TRABAJO ESPECIAL DE GRADO

IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED LOCAL INALÁMBRICA BASADA EN LA TECNOLOGÍA BLUETOOTH

Presentado ante la Ilustre
Universidad Central de Venezuela
Por el Br. Leguízamo L., Daniel A.
Para optar al título de
Ingeniero Electricista

Caracas, 2006

TRABAJO ESPECIAL DE GRADO

IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED LOCAL INALÁMBRICA BASADA EN LA TECNOLOGÍA BLUETOOTH

Prof. Guía: Ing. Paolo Maragno
Tutor Industrial: Ing. Rafael Rodríguez

Presentado ante la Ilustre
Universidad Central de Venezuela
Por el Br. Leguízamo L., Daniel A.
Para optar al Título de
Ingeniero Electricista

Caracas, 2006

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERIA ELECTRICA
DEPARTAMENTO DE COMUNICACIONES

CONSTANCIA DE APROBACIÓN

Caracas, 01 de agosto de 2006

Los abajo firmantes, miembros del Jurado designado por el Consejo de Escuela de Ingeniería Eléctrica, para evaluar el Trabajo Especial de Grado presentado por el Bachiller Leguizamo L. Daniel A., titulado:

"IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED LOCAL INALÁMBRICA BASADA EN LA TECNOLOGÍA BLUETOOTH"

Consideran que el mismo cumple con los requisitos exigidos por el plan de estudios conducente al Título de Ingeniero Electricista en la mención de Comunicaciones, y sin que ello signifique que se hacen solidarios con las ideas expuestas por el autor, lo declaran APROBADO.

Prof. Freddy Brito
Jurado

Prof. Rafael Arruebarrena
Jurado

Prof. Paolo Maragno
Prof. Guia

Edificio Escuela de Ingeniería Eléctrica, piso 1, oficina 201, Ciudad Universitaria, Los Chaguaramos, Caracas 1051, D.F.
TELÉFONOS. (VOZ) +58 212 6053300 (FAX) +58 212 6053105
Mail: eie-com@elecrisc.ing.ucv.ve

DEDICATORIA

Este trabajo lo quiero dedicar a mis padres y a mi hermana, con quienes comparto a diario y quienes me han brindado todo el apoyo y toda la fuerza necesaria desde mi primer día de vida. Son los seres que más quiero en este mundo. Sé que siempre me seguirán apoyando y ustedes saben que yo también estaré ahí para lo que sea, porque nosotros cuatro formamos un equipo grandioso.

También quiero dedicar este fruto del esfuerzo a todas aquellas personas, amigos y familiares, que se encuentran alrededor de mi vida y que siempre han creído en mí, en especial a mi hermano del alma David Cruz y a los integrantes del grupo EMAUS, que aunque a veces no estén cerca físicamente, siempre están presentes de alguna u otra forma con su cariño incondicional.

RECONOCIMIENTOS Y AGRADECIMIENTOS

Le doy gracias a Dios por haberme regalado la dicha de nacer en el seno de un hogar donde siempre ha existido la unión y el amor.

Gracias mamá, gracias papá, porque han sido ustedes los que han logrado que yo cumpla hoy una de mis grandes metas. Siempre han hecho todo el esfuerzo posible y han invertido gran parte de su vida para que mi hermana y yo nos superemos profesional y personalmente. Gracias por ser las personas que son.

A mi hermana que siempre me ha ayudado en todos mis estudios desde muy niño, y me ha dado un aporte invaluable para la elaboración de este trabajo.

Al Ingeniero Alejandro Paredes, quien fue la persona que me dio todas las herramientas necesarias para realizar este trabajo, desde antes de su inicio hasta su culminación. Mucha suerte y que sigas alcanzando todas las metas que te propones.

Al Ingeniero Rafael Rodríguez, que me dio la oportunidad de desarrollar este trabajo en la empresa BCCOM Business y me brindó todas las facilidades y consideraciones que fueron necesarias para la elaboración del proyecto.

A la bella y única María, que con su buena atención y excelente disposición me asesoró en todo momento para realizar los trámites necesarios antes y después de la presentación de este trabajo de grado, muchas gracias, hace falta mas gente como tú en la Escuela de Ingeniería Eléctrica.

