

1  
005.71  
M765  
C.2



# **ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

## **Facultad de Ingeniería en Electricidad y Computación**

**"DESARROLLO DE UN CONTROLADOR DE CAJEROS  
AUTOMATICOS (SISTEMA SERVATM)"**

### **INFORME TECNICO**

Previo a la Obtención del Título de:

**INGENIERO EN ELECTRONICA**

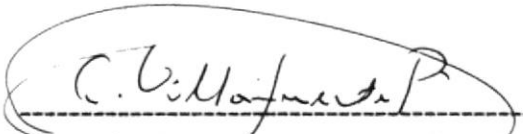
Presentado por:  
**Eugenio Montaño Angulo**


**Guayaquil - Ecuador  
1.995**

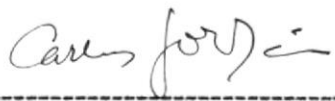
## DEDICATORIA

**Después de algunos años de esfuerzo y trabajo en proyectos de interconectividad de computadores e integración de sistemas, quiero dedicar este tema a mis Padres, Esposa e Hija.**

TRIBUNAL

  
-----  
ING. CARLOS VILLAFUERTE P.  
PRESIDENTE

  
-----  
ING. JAIME PUENTE P.  
PROFESOR SUPERVISOR

  
-----  
ING. CARLOS JORDAN V.  
MIEMBRO PRINCIPAL

## AGRADECIMIENTO

**Agradezco:**

**Al esfuerzo y apoyo económico de mis padres.**

**La apertura y facilidades que siempre me brindo la Escuela Superior Politécnica para culminar mis estudios.**

**La gentileza y cordialidad del Ing. Jaime Puente por la revisión de este proyecto.**

## DECLARACION EXPRESA

“La responsabilidad por los hechos, ideas y doctrinas expuestos en este Informe Técnico, me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral”.

A handwritten signature in black ink, consisting of stylized, overlapping letters that appear to be 'E', 'M', and 'A', with a long horizontal stroke extending to the right.

---

**Eugenio Montaña Angulo**

## RESUMEN

El **SERVATM** es un Software que permite la integración y conectividad de cajeros automáticos a una de las redes privadas más importantes del país, la cual es dirigida por la compañía **BANRED**.

El **SERVATM** fue desarrollado con lenguaje de programación orientada a objetos (C++) , bajo el Sistema Operativo OS/2 .

Las funciones que realiza el sistema **SERVATM** están resumidas en los siguientes tópicos:

- Controlador de ATM (Automated Teller Machine).
- Convertidor de Protocolo.
- Ruteador de Mensajes.
- Autorización en modo Off-Line.
- Manejo de Reentry.

## CONTROLADOR DE ATM

Para conectar y controlar los cajeros automáticos el **SERVATM** soporta los siguientes protocolos de comunicación :

- NCR ISO ASINCRONICO (UTP).
- X.25

Una representación gráfica de dicha conexión es mostrada en la figura No 1.

# Esquema Actual BANRED

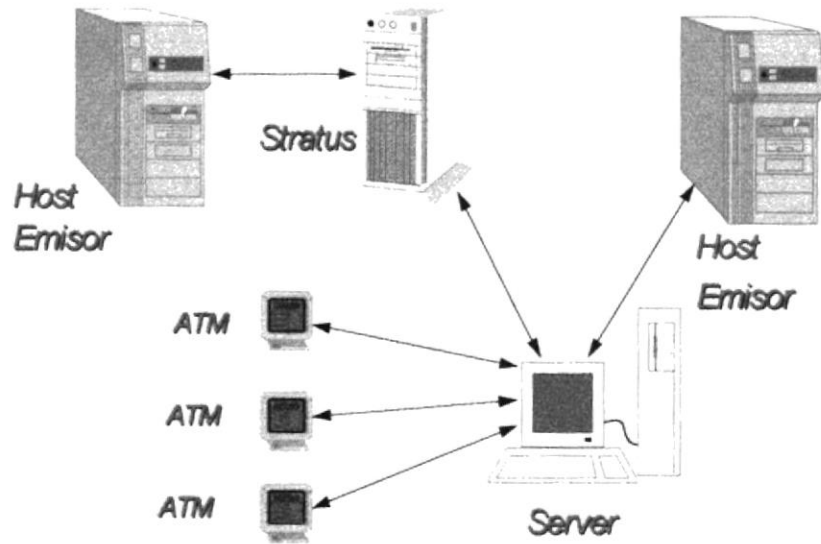


Figura No. 1

## CONVERTIDOR DE PROTOCOLO

El **SERVATM** realiza la conversión de protocolos de comunicación tal como se muestra en la Fig. No. 2.

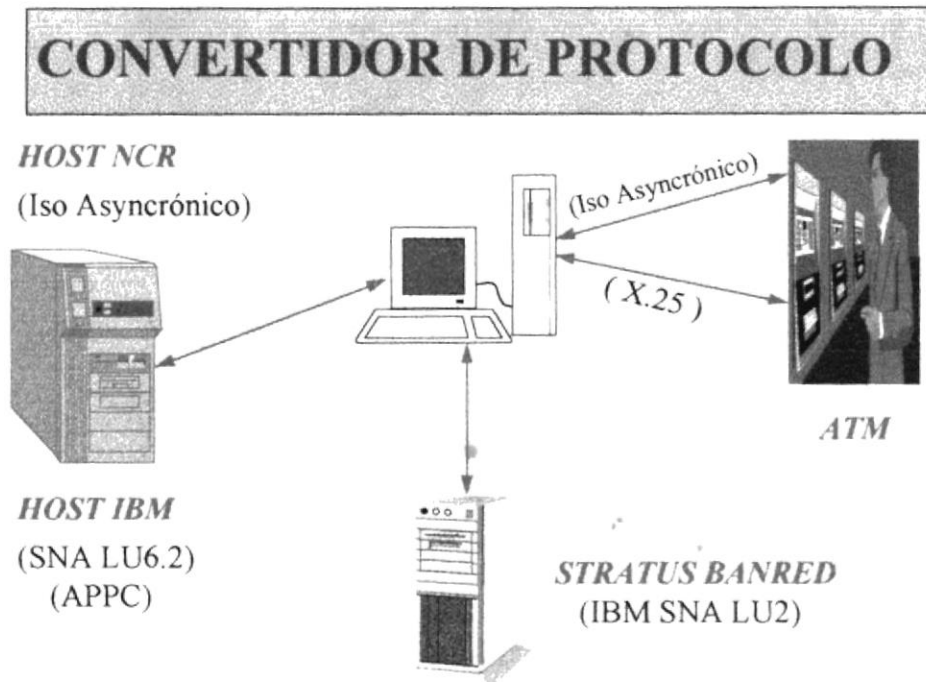


Figura No. 2

## RUTEADOR DE MENSAJE.

Las transacciones originadas en los distintos ATM son ruteadas en una de las siguientes direcciones :

- Hacia el banco emisor de la tarjeta de débito.  
Este caso aplica para aquellas transacciones realizadas en los cajeros del mismo banco.
- Hacia el Host de **BANRED**.  
Las transacciones ejecutadas por los clientes de un determinado banco en otros cajeros automáticos son enviadas a **BANRED**.



## AUTORIZACION EN MODO OFF-LINE

Cuando el Host del banco emisor se encuentra en modo Off-Line el **SERVATM** autoriza aquellas transacciones que cumplan con los límites de cupo establecidos por el banco emisor de la tarjeta de débito.

## MANEJO DE REENTRY

Las transacciones procesadas en modo Off-Line son enviadas al Host emisor de la Tarjeta de Débito en forma automática una vez que se restablezca la comunicación con el mismo.

## INDICE

1.	ANALISIS DEL PROYECTO	12
1.1	DEFINICION Y ALCANCE DEL PROYECTO	12
1.2	OBJETIVOS	14
1.3	REQUERIMIENTOS DE SOFTWARE	18
1.4	REQUERIMIENTOS DE HARDWARE	19
2.	TUTORIAL DE PROTOCOLOS	20
2.1	PROTOCOLO DE COMUNICACION APPC	20
2.1.1	TRANSACTION PROGRAM (TP)	21
2.1.2	LOGICAL UNIT (LU)	21
2.1.3	SNA SESSION	21
2.1.4	CONVERSACIONES	22
2.1.5	PHYSICAL UNIT (PU)	23
2.1.6	SESIONES Y CONVERSACIONES EN PARALELO	23
2.1.7	VERBOS DE CONTROL PARA APPC	24
2.2	PROTOCOLO DE COMUNICACION X.25	27
2.2.1	PHYSICAL LEVEL (NIVEL I)	27
2.2.2	FRAME LEVEL (NIVEL II)	29
2.2.3	PACKET LEVEL (NIVEL III)	29
2.2.4	CIRCUITOS VIRTUALES	30

