

End-to-End Testing Automation in TTCN-3 environment using Conformiq Qtronic™ & Elvior MessageMagic™

Andres Kull, Elvior, andres.kull@elvior.ee
Kullo Raiend, Elvior, kullo.raierend@elvior.ee
Ajay Garg, Conformiq, ajay.garg@conformiq.com

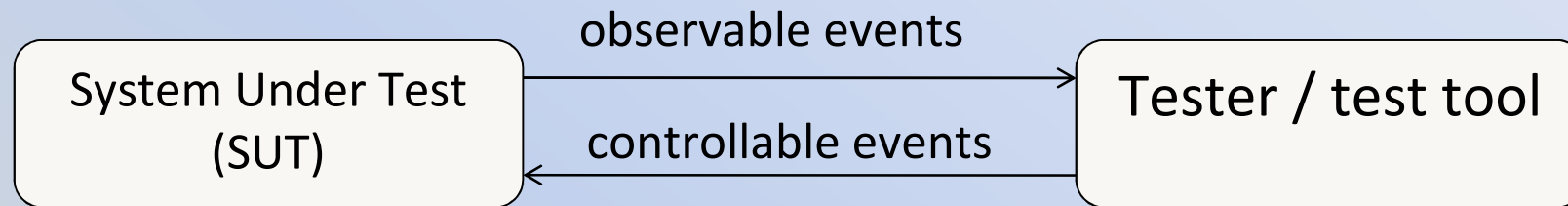
Agenda

- Introduction
- Model-Based Testing & Automated Test Design
- X-Lite softphone testing: A case study
- Test generation and test execution workflow
- SUT (X-Life Softphone) model
- Generating the TTCN-3 test. suite
- Test execution environment
- The value of end-to-end test automation

Introduction

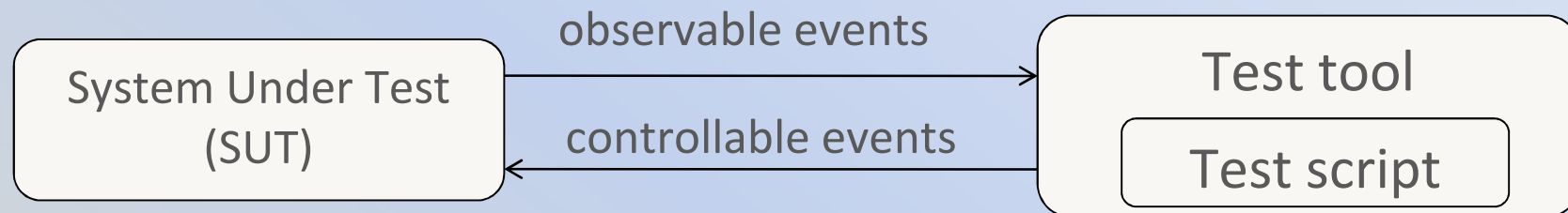
- A Joint case study from Elvior & Conformiq
- Confomiq - model-based automated test design tool provider
 - Conformiq Qtronic™
 - Automatically generates human readable test plans, test cases, and executable test scripts from UML models
- Elvior – TTCN-3 test tool provider
 - MessageMagic
 - TTCN-3 test development and execution platform

Black-box functional testing



- Purpose: to verify that system conforms to its requirements
- Precondition: SUT must be controllable and observable from outside
- SUT examples:
 - controllers in telecommunications, automotives, avionics,...
 - software embedded in smart gadgets
 - web applications
 - software subsystems on any system hierarchy levels

