Practical Model-based Testing
With Papyrus and RT-Tester

Jan Peleska and Wen-ling Huang
University of Bremen
Verified Systems International GmbH
Fourth Halmstad Summer School on Testing,
2014-06-11
Acknowledgements. *This presentation has been elaborated in the context of the EU FP7 COMPASS project under grant agreement no.287829.*
Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises
Overview

- Model-based testing
  - Test Modelling With Papyrus
  - Model-based Testing With RT-Tester
  - Requirements, test cases, procedures, results, and Traceability
  - Demonstration and Practical Exercises
Our MBT Approach

Instead of writing test procedures,

• develop a **test model** specifying expected behaviour of SUT ➔ the first MBT variant

• use **generator** to identify “relevant” test cases from the model and calculate concrete test data

• generate **test procedures** fully automatic

• perform **tracing requirements ↔ test cases** in a fully automatic way
MBT-Paradigm

**Model**
- Is a partial description of
- Are derived from

**System**
- Can be run against

**Abstract Tests**
- Are abstract versions of

**Executable Tests**
Overview

- Model-based testing
- **Test Modelling With Papyrus**
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises
Papyrus

- Modelling with EMF-based formalisms
- EMF – Eclipse Modelling Framework
- Papyrus provides UML, SysML, DSL support
- Open source – free to use
- http://www.eclipse.org/papyrus/
SysML

• Block definition diagrams
• Internal block diagrams
• Item flows
• State machines with timers
• Operations
• Requirements
• <<satisfy>> relationship between requirements and model elements
Case Studies With SysML

• Simplified version of the turn indication and emergency flashing function in Daimler vehicles

• Full model available under

http://www.mbt-benchmarks.org

  ➔ Benchmarks

  ➔ Turn Indicator Model Rev. 1.4
Case Studies With SysML

• New model available: the Ceiling Speed Monitor of the ETCS (European Train Control System)

• Full model available under

http://www.mbt-benchmarks.org

➔ Benchmarks

➔ openETCS/ceiling-speed-monitoring
Model Introduction With Papyrus

• System interface – block diagram
• Requirements
• System Under Test – internal block diagram
• Further decompositions – internal block diagrams and block references
• Behaviour associated with block leaves – state machines and operations
Overview

- Model-based testing
- Test Modelling With Papyrus
- **Model-based Testing With RT-Tester**
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises
RT-Tester Internals

Reference Tool RT-Tester

- Supports all test levels – from unit to system integration testing
- Software tests and hardware-in-the-loop tests
- Test projects may combine hand-written test procedures with automatically generated procedures

→ The tool capabilities are presented here to stimulate benchmarking activities
Eclipse – Papyrus – RT-Tester Integration

Your Laptop – Client

Eclipse

Papyrus Plugin

RT-Tester Plugin

Server

RT-Tester
MBT Server
Eclipse – Papyrus – RT-Tester Integration

Your Laptop – Client

- Eclipse
- Papyrus Plugin
- RT-Tester

Server

- RT-Tester MBT Server

Free

Free for academic use

Server located at University of Bremen
Tool Components and Data Structures

- Modelling Tool
  - Model (XMI)
- RT-Tester Model Parser
- Model Transformers
- Transition Relation Generator
- Transition Relation
- Test Case Generator
  - Test Case-Specific Goal
- RT-Tester IMR (AST)
- SMT-Solver SONOLAR
- SMT-Solver SONOLAR
- SMT-Solver SONOLAR
- Model State Abstractions
- Concrete Test Data
- Concrete Interpreter
- Abstract Interpreter
- Test Procedure Generator
- RT-Tester Test Procedure
Tool Components and Data Structures

Modelling Tool
- UML/SysML subset
- Enterprise Architect
- Artisan Studio
- Rhapsody
- Papyrus

Alternatively:
- DSL
- MetaEdit+

Model (XMl)

