Estelle Specification of MIL-STD-188-220B

Datalink Layer Class 3 (Types 1 and 4) Operation

Protocol Engineering Laboratory
Department of Computer and Information Science
University of Delaware

Paul D. Amer
Greg Burch
Mariusz Fecko
Adarsh Sethi
M. Umit Uyar
Dong Zhu

Date: 4/2/96
Last revised: 8/18/98
Revised March 1997 - Removal of XID/NCTNC
Revised: Aug 97 Class 3 becomes C; Immediate Retrans
Update DL_Unitdata/Status.primitives
Revised: Dec 97 Update DL primitives according to
19Dec97 comment resolution
Revised: Aug 98 Modified Type4 SAP Component:
1) removed quiet mode from Type4 SAP;
2) replaced variables 'buffers_full' and 'last_sent'
   with 'v_busy_status';
3) changed logic in transition TYPE4SAP_3 where operator
   sets/clears station’s busy status;
4) added (a) mark destination as FREE, and (b) stop
   dailtalk layer PDU upon receiving DRRR;
5) copied 'Notify UL of successful ack' from transition
   TYPE4SAP_6 to TYPE4SAP_8 (since both DRR and DRNR
   are an ack!);
6) made processing queued entries in transition
   TYPE4SAP_16 periodic as in Type1 SAP spec.

Note: * denotes a recommendation not currently specified in MIL-STD-188-220A.

Definitions:
(for convenience of reference and consistency in the Estelle code.)
request: This is an internal item used by the SAP component only. The "queue"
   is, as the name implies, a sequence of outstanding requests. The "queue"
   is, as the name implies, a sequence of outstanding requests. The request is a representation of a Protocol Request
   (e.g., DL_Unitdata_req) which the SAP component knows how to handle. Included in the request is the request_id, which is the id assigned to the Request by the entity that initiated it
   (e.g., OPERATOR, NL).
message: An internal exchange of information between the SAP component and
   Station component of the Estelle architecture. These "messages" are
   used only locally, and consist of a DL_PDU, unique message_id, and P-bit (from SAP to Station only).
transmission: This signifies the bit pattern that the datalink layer sends to
   physical layer; as such, a transmission consists of a transmission
   header, and one or more "DL_PDU"s (after they have seen
   bit-stuffing, FEC, TDC, and scrambling).
DL_PDU: A datalink layer PDU (with length information) before applying FEC,
   TDC, data scrambling and 0 bit stuffing.

Other: There are two data structures used implicitly by both the
   type4SAP and type1SAP components. These
   structures are not declared explicitly, and as such their detailed
   contents and organizations are not listed here. The first such structure
   is the "queue." This structure is, as the name suggests, a FIFO list of
   outstanding requests (Test Requests, Unitdata Requests) waiting to be
   handled by the type1SAP component. The second structure does not have a name
   and is referenced by procedures such as "add_message." This structure
   holds information about all outgoing PDU's requiring acknowledgements
   (including transmission count, which stations have responded, what the
   DL_Unitdata_id is for the corresponding request which resulted in the
   construction of that DL_PDU, etc.)

Also, we have implemented the busy-status table and timers as
   2 orthogonal timers (one for type 1 and one for type 4). When a type 1
   UNRRC or UNRNR message comes in, a copy is forwarded to
   both the Type 1 and Type 4 SAP components, and both mark their
   stations as busy. When a type 1 CLEAR (UNRC or whatever) comes in,
   no copy is forwarded to the type4SAP component... it must wait to
   timeout. This method of specifying the busy-status table does not
   preclude a one-timer, one-table implementation.

specification datalink_layer systemactivity;
default
   individual queue;
timescale millisecond;
const
   K = any integer; ( 5-20; the maximum number of outstanding PDU's in type 4 )
   N4 = any integer; ( Type 1 max retransmission count )
   N3 = any integer; ( Type 4 max retransmission count )
   NS = any integer; ( Number of stations > 0 )
   NS_LESS_1 = any integer; ( NS - 1 )
   NULL_ID = -1; ( No ID needed for station message )
   NEED_COUPLED_ACK = 1;
   NOT_NEED_COUPLED_ACK = 0;
   TYPE4_ID_NUM_MAX = 255; ( Max value for PDU ID number in type 4 operation )

{ Estelle-defined transition firing priorities }  
   LOW = 2;
   MEDIUM = 1;
   HIGH = 0;
{ Timer values }
   BUSY_TIMER_TIMEOUT_PERIOD = any integer;
   QUEUED_REQUEST RETRIEVAL_PERIOD = any integer; { Fecko, 05/30/97 } 

{ Link addresses }
   GLOBAL_MULTICAST_ADDRESS = 127;
   NETCON_ADDRESS = 2; { Amer, 8/27/97 }  
   IMMEDIATE_RETRANS_ADDRESS = 3;  
   NET_ENTRY_ADDRESS = 1;  
   ALL_ADDRESSEE_LIST_SIZE = any integer;
   MAX_DL_UNITDATA_ID = 16;
   STATION_MESSAGE_TABLE_SIZE = any integer;

{ Other }  
   type 8_bit_field = integer; ( More precisely: array [0..7] of bit )
   type 16_bit_field = integer; ( More precisely: array [0..15] of bit )  
   type 32_bit_field = integer; ( More precisely: array [0..31] of bit )
   type DL_UNITDATA_ID = integer;  
   type_FEC_TDC_scrambling = type_8_bit_field; 
   type P_bit = 0..1; 
   type DL_PDU_type = (UI_COMMAND, 
                     URR_COMMAND, URR_RESPONSE, 
                     UNR_COMMAND, UNR_RESPONSE, 
                     TEST_COMMAND, TEST_RESPONSE);
   type address = type 8_bit_field;
   type all_addressee_list = array [1..ALL_ADDRESSEE_LIST_SIZE] of
record
    address: type_address;
    P_bit: type_P_bit;
end;

type_data = ...; (variable length bit string)
type_DL_PDU_addresssee_list = array [1..MAX_NO_OF_DEST_ADDRESS] of
    record
        address: type_address;
        P_bit: type_P_bit;
    end;

type_id = integer; (*

type_immediate_retrans_req = (YES_IMMED_RETRANS, NO_IMMED_RETRANS); (Amer, 8/27/97)
type_individual_address = 4..95; ( 1-Net Entry; 2-NETCON; 3-Immediate Retrans;
96-126-Multicast Groups; 127-Global ) (Amer)
type_link_parameter_table = record
    message_number: type_8_bit_field;
    unique_id: type_32_bit_field;
    group_address: type_32_bit_field;
    link_address: type_8_bit_field;
    station_class: type_8_bit_field;
    robust_frame_format: type_8_bit_field;
    EDC_mode: type_8_bit_field;
    net_busy_detect_time: type_16_bit_field;
    equipment_preamble_time: type_16_bit_field;
    key_time: type_16_bit_field;
    equipment_postamble_time: type_16_bit_field;
    carrier_dropout_time: type_8_bit_field;
    equipment_lag_time: type_16_bit_field;
    tolerance_time: type_16_bit_field;
    processing_time: type_8_bit_field;
    MAD_method: type_8_bit_field;
    subscriber_rank: type_8_bit_field;
    number_of_stations: type_8_bit_field;
    urgent_percent: type_8_bit_field;
    priority_percent: type_8_bit_field;
    traffic_load: type_8_bit_field;
    max_transmit_seconds: type_8_bit_field;
    physical_encapsulation: type_8_bit_field;
    data_link_encapsulation: type_8_bit_field;
    max_UIT_DIA_L_info_bytes: type_16_bit_field;
    type2_ack_timer: type_16_bit_field;
    type2_K_window: type_8_bit_field;
    type4_ack_timer: type_16_bit_field;
    type4_K_window: type_8_bit_field;
    quiet_mode: type_8_bit_field;
end;

type_DL_PDU = record
    DL_PDU: type_data;
    length: type_data_length;
end;

type_type1SAP_message = type_DL_PDU;
type_type4SAP_message = type_DL_PDU;
type_station_message = type_DL_PDU;
type_station_message_table = array [1..STATION_MESSAGE_TABLE_SIZE] of
    record
        message_id: type_message_id;
        message: type_DL_PDU;
        from_component: type_from_component;
        precedence: type_precedence;
        P_bit: type_P_bit;
    end;

type_address = record
    address: type_address;
    P_bit: type_P_bit;
end;

type_addr_list = record
    number_of_address: integer;
    address: array [0..NS_LESS_1] of type_address;
end;

type_assigned_address_table = array [type_individual_address] of
    record
        unique_id: type_unique_id;
        address: type_address;
    end;

type_busy = (STATION_BUSY, STATION_FREE);

