Protocol Conformance Testing

Prof Pallapa. Venkataram,
Department of Electrical Communication
Engineering
Indian Institute of Science
Bangalore – 560012, India
Conformance Testing

- Conformance Testing is process of testing the extent to which implementations of protocol entities adhere to the requirements stated in the relevant standard or specification.

- Problems solved by Conformance Testing
  - Finding a generally applicable, efficient procedure for generating a conformance test suite for a given protocol implementation.
  - Finding a method for applying the test suite to a running implementation.

- Conformance Testing Requirements
  - mandatory requirements
  - conditional requirements
  - prohibitions and optional requirements.
Conformance Testing Methodology & Framework

- Part 1: protocol concepts,
- Part 2: test suite specification and test system architectures,
- Part 3: test notation,
- Part 4: test realization,
- Parts 5, 6 and 7: means of testing and organizational aspects.
Conformance Test Architecture

- IUT (Implementation Under Test)
- PCO (Point of Control and observation)
- ASPs (Abstract Service Primitives)
- PDUs (Protocol Data Units)
- LT (lower Tester)
- UT (Upper Tester)
- Test Coordination Procedures (TCPs)
Conformance Test Architectures

- Local test method,
- Distributed method,
- Remote method,
- Coordinated method, and
- distributed combined with local methods.
Local conformance test architecture

Upper Tester
(Nt) - ASPs

Implementation Under Test

N-PDUs
(Nb - 1) - ASPs

Lower Tester

Test Coordination Procedures
Local conformance test architecture - Example

System under test (SUT)

PCO

ASPs

IUT

2) CONind

3) CONresp

Test system (TS)

Upper Tester

TCPs

Lower Tester

ASPs

PCO

1) CR

4) CC

Underlying service

Figure 6.5: Example of local test method
Distributed conformance testing architecture
Distributed conformance testing architecture - Example

: Example of distributed test method architecture
Coordinated Method

Lower Tester

Test Management Procedures

Upper Tester

Implementation Under Test

N-PDUs

Test Management Procedures

TM PDUs

(Nb -1) – ASP’s

(N-1) Service Provider

Coordinated conformance testing architecture
Coordinated Method - Example

System under test (SUT)

2) CONind  3) CONresp

Upper Tester

IUT

Test system (TS)

Lower Tester

Test management protocol

1) CR

4) CC

Underlying service

Example of coordinated test method
Remote Method

Test Coordination Procedures

Remote conformance testing architecture

(Nb -1) – ASP”s

(N-1) Service Provider

Lower Tester

Implementation Under Test

N-PDUs
Remote Method - Example

System under test (SUT)

IUT

Test system (TS)

Lower Tester

ASPs

PCO

Remote method example

1) CR

4) CC

Underlying service
Test Sequence Generation Methods

- A conformance test sequence for a protocol is a sequence of input/output pairs derived from protocol specification.

A transition diagram and table for a machine M
T-Method

A transition diagram and table for a machine M

<table>
<thead>
<tr>
<th>Current state</th>
<th>Input</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>B</td>
<td>S0</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>S0</td>
<td>A</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>B</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>A</td>
<td>S4</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>B</td>
<td>S0</td>
<td>1</td>
</tr>
<tr>
<td>S0</td>
<td>A</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>A</td>
<td>S4</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>A</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>A</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>A</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>A</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>A</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>B</td>
<td>S2</td>
<td>1</td>
</tr>
</tbody>
</table>
# U - Method

