Model-Based Testing
Using LSCs and S2A
(a preliminary industrial experience)

Shahar Maoz\textsuperscript{1}, Jani Metsä\textsuperscript{2}, Mika Katara\textsuperscript{2}
\textsuperscript{1}The Weizmann Institute of Science, Israel
\textsuperscript{2}Tampere University of Technology, Finland
Short presentation at MoDELS 2009, Empirical Results Track
One Slide Abstract

• We report on using high-level visual scenario-based models for test specification, test generation, and aspect-based test execution, in the context of an industrial application
  – To specify scenario-based tests, we used a UML2-compliant variant of live sequence charts (LSC)
  – To automatically generate testing code from the models, we used a modified version of the S2A Compiler, outputting AspectC++ code
  – Finally, to examine the results of the tests, we used the Tracer, a prototype tool for model-based trace visualization and exploration

• Our experience reveals the advantages of model-based test specification and aspect-based execution
  – Generating aspect code from visual models enables exploiting the expressive power of aspects for testing without manual coding and without knowledge of their rather complex syntax and semantics

• We discuss technological and other barriers for the future successful integration of our initial work in industrial context
Live Sequence Charts [DH01]

• LSC is a visual formalism for inter-object scenario-based specifications

• Extends classical sequence diagrams mainly with a universal interpretation, must/may (hot/cold) and monitor/execute modalities
  – Allows to express liveness and safety properties
  – Allows to differentiate between monitoring and execution

• Has formal semantics, translation into TL [KHPLB05], UML2-compliant version using a profile [HM06], execution/synthesis [MH06,HK01]
Scenarios to Aspects [MH06,HKM07]

• S2A: a compiler that translates LSCs into aspects

• Each LSC is translated into a monitoring scenario aspect, simulating an automaton that follows the scenario’s progress
  – aspect pointcuts listen out for relevant events
  – aspect advice advances the automaton to the next cut

• The generated aspect code reports the scenarios’ activation, progress, completions, and violations

• Original implementation uses AspectJ as the target language
  – we created and used a modified version, outputting AspectC++ code
Trace Visualization and Exploration [MKH07]

- The Tracer is a tool for scenario-based trace analysis, visualization and interactive exploration

- Basic view: hierarchical Gantt chart, use-cases and scenarios in the model, following the scenarios progress

- Identifying completions and violations

- Computing trace metrics

- Interactive exploration: navigation, filters, comparisons
Case Study Setup

- Conducted by the second listed author Jani Metsä while he was working for Nokia Corp., Devices R&D

- Testing a C++ application written by Nokia, running on Symbian OS inside a Nokia N96 smartphone

- Due to confidentiality restrictions, some details of the case study cannot be made public
Case Study

• Specified 32 scenarios
  – some represented existing textual test documents
  – drawn using IBM Rational Software Architect

• Scenarios translated into AspectC++ aspects
  – using S2A

• Generated testing code weaved into application code, executed on the Nokia smartphone

• Exploring test results
  – using the Tracer
Case Study Example Scenario (1)
Case Study Example Scenario (2)
Case Study Example Trace Output
Case Study Evaluation (1)

- Testing functionality specified at a behavioral level, without implementation details

- No need to know aspects
  - access to SUT internals without requiring the engineer to know aspects
  - automatic generation guarantees certain quality in the code executing the tests

- End-to-end visualization
  - believed to be a positive adaptation factor
Case Study Evaluation (2)

- Access to a model of the SUT is required
- Good command of the modeling language is required
  - suggest the use of test templates

- Modeling language expressive power and semantics; LSC sometimes unable to express the tests we wanted to specify
  - lacking explicit reference to threads
  - not easy to define complex control structures

- Interoperability problems, immature tool implementations
  - mix of academic prototypes and commercial software
  - no single IDE
  - a lot of process overhead, difficulties in identifying source of problems
Conclusion and Future Work

• We presented a new tool chain for model-based testing using aspect code generated from scenario-based models

• Evaluation using an initial case study in an industrial context

• Future work
  – Additional experiments
  – Improving tool implementations
Model-Based Testing
Using LSCs and S2A
(a preliminary industrial experience)

Thank You!

Shahar Maoz¹, Jani Metsä², Mika Katara²
¹The Weizmann Institute of Science, Israel
²Tampere University of Technology, Finland
Short presentation at MoDELS 2009, Empirical Results Track