Automated black-box functional testing

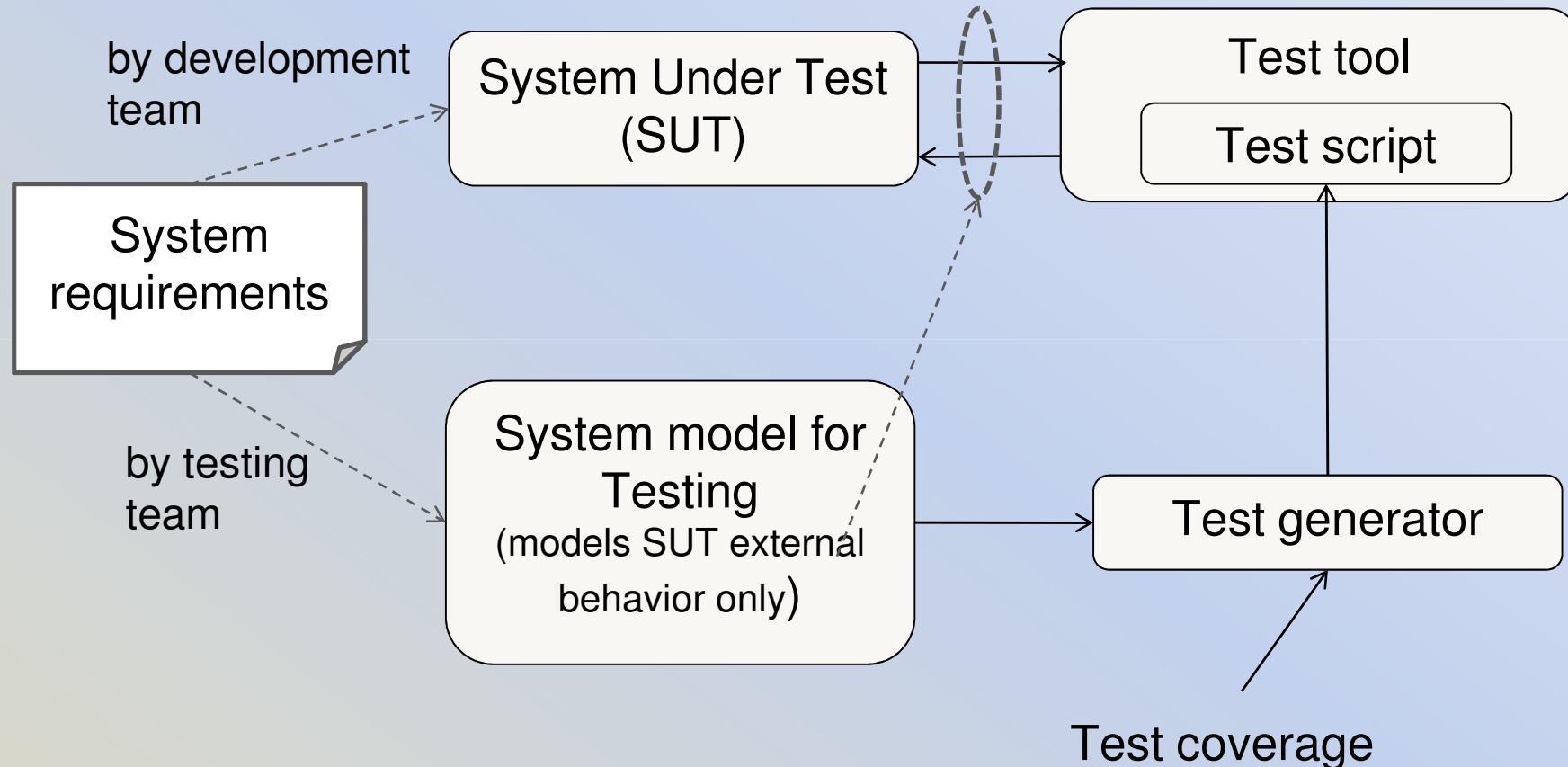


- Test is controlled by test script
- Suitable test types:
 - Functional Tests
 - Regression tests
 - Conformance tests
 - Load tests
 - Smoke tests

Problems with manual test scripting

- Tests creation or design phase:
 - Writing of test scripts is time consuming
 - Poor test coverage
 - Missing Test Cases
 - Error Prone Tests
 - Often only test cases for basic (“normal”) behaviour are automated
- Tests maintenance phase:
 - Number of test scripts grows over time
 - Requirement or Function Changes requires going through entire test suite to make changes
 - Overall very high maintenance costs

Model-Based Testing in a nutshell



Model-based testing benefits

- Tests creation or design phase:
 - Models are created instead of scripts
 - Good systematic test coverage
 - Tests are free of bugs
- Tests maintenance phase:
 - Models are maintained instead of tests
 - Requirement or Function Changes requires going through relevant models to make changes
 - Acceptable maintenance costs

X-Lite softphone testing case study

Test generation and execution workflow



Generating TTCN-3 Test suite

The screenshot displays the Qtronic Eclipse SDK interface for generating TTCN-3 test suites. The main window is titled "Qtronic - Coverage Goals for UAC_TP - Eclipse SDK".

Project Explorer: Shows the project structure for "UAC_TP", including "DC 1", "Generated TTCN-3", "QtronicTypes.ttcn3", "TestSuite.ttcn3", "model", "SIPClient.java", and "SIPClient.xmi".

Coverage Editor: UAC_TP: Displays testing goals and their coverage status for "DC 1".