2.2.5	PASOS PARA ESTABLECER UNA CONEXIÓN EN X.25	31
2.3	NCR ISO ASINCRONICO	33
2.3.1	CARACTERES DE CONTROL	33
2.3.2	DESCRIPCION DE LA SECUENCIA POLL/SELECT	34
3.	ESTRUCTURA DEL SERVATM	36
3.1	LEER CONFIGURACION	36
3.2	CREAR QUEUES	38
3.3	CREAR PROCESOS	39
3.4	CREAR TAREAS PARA LAS COMUNICACIONES	41
4.	USO DE PROTOCOLOS DE COMUNICACION	44
4.1	COMO EL SERVATM MANEJA EL PROTOCOLO APPC	44
4.2	COMO EL SERVATM MANEJA EL PROTOCOLO X.25	47
4.3	COMO EL SERVATM MANEJA EL PROTOCOLO NCR ISO ASINCRONICO	49
5.	CONCLUSIONES Y RECOMENDACIONES	53

# **1 ANALISIS DEL PROYECTO**

## **1.1 Definición y Alcance del Proyecto**

Los principales aspectos considerados en el desarrollo de este sistema se detallan a continuación :

- Utilizar las estructuras actuales de cada institución a fin de que no existan cambios en los procesos actuales contra los nuevos que presenta el sistema.
- Manejo de la mínima cantidad de información monetaria en los procesos con el objetivo principal de evitar riesgos en la confidencialidad de los datos.
- Control de Seguridad en la información a fin de garantizar que las personas autorizadas puedan tener acceso a la información requerida.
- Generación de la aplicación estándar a fin de mantener un equilibrio en los procesos que realizan las diferentes instituciones miembros de la red.
- Apertura total en el diseño hacia nuevos requerimientos o transacciones, tanto de la red cuanto para la institución local.
- Proceso de Autorización alternos para mantener un servicio continuo en la atención a los clientes (procesos off-line).
- Portabilidad de la aplicación a fin de no depender de marcas ni modelos de computadores.
- Utilización óptima de los protocolos de comunicación entre los computadores a fin de mantener un enlace confiable.
- Soporte de varios protocolos de comunicación, sin crear dependencia de uno de ellos, permitiendo cambiar en forma dinámica de un protocolo a otro.
- Transparencias en los procesos a fin de que las personas asignadas puedan comprender y modificar los programas sin

necesidad de la intervención de los programadores originales, esto permite a las instituciones tener un control de los procesos y no genera dependencia de terceros.

- Mantener la operativa actual de los cajeros automáticos, a fin de no ocasionar problemas de adaptación al personal operativo.
- Aceptar todas las tarjetas actuales en el mercado, lo que evita que se deba emitir un nuevo parque de las mismas.
- Ingreso dinámico de nuevas instituciones a la red, sin necesidad de modificar el programa de cajero automático.
- Control remoto de los servidores, con la finalidad de resolver problemas de comunicaciones y actualizaciones de los mismos en forma remota.
- Envío automático al computador central, de las transacciones realizadas "off-line" para su respectivo débito.
- Verificación de transacciones cruzadas contra las autorizadas por el switch de la red.
- Emisión de listados de inconsistencias para determinar el correcto funcionamiento de los procesos de conciliación.
- Manejo de varios protocolos de comunicación : Iso Asincrónico X.25, SNA LU2 y APPC.

## **1.2 OBJETIVOS**

El objetivo básico del **SERVATM**, es controlar el correcto funcionamiento de los componentes de la red de cajeros automáticos, creando una armonía de procesos que interrelacionan a los cajeros automáticos, servidores, autorizadores, computadores centrales y switch de interconexión.

Los procesos básicos de este sistema, sin llegar a limitarse a ellos, son los siguientes :

- Control del flujo de transacciones generadas desde un cajero automático a través de la utilización de una tarjeta de débito o de crédito.
- Envío de transacciones al computador central de la institución local propietaria de la tarjeta y el cajero para la aprobación de las mismas.
- Autorización local de las transacciones en caso de ausencia temporal o permanente de los procesos que llevan a cabo estas funciones en el computador central.
- Determinación, selección y envío de transacciones no propias de la institución hacia el switch concentrador de transacciones para que sean enviadas a la institución propietaria de la tarjeta.
- Recepción y Autorización de transacciones generadas en otras instituciones y enviadas hacia el switch para su autorización.
- Control del estado de cada componente de los cajeros automáticos a fin de verificar el correcto funcionamiento de los mismos.
- Almacenamiento estadístico de los estados presentados por los cajeros para determinar causas comunes de probables fallas y mejorar el servicio de atención al público.
- Generación de procesos en lotes que permitan mantener actualizada la información, que realiza la autorización alterna de las transacciones, para casos de caídas del computador central.

- Control de fechas contables y emisión de reportes para cuadratura de transacciones locales contra valores monetarios.
- Emisión de reportes que permitan realizar la compensaciones monetarias con las entidades pertenecientes a una red.

Las funciones que realiza el sistema **SERVATM** están resumidas en los siguientes tópicos:

- Controlador de ATM (Automated Teller Machine).
- Convertidor de Protocolo.
- Ruteador de Mensajes.
- Autorización en modo Off-Line.
- Manejo de Reentry.

## CONTROLADOR DE ATM

### Esquema Actual BANRED

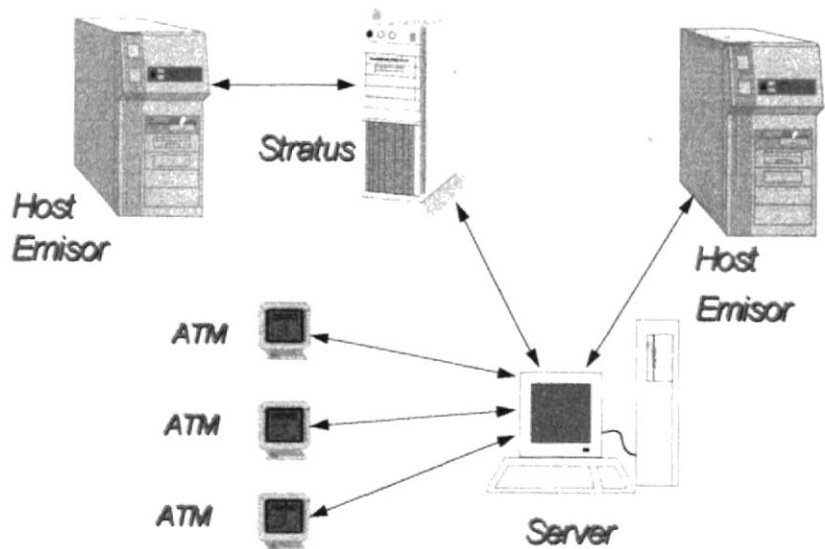


Figura No. 1

Para conectar y controlar los cajeros automáticos el **SERVATM** soporta los siguientes protocolos:

- NCR ISO ASINCRONICO (UTP).
- X.25

Es decir que los cajeros automáticos pueden enviar o recibir mensajes o transacciones en uno de los protocolos anteriores.

## CONVERTIDOR DE PROTOCOLO

El **SERVATM** realiza la conversión de protocolos de comunicación tal como se muestra en la Fig. No. 2.

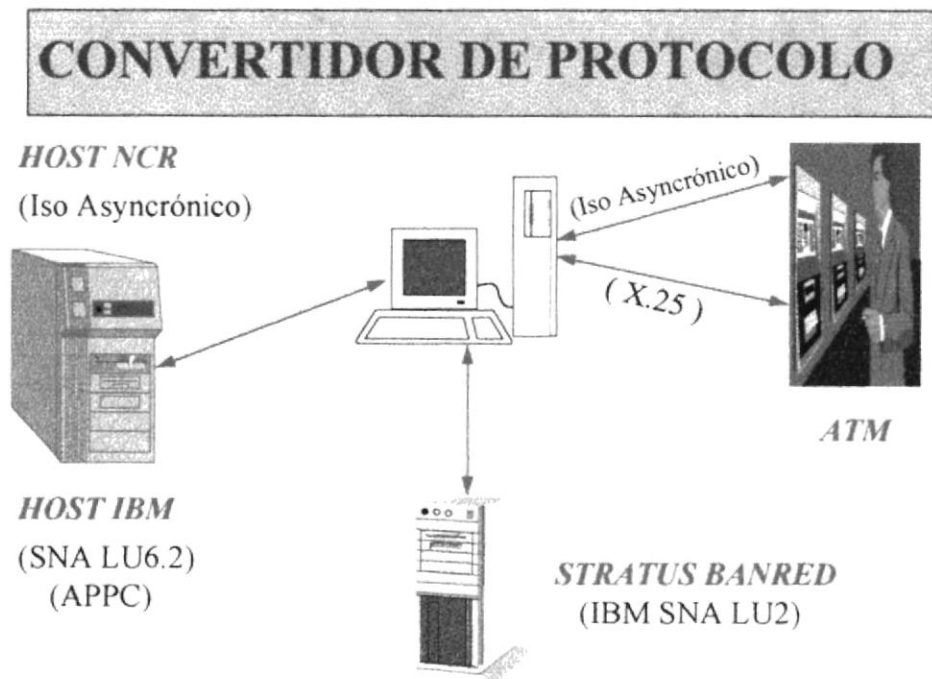


Figura No. 2

## RUTEADOR DE MENSAJE.

Las transacciones originadas en los distintos ATM son ruteadas en una de las siguientes direcciones :



- Hacia el banco emisor de la tarjeta de débito.  
Este caso aplica para aquellas transacciones realizadas en los Cajeros del mismo banco.
- Hacia el Host de **BANRED**.

