SIMPL-T:

SDL Intended for Management and Planning of Tests

By

Qing Li, Robert Probert, William Skelton, Yiqun Xu

SAM’04 Workshop, Ottawa, Canada

June, 2004
Outline

- Background and Motivation
- Approach
- Assessment
- Contributions and Future Work
Background and Motivation

Relative Error Correction Cost in a Software Life Cycle

Specification must be tested!
No Existing Formal Language is Suitable for Testing SDL Specifications

- TTCN
- MSC
- UML
- URN/UCM
- LOTOS
- SDL
Background and Motivation

SDL Task Force

- The graphical representation, ensuring auto-layout is possible
- **Test capabilities, such as SDL based test scripts**
- ASN.1(1994) support, including encoding/decoding of PDUs
- Associated methodology issues, such as maximum integration of tool chain
Statement of Research Problem

To define and investigate the applicability of a simple, useful and efficient language for describing tests of SDL specifications

SIMPL-T
-- SDL Intended for Management and Planning of Tests
Approach

- Basic Testing Concepts
- Key Requirements
- Suitability of SDL for Test Specification
- SIMPL-T – SDL with Extensions
SDL & TTCN Overlap

Approach

Extensions to SDL
Test Architecture (ITU-T Z.500)

Basic Testing Concepts
Test Architecture (ISO 9646)

* Test Configuration
Test Architecture

- Tester – Run test suite
- IUT
- Connection – PCOs
- Communication Channels
Basic Testing Concepts

Test Case and Test Suite

• Test Suite

• Test Case
  – Test Purpose
  – Test Case Behaviour
    • Sending a stimulus to the IUT
    • Specifying expected response
    • Store and Transfer data
    • Take alternative actions
    • Repeated test steps or actions
Observations

- Check the responses
- Measure the timing of response
- Assign Verdict
Key Requirements

- Test Architecture – Tester and SUT
- Connection between the Tester and the SUT
- Communication between the Tester and the SUT
- Organization and Management of Tests
- Sending Stimuli to the IUT
- Receiving Response from the IUT
- Storing and Transferring Data
- Flow Control
- Test Step Repetition
- Checking Responses and Matching Mechanism
- Measuring the Timing of Responses.
- Assigning and Handling of Verdict
# Suitability of SDL for Test Specification

<table>
<thead>
<tr>
<th>Key Requirements</th>
<th>SDL Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Architecture – Tester, SUT &amp; Test Context</td>
<td>SDL Blocks</td>
</tr>
<tr>
<td>Test Architecture - Connecting between Tester and SUT (PCOs &amp; IAPs)</td>
<td>Gate &amp; Channel</td>
</tr>
<tr>
<td>Test Architecture - Communication between Tester and SUT</td>
<td>Signal Exchange</td>
</tr>
<tr>
<td>Organization and Management of Tests</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Test Case Behaviour - Sending Stimuli to SUT</td>
<td>Output</td>
</tr>
<tr>
<td>Test Case Behaviour – Receiving Responses from SUT</td>
<td>Input</td>
</tr>
<tr>
<td>Test Case Behaviour - Storing and Transferring data</td>
<td>Variable &amp; Data Type</td>
</tr>
<tr>
<td>Test Case Behaviour – Flow Control</td>
<td>Decision</td>
</tr>
<tr>
<td>Test Case Behaviour - Test Step Repetition</td>
<td>Procedure</td>
</tr>
<tr>
<td>Observation - Checking Responses</td>
<td>Partially Supported</td>
</tr>
<tr>
<td>Observation - Measuring the Timing of Responses</td>
<td>Timer</td>
</tr>
<tr>
<td>Assigning and Handling of Verdicts</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>
Approach

Extensions

- Organization and Management of Tests
- Checking Responses
  -- “Input Via” and Matching mechanism
- Assigning and Handling of Verdicts
Organization and Management of Tests

Testsuite_Definition ::= "TESTSUITE" TestsuiteName ";"
    [ Gate_Definition ]
    [ Testsuite_Component ]
"ENDTESTSUITE;"

Gate_Definition ::= "GATE" GateName ";"
    [ In_Signal_List ] ";"
    [ Out_Signal_List ] ";

In_Signal_List ::= Signal_Identifier
    [ "," In_Signal_List ]

Out_Signal_List ::= Signal_Identifier
    [ "," Out_Signal_List ]

Testsuite_Component ::= ([Signal_Definition]
    [Signal_List_Definition]
    [Timer_Definition]

......
    [Test_Group_Definition]
    [Test_Case_Definition] )
    [Testsuite_Component ]

Test_Group_Definition ::= "TESTGROUP" TestGroupName ";"
    Test_Case_Definition_List

......
New INPUT VIA A Construct

STATE S1;
  INPUT A VIA Gate1;
NEXTSTATE S2;
Specifying Expected Values of Parameters inside INPUT

Test case 5

Start

Password

S

Display("Please Come In", "Invalid Password! Please try again")

Pass

Fail

SAM'04
Matching Mechanism

- Unmatched Signal Handling:
  - Disregard by default
  - Explicitly use “Save” construct when necessary
Matching Mechanism

• Overlapped Signal Handling:
  (1) the same signal arriving from different gates/channels;
    
    \[ Y \text{ via GateA} \quad \quad Y \text{ via GateB} \]
    
    \[ Y(2) \quad \quad Y(1:100) \]
    
    -- > They are not considered as overlap in SIMPL-T
  
  (2) the parameters carried by the same signal have different values and the values have overlap

  
    \[ Y(2) \quad \quad Y(1:100) \]
    
    -- > They are not allowed in SIMPL-T
An Example of a SI MPL-T Test Case

Test_case 6

Start

Startup

B

SET(NOW+5, Lower)

SET(NOW+9, Upper)

SS

Lower

Z(1:100) Via Gate1

Upper

Otherwise

DCL
V Boolean := False;

TIMER
Lower := 5;
Upper := 9;

Preliminary Result

Final Verdict

pass

V=True

pass

fail

fail

inconc

V=Tru

False

pass

fail

fail

SAM'04
Approach

**An Example of a SIMPL-T Test Case (Cont.)**

![Flowchart]

**Procedure Startup**

1. Start
2. Reset
3. SET(NOW+4, ResetTimer)
4. S
5. ResetACK
6. ResetTimer
7. * (Inconce)

- **Pass**
- **Fail**
- **Inconce**
The Strengths and Limitations of SIMPL-T Comparing to TTCN

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>SIMPL-T</th>
<th>TTCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Easy to Learn for SDL Users</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Simplicity of Test Suites</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Support Reusability -- Efficiency and Reliability</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Tool Support</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Handle Ordering Problem</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Handle Race Condition</td>
<td>==</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Handle Concurrency</td>
<td>==</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Scalability</td>
<td>-+</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Number of Available Features</td>
<td>-+</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Support testing non-SDL Specifications</td>
<td>-+</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Undefined-object (Exception) Handling</td>
<td>/+</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Structure Type Support &amp; Access Components of Complex Types</td>
<td>/+</td>
<td></td>
</tr>
</tbody>
</table>
Ordering Problem

- Two or more signals can arrive in arbitrary order
- The order is irrelevant,
- The test language does not have a mechanism to specify this situation

**SIMPL-T**

-- solve it using “save” construct
Contributions

- Submitted to the SDL Task Force
- Defined a simple, easy to learn test language
- Create a potential for lower cost tools
- Lead to more interest in SDL and testing
Future Work

- Concurrency
- Defaults
- Extensions for larger applications