Al Profesor Paolo Maragno, por sus atenciones oportunas y asesoramientos finales para la entrega y presentación de este trabajo.

A mis compañeros Oswaldo Denis, Victor Contreras, René Gruber y Rafael Quiroz, que me han dado esa ayuda extra que siempre es importante recibir.

A los Ingenieros Edwar Villegas y Giovanny Jurado, que me han dado la oportunidad de empezar a desarrollarme profesionalmente en una empresa reconocida y me brindaron la disponibilidad de tiempo y la motivación necesaria para poder culminar mi trabajo de grado.

Leguízamo L., Daniel A.

IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED LOCAL INALÁMBRICA BASADA EN LA TECNOLOGÍA BLUETOOTH

Prof. Guía: Ing. Paolo Maragno. Tutor Industrial: Ing. Rafael Rodríguez. Tesis. Caracas. UCV. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista. Opción: Comunicaciones. Institución: BCCOM Business, C.A. 2005, 119 h + anexos.

Palabras Claves: Bluetooth, Comunicaciones Inalámbricas, Picored Local.

Resumen. Siendo el tema de seguridad y vigilancia uno de los que más aqueja nuestras comunidades, la empresa BCCOM Business plantea el siguiente trabajo para implementar y desarrollar una picored local inalámbrica, haciendo uso de módulos Bluetooth estudiados y adquiridos previamente, orientada hacia una aplicación de seguridad para la supervisión de accesos de entrada y salida de una instalación privada. Para esto se hizo una propuesta de diseño de un sistema de vigilancia que integra dentro de sus equipos, módulos *Blu2i* marca TDK para implementar una picored Bluetooth y que esta fuera el eje principal del sistema propuesto. Luego de implementar la picored, cumpliendo con los requerimientos de Hardware y Software necesarios para tal fin, fue incorporada al sistema de seguridad propuesto, formado por un módulo maestro conectado a un computador y dos módulos esclavos remotos conectados a detectores de movimiento, a través de las interfaces adecuadas. La picored Bluetooth cumplió satisfactoriamente con las demandas del sistema de seguridad y vigilancia propuesto

ÍNDICE

DEDICATORIA ..... **II**
RECONOCIMIENTOS Y AGRADECIMIENTOS ..... **IV**
ÍNDICE ..... **VI**
INDICE DE TABLAS ..... **X**
INDICE DE FIGURAS ..... **XI**
ACRÓNIMOS ..... **XIII**
ACRÓNIMOS ..... **XIII**
INTRODUCCIÓN **1**
PLANTEAMIENTO DEL PROBLEMA **3**
OBJETIVOS 5
Objetivo General 5
Objetivos Específicos 5
CAPÍTULO I **6**
1 DESCRIPCIÓN GENERAL DE LA TECNOLOGÍA INALÁMBRICA BLUETOOTH **6**
1.1 RESEÑA HISTÓRICA 6
1.2 BLUETOOTH 7
1.3 ¿CÓMO FUNCIONA BLUETOOTH? 8
1.3.1 Banda Base 17
1) Descripción general 17
2) Canal físico. 19
3) Enlace físico. 20
4) Paquetes. 22
5) Corrección de errores. 26
6) Transmisión/Recepción. 27
7) Control de Canal. 27
8) Seguridad en Bluetooth. 31
1.3.2 Protocolo de Gestión de Enlace (LMP) 31
1) Establecimiento de conexión. 33
1.3.3 Interfaz del Controlador de Host (HCI) 33
1) Capas mas bajas del conjunto Bluetooth. 33
2) Posibles arquitecturas de bus físico. 34
1.3.4 Protocolo de Control y Adaptación de Enlace Lógico (L2CAP) 35
1) Canales. 35
2) Operaciones entre Capas. 36
3) Eventos. 37