Las transacciones ejecutadas por los clientes de un determinado banco en otros cajeros automáticos son enviados a **BANRED**.

### **AUTORIZACION EN MODO OFF-LINE**

Quando el Host del banco emisor se encuentra en modo Off-Line el **SERVATM** autoriza aquellas transacciones que cumplan con los límites de cupo establecidos por el banco emisor de la tarjeta de débito.

### **MANEJO DE REENTRY**

Las transacciones procesadas en modo Off-Line son enviadas al Host emisor de la Tarjeta de Débito en forma automática una vez que se reestablezca la comunicación con el mismo.

### **1.3 SOFTWARE REQUERIDO PARA EL DESARROLLO**

Para el desarrollo del **SERVATM** se utilizó el siguiente Software:

- Sistema Operativo IBM OS/2 Ver 2.1
- IBM Communications Manager/2 Ver 1.0
- Eicon X.25 Network-Level Developer's ToolKit
- Borland C++ for OS/2 Ver 1.0
- Microsoft COBOL for OS/2 Ver 4.5

## **1.4 HARDWARE REQUERIDO PARA EL DESARROLLO**

Para la puesta en marcha del **SERVATM** se requiere del Hardware que a continuación se menciona :

### **CARACTERISTICAS DEL COMPUTADOR**

- Arquitectura ISA o EISA
- Procesador 80486
- Memoria Base de 16 MegaBytes
- Velocidad Mínima de 60 Mhz
- Disco Duro de 500 MegaBytes
- Monitor VGA a color

### **COMUNICACION STRATUS - SERVER**

- Tarjeta IBM SDLC ISA MULTIPROTOCOL

### **COMUNICACION SERVER - HOST**

- Tarjeta Serial DIGI-BOARD de 4 u 8 puertos

### **COMUNICACION SERVER - ATM**

- Tarjeta Serial DIGI-BOARD de 4 u 8 puertos, para la conexión Iso Multiprotocol.
- Tarjeta EICON CARD, para la conexión X.25.

## 2 TUTORIAL DE PROTOCOLOS

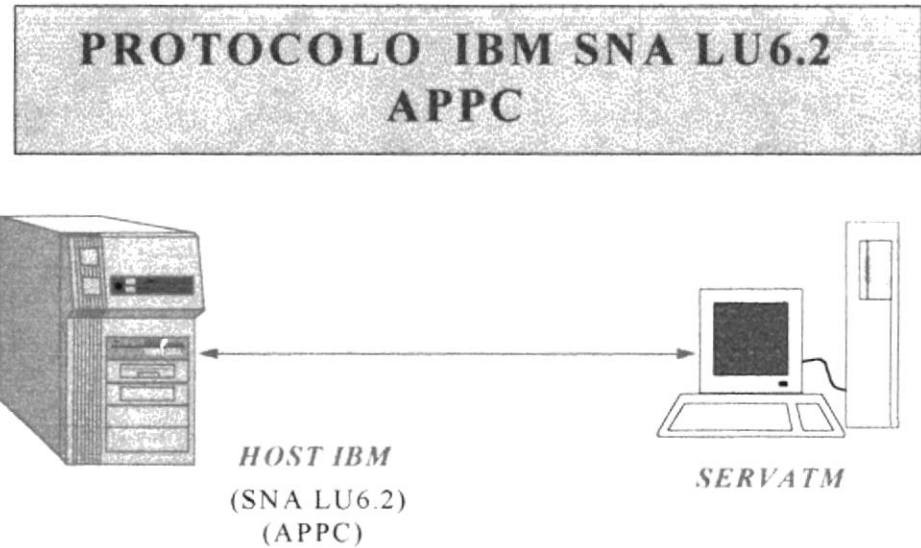


Figura No. 3

### 2.1 PROTOCOLO DE COMUNICACION APPC

Advanced Program to Program Communication (APPC), es un protocolo peer-to-peer el cual garantiza que dos programas puedan conversar o comunicarse a través de una serie de verbos, sin importar el sistema operativo en donde estén ejecutándose dichos programas.

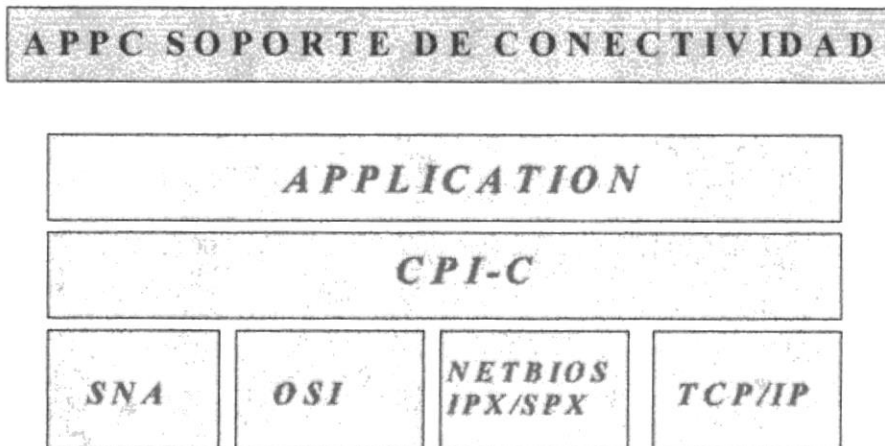


Figura No. 4

Para trabajar en este protocolo es necesario comprender los siguientes conceptos o términos :

### **2.1.1 Transaction Program (TP)**

Es el nombre lógico que una aplicación usa para comunicarse con otro programa el cual reside en otra máquina. Un TP puede tener hasta 64 bytes, por cada conversacion activa APPC asigna un TP ID (Transaction Program Identifier) y un Conversation Identifier.

### **2.1.2 Logical Unit (LU)**

Es el "socket" usado por un TP para acceder y transmitir data a traves de la red SNA.

Múltiples TP pueden usar los servicios proveídos por una LU.

Una Lu local se comunica con una LU remota (partner LU), por lo tanto es necesario definir y configurar una LU en cada máquina.

APPC usa un particular tipo de LU conocida como LU 6.2, en la cual ambas LU (Local y Partner) negocian la iniciación de la sesión de comunicación, a demás de esto la recuperación de errores es manejado en forma automática por el protocolo.

En APPC las LU son configuradas como independientes, es decir que cualquiera de las dos LU (Local y Partner) puede iniciar una sesión de comunicación.

### **2.1.3 SNA Session**

Una sesión de comunicación es una conección lógica entre dos LU, antes de que dos TP puedan conversar, es necesario que ambas LU esten en sesión.

Una sesión puede ser vista como un enlace entre dos LU bajo el cual fluyen las conversaciones entre dos TP.

El número de sesiones activas es configurada a través del parámetro “**Session Limits**”, cuyo valor por default es 8 y el número máximo que puede llegar es 32767.

**Sesiones Paralelas (Parallel Session)**, son usadas cuando es necesario tener mas de una sesión activa entre un par de LU.

En APPC existe una sesión de control **SNASVSG**, la cual permanece activa aunque no haya conversación entre dos TP, esto asegura que una sesión siempre esta disponible para el requerimiento de una conversación.

#### **2.1.4 Conversation**

Una conversación es el intercambio de información entre dos TP, preferiblemente una conversación dura poco tiempo, esto es el tiempo que le toma a un TP procesar una transacción.

Conversaciones en paralelo pueden ser establecidas a través de un par de LU.

En el esquema tradicional de SNA se requiere una sesión de comunicación por cada estación de trabajo, en APPC a través de la sesión de control se pueden activar múltiples conversaciones.

APPC maneja 2 tipos de conversaciones :

- **Básica**

Requiere programación de bajo nivel y los verbos requieren muchos más parámetros para ejecutar los comandos APPC.

- **Mapeada**

Su programación es de alto nivel y los verbos para ejecutar un comando APPC son más directos, es decir sin muchos parámetros.

### 2.1.5 Physical Unit (PU)

Una PU maneja el enlace físico sobre el cual las LU tienen una sesión de comunicación.

En APPC es necesario configurar o definir una **PU tipo 2.1** la cual soporta múltiples enlaces, múltiples sesiones y sesiones paralelas entre dos nodos, por ejemplo un servidor de comunicaciones sobre una LAN el cual está conectado al host se comporta como una PU tipo 2.1.

La relación entre Tps, Lus, Pus, Conversaciones y Sesiones es mostrada en la figura No. 5.

