly&lt;sup&gt;b&lt;/sup&gt;</td>
</tr>
<tr>
<td>Internal interaction checking</td>
<td>Yes&lt;sup&gt;c&lt;/sup&gt;</td>
<td>Partly&lt;sup&gt;d&lt;/sup&gt;</td>
<td>No</td>
<td>Partly&lt;sup&gt;e&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;f&lt;/sup&gt;</td>
</tr>
<tr>
<td>Loose conformance checking</td>
<td>Yes</td>
<td>No&lt;sup&gt;g&lt;/sup&gt;</td>
<td>No</td>
<td>Partly&lt;sup&gt;h&lt;/sup&gt;</td>
<td>Yes</td>
</tr>
<tr>
<td>Test stub injection</td>
<td>Yes&lt;sup&gt;i&lt;/sup&gt;</td>
<td>No</td>
<td>No</td>
<td>Partly&lt;sup&gt;j&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;k&lt;/sup&gt;</td>
</tr>
<tr>
<td>Interaction operators</td>
<td>No&lt;sup&gt;l&lt;/sup&gt;</td>
<td>No</td>
<td>No</td>
<td>Partly&lt;sup&gt;m&lt;/sup&gt;</td>
<td>Yes</td>
</tr>
<tr>
<td>Complex value specifications</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes&lt;sup&gt;n&lt;/sup&gt;</td>
</tr>
<tr>
<td>Test code generation</td>
<td>No&lt;sup&gt;o&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;p&lt;/sup&gt;</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Test results in the model</td>
<td>Partly&lt;sup&gt;q&lt;/sup&gt;</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Model coverage analysis</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Conclusions

- Presented a lightweight MDE approach
  - Based on lightweight behavioral and structural models
  - (Partial) production code and (full) test code generation from models
- That is “PSP friendly” (PSP – Personal Software Process)
  - Promotes complete (in a sense), precise and reviewable designs
  - Embeds test specification in the design phase (as behavior specs)
  - Is designed to bring short term productivity and quality benefits
- And “agile friendly”
  - Compilable models are not just documentation
  - TDD/BDD [create a test = create an (external + internal) behavior spec]
Ongoing work

- Extend UI modeling and testing features for GUls
- Automatically generate test data (i.e., actual values for scenario parameters) through constraint satisfaction
- Conduct more extensive experimentation and process performance and usability analysis
- Support the testing of time constrained, concurrent and distributed systems, particularly for integration testing
References and further reading

- See: [https://blogs.fe.up.pt/sdbt/](https://blogs.fe.up.pt/sdbt/)
- **Integrating Model-Driven Engineering Techniques in the Personal Software Process**, João Pascoal Faria, TSP Symposium 2012, St. Petersburg, Florida, USA