Testing Goals	DC 1
Call establishment	✓ 100
Timers	✓ 100
TPId: SIP_CC_OE_CE_TI_001	✓ ✓
TPId: SIP_CC_OE_CE_TI_003	✓ ✓
TPId: SIP_CC_OE_CE_TI_004	✓ ✓
Valid Behaviour	✓ 100
TPId: SIP_CC_OE_CE_V_005	✓ ✓
TPId: SIP_CC_OE_CE_V_009	✓ ✓
TPId: SIP_CC_OE_CE_V_010	✓ ✓
TPId: SIP_CC_OE_CE_V_011	✓ ✓
TPId: SIP_CC_OE_CE_V_012	✓ ✓
TPId: SIP_CC_OE_CE_V_013	✓ ✓
TPId: SIP_CC_OE_CE_V_014	✓ ✓
Call release	✓ 100
State Check	✓ 100

Traceability: DC 1: Shows a table of requirements and their coverage status.

Requirements	1	2	3	4	5
Requirements					
17.1.1.2 INVITE timers					
Terminates INVITE cycle after B timer				X	
17.1.2.2 Non-INVITE timers					
Resends CANCEL after E timeout					X
Terminates CANCEL cycle after F timer					X
Terminating					
Send OK in response to BYE					
Wait for OK in response to BYE					
Call establishment					
Timers					
TPId: SIP_CC_OE_CE_TI_001				X	
TPId: SIP_CC_OE_CE_TI_003				X	
TPId: SIP_CC_OE_CE_TI_004				X	
Valid Behaviour					

Test Cases: UAC_TP > DC 1: Lists 11 test cases, all with a coverage of 200%.

Name	Cre
1 Test Case 1	200%
2 Test Case 2	200%
3 Test Case 3	200%
4 Test Case 4	200%
5 Test Case 5	200%
6 Test Case 6	200%
7 Test Case 7	200%
8 Test Case 8	200%
9 Test Case 9	200%
10 Test Case 10	200%
11 Test Case 11	200%

Test Case 11: Shows a sequence diagram between "Tester" and "SIP UAC".

```

sequenceDiagram
    participant Tester
    participant SIP_UAC as SIP UAC
    Note over Tester: t=0,0
    Tester->>SIP_UAC: UserInput -> userIn
    Note over SIP_UAC: t=0,0
    SIP_UAC->>Tester: SIPReq <- netOut
    Note over Tester: t=0,0
    Note over Tester: TPId: SIP_CC_OE_CE_V_005
    Tester->>SIP_UAC: SIPResp -> netIn
    Note over Tester: t=0,0
    Tester->>SIP_UAC: UserInput -> userIn
    Note over Tester: t=0,0
    SIP_UAC->>Tester: SIPReq <- netOut
    
```

Steps: Test Case 11: Shows the execution flow of the test case.

```

graph TD
    subgraph main
        direction TB
        A[SIPClient.SIPClient()] --> B[SIPClient.run()]
        B --> C[SIPClient-initial-4]
        C --> D[SIPClient-Init]
    end
    D --> E[SIPClient.Invite()]
    E --> F[SIPClient-Calling-initial-0]
    
```