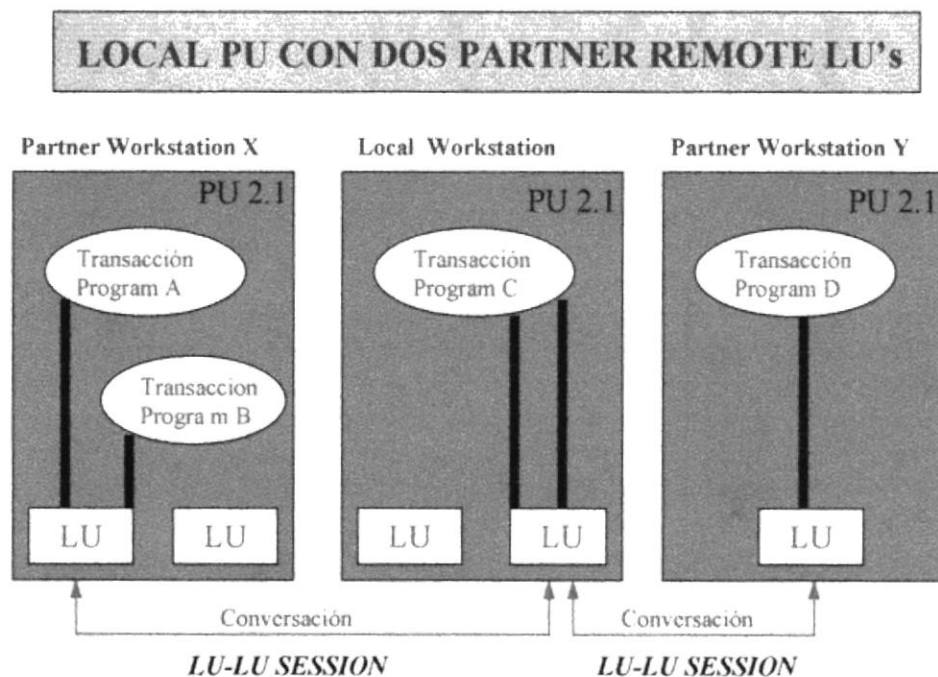


Figura No. 5

### 2.1.6 Sesiones y Conversaciones Paralelas

La siguiente figura demuestra como sesiones paralelas entre dos LU pueden ser usadas para proveer conversaciones concurrentes.

En este ejemplo, tres conversaciones estan siendo realizadas concurrentemente bajo tres sesiones paralelas entre la LU X (Local) y la LU Y (Partner).

Usando sesiones paralelas nos ahorramos tener que definir o crear LUs por cada sesión de comunicación.

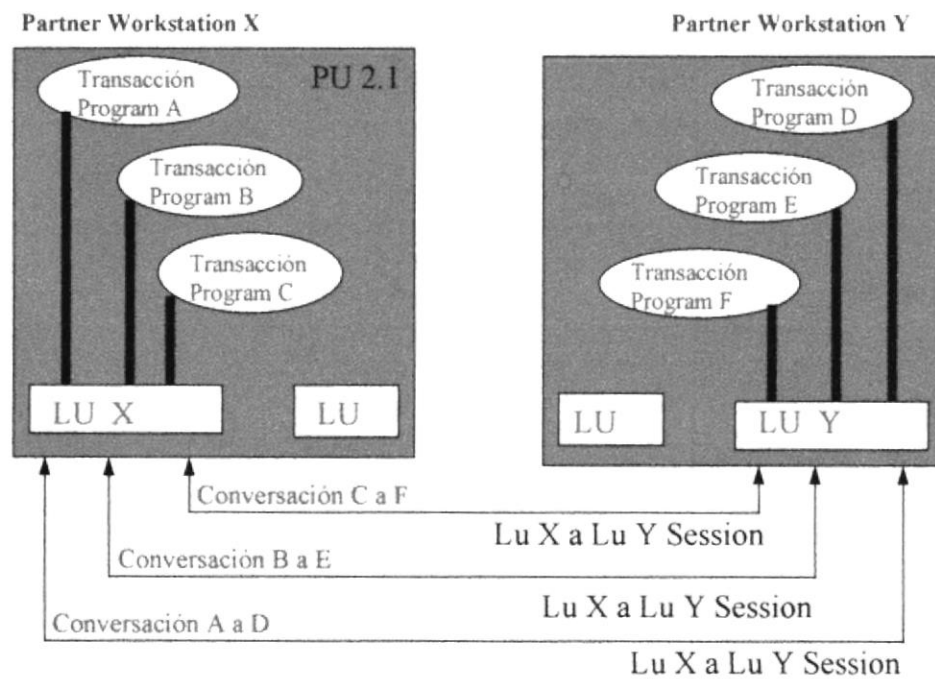


Figura No.6

## Attach Manager

El Attach Manager es un servicio que provee el Communications Manager para iniciar un APPC Transaction Program en respuesta a un requerimiento de una conversación, este permite cargar y ejecutar programas en forma automática sobre otros nodos u otras máquinas de la red.

### 2.1.7 VERBOS DE CONTROL PARA APPC

Existen categorías de verbos definidos para APPC, entre los cuales detallamos los siguientes :



- Appc TP Control Verbs
- Appc Conversation Control Verbs
- Appc Operator Control Verbs

### • APPC TP CONTROL VERBS

Nombre del Verbo	Descripción de la Función
Receive_Allocate	Acepta una Conversación, este verbo es usado como respuesta a un ALLOCATE.
Tp_Started	Inicia y reserva los recursos para un nuevo TP.
Tp_Ended	Termina y libera los recursos asignados a un TP
Get_Type	Consigue el tipo de conversación: Basic o Mapped.
Get_Tp_Properties	Este Verbo retorna las propiedades asociadas con un TP.

TABLA No. 2

### • APPC CONVERSATION CONTROL VERBS

Nombre del Verbo	Descripción de la Función
Allocate	Este verbo establece una conversación entre dos TP, y si no hay una sesión entre LU-to-LU esta es creada automáticamente.
Confirm	Es comando sirve para confirmar que un requerimiento haya llegado al partner o LU remota.
Confirmed	Este comando es usado como respuesta al verbo CONFIRM.
Deallocate	Termina una conversación.
Flush	Este verbo envia todos los requerimientos pendientes en forma inmediata.
Get_Attributes	Retorna los atributos de una determinada conversación.
Prepare_to_receive	Este verbo cambia el estado de la conversación, esto es de enviar a recibir.
Receive_and_post	Este comando sirve para recibir información. Usa un semáforo para avisarle al TP cuando las datos estan disponibles.

Receive_and_wait	Recibe información, el control no es retornado al TP hasta que lleguen datos al buffer de Appc.
Receive_inmediate	Recibe cualquier información disponible, el control es retornado inmediatamente al TP.
<b>Nombre del Verbo</b>	<b>Descripción de la Función</b>
Request_to_send	Este comando sirve para colocar la conversación en estado de SEND.
Send_Conversation	Este es un nuevo verbo el cual combina los siguientes verbos: ALLOCATE, SEND_DATA, DEALLOCATE.
Send_Data	Envia datos al TP remoto.
Send_Error	Este verbo sirve para indicarle al TP que un error fue detectado.
Test_Rts	Este verbo sirve para verificar que un REQUEST_TO_SEND fue recibido.

TABLA No. 3

• APPC OPERATOR CONTROL VERBS

Nombre del Verbo	Descripción de la Función
Activate_Dlc	Activa el enlace de comunicación.
Activate_Logical_Links	Activa un específico enlace lógico.
Cnos	Permite cambiar el límite del número de sesiones entre dos LU.
Deactivate_Conversation_Group	Desactiva una sesión identificada por un "conversation group ID".
Deactivate_Dlc	Desactiva un DLC (Data-Link-Control)
Deactivate_Logical_Link	Desactiva un específico enlace de SNA.
Deactivate_Session	Desactiva una sesión SNA LU 6.2.
Display	Muestra información a cerca de SNA LU 6.2, esto es: Network name, Node-ID, Machine type, Pu name.
Display_Appn	Muestra Información a cerca de APPN.
Start_Am	Inicia el Attach Manager.
Stop_Am	Termina el Attach Manager.

## TABLA No. 4

### X 2.2 PROTOCOLO DE COMUNICACION X.25

X.25 es una recomendación para acceder packet-switched network.

Packet-switched network es un Sistema de Comunicación de Datos, en el cual los datos son manejados y transportados en forma de paquetes.

Los paquetes son almacenados y dirigidos a cada nodo a través de una ruta de comunicación.

En X.25 el Data-Terminal Equipment (DTE) tiene acceso a la red a través de Network-Terminating Equipment llamado Data Circuit-Terminating Equipment (DCE). X

### Las tres capas de X.25

Los tres niveles de X.25 coinciden con los niveles del modelo OSI (International Standard Organization) tal como se muestra en la siguiente figura :

OSI	X.25
Application	
Presentation	
Session	
Transport	
Network	Packet Level
Data Link	Frame Level
Physical	Physical Level

Figura No. 7



## 2.2.1 PHYSICAL LEVEL (NIVEL I)

El nivel físico es el más bajo y más básico, este define la interfase física entre DTE y DCE.

La recomendación para el nivel físico incluye voltaje, señalización, tipo de conector y conexión de pines.

REFERENCIA	CIRCUITO	DIRECTION	
		DTE	DCE
G	Signal Groud		-----
GA	DTE Common Return		-----
T	Transmit	← - - - - -	
R	Receive	- - - - - →	
C	Control	← - - - - -	
I	Indication	- - - - - →	
S	Signal Element Taiment	- - - - - →	
B	Byte Timing (Opcional)	- - - - - →	

Figura No.8

### Transmit ( T )

Este es el circuito para la señal que lleva data para el DTE al DCE.

