Evolution of testing techniques: from active to passive testing

Ana Cavalli
TELECOM SudParis
August 28-2012
Planning

- Conformance testing
- Active testing techniques
- Fault models
- Control and observation
- Passive testing (Monitoring)
- Tool and case study (DIAMONDS project)
Conformance testing

- Conformance testing:
  - to check that an implementation conforms to a specification

- Faults detected:
  - output faults, if the implementation transition produce a wrong output
  - transfer faults, if the implementation transition go in a wrong state of the machine
  - mixtes faults, both output and transfer faults
What is active testing?

- It is assumed that the tester controls the implementation
- Control means: after sending an input and after receiving an output, the tester knows what is the next input to send
- The tester can guide the implementation towards specific states
- Automatic test generation methods can be defined
Formal methods for test generation

• Objectives
  
  - To optimize tests production by reduction of time and cost
    
    • An engineer produces three handcrafted tests per day
    
    • A test suite of a real protocol is composed of an average of 800 tests
  
  - To improve faults coverage
Fault Models

SPECIFICATION

IMPLEMENTATION

1  i1 / o1  2

1  i1 / o2
output fault

1  i1 / o1
transfer fault

1  i1 / o2
mixte fault
Example: soda vending machine (Lamport’s example)
Example: Soda Vending Machine

Specification

2€ / OK

Choice / Soda, Juice

1€ / another 1€

output fault

2€ / another 1€

Choice / Soda, Juice
Example: Soda Vending Machine

I2

2€ / OK

1€ / another 1€

1€ / OK

transfer fault

Choice / Soda, Juice

I3

1€ / another 1€

Choice / Soda, Juice

1€ / OK

2€ / OK
Definition of a test

• Step 1 : Put the finite state machine implementation into $S_i$

  (Drive the protocol implementation into the head state of the transition to be tested)

• Step 2 : Apply input, observe output

  (Apply the input corresponding to the transition to be tested and check that the output is as expected)

• Step 3 : Verify $S_j$

  (Verify that the new state of the finite state machine is the same as the tail state of the transition being tested)
Controllability issue in active testing

• How to bring the finite state machine implementation into any given state at any given time during testing ? (Step 1)
  – Non trivial problem because of limited controllability of the finite state machine implementation.
  – It may not be possible to put the finite state machine into the head state of the transition being tested without realizing several transitions.
Controllability: examples

Specification

Controllable

Imp1

Non controllable

Imp2

Non controllables

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp1

Imp2

Non controllable

Non controllables

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption

Controllable

a/b

Non controllable

a/b

Imp2

Non controllable

a/b

Controllable under fairness assumption
Observability issue in testing

• How to verify that the finite state machine implementation is in a correct state after input/output exchange? (Step 3)
  – State identification problem. Difficult because of limited observability of the finite state machine implementation, it may not be possible to directly verify that the finite state machine is in the desired tail state after the transition has been fired.
Solutions to observability issue

• To solve this problem different methods have been proposed:
  – DS (Distinguishing Sequence Method);
  – UIO (Unique Input/Output Sequence Method);
  – W (Distinction Set Method).
Unique Input/Output sequence (UIO sequence)

- Find an input sequence for each state such that the output sequence generated is unique to that state
- Detects output and transfer faults.

<table>
<thead>
<tr>
<th>State</th>
<th>UIO sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>c/x</td>
</tr>
<tr>
<td>S2</td>
<td>c/y</td>
</tr>
<tr>
<td>S3</td>
<td>b/y</td>
</tr>
</tbody>
</table>

Test of (1): a/y a/x b/y
Test of (2): a/y c/z b/y
Transfer and output error detection

Test of (1): a/y a/x b/y
Test of (2): a/y c/z b/y

Application du test of (1) to the implementation: a/y a/x b/z (transfer error)
Application of test (2) to the implementation: a/y c/x (output error)
Limitations of active testing

• Non applicable when no direct access to the implementation under test
• Non controllable interfaces
• Interference on the behaviour of the implementation
• Example: components testing
Components Testing

- Test in context, embedded testing
  - Tests focused on some components of the system, to avoid redundant tests
  - Interfaces semi-controllables
  - In some cases it is not possible to apply active testing
Why passive testing?

• Conformance testing is essentially focused on verifying the conformity of a given implementation to its specification
  – It is based on the ability of a tester that stimulates the implementation under test and checks the correction of the answers provided by the implementation

• Closely related to the controllability of the IUT
  – In some cases this activity becomes difficult, in particular:
    • if the tester has not a direct interface with the implementation
    • or when the implementation is built from components that have to run in their environment and cannot be shutdown or interrupted (for long time) in order to test them
Test by invariants: principle

• Definition: an invariant is a property that is always true

• Two step test:
  – extraction of invariants from the specification or proposed by protocol experts
  – application of invariants on execution event traces from implementation

• Solution: I/O invariants
Test by invariants: I/O invariants

• An invariant is composed of two parts:
  – the test (an input or an output)
  – the preamble (I/O sequence)

• 3 kind of invariants:
  – output invariant (simple invariant)
  – input invariant (obligation invariants)
  – succession invariant (loop invariants)
Test by invariants: Simple (Output)

- Definition: invariant in which the test is an output
- Meaning: « immediately after the sequence *préambule* there is always the expected output »
- Example:

\[(i_1 / o_1) (i_2 / o_2)\]

(preambule in blue, expected output in red)
Test by invariants: Obligation (Input) invariant

