Automation of Test Design with Model-Based Testing
(A Statechart-based Approach)

MBT UC 2011, Berlin, Germany
INTRODUCTION TO AUTOMATED TEST DESIGN WITH MBT
Evolution of Software Testing

- MBT
- Frameworks
- Keyword Driven
- Scripts-Based
- Capture/Replay
- Manual
Model-Based Testing

- Umbrella term for any approach that uses models for testing
- One of them is to use MBT for automating test design
  - Models reflect externally observable behavior of the system to be tested
  - MBT complements test execution
- First industrial standards (ES) being developed at ETSI
  - Multiple tools available in the market
  - Involves stakeholders from different application domains
Outline of ETSI Standard 202 951
(Requirements for Modelling Notations)

- Model-Based Test Development Overview
- General Modeling Notation Requirements
  - Modularization, Algorithms, Documentation
- Modeling System Interface
  - Actions, Operations, Ports, Configurations
- Modeling System Behavior
  - System State, System State Transition, Non-Determinism
- Examples of Modeling Notation Styles (Annex)
  - Rule-Based/EFSM, Statechart, Process-Oriented
Automated Test Design Workflow

Develop (System) Model

Direct & Review Test Design

Generate Test Scripts & Documentation
MBT - A New Way of Testing

Functionality vs Path
SUT vs Environment
Coverage vs Number of Tests
Abstraction vs Detail
Optimization vs Expert Opinion
Flexible vs Fixed Test Sets
What Remains the Same?

• The starting point will be an informal specification
  – Customer: "I want a [ sæt ]"

• Tests have to be validated
  – Any testing artefact remains as good as the *agreed*
    understanding of the specification

• Still infinitely many tests have to be executed to prove
  that a SUT faithfully follows a specification
  – But a *limited* amount of time is available for test execution
The Business Case for Automated Test Design

- Model reuse
- Savings in test maintenance
- Improved test coverage
- Higher productivity
- Customizable test sets
- Enabling new ways of working
- Test scripting language independence

ROI
Automated Test Design Benefits

- Less customer found defects
- More available resources
- Shorter testing turnaround time
- Reliable reports and documentation
- Test suites with optimal coverage

Less Time
Spend less calendar time in test process, get product out faster

Higher Quality
More effective test sets resulting in a higher quality product

Reduced Cost
With less resource to higher quality
The BIG Picture: From Manual Test Design …
The BIG Picture: ... to Automated Test Design
Introduction Summary

• MBT is next step in evolution of software testing
• First industrial standards attest maturity
• Use MBT allows engineers to work (usually >5x) more effectively and focus on the essence of testing
• Next to new notations & tools MBT requires a change in thinking for management and engineers
• Automatic test generation is only one part of MBT
MODELING FOR TESTING
About (System) Models

• Structural view: SUT interface available for testing
  – 1+ (logical) ports and parameterized input & output (actions) reflecting controllable & observable interfaces
  – At least one but possibly multiple parallel model components

• Behavioral view: The expected SUT operation
  – Should focus on one or more aspects to be tested
  – Can be defined using states, transitions, and operations on inputs, outputs and component variables
  – Possibly defined across a set of communicating components
About Modeling

• Who should write models?
  – The people that write or script tests today!

• What is the starting point?
  – Selecting first aspect of the functionality to be tested
  – Identifying a suitable level of abstraction
Abstraction Level

• Level of detail used to describe functionality to be tested
• Selection from the specification
  – Which interfaces are available for testing selected functionality?
  – What are the relevant inputs & outputs on these interfaces?
  – Extend inputs/outputs with parameters only as you model behavior
• Alternative: Selection from existing test framework
  – Model functions/operations available in test automation (framework) as inputs/outputs in system interface specification
  – Extend inputs/outputs with parameters only as you model behavior
Conformiq’s Modeling Notation (QML)

- Hierarchical UML Statecharts
- Java based action language
- System interface specification
- Rich type system
- Multi component models
- Timing constraints
- Data constraints
Our Task: Testing of a VoIP Terminal

• Subject of Testing: Session Layer
• Specification: RFC 3261 “Session Initiation Protocol”
• Interfaces available: user & network
• Scope: Basic call functionality
  – Call establishment
  – Call termination (callee vs. callee initiated)
  – Call cancelation
  – Call timeouts (re-transmission & transaction)
• See also Conformiq white paper
  – “Case Study: Automated Testing of X-Lite SIP Softphone”
Requirements Extracted from RFC 3261

A SIP User Agent must:

1. Establish a session with SIP ACK (Clause 13.2.2.4)
2. Terminate a session with SIP BYE (Clause 15.1.1)
3. Confirm a SIP BYE with a SIP 200 OK (Clause 15.1.2)
4. Re-send an SIP INVITE after timeout A (Clause 17.1.1.2)
5. Terminate an SIP INVITE after timeout B (Clause 17.1.2.2)
6. Terminate a SIP BYE request after timeout F (Clause 17.1.2.2)
7. Terminate a SIP CANCEL request after timeout F (Clause 17.1.2.2)
Identification of System Boundary

Make a call to...

