SDL:
A Protocol Specification Language

Mrs. Anandi Giridharan
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore – 560012, India
Specification Languages

- Specifies Communication Protocol either by using Formal or graphical notation.
- Follows syntax and semantics
- LOTOS (Language of Temporal Ordering Specification), ESTELLE (Extended StateTransition Language), SDL (Specification and Description Language) and CPN (Coloured Petri-Nets) etc are some of the specification languages.

Why all these languages??

- To create certain standard and make life simpler
- To increase the reliability and efficiency of goods and services.
- Possible to analyze and simulate alternate system solution.
SDL (Specification and Description Language)

- SDL formal language defined by ITU-T standardization.
- Used worldwide for all kinds of complex, communication systems.
- In telecommunication: used for development of Software and hardware.
- e.g.: 3G, Cellular Phones, switches, WLANs, Bluetooth devices, etc.
Salient Features of SDL

- well defined set of concepts.
- unambiguous, clear, precise, and concise specifications
- (SDL describes the structure, behavior and data of real time and distributed systems with mathematical rigor that eliminates ambiguities and guarantees system integrity).
- thorough basis for analyzing specifications and conformance testing.
- basis for determining the consistency of specifications.
- good computer support interface for generating applications without the need for the traditional coding phase.
- Use computer based tools to create, maintain and analyze and simulate.
- high degree of testability as a result of its formalism for parallelism, interfaces, communication and time.
- portability, scalability and open specification: Abstract mechanism for protabilty between cross compliers and OS, process etc and can use different kernels ranging form small system to multiprocess high end system.
- high degree of reuse because of visual clarity, object oriented concepts, clear interfaces, and abstraction mechanisms.
- facility for applying optimization techniques improving protocol efficiency.
Description of a communication system by SDL

- structure: a system, block, process, and procedure hierarchy
- communication: signals with optional signal parameters and channels (or signal routes)
- behavior: processes or entities
- data: abstract data types (ADTs) or predefined data types
- inheritance: describing relations and specialization
Components of a communication system structure in SDL

System

Block 1
- Process A
  - Procedure
- Process B
  - Procedure

Block 2
- Process C
  - Procedure
- Process D
  - Procedure

ENVIRONMENT

Components of a communication system structure in SDL
**SDL Structure**

- **System:**
  - set of instances of different kinds such as block, process and service instances
  - blocks are connected with each other and the environment using channels.
  - System interacts with its environment through signals sent on channels.
A system diagram usually contains the following elements:

- **system name**
- **signal descriptions**
- **channel descriptions**
- **data type descriptions**
- **block descriptions**

SDL/GR the system description is called system diagram
Block:
- Usually contains the following: block name, signal descriptions, channel to route connection and process descriptions.
System components & Block components

a) System components

b) Block components

a) System components, b) Block components
### SDL Structure

- **Process:**
  A process diagram usually contains the following elements:
  * Process name,
  * formal parameters,
  * variables descriptions,
  * timer descriptions,
  * procedure description, and
  * process graph (for the description of the finite state machine of the process).

- **Procedure:**
  - are part of processes and they define the patterns of behavior of a protocol.
  - The behavior of a procedure is defined by means of states and transitions.
  - procedure may have local variables.
SDL Structure

a) example using inputs/outputs

S0

in1

out1:=in1+1;

out1

S1

b) example using decision making

S0

count:=0

count:=count+1;

count <100

S1

Process diagram in SDL
Data Types

- Predefined Datatypes in SDL
  - Integer
  - Real
  - Natural
  - Boolean
  - Integer
  - Real
  - Natural
  - Boolean
Abstract Data Types

- A data type with no specified data structure.
- It specifies a set of values, a set of operations allowed, and a set of equations that the operations must fulfill.
- T'simple to map an SDL data type to data types used in other high-level languages.
- A new data type is often introduced with new operators on existing datatypes.
- Each data type is associated with a set of literals and operators, for example, boolean literals (true or false), and its operators are and, not, or, and xor.
Communication paths

a) Channels between the blocks

Block 1

x1, x2, x3

buffer

C12

Block 2

C21

k1, k2

buffer

b) Signal routes between the Processes

Process 1

z1, z2

buffer

R12

Process 2

y1

R21

buffer

Communication paths: a) channels, b) signal routes
Example – Communication Path

System

Block B1

Block B2

Process pr1

Process pr2

Procedure pr1

C - Channel

R - Router

Structural view of a communication system by SDL
Example – Communication Path

- It consists of a system, two blocks B1 and B2, two processes in block B1 and B2 (not shown in the figure), each process pr1 and pr2 consists of several states, transitions and procedures, and each procedure with several states and transitions.