### Receive ( R )

Esto es similar a transmit pero en la dirección de DCE a DTE.

### Control ( C )

Este circuito siempre estará en "ON" durante la transferencia de datos. Durante la fase de control puede estar ya sea en "ON" o "OFF" bajo el control del DTE al DCE.

## **Indicate ( i )**

Esto permite al DCE especificarle al DTE el tipo de data que esta recibiendo.

Este indicador siempre esta en "ON" durante la transferencia de datos y estará en "ON" o en "OFF" durante el Call Control Phases.

## **Signal ElementTiming ( S )**

Este provee simplemente la señal del reloj (clock signal).

## **2.2.2 FRAME LEVEL (NIVEL II)**

Este nivel es responsable por el transporte de datos a traves del nivel físico, es una manera eficiente y rápida.

El X.25 Frame Level tiene cuatro distintas funciones :

- 1.- Este provee una eficiente manera de transferir datos a traves del enlace de comunicación.
- 2.- Este mantiene la sincronización entre el DTE y el DCE.
- 3.- Este chequea y maneja la recuperación de errores.
- 4.- Este informa al packet level el status del link.

## **2.2.3 PACKET LEVEL (NIVEL III)**

Esta es la capa mas importante de la recomendación X.25, algunas veces llamado Nivel Tres, este provee los procedimientos para controlar los Circuitos Virtuales entre el DTE y el DCE, como se muestra en la siguiente figura.

## FUNCION DEL PACKET LEVEL

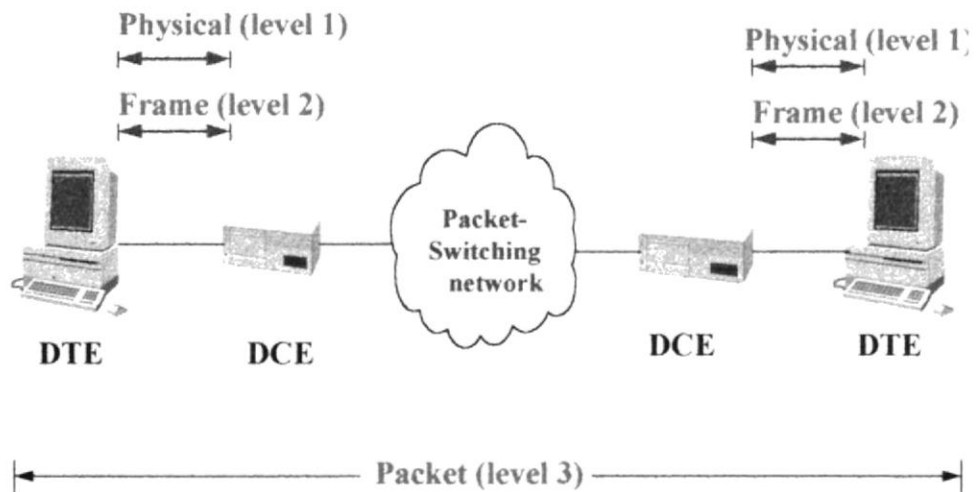


Figura No. 9

Las principales funciones del nivel tres son :

- 1.- Establecer y liberar los Circuitos Virtuales.
- 2.- Manejo de permanentes Circuitos Virtuales.
- 3.- Provisión de procedimiento.
- 4.- Control de flujo.
- 5.- Recuperación de errores.

### 2.2.4 CIRCUITOS VIRTUALES

Un circuito virtual da al DTE la impresión que tiene una exclusiva y privada conexión con otro DTE, sin importar que otros usuarios compartan el mismo enlace de comunicación.

En X.25 en un enlace de comunicación se pueden tener hasta 4096 circuitos virtuales activos concurrentemente.

X.25 tiene dos tipos de circuitos virtuales :

**(SVC) Swithed Virtual Circuit**, este es similar a usar una ordinaria línea telefónica donde el usuario marca un número telefónico, en X.25 simplemente se proporciona la dirección del nodo con el cual el DTE quiere conectarse y la comunicación se mantendrá mientras dure la llamada.

Las direcciones en X.25 pueden tener hasta 15 caracteres.

**(PVC) Permanet Virtual Circuit**, este es similar a una línea telefónica dedicada, los PVC tienen las mismas funciones que los SVC, en algunas circunstancias es preferible usar SVC debido a que el tiempo empleado en establecer la llamada es mínimo o insignificativo.

## **2.2.5 PASOS PARA ESTABLECER UNA CONECCION EN X.25**

**1.- CALL REQUEST**, este paquete es transmitido por el local DTE en el cual se especifica la dirección de destino, este paquete puede contener información adicional y servicios opcionales que pueden ser requeridos por el nodo remoto.

**2.- CALL ACCEPT**, Este comando o paquete es invocado por el nodo o DTE remoto, en respuesta a un Call Request.

**3.-** Si la conexión es establecida, el X.25 asigna un número de circuito virtual o Logical Chanel Number (LCN).

**4.-** Después de haber ejecutado los pasos de uno a tres, los dos DTE pueden empezar a realizar tranferencia de datos, esto es SEND y RECEIVED.

Los pasos anteriores son mostrados en la siguiente figura :



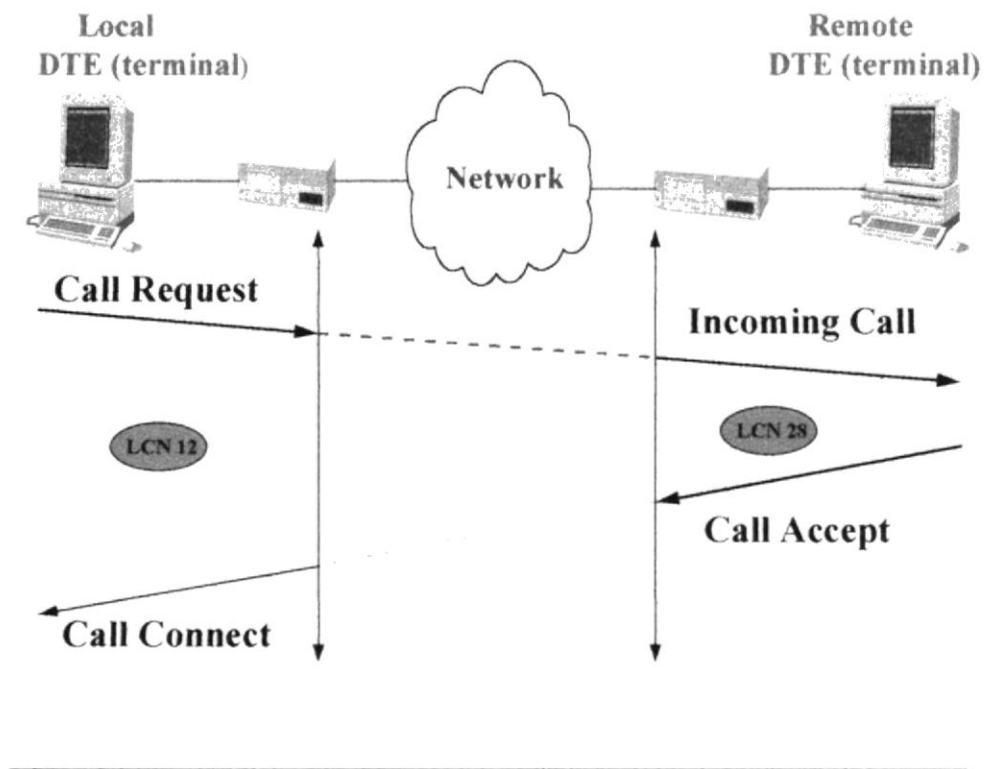


Figura No. 10



## 2.3 NCR / ISO ASYNCRONO

El hardware requerido para implementar este protocolo es un puerto serial con las siguientes características: 7 bit por carácter, paridad par, 1 stop bit, cada carácter es transmitido desde el menos significativo bit al más significativo.

La siguiente tabla muestra una descripción de las señales eléctricas con sus respectivos pines.

Signal	Pin	Source	Description
TXD	2	DTE	Transmit Data
RXD	3	DCE	Receive Data
RTS	4	DTE	Request To Send
CTS	5	DCE	Clear To Send
DSR	6	DCE	Data Set Ready. High while the port is enable
SGR	7	-	Signal Ground
DCD	8	DCE	Data Carrier Detect. High while the port is enable
DTR	20	DTE	Data Terminal Ready

TABLA No. 5

### 2.3.1 CARACTERES DE CONTROL

La siguiente tabla define los caracteres de control usados por este protocolo.