User Interface (proprietary)

Network Interface (SIP)

SIP Proxy

VoIP Phone

Network

User

SUT

VoIP Phone
Identification of Logical Interface

UserInput:
"call", "hang up", "cancel"

UserOutput:
"ringing", "call ended",
"call established", "timeout"

SipRequest: "BYE"
SipResponse: "180 Ringing", "200 OK", "486 Busy Here", "487 Request Terminated"

SipRequest: "ACK", "BYE", "CANCEL", "INVITE"
SipResponse: "200 OK"
QML Representation of System Interface

```qml
system {  // system block alias system boundary definition
    Inbound  userIn  : UserInput;  // port instance definition with valid message list
    Outbound userOut : UserOutput;
    Inbound  netIn   : SipResponse, SipRequest;
    Outbound netOut  : SipRequest, SipResponse;
}

record SipRequest {  // message type definition (ordered sequence)
    RequestLine startLine;  // field of other structured type (definition elsewhere)
    String callId;
    String contact;
    CSeq cSeq;
    From from;
    int maxForwards;
    To to;
    Via[] vias;  // unbounded list of via headers
    String msgbody;
}
```
Available QML Data Types

- Value types:
  - boolean
  - numbers
  - strings
  - record
  - union

- Reference types:
  - objects
  - arrays
  - ports
  - other objects

- Data types:
  - int (integers)
  - float (rationals)
Modeling Behavior: State Machines

• One way to represent the behavior of a system
• At any moment in time the system is ”in” a state
• Transitions define state changes
  – Can be triggered by an external input or timeout and/or fulfilment of guard conditions
  – When triggered can perform one or more actions such as sending external output(s), marking the coverage of a requirement, or operating on received or component data
  – Actions can also be implemented using methods
Example Statechart: Sip User Agent Client
Example Transition: Call initiation

Paragraph numbers in comments and requirements refer to RFC 3261 SIP: Session Initiation Protocol

21.1.2 180 Ringing
waiting for other end to answer

Triggered by user action, send INVITE request. (We are initiating a new call.)

userIn:UserInput[msg.cmd == "call"]
sendInvite(msg);

netIn:SIPResponse[msg.statusLine.statusCode == 180] /
requireResponse(msg, theINVITE);
remoteTag = msg.to.tag;
sendUserInd("Ringing");
Example QML Method

• Implementation of action to send a SIP INVITE request via the network interface:

```c
void sendInvite() {
    // construct first SIP request value by calling 'getRequestBase' method
    SipRequest theINVITE = getRequestBase("INVITE", newCallId());
    // store 'from' header tag in component variable for later checks
    localTag = theINVITE.from.tag;
    // overwrite contact header and message body default values
    theINVITE.contact = "sip:" + getCallerSipUri();
    theINVITE.msgbody = newMsgBody();
    netOut.send(theINVITE);
}
```
A Requirement in the Model

21.1.2 180 Ringing waiting for other end to answer

It is ringing, but we want to cancel (hang-up) before other end answers

21.4.25 487 Request call is now terminated
Loading the Model
Modeling Summary

- Functional models have structural and behavioral aspects
- System interface is defined via ports and message types
- Handling of inputs and computation of outputs is defined in triggers & actions on Statechart transitions
- Start modeling with a high level of abstraction, e.g., no message parameters, then refine model structure and behavior iteratively
- Model only information required by aspect to be tested
TEST GENERATION
Expectations from Test Generation

- Complete message flow, data, timing, and oracle
- Test generation (regardless of complexity) within minutes
- At least one test per requirement
- Exercise all (possible) SUT source code
- Shortest possible test set execution time
- Do not test "the same thing" multiple times
- Automatic test maintainence
Test Generation at a Second Glance

• Big test sets require a lot of validation
  – There needs to be a way to limit test set size
• Where is the test that covers criteria XYZ?
  – Generation results must be easily comprehensible
• Even at 100% coverage my test is not in the test set
  – Optimal test sets do not always fulfill all expectations
  – Users must be able to affect/guide test generation
But Most Important at the End is …