- The channels C1 and C3 are used to communicate with the environment by the blocks 1 and 2, respectively, and the channel C2 is used to communicate between the blocks 1 and 2.

- The signal routes R1 to R4 are used to communicate among the processes of a block and the environment. The signal routes R2 and R3 are used for communication among the processes pr1 and pr2.

- The signals 'a' and 'b' are used to communicate with the environment using the channels C1 and C3, respectively. Similarly, signals X and Y are used in the block B1 which communicate with other blocks via signal routes R1 and R4, respectively.

- A set of signals [W,Z] and [J,K] are used to communicate among the processes in the block B1 via signal routes R2 and R3, respectively.
THANK YOU............
Examples of SDL based protocol specifications

- QA protocol

Service specification of Question Answer protocol
**QA protocol**

- A QA service provider provides the service in the form of answers from the user A to the questions framed by the user Q.

- The sequence of operations of the protocol is as follows:
  - Signal 'Q-req' is sent to the protocol entity of user Q from the user Q.
  - Signal 'Q-req' is forwarded as signal 'D-req' to service provider protocol entity.
  - QA service provider (provides connectionless service) forwards the 'D-req' signal as 'D-ind' signal to the user A's protocol entity.
  - Signal 'D-ind' is sent as signal 'Q-ind' to the protocol entity of user A.
  - User A generates the signal 'A-req' as an answer to 'Q-req' and sends it to the service provider.
  - Signal 'A-req' is sent as signal 'D-req' to the service provider protocol entity.
  - QA service provider sends signal 'D-ind' to protocol entity of user Q.
  - Protocol entity at user Q forwards this as an 'A-ind' to the user Q.
Protocols Spec in SDL - QA protocol

a) QA Protocol Block Diagram

b) Question Entity

Process Qentity

- WaitQ
- Qreq
- Dreq
- WaitA

Process Aentity

- WaitQ
- Dind
- Aind
- WaitA

c) Answer Entity

- WaitQ
- Dind
- Areq
- WaitA

SDL specification for QA: a) block diagram, b) Question entity, c) Answer entity
The components of the QA protocol are as follows:

- One block called QA protocol.
- Two processes called 'Qentity' and 'Aentity' representing the user Q and user A, respectively.
- Three bidirectional signal routes represented as 'Qi', 'Ai', and 'CL' are used in the protocol model.
- Six signals are declared using the keyword 'SIGNAL': Dreq, Dind, Qreq, Qind, Aind, and Areq. Dreq (from process 'Qentity' to 'Aentity' and vice versa) and Dind (from process 'Qentity' to 'Aentity' and vice versa) are the signals routed on route 'CL'. 'Aind' (from process Qentity to user Q) and 'Qreq' (from user Q to process Qentity) are the signals routed on routes 'Qi'.

Xon-Xoff protocol

- primitive flow control protocol
- a receiver controls the data transfer by signaling the sender
- This protocol uses three types of PDUs:
  - Data PDU: this is sent to the receiver PE (Protocol Entity) by the sender PE, it contains the user data;
  - Suspend PDU: it is sent to the sender PE by the receiver PE to signal the sender to stop sending Data PDUs;
  - Resume PDU: it is sent to the sender PE by the receiver PE to signal the sender to resume sending Data-PDUs.
Xon-Xoff protocol

- The sequence of operations in this protocol are as follows
  - The sender sends the Data PDU along with the signal 'LDreq(pdu)' to the receiver
  - The receiver sends either Resume PDU or Suspend PDU to the sender along with the signal 'LDreq(pdu)' (sends the resume signal if it has received minimum Data PDUs or sends the suspend signal if it has received maximum Data PDUs set for the sender).
  - Signal 'LDreq(pdu)' from the receiver is received at the sender.
Block Xon-Xoff

SIGNAL Dind (Octet_string), Dreq(Octet_string), LDreq(pdu);
newtype pdu type
    literals Data, Suspend, Resume;
end newtype pdu type;
newtype pdu struct
    Ptype pdu type;
data Octet_string;
end newtype pdu;

a) Xon-Xoff Protocol Block Diagram
Xon-Xoff Protocol

b) Sender Entity

dcl d Octet_string;
dcl p pdu;

