L6/GCOS 6

SYSTEMS NETWORK ARCHITECTURE MANAGER

Design Specification

706-09-03-002

MASTER

Prepared by:

Richard Bockenek
Airlines and Financial Industries Operation
Honeywell Information Systems Inc.
65 Walnut St.
Wellesley Hills, Mass. 02181

February 29, 1980
Preliminary

TABLE OF CONTENTS

1. INTRODUCTION 1-1
1.1. Scope 1-1
1.2. Software Environment 1-1
1.3. Features 1-1
1.4. User Model 1-2
2. REFERENCES 2-1
3. GLOSSARY 3-1
4. DESIGN OVERVIEW 4-1
4.1. Layers 4-1
4.2. Node Structure 4-3
4.3. Task Scheduling 4-3
5. INTERNAL DESIGN 5-1
5.1. Data Link Control 5-1
5.1.1. Function Description 5-1
5.1.2. Algorithm 5-1
5.2. Path Control 5-1
5.2.1. Function Description 5-1
5.2.2. Algorithm 5-2
5.2.2.1. PC Receive 5-2
5.2.2.2. PC Send 5-3
5.2.2.3. TH Format Check 5-3
5.2.2.4. DAF' Validity Check 5-4
5.3. Transmission Control 5-5
5.3.1. Function Description 5-5
5.3.2. Algorithm 5-6
5.3.2.1. CSC Receive 5-6
5.3.2.2. CPMGR Receive 5-7
5.3.2.3. CPMGR Send 5-8
5.3.2.4. Session Control 5-9
5.3.2.5. SQN Validity Check 5-10
5.3.2.6. BIND Receive 5-10
5.4. DATA FLOW CONTROL 5-11
5.4.1. Function Description 5-11
5.4.1.1. Boundary Layers 5-11
5.4.1.2. Functions 5-11
5.4.2. Algorithm 5-12
5.4.2.1. DFC Receive 5-12
5.4.2.2. Receive Request 5-12
5.4.2.3. Receive Response 5-13
5.4.2.4. DFC Send 5-13
5.4.2.5. Send State Checks 5-14
5.4.2.6. Send Request 5-14
5.4.2.7. Send Response 5-15
5.4.2.8. CT Entry Add 5-15
5.4.2.9. CT Entry Add or Update 5-16
5.4.2.10. CT Key Search 5-16
5.4.2.11. CT Update 5-17
5.4.2.12. Receive State Checks 5-17
5.4.2.13. Receive Request State Checks 5-17
5.4.2.14. Receive Response State Checks 5-17
5.4.2.15. Generate RH 5-18
5.4.2.16. Send Request State Checks 5-19
5.4.2.17. Process Normal Response 5-19
5.4.2.18. Generate Normal Response 5-20

1. INTRODUCTION

1.1. Scope

This document provides the design specification of a Systems Network Architecture (SNA) manager supporting cluster controller node (PU.T2) operations. The SNA manager will run on a Honeywell Series 60 (Level 6) minicomputer under the GCOS 6 operating system.

1.2. Software Environment

The SNA manager provides support for communications applications in a SNA/SDLC PU.T2. It runs under MOD 400 and MOD 600 operating system in which the Honeywell SLDC line protocol handler has been configured. The SNA manager, along with the SDLC lph, performs all actions necessary to send end user data through the network. It does not change or examine the end user data. This function is left to the application which uses the SNA manager. The SNA manager is linked with the application program and runs as a task group under GCOS 6.

1.3. Features

The SNA manager supports SNA cluster controller node (PU.T2) capabilities. These capabilities include:

- up to 256 network addressable units
- FM Profile 3
- TS Profile 3
- subset of SNA requests and responses

- SNA Commands

  SC ACTPU
  DACTPU
  ACTLU
  DACTLU
  BIND
  UNBIND
  SDT
  CLEAR

  DFC CANCEL
  CHASE
  LUSTAT
  SHUTD
  SHUTC
  RTR
  BID
  SIGNAL

  FMD DATA
  REQMS
  RECFMS

- FM Data Protocols:

  HDX Flip/Flop
  Change Direction
  Chaining
  BRACKETS
  BRACKET TERMINATION Rule 1
  CONTENTION between brackets (LU.T2)

1.4. User Model

The SNA manager supports SNA/SDLC communications via five types of requests. Each of these requests are issued via the $RQTSK task request call to various tasks in the SNA manager. The first request is made to identify a buffer pool to the SNA manager. The SNA manager uses buffers in this pool to perform transmit and receive operations on the data link. The second request is used by the application task to acquire data buffers for use in sending data to the remote node (the SNA manager releases transmit buffers). The third request is used by the application task to release data buffers used to receive data from the data link (the SNA manager acquires receive data buffers). The last request is used to pass buffers containing end user data to the SNA manager for evenual transmission on the data link. These requests are described in detail in Chapter 6.

2. REFERENCES

1. Systems Network Architecture General Information, IBM Document No. GA27-3102-0, First Edition, January, 1975

2. System Network Architecture Format and Protocol Reference Manual, IBM Document No. SC30-3112-1, Second Edition, June, 1978

3. IBM 3270 Information Display System Component Description, GA27-2749-8, Ninth Edition, December, 1978

4. INCOTERM 3276 SNA/SDLC Remote Emulator Functional Specification, Memo PKG-79-012, June 22, 1979

5. SNA/SDLC Study Report, Memo PKG-79-017, August 17, 1979

6. L6/GCOS 6 SYNCHRONOUS DATA LINK CONTROL LINE PROTOCOL HANDLER, Design Specification, 706-09-03-001, January 11, 1980, Preliminary.

3. GLOSSARY

ACTLU	ACTIVATE LOGICAL UNIT
ACTPU	ACTIVATE PHYSICAL UNIT
BB	Begin Bracket
BBI	Begin Bracket Indicator
BBIU	Begin BIU
BBIUI	BBIU Indicator
BC	Begin Chain
BCI	Begin Chain Indicator
BIND	BIND SESSION
BIU	Basic Information Unit
BLU	Basic Link Unit
BSM	Bracket State Manager
BTU	Basic Transmission Unit
CD	Change Direction
CDI	Change Direction Indicator
CPM	Connection Point Manager
CPMGR	CPM
CSC	Common Session Control
CT	Correlation Table
CTGY	Category
DACTLU	DEACTIVATE LOGICAL UNIT
DACTPU	DEACTIVATE PHYSICAL UNIT
DAF'	Destination Address Field
DFC	Data Flow Control
DLC	Data Link Control
DR1	Definite Response 1
DR1I	DR1 Indicator
DR2	Definite Response 2
DR2I	DR2 Indicator
DT	Data Traffic
EB	End Bracket
EBI	End Bracket Indicator
EBIU	End BIU
EBIUI	EBIU Indicator
EC	End Chain
ECI	End Chain Indicator
EFI	Expedited Flow Indicator
ERI	Exception Response Indicator
EXR	Exception Request
FID	Format Identification Field
FM	Function Management
FMD	Function Management Data
FMH	Function Management Header
FSP	First Speaker
HSAP	Half-Session Activation Parameters
HSID	Half-Session Identification
INB	In Bracket
IPR	Isolated Pacing Response
LD	Lost Data
LDI	Lost Data Indicator
LU	Logical Unit