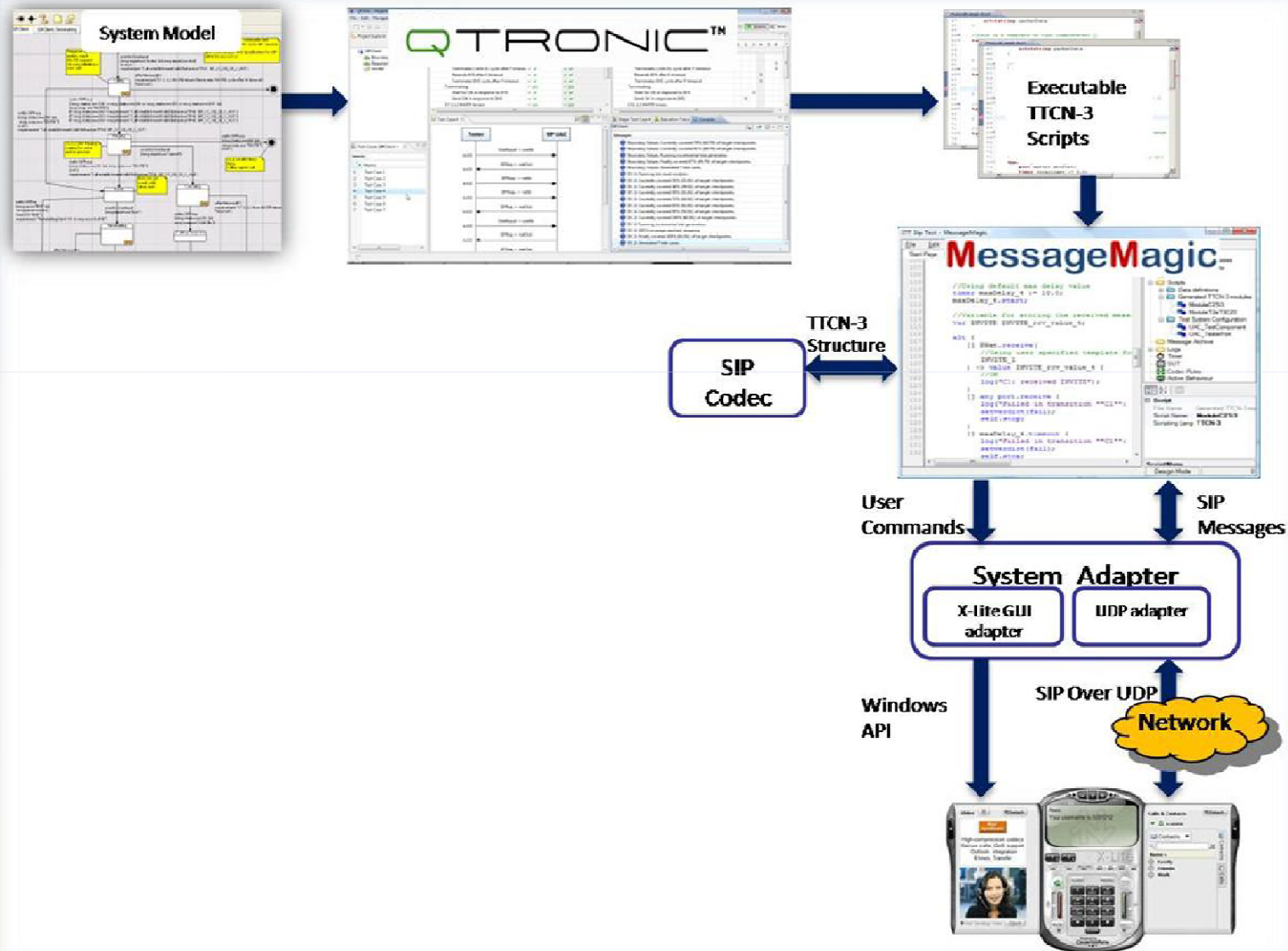
Generated TTCN-3 test suite

```

module TestSuite
{
  import from QtronicTypes all;
  /* User provided imports begin */
  import from QtronicTestHarness all;
  /* User provided imports end */
  /* Qtronic generated alt step */
  altstep QtronicDefaultAlt() runs on QtronicMTC
  {
    [] any port.receive
    {
      harnessTimer.stop;
      setverdict(fail);
      qtronic_end_test_case();
      stop;
    }
    [] harnessTimer.timeout
    {
      setverdict(fail);
      qtronic_end_test_case();
      stop;
    }
  }
}
/* Generated test case #1 */
testcase Test_Case_1() runs on QtronicMTC system QtronicHarnessSystem
{
  var float oldtimer := 0.0;
  var float SLACK := 10.0;
  var default default_behaviour_ref;
  qtronic_start_test_case();
  default_behaviour_ref := activate(QtronicHarnessAlt());
  log("Structural feature: method: main()");
  log("Structural feature: method: SIPClient.SIPClient()");
  log("Structural feature: method: SIPClient.run()");
  log("Structural feature: state: SIPClient-initial-4");
  log("Structural feature: transition: SIPClient-initial-4->SIPClient-Init-9");
  log("Structural feature: state: SIPClient-Init");
  qtronic_send_UserInput_to_userIn(UserInputTemplate1);
  oldtimer := 0.0;
  log("Structural feature: transition: SIPClient-Init->SIPClient-Calling-initial-0-0");
  log("Structural feature: method: SIPClient.Invite()");
  harnessTimer.start((0.0 - oldtimer) + SLACK);
  qtronic_receive_SIPReq_from_netOut(SIPReqTemplate2);
  harnessTimer.stop;
  oldtimer := 0.0;
  log("Requirement: requirement: Call establishment/Valid Behaviour/IPid: SIP_CC_OE_CE_V_005");
}

```

Test Execution Environment



Value of end-to-end- test automation

- Direct link between design and quality assurance
 - system models are important testing assets
 - reduces test maintenance costs (models are easier to change than the actual test scripts)
- Covers the whole test process
 - from the system modelling
 - to the tests execution against the SUT and test results evaluation
- Modelling, test generation and test execution tools from different providers can form an quality assurance process
- Automatically generated test scripts can be stored in configuration management / version control systems, and they can be executed Independent of the test generation system
- Case study shows how model-based end-to-end test automation can be employed in TTCN-3 driven quality assurance process using the tools available from Conformiq and Elvior

More Information

www.elvior.com

www.conformiq.com

Andres Kull, Elvior, andres.kull@elvior.ee
Kullo Raiend, Elvior, kullo.raierend@elvior.ee
Ajay Garg, Conformiq, ajay.garg@conformiq.com