Process SenderPE

Go

True

Dreq(d)

ptype := Data;
data := d;

LDreq(p)

Go

Go Waiting

LDreq(p1)

resume

p1!ptype

suspend

Go

Waiting
variable declarations: \( d \) is the data to be transmitted, which is declared as octet-string (string of length 8 bits), \( p \) is declared as a pdu type (contains type of PDU and the data). These declarations are given in the package shown by dotted lines as the new data types.

states of the sender: Go, Waiting and a combination of Go and waiting

Signals with parameters: output signal LDreq\((p)\), an input signal LDreq\((p1)\), and Dreq\((d)\) from the sender.

PDU type (ptype): Data, resume and suspend \( (p!ptype := \) Data, indicates that the Data is assigned to member ptype of pdu; the operator '!' is used to access the member of the structure).
The receiver protocol entity

- variable declarations: d as octet string; p and p1 as pdu data type; n, max and min as integers (n=counter, min=minimum number of PDUs, max=maximum number of PDUs); buf is a FIFO buffer data type.
- states of the receiver: Go and Stopped.
- signals with parameters: output signals LDreq(p1) and Dind(d); high priority input signal LDreq(p).
- FIFO buffer: the buffer, buf, will be appended with the received octet string from the sender (the operator '//' indicates appending the buffer, the operator '/=' indicates not equal, the function Substring finds a string from buf).
- PDU type (ptype): resume and suspend PDUs are generated by the protocol entity.
Alternating Bit Protocol

System  Altbitprotocol

[ Lose_Ack ]  ABP_Block  [ Lose_Data ]

ack_loss_signal  data_loss_signal

use signals;
Alternating Bit Protocol

- One block which has sender and receiver entities along with the channel process (ack and data medium).
- A package defining all the signals used in the protocol model: DataS_0 and DataS_1 are the data sent from the sender to data medium process; DataR_0 and DataR_1 are the data sent from the data medium to receiver process; AckS_0 and AckS_1 are the acknowledgments from ack medium to sender; AckR_0 and AckR_1 are the acknowledgments from receiver to ack medium; Receiver Ready, Receive, and Deliver are signals exchanged between receiver and upper layer processes.
- Two communication channels: ack loss signal and data loss signal connects the system to environment.
- Input signals from the environment: Lose Data and Lose Ack.
signal DataS_0, DataS_1, DataR_0, DataR_1;
signal AckS_0, AckS_1, AckR_0, AckR_1;
signal Sender_Ready, Send;
signal Receiver_Ready, Receive, Deliver; signal Lose_Ack, Lose_Data;
Alternating Bit Protocol
Alternating Bit Protocol

- The block consists of the following processes:
  - ULsenderP: upper layer sender process
  - ULReceiver: upper layer receiver process
  - DataMedium: provides service to transmit data from sender to receiver
  - SenderP: sender process to transmit the frames
  - ReceiverP: receiver process to receive the data and deliver to upper layer process as well as acknowledge the received data.
  - AckMedium: provides service to transmit the acknowledgments received from the receiver to sender.
Sender Process of ABP
Receiver Process of ABP

Signalset
DataR_0, DataR_1, Receive;

Process ReceiverP

Idle_0

Wait_0

DataR_1

Receive

DataR_0

Deliver

AckR_0

Idle_1

DataR_0

AckR_0

DataR_1

AckR_1

Receiver_Ready

Wait_1

Wait_1

DataR_0

Receive

DataR_1

Deliver

AckR_1

Idle_0

DataR_1

AckR_1

DataR_0

AckR_0

Receiver_Ready

Wait_0
Sender & Receiver Process of ABP

- **Sender Process:**
  - data declarations: Timer1
  - states: Wait_0, Wait_1, Idle_0, Idle_1
  - inputs: AckS_0, AckS_1, Send
  - outputs: DataS_0, DataS_1, Sender_Ready

- **Receiver Process:**
  - states: Wait_0, Wait_1, Idle_0, Idle_1
  - inputs: DataR_1, DataR_0, Receive
  - outputs: AckR_0, AckR_1, Deliver, Receiver Ready
Upper Layer of sender of ABP

Process ULSenderP

Signalset Sender_Ready;

Wait

Sender_Ready

Send
Upper Layer of reciever of ABP

Process

ULReceivererP

Signalset
Sender_Ready;