- **Definition:** invariant in which the test is an input.
- **Meaning:** « immediately before the sequence *preamble* there is always the input *test* »
- **Example:**

  \[(i_1 / o_1) (i_2 / o_2)\]

  (preamble in blue, test in red)
Test by invariants : succession invariant

• Definition : I/O invariant for complex properties (loops …)

• Example :
  – the 3 invariants below build the property :
    « only the third \( i_2 \) we meet is followed by \( o_3 \) »

\[
(i_1 / o_1) (i_2 / o_2) \\
(i_1 / o_1) (i_2 / o_2) (i_2 / o_2) \\
(i_1 / o_1) (i_2 / o_2) (i_2 / o_2) (i_2 / o_3)
\]
SIMPLE INVARIANT

• Simple invariant
  – A trace as $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/O$ is a simple invariant if each time that the trace $i_1/o_1, \ldots, i_{n-1}/o_{n-1}$ is observed, if we obtain the input $i_n$ then we necessarily get an output belonging to $O$, where $O$ is included in the set of expected outputs.
  – $i/o, *, i'/O$ means that if we detect the transition $i/o$ then the first occurrence of the symbol $i'$ is followed by an output belonging to the set $O$.
  – * replaces any sequence of symbols not containing the input symbol $i'$ and $?$ replaces any input or output.
Example

Invariants | Verdicts
---|---
a/?, c/z, b/{y} | True
b/z, a/{x} | False
a/x, *, b/{y, z} | True
a/y, ?/{z} | False
a/{x} | False
a/x, *, ?/{y} | True

Traces
a/x c/x a/y a/x c/z b/y
c/x a/y a/x c/z b/y
c/y a/x b/z b/x a/y
PASSIVE vs ACTIVE TESTING

**Pros & Cons:**

**Passive Testing:**
- Possibility to focus on a specific part of the specification
- Automatic test generation
- May modify (crash) the IUT behavior

**Active Testing:**
- No interferences with the IUT
- Test of components
- System user

**Verdict:**
- PASS, FAIL, INCONC.
DIAMONDS PROJECT
A tool and a case study
Montimage Monitoring Tool

- User defined reports and views
- Add analysis modules
- HW/SW Probe: Can be installed on dedicated HW
- Software library (SDK): Can be integrated in 3rd party SW
- Add plugins

Modular solution
MMT in a transport network scenario
MMT characteristics

- Use of security properties to describe both wanted and unwanted behaviour
  - Not exclusively based on pattern matching like most intrusion detection techniques
  - More abstract description of sequence of events (MMT properties)
  - Can integrate performance indicators, statistics and machine learning techniques; as well as countermeasures
- Allows combining centralised and distributed analysis to detect 0-day attacks (under development)
- Applicable in several domains (at protocol, application and business levels)
- Allows combining active and passive approaches
Composing Active and Passive Testing

- Security Requirements
- Risk Analysis
- Security Test Purposes
- Security Test Engineer
- Security Properties
- Test Generation
- Security Test Generation Model
- Security Tests
- Defects
- Monitoring
- SUT
A security property is composed of 2 parts:
- A Context
- A Final condition (trigger)

The “Context” and “Trigger” are composed of:
- Events (based on temporal logic)
  - Simple events
  - Complex events linked by logical operators (AFTER/BEFORE/AND/OR)
  - Time constraints, message order, occurrence or non-occurrence, repetition, combination
- A simple event is composed of:
  - Attributes (values of packet fields, values of sessions attributes, time of reception, length of message, statistics …)
  - Conditions on attributes (IP @ equal to 132.3.4)
Radio Protocol case study

- Provided by Thales: definition of ad-hoc « networking » protocols and algorithms
  - High Data Radio Network Wave Form

- Technical challenges
  - Automatic network: no initial planning
  - Network continuity whatever are the stations in the network
  - “On the move” automatic network re-organization and operation
  - End-to-end heterogeneous user services transmission: voice, messages
  - Decentralized mesh network. No infrastructure, e.g. base stations
Thales Case study
Security rules specification

**Threat:** Deny of service by flooding of RLC_CL_UNIT_DATA_ACK messages

**Security property:** A message RLC_CL_UNIT_DATA_ACK must be preceded with a message RLC_CL_UNIT_DATA_REQ that asked for acknowledgement (R == 00010000) (correlation with the USER_TRANSACTION_ID)

BEFORE

<table>
<thead>
<tr>
<th>RLC_CL_UNIT_DATA_ACK message</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE.PROTO == 5152 &amp;&amp;</td>
</tr>
<tr>
<td>MSG RLC_CL_UNIT_DATA_ACK.</td>
</tr>
<tr>
<td>USER_TRANSACTION_ID ==</td>
</tr>
<tr>
<td>MSG RLC_CL_UNIT_DATA_REQ.</td>
</tr>
<tr>
<td>USER_TRANSACTION_ID.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RLC_CL_UNIT_DATA_REQ message that asked for acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE.PROTO == 1056 &amp;&amp;</td>
</tr>
<tr>
<td>MSG RLC_CL_UNIT_DATA_REQ.</td>
</tr>
<tr>
<td>QOS_R == 128</td>
</tr>
</tbody>
</table>
A set of 20 security properties have been specified and checked by Montimage

- Detection of several errors due to a bad generation of traces (using OMNET) and online detection done
- More properties (~50) are in the design phase
Other works

• Passive testing with time constraints and parameters
• Distributed passive testing
• Differents applications domains (communication and routing protocols, web services)
REFERENCES