LUSTAT	LOGICAL UNIT STATUS
LU_Ti	Logical Unit type i
MPF	Mapping Field
NAU	Network Addressable Unit
NC	Network Control
NG	No Good
NS	Network Services
OAF	Origin Address Field
PAC	Pacing Request/Response
PC	Path Control
PI	Pacing Indicator
PIU	Path Information Unit
PLU	Primary Logical Unit
PS	Presentation Services
PU	Physical Unit
PU_Tj	Physical Unit type j
QR	Queued Response
QRI	Queued Response Indicator
RC	Return Code
RECFMS	RECORD FORMATTED MAINTENANCE STATISTICS
REQMS	REQUEST MAINTENANCE STATISTICS
RH	Request/Response Header
RQ	Request
RQD	Request Indicating Definite Response
RQE	Request Indicating Exception Response
RQN	Request Indicating No Response
RRI	Request/Response Indicator
RSP	Response
RTI	Response Type Indicator
RTR	READY TO RECEIVE
RU	Request/Response Unit
SC	Session Control
SDI	Sense Data Indicator
SDT	START DATA TRAFFIC
SHUTC	SHUTDOWN COMPLETE
SHUTD	SHUTDOWN
SIG	SIGNAL
SLU	Secondary Logical Unit
SNA	Systems Network Architecture
SNC	Sense Code
SNF	Sequence Number Field
SQN	Sequence Number
SSCP	System Services Control Point
TC	Transmission Control
TH	Transmission Header
TS	Transmission Subsystem
UNBIND	UNBIND SESSION
UPM	Undefined Protocol Machine

4. DESIGN OVERVIEW

This section outlines the functional components of an SNA/SDLC physical unit type 2. It describes the intermediate level of a top-down system design, whose top-layer design was presented previously (ref 5). Since the design adheres to the SNA layered architecture it should be relatively easy to modify one or more layers in order to extend SNA/SDLC capabilities and support other SNA products.

4.1. Layers

Layering is essential to the structure of SNA. Most layers, although differing in specifics, have the generic internal structure shown in Figure 1. Each layer handles two principal flows: the inbound flow toward the center of the network from the outer layers, and the outbound flow toward the outer layers from the center of the network. A layer consists of two complementary elements: Element.SEND and Element.RCV. Element.SEND handles the flow toward the center and Element.RCV handles the flow toward the outer layers. The elements within a layer are coupled to allow for synchronization. The term "half-session" designates the sum of these elements within a node for a specific network addressable unit (i.e., SSCP, PU, and LU).

In the network at large, elements are paired with their complementary element in the same layer of the remote node. These paired elements interact with one another by exchanging control indicators carried by SNA requests and responses. Details of these protocol machines are given in the next section.

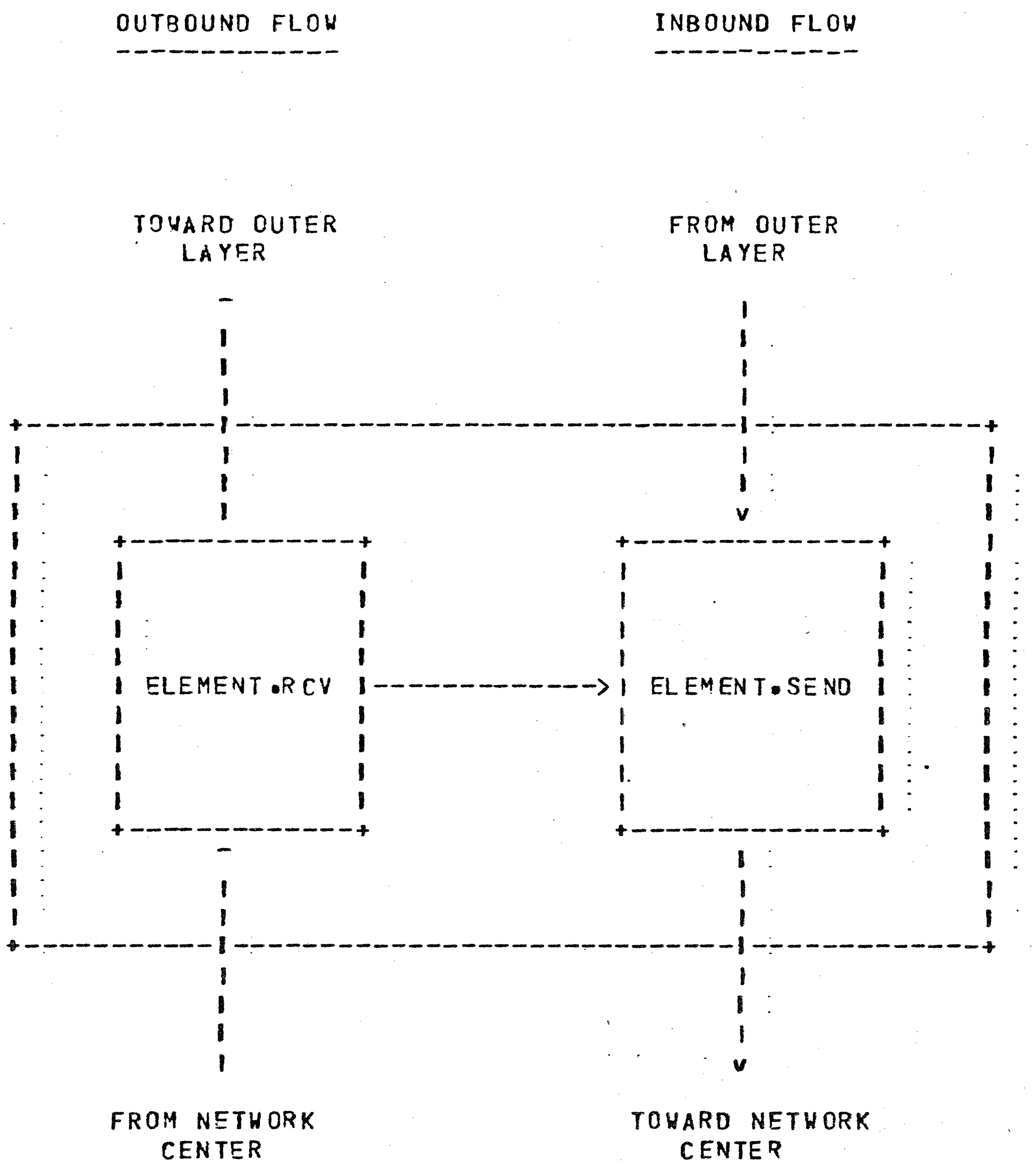

Figure 1. Generic Layer Structure

4.2. Node Structure

The basic node structure (as proposed in Ref 5) is shown in figure 2. It indicates the Function Management, Data Flow Control, Transmission Control, Path Control, and Data Link Control layers.