Caracter	Hex	Descripción
ACK	06	Afirmativa respuesta a una selección normal (Indica mensaje aceptado)
BCC	-	Block Check Character. Redundante carácter adicionado al final del mensaje, este sirve para detección y control de errores. BCC es calculado tomando la suma binaria (XOR) de cada uno de los 7 bits de los caracteres transmitidos. El STX no es incluido en el cálculo del BCC, pero el ETX si se incluye.
ENQ	05	El ENQ es usado como un carácter final, si hay una

Caracter	Hex	Descripcion
EOT	04	End Of Transmission. Este es usado para indicar la conclusión de una secuencia. Cuando el terminal recibe un EOT este espera por un Poll o por una Selección. El terminal Responde con un EOT al Poll cuando no hay datos.
ETX	03	End Of TeXt. Este caracter es usado para indicar el fin de mensaje.
NAK	15	Negative AcKnowlegement. Esto es una respuesta negativa a una selección (indica Not Ready to Receive) o a una transmisión indica error en BCC.
STX	02	Start of TeXt. Este caracter se usa para indicar el inicio del mensaje a ser transmitido.
US	1F	Unit Separator. Separador de Campos dentro del Mensaje enviado.

TABLA No. 6

### 2.3.2 DESCRIPCION DE LA SECUENCIA POLL/SELECT

El protocolo NCR ISO / ASYNCRONO usa la secuencia Poll/Select la cual es mostrada en la siguiente figura.

#### POLLING

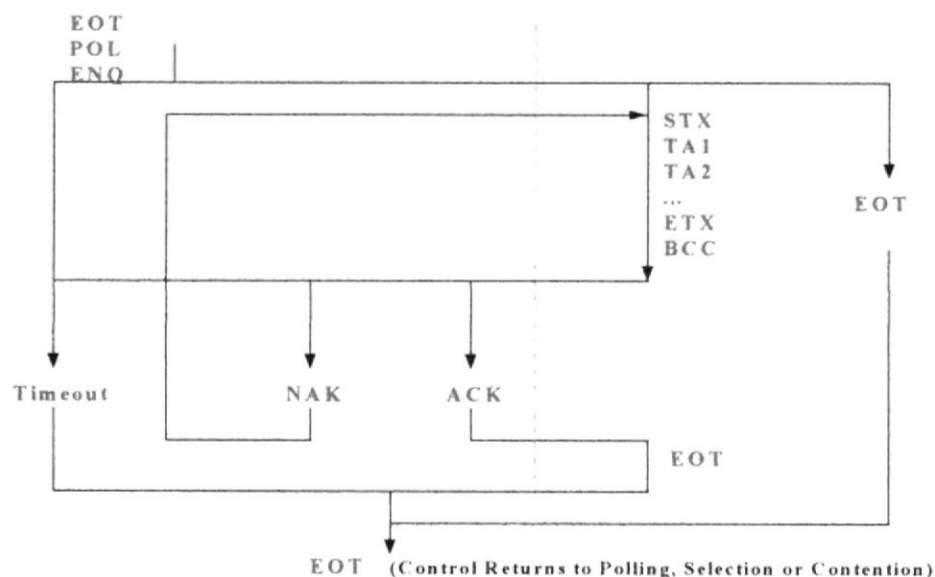


Figura No. 11

# SELECT

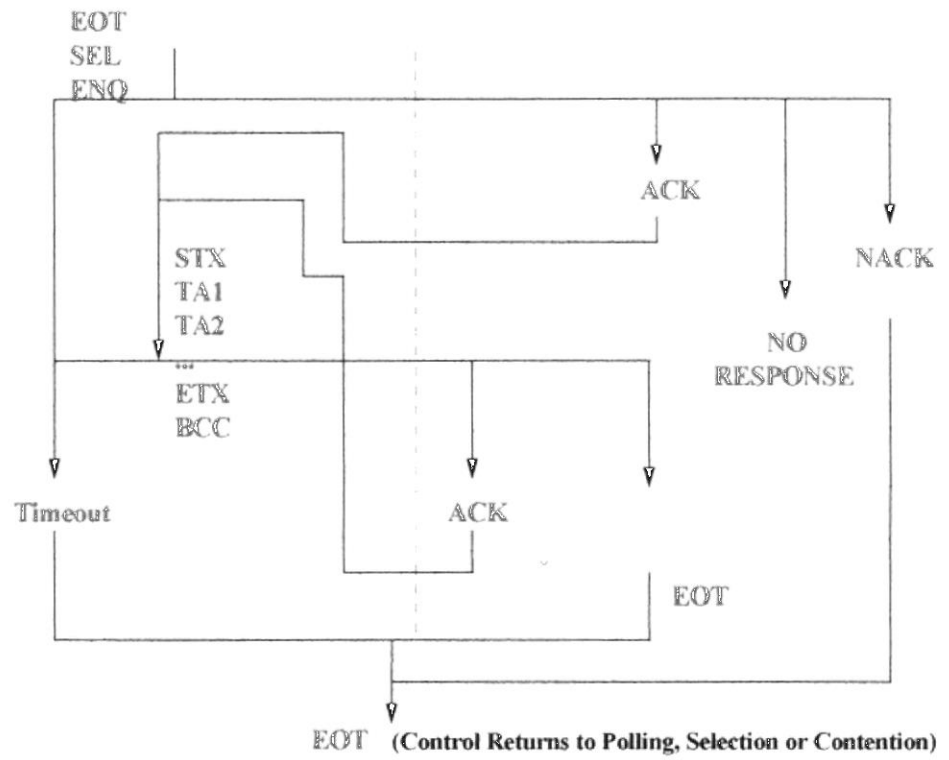


Figura No. 12

### 3. ESTRUCTURA DEL SERVATM

El **SERVATM** esta estructurado de acuerdo al siguiente diagrama de bloques.



---

Figura No 13

Cada uno de estos bloques serán descritos a continuación.

#### 3.1 LEER CONFIGURACION

Este bloque o función lee el archivo de configuración SERVATM.CFG, en este archivo estan definidos los dispositivos con los cuales el SERVATM se va a comunicar y los protocolos de comunicación, esto es:

- HOST STRATUS  
IBM SNA LU2.
- HOST EMISOR  
NCR ISO ASINCRONICO UTP.  
IBM SNA LU 6.2 (APPC).

- ATM  
NCR ISO ASINCRONICO UTP.  
X.25.

El formato del archivo **SERVATM.CFG** es:

```

*****
;
; Archivo de Configuracion SERVATM.CFG para
; definir los dispositivos de comunicacion conectados
; al SERVATM.EXE
;
; SE DEBE RESPETAR ESTRICTAMENTE EL
; FORMATO A CONTINUACION DETALLADO
;
; NAMEDEVICE PROTOCOLO FILENAME TARJETA/COM
; =====
; STRATUS SNALU2 STRATUS.CFG TARJETA: 0
; HOSTEMISOR SNALU6.2 HOSTAPPC.CFG TARJETA:
0
; HOSTEMISOR ISONCRUTP HOSTUTP.CFG COM : 2
;
; ATM ISONCRUTP ATMCOM3.CFG COM : 3
; ATM ISONCRUTP ATMCOM4.CFG COM : 4
; ATM X.25 ATMX25.CFG TARJETA: 0
;
; CONFIGURACION PARA EL BANCO DE CREDITO
*****
;
; STRATUS SNALU2 STRATUS.CFG TARJETA: 0
; HOSTEMISOR ISONCRUTP HOSTUTP.CFG COM : 2
; ATM ISONCRUTP ATMCOM3.CFG COM : 3
; ATM ISONCRUTP ATMCOM4.CFG COM : 4
; ATM ISONCRUTP ATMCOM5.CFG COM : 5

```

**NOTA:**

Las líneas que comienzan con punto y coma son consideradas como comentario.

### 3.2 CREAR QUEUES

El **SERVATM** crea a tiempo de ejecución 6 Colas (Queues), las cuales sirven para almacenar y procesar los mensajes o requerimientos que vienen y van hacia los siguientes dispositivos:

- HOST STRATUS
- HOST EMISOR
- HOST ATM

Al crear cada Queue también se crea una tarea o thread, esta tarea es la encargada de controlar, procesar los requerimientos que llegan a la misma y enviar la respuesta a dichos requerimientos a otra Queue, tal como se muestra en la siguiente figura.

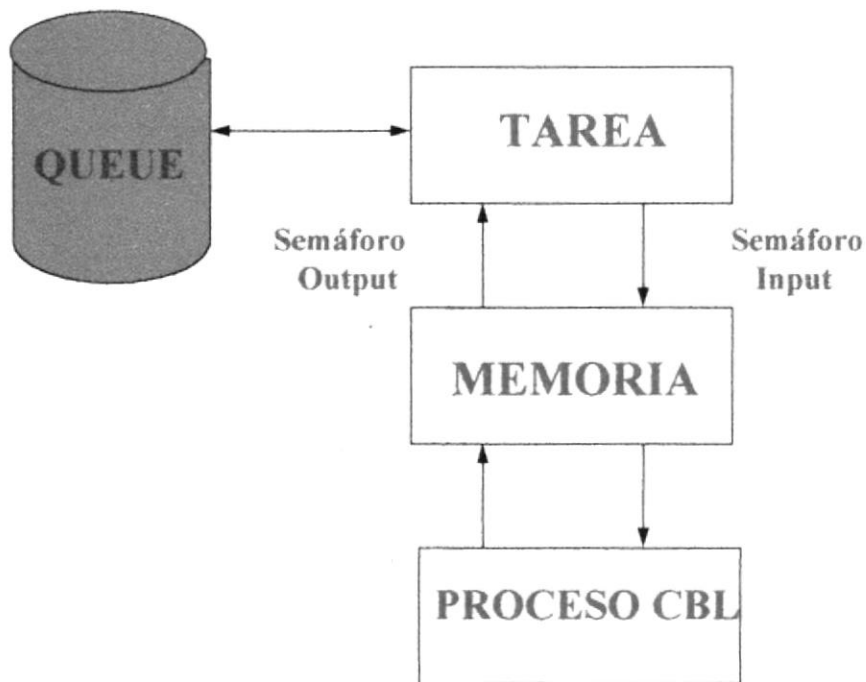


Figura No 14

los mensajes que llegan a la Queue son enviados a los PROCESOS CBL, a través de una área de memoria y un par de semáforos.

