Model-Based Testing: Automated Generation of Test Cases, Test Data, and Test Procedures from SysML Models

Jörg Brauer and Jan Peleska
Verified Systems International GmbH

support@verified.de

Space Tech Expo Europe – 2015-11-19
Motivation

• Model-driven system and software development has become an established best practice – at least in certain application fields

• Model-based testing, however, is an active research area, but many enterprises are hesitant to adopt it as an integrated part of their V&V processes
Motivation

• In this presentation, it is explained why
  – Model-based testing is fit for industrial application
  – The return of investment into test model development is significant, leading to
    • reduced V&V costs
    • increased test strength
    • simpler accumulation of certification evidence
Overview

• Workflow – overview
• Test model development
• Model-based requirements tracing
• Test procedure generation and execution
• Conclusions
Overview

• **Workflow – overview**
• Test model development
• Model-based requirements tracing
• Test procedure generation and execution
• Conclusions
MBT Workflow

Manual activities

Requirements
  Test Model Development
  Test Model
  Requirements Tracing
  Traceable Test Model

Test Case Identification
  Traceable Test Cases
  Test Data Computation
  Test Data per Test Case
  Test Procedure Generation

Test Procedures
  Test Execution and Evaluation
  Test Results
  Compilation of Traceability Data
  Traceability Matrix
MBT Workflow

- Fully automated activities

Requirements

- Test Model Development

Test Model

- Requirements Tracing

Requirements Tracing

- Traceable Test Model

Test Case Identification

- Traceable Test Cases

Test Data Computation

- Test Data per Test Case

Test Procedure Generation

Test Procedures

- Test Execution and Evaluation

Test Results

Compilation of Traceability Data

Traceability Matrix
Overview

• Workflow – overview
• **Test model development**
• Model-based requirements tracing
• Test procedure generation and execution
• Conclusions
Test Model Development

• The test model specifies
  – **Interfaces** between SUT and environment are represented as readable or writable in the test campaign
  – **Structural aspects** – functional decomposition
  – **Behavioral aspects** – transformation of inputs into outputs, sequencing, synchronisation ...

• Two alternatives for test model creation and utilisation (see next slides)
Validated Test Models – Variant 1

Development Model

- Model created by the development team

Test Model

- Redundant model developed from requirements by the V&V team

Code

- Manually developed or automatically generated from development model
- Automatically generated from test model

SUT

- Integrated HW/SW System

Test Procedures

- Test Engine

HW/SW integration tests check consistency of SUT with test model
Validated Test Models – Variant 2

- Development and Test Model
  - Code: Manually developed or automatically generated from model
  - Test Procedures: Automatically generated from model

SUT: Integrated HW/SW System

HW/SW integration tests check consistency of SUT with model

Test Engine
Overview

• Workflow – overview
• Test model development
• **Model-based requirements tracing**
• Test procedure generation and execution
• Conclusions
Model-Based Requirements Tracing

• Objective
  – Link model elements to associated requirements that are represented by these elements
  – This allows us to identify test cases suitable for verifying a given requirement in an automated way

• SysML is a system modeling language providing (graphical and textual) syntax for linking requirements to behavioral and structural model elements
Model-Based Requirements Tracing

FLASHING

ON
Entry/t.reset()

OFF
Entry/t.reset();
LampsLeft = 0;
LampsRight = 0;
ctr = ctr + 1;

[t.elapsed(320)]/
LampsLeft = lOld;
LampsRight = rOld;

«requirement»
REQ-002 Flashing with
340ms/320ms on-off periods

«satisfy»

[t.elapsed(340)]/
Overview

• Workflow – overview
• Test model development
• Model-based requirements tracing
• **Test procedure generation and execution**
• Conclusions
Test Procedure Generation and Execution

• Test cases can be identified by evaluating the links between model elements and requirements
• The behavior expressed by the model can be internally encoded by logical formulas
• The test case is internally represented by a logical formula as well
• → **Concrete test data can be automatically calculated using mathematical constraint solvers**
Test Procedure Generation and Execution

- For test procedure generation, users just configure which test cases should be covered by the procedure to be created
  - **Requirements-driven generation** is performed by identifying the requirement (or a subset of related test cases) to be tested by the procedure
  - **Model-driven generation** is performed by identifying the model portions to be covered by the test procedure
Requirements-Driven Test Procedure Generation

**Basic Control State Coverage**
- TC-turn_indication-BCS-0001
- TC-turn_indication-BCS-0002
- TC-turn_indication-BCS-0003
- TC-turn_indication-BCS-0004
- TC-turn_indication-BCS-0005
- TC-turn_indication-BCS-0006
- TC-turn_indication-BCS-0007
- TC-turn_indication-BCS-0008
- TC-turn_indication-BCS-0009
- TC-turn_indication-BCS-0010

**Basic Control State Pairs Coverage**

**Hierarchical Transition Coverage**

**MC/DC Transition Coverage**

**Transition Coverage**
- TC-turn_indication-TR-0001
  Cover transition of component IMR.SystemUnderTest.FLASH_CTRL
  FLASH_CTRL.EMER_OFF
  -- [ IMR.EmerSwitch ] -->
  FLASH_CTRL.EMER_ON
- TC-turn_indication-TR-0002
Requirements are displayed with their associated test cases, to be selected for test procedure generation.
Model-Driven Test Procedure Generation
Model browser allows to select elements to be covered by the test procedure.
Test Procedure Generation and Execution

• 6 Different testing strategies are supported
  – **Simple**: Basic control state coverage, Transition coverage, Hierarchic transition coverage.
  
  – **Complex**:
    • **MC/DC coverage**. Complex guard conditions are exercised with different valuations of their atomic condition parts
    • **Basic control state pairs coverage**. Interacting state machine pairs are tested in every possible state combination
    • **Equivalence class testing strategy**. Inputs with large data types are automatically partitioned into equivalence classes
Test Procedure Generation and Execution

• Applicable strategies can be automatically adapted to the criticality of SUT component under consideration
  – For DAL-C components, transition coverage typically suffices
  – For DAL-A components, the more complex strategies have to be applied as well
Overview

• Workflow – overview
• Test model development
• Model-based requirements tracing
• Test procedure generation and execution
• Conclusions
Conclusions

• Model-based testing is fit for large-scale application in industry

• Efficiency measurements performed by Verified Systems show that the following effort reductions in comparison to conventional testing approaches can be expected
  – 30% for a new project, where a new test model has to be created
  – 90% in the best case, where an optimised re-usable workflow has been set up for regular regression testing of a long-lived product