A more detailed node structure is shown in figure 3. The elements handling the inbound and outbound flow are depicted, along with intertask communications (queues) and control paths. A procedural description is provided for each element in chapter 5 of this document. Note that the FM layer is left undifferentiated, as it will be dependent on the particular application or device being emulated.

4.3. Task Scheduling

The scheduling of tasks within the node is a function of the arrival of information units from outside the node (received from the data link or generated by an end user), the conditioning of task readiness, and the priority of ready tasks.

Tasks within the node synchronize their activity by a queue mechanism implemented via the $RQTSK system service macro call. A running task which has an information unit to pass to an adjacent element will issue a $RQTSK to the requested LRN and starting address. The requested task will run when it becomes the highest priority ready task.

No explicit mechanism at this level of design governs how long a task will run before pending. Likewise, the length of any level or task queue can grow without limit. However, there exist SNA controls or protocols, such as Immediate Request Mode and Pacing, that are used to assure that all tasks pend and that no queue grows excessively large.

Executive functions are not explicitly shown in this design specification. The task scheduler, memory management, queue management, and semaphore facility are intrinsic functions provided by systems software.

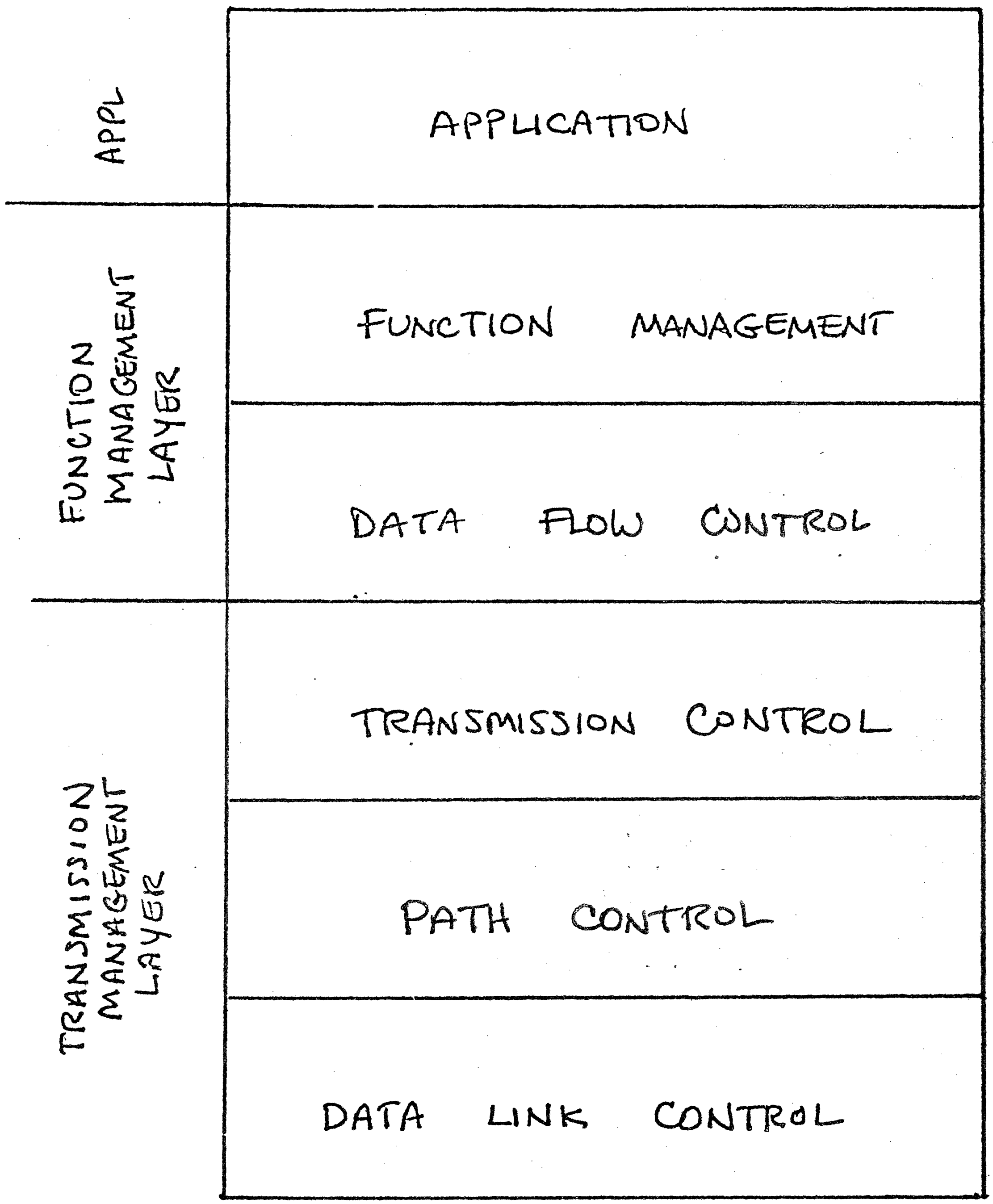

Figure 2. Intermediate View

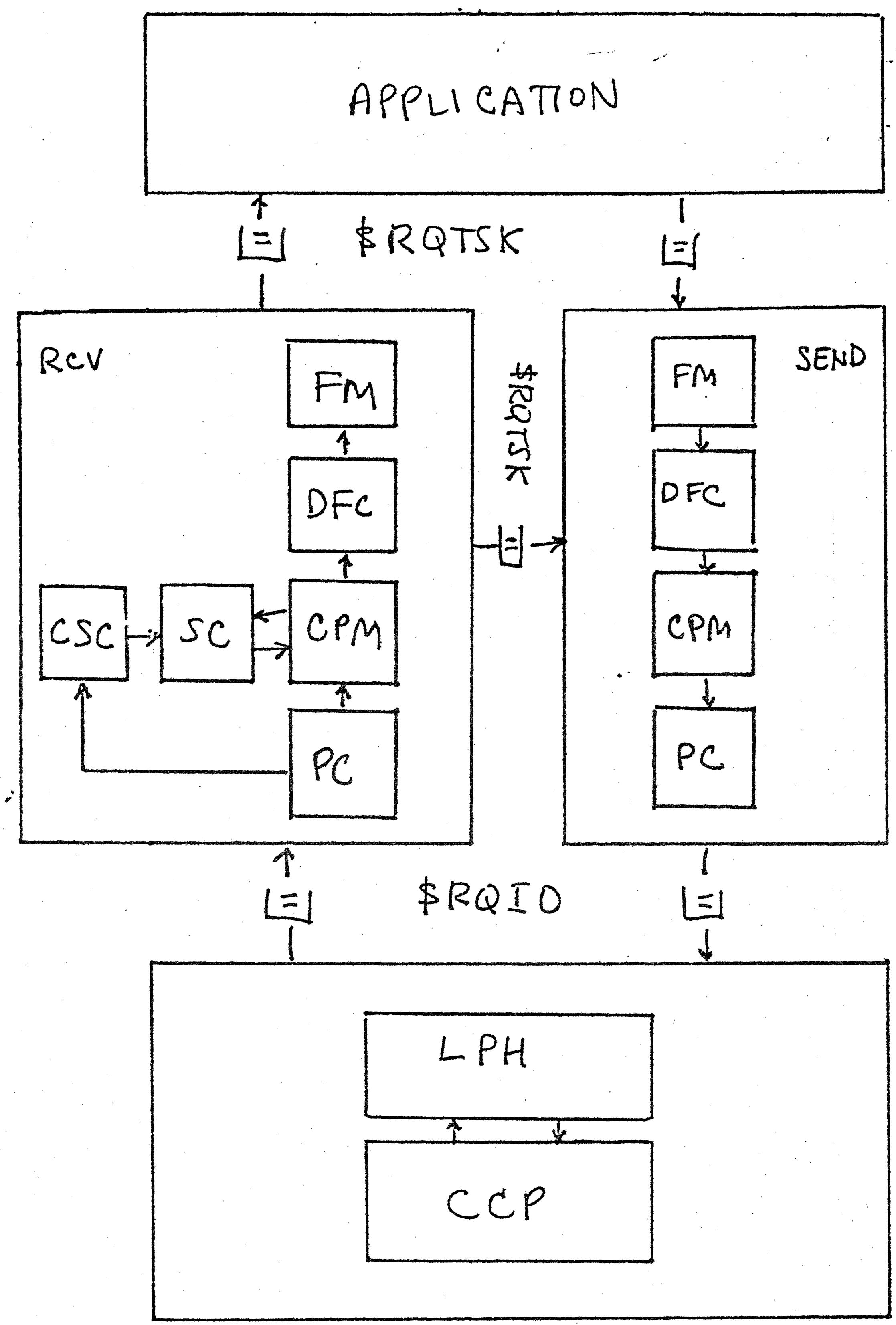

Figure 3. Detailed Node Structure

5. INTERNAL DESIGN

5.1. Data Link Control

5.1.1. Function Description

DLC manages the transfer of information over the data communications channel. DLC does not examine, change, or use the PIU portion of the BLU.

5.1.2. Algorithm

The DLC layer is implemented via the Incoterm Synchronous Data Link Control line protocol handler (See Reference 6).

5.2. Path Control

5.2.1. Function Description

PC manages the routing of PIUs in the outbound flow to the appropriate element in the TC layer - either CSC or CPMGR.RCV. It also performs usage checks on the TH portion of the outbound PIUs and generates the TH field of inbound PIUs.

5.2.2. Algorithm

5.2.2.1. PC Receive

```
PC.RCV: PROCEDURE;
  CALL $RQIO (IORB);                /*EXTERNAL*/
  IF TH_RCV = OK THEN               /*5.2.2.3*/
    DO;
    . IF PU RESET THEN
    .   DO;
    .   . IF  OAF' -= SSCP |         /*OAF'=0=>SSCP*/
    .   .     DAF' -= PU  THEN       /*DAF'=0=>PU  */
    .   .   IF -BBIU IRQNIRSP THEN
    .   .     DISCARD PIU;