Wait

Receiver_Ready

Receive

Wait_1

Deliver
Ack Medium Process

Process AckMedium

signalset
AckR_0, AckR_1, Lose_Ack;

Empty

AckR_0                Lose_Ack                AckR_1

AckS_0                Lose                AckS_1

AckR_0                AckR_1

Empty
Specifications of a bridge connecting CSMA/CD and CSMA/CA protocols
Specifications of a bridge connecting CSMA/CD and CSMA/CA protocols

- This scenario is modeled as a system comprising of three blocks, variable declarations and signal routes.
- The components defined in the system are as follows:
  - It has three blocks: bridge, csma/cd and csma/ca.
  - A package defining all the signals used in the block and process specifications is given in the next slide.
  - It has four channels connecting the blocks: c1, c2, c3 and c4 where each carries different signals which are defined in the package.
  - Input signals from the environment: user 1 and user 2 are the users of csma/cd block; and, user3 and user4 are users of csma/ca.
Specifications of a bridge connecting CSMA/CD and CSMA/CA protocols

```java
Package pac

signal packet(Integer, Integer, Integer), fpacket(Integer, Integer, Integer);
signal res(Integer), req_chan_sta(Integer), req_chan_state(Integer);
signal packet12(Integer, Integer, Integer), packet21(Integer, Integer, Integer), outpac(Integer, Integer, Integer);
signal fpacket1(Integer, Integer, Integer), packet_bridge(Integer, Integer, Integer);
signal packet1_bridge(Integer, Integer, Integer), end_packet, end_pac;
signal user3(Integer, Integer, Integer), user4(Integer, Integer, Integer);
signal user1(Integer, Integer, Integer), user2(Integer, Integer, Integer);
```
Specifications of a bridge connecting CSMA/CD and CSMA/CA protocols

- The bridge block consists of the following:
  - Process: bridge p
  - Signal routes: p1 and p2 which are connected to channels c1 and c2 respectively
  - Input signals carried by p1 and p2 are: fpacket1, res, and end pac
  - Output signals carried by p1 are: packet1 bridge, req chan state and end packet
  - Output signals carried by p2 are: packet bridge, req chan state and end packet.
The block CSMA/CA

- Processes:
  - **channel csma_ca**: maintains the channel status and forwards the data to bridge and other node(s)
  - **csma_ca_3**: implements the csma/ca operations for user3
  - **csma_ca_4**: implements the csma/ca operations for user4
  - **pac_gen**: generates packets according to input from environment
  - **signal routes**: p1, p2, p3, p4 and p5 each carrying the signals as indicated
  - Signals from the block are routed to other blocks by using channels c1 and c4
The block CSMA/CA
The block CSMA/CD

Processes:

- channel csma_cd: maintains the channel status and forwards the data to bridge and other node(s)
- csma_cd_1: implements the csma/cd operations for user1
- csma_cd_2: implements the csma/ca operations for user2
- pac_gen: generates packets according to input from environment

- signal routes: p1, p2, p3, p4 and p5 each carrying the signals
- signals from the block are routed to other blocks by using channels c2 and c3
The block CSMA/CD
Process diagram of Bridge_p

Process bridge_p

req1

req_chan_state(j) via p2

wait1

res(i)

fpacket1

end_pac

wait_for_timer

set(now+1,t)

c:=c-1

back

i=0

t

send_bridge

req

req_chan_state(j) via p1

wait

end_pac

c:=5

true

c=0

false

fpacket

req

true

false

i=0

send_bridge

back1

set(now+1,t)

wait_for_timer1

c:=c-1

t

fpacket1

end_pac

c:=5

true

false

req1
Process diagram of csma_ca channel
Process diagram of csma_ca channel
Process diagram of csma_ca_3 process

Process csma_ca_3:

```plaintext

dcl src, dest, n  integer;
dcl node, at, d, c integer := 3;
dcl timer t;
```