********************************************************************************
| Functions/Procedures Used In Module m_station_component                    |
|********************************************************************************

function get_DL_PDU_type(data: type_DL_PDU): type_DL_PDU_type; primitive;

function get_P/F bit of a given DL_PDU
    function get_PF_bit(message: type_DL_PDU): type_P_bit; primitive;

function get the SAP a PDU is aimed for
    function get_typeSAP(station_message: type_DL_PDU): type_service_type; primitive;

function get source address of a given DL_PDU
    function get_src_address(station_message: type_DL_PDU): type_address; primitive;

function get_unique ID of a station who send the DL_PDU
    function get_unique_id(DL_PDU: type_DL_PDU): type_unique_id; primitive;

function determine whether the type1 message is a URNRC or URNRR PDU
    function URNR_message(station_message: type_DL_PDU): boolean; primitive;

********************************************************************************
| Functions/Procedures Used In Module m_type1SAP_component                  |
********************************************************************************

procedure add_message(message_id: integer;
    message: type_type1SAP_message;
    nl_id: type_id); primitive;
function get_data_length(message : type_station_message) : type_data_length; primitive;  
( For abstraction purposes. )

function get_dest_addr_list(message : type_station_message) : type_addr_list; primitive;  
( For abstraction purposes. )

function get_message_id_being_acked(message : type_station_message): integer; primitive;  
( Get the message_id being responded to in the current URR or URNR  
  sent from the station component in argument "message". )

function get_message_transmission_count(message_id : integer) : integer; primitive;  
( Return the transmission count for the DL_PDU assoc. with id message_id )

function get_message_type(message : type_station_message) : type_incoming_DL_PDU_type; primitive;  
( For abstraction purposes. )

function all_acked(message_id : integer) : boolean; primitive;  
( Return TRUE if all acks for msg_id are received, else return FALSE. )

function choose_message_id : integer; primitive;  
( Return an unused message id number )

function create_busy_address_list(addr_list: type_addr_list): type_addr_list; primitive;  
( Take an address list, check all entries against the array BUSY.  
  Return an address list containing all busy addresses. )

function create_non_busy_address_list(addr_list: type_addr_list) : type_addr_list; primitive;  
( Take an address list, check all entries against the array BUSY.  
  Return an address list containing all non-busy addresses. )

function create_queued_request(addr_list : type_addr_list;  
  src_addr : type_address;  
  precedence : type_precedence;  
  acknoack : type_ack_required;  
  immediate_retrans_req : type_immediate_retrans_req;  
  request_id : type_id) : type_queued_request; primitive;  
( Make a request from the given information )

function create_type1SAP_message(userdata : type_data;  
  addr_list : type_addr_list;  
  data_length : type_data_length;  
  P_bit: type_P_bit;  
  immediate_retrans_req : type_immediate_retrans_req) : type_type1SAP_message; primitive;  
( Make a message for the station component out of the given args.  
  If an immediate retransmission is requested, include address 3 in the  
  address list in its appropriate place. See restrictions in 188-220  
  Section 3.4.2.2.2.2.5) (Amer 9/4/97)

function create_updated_type1SAP_message(message_id : integer) : type_type1SAP_message; primitive;  
( Take an already existing message id and create a new type1SAP message,  
  changing the address list to include only those addresses which  
  have not yet acked (using the ack info for that id as well). )

function get_DL_Unitdata_id_from_id(message_id : integer) : type_DL_Unitdata_id; primitive;  
( For abstraction purposes. )

function get_addr_from_list({var} addr_list: type_addr_list) : type_address; primitive;  
( Remove the first address from the list addr_list, and returns it.  
  Side effect of changing addr_list IS permanent and intended. )

function get_data(message : type_station_message) : type_data; primitive;  
( For abstraction purposes. )
function buffers_free_event: boolean; primitive;
{ Return TRUE if buffers have become free without being
recognized as such. Once this procedure returns TRUE, it
will return FALSE until the buffers become free anew. }

function buffers_full_event: boolean; primitive;
{ Return TRUE if buffers have become full without being
recognized as such. Once this procedure returns TRUE, it
will return FALSE until the buffers become full anew. }

function create_type4_queued_request(addr_list: type_addr_list;
function get_request_src_addr(request: type_queued_request): type_address; primitive;
{ Make a request from the given information }

function get_request_type(request: type_queued_request): type_request_type; primitive;
{ For abstraction purposes. }

function get_topology_update_id: type_topology_update_id; primitive;
{ This could be the topology_update_id field in DL_Unitdata_Reg; or if a control PDU
is to be transmitted, use the newest topology_update_id from network layer }

procedure increment_trans_count(msg_id: integer); primitive;
{ Increment the transmission count for the DL_PDU assoc. with id message_id }

function make_addr_list(address: type_address): type_addr_list; primitive;
{ Type conversion from type_address to type_addr_list }

procedure process_queue_entry; primitive;
{ Similar to transition TYPE1SAP_2 below. Must split the address list into busy
and nonbusy addresses, create type1SAP message for non-busy destinations, and
queue the balance of the request for later try }

procedure queue_request(request: type_queued_request); primitive;
{ Add the request to the "queue" }

procedure remove_message(msg_id: integer); primitive;
{ Remove and return head of queue. Decreases variable current_no_queue_entries by 1. }

function remove_queued_request: type_queued_request; primitive;
{ Remove and return head of queue. Decreases variable current_no_queue_entries by 1. }

function test_neighbor_detected(msg_id: integer): type_neighbor_detected; primitive;
{ Test to see if a new neighbor has been detected }   {Amer, 8/27/97}

procedure update_ack_list(msg_id: integer; addr: type_address); primitive;
{ Mark addr as having replied to id msg_id }

procedure update_addr_table(addr: type_address); primitive;
{ Update address list with argument addr (if necessary). }

function assign_free_type4_id_num: integer; primitive;
{ Return an unused and allowed message id number }
function type4_all_acked(message_id : integer) : boolean; primitive;
{ Return TRUE if all acks for msg_id are received, else return FALSE. }

function type4_get_nonresponding_addresses(message_id : integer) : type_addr_list; primitive;

procedure type4_increment_trans_count(msg_id : integer); primitive;
{ Increment transmission count for the DL_PDU assoc. with id message_id }

procedure type4_process_queue_entry; primitive;
{ Similar to transition TYPE4SAP_2 below. Must split the address list into busy and nonbusy addresses, create type4SAP message for non-busy destinations, and requeue the balance of the request for later try }

procedure type4_remove_message(msg_id : integer); primitive;
{ Remove the message with (unique) id msg_id from the type4SAP structure }

function type4_remove_queued_request : type_queued_request; primitive;
{ Remove and return head of queue. Decrease variable type4_current_no_queue_entries by 1. }

procedure type4_update_ack_list(msg_id : integer; addr : type_address); primitive;
{ Mark addr as having replied to id msg_id }

****** Functions/Procedures Used In Module m_ack_timer ******
{ Compute the time required for coupled responses to be issued, in succession, by a number of stations equal to number_acks }