    .   .   ELSE
    .   .     DO;
    .   .     . CHANGE PIU TO -RSP (8008);
    .   .     . SEND PIU TO PC.SEND;  /*5.2.2.2*/
    .   .     END;
    .   . ELSE
    .   .   IF LDI = LD THEN
    .   .     IF -BBIUIRQNIRSP THEN
    .   .       DISCARD PIU;
    .   .     ELSE
    .   .       DO;
    .   .       . CHANGE PIU TO -RSP(SNC);
    .   .       . SEND PIU TO PC.SEND;    /*5.2.2.2/
    .   .       END;
    .   .   ELSE
    .   .     SEND PIU TO CSC;        /*5.3.2.1*/
    .   END;
    . ELSE
    .   DO;
    .   . IF -VALID_DAF'  THEN          /*5.2.2.4*/
    .   .   IF -BBIUIRQNIRSP  THEN
    .   .     DISCARD PIU;
    .   .   ELSE
    .   .     DO;
    .   .     . CHANGE PIU TO -RSP (8004);
    .   .     . SEND PIU TO PC.SEND; /*5.2.2.2*/
    .   .     END;
    .   . ELSE
    .   .   DO;
    .   .   . IF (LDI=LD) THEN
    .   .   .   IF -BBIUIRQNIRSP THEN
    .   .   .     DISCARD PIU;
    .   .   .   ELSE
    .   .   .     DO;
    .   .   .      CHANGE PIU TO -RSP(SNC);
    .   .   .      SEND PIU TO PC.SEND;    /*5.2.2.2*/
    .   .   .     END;
    .   .   . ELSE
    .   .   .   DO;
    .   .   .   . FIND HSCB;
    .   .   .   . IF (LU = RESET) THEN
    .   .   .   .    SEND PIU TO CSC;         /*5.3.2.1*/
```

```
  .   .   .   . ELSE
  .   .   .   .    SEND PIU TO CPMGR.RCV; /*5.3.2.2*/
  .   .   .   END;
  .   .   END;
  .   END;
  END;
ELSE
  DISCARD PIU;
RETURN;
END PC.RCV;
```

5.2.2.2. PC Send

```
PC.SEND: PROCEDURE;
  TH(OAF')  = HSCB(OAF');
  TH(DAF')  = HSCB(DAF');
  TH(SNF)   = HSCB(SNF);
  TH(FID)   = GEN_PARAMS (FID);
  TH(EFI)   = HSCB(EFI);
  TH(MPF)   = BBIU + EBIU;
  CALL $RQIO (IORB);       /*EXTERNAL*/
  RETURN;
END PC.SEND;
```

5.2.2.3. TH Format Check

```
TH_RCV: PROCEDURE (RETURN_CODE);
  SELECT;
    WHEN (PIU_LENGTH<1):
      DO;
        RETURN_CODE=NG;
        SENSE_CODE=X'800B';
      END;
    WHEN (FID -= 2):
      DO;
        RETURN_CODE=NG;
        SENSE_CODE=X'8006';
      END;
    WHEN (BBIU & PIU_LENGTH < 9):
      DO;
        RETURN_CODE=NG;
        SENSE_CODE=X'4005';
      END;
    OTHERWISE:
      DO;
        RETURN_CODE=OK;
        SENSE_CODE=X'0000';
      END;
  END;
  IF RETURN_CODE -= OK THEN
    OPTIONALLY_LOG(SENSE_CODE);
  RETURN (RETURN_CODE);
END TH_RCV;
```

5.2.2.4. DAF' Validity Check

```
VALID_DAF': PROCEDURE;
  IF (DAF' >= 0 &
      DAF' <= 34  &
      DAF' -= 1) THEN
    VALID_DAF' = TRUE;
  ELSE
    VALID_DAF' = FALSE;
  RETURN;
END VALID_DAF';
```

5.3. Transmission Control

5.3.1. Function Description

TC is comprised of four elements: Common Session Control (CSC), Connection Point Manager send (CPMGR.SEND), Connection Point Manager receive (CPMGR.RCV), and Session Control (SC).

