A scenario-based test approach for testing reactive concurrent systems

Andreas Ulrich

Siemens AG, Corporate Technology
München, Germany

ETSI MBT User Conference 2011
October 18–20, 2011, Berlin, Germany
Outline

- Motivation
- Scenario-based testing
- Case study: clinical imaging device
- Modeling test scenarios
- Tool snapshot
- Conclusions
Motivation

Limits of current MBT approaches and tools
- Rely on models that are expensive to create
- Focus on structural coverage of model, but not fault detection
- Insufficient support for concurrent interactions

Ways out from the MBT crisis
- Simplify models to carry only essential parts
- Support concurrency directly in the model
- Provide sound test implementations with known fault detection
What is scenario-based testing?

Cam Kaner on Scenario Testing, STQE Magazine, Sep./Oct. 2003

- The scenario is a story about someone trying to accomplish something with the product under test.

- Scenarios are useful to connect to documented software requirements, especially requirements modeled with use cases.

- A scenario test provides an end-to-end check on a benefit that the program is supposed to deliver.

→ Here we use scenarios to systematically test for the correct implementation of requirements in the system.
Scenario-based Testing

- Support for **embedded software testing** of mechatronic components
  - Event-triggered systems
  - Asynchronous, i.e. message-passing
  - Multiple ports / interfaces
  - Concurrent messages

- **Test scenario** derived from a use case
  - Detailed interactions at SUT interfaces
    - Partial system spec
  - Specified as UML sequence diagrams (MSC)

- **Test generation** produces typically one test implementation per test scenario
Case study: Digital radiographic system Ysio

- **Ysio**
  - Latest generation of clinical X-ray devices
  - Digital image processing
  - Operates fully automatically

- **System integration testing** of a controller unit
  - Ethernet, TCP/IP interface: commands
  - CANopen interface: resulting interactions

![Diagram of X-ray system components](image)
Modeling test scenarios – Overview

Static view on the SUT with its external ports and events/messages.

Set of scenarios that describe interactions at the SUT’s ports (black-box approach). Each scenario represents a test.

Optional graph that links scenarios together. Useful when describing choices over SUT inputs. Used for generating tests across scenarios.
Modeling test scenarios – Test architecture

- SUT is modeled as a **single instance**, even if comprised of several distributed components

- **All ports / interfaces** of the SUT that are exposed in testing must be defined together with its events / messages
  - Points of control and observation – SUT inputs and outputs
  - Points of observation – SUT outputs only

→ Multi-port system
→ Black-box testing approach

- Assigning event / message types to port types enables validation of test scenario models → e.g. misuse of messages at a given port
A scenario describes the behavior of a (possibly distributed) SUT as it is observable at its (multiple) ports by an assumed ideal global tester.

A scenario describes the expected behavior of the SUT:
- Hence, any deviation observed in testing is a failure
- Derived from system requirements and use cases

Modeling notation:
- UML sequence diagram (MSC)
- UML interaction overview diagram (optional)

One scenario relates to one executable test
Basic concepts for behavioral modeling taken from CSP – Communicating Sequential Processes (Hoare 1978)
- (MSC) Sequence \(\rightarrow\) (CSP) Prefixing, sequence
- (MSC) Loop \(\rightarrow\) (CSP) Recursion
- (MSC) Alternative \(\rightarrow\) (CSP) Non-deterministic choice
- (MSC) Parallel \(\rightarrow\) (CSP) Concurrency (interleaving)
- (MSC) Unless \(\rightarrow\) (CSP) Interruption

Not all concepts are expressible in UML2/MSC!

Some extensions to cope with testing
- Optional messages \(\rightarrow\) variant of alternative
- Unless \(\rightarrow\) Exceptional behavior within a defined scope
- Requirement tracing
- Ignore messages \(\rightarrow\) ignore superfluous SUT outputs
Scenario based testing for Ysio, Example 1

Requirement tracing

Single SUT lifeline

References to sub-scenarios
Scenario based testing for Ysio, Example 2

Flow of system interactions

Concurrency occurs naturally at different system ports

Termination of infinite loop
Implementing scenario-based testing – The ScenTest Tool

Test Scenario Construction
(Enterprise Architect UML2 Editor)

Test Implementation
(Eclipse based)

Test Adaptation
(JUnit, Log4J)

Use Cases → Manual Operation → Test Scenarios

Building Coordinated Test Implementation → MSC Test Impl. → Promela Code Generation → Promela Model

Mapping Promela into Executable Code → Java
ScenTest Tool Snapshot

Test scenario specification in Enterprise Architect

Generated tests run under JUnit

Tests support the TTCN-3 runtime interface

Fully integrated into Eclipse
Conclusions

Test scenarios
- Describe interactions of SUT with its environment
  - Expected behavior
  - Concurrency in case of multi-port SUTs
  - Can be linked with requirements
- Simple mean to lift the specification of tests to model level
- Highly accepted by practitioners

Tooling
- Similar approach has been tried in functional HiL testing of embedded SW, see e.g. EXAM tool by VW/Audi
- However, no commercial tool for integration testing based on message passing exists so far
- Build your own tool gradually with increasing demands for new features
Thank you for your attention!