A Generic Framework for implementing TTCN-3 Logging Modules

Anthony Baire, Radu Muresan, César Viho

November 2009

TTCN-3 Asia User Conference
TTCN-3 and Test Logging

- The language is well-formalised
  - most of available tools provide a generic logging module
  - test developers do not have to implement logging
Why do we need specialised logging modules?

• Some examples:
  – to describe the content of the messages
  – to use a specific presentation format
  – to reconstruct a session from different packets
Example 1: describing a message
Existing solution in LibSip

type record Request
{
  RequestLine requestLine,
  MessageHeader msgHeader,
  MessageBody messageBody optional,
  Payload payload optional
}

Generic type definition for a SIP Request

14 similar definitions for each variant of SIP Request

« the introduction of the specific types is to enable better means for logging »
Example 1: log traces

<table>
<thead>
<tr>
<th>using a generic type</th>
<th>using multiple types (workaround)</th>
<th>our objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>INVITE_Request</td>
<td>INVITE Request</td>
</tr>
<tr>
<td>Response</td>
<td>Response</td>
<td>100 Trying</td>
</tr>
<tr>
<td>Response</td>
<td>Response</td>
<td>180 Ringing</td>
</tr>
<tr>
<td>Response</td>
<td>ACK_Request</td>
<td>ACK_Request</td>
</tr>
<tr>
<td>Request</td>
<td>Response</td>
<td>200 OK</td>
</tr>
<tr>
<td>Response</td>
<td>BYE_Request</td>
<td>BYE_Request</td>
</tr>
<tr>
<td>Request</td>
<td>Response</td>
<td>200 OK</td>
</tr>
</tbody>
</table>
Example 2: presentation format

• How to represent an IP Address in TTCN-3?

  - octetstring  ➞  'C0A8002A'O
  - bitstring    ➞  '1100000010101000 00000000'
  - integer      ➞  3232235562
  - charstring   ➞  "192.168.0.42"
  - record of    ➞  {192, 168, 0, 42}
  - record       ➞  {a:=192, b:=168, c:=0, d:=42}

Preferred format for **manipulating** IP addresses

Preferred format for **presenting** IP addresses
Example 3: reconstruct a conversation

- A useful feature *wireshark* provides

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source 131.254.254.45</th>
<th>Destination 131.254.254.45</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.002000</td>
<td>131.254.14.21</td>
<td>131.254.254.45</td>
<td>TCP</td>
<td>55309 &gt; ftp [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=560846 TSER=</td>
</tr>
<tr>
<td>2</td>
<td>0.002004</td>
<td>131.254.254.45</td>
<td>131.254.14.21</td>
<td>TCP</td>
<td>ftp &gt; 55309 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=</td>
</tr>
<tr>
<td>3</td>
<td>0.002392</td>
<td>131.254.14.21</td>
<td>131.254.254.45</td>
<td>TCP</td>
<td>55309 &gt; ftp [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=560847 TSER=47</td>
</tr>
<tr>
<td>5</td>
<td>0.003338</td>
<td>131.254.14.21</td>
<td>131.254.254.45</td>
<td>TCP</td>
<td>55309 &gt; ftp [ACK] Seq=1 Ack=35 Win=5888 Len=0 TSV=560847 TSER=</td>
</tr>
<tr>
<td>10</td>
<td>11.960012</td>
<td>131.254.14.21</td>
<td>131.254.254.45</td>
<td>FTP</td>
<td>Request: PASS <a href="mailto:someone@nowhere.com">someone@nowhere.com</a></td>
</tr>
</tbody>
</table>

Stream Content:

```
220 Welcome to IRISA FTP service.
USER anonymous
331 Please specify the password.
PASS someone@nowhere.com
230 Login successful.
SYST
215 UNIX Type: L8
CWD /pub/OpenBSD
250 Directory successfully changed.
PASV
227 Entering Passive Mode (131,254,254,45,76,191)
LIST
150 Here comes the directory listing.
226 Directory send OK.
TYPE I
200 Switching to Binary mode.
PASV
227 Entering Passive Mode (131,254,254,45,74,144)
RETR README
```
How to implement a logging module for a TTCN-3 test suite?

- Use the TCI-TL (*TTCN-3 Control Interface*)
  
- a standard interface

- it reports all events in the test execution (eg. message sent)

- usable in C/Java/XML
Implement the TCI-TL interface... at what cost?

- 100+ functions to be provided by the TL module
- ~9 parameters in each function
- Flat design: one event ➞ one TL function
- TCI-TL is still evolving

→ a huge task... and this just to support the TL interface
The TL Dilemma

– Most of the tasks of a TL module are generic
  • display the events, draw sequence charts
  • load/save the logs on the disk
  • provide a GUI

– A tiny part in the TL is specific to the actual test suite
  • describe the content of the messages
  • present a type in a specific format

→ Choosing between a generic logger or a home-made logger is an all-or-nothing situation
Objectives for our TL Framework

– provide a generic representation for TL events
  • easy to define
  • easy to update (futures changes in the standard)
  • easy to browse (without knowing the their structure)

– provide means for storing the logs

– allow to implement new back-ends independently
Example of TCI-TL functions

void tliTcStop
(void tliTcStop (String am, long ts, String src, long line, TriComponentId c))

void tliLog
(void tliLog (String am, long ts, String src, long line, TriComponentId c, String log))

void tliPMap
(void tliPMap (String am, long ts, String src, long line, TriComponentId c, TriPortId port1, TriPortId port2))

event-specific parameters

common parameters
Representation of Events

Event

Event0
Event1<T>
Event2<T,U>
...
Event10<...>

EventTliTcStop
EventTliAEnter
EventTliLog
EventTliPMap
EventTliPrReply_m
TL Module layout

- TCI TL implementation
- Event Publisher
  - Console Back End
  - XML Back End
  - possible future backends
    - GUI
    - HTML report

Generation of Event
Advantages of the framework

• A generic *Event* format
  – Backend implementation is easier
    • no need to know about all possible events
    • insensitive to future changes in the TCI standard

• XML format for storing event
  – standard format → good interoperability
  – ability to convert old log files to future TL formats using XSL transformations → good durability
Future tasks

• Define interfaces to allow:
  – implementing specific presentation formats
  – analysing the messages and generating a description

• Implement a GUI
Conclusion

• Specialised Test Loggers are useful

• TCI-TL does not facilitate the development of modular & reusable TL modules

• The proposed framework aims to solve this issue
  – Modular design
  – Insensitive to future API changes
  – Open source
Questions ?

Contacts: abaire@irisa.fr / viho@irisa.fr

The logging framework will be distributed in future releases of T3DevKit

→http://t3devkit.gforge.inria.fr/