4) Acciones. 37
5) Formato del paquete de datos. 37
6) Segmentación y Reensamblado. 38
7) Calidad de servicio (QoS). 39
1.3.5 Protocolo de Descubrimiento de Servicio (SDP) 39
1) Descripción General. 40
2) Registros de servicio. 40
3) El protocolo. 40
1.3.6 RFCOMM 41
1.3.7 Perfiles Bluetooth 42
1) Perfil Genérico de Acceso (GAP). 43
2) Perfil de Puerto Serie. 44
3) Perfil de Aplicación de Descubrimiento de Servicio (SDAP). 44
4) Perfil Genérico de Intercambio de Objetos (GOEP). 44
5) Perfil de Telefonía Inalámbrica. 44
6) Perfil de Intercomunicador. 45
7) Perfil de Manos Libres. 45
8) Perfil Dial-up Networking. 45
9) Perfil de Fax. 45
10) Perfil de Acceso LAN. 45
11) Perfil Object Push. 45
12) Perfil de Transferencia de Archivos. 46
13) Perfil de Sincronización. 46
CAPITULO II 47
2 CONCEPTOS BÁSICOS PARA LA IMPLEMENTACIÓN DE UNA RED DE ÁREA PERSONAL BLUETOOTH. 47
2.1 PROTOCOLO DE ENCAPSULAMIENTO DE RED BNEP 47
2.1.1 Consideraciones 47
2.2 PICORED Y SCATTERNET. 49
2.3 PERFIL DE RED DE AREA PERSONAL (PAN PROFILE) 52
2.3.1 Consideraciones 53
2.3.2 Puntos de Acceso a una Red (NAP) 53
2.3.3 Grupo de Red ad-hoc 54
CAPITULO III 56
3 DISEÑO DE UN ALGORITMO PARA DESARROLLAR UNA APLICACIÓN SOBRE MÓDULOS BLUETOOTH PROGRAMABLES. 56
CAPITULO IV 59

4 DESCRIPCIÓN DE LOS EQUIPOS BLUETOOTH SELECCIONADOS PARA EL DESARROLLO DEL PROYECTO 59
4.1 PROCESO DE SELECCIÓN Y ADQUISICIÓN DE EQUIPOS BLUETOOTH. 59
4.2 MÓDULO SERIE INTELIGENTE BLUETOOTH DE TDK - BLU2I - 61
4.2.1 Introducción 61
4.2.2 Funcionamiento 61
4.2.3 Interfaz 63
1) Interfaz Serie: 63
2) Fuente de Poder: 64
3) Especificaciones Eléctricas de la Interfaz: 64
4.2.4 Rangos Máximos Absolutos 66
4.2.5 Alcance Del Blu^{2i} 66
4.3 ADAPTADOR USB GO BLUE DE TDK 67
4.4 TARJETA DE PRUEBAS O DESARROLLO PARA BLU ^{2I} DE TDK. 68
CAPITULO V 70
5 PROPUESTA PARA EL DESARROLLO DE UN SISTEMA DE SEGURIDAD QUE SUPERVISE LOS ACCESOS DE ENTRADA Y SALIDA A INSTALACIONES PRIVADAS HACIENDO USO DE LA TECNOLOGÍA BLUETOOTH. 70
5.1 SISTEMA DE SEGURIDAD Y VIGILANCIA FORMADO POR EQUIPOS DE SUPERVISIÓN REMOTA INALÁMBRICOS SOPORTADOS POR MÓDULOS BLUETOOTH. 71
5.2 DELIMITACIÓN DE LA PROPUESTA A DESARROLLAR. 73
CAPITULO VI 75
6 IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED BLUETOOTH QUE RESPONDA A LAS DEMANDAS DEL SISTEMA DE SEGURIDAD PROPUESTO. 75
6.1 ELABORACIÓN DE UNA INTERFAZ FÍSICA PARA LA COMUNICACIÓN ENTRE EL BLU ^{2I} Y EL PC (HARDWARE). 75
6.2 INTERFAZ PARA COMUNICAR LOS MÓDULOS BLU ^{2I} CON MICROCONTROLADORES PIC (HARDWARE). 77
6.3 ESTABLECIMIENTO DE LA COMUNICACIÓN INALÁMBRICA BLUETOOTH. 79
6.4 CONFIGURACION DE LOS BLU ^{2I} PARA LA IMPLEMENTACIÓN DE UNA PICORED. 84
6.5 ELABORACIÓN DE UNA INTERFAZ GRÁFICA PARA LA COMUNICACIÓN ENTRE EL USUARIO Y LOS MÓDULOS BLUETOOTH PARA CONTROLAR LA PICORED INALÁMBRICA. 86