****** Functions/Procedures Used In Module m4_ack_timer ******
{ Return current value for the type 4 timeout }

****** Functions/Procedures Used In Module m_station_component ******
{ Test if the DL_PDU is valid. A DL_PDU is invalid if it has one or more of the following:
  a. not bounded by a beginning and ending flag
  b. too short
  c. too long
  d. has an invalid address or control field
  e. has an FCS error }

function DL_PDU_valid(DL_PDU: type_DL_PDU): boolean; primitive;

function get_no_expected_acks (DL_PDU: type_DL_PDU) : integer; primitive;
{ Get number of expected acks }  {Fecko}

procedure buffer_message_in_station_message_table(var station_message_table: type_station_message_table;message_id: type_message_id;message: type_DL_PDU;from_component: type_from_component;precedence: type_precedence); primitive;

procedure construct_and_concatenate_DL_PDU_of_highest_priority(var transmission_to_be_sent: type_data;var transmission_length: type_data_length;var need_coupled_ack: boolean; { TRUE if coupled ack is expected }
{ Get message(s), applying FEC, TDC, data scrambling (if appropriate) and 0 bit stuffing, concatenate them together if desired and possible })

procedure construct_coupled_ack(var coupled_ack: type_DL_PDU;src_address: type_address;DL_PDU_received: type_DL_PDU); primitive;
{ Construct a coupled ack for the DL_PDU_received }
var FEC_TDC_scrambling: type_FEC_TDC_scrambling); primitive;

procedure init_link_parameter_table(
  var link_parameter_table: type_link_parameter_table); primitive;

procedure init_station_message_table(
  var station_message_table: type_station_message_table); primitive;

{ Assign an unique ID to this station }
procedure init_unique_id(var unique_id: type_unique_id); primitive;

{ Test if the PDU is a DRNRC }
function is_DRNRC(PDU : type_DL_PDU) : boolean; primitive;

{ Test if the received DL_PDU is coupled ack or not }
function is_coupled_ack(DL_PDU_received: type_DL_PDU): boolean; primitive;

{ Test if the message is addressed to this station }
function is_for_me(DL_PDU_received: type_DL_PDU; my_address: type_address): boolean; primitive;

{ Look in transmission for all Type 4 DIA PDU's. Make a list of the the id numbers in those PDU's so that the ack timers can be started when the transmission is sent }
function make_type4_ack_list(v_transmission_to_be_sent : type_data) : type_type4_ack_list; primitive;

{ Having retrieved no_retrieved DL_PDU from transmission, test if there are more }
function more_DL_PDU_in_transmission(
  transmission: type_data; transmission_length: type_data_length;no_retrieved: integer): boolean; primitive;

{ First determine the source address of argument PDU. Then search the Station Message table (outgoing messages waiting for transmission) for any messages with that address as a destination AND a transmission count greater than 1. Any such messages have the address in question deleted from their destination list. If that is the only destination, the message is removed entirely. }
procedure purge_retrans_queue(PDU : type_DL_PDU); primitive;

{ When a response is received, source address of response is removed }
procedure remove_addressee_from_DL_PDU_addressee_list(
  addressee: type_address;
  var DL_PDU_addressee_list: type_DL_PDU_addressee_list); primitive;

{ Check parameters Delay, Throughput, and Reliability. Determine from the table provided in MIL-STD-188-220 which type of service is desired for these parameters. Then check what type of services are available. Return the type of service to be USED given the computed desired service. }
function service_type_to_be_used(
  delay_req: type_delay;throughput: type_throughput;reliability: type_reliability) : type_service_type; primitive;

function is_immediate_retrans_requested(
  delay_req: type_delay;
  throughput: type_throughput;
  reliability: type_reliability;
  precedence: type_precedence) : type Immediate_retrans_req; primitive;

{ Check if type4 ack list is empty }
function type4_ack_list_empty(type4_ack_list: type_type4_ack_list): boolean; primitive;

****************************
{ Channels Definitions } ****************************

channel NL_DL (network, datalink);
by network:
  DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;  
                   dest_addr_list: type_addr_list;src_addr: type_address;topology_update_id: type_topology_update_id;precedence: type_precedence;
                   delay_req: type_delay;throughput: type_throughput;reliability: type_reliability;data: type_data;data_length: type_data_length);

by datalink:
  DL_Unitdata_Ind (dest_addr_list: type_addr_list; src_addr: type_address; 
                   topology_update_id: type_topology_update_id;data: type_data; 
                   data_length: type_data_length);
  DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;
                 ack_success_failure: type_ack_success_failure; 
                 address_list: type_addr_list;type2_connection_status: type_type2_connection_status;
                 [ not used in type 1 ] neighbor_detected: type_neighbor_detected); 

channel DL_PL (datalink, physical);
by datalink:
  PL_Unitdata_Req (data: type_data; 
                   data_length: type_data_length;FEC_TDC_scrambling: type_FEC_TDC_scrambling;
                   multidwell: type_multidwell;coupled: type_coupled ({*}));

by physical:
  PL_Unitdata_Ind (data: type_data; 
                   data_length: type_data_length;FEC_TDC_scrambling: type_FEC_TDC_scrambling;
                   multidwell: type_multidwell); 
  PL_Status_Ind (net_activity: type_net_activity;
                 transmission_status: type_transmission_status);

channel OPERATOR_DL (operator, datalink);
by operator:
  OP_quiet_mode_Req (quiet_mode: type_quiet_mode);

channel OPERATOR_type1SAP (operator, type1SAP);
by operator:
  OP_test_req(OP_test_req_id: type_id; (*))
destinations: type_addr_list;
src_addr: type_address;
precedence: type_precedence;
acknoack: type_ack_required;
data: type_data;
data_length: type_data_length);

by type1SAP:
OP_test_ind (result: type_test_result;
src_addr: type_address;
data: type_data;
data_length: type_data_length);

channel OPERATOR_type4SAP (operator, type4SAP);
by operator:
OP_type4_change_busy_status_Req(busy_status: type_type4_busy_status);

channel transmission_to_DL_PDU_convertor (transmission, DL_PDU);
by transmission:
DL_PDU_form_PL_Unitdata_Ind(DL_PDU: type_DL_PDU;
topology_update_id: type_topology_update_id);
( Station Component takes PL_Unitdata_Ind and deconcatenates, destuff 0 bit, 
strip off FEC, TDC, Data Scrambling to get DL-PDU(s) and send them through 
this channel. The queue at the other end holds separated DL_PDU’s. )

channel type1SAP_busy_timer (type1SAP, busy_timer);
by type1SAP:
start_busy_timer;
stop_busy_timer;
by busy_timer:
busy_timer_timeout;

channel type4SAP_busy_timer (type4SAP, busy_timer);
by type4SAP:
start_busy_timer;
stop_busy_timer;
by busy_timer:
busy_timer_timeout;

channel type1SAP_ack_timer (type1SAP, ack_timer);
by type1SAP:
stop_ack_timer;
by ack_timer:
ack_timer_timeout(message_id: type_message_id);

channel type4SAP_ack_timer (type4SAP, ack_timer);
by type4SAP:
stop_ack_timer;
by ack_timer:
ack_timer_timeout(message_id: type_message_id);

channel station_type1SAP_lower_mux (station, type1SAP);
by station:
station_message(message_id: type_message_id;
message: type_DL_PDU;
topology_update_id: type_topology_update_id);

by type1SAP:
type1SAP_message(message: type_type1SAP_message;
precedence: type_precedence;
message_id: type_message_id;
topology_update_id: type_topology_update_id);

channel station_type4SAP_lower_mux (station, type4SAP);
by station:
station_message(message_id: type_message_id;
message: type_DL_PDU;
topology_update_id: type_topology_update_id);

by type4SAP:
type4SAP_message(message: type_type4SAP_message;
precedence: type_precedence;
message_id: type_message_id;
topology_update_id: type_topology_update_id);

channel station_type1SAP_upper_mux (station, type1SAP);
by station:
DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;
dest_addr_list: type_addr_list;
src_addr: type_address;
topology_update_id: type_topology_update_id;
precedence: type_precedence;
acknoack: type_ack_required;
immediate_retrans_req: type_immediate_retrans_req; {Amer 8/27/97}
data: type_data;
data_length: type_data_length);

by type1SAP:
DL_Unitdata_Ind (dest_addr_list: type_addr_list;
src_addr: type_address;
topology_update_id: type_topology_update_id;
data: type_data;
data_length: type_data_length);

DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;
ack_success_failure: type_ack_success_failure;
address_list: type_addr_list;
type2_connection_status: type_type2_connection_status
{ not used in type 1,4 };
neighbor_detected: type_neighbor_detected);  {Amer 8/27/97}

channel station_type4SAP_upper_mux (station, type4SAP);
by station:
DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id;
dest_addr_list: type_addr_list;
src_addr: type_address;
topology_update_id: type_topology_update_id;
precedence: type_precedence;
data: type_data;
data_length: type_data_length);

by type4SAP:
DL_Unitdata_Ind (dest_addr_list: type_addr_list;
src_addr: type_address;
topology_update_id: type_topology_update_id;
data: type_data;
data_length: type_data_length);

DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;
ack_success_failure: type_ack_success_failure;
{renamed from simply ack_failure Amer 9/4/97}
address_list: type_addr_list;
type2_connection_status: type_type2_connection_status
{ not used in type 1,4 };
neighbor_detected: type_neighbor_detected);  {Amer 8/27/97}

channel station_ack_timer (station, ack_timer);
by station:
start_ack_timer(message_id: type_message_id; no_expected_acks: integer);
start_timer(time: type_time);
stop_ack_timer; { Fecko, 05/30/97 }

by ack_timer:
ack_timer_timeout;
ack_timer_stopped; { Fecko, 05/30/97 }
channel station_TP_timer (station, TP_timer); { Fecko, 05/30/97 }
  by station:
    start_TP_timer(no_expected acks: integer);
    stop_TP_timer;
  by TP_timer:
    TP_timer_timeout;

channel station_ack4_timer (station, ack_timer);
  by station:
    start_ack_timer(message_id: type_message_id);
{*************************************}
{            Network layer            }{*************************************}
module m_NL activity;
  ip ip_datalink: NL_DL (network);
end;

body b_NL for m_NL;
  { Network layer not specified here. }
{*************************************}
{           Physical layer            }{*************************************}
module m_PL activity;
  ip ip_datalink: DL_PL (physical);
end;

body b_PL for m_PL;
  { Physical layer not specified here. }
{*************************************}
{          Operator module            }{*************************************}
module m_OPERATOR activity;
  ip ip_datalink: OPERATOR_DL (operator);
  ip_type1SAP: OPERATOR_type1SAP (type1SAP);
  ip_type4SAP: OPERATOR_type4SAP (type4SAP);
end;

body b_OPERATOR for m_OPERATOR;
  { Operator module not specified here. }
{*************************************}
{            Datalink layer           }{*************************************}{  Station Component }{********************}
module m_station_component activity;
  ip ip_LSAP: NL_DL (datalink);
  ip_PSAP: DL_PL (datalink);
  ip_OSAP: OPERATOR_DL (datalink);
  ip_O_type1SAP_comp: OPERATOR_type1SAP (type1SAP);
  ip_O_type4SAP_comp: OPERATOR_type4SAP (type4SAP);
end;

body b_station_component for m_station_component;
  ip ip_type1SAP_lower_mux: station_type1SAP_lower_mux (station);
  ip_type1SAP_upper_mux: station_type1SAP_upper_mux (station);
  ip_type4SAP_lower_mux: station_type4SAP_lower_mux (station);
  ip_type4SAP_upper_mux: station_type4SAP_upper_mux (station);
  ip_ack_timer: station_ack_timer (station);
  ip_TP_timer: station_TP_timer (station);
  ip_type4_ack_timer: array [1..TYPE4_ID_NUM_MAX] of station_ack4_timer (station);
  ip_transmission: transmission_to_DL_PDU_convertor (transmission);
  ip_DL_PDU: transmission_to_DL_PDU_convertor (DL_PDU);

{**************************************************************************}
{ Type1SAP Component }
{**************************************************************************}
module m_type1SAP_component activity;
  ip ip_station_upper_mux: station_type1SAP_upper_mux (type1SAP);
  ip_station_lower_mux: station_type1SAP_lower_mux (type1SAP);
  ip_busy_timer: array [0..NS_LESS_1] of type1SAP_busy_timer (type1SAP);
  ip_ack_timer: type1SAP_ack_timer (type1SAP);
  ip_OPERATOR: OPERATOR_type1SAP (type1SAP);
end;

body b_type1SAP_component for m_type1SAP_component;
state
ACTIVE_STATE;
var
  ack_success_failure_ind : type_ack_success_failure;
  type1_busy_status : array[0..NS_LESS_1] of type_busy;
  non_busy_addresses : type_addr_list;
  non_responding_addresses : type_addr_list;
  current_no_queue_entries : integer;
  id_num : integer;
  index : integer;
  message : type1SAP_message;
  msg_type : type_incoming_DL_PDU_type;
  request : typequeued_request;
  temp_addr : type_address;
initialize
to ACTIVE_STATE
  var count : integer;
  begin
    ack_success_failure_ind := SUCCEED; { Amer 9/4/97 }
    for count := 0 to NS_LESS_1 do
      begin
        type1_busy_status[count] := STATION_FREE;
      end;
  end;

trans
  { Unitdata Request (NOACK) received from upper layer }
  from ACTIVE_STATE to same
  when ip_station_upper_mux.DL_Unitdata_Req
  provided (acknoack = NOACK) name TYPE1SAP_2:
  begin
    non_busy_addresses := create_non_busy_address_list(dest_addr_list);
    if (add size (non_busy_addresses) > 0) then { Fecko, 05/30/97 }
      begin

message := create_type1SAP_message(
data, non_busy_addresses, data_length, 0,
immediate_retrans_req); {Amer 9/4/97}
id_num := 0;
output ip_station_lower_mux.type1SAP_message(
message, precedence, id_num, topology_update_id)
end;
if (addr_list_size (busy_addresses) > 0) then  { Fecko, 05/30/97 }
begin
request_type := DL_REQUEST;
request := create_queued_request(business_addresses, src_addr, precedence,
acknoack, immediate_retrans_req, {Amer 9/4/97}
data, data_length, request_type,
DL_Unitdata_id);
queue_request(request);
end
end;

{ Unitdata Request (YESACK) received from upper layer }
from ACTIVE_STATE to same
when ip_station_upper_mux.DL_Unitdata_Req
provided (acknoack = YESACK)
name TYPE1SAP_3:
begin
busy_addresses := create_busy_address_list(destinations);
if (addr_list_size (busy_addresses) > 0) then  { Fecko, 05/30/97 }
begin
message := create_type1SAP_message(
data, non_busy_addresses, data_length, 1,
immediate_retrans_req); {Test never use immed retrans; Amer 9/4/97}
id_num := choose_message_id;
output ip_station_lower_mux.type1SAP_message(
message, precedence, id_num, get_topology_update_id)
end;
if (addr_list_size (busy_addresses) > 0) then  { Fecko, 05/30/97 }
begin
request_type := DL_REQUEST;
request := create_queued_request(business_addresses, src_addr, precedence,
acknoack, immediate_retrans_req, {Test never use immed retrans; Amer 9/4/97}
data, data_length, request_type,
DL_Unitdata_id);
queue_request(request);
add_message(id_num, message, DL_Unitdata_id)
end
end;

{ Test Request (NOACK) received from Operator }
from ACTIVE_STATE to same
when ip_OPERATOR.OP_test_req
provided (acknoack = NOACK)
name TYPE1SAP-5:
begin
busy_addresses := create_non_busy_address_list(destinations);
if (addr_list_size (non_busy_addresses) > 0) then  { Fecko, 05/30/97 }
begin
message := create_type1SAP_message(
data, non_busy_addresses, data_length, 0,
NO_IMMED_RETRANS); {Test never uses immediate retrans; Amer 9/4/97}
id_num := 0;
output ip_station_lower_mux.type1SAP_message(
message, precedence, id_num, get_topology_update_id)
end;
if (addr_list_size (busy_addresses) > 0) then  { Fecko, 05/30/97 }
begin
request_type := TEST_REQUEST;
request := create_queued_request(business_addresses, src_addr, precedence,
acknoack, NO_IMMED_RETRANS, {Test never uses immed retrans; Amer 9/4/97}
data, data_length, request_type,
OP_test_req_id);
queue_request(request);
end
end;

{ Test Request (YESACK) received from Operator }
from ACTIVE_STATE to same
when ip_OPERATOR.OP_test_req
provided (acknoack = YESACK)
name TYPE1SAP-9:
begin
request_type := TEST_REQUEST;
request := create_queued_request(business_addresses, src_addr, precedence,
acknoack, immediate_retrans_req, {Test never use immed retrans; Amer 9/4/97}
data, data_length, request_type,
OP_test_req_id);
queue_request(request);
add_message(id_num, message, OP_test_req_id)
end
end;