<table>
<thead>
<tr>
<th>State</th>
<th>UIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B/λ</td>
</tr>
<tr>
<td>1</td>
<td>A/1 A/1</td>
</tr>
<tr>
<td>2</td>
<td>B/0</td>
</tr>
<tr>
<td>3</td>
<td>B/1 B/1</td>
</tr>
<tr>
<td>4</td>
<td>A/1 A/0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Test subsequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>state S0 to S0</td>
<td>r B B</td>
</tr>
<tr>
<td>state S0 to S3</td>
<td>r A BB</td>
</tr>
<tr>
<td>state S2 to S1</td>
<td>r AAA AA AA AA</td>
</tr>
<tr>
<td>state S2 to S2</td>
<td>r AAA AB B</td>
</tr>
<tr>
<td>state S2 to S1</td>
<td>r AAA A A A A</td>
</tr>
<tr>
<td>state S4 to S3</td>
<td>r AA AB BB</td>
</tr>
<tr>
<td>state S3 to S4</td>
<td>r A A AA</td>
</tr>
<tr>
<td>state S3 to S2</td>
<td>r A A A B</td>
</tr>
<tr>
<td>state S3 to S3</td>
<td>r A B BB</td>
</tr>
<tr>
<td>state S4 to S2</td>
<td>r AA A A A B</td>
</tr>
<tr>
<td>state S3 to S0</td>
<td>r A AB B</td>
</tr>
</tbody>
</table>
U - Method

• Algorithm – Test subsequence generation using U method

• Begin
  – apply reset input r to M so that M is reset to initial state S0;
  – if the transition starts at state si, then, find the shortest path SP(si) from state S0 to state si;
  – apply an input symbol such that M makes the state transition from state si to state sj;
  – apply UIO sequence for state sj.

• End
**D - Method**

<table>
<thead>
<tr>
<th>State</th>
<th>UIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>λλ</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>1λ</td>
</tr>
</tbody>
</table>

**DS sequence for M**

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Test subsequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>state S3 to S3</td>
<td>r A BB</td>
</tr>
<tr>
<td>state S0 to S0</td>
<td>r B BB</td>
</tr>
<tr>
<td>state S1 to S4</td>
<td>r AAAAA A BB</td>
</tr>
<tr>
<td>state S1 to S2</td>
<td>r AAAAA B BB</td>
</tr>
<tr>
<td>state S2 to S1</td>
<td>r AAA A BB</td>
</tr>
<tr>
<td>state S4 to S3</td>
<td>r AA AB BB</td>
</tr>
<tr>
<td>state S3 to S4</td>
<td>r A A BB</td>
</tr>
<tr>
<td>state S3 to S2</td>
<td>r A AA BB</td>
</tr>
<tr>
<td>state S4 to S0</td>
<td>r AA B BB</td>
</tr>
<tr>
<td>state S4 to S2</td>
<td>r AA A A BB</td>
</tr>
<tr>
<td>state S3 to S0</td>
<td>r A AB BB</td>
</tr>
</tbody>
</table>
D - Method

• Algorithm: Test subsequence generation using D-method

• Begin

  – apply reset input r to M so that M is reset to initial state 0;
  – If the transition starts at state si, then, find the shortest path SP(si) from state S0 to state si;
  – apply an input symbol such that M makes a state transition from state si to state sj;
  – apply DS sequence for state sj.

• End
## W - Method

<table>
<thead>
<tr>
<th>State</th>
<th>$M_s(A)$</th>
<th>$M_s(AA)$</th>
<th>$M_s(B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>λ</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Last output symbols on W for M

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Test subsequences with input strings A, AA and B</th>
</tr>
</thead>
<tbody>
<tr>
<td>state S3 to S3</td>
<td>r A A; r A AA; r A B</td>
</tr>
<tr>
<td>state S0 to S0</td>
<td>r B A; r B AA; r B B</td>
</tr>
<tr>
<td>state S1 to S4</td>
<td>r AAAAA A A; r AAAAA A AA; r AAAAA A B</td>
</tr>
<tr>
<td>state S1 to S2</td>
<td>r AAAAA B A; r AAAAA B AA; r AAAAA B B</td>
</tr>
<tr>
<td>state S2 to S1</td>
<td>r AAA A A; r AAA A AA; r AAA A B</td>
</tr>
<tr>
<td>state S4 to S3</td>
<td>r AA AB A; r AA AB AA; r AA AB B</td>
</tr>
<tr>
<td>state S3 to S4</td>
<td>r A A A; r A A AA; r A A B</td>
</tr>
<tr>
<td>state S3 to S2</td>
<td>r A AA A; r A AA AA; r A AA B</td>
</tr>
<tr>
<td>state S4 to S0</td>
<td>r AA B A; r AA B AA; r AA B B</td>
</tr>
<tr>
<td>state S4 to S2</td>
<td>r AA A A; r AA A AA; r AA A B</td>
</tr>
<tr>
<td>state S3 to S0</td>
<td>r A AB A; r A AB AA; r A AB B</td>
</tr>
</tbody>
</table>