Transferencia de archivos de texto: .txt, .doc, .xls..... 89
Transferencia de archivos de imágenes 92
6.6 DESARROLLO DE SOFTWARE PARA EL ESTABLECIMIENTO Y CONTROL DE LA PICORED BLUETOOTH. 95
6.6.1 Programa elaborado para los dispositivos esclavos. 96
6.6.2 Software desarrollado para el módulo maestro..... 96
6.7 IMPLEMENTACIÓN DE LA PICORED BLUETOOTH INTEGRANDO LOS DISPOSITIVOS DEL SISTEMA DE SEGURIDAD PROPUESTO..... 101
6.7.1 Dispositivo maestro incorporado al sistema de seguridad. 101
6.7.2 Dispositivos esclavos incorporados al sistema de seguridad. 102
6.8 PRUEBAS DE OPERATIVIDAD DEL SISTEMA COMPLETO, CONFORMADO POR UNA PICORED LOCAL BLUETOOTH. 103
6.8.1 Validación de la comunicación inalámbrica a través de la picored. 105
EVALUACIÓN DEL PROYECTO 107
VIABILIDAD TÉCNICA 107
VIABILIDAD LEGAL 108
VIABILIDAD ECONÓMICA 109
Valor Actual Neto (VAN): 109
Tasa Interna de Retorno (TIR): 111
Coeficiente Beneficio Costo (BC): 112
Periodo de Recuperacion (PR): 113
RECOMENDACIONES 114
CONCLUSIONES 115
REFERENCIAS BIBLIOGRÁFICAS 117

INDICE DE TABLAS

Tabla 1-1. Especificaciones de radio. 11
Tabla 1-2. Tipos de enlaces físicos Bluetooth. 21
Tabla 4-1. Módulo Serie Inteligente de TDK. 62
Tabla 4-2. Especificaciones de la interfaz eléctrica. 64
Tabla 4-3. Rangos máximos absolutos. 66
Tabla 5-1. Pruebas de transferencia de archivos de texto vía Bluetooth. 91
Tabla 5-2. Pruebas de transferencia de archivos de imágenes vía Bluetooth. 93
Tabla 7.1. Valor de desembolso inicial del proyecto 110
Tabla 7.2 Cálculo del Valor Actual Neto 111
Tabla 7.3 Tasa Interna de Retorno 112
Tabla 7.4 Coeficiente Beneficio Costo 112
Tabla 7.5 Período de Recuperación 113