Common Session Control handles the messages destined for reset (inactive) half-sessions. For reset half-sessions it discards or negatively responds to all BIUs other than the session activation requests - ACTPU, ACTLU, and BIND. When one of these is received, CSC acquires data structures needed to manage a half-session and forwards the request to Session Control.

CPMGR.SEND handles immediate response mode and pacing control on the inbound flow.

CPMGR.RCV routes outbound SNA request and responses to the appropriate element for function interpretation. SNA requests of RU category SC destined for active half sessions are sent to SC. SNA requests and responses of RU category DFC and FMD are sent to DFC.RCV.

SC is a collection of procedures which processes outbound SNA requests of RU category SC, and issues appropriate response on the inbound flow. SC protocols also control the normal data traffic in both directions.

5.3.2. Algorithm

5.3.2.1. CSC Receive

```
CSC: PROCEDURE;
  SELECT (SESSION_TYPE);
  . WHEN (LU_LU):
  .   DO;
  .   . IF SSCP_LU.SESS  -= ACTIVE THEN
  .   .   DO;
  .   .   . CHANGE PIU TO -RSP (8009);
  .   .   . SEND PIU TO PC.SEND;                    /*5.2.2.2*/
  .   .   END;
  .   . ELSE
  .   .   IF RQ_CODE -= BIND THEN
  .   .     DO;
  .   .     . CHANGE PIU TO -RSP (8005);
  .   .     . SEND PIU TO PC.SEND;                  /*5.2.2.2*/
  .   .     END;
  .   .   ELSE
  .   .     SEND PIU TO SC;                         /*5.3.2.4*/
  .   END;
  . WHEN (SSCP_LU)
  .   DO;
  .   . IF SSCP_PU.SESS -= ACTIVE THEN
  .   .   DO;
  .   .   .  CHANGE PIU TO -RSP(8005);
  .   .   .  SEND PIU TO PC.SEND;                   /*5.2.2.2*/
  .   .   END;
  .   ELSE
  .   . IF RQ_CODE -= ACTLU THEN
  .   .   DO;
  .   .   . CHANGE PIU TO -RSP(8005);
  .   .   . SEND PIU TO PC.SEND;                    /*5.2.2.2*/
  .   .   END;
  .   . ELSE
  .   .   SEND PIU TO SC;                           /*5.3.2.4*/
  .   END;
  . WHEN (SSCP_PU)
  .   DO;
  .   . IF RQ_CODE -= ACTPU THEN
  .   .   DO;
  .   .   . CHANGE PIU TO -RSP (8005);
  .   .   . SEND PIU TO PC.SEND;                    /*5.2.2.2*/
  .   .   END;
  .   . ELSE
  .   .   SEND PIU TO SC;                           /*5.3.2.4*/
  .   END;
  . OTHERWISE
  .   DO;
  .   . CHANGE PIU TO -RSP (800F);
  .   . SEND PIU TO PC.SEND;                        /*5.2.2.2*/
  .   END;
  END;
  RETURN;
END CSC;
```

5.3.2.2. CPMGR Receive (RQN | RSP !)

```
CPMGR.RCV: PROCEDURE;
  IF HALF_SESSION -= ACTIVE THEN
    IF RQNIRSP THEN
      DISCARD PIU;
    ELSE
      DO;
      . CHANGE RQ TO -RSP (8005);
      . SEND PIU TO CPMGR.SEND;                    /*5.3.2.3*/
      END;
  ELSE
    DO;
  . IF HSAP.SND_PACING = YES THEN
  .   CALL PAC.RSP_RCV;                            /*TBS*/
  . ELSE
  .   IF (RRI  = RSP &                             /*CHK FOR IPR*/
  .       PI   = PAC &
  .       DR1I = -DR1 &
  .       DR2I = -DR2) THEN
  .     DISCARD PIU;
  .   ELSE
  .     DO;
  .     . IF (EFI = EXP &
  .     .       RRI = RSP) THEN
  .     .       CALL CNTL_IMMED_EXP ("EXPEDITED_RSP_RCV'ED");
  .     .                                          /*TBS*/
  .     . SELECT (RU_CTGY);
  .     .   WHEN (NC):
  .     .     DO;
  .     .     . CHANGE PIU TO -RSP(SNC);
  .     .     . SEND PIU TO CPMGR.SEND;     /*5.3.2.3*/
  .     .     END;
  .     . WHEN (SC):
  .     .   SEND PIU TO SC;                  /*5.3.2.4*/
  .     . WHEN (DFC|FMD):
  .     .   SELECT;
  .     .     WHEN (DT_RCV -= ACTIVE):
  .     .       IF RQN|RSP THEN
  .     .         DISCARD PIU;
  .     .       ELSE
  .     .         DO;
  .     .         . CHANGE PIU TO -RSP(2005);
  .     .         . SEND PIU TO CPMGR.SEND;     /*5.3.2.3*/
  .     .         END;
  .     .     WHEN (EFI = EXP):
  .     .       SEND PIU TO DFC.RCV;            /*5.4.2.1*/
  .     .     OTHERWISE:
  .     .       DO;
  .     .       . IF HSAP.MAX_RCV_RU_SIZE -= NOT_SPEC &
  .     .       .    RU_LENGTH > HSAP_RCV_RU_SIZE THEN
  .     .       .   CHANGE PIU TO EXR (1002);
  .     .       . IF RRI = RSP THEN
  .     .       .   SEND RSP TO DFC.RCV;            /*5.4.2.1*/
  .     .       . ELSE
  .     .       .   DO;
  .     .       .   . CALL SQN_RCV;                 /*TBS*/
  .     .       .   . CALL PAC_RQ_RCV;              /*TBS*/
  .     .       .   . SEND RQ TO DFC.RCV;           /*5.4.2.1*/
```

```
     .      .        .   END;
     .      .        END
     .      END;
     END;
   RETURN;
 END CPMGR.RCV;
```