Test subsequences for W-method
W - Method

- Algorithm: Test subsequence generation using D-method
- Begin
  - apply reset input r to M so that M is reset to initial state S0;
  - If the transition starts at state si, then, find the shortest path SP(si) from state S0 to state si;
  - apply an input symbol (z) such that M makes the state transition from state si to state sj;
  - apply W sequence for state sj with understanding that,
    - \( S@W = S(@_1; ...; @_k) \)
    - \( = \{S@_1; S@_2; ...; S@_k\} \)
  - Where S is concatenation of the inputs used in first three steps of algorithm, i.e.,
  - \( S = r@SP(si)@z. \)
- End
Distributed architecture by local methods

General distributed test architecture
FSM representation of IUT with n ports

- A multiport FSM with n ports (np-FSM) - 6-tuple \((S_t, \ell, T, O, S_0)\) where \(n > 1\)
- \(S_t\) is finite set of states or labels,
- is a \(n\)-tuple \(\ell_k = (L_{i1}; L_{i2}; \ldots L_{in})\), where \(L_{ik}\) is set of inputs for port \(k\)
- is a \(n\)-tuple \(O_k = (Lo1; Lo2; \ldots Lo_n)\), where \(O_k\) is set of outputs for port \(k\)
- \(T\) is a transition function over subset \(D\)
- \(O\) is an output function, that maps \(D\) to \(O \cup \{\}\\)
- \(S_0\) is the initial state belongs to \(S_t\)
Functioning of 3p-FSM

1. \( t_1 = 1 = < a; c > \)

2. \( t_2 = 2 = < d > \)

3. \( t_3 = 3 = < d > \)

- Execution of transitions are as follows:
  - \( t_1: A \text{ to } B = 1/< a; c >, \)
  - \( t_2: B \text{ to } C = 2/< d >, \)
  - \( t_3: C \text{ to } A = 3/< d >, \)
  - \( t_4: A \text{ to } A = 3/< c >, \)
  - \( t_5: A \text{ to } B = 2/< a; d >, \)
  - \( t_6: B \text{ to } B = 1/< d >, \)
  - \( t_7: B \text{ to } C = 3/< a; c >, \)
  - \( t_8: C \text{ to } C = 2/< a >, \)
  - \( t_9: C \text{ to } A = 1/< a >. \)
Functioning of 3p-FSM

The initial state is A

Li₁ = {1}, Lo₁ = {a,b},
Li₂ = {2}, Lo₂ = {c},
Li₃ = {3}, Lo₃ = {d},
Synchronizable Test Sequence

- **Synchronization problem:**
  - Considering two consecutive transitions $t_1$ and $t_2$ of a given np-FSM ($n \geq 2$), one of the testers is said to face a synchronization problem if the tester did not take part in the first transition and if the second transition requires that it sends a message to the machine $I$.