INDICE DE FIGURAS

Figura 1-1 Conjunto de Protocolos Bluetooth. 11
Figura 1-2. Modelo de Referencia OSI y Bluetooth. 16
Figura 1- 3. Diagrama general “end to end” de las capas Bluetooth más bajas. 17
Figura 1-4. Diagrama de una Picored. 18
Figura 1-5. Conexión Punto a Punto (a), Picored (b) y Scatternet (c) 18
Figura 1-6. Transmisión de paquetes sencillos en una Picored. 20
Figura 1-7 Transmisión de paquetes multi-ranura en una Picored 21
Figura 1-8. Tráfico SCO y ACL. 22
Figura 1-9. Formato de Paquete General. 23
Figura 1-10. Estructura del código de acceso 23
Figura 1-11 Formato de Cabecera de Paquete. 25
Figura 1-12. Carga útil o Payload 26
Figura 1-13. Diagrama de transición de estados Bluetooth. 28
Figura 1-14. Iniciación de Comunicación sobre el nivel banda base 30
Figura 1-15 Dirección de dispositivo Bluetooth 31
Figura 1-16 Establecimiento de la Conexión Bluetooth. 33
Figura 1-17. Diagrama general de las capas más bajas 34
Figura 1-18. Establecimiento de canales en L2CAP. 36
Figura 1-19. Arquitectura L2CAP 37
Figura 1-20. Paquete L2CAP 38
Figura 1-21. Segmentación y reensamblado de paquete L2CAP. 39
Figura 1- 22. Puertos Serie emulados mediante RFCOMM. 42
Figura 1- 23. Perfiles Bluetooth 43
Figura 2-1. Ubicación del BNEP dentro del conjunto de Protocolos Bluetooth. 49
Figura 2-2. Esquema de una Picored. 50
Figura 2-3. Ejemplo de una Picored. 50
Figura 2-4. Esquema de una Scatternet. 51
Figura 2-5. Ejemplo de una Scatternet. 52
Figura 2-6. Dos puntos de Acceso a Red. 54
Figura 2-7. Conjunto de protocolos para el rol NAP. 54
Figura 2-8. Esquema para un Grupo de Red Ad-hoc (GN). 55
Figura 2-9. Protocolos para un enlace en una red Ad-hoc Bluetooth. 55
Figura 4-1. Módulo *Blu*^{2i} 61
Figura 4-2. Interfaz UART entre el *Blu*^{2i} y una aplicación. 64
Figura 4-3. Tasa de Transferencia de datos Vs distancia entre módulos. 67
Figura 4-4. Adaptador USB Go Blue. 68
Figura 4-5. Tarjeta de pruebas para *Blu*^{2i}. 69
Figura 5-1. Sistema de equipos de supervisión remotos incorporados al sistema ya implementado en la empresa. 72
Figura 5-2. Esquema del sistema de seguridad y vigilancia delimitado. 74

Figura 6-2. Esquema de la interfaz para comunicar el PC con el *Blu*^{2i} 77
Figura 6-4. Comunicación entre el computador y el *Blu*^{2i}. 81
Figura 6-6. Paquetes para envío de caracteres. 83
Figura 6-7. Menú principal de la interfaz gráfica elaborada. 87
Figura 6-8. Configuración del puerto COM. 88
Figura 6-9. Configuración del *Blu*^{2i}. 89
Figura 6-10. Selección y envío de archivos de texto. 90
Figura 6-11. Recepción de archivos de texto. 91
Figura 6-12. Selección y envío de archivos de imagen. 92
Figura 6-13. Recepción de archivos de imagen. 93
Figura 6-13. Programa que controla los dispositivos esclavos. 99
Figura 6-14. Diagrama de flujo del Software para el módulo maestro. 100
Figura 6-15. Sistema maestro integrado. 102
Figura 6-16. Dispositivos esclavos de supervisión remota 103
Figura 6-17. Plano vista superior de BCCOM Business con la picored implementada. 104

ACRÓNIMOS

ACL: (Asynchronous Connectionless). Asíncrono no orientado a la conexión.

ANSI: (**A**merican National Standards Institute). Instituto Nacional Americano de Estándares.

API: (Application Program Interface). Interfaz del Programa de Aplicación.

ARQUITECUTURA: intel, ARM (iPaq, ARM9 y otras) y PowerPC (iMac) entre otras.

AT (comando): (Attention). Atención. Es el prefijo de la línea de comando AT. Todos los comandos AT, exceptuando dos o tres, están precedidos por estos caracteres. (http://www.zoltrix.com/support_html/modem/USEMODEM.HTM).

ATM:(Asynchronous Transfer Mode). Modo de Transferencia Asíncrono. Dentro del medio informático hace referencia a la conmutación de paquetes (celdas o células) de un tamaño fijo con alta carga, rápida velocidad (entre 1,544 Mbps. y 1,2 Gbps) y una asignación dinámica de ancho de banda. También se conoce como "paquete veloz" (fast packet).

BD_ADDR: dirección del dispositivo Bluetooth.

BNEP: (Bluetooth Network Encapsulation Protocol). Protocolo de Encapsulación de Red Bluetooth.

BQP: (Bluetooth Qualification Program). Programa de Calificación Bluetooth.

CAD: (Computer Aided Design). Diseño Asistido por Computador.

CID: Identificador de Canal.

CRC: Chequeo de Redundancia Cíclica.