5.3.2.3. CPMGR Send

```
 CPMGR.SEND: PROCEDURE;
   IF RRI = RQ THEN
     DO;
     . IF EFI = EXP THEN                              /*SHUTC & SIG*/
     .    DO;
     .    . CALL CTRL_IMMED_EXP;
     .    . IF RU_CTGY ¬= DFCIFMD THEN
     .    .    CALL ID_EXP_SND;
     .    END;
     . ELSE
     .    CALL PAC_RQ_SND;
     . SEND PIU TO PC.SEND;                            /*5.2.2.2*/
     END;
   ELSE
     DO;
     . IF EFI ¬= EXP THEN
     .    IF PAC_RQ_RCV = PEND &
     .       PAC_SND_CT = 0 THEN
     .       DO;
     .       . PI = PAC;
     .       . CALL PAC_RQ_RCV ('PAC RSP SENT')
     .       END;
     . SEND PIU TO PC.SEND;                           /*5.2.2.2*/
     END;
   RETURN;
 END CMPGR.SEND;
```

5.3.2.4. Session Control

```
SC: PROCEDURE;
  DO CASE RQ_CODE;
    ACTPU:
       DO;
         HSCB(PU.ACT) = 1;
         SEND +RSP (ACTPU.COLD) TO CPMGR.SEND; /*5.3.2.3*/
       END;
    DACTPU:
       DO;
         HSCB(PU.ACT) = 0;
         SEND +RSP (DACTPU) TO CPMGR.SEND;     /*5.3.2.3*/
       END;
    ACTLU:
       DO;
         RESET DT;
         HSCB (LU.ACT) = ACTIVE;
         SEND +RSP (ACTLU.COLD) TO CPMGR.SEND; /*5.3.2.3*/
       END;
    DACTLU:
       DO;
         RESET DT;
         ASCB (LU.ACT) = RESET;
         SEND +RSP (DACTLU) TO CPMGR.SEND;     /*5.3.2.3*/
       END;
    BIND:
         CALL BIND_RCV;                        /*5.3.2.6*/
    UNBIND:
       DO;
         RESET DT;
         HSCB(SESS) = RESET;
         SEND +RSP(UNBIND) TO CPMGR.SEND;      /*5.3.2.3*/
       END;
    CLEAR:
       DO;
         RESET DT;
         SEND +RSP (CLEAR) TO CPMGR.SEND;      /*5.3.2.3*/
       END;
    SDT:
       DO;
         SET DT;
         SEND +RSP (SDT) TO CPMGR.SEND;        /*5.3.2.3*/
       END;
  END;
  RETURN;
END SC;
```

5.3.2.5. SQN Validity Check

```
SQN_RCV: PROCEDURE;
  SQN_RCV_CNT = SQN_RCV_CNT + 1;
  IF SNO ¬= SQN_RCV_CNT THEN
    DO;
      SQN_RCV_CNT = SQN_RCV_CNT - 1;
      CHANGE RQ TO EXR (2001);
    END;
  RETURN;
END SQN_RCV;
```

5.3.2.6. BIND Receive

```
BIND_RCV: PROCEDURE;
  IF LU_LU.SESS = ACTIVE THEN
    DO;
    . IF OAF' = PLU THEN
    .   SNC = X'0815';
    . ELSE
    .   SNC = X'0805';
    . CHANGE PIU TO -RSP(SNC);
    . SEND PIU TO PC.SEND;                          /*5.2.2.2*/
    END;
  ELSE;
    DO;
    . CALL BIND_PARAM_USAGE_CHKS (SNC);     /*TBS*/
    . IF SNC ¬= X'0000' THEN
    .   DO;
    .   . CHANGE PIU TO -RSP (SNC);
    .   . SEND PIU TO PC.SEND;                      /*5.2.2.2*/
    .   END;
    . ELSE
    .   DO;
    .   . RESET DT SUBTREE;
    .   . PLU = OAF';
    .   . LU_LU.SESS = ACTIVE;
    .   . SEND +RSP (BIND_RU) TO PC.SEND;           /*5.2.2.2*/
    .   END;
    END;
  RETURN;
END BIND_RCV;
```

5.4. DATA FLOW CONTROL

5.4.1. Function Description

The function of DFC is to control the flow of FM (User) data requests and responses between the session's FM pairs. Only FM data and DFC requests are processed by DFC.

The following list names and describes the functions of the major DFC components in the order in which they are generally used for a session:

- SESSAD.DFC - RESET - called during session activation processing and when data traffic is reset in order to reset all DFC states and correlation tables (used to correlate requests and responses).

- DFC.RCV - the entry to which all FM data and DFC requests and responses are sent for processing by CPMGR.RCV.

- DFC.SEND - the entry form which all FM data and DFC requests and responses are sent for processing and sending to CPMGR.SEND.

5.4.1.1. Boundary Layers

DFC interacts with the following boundary layers:

- CPMGR (Rcv/Send requests and responses)

- APPL_FM (Application data)

- APPL_DFC (Application DFC)

5.4.1.2. Functions

Enforcement of proper data flow procedure is accomplished by the following DFC controls:

- Correlation of requests and responses.

- Control of normal flow half duplex flip flop send/receive mode.

- Control of immediate request and response control modes.

- Control of chaining.

- Control of brackets (transaction).

- Shutdown processing for termination of normal flow traffic prior to session deactivation.

- Enforcement of correct request/response RH formats.

5.4.2. Algorithm

5.4.2.1. DFC Receive

```
DFC.RCV: PROCEDURE;
  IF RRI=RQ THEN
    CALL RCV_RQ;                                  /*5.4.2.2*/
  ELSE
    CALL RCV_RSP;                                 /*5.4.2.3*/
  RETURN;
END DFC.RCV;
```

5.4.2.2. Receive Request

```
RCV_RQ: PROCEDURE;
  IF EFI = NORM THEN
    DO;
    . CT_PTR = RCV_RQ_NORM_CT;
    . CALL CT_ENTRY_ADD_OR_UPDATE; /*SAVE CORRELATION INFO
    .                                             5.4.2.9*/
    . IF RCV_CHAIN_STATE = PURGE THEN
    .   CALL RCV_RQ_NORM_PURGE; /*PURGE CHAIN 5.4.2.12*/
    . ELSE
    .   DO;
    .   . IF SDI = -SD THEN
    .   .   CALL RCV_RQ_NORM_ERROR_CHECKS; /*VALIDATE RQ
    .   .                                        5.4.2.13*/
    .   . CALL FSM_CHAIN_RCV;     /*UPDATE CONTROL STATES*/
    .   . CALL FSM_HDX_FF_FSP;
    .   . CALL FSM_BSM;
    .   . IF RU_CTGY=DCF & RQ_CODE -= CANCEL THEN
    .   .   SEND RQ TO USER_DCF;  /*SEND RQ TO NXT LAYER*/
    .   . ELSE
    .   .   SEND RQ TO USER_FMD;
    .   END;
    .
    END;
  ELSE                                            /*EXPEDITED RQ*/
    DO;
    . IF SDI -= SD & RQ_CODE = SHUTDISIG THEN
    .   CALL FSM_SHUTD;
    . SEND RQ TO USER_DFC;
    END;
  RETURN;
END RCV_RQ;
```