- **Interaction Points**
  - Interaction points (IPs) of a given transition $[p_i; PO]$. Let $p_i = \{1; 2; n\}$, and $PO$ is a subset of $\{1; 2; \ldots n\}$ ($n$ is number of ports). $[p_i; PO]$ is said to be an interaction point of a given transition $t$ if $t$ receives an input from the port $p_i$, and sends output(s) on port $PO$ (if $PO$=null, $t$ does not send any output).
Synchronizable Test Sequences

- Given an ordered pair of transitions $t_1$ and $t_2$ of machine $I$, let $[p_i, P_0]$ and $[p'_i, P_0]$ be their IPs, respectively, $t_1$ and $t_2$ are said to be synchronizable if $p_i = p'_i$ or $p'_i = P_0$.  

$$\begin{align*}
\text{Input Sequence:} & \quad \text{Resulting Global Sequence:} \\
& \quad \text{Port 1: } 0, 1, 1, 0 \\
& \quad \text{Port 2: } 0, 1, a, b, 1, c, 1, b
\end{align*}$$

**PIs:** $<1, \{1,2\}>, <2, \{2\}>, <2, \{1\}>, <1, \{1\}>$

2p-FSM and its synchronizable transition tour:

- Tester 1 – Port 1
- Tester 2 – port 2
- The initial state is A
Conformance Testing with TTCN

- TTCN is a part of Conformance Testing Methodology and Framework (CTMF) for the specification and description of abstract test suites for conformance testing of communication protocols.

<table>
<thead>
<tr>
<th>No</th>
<th>Label</th>
<th>Behaviour Description</th>
<th>Constraints Ref</th>
<th>Verdict</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(PS_Init:= ResetParcelService( ))</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[PS_Init]</td>
<td>(P)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[NOT PS_Init]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Detailed Comments:

B) TTCN/mp

$Begin_TestStep
$TestStepId Preamble
$TestStepRef Example_ATS/
$Objective /* To bring the SUT into the initial state */
$DefaultsRef $LabelId
$BehaviourDescription $Line [1] [PS_Init]
$BehaviourLine $Line [2] [PS_Init:= ResetParcelService()]
$Cref
$VerdictId (P)
$End_BehaviourLine
$BehaviourLine $Line [3] [NOT PS_Init]
$Cref
$VerdictId I
$End_BehaviourLine
$End_BehaviourDescription
$End_TestStep

TTCN forms: TTCN/gr (graphical) and its corresponding TTCN/mp (machine processable form) code below the table for test case dynamic behavior.
Conformance Testing with TTCN

TTCN Parts:

- **Overview Part:**
  - identifies the protocol test suite and the test method used.
  - defines the structure of the whole test suite.

- **Declaration Part:**
  - declares all items used in the test suite.

- **Constraints part:**
  - assigns values for parameters of items declared above.

- **Dynamic part:**
  - contains the test trees that should be executed to check a protocol implementation against the standard.
TTCN Example

<table>
<thead>
<tr>
<th>Test Case Dynamic Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case Name:</td>
</tr>
<tr>
<td>Group:</td>
</tr>
<tr>
<td>Purpose:</td>
</tr>
</tbody>
</table>

**behaviour**

<table>
<thead>
<tr>
<th></th>
<th>constraints</th>
<th>verdict</th>
</tr>
</thead>
</table>
|+preamble
| LT ! T-PDU-connect-request |
| UT ? T-SP-connect-indication |
| UT ! T-SP-connect-response |
| LT ? T-PDU-connect-confirm |
| OTHERWISE |
| LT ? T-PDU-disconnect-request |
| OTHERWISE |

TTCN example of behavior description
Conformance Testing in Systems with Semicontrollable Interfaces.

Testing (N)-layer IUT with an (N+1)-layer semicontrollable interface
Each input can be one of three different types:

- directly controllable: a tester can directly apply the input to the IUT through the PCO;

- semicontrollable: a tester cannot directly apply the input to the IUT through the PCO (or IAP), and

- uncontrollable: the input may be supplied through a PCO (or an IAP) without any explicit action of the tester.
System model building taking into account controllability problem

Test Component: IUT
Test Context:
    .... FSM_1 ... FSM_F
    .... l_1 ... l_F
System model building taking into account controllability problem

- derive a set of tests exercising each transition in an IUT's FSM at least once.
- given a graph representing an IUT's FSM, to find a minimum cost tour of G such that each transition is covered at least once.
- Given a graph $G(V < E)$ representing an FSM model of an IUT with multiple semicontrollable interfaces.