{ Test Request (NOACK) received from Operator }
from ACTIVE_STATE to same
when ip_OPERATOR.OP_test_req
provided (acknoack = NOACK)
name TYPE1SAP-10:
begin
request_type := TEST_REQUEST;
request := create_queued_request(business_addresses, src_addr, precedence,
acknoack, NO_IMMED_RETRANS, {Test never uses immed retrans; Amer 9/4/97}
data, data_length, request_type,
OP_test_req_id);
queue_request(request);
end
end;

{ Message received from Station Component }
from ACTIVE_STATE to same
when ip_station_lower_mux.station_message
name TYPE1SAP-14:
begin
msg_type := get_message_type(message);
case msg_type of
  UIC: begin { TYPE1SAP-6 }
    (Station Component received a UI Command packet)
    output ip_station_upper_mux.DL_Unitdata_Ind(
      get_dest_addr_list(message),
      get_src_addr(message),
      get_data(message),
      get_data_length(message));
  end;
  TESTR: begin { TYPE1SAP-9 }
    (Station Component received a TEST Response packet)
    output ip_OPERATOR.OP_test_ind(SUCCESS,
      get_src_addr(message),
      get_data(message),
      get_data_length(message));
  end;
end;
create_non_busy_address_list (non_responding_addresses);
business_addresses := create_busy_address_list (non_responding_addresses);

if (addr_list_size (non_business_addresses) > 0) then (Fecko, 05/30/97)
begin 
message := create_type1SAP_message(
get_msg_src_addr (message_id),
get_msg_src_addr (message_id),
NO_IMMEDIATE_RETRANS);
output ip_station_lower_mux.type1SAP_message(
message, PREC_ROUTINE, 0, topology_update_id);
end;

end; {case}
end; {transition}

{ Ack timer for current outgoing packet (at lowest level) times out, and this is the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout provided (get_message_transmission_count (message_id) = N4)
name TYPE1SAP_16:
begin
{ set ack_success_failure_ind and connection_status first, then...}ack_success_failure_ind := FAIL;   {Amer 9/4/97}
output ip_station_upper_mux.DL_Status_Ind(
get_DL_Unitdata_id_from_id (message_id),ack_success_failure_ind,
get_nonresponding_addresses (message_id), connection_status,
test_neighbor_detected (message_id));  {Amer 8/27/97}
ack_success_failure_ind := SUCCEED;   {Amer 9/4/97}
remove_message (message_id);{ Fecko, 05/30/97 - removed marking non-responding stations as busy }
end;

{ Any station which was BUSY has its busy_timer timeout }
from ACTIVE_STATE to same
any i: 0..NS_LESS_1 do
when ip_busy_timer[i].busy_timer_timeout name TYPE1SAP_17:
begin
{ Note that this does NOT constitute acknowledgment of the corresponding UI PDU (see 12MAR96,30APR96 WG meeting notes) 
index := addr_index (get_src_addr (message_id));   {Fecko, 2/3/97 Amer} 
output ip_station_lower_mux.type1SAP_message(
message, PREC_ROUTINE, 0, topology_update_id);
end;

end; {case}
end; {transition}

{ Ack timer for current outgoing packet (at lowest level) times out, but this is not the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout provided (get_message_transmission_count (message_id) < N4)
name TYPE1SAP_15:
begin
increment_trans_count (message_id);
non_responding_addresses := get_nonresponding_addresses (message_id);
non_busy_addresses :=
end;
end; { b_type1SAP_component }

end;

end; { Type4SAP Component }

end;

module m_type4SAP_component activity;
ip ip_station_upper_mux: station_type4SAP_upper_mux (type4SAP);
ip_ack4_timer: array [1..TYPE4_ID_NUM_MAX] of type4SAP_ack_timer (type4SAP);
ip_busy_timer: array [0..NS_LESS_1] of type4SAP_busy_timer (type4SAP);
ip_station_lower_mux: station_type4SAP_lower_mux (type4SAP);
ip_OPERATOR: OPERATOR_type4SAP (type4SAP);

end;

body b_type4SAP_component for m_type4SAP_component;
state
ACTIVE_STATE;

var
DRRC_data : type_data;
DRRC_data_length : type_data_length;
DRRR_data : type_data;
DRRR_data_length : type_data_length;
DRNRC_data : type_data;
DRNRC_data_length : type_data_length;
ack_success_failure_ind : type_ack_success_failure;    {Amer 10/97}
busy_addresses : type_addr_list;
v_busy_status : type_type4_busy_status;
connection_status : type_type2_connection_status;
global_addr_list : type_addr_list;
index : integer;
msg_type: type_incoming_type4_DL_PDU_type;
non_busy_addresses : type_addr_list;
non_responding_addresses : type_addr_list;
number_of_addresses : integer;
reply_to_id : integer;
reply_data : type_data;
reply_data_length : type_data_length;
request : type_queued_request;
request_type : type_request_type;
temp_addr : type_address;
type4_current_no_queue_entries : integer;
type4_id_num : integer;
type4_no_outstanding_frames : integer;
type4_busy_status : array [0..NS_LESS_1] of type_busy;
v_message : type_type4SAP_message;
v_precedence : type_precedence;

initialize
to ACTIVE_STATE
var count : integer;
begin
for count := 0 to NS_LESS_1 do
begin
  type4_busy_status[count] := STATION_FREE;
end;
type4_no_outstanding_frames := 0;
type4_current_no_queue_entries := 0;
initialize_type4_used_parameters;

trans
{ Unitdata Request received from upper layer }
from ACTIVE_STATE to same
when ip_station_upper_mux.DL_Unitdata_Req
name TYPE4SAP_2:
begin
if type4_no_outstanding_frames >= K then
begin
  request_type := DL_REQUEST;
  request := create_type4_queued_request(dest_addr_list,
src_addr, precedence,
data, data_length,
request_type, DL_Unitdata_id);
  queue_type4_request(request);
else
begin
  non_busy_addresses := create_non_busy_address_list(dest_addr_list);
  busy_addresses := create_busy_address_list(dest_addr_list);
  v_message := create_type4SAP_message(data, non_busy_addresses,
data_length);
  type4_id_num := assign_free_type4_id_num;
  output ip_station_lower_mux.type4SAP_message(
    v_message, precedence, type4_id_num, topology_update_id);
  add_type4_message(type4_id_num, v_message, DL_Unitdata_id);
  type4_no_outstanding_frames := type4_no_outstanding_frames + 1;
  request_type := DL_REQUEST;
  request := create_type4_queued_request(busy_addresses,
src_addr, precedence,
data, data_length,
request_type, DL_Unitdata_id);
  queue_type4_request(request);
end;
end;

{ DRR Request received from operator } {Rewritten by Fecko, 08/05/98}
from ACTIVE_STATE to same
when ip_operator.OP_type4_change_busy_status_Req
name TYPE4SAP_3:
begin
  type4_id_num := 256;    { not a DIA PDU, so *invalid* id_num passed }
  v precedeace := PREC_ROUTINE;
  if (busy_status <> v_busy_status) and (busy_status = TYPE4_NOTBUSY) then
begin
  v_busy_status := TYPE4_NOTBUSY;
  v_message := create_type4SAP_message (DRRC_data, global_addr_list, 
    DRRC_data_length);
  output ip_station_lower_mux.type4SAP_message(
    v_message, v_precedence, type4_id_num, get_topology_update_id);
end
else if (busy_status <> v_busy_status) and (busy_status = TYPE4_BUSY) then
begin
  v_busy_status := TYPE4_BUSY;
  v_message := create_type4SAP_message (DRNRC_data, global_addr_list, 
    DRNRC_data_length);
  output ip_station_lower_mux.type4SAP_message(
    v_message, v_precedence, type4_id_num, get_topology_update_id);
end
end;
{ Message received from Station Component }
from ACTIVE_STATE to same
when ip_station_lower_mux.station_message
name TYPE4SAP_4to8:
begin
    msg_type := get_type4_message_type(message);
    case msg_type of
        DIAC:  begin { TYPE4SAP_4 }
            { Station Component received a DIA PDU }
            output ip_station_upper_mux.DL_Unitdata_Ind( get_dest_addr_list(message),
            get_src_addr(message),
            topology_update_id, get_data(message),
            get_data_length(message));