5.4.2.3. Receive Response

```
RCV_RSP: PROCEDURE:
  IF EFI = NORM THEN
    DO;
    • CT_PTR = SEND_RQ_NORM_CT;
    • KEY = SNF;
    • IF CT_KEY_SEARCH = FOUND &  /*5.4.2.10*/
    •    CT_RSP_RCVD_OR_SENT = NO THEN
    •   DO;
    •   • CALL RCV_RSP_NORM_CNTL_MGR; /*UPDATE CNTL STATES*
    •   •                                  5.4.2.14*/
    •   • CALL FSM_HDX_CONTROL_RCV_RSP;
    •   • CALL FSM_BSM;
    •   • CALL FSM_RTR;
    •   • IF RU_CTGY = DFC & RQ_CODE - = CANCEL THEN
    •   •   SEND RSP TO USER_DFC;   /*SND RSP TO NXT LAYER*/
    •   • ELSE
    •   •   SEND RSP TO USER_FMD;
    •   • IF CT_ECI = EC THEN
    •   •   REMOVE CURRENT FROM CT_PTR;   /*REMOVE RQ INFO*/
    •   • ELSE
    •   •   CT_RSP_RCVD_OR_SENT = YES;   /*RSP WHILE INC*/
    •   END;
    • ELSE
    •   DISCARD RSP;                    /*INVALID RSP*/
    END;
  ELSE                                  /*EXPEDITED RSP*/
    DO;
    • CT_PTR = SEND_RQ_EXP_CT;
    • IF SEND_RQ_EXP_CT_ID = SNF THEN
    •   DO;
    •   • CALL FSM_HDX_FF_FSR;
    •   • REMOVE CURRENT FROM CT_PTR;  /*REMOVE RQ INFO*/
    •   END;
    • ELSE
    •   DISCARD RSP;                      /*INVALID RSP*/
    END;
  RETURN;
END RCV_RSP;
```

5.4.2.4. DFC Send

```
DFC.SEND: PROCEDURE;
  IF CPMGR_SEND_CHECKS = OK THEN      /*5.4.2.5*/
    DO;
    • IF RRI = RQ THEN
    •   CALL SEND_RQ;                 /*5.4.2.6*/
    • ELSE
    •   CALL SEND_RSP;                /*5.4.2.7*/
    END;
  ELSE                                /*SESSION INACTIVE*/
    SEND 'REJECT' TO SENDING_PROCEDURE;
  RETURN;
END DFC.SEND;
```

5.4.2.5. Send State Checks

```
CPMGR_SEND_CHECKS: PROCEDURE;

  IF FSM_SESS_RCV ¬= ACTIVE |
     FSM_DT_RCV ¬= ACTIVE THEN
    RC = NG;                                    /*SESS NOT ACTIVE*/
  ELSE
    RC = OK;
  RETURN;
END CPMGR_SEND_CHECKS;
```

5.4.2.6. Send Request

```
SEND_RQ: PROCEDURE;

  IF RU_CTGY = DFC THEN
    CALL SEND_RQ_FORMAT_DFC;     /*FORMAT DFC RQ'S 5.4.2.15*/
  IF SEND_RQ_STATE_ERRORS = OK THEN   /*5.4.2.16*/
    DO;
    . IF EFI = NORMAL THEN
    .   SQN_SEND_CNT = SQN_SEND_CNT + 1;  /*INCR SEQ #*/
    .   SNF = SQN_SEND_CNT;
    .   CT_PTR = SEND_RQ_NORM_CT;       /*SAVE RQ INFO TO*/
    .   CALL CT_ENTRY_ADD_OR_UPDATE; /*CORRELATE RSP 5.4.2.9/
    .   IF (BCI=BC|CT_RSP_RCVD_OR_SENT=NO) THEN
    .     CALL FSM_CNTL_NORM;       /*ENFORCE IMMED. RQ MODE*/
    .   CALL_FSM_CHAIN_SEND;        /*UPDATE CNTL STATES*/
    .   CALL_FSM_HDX_FF_FSP;
    .   CALL_FSM_HDX_CONTROL_RCV_RSP;
    .   CALL_FSM_BSM;
    .   CALL_FSM_RTR;
    .   SEND RQ TO CPMGR.SEND;           /*5.3.2.3*/
    . ELSE                               /*EXPEDITED RQ*/
    .   DO;
    .   . SND_EXP_ID=SND_EXP_ID+1;
    .   . SNF=SND_EXP_ID;
    .   . CALL FSM_HDX_FF_FSP;
    .   . CALL SEND_RQ_EXP_CT_ELEM_ADD;  /*5.4.2.11*/
    .   . SEND RQ TO CPMGR.SEND;
    .   END;
    END;
  ELSE;                     /*STATE VIOLATION-REJECT*/
    SEND 'REJECT' TO SENDING PROCEDURE;
  RETURN;
END SEND_RQ;
```

5.4.2.7. Send Response

```
SEND_RSP: PROCEDURE;
  IF EFI=NORMAL THEN
    DO;
    . CT_PTR=RCV_RQ_NORM_CT;
    . CALL SEND_RSP_NORM_PROCESS;  /*GENERATE NORM RSP
    .                                    5.4.2.17*/
    END;
  ELSE
    DO;
    . CALL SEND_RSP_EXP_GENERATE;   /*GENERATE EXP RSP
    .                                     5.4.2.19*/
    . CALL FSM_SHUTD;
    . SEND RSP TO CPMGR.SEND;       /*5.3.2.3*/
    END;
  RETURN;
END SEND_RSP;
```

5.4.2.8. CT Entry Add

```
CT_ENTRY_ADD: PROCEDURE;
  CT_BEG_SNF=SNF;                    /*SAVE RQ INFO*/
  CT_END_SNF=SNF;
  CT_RU_CTGY=RU_CTGY;
  CT_FI=FI;
  CT_DR1I=DR1I;
  CT_DR2I=DR2I;
  CT_ERI=ERI;
  CT_ECI=ECI;
  CT_ORI=ORI;
  CT_BBI=BBI;
  CT_EBI=EBI;
  CT_RQCODE=RQ_CODE;
  RETURN;
END CT_ENTRY_ADD;
```