Los PROCESOS CBL son programas ejecutables, los cuales serán descritos en la siguiente sección.

A continuación se describen cada una de las Queues.

- **QUEUE R5MSG**

Esta Queue es la encargada de procesar y controlar los requerimientos que vienen y van hacia el STRATUS.

- **QUEUE HOSTMSG**

Esta Queue es la encargada de procesar y controlar los requerimientos que vienen y van hacia el HOST.

- **QUEUE ATMMSG**

Esta Queue es la encargada de procesar y controlar los requerimientos que vienen y van hacia los ATM.

- **QUEUE ATMX25MS**

Esta Queue es la encargada de procesar y controlar los requerimientos que vienen y van hacia los ATM conectados bajo protocolo de comunicación X.25.

- **QUEUE AUTHMSG**

Esta Queue es la encargada de procesar y controlar los requerimientos que van a ser autorizados localmente.

- **QUEUE REENTRY**

Esta Queue es la encargada de activar, procesar y controlar los requerimientos pendientes por enviar al HOST.

### **3.3 CREAR PROCESOS**

El **SERVATM** a tiempo de ejecución, crea y carga los PROCESOS CBL, esto procesos reciben los requerimientos

que llegan a las respectivas Queue, a demás de esto deciden a donde van enviar la respuesta.

Los PROCESOS CBL, son programas ejecutables realizados en Cobol for OS/2 , ellos son:

R5MSG.EXE  
ATMMSG.EXE  
ATMX25MS.EXE  
HOSTMSG.EXE  
AUTHMSG.EXE  
REENTRY.EXE

Por cada proceso el **SERVATM** crea una área de memoria común y un par de semáforos, tal como se muestra en la figura No 14.

- **AREA DE MEMORIA COMUN**

Esta área de memoria sirve para que el PROCESO CBL tome los requerimientos que llegan a su respectiva Queue, y esta estructurada de la siguiente manera:

```
typedef struct tag_comonarea
{
char id_device[9];           // Id. del dispositivo que procesa el mensaje
char modo_lu;               // Modo de la LU: R = Rcvd , S = Send
char strsecuencia[4];       // Secuencia de Transacciones de los ATM
char modo_utp;              // Modo del Terminal NCR UTP R = Rcvd , S = Send
char strnumutp[3];          // Numero del Terminal NCR UTP
char strnumqueue[3];        // Numero de la Queue para los ATM
char strasecuen[4];         // Secuencia de los mensajes recibidos desde el
stratus
char filler1[5];            // Filler
char flag_fin;              // Fin de proceso
char comand_origen[2];      // Comando de Origen
char comand_destino[2];     // Comando de Destino
char result_code[4];        // Result code
char strlendata[4];         // Longitud de la data
char filler2[57];           // Filler
char data[1024];            // Data
char filler3[924];          // Filler
} COMONAREA;
```

- **SEMAFORO DE INPUT**



El semáforo de input sirve para avisarle al PROCESO CBL que tiene un mensaje en la área de memoria común.

- **SEMAFORO DE OUTPUT**

El semáforo de output sirve para que el PROCESO CBL le avise al **SERVATM** que tome la respuesta que se encuentra en la área de memoria común.

### **3.4 CREAR TAREAS PARA LAS COMUNICACIONES**

Para el manejo de las comunicaciones con los diferentes dispositivos el **SERVATM** crea las siguientes tareas:

- **RCVD\_REQ\_STRATUS**

Esta tarea realiza las siguientes funciones:

1. Recibe el BIND enviado por el STRATUS para abrir la sesión de comunicación.
2. Recibe los requerimientos enviados por el STRATUS.
3. Coloca los requerimientos en la Queue R5MSG.
4. Devuelve una respuesta al STRATUS.

- **SEND\_REQ\_STRATUS**

Esta tarea realiza las siguientes funciones:

1. Recibe el BIND enviado por el STRATUS para abrir la sesión de comunicación.
2. Recibe los requerimientos enviados desde los ATM, los cuales se encuentra en la Queue R5MSG.
3. Envía los requerimientos al STRATUS.

4. Colocan la respuesta enviada por el STRATUS en la Queue R5MSG.

- **HOST\_COM**

Esta tarea realiza las siguientes funciones:

1. Maneja el protocolo NCR ISO ASINCRONICO UTP.
2. Recibe los requerimientos que llegan a la Queue HOSTMSG.
3. Envía los requerimientos al Host.
4. Recibe la respuesta del Host.
5. Coloca la respuesta del Host en la Queue HOSTMSG.

- **HOST\_APPC**

Esta tarea realiza las siguientes funciones:

1. Maneja el protocolo IBM SNA LU 6.2 (APPC).
2. Recibe los requerimientos que llegan a la Queue HOSTMSG.
3. Envía los requerimientos al Host vía APPC.
4. Recibe la respuesta del Host.
5. Coloca la respuesta del Host en la Queue HOSTMSG.

- **ATM\_COM**

Esta tarea realiza las siguientes funciones:

1. Maneja el protocolo NCR ISO ASINCRONICO UTP.

2. Recibe los requerimientos enviados desde los ATM'S.
3. Coloca los requerimientos en la Queue ATMMMSG.
4. Envía la respuesta a los ATM'S.

- **ATM\_X25COM**

Esta tarea realiza las siguientes funciones:

1. Maneja el protocolo de comunicación X.25.
2. Recibe los requerimientos enviados desde los ATM'S.
3. Coloca los requerimientos en la Queue ATMX25MS.
4. Envía la respuesta a los ATM'S.

## 4 USO DE PROTOCOLO DE COMUNICACIONES

### 4.1 COMO EL *SERVATM* MANEJA EL PROTOCOLO APPC

El protocolo APPC es manejado a través del Communication manager/2 (CM/2) Version 1.0.

El CM/2 tiene un conjunto de API (Application Program Interface), los cuales me permiten invocar los verbos que confirman al protocolo APPC.

Estos API están descritos en el manual (APPC Program Refenece).

El *SERVATM* está desarrollado con programación orientada a objetos en lenguaje C++, por lo tanto a continuación se describe el objeto que maneja la comunicación APPC.

```
class APPC_MAP
{
public:
    UCHAR tp_id[8];           // TP ID
    ULONG conv_id;           // Conversation ID
    PCH appcbuff;            // Address of allocated buffer
    USHORT lenappcbuff;      // Len of appcbuff
    char modedescri[9];      // Mode Description Name
    short int timeout;       // Time Out

    USHORT pri_rc;           // Primary RETURN_CODE
    ULONG sec_rc;           // Secondary RETURN_CODE

    // Constructor del Objeto APPC_MAP
    APPC_MAP();

    // Destructor del Objeto APPC_MAP
    ~APPC_MAP();

    // Reserva una area de memoria la cual es utilizada por los API de APPC
    void Appcalloc(USHORT len)

    // Libera el area de memoria reservada por APPC
    void Appcfree(void);

    // Inicia un Transaction Program
    USHORT Starttp(char *llu_alias, char *tp_name);

    // Termina un transaction program
    USHORT Endtp(void);
```

```

// Estable o inicia una conversacion
USHORT Allocate(UCHAR * plu_name, UCHAR * tp_name, UCHAR * mode_name,
                UCHAR rtn_ctl, UCHAR synch_level);
// Acepta una conversacion invocada por un Transaction Program
USHORT Rcvdallocate(UCHAR * tp_name);

// Envia datos
USHORT Senddata(UCHAR *data, USHORT len, UCHAR type);

// Recibe Datos
USHORT Rcvddata(UCHAR *data, USHORT &maxlen);

// Consigue el codigo de retorno del ultimo comando
void GetStrRC(char *pc);
};

```

## Ejemplo de como usar este objeto:

```

#include <iostream.h>
#include <string.h>

// Header en donde se define al objeto APPC_MAP
#include "appc.hpp"

// Declaro y defino el objeto APPC_MAP
APPC_MAP appc;

char buffer[1024];
int len;

int main()
{
// Reservo una area de memoria la cual es usada internamente por el API de APPC
appc.Appcalloc(1024);

// Inicia un TP
appc.Starttp("LOCALLU", "TP01");

// Inicia una conversacion con un remoto TP
appc.Allocate("REMOTELU", "TP01", "APPCMODE",
             AP_WHEN_SESSION_ALLOCATED,
             AP_CONFIRM);

strcpy(buffer, "Mensaje para el remoto TP");
len = strlen(buffer);

// Envia en mensaje en forma inmediata y se prepara para recibir una respuesta
appc.Senddata(buffer, len, AP_SEND_DATA_P_TO_R_FLUSH);
// Recibe Data
len = 1024;
appc.RcvdData(buffer, len);
// Muestra el dato recibido
cout << buffer;

```

```
// Termina la conversacion y el TP
appc.Endtp();

return 0;
}
```

Este ejemplo muestra lo fácil que es trabajar bajo el protocolo APPC, dentro de una programación orientada a objetos.