            if (v_busy_status = TYPE4_BUSY) then {Fecko, 08/05/98}
                begin
                    reply_data := DRNRR_data;
                    reply_data_length := DRNRR_data_length;
                end
            else
                begin
                    reply_data := DRRR_data;
                    reply_data_length := DRRR_data_length;
                end;

            v_message := create_type4SAP_message(
                reply_data,
                make_addr_list(get_src_addr(message)),
                reply_data_length);

            type4_id_num := 256; { not a DIA PDU, so "invalid"
            id_num passed }

            output ip_station_lower_mux.type4SAP_message(
                v_message, get_precedence(message), type4_id_num,
                topology_update_id);

        end;
        DRRC: begin { TYPE4SAP_5 }
            { Station Component received a DRR Command PDU }
            index := addr_index(get_src_addr(message));
            type4_busy_status[index] := STATION_BUSY;
            output ip_busy_timer[index].start_busy_timer;

        end;
        DRRR: begin { TYPE4SAP_6 }
            { Station Component received a DRR Response packet }
            reply_to_id := get_message_id_being_acked(message);
            type4_update_ack_list(reply_to_id, get_src_addr(message));

            if type4_all_acked(reply_to_id) then
                begin
                    output ip_ack4_timer[reply_to_id].stop_ack_timer;
                    type4_remove_message(reply_to_id);
                    type4_no_outstanding_frames := type4_no_outstanding_frames-1;
                end;

            output ip_station_upper_mux.DL_Status_Ind( get_DL_Unitdata_id_from_id(message_id),
                ack_success_failure_ind,
                type4_get_nonresponding_addresses(message_id),
                connection_status,
                test_neighbor_detected(message_id)); { Amer 10/20/97}

        end; {case}
    end; {transition}

DRNRC: begin { TYPE4SAP_7 }
    { Station Component received a DRNRC Response packet }
    index := addr_index(get_src_addr(message));
    type4_busy_status[index] := STATION_BUSY;
    output ip_busy_timer[index].start_busy_timer;

end;

DRNRR: begin { TYPE4SAP_8 }
    { Station Component received a DRNR Response packet }
    { Note that this DOES constitute acknowledgment of the corresponding DIA PDU }
    reply_to_id := get_message_id_being_acked(message);
    temp_addr := get_src_addr(message);
    index := addr_index(temp_addr);
    type4_update_ack_list(reply_to_id, temp_addr);
    if type4_all_acked(reply_to_id) then
        begin
            output ip_ack4_timer[reply_to_id].stop_ack_timer;
            type4_remove_message(reply_to_id);
            type4_no_outstanding_frames := type4_no_outstanding_frames-1;
        end;

    ack_success_failure_ind := SUCCEED; {Amer 2/17/98}
    output ip_station_upper_mux.DL_Status_Ind( get_DL_Unitdata_id_from_id(message_id),
        ack_success_failure_ind,
        type4_get_nonresponding_addresses(message_id),
        connection_status,
        test_neighbor_detected(message_id)); {Amer 10/20/97}

end; {case}
end; {transition}

URNRC1: begin { TYPE4SAP_9 }
    { Station Component received a URNRC Response packet } (A copy forwarded by the station component to shut off type4 messages too) 
    index := addr_index(get_src_addr(message));
    type4_busy_status[index] := STATION_BUSY;
    output ip_busy_timer[index].start_busy_timer;

end;

URNRR1: begin { TYPE4SAP_10 }
    { Station Component received a URNRR Response packet } (A copy forwarded by the station component to shut off type4 messages too) 
    index := addr_index(get_src_addr(message));
    type4_busy_status[index] := STATION_BUSY;
    output ip_busy_timer[index].start_busy_timer;

end; {case}
end; {transition}
{ Ack timer for a packet times out; not its N2th transmission }
from ACTIVE_STATE to same
any i: 1..TYPE4_ID_NUM_MAX do
when ip_ack4_timer[i].ack_timer_timeout
provided (get_message_transmission_count(message_id) < N2)
name TYPE4SAP_11:
begin
  type4_increment_trans_count(message_id);
  v_message := create_updated_type4SAP_message(message_id);
  output ip_station_lower_mux.type4SAP_message (v_message,
       get_msg_precedence(message_id), message_id, get_topology_update_id);
end;

{ Ack timer for current outgoing packet (at lowest level) times out,
and this is the N4th transmission of that packet }
from ACTIVE_STATE to same
any i: 1..TYPE4_ID_NUM_MAX do
when ip_ack4_timer[i].ack_timer_timeout
provided (get_message_transmission_count(message_id) = N2)
name TYPE4SAP_12:
begin
  set ack_success_failure_ind, then...
  ack_success_failure_ind := FAIL;  {Amer 2/17/98}
  output ip_station_upper_mux.DL_Status_Ind(
       get_DL_Unitdata_id_from_id(message_id), ack_success_failure_ind,
       type4_get_nonresponding_addresses(message_id), connection_status,
       test_neighbor_detected(message_id));  {Amer 10/20/97}
  non_responding_addresses :=
       type4_get_nonresponding_addresses(message_id);
  number_of_addresses := addr_list_size(non_responding_addresses);
  while number_of_addresses > 0 do
    begin
      temp_addr := get_addr_from_list(non_responding_addresses);
      type4_busy_status[temp_index(temp_addr)] := STATION_BUSY;
      output ip_busy_timer[temp_index(temp_addr)].start_busy_timer;
      number_of_addresses := number_of_addresses - 1;
    end;
  type4_remove_message(reply_to_id);
  type4_no_outstanding_frames := type4_no_outstanding_frames - 1;
end;

{ Any station which was BUSY has its busy_timer timeout }
from ACTIVE_STATE to same
any i: 0..NS_LESS_1 do
when ip_busy_timer[i].busy_timer_timeout
name TYPE4SAP_13:
begin
  type4_busy_status[i] := STATION_FREE;
end;

end;

end;  { b_type4SAP_component }

{ Automatic DRR send when buffers become free. }
from ACTIVE_STATE to same
provided buffers_free_event
name TYPE4SAP_15:
begin
  if v_busy_status = TYPE4_BUSY then 
    {Fecko, 08/05/98} 
    begin
      v_busy_status := TYPE4_NOTBUSY;
      v_message := create_type4SAP_message(DRRC_data, global_addr_list,
                                    DRRC_data_length);
      type4_id_num := 256; { not a DIA PDU, so "invalid" id_num passed }
      v_precedence := PREC_ROUTINE;
      output ip_station_lower_mux.type4SAP_message(
       v_message, v_precedence, type4_id_num, get_topology_update_id);
    end;
  end;

{ Periodically process the request at the head of the request queue.
  Transition is made periodic to prevent processing requests in an
  infinite loop. HIGH priority allows to avoid non-determinism, should
  TYPE4SAP_11 and TYPE4SAP_12 become fireable at the same time. }
{ Fecko, 05/30/97 }
from ACTIVE_STATE to same
provided (type4_current_no_queue_entries > 0)
delay (QUEUED_REQUEST_RETRIEVAL_PERIOD)  {Fecko, 08/05/98}
priority HIGH
name TYPE4SAP_16:
begin
  type4_process_queue_entry;
end;

end; { b_type4SAP_component }
{ Request to restart an already running busy timer }
from TIMER_RUNNING to TIMER_RESTART
when ip_type4SAP.start_busy_timer
name BT_3:
begin
  { Change state to restart the timer }
  end;

{ Stop the busy timer }
from TIMER_RUNNING to TIMER_OFF
when ip_type4SAP.stop_busy_timer
name BT_4:
begin
  end;

{ Busy timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(BUSY_TIMER_TIMEOUT_PERIOD)
name BT_5:
begin
  output ip_type4SAP.busy_timer_timeout;
end;

{ Spontaneous transition to restart the timer }
from TIMER_RESTART to TIMER_RUNNING
name BT_6:
begin
  end;
end;