- **set(now+1, t)**
- **wait**
- **outpac(src, dest, n)**
  - **req_chan_stata(node)**
    - **wait_for_res1**
      - **res(st)**
        - **true**
          - **st=0**
            - **false**
              - **b2**
        - **false**
          - **back**
          - **reset(t)**
          - **req_chan_stata(node)**
          - **wait_for_res**
            - **res(st)**
              - **true**
                - **st=0**
                  - **false**
                    - **b1**
            - **false**
              - **c:=c-1**
                - **true**
                  - **c=0**
                    - **false**
                      - **b1**
                - **false**
                  - **c:=10**
                    - **true**
                      - **b1**
                    - **false**
                      - **b2**
          - **back**
          - **reset(t)**
          - **req_chan_stata(node)**
          - **wait_for_res**
            - **res(st)**
              - **true**
                - **st=0**
                  - **false**
                    - **b2**
              - **false**
                - **back**
```

- **b2**
- **false**
  - **n=d**
    - **true**
      - **b2**
    - **false**
      - **packet12(src, dest, d)**
        - **d:=d+1**
          - **b2**
          - **true**
            - **end_packet**
              - **d:=0**
                - **true**
                  - **c:=0**
                    - **false**
                      - **b1**
                - **false**
                  - **b2**
          - **false**
            - **back**
```

- **t**
- **back**
- **reset(t)**
- **req_chan_stata(node)**
- **wait_for_res**
- **res(st)**
Process diagram of pac_gen process

Process pac_gen

dcl src, dest, n Integer;

wait

user3(src, dest, n)

outpac(src, dest, n) via p4

wait

user4(src, dest, n)

outpac(src, dest, n) via p5

wait
Process diagram of channel csma_cd

Process channel_csma_cd

wait

reg_chan_state(i)

i=0

req

true

j=1

false

res(i) via p1

true

j=2

false

res(i) via p2

j=3

wait_for_pack

reg_chan_state

packet_bridge(src,dest,d)

packet12(src,dest,d)

packet21(src,dest,d)

end_packet

packet(src,dest,d) via p2

fpacket(src,dest,d)

fpacket1(src,dest,d)

end_packet

packet(src,dest,d) via p1

end_packet

i:=0

wait

i:=0

wait
Process diagram of channel csma_cd

Process channel_csma_cd

req

true

j=1

false

res(i) via p1

true

j=2

false

res(i) via p2

res(i) via p3

wait
Process diagram of channel csma_cd_2 process

Process csma_cd_2

wait

packet(src,dest,n)

wait

outpac(src,dest,n)

req_chan_sta(node)

wait_for_res

res(st)

st=0

false

true

n=d

end_packet

d:=0

false

packet21(src,dest)

set(now+1,t)

time_sta

reset(t)

c:=c-1

c:=5

c:=0

true

false

back

back

dcl src,dest,n,c Integer;
dcl node Integer:=2;
dcl st Integer;
dcl d Integer:=0;
timer t;
Sliding Window Protocol

System GBN_ARQ

Transmitter

Receiver

SIGNAL pdu(Int4), ack(Int4);
SYNONYM N Integer = 4;
SYNONYM W Integer = 2;
SYNTYPE Int4 = Integer
Constants 0:3
ENDSYNTYPE Int4;

3 blocks in system: Transmitter, Receiver, Channel
Signals: pdu and ack, Channels: tx_ch and rx_ch

a) GBN_ARQ System
sliding window protocol

Process Transmitter

- transfer

ack (nr)

win := 0;
ns := nr;

reset (T)

pdu(ns)

win := win + 1;
ns := (ns + 1) mod N;

win = W

true

- 

false

set (T)

- 

win := 1;
ns := nr;

pdu(ns)

T

b) Transmitter Entity
Sliding Window Protocol

c) Channel Entity

```
<table>
<thead>
<tr>
<th>Process Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcl n Int4:=0</td>
</tr>
<tr>
<td>idle</td>
</tr>
<tr>
<td>pdu (n)</td>
</tr>
<tr>
<td>any</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>pdu(n) via rx_rt</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

-```

d) Receiver Entity

```
<table>
<thead>
<tr>
<th>Process Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>reception</td>
</tr>
<tr>
<td>pdu (ns)</td>
</tr>
<tr>
<td>ns = nr</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>nr:=(nr+1) mod N;</td>
</tr>
<tr>
<td>reception</td>
</tr>
<tr>
<td>pdu(n) via rx_rt</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