## 4.2 COMO EL *SERVATM* MANEJA EL PROTOCOLO X.25

El protocolo X.25 esta implementado usando el software Eicon X.25 Network-Level Developer ToolKit.

A continuación se describe el objeto que maneja la comunicación en X.25.

```
class X25PORT
{
public:
char remote[16];           // Remote DTE address
char local[16];           // Local DTE address
char *sendbuf;            // Buffer for send
char *rcvdbuf;           // Buffer for receive
struct x25data udata;     // Call / Listen user data
struct x25data *p_facility; // Call / Listen facility
short int info;           // Call / Listen info parameter
short int port;           // Call / Listen port parameter
short int cid;            // Connection identifier
unsigned short int lsn;   // Logical Sesion Number
struct x25data *p_inc_udata; // Incomming Call user data.
struct x25doneinfo done_info; // Info about a completed request.
struct x25stat status_connexion; // Status buffer.

int timeout;              // Time Out
// Constructor del objeto X25PORT
X25PORT(int portid = 1);

// Destructor del objeto X25PORT
~X25PORT();

// Asigna las variables de un puerto X.25 a otro Objeto
void Setport(X25PORT *x25p);

// Recibe una Llamada
int RcvdCall(int waittime);

// Cierra una Llamada
CloseCall(void);

// Hace una llamada a una direccion
int Callto(int waittime,char *remoteadd, char *localadd = 0, char *myuserdata = 0);

// Envia Datos
int SendData(char *data, int len);

// Recibe Datos
int RcvdData(char *data, int &len);

// Espera que un comando de X.25 sea realizado
int Done(int waittime);

// Cancela la coneccion de un circuito virtual
```

```

int Cancel(void);

// Retorna el buffer el ultimo mensaje recibido
char *GetData(void) const { return rcvdbuf; }

// Retorna la Longitud del ultimo mensaje recibido
int GetLenData(void)

// Consigue el User Data enviado en el Call Request
char *Getuserdata(void) { return udata.xd_data;}

// Define el Time de espera para la culminacion de un comando X.25
void Settimeout(int sec)
};

```

## Ejemplo de como usar este objeto:

```

#include <iostream.h>
#include <string.h>

// Header en donde se define al objeto X25PORT
#include "x25port.hpp"

// Declaro y defino el objeto X25PORT
X25PORT x25;

char buffer[1024];
int len;

int main()
{

// Realiza una llamada a un remoto DTE y establece un nuevo circuito virtual
x25.Callto(20, "1234", "5678");

strcpy(buffer,"Mensaje para el remoto DTE");
len = strlen(buffer);

// Envia en mensaje
x25.SendData(buffer, len);

// Recibe Data
len = 1024;
x25.RcvdData(buffer, len);

// Muestra el dato recibido
cout << buffer;

// Cierro el circuito virtual
x25.CloseCall();

return 0;
}

```





### 4.3 COMO EL *SERVATM* MANEJA EL PROTOCOLO NCR ISO / ASYNCHRONO

El *SERVATM* usa una tarjeta DIGIBOARD de 4 o 8 puertos asincrónicos, de esta forma en cada puerto se pueden conectar varios ATM.

Los puertos seriales son manejados usando los driver que vienen con la tarjeta DIGIBOARD.

Para hacer la programación mas sencilla se creó un objeto que maneja al puerto serial, el cual es descrito a continuación:

#### // ESTRUCTURA PARA DEFINIR DATOS GENERALES DEL PUERTO SERIAL

```
typedef struct tag_ncrport {
char comport[9];           // Nombre del puerto, COM3, COM4 ..
HFILE com;                // Manejador del Puerto
ULONG act;                // Codigo de la accion a tomar cuando falla la apertura
                           // del puerto serial
USHORT baud;              // Velocidad
BYTE dbits;                // Longitud
BYTE parity;              // Paridad
BYTE sbits;                // Stop Bit
char databuff[1024];      // Buffer del Puerto
LONG WriteTimeout;        // 25 * 0.01 sec transmit timeout
LONG ReadTimeout;         // 25 * 0.01 sec receive timeout

} PORTNCR;
```

#### // CLASE U OBJETO QUE MANEJA EL PUERTO SERIAL

```
class SERIALPORT {

public:

PORTNCR desport;          // Descripcion General del puerto
PORTNCR *portncr;

// Constructor del objeto SERIALPORT
SERIALPORT() { portncr = &desport; }

// Abre el Puerto Serial
int Openport(void);

// Cierra el Puerto Serial
int Closeport(void);

// Inicializa y Configura el Puerto Serial
```

```

int Initcomm(void);

// Escribe un Caracter sobre el puerto
APIRET Comout(char c);

// Lee un Caracter del Buffer del Puerto
APIRET Cominp(char *pc);

// Envia una cadena de caracteres
APIRET Sendstr(char *pc);

// Envia una Cadena de caracteres especificando el numero de bytes a ser
// transmitidos
APIRET SendData(char *pc, int lenpc);

// Espera un determinado tiempo para que llegue un caracter al buffer del puerto
APIRET RcvdWaitChar(char *pc, LONG mytimeout);

// Retorna el Nombre del Puerto
char *Getnameport(void) { return desport.comport; }
};

```

## EJEMPLO DE COMO USAR ESTE OBJETO

```

#include <iostream.h>
#include <string.h>

// Contiene la declaracion del objeto SERIALPORT
#include "portseri.hpp"

// Defino al objeto SERIALPORT
SERIALPORT asyncport;

char buffer[1024];
int len;

int main()
{
// Defino el Nombre del Puerto COM3, COM4,... COM8
strcpy(asyncport.desport.comport, "COM3");

// Defino la Velocidad de transmision
asyncport.desport.baud = 9600;

// Defino el numero de Bits por cada Byte
asyncport.desport.dbits = 7;

// Defino un Stop Bit por cada Byte Transmitido
asyncport.desport.sbits = 1;

// Defino la Paridad E = EVEN O = ODD
asyncport.desport.parity = 'E';

// Abro e Inicializo el puerto con los parametros especificados anteriormente
if (asyncport.Openport())
{

```

```

    cout << "\nPUERTO: " << asyncport.GetNameport() << "NO PUDO SER ABIERTO..";
    return 0;
}

strcpy(buffer, "ESTOY PRUEBANDO EL PUERTO SERIAL ");
strcat(buffer, asyncport.GetNameport());
len = strlen(buffer);

// Envio datos por el puerto serial
asyncport.SendData(buffer, len);

// Espero a que llegue un caracter al buffer del puerto
// El tiempo de espera es de 15 Seg.
if (asyncport.RcvdWaitChar(buffer, 15))
    cout << "\nNO HAY DATOS EN EL BUFFER DEL PUERTO SERIAL";
else
    cout << "\nBYTE RECIBIDO: " << buffer[0];

asyncport.Close();

return 0;
}

```

## **5 CONCLUSIONES**

De los protocolos de comunicación usados por el **SERVATM**, el más recomendado es APPC (Advanced Program-to-Program Communications), las principales ventajas son:

Eficiencia en la Transmisión de Datos.

Niveles de Seguridad para poder ejecutar un programa en otro computador.

Enrutamiento de transacciones, una transacción puede ser ejecutada en otro computador sin importar la ubicación física.

Interconectividad, el protocolo APPC corre bajo los siguientes ambientes:

- **SDLC**
- **TOKEN RING**
- **ETHERNET**
- **TCP/IP**
- **IPX/SPX**
- **NETBIOS**

Fácil de configurar y mantener.

La Recuperación de errores es manejada internamente por el protocolo APPC.

Las sesiones de comunicación y conversaciones pueden ser iniciadas en forma independiente, es decir que cualquiera de los dos computadores puede iniciar una sesión y una conversación.

## BIBLIOGRAFIA:

OS/2 NOTEBOOK Versión 2.0  
Dick Conklin / General Editor  
Publicado por Microsoft Press.

IBM Conventional LU Application Programming  
Reference  
Versión 1.3  
Publicado por IBM.

IBM APPC Programming Reference  
Versión 1.3  
Publicado por IBM

X.25 MADE EASY  
Nicolas M. Thorpe and Derek Ross.  
Publicado por Prentice Hall.

CLIENT/SERVER PROGRAMMING WITH OS/2  
Versión 2.0  
Robert Orfali and Dan Harkey.  
Publicado por VNR COMPUTER LIBRARY.



BIBLIOTECA