- ... good support for sharing and communicating test generation results (and models) with testers and others!
From Modeling to Black-Box Testing

(System) Model → Goals → Environment → Real System

Synthesize
# Example Model Coverage Criteria

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Typically Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Coverage</td>
<td>Cover “requirement” statements</td>
<td></td>
</tr>
<tr>
<td>Use Case Coverage</td>
<td>Cover independently specified use cases</td>
<td></td>
</tr>
<tr>
<td>State Coverage</td>
<td>Cover states of every state chart</td>
<td>Basic test generation</td>
</tr>
<tr>
<td>Transition Coverage</td>
<td>Cover transitions of every state chart</td>
<td></td>
</tr>
<tr>
<td>Condition Coverage</td>
<td>Cover “true” and “false” branches of conditional constructs</td>
<td></td>
</tr>
<tr>
<td>Parallel Transition Coverage</td>
<td>Cover all interleavings of independent transitions in multi component models</td>
<td></td>
</tr>
<tr>
<td>Data Coverage</td>
<td>Cover all pairs or all combinations of data values</td>
<td>Extended test generation</td>
</tr>
<tr>
<td>2-Transition Coverage</td>
<td>Cover combinations of entry/exit transitions of all states</td>
<td></td>
</tr>
<tr>
<td>Atomic Condition Coverage</td>
<td>Cover all “true” and “false” evaluations of Boolean expressions</td>
<td></td>
</tr>
<tr>
<td>Boundary Value Analysis</td>
<td>Cover integer boundary conditions</td>
<td></td>
</tr>
<tr>
<td>All Paths - States</td>
<td>Cover all possible states sequences</td>
<td>Exhaustive test generation</td>
</tr>
<tr>
<td>All Paths - Transitions</td>
<td>Cover all possible transition sequences</td>
<td></td>
</tr>
</tbody>
</table>
How does Test Generation work?

• Different technologies available to generate tests
  – Graph traversal vs. symbolic model checking plus constraint solving

• Different approaches how to expose users to results

• What should be expected from a MBT tool today
  – Generation of complete message flow, test data, timing, and oracle
  – Support for specification of coverage criteria to control test suite size
  – Options for generating different types of test sets
  – Incremental & deterministic test generation
  – Automatic marking/elimination of invalid tests in re-generations
Generating Tests from Models
Making Sense of Generation Results

- Integrated graphical IDEs with interconnected views
- Ability to influence test case naming – a first glance should give a good idea
- Traceability matrix linking coverage criteria to tests
- Specific test preview of message flow, test data, and timing
- Highlighting of test and criteria in the (system) model
- Enable analysis of tests and model defects via debugger
Example Test Review
### Traceability Matrix

<table>
<thead>
<tr>
<th>Testing Goals</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Cases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.2.2.4.2.20 Responses</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>15.1 Terminating a session</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.1.1.2.2 INVITE timers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.1.2.2 Non-INVITE timers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State Chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transitions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4 SIPUserAgentClient.Calling -&gt; SIPUserAgentClient.Ringing-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5 SIPUserAgentClient.Calling -&gt; SIPUserAgentClient.final-state-1-3</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6 SIPUserAgentClient.Calling.Wait -&gt; SIPUserAgentClient.Calling initial-state-6-7</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7 SIPUserAgentClient.Calling.junction-state-7-13</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8 SIPUserAgentClient.Calling.junction-state-8-14</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1.9 SIPUserAgentClient.Cancelling -&gt; SIPUserAgentClient.Waiting Response-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.10 SIPUserAgentClient.Cancelling -&gt; SIPUserAgentClient.final-state-2-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1.11 SIPUserAgentClient.Cancelling.Wait -&gt; SIPUserAgentClient.Cancelling initial-state-11-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1.12 SIPUserAgentClient.Cancelling.initial-state-10-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- X indicates the presence of a test case for the corresponding entry.
From Test Generation to Test Execution

- Modeling and test generation is an iterative process
  - Regular reviews of model & tests with stakeholders locate problems prior to test execution
- Integration with automated test execution frameworks is realized via scripting backends
  - Full test script generation in case interfaces of model and test framework match
  - In other cases test scripts with stubs for a generic mapping to a test framework
  - Same mechanism can also generate documentation
- Open API allows creation of own or adaptation of generic backends for any testing framework
Rendering of Tests
## Example Manual Test Report