SDL Specifications of TCP

- Model consists of three processes
  - variable declarations
  - signal package
  - two signal routes
  - signals: Startsim, ACK1, Ack, packet, thr, packet1 and packet junk
  - A package p in which all the signals used in the block are defined
SDL Specifications of TCP

```
Package p

signal packet(Integer), ACK(Integer);
signal Startsim;
signal packet_junk, ACK_junk;
signal packet1(Integer), ACK1(Integer);
signal thr(Integer, Integer);
```
Transmitter Process of TCP

dcl base Integer:=1;
dcl next_seq_no Integer:=1;
dcl N Integer:=1;
dcl counter Integer:=100;
dcl i Integer;
dcl w Integer:=8;
dcl x,y Integer:=0;
timer1,tt1;

wait_for_start_sim
Startsim

wait_ack

ACK1()

false

false

next_seq_n...
N:=1

true

N:=w,
base:=i+1,
x:=x+1

false

true

N:=N+1,
base:=i+1,
x:=x+1

h

h

h

thi(x,y)
thi(x,y)

false

next_seq_no=base+N

true

next_seq_no<eq...

false

true

next_seq_no:=next_seq_no+1,
y:=y+1

packet(next_seq_no);

stop_sim

stop_sim

set(now+600,tt1)

set(now+6,t)
Receiver Process of TCP

```plaintext
dcl seq_no Integer := 1;
dcl i Integer;
```

1. `wait_for_pack`
2. `packet1(i)`
3. `packet_junk`
4. `false`
   -
5. `true`
   - `i = seq_no`
   - `seq_no := seq_no + 1`
   - `Ack(seq_no)`
   - `wait_for_pack`
Simulation model representing the simulated network in one SDL system
Link modeling
The 3 topologies
Components of an OSPF router model which were expressed in SDL.
3-router mesh topology in SDL with Point-to-Point links.
A router will select the "best BGP" route when there are multiple BGP route possibilities of the same specificity.

- It goes (basically)
- Route specificity and reachability and reachability
- BGP weight metric MED (Multi exit discriminator)
- BGP local_pref metric
- Internally originated vs. Externally originated
- AS-PATH (Autonomous System PATH) length
Following are different states of BGP FSM or BGP process for its peers:

- **IDLE**: State when BGP peer refuses any incoming connections
- **CONNECT**: State in which BGP peer is waiting for its TCP connection to be completed.
- **ACTIVE**: State in which BGP peer is trying to acquire a peer by listening and accepting TCP connection.
- **OPENSENT**: BGP peer is waiting for OPEN message from its peer
- **OPENCONFIRM**: BGP peer is waiting for KEEPALIVE or NOTIFICATION message from its peer.
- **ESTABLISHED**: BGP peer connection is established and exchanges UPDATE, NOTIFICATION, and KEEPALIVE messages with its peer.
MPLS

SDL based representation of a 4-node Topology.
SDL based representation of BGP Speaker using Block type
simulation manager process
MPLS operation

- **Label Creation and Distribution**
  - Before any traffic begins, the routers make the decision to bind a label to a specific FEC and build their tables.
  - In LDP, downstream routers initiate the distribution of labels and the label/FEC binding.
  - In addition, traffic-related characteristics and MPLS capabilities are negotiated using LDP.
  - A reliable and ordered transport protocol should be used for the signaling protocol. LDP uses TCP.

- **Table Creation**
  - On receipt of label bindings, each LSR creates entries in the label information base (LIB).
  - The contents of the table will specify the mapping between a label and an FEC.
MPLS operation

- **Label-Switched Path Creation**
  - LSPs are created in the reverse direction to the creation of entries in the LIBs

- **Label Insertion/Table Lookup**
  - The first router LER1 uses the LIB table to find the next hop and request a label for the specific FEC
  - Subsequent routers just use the label to find the next hop.
  - Once the packet reaches the egress LSR (LER4), the label is removed and the packet is supplied to the destination.
MPLS Operation

LSP Creation and Packet Forwarding through an MPLS Domain
MPLS operation

Packet Forwarding

- examining the path of a packet as it moves from LER1, the ingress LSR, to LER4, the egress LSR
- LER1 may not have any labels for this packet as it is the first occurrence of this request.
  In an IP network
- LER1 will initiate a label request toward LSR1.
- Each intermediary router will receive a label from its downstream router starting from LER4 and going upstream till LER1
- LER1 will insert the label and forward the packet to LSR1
- Each subsequent LSR will examine the label in the received packet, replace it with the outgoing label, and forward it.
- When the packet reaches LER4, it will remove the label because the packet is departing from an MPLS domain and deliver it to the destination
MPLS operation

SDL based implementation two LDPs
Other protocol specification languages

- SPIN (Simple ProMeLa Interpreter)
- Estelle
- LOTOS (Language Of Temporal Ordering Specifications)
- CPN (Colored PetriNets)
- Uppaal
- UML (Unified Modeling Language)