{*******************************
    URR/URNR Busy Timer
*******************************}
module m_URR_URNR_busy_timer activity;
  ip ip_type1SAP: type1SAP_busy_timer (busy_timer);
end;

body b_URR_URNR_busy_timer for m_URR_URNR_busy_timer;

state
  TIMER_OFF, TIMER_RUNNING, TIMER_RESTART;
initialize
to TIMER_OFF
begin
end;

trans
  { Start the busy timer }
  from TIMER_OFF to TIMER_RUNNING
  when ip_type1SAP.start_busy_timer
  name BT_1:
  begin
  end;

  { Request to stop a non-running busy timer }
  from TIMER_OFF to same
  when ip_type1SAP.stop_busy_timer
  name BT_2:
  begin
  end;

  { Request to start an already running busy timer }
  from TIMER_RUNNING to TIMER_RESTART
when ip_type1SAP.start_busy_timer
name BT_3:
begin
  { Change state to restart the timer }
  end;

{ Stop a running busy timer }
from TIMER_RUNNING to TIMER_OFF
when ip_type1SAP.stop_busy_timer
name BT_4:
begin
  end;

{ Busy timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(BUSY_TIMER_TIMEOUT_PERIOD)
name BT_5:
begin
  output ip_type1SAP.busy_timer_timeout;
end;

{ Spontaneous transition to restart the timer }
from TIMER_RESTART to TIMER_RUNNING
name BT_6:
begin
  end;
end;

{**************************
    Type 4 Ack Timer
**************************}
module m4_ack_timer activity;
  ip ip_type4SAP: type4SAP_ack_timer (ack_timer);
  ip_station: station_ack4_timer (ack_timer);
end;

body b4_ack_timer for m4_ack_timer;

state
  TIMER_OFF, TIMER_RUNNING;
var
  delay_value          : type_time;
  message_idnum        : type_message_id;
initialize
to TIMER_OFF
begin
  delay_value := lookup_type4_timeout_value;
end;

trans
  { Start ack timer }
  from TIMER_OFF to TIMER_RUNNING
  when ip_station.start_ack_timer
  name ACK_TIMER_1:
  begin
  end;

  { Request to stop a non-running ack timer }
  from TIMER_OFF to same
  when ip_type4SAP.stop_ack_timer
  name BT_2:
  begin
  end;

  { Request to start an already running ack timer }
  from TIMER_RUNNING to TIMER_RESTART
when ip_type4SAP.start_ack_timer
name BT_3:
begin
  { Change state to restart the timer }
  end;

{ Stop a running ack timer }
from TIMER_RUNNING to TIMER_OFF
when ip_type1SAP.stop_ack_timer
name BT_4:
begin
  end;

{ Busy ack timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(BUSY.Timer_TIMEOUT_PERIOD)
name BT_5:
begin
  output ip_type1SAP.busy_ack_timer_timeout;
end;

{ Spontaneous transition to restart the timer }
from TIMER_RESTART to TIMER_RUNNING
name BT_6:
begin
  end;
end;
begin
    { Ignore }
end;

{ Stop ack timer which is running }
from TIMER_RUNNING to TIMER_OFF
when ip_type4SAP.stop_ack_timer
name ACK_TIMER_3:
begin
    ( This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. ) { Fecko, 05/30/97 }
    output ip_station.ack_timer_stopped; {Fecko}
end;

{ Start ack timer which is already running }
from TIMER_RUNNING to same
when ip_station.start_ack_timer
name ACK_TIMER_4:
begin
    { Ignore - allows timer to continue; do not restart }
end;

{ Timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(delay_value)
name ACK_TIMER_5:
begin
    output ip_type4SAP.ack_timer_timeout(message_idnum); { This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. } { Fecko, 05/30/97 }
    output ip_station.ack_timer_stopped; {Fecko}
end;
end; { b4_ack_timer }

{***************************}
{    Type 1 Ack Timer    }{***************************}
module m_ack_timer activity;
ip ip_type1SAP: type1SAP_ack_timer (ack_timer);
ip_station: station_ack_timer (ack_timer);
end;
body b_ack_timer for m_ack_timer;

state
    TIMER_OFF, TIMER_RUNNING;
var
    delay_value          : type_time;
    message_idnum        : type_message_id;
initialize
to TIMER_OFF
begin
    delay_value := 0;
end;

trans
    ( Start ack timer )
from TIMER_OFF to TIMER_RUNNING
when ip_station.start_ack_timer
name ACK_TIMER_1:
begin
    message_idnum := message_id;
    delay_value := compute_delay(no_expected_acks);
end;

( Stop ack timer which is not running )
from TIMER_OFF to same
when ip_type1SAP.stop_ack_timer
name ACK_TIMER_2:
begin
    ( Ignore )
end;

{ Stop ack timer which is running }
from TIMER_RUNNING to TIMER_OFF
when ip_type1SAP.stop_ack_timer
name ACK_TIMER_3:
begin
    ( This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. ) { Fecko, 05/30/97 }
    output ip_station.ack_timer_stopped; {Fecko}
end;

{ Start ack timer which is already running }
from TIMER_RUNNING to same
when ip_station.start_ack_timer
name ACK_TIMER_4:
begin
    ( Ignore - allows timer to continue; do not restart )
end;

{ Timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(delay_value)
name ACK_TIMER_5:
begin
    output ip_type1SAP.ack_timer_timeout(message_idnum); { This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. } { Fecko, 05/30/97 }
    output ip_station.ack_timer_stopped; {Fecko}
end;
end; { b_ack_timer }

{***************************}
{    Type 1 TP Timer     } { Fecko, 05/30/97 }{***************************}
module m_TP_timer activity;
ip ip_station: station_TP_timer (TP_timer);
classC.AUG97.e

MIL-STD 188-220B Data Link Layer Class C (Types 1&4) Estelle Specification (MAR98)

state
TIMER_OFF, TIMER_RUNNING;

var
delay_value : type_time;
initialize
to TIMER_OFF
begin
delay_value := 0;
end;
trans
{ Start TP timer }
from TIMER_OFF to TIMER_RUNNING
when ip_station.start_TP_timer
name TP_TIMER_1:
begin
delay_value := compute_delay(no_expected_acks);
end;

{ Stop TP timer which is not running }
from TIMER_OFF to same
when ip_station.stop_TP_timer
name TP_TIMER_2:
begin
{ Ignore }
end;

{ Stop TP timer which is running }
from TIMER_RUNNING to TIMER_OFF
when ip_station.stop_TP_timer
name TP_TIMER_3:
begin
end;

{ Start TP timer which is already running }
from TIMER_RUNNING to same
when ip_station.start_TP_timer
name TP_TIMER_4:
begin
{ Ignore - allows timer to continue; do not restart }
end;

{ Timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(delay_value)
name TP_TIMER_5:
begin
output ip_station.TP_timer_timeout;
end;
end; { b_TP_timer }
*****************************************************************************
{ Station Component body definition: 1 phase -- Active }
*****************************************************************************

state
ACTIVE, WAIT_FOR_TP, WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION;

stateset
ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION = [
ACTIVE,
WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION ];

ALL_STATES = [
ACTIVE,
WAIT_FOR_TP,
WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION ];

var
{ Main station component parameters and data structures }
v_FEC_TDC_scrambling: type_FEC_TDC_scrambling;
v_link_parameter_table: type_link_parameter_table;
v_multidwell: type_multidwell;
v_my_address: type_address;
v_net_activity: type_net_activity;
v_quiet_mode: type_quiet_mode;
v_rexmls_count: type_rexmls_count;
v_station_message_table: type_station_message_table;
v_topology_update_id_received: type_topology_update_id;
v_unique_id: type_unique_id;