<table>
<thead>
<tr>
<th>Step</th>
<th>Action(s)</th>
<th>Verification Point(s)</th>
<th>Verdict</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stimulate system via userinput where cmd is &quot;call&quot; and params are &quot;sip:body@127.0.0.1:5061&quot;</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>No action</td>
<td>After 0.5s.: System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Fail</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>No action</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No action</td>
<td>After 2.0s.: System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>No action</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>No action</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>No action</td>
<td>After 0.5s.: System responds on userOut with UserOutput where cmd is &quot;call ended - timeout&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>No action</td>
<td></td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Action(s)</th>
<th>Verification Point(s)</th>
<th>Verdict</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stimulate system via userinput where cmd is &quot;call&quot; and params are &quot;sip:body@127.0.0.1:5061&quot;</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&lt;MsgBody_5&gt;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Stimulate system via netIn with SIPResponse where statusCode is 401 and body is &quot;&quot;.</td>
<td>System responds on userOut with UserOutput where cmd is &quot;cancel call&quot; and params are &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Stimulate system via userinput where cmd is &quot;cancel call&quot; and params are &quot;&quot;.</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Stimulate system via netIn with SIPResponse where statusCode is 202 and body is &quot;&quot;.</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Stimulate system via userinput where cmd is &quot;cancel call&quot; and params are &quot;&quot;.</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>

**Test case 2:** State SIPUserAgentClient, Waiting Response

<table>
<thead>
<tr>
<th>Step</th>
<th>Action(s)</th>
<th>Verification Point(s)</th>
<th>Verdict</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Stimulate system via userinput where cmd is &quot;cancel call&quot; and params are &quot;&quot;.</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>

**Test case 3:** State SIPUserAgentClient, Waiting Response

<table>
<thead>
<tr>
<th>Step</th>
<th>Action(s)</th>
<th>Verification Point(s)</th>
<th>Verdict</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Stimulate system via userinput where cmd is &quot;cancel call&quot; and params are &quot;&quot;.</td>
<td>System responds on netOut with SIPRequest where method is &quot;INVITE&quot; and body is &quot;&quot;.</td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>
Example Test Script with Stubs

```java
module CQ_TestSuite {
  import from CQ_Defaults all;
  import from CQ_TestSystem all;
  import from CQ_Type all;
  import from CQ_TestHarnessTemplate all;

  modulepar float mp_max_response_time := 10.0;

  /*
   * @desc
   * When this module parameter value is set to true all requirements
   * targeted in each test case will be logged.
   * The default value of this module parameter is true.
   */
  modulepar boolean mp_log_targeted_requirements := true;

  /**
   * @desc
   * Text case 'to_Resends_INVITE_after_A_timeout' generated from the Conformiq 'SIP UAC' project
   * runs on CQ_RTC system CQ_TestHarnessSystem
   * log("CQ_DEBUG": Starting execution of test case: 'to_Resends_INVITE_after_A_timeout');
   * var float v_last_wait_time := 0.0;
   * var default v_qg_default;
   * 
   * /******** set up test configuration, ITCH-3 harness, and adapter ********/
   * f_qg_start_test_case();
   * // default handler waiting beyond maximum response time and reception of any
   * // other than the expected message with setting a fail verdict and stopping the test
   * v_qg_default := activate_a_qg_default();
   * 
   * /******** Step 1: t = 0.000 ********
   * log("CQ_DEBUG": to_Resends_INVITE_after_A_timeout Step 1:);
   * f_qg_send_UserInput_to_userIn(UserInputTemplate1);
   * 
   * /******** Step 2: t = 0.000 ********
   * log("CQ_DEBUG": to_Resends_INVITE_after_A_timeout Step 2:);
   * t_qg_timer.stop(); mp_max_response_time := 10.0;
   * Note: In below receive v_qg_default() is active!
   * f_qg_receive_SIPRequest_from_netOut(expectedSIPRequestTemplate);
   * t_qg_timer.stop();
   * 
   * /******** Step 3: t = 0.000 ********
   * log("CQ_DEBUG": to_Resends_INVITE_after_A_timeout Step 3:);
   * f_qg_send_UserInput_to_userIn(UserInputTemplate1);
   */

  /**
   * @desc
   * This function receives a ITCH-3 value corresponding to a SIPResponse
   */
```

© 2011 Conformiq, Inc.
Test Generation Summary

• The ability to generate tests automatically is a good start
  – Equally important however is the ability to control & guide test generation and to be able to understand the purpose of a test
  – Coverage criteria selection is one way of steering test generation

• Model and test review clear up misunderstandings & defects earlier where it is (much) cheaper to fix them
  – Makes test quality much higher during test execution

• Finally tests can then be rendered for test execution to any desired scripting language or documentation format
Thank you for your attention!

stephan.schulz@conformiq.com
www.conformiq.com