5.4.2.9. CT Entry Add or Update

```
CT_ENTRY_ADD_OR_UPDATE:  PROCEDURE;
   IF RQ_CODE=CANCEL THEN
     DO;
     • CALL CT_ENTRY_ADD;                             /*5.4.2.8*/
     • CT_CDI=CDI;                                    /*CANCELl IS END*/
     END;
   ELSE
     DO;
     • IF EMPTY (CT_PTR) THEN
     •   DO;                                          /*START NEW CHAIN*/
     •   • CALL CT_ENTRY_ADD;                         /*5.4.2.8*/
     •   • IF ECI=EC THEN
     •   •   CT_CDI=CDI;
     •   END;
     • ELSE
     •   DO;                                              /*IN CHAIN*/
     •   • CT_END_SNF=SNF;
     •   • IF ECI=EC THEN
     •   •   DO;
     •   •   • CT_ECI=EC;
     •   •   • CT_DR1I=DR1I;
     •   •   • CT_DR2I=DR2I;
     •   •   • CT_ERI=ERI;
     •   •   • CT_CDI=CDI;
     •   •   END;
     •   END;
     END;
   RETURN;
END CT_ENTRY_ADD_OR_UPDATE;
```

5.4.2.10. CT Key Search

```
CT_KEY_SEARCH:  PROCEDURE (RC);
   RC=NOTFOUND;
   IF CT_END_SNF_CT_BEG_SNF>=0 THEN
     DO;
     • IF KEY>=CT_BEG_SNF &
     •      KEY<=CT_END_SNF  THEN
     • RC=FOUND;
     END;
   ELSE
     DO;
     • IF KEY<=CT_END_SNF|
     •      KEY>=CT_BEG_SNF THEN
     • RC=FOUND;
     END;
   RETURN (RC);
END CT_KEY_SEARCH;
```

5.4.2.11. CT Update

```
SEND_RQ_EXP_CT_ELEM_ADD:  PROCEDURE;

/*SAVE EXP RQ INFO FOR RSP CORRELATION*/

   SEND_RQ_EXP_CT_ID=SNF;
   SEND_RQ_EXP_CT_RQCODE=RQ_CODE;
   RETURN;
END SEND_RQ_EXP_CT_ELEM_ADD;
```

5.4.2.12. Receive State Checks

```
RCV_RQ_NORM_PURGE:  PROCEDURE;

   CALL FSM_CHAIN_RCV;
   CALL FSM_HDX_FF_FSP;
   CALL FSM_HDX_CONTROL_SEND_RSP;
   CALL FSM_BSM('PURGE');
   IF RQ_CODE=CANCEL THEN
     SEND RQ TO USER_FMD;
   ELSE
     DISCARD RQ;
   RETURN;
END RCV_RQ_NORM_PURGE;
```

5.4.2.13. Receive Request State Checks

```
RCV_RQ_NORM_ERROR_CHECKS:  PROCEDURE;
   IF FSM_RES=ERROR|                       /*EXR CREATED ON ERROR*/
     FSM_HDX_FF_FSP=ERROR|
     FSM_CHAIN_RCV=ERROR|
     FSM_BSM=ERROR|
     FSM_RTR=ERROR THEN
   ;
   RETURN;
END RCV_RQ_NORM_ERROR_CHECKS;
```

5.4.2.14. Receive Response State Checks

```
RCV_RSP_NORM_CNTL_MGR:  PROCEDURE;

/*SEND CHAIN RSP WHEN REQUIRED TO IMMEDIATE CNTL STATE*/

   IF FSM_CNTL_NORM=PEND &
   ((CT_BEG_SNF<=SNF & SNF<=CT_END_SNF)|
   (SNF<=CT_END_SNF & CT_END_SNF<CT.BEG_SND)|
   (CT_END_SNF<CT_BEG_SNF & CT_BEG_SNF<=SNF)) THEN
   CALL FSM_CNTL_NORM ('RSP_TO_CURRENT_CHAIN');
   RETURN;
END RCV_RSP_NORM_CNTL_MGR;
```

5.4.2.15. Generate RH

```
SEND_RQ_FORMAT_DFC:  PROCEDURE;

/*FORMAT DFC REQUESTS*/

  RRI=RQ;
  RU_CTGY=DFC;
  FI=FMH;
  BCI=BC;
  ECI=EC;
  DR1I=DR1;
  CSI=CODE0;
  IF EFI=NORMAL THEN
    SELECT (RQ_CODE);
    . WHEN (CANCEL)
    .   DO;
    .   . BBI=-BB;
    .   . DR2I=-DR2;
    .   END;
    . WHEN (LUSTAT)
    .   CALL USER_LUSTAT_FORMAT;
    . WHEN (RTR)
    .   DO;
    .   . DR2I=-DR2;
    .   . ERI=-ER;
    .   . BBI=-BB;
    .   . EBI=-EB;
    .   . CDI=-CD;
    .   END;
    END;
  ELSE                                           /*EXPEDITED*/
    DO;
    . DR2I=-DR2;
    . ERI=-ER;
    . QRI=-QR;
    . BBI=-BB;
    . EBI=-EB;
    . CDI=-CD;
    END;
  RETURN;
END SEND_RQ_FORMAT_DFC;
```

5.4.2.16. Send Request State Checks

```
SEND_RQ_STATE_ERRORS:  PROCEDURE(RC);

  RC=OK;
  IF EFI=NORMAL THEN
    DO;
    • IF FSM_CHAIN_SEND=REJECT|
    •      FSM_CNTL_NORM=REJECT THEN
    •    RC=NG;
    • ELSE;
    •    IF FSM_HDX_FF_FSP=REJECT|
    •         FSM_BSM=REJECT|
    •         FSM_RTR=REJECT THEN
    •       RC=NG;
    END;
  RETURN (RC);
END SEND_RQ_STATE_ERRORS;
```

5.4.2.17. Process Normal Response

```
RSP_NORM_PROCESS:  PROCEDURE;

  IF RQD (CT_RQI)|
     (RQE(CT_RQI) &
     PRE_RSP_TYPE=NEG) THEN
    DO;
    • CALL SEND_RSP_NORM_GENERATE; /*BUILD RSP FIG 4-18*/
    • CALL FSM_HDX_CONTROL_SEND_RSP;
    • CALL FSM_BSM;
    • SEND RSP TO CPMGR.SEND;
    END;
  IF CT_ECI=EC THEN
    REMOVE CURRENT FROM CT_PTR;
  ELSE
    CT_RSP_RCVD_OR_SENT=YES;   /*RSP PRIOR TO END CHAIN*/
  RETURN;
END_RSP_NORM_PROCESS;
```

5.4.2.18. Generate Normal Response

```
SEND_RSP_NORM_GENERATE:  PROCEDURE;

   EFI=NORMAL;
   SNF=PRE_RSP_SNF;
   PRI=RSP;
   RU_CTGY=CT_RU_CTGY;
   FI=CT_FI;
   BCI=BC;
   ECI=EC;
   DR1I=CT_DR1I;
   DR2I=CT_DR2I;
   QRI=CT_QRI;
   RQ_CODE=CT_RQCODE;
   RETURN;
END SEND_RSP_NORM_GENERATE;
```