Parameters:
- $|V|$-number of nodes in G;
- $F$-number of semicontrollable interfaces interacting with the IUT;
- $T_i \subseteq E$—subset of edges in G triggered by the inputs from the $i^{th}$ semicontrollable interface;
- $b_i$-buffer size at the $i$-th semicontrollable interface $I_i$;
- $A_i$-set of inputs triggering transitions in $T_i$;
System model building taking into account controllability problem

- Parameters (contd.)
  - $O_i$ - set of outputs of the IUT that are consumed by the semicontrollable FSM$_i$;
  - $c_i$ - number of different transition classes in the IUT triggered by inputs at $I_i$.
  - $U_{i,j}$ (E - set of transitions in the IUT with output $o_{i,j}$, an input $a_{i,j}$, $A_i$ is buffered at $I_i$);
  - $W_{i,j}$ (E - set of transitions in the IUT with output $o_{i,j}$, such that, in response to $o_{i,j}$, no output is generated by FSM$_i$).

- Let $A_i = \{a_{i,1}, \ldots, a_{i,c_i}\}$ and $O_i = \{o_{i,1}, \ldots, o_{i,m}\}$. Let the sets of $T_i$ and $U_i$ be defined as follows

  there may be several outputs in set $O_i$ that force input $a_{i,j}$ to be buffered at $I_i$.

$$T_i \overset{def}{=} \bigcup_{j=1}^{c_i} T_{i,j}, \text{ and } U_i \overset{def}{=} \bigcup_{j=1}^{c_i} U_{i,j}.$$
IUT interacting with two semicontrollable interfaces

<table>
<thead>
<tr>
<th>Edge Name</th>
<th>Input from</th>
<th>Output to</th>
<th>Edge Name</th>
<th>Input from</th>
<th>Output to</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>LT?x₁</td>
<td>FSM₁!o₁₁</td>
<td>e6</td>
<td>LT?x₆</td>
<td>LT!y₆</td>
</tr>
<tr>
<td>e2</td>
<td>LT?x₂</td>
<td>FSM₂!o₂₁</td>
<td>e7</td>
<td>LT?x₇</td>
<td>LT!y₇</td>
</tr>
<tr>
<td>e3</td>
<td>FSM₁?a₁₁</td>
<td>LT!y₃</td>
<td>e8</td>
<td>FSM₁?a₁₂</td>
<td>LT!y₈</td>
</tr>
<tr>
<td>e4</td>
<td>FSM₂?a₂₁</td>
<td>FSM₁!o₁₂</td>
<td>e9</td>
<td>LT?x₉</td>
<td>LT!y₉</td>
</tr>
<tr>
<td>e5</td>
<td>LT?x₅</td>
<td>FSM₂!o₂₂</td>
<td>e10</td>
<td>LT?x₁₀</td>
<td>LT!y₁₀</td>
</tr>
</tbody>
</table>

IUT interacting with two semicontrollable interfaces
Conformance Testing for RIP

A, B, C - Gateways
a, b, c, ai, bi, ci - Networks

An Illustration of a test application for RIP based routing
Multimedia applications testing

- Telcom TSL - A framework for QoS testing
A testing architecture for synch of audio and video

A testing architecture for synch of audio and video
A testing architecture for synch of audio and video revisited
QoS testing with TTCN and MSC

- definition of the functional requirements.
- time constraints imposed by non-functional requirements.
- language used for functional QoS test specification facilitate expressing non-functional requirements.

- language should:
  - allow the description of static test information,
  - allows the definition of dynamic test information,
  - support the treatment of test specific information,
  - include facilities to express non-functional requirements
QoS testing with TTCN and MSC
SDL based tools for conformance testing

- **TESDL**
  - prototype tool for the automatic generation of test cases from SDL specifications in the context of the OSI CTMF.