{ Other station component variables and data structures }
v_DL_PDU_received: type_DL_PDU;
v_DL_PDU_retrieved: type_DL_PDU;
v_DL_PDU_addresses_list: type_DL_PDU_addresses_list;
v_DL_PDU_type: type_DL_PDU_type;
v_all_address_list: type_all_address_list;
v_coupled_ack: type_DL_PDU;
v_from_component: type_from_component;
v_message_id: type_message_id;
v_need_coupled_ack: boolean;
v_n-th: integer;
v_no_of_expected_acks: integer;
v_service: type_service_type;
v_timer: integer;
v_transmission_to_be_sent: type_data;
v_transmission_length: type_data_length;
v_type4_ack_list: type_type4_ack_list;

modvar
URR_URNR_busy_timer: array [0..NS_LESS_1] of m_URR_URNR_busy_timer;
DRR_DRNR_busy_timer: array [0..NS_LESS_1] of m_DRR_DRNR_busy_timer;
type1SAP_component: m_type1SAP_component;
type4SAP_component: m_type4SAP_component;
ack_timer: m_ack_timer;
ack4_timer: array [1..TYPE4_ID_NUM_MAX] of m4_ack_timer;
TP_timer: m_TP_timer; {Fecko/Amer 10/20/97}

initialize to ACTIVE
var
i: integer;
begin
init type1SAP_component with b_type1SAP_component;
init type4SAP_component with b_type4SAP_component;
init ack_timer with b_ack_timer;
for i := 1 to TYPE4_ID_NUM_MAX
init ack4_timer[i] with b4_ack_timer;
for i := 0 to NS_LESS_1 do
init URR_URNR_busy_timer[i] with b_URR_URNR_busy_timer;
for i := 0 to NS_LESS_1 do
init DRR_DRNR_busy_timer[i] with b_DRR_DRNR_busy_timer;
connect ip_transmission to ip_DL_PDU;
for i:= 0 to NS_LESS_1 do
    connect type1SAP_component.ip_busy_timer [i] to
        URR_URNR_busy_timer [i].ip_type1SAP;
for i:= 0 to NS_LESS_1 do
    connect type4SAP_component.ip_busy_timer [i] to
        DRR_DRNR_busy_timer [i].ip_type4SAP;
connect type1SAP_component.ip_ack_timer to ack_timer.ip_type1SAP;
for i:= 1 to TYPE4_ID_NUM_MAX do
    connect type4SAP_component.ip_ack4_timer [i] to
        ack4_timer [i].ip_type4SAP;
attach ip_O_type1SAP_comp to type1SAP_component.ip_OPERATOR;
attach ip_O_type4SAP_comp to type4SAP_component.ip_OPERATOR;
connect ip_type1SAP_lower_mux.type1SAP_message to
    samewhen ip_type1SAP_message.priority MEDIUMname A_1:
begin
    buffer_message_in_station_message_table(  
        v_station_message_table, message_id, message, FROM_TYPE1SAP, precedence);
end;

{ PL_Unitdata_Ind arrives from Physical Layer by way of internal queue }  
{ Fecko, 05/30/97 }from ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to WAIT_FOR_TP
when ip_DL_PDU.DL_PDU_form PL_Unitdata_Ind provided require_coupled_ack(DL_PDU)
priority MEDIUM
name A_6:begin
v_DL_PDU_received := DL_PDU;
if is_for_me(v_DL_PDU_received, v_my_address) then
begin
    v_DL_PDU_type := get_typeSAP(v_DL_PDU_received);
case v_DL_PDU_type of
TYPE1NOACK:
begin
output ip_type1SAP_lower_mux.station_message(  
    v_message_id, v_DL_PDU_received, topology_update_id);
if URNR_message(v_DL_PDU_received) then
    output ip_type4SAP_lower_mux.station_message(  
        v_message_id, v_DL_PDU_received, topology_update_id);
end;
end{case}
end{end{TRANSMISSION}}
{ Wait for Physical Layer transmission until transmit is COMPLETE, IDLE OR ABORTED }
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to ACTIVE
when ip_PSAP.PL_Status_Ind
provided ((v_need_coupled_ack = FALSE) and
( (transmission_status = TRANSMIT_COMPLETE_OR_IDLE) or
( transmission_status = TRANSMIT_ABORTED) ))
priority MEDIUM
name A_7_1:
begin
while not type4_ack_list_empty(v_type4_ack_list) do
begin
  v_timer := get_next_id_that_need_type4_ack(v_type4_ack_list);
  output ip_type4_ack_timer[v_timer].start_ack_timer(v_timer);
end;

end;

{ Wait for Physical Layer transmission if TRANSMIT_IN_PROCESS }
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to WAIT_FOR_TP
when ip_PSAP.PL_Status_Ind
provided ((v_need_coupled_ack = TRUE) and
( (transmission_status = TRANSMIT_COMPLETE_OR_IDLE) or
( transmission_status = TRANSMIT_ABORTED) ))
priority MEDIUM
name A_7_2:
begin
  output ip_ack_timer.start_ack_timer(
    v_message_id, v_no_of_expected_acks);

end;

end;

{ Wait for ack timer to be stopped }
from WAIT_FOR_TP to ACTIVE
begin
  v_n_th := 1;
while more_DL_PDU_in_transmission(data, data_length, v_n_th) do
begin
  v_DL_PDU_retrieved := get_DL_PDU_from_transmission(
    data, data_length, v_n_th, v_FEC_TDC_scrambling, v_multidwell);
  v_topology_update_id_received := get_topology_update_id_from_transmission(data);
  v_n_th := v_n_th + 1;
  if v_DL_PDU_valid(v_DL_PDU_retrieved) then output ip_transmission.DL_PDU_form_PL_Unitdata_Ind(
    v_DL_PDU_received, v_topology_update_id_received);
end;

end;

end;

{ Set v_net_activity according to PL_Status_Ind from Physical Layer if not in WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION }
from ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same
when ip_PSAP.PL_Status_Ind
priority MEDIUM
name C_4:
begin
  v_net_activity := net_activity;
end;

{ Operator ask to change quiet mode }
from ALL_STATES to same
when ip_OSAP.OP_quiet_mode_Req
priority HIGH
name C_5:
begin
  v_quiet_mode := quiet_mode;
end;

{ Forward DL_Unitdata_Req from NL to appropriate SAP_component }
from ALL_STATES to same
when ip_LSAP.DL_Unitdata_Req
priority HIGH
name C_12:
begin
  { check QOS information (delay, throughput, reliability) in the DL_Unitdata_Req to determine which type service is required. From that, determine which type of service is to be used (based on what is available). Forward the request to the appropriate SAP component. }
  v_service := service_type_to_be_used(delay_req, throughput, reliability);
case v_service of

begin
  buffer_message_in_station_message_table(
    v_station_message_table, message_id, message, FROM_TYPE4SAP, precedence);
end;
( End of Active Phase Transitions )
TYPE1ACK:
output ip_type1SAP_upper_mux.DL_Unitdata_Req(
DL_Unitdata_id, dest_addr_list, src_addr,
topology_update_id, precedence, YESACK,
is_immediate_retrans_requested(delay_req, throughput,
reliability, precedence), (Amer 9/4/97) (Amer 2/17/98)
data, data_length);

TYPE4:
output ip_type4SAP_upper_mux.DL_Unitdata_Req(
DL_Unitdata_id, dest_addr_list, src_addr,
topology_update_id, precedence, data, data_length);

end; {case}
end;

{ Forward DL_Unitdata_Ind from type1SAP_component to NL}
from ALL_STATES to same
when ip_type1SAP_upper_mux.DL_Unitdata_Ind
priority HIGH
name C_13:
begin
output ip_LSAP.DL_Unitdata_Ind(dest_addr_list, src_addr,
topology_update_id, data, data_length);
end;

{ Forward DL_Status_Ind from type1SAP_component to NL}
from ALL_STATES to same
when ip_type1SAP_upper_mux.DL_Status_Ind
priority HIGH
name C_18:
begin
output ip_LSAP.DL_Status_Ind(
DL_Unitdata_id, ack_success_failure, address_list,
type2_connection_status, neighbor_detected); {Amer 8/27/97}
end;

end; { b_station_component }

{********************************************************}
{             Main Specification body                    }{********************************************************}
modvar
NL: array [0..NS_LESS_1] of m_NL;
estation_component: array [0..NS_LESS_1] of m_station_component;
OPERATOR: array [0..NS_LESS_1] of m_OPERATOR;
PL: array [0..NS_LESS_1] of m_PL;

initialize
var
i: integer;

begin
for i := 0 to NS_LESS_1 do
begin
init NL[i] with b_NL;
init station_component[i] with b_station_component;
init OPERATOR[i] with b_OPERATOR;
init PL[i] with b_PL;
connect NL[i].ip_datalink to station_component[i].ip_LSAP;
connect station_component[i].ip_PSAP to PL[i].ip_datalink;
end;
end; { end of entire specification }