RT-Tester Model Parser

Transition Relation Generator

SMT-Solver SONOLAR

Test Procedure Generator

RT-Tester Test Procedure

Model Transformers

Test Case Generator

Test Case-Specific Goal

RT-Tester IMP

Concrete Test Data

Concrete Interpreter

Abstract Interpreter

Model State Abstractions
**Parser Front Ends**

- Transform model representations in XMI format into abstract syntax tree
- \(\text{AST} = \text{Internal Model Representation IMR}\)
Model Transformers provide alternative AST representations
- Cone of influence reduction
- Test oracles
- Equivalence class abstraction
Test Case Generator
- identifies “relevant” test cases
- uses ASTs as identification basis
- exploits traceability information from requirements to model elements
- encodes test case goals as propositions

\[ G(s_0, s_1, \ldots, s_C) \]
Transition Relation Generator

- encodes operational semantics of the model by relating pre-states to post states

\[ \Phi(s, s') \]
SMT-Solver

- calculates solution of test goals which are compatible with the transition relation

\[ J(s_0) \land \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \land G(s_0, \ldots, s_{n+1}) \]

Can handle Boolean, Integer, Float, Array data types
Concrete interpreter
- executes the model from current pre-state with the input data calculated by the solver
Abstract interpreter

- speeds up SMT-solver by
- calculating minimal number of steps required for finding solutions
- restricting the ranges of inputs and other model variables in traces leading to a solution of

\[ J(s_0) \land \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \land G(s_0, \ldots, s_{n+1}) \]
Test Procedure Generator
- is a compile back-end for transforming test case solutions to executable test procedures
- provides different compile back-ends for RT-Tester Real-Time Test Language, PROVEtech:TA, and TTCN-3
Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises
Model Semantics

• Based on Kripke Structures

• Equivalent to alternative operational semantics based on labelled transition systems

\[ K = (S, S_0, R, L) \]

\[ S : \text{State space} \]

\[ S_0 \subseteq S : \text{Initial states} \]

\[ R \subseteq S \times S : \text{Transition relation} \]

\[ L : S \rightarrow 2^{AP} : \text{Labelling function} \]

\[ AP : \text{Atomic propositions} \]
Requirements

Each requirement is reflected by set of model computations

\[ \pi = s_0.s_1.s_2 \ldots \]

Computation sets can be characterised by Linear Temporal Logic (LTL)

\[ G\phi : \text{Globally } \phi \text{ holds on path } \pi \]
\[ X\phi : \text{In the next state on path } \pi, \text{ formula } \phi \text{ holds.} \]
\[ F\phi : \text{Finally } \phi \text{ holds on path } \pi \]
\[ \phi U \psi : F\psi \text{ and } \phi \text{ holds on path } \pi \text{ until } \psi \text{ is fulfilled} \]
Requirements Tracing – Complex Requirements

- Computations contributing to complex requirements require full LTL expressions
- Insert LTL formula in constraint
- Link constraint to requirement via <<satisfy>> relation
Test Cases

- Test cases are finite witnesses of model computations
- Trace = finite prefix of a computation
- If computation satisfies LTL formula associated with a requirement, trace prefixes must at least not violate this formula
- Some formulas can only be verified on an infinite computation (liveness formulas, e.g. fairness properties)
- But these properties can only be partially verified by testing
Test Data Computation

• LTL formulas interpreted on finite traces can be transformed into first order expressions

\[ tc \equiv J(s_0) \land \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \land G(s_0, \ldots, s_{n+1}) \]

• Recall. These formulas can be solved by an SMT solver
Model Coverage Strategies

Strategies currently realised in RT-Tester

- Basic control state coverage
- Transition coverage
- MC/DC coverage
- Hierarchic transition coverage
- Equivalence class and boundary value coverage
- Basic control state pairs coverage
- Interface coverage (under construction)
- Block coverage (under construction)
- Equivalence class partitioning (under construction)
Overview

• Model-based testing
• Test Modelling With Papyrus
• Model-based Testing With RT-Tester
• Requirements, test cases, procedures, results, and Traceability

• Demonstration and Practical Exercises
Test Generation Context and Test Execution Context

• Test generation context. Configure the test procedure to be generated

• Test execution context. Execute the test procedure against the system under test
Work Flow

• Create the test model (Papyrus perspective)
• Create RT-Tester project (RT-Tester perspective)
• Import model to RT-Tester project
• Configure and create initial test procedure – model-coverage approach
  • Configuration file
  • Signal map
• Analyse signal flow
Work Flow

• Optional: create a simulation
• Compile and run test procedure
• Replay test procedure
• Analyse requirements and test cases
• Create new generation context
• Allocate test cases to procedure to be generated