- **TTCN Link**
  - environment for efficient development of TTCN test suites based on SDL specifications in SDT3.0 (SDL Description Tool)

- **SAMSTAG**
  - formalizes test purposes and defines the relation between test purposes, protocol specifications and test cases.

- **TOPIC V2**
  - works by cosimulating the SDL specification and an observer representing the test purpose.

- **Tveda V3**
  - tool for automatic test case generation

- **TAG Tool**
  - generates test cases from SDL systems.
SDL based conformance testing of MPLS

- Test sequence generation approach

- Alg. 1: Find a path that covers a maximum number of transitions:
  - Consider a initial state which makes transition to n states.
  - These n states may in turn make transitions to other states and so on.
  - Find out the maximum number of outgoing transitions from all states and retain those transitions that are of maximum length from each state discarding other transitions.
  - Finally initial state will choose the state path having maximum number of transitions.

- Use the generated test sequence and apply Alg. 2:
  - Give this test sequence to MPLS Simulation, see the behavior of the simulation.
  - Find again a path that covers a second maximum number of transitions by the method given above.
  - Repeat the same process until all the transitions have been covered.
SDL based MPLS testing

- In the initial phase MPLS Extended Communication Finite State Machine (ECFSM) was transformed into an equivalent Formal SDL model.

- To view the typical messages exchanged between co-MPLS peers, the SDL model was simulated to view the TRANSPORT and MPLS messages exchanged between the two LDPs.

- During this exchange of messages that is witnessed in the Message Sequence Chart each of these MPLS nodes traverses through all States in the FSM thereby making associated transitions.

- Simulate a 3-node topology (ingress router, LSR1, egress router) and look into the stability issue of MPLS.

- After Initializing the system, each MPLS router gets to know about the current topology from their neighbors. Now that the topology is stabilized.

- Apply the message to ingress node, you will get the same message at egress node.

- Now MPLS 3-node topology is tested for its functionality.

- Find test sequences
SDL based MPLS testing

• Results

  - protocol satisfies all its properties, implies the Formal Model of MPLS is verified.

Test Sequence 1:
session tcp connection establishment
message arrival (after that 10 secs delay)
message
junk
ldp withdraw

Test Sequence 2:
session tcp connection establishment
message arrival (after that 10 seconds delay)
ldp withdraw
upstream lost

Test Sequence 3:
session tcp connection establishment
message arrival (after that 10 seconds delay)
ldp withdraw
ldp upstream abort

Test Sequence 4:
session tcp connection establishment
message arrival (after that 10 seconds delay)
ldp withdraw
shut down msg
A snapshot of SDL simulation

| Success               | To lower_testing_entity:9 From upper_test| test_performed(1) To upper_testing_entity:2 From ldp_release1 To ler11:5 From isr22:6 ldp_release To isr22:6 From ler33:7 ldp_withdraw2 To ler33:7 From isr22:6 message To Env From ler33:7 ldp_withdraw1 To isr22:6 From ler11:5 message2 To ler33:7 From isr22:6 message1 To isr22:6 From ler11:5 ldp_mapping1 To ler11:5 From isr22:6 ldp_mapping To isr22:6 From ler33:7 ldp_request1 To ler33:7 From isr22:6 ldp_request To isr22:6 From ler11:5 start_operation To ler11:5 From ler1:8 keepalive_msg3 To ler1:8 From ler3:4 keepalive_msg2 To ler3:4 From isr2:3 keepalive_msg1 To ler2:3 From ler1:8 init_msg3 To ler1:8 From ler3:4 init_msg2 To ler3:4 From isr2:3 init_msg1 To ler2:3 From ler1:8 session_tcp_connection_est2 To ler3:4 From isr2 ldp_withdraw To ler11:5 From lower_testing_entity1 session_tcp_connection_est1 To ler2:3 From ler1 message To ler11:5 From lower_testing_entity9 message_arrival To ler1:8 From lower_testing_entity message give_seq1 To lower_testing_entity9 From Env |
A snap shot of SDL simulation

Output of sequence no. 2