CTS: (Clear to send). Indica cuando el DCE está listo para transmitir y recibir datos.

DBI: sigla para “decibeles relativos a una fuente isotrópica”.

DCD: (Data Carrier Detect). Detector de transporte de datos. Señal generada por el DCE para indicar la presencia de una portadora de datos válida.

DCE: (Data Communications Equipment). Dispositivo que provee una trayectoria de comunicación entre dos equipos. Por ejemplo el módem en un computador.

DRIVER: controlador que permite gestionar los periféricos que están conectados al ordenador (http://www.guiahost.com/servicios/glosario).

DSR: (Data Set Ready). El equipo DCE está en operación.

DTE: (Data Terminal Equipment). Identifica al equipo terminal de datos (una PC) frente al equipo de comunicaciones DCE (el modem de datos)

DTR: (Data Terminal Ready). Indica que el DTE está operando.

ETHERNET: red de área local (LAN) desarrollada por Xerox, Digital e Intel. Es el método de acceso LAN que más se utiliza (seguido por Token Ring). Ethernet es una LAN de medios compartidos. Todos los mensajes se diseminan a todos los nodos en el segmento de red (http://www.guiahost.com/servicios/glosario).

ETSI: (European Telecommunications Standards Institute). Instituto de Estándares de Telecomunicaciones Europeo. Organización sin fines de lucro cuya misión es crear estándares de telecomunicaciones para ser usados por décadas en Europa (http://www.etsi.org).

FCC: Comisión Federal de Comunicaciones. Su principal función es la de mantener el control sobre el amplio sector de las telecomunicaciones en los Estados Unidos (http://www.guiahost.com/servicios/glosario).

FIFO: (First In First Out). El primero en entrar es el primero en salir.

FIRMWARE: parte del software de un ordenador que no puede modificarse por encontrarse en la ROM o memoria de sólo lectura, Read Only Memory. Es una mezcla o híbrido entre el hardware y el software, es decir tiene parte física y una parte de programación consistente en programas internos implementados en memorias no volátiles. Un ejemplo típico de Firmware lo constituye la BIOS (Basic Input Output System), que es un pequeño programa que coordina las actividades de los distintos componentes de un ordenador y comprueba su estado. Sistema básico de entrada y salida. (http://www.guiahost.com/servicios/glosario).

GAP: (Generic Access Profile). Perfil Genérico de Acceso.

GOEP: (Generic Object Exchange Profile). Perfil Genérico de Intercambio de Objetos).

GPL: (General Public License). Licencia Pública General, desarrollada por la FSF o Free Software Foundation. Puede ser instalado sin limitación en uno o varios ordenadores. En las distribuciones de estos programas debe estar incluido el código fuente.

HARDWARE: componentes electrónicos y electro-mecánicos de una computadora o cualquier otro sistema. Este término es usado para distinguir estos componentes físicos de los datos y programas.

HCI: (Host Controller Interface). Interfaz Controladora del Host o anfitrión.

HEC: chequeo de redundancia cíclica de encabezados.

HOST: sistema microprocesado programable (PCs, teléfonos celulares, mouse, impresoras, teclados, sensores inalámbricos, etc.), capaz de ejecutar las líneas de código correspondientes al stack de protocolos Bluetooth.

HW / FW: hardware / firmware.

I 2 C: interfaz de datos de dos líneas (SDA y SCL) que utiliza palabras de 8 bits y es usada para comunicar memorias, controladores de video, preamplificadores de audio y otros dispositivos.

IAC: Código de Acceso de Búsqueda.

ISM: (Industrial, Scientific, Medical). Industrial, Científica y Médica.

L2CAP: (Logical Link Controller and Adaptation Protocol). Protocolo de Adaptación y Control de Enlace Lógico.

LMP: (Link Manager Protocol). Protocolo del Administrador de Enlace.

ME: (Management Entity). Entidad de Administración o Manejo.

MODULO BLUETOOTH: módulo multichip que implementa en hardware y firmware las capas bajas del conjunto de protocolos Bluetooth.

MTU: (Maximum Transmission Unit). Unidad de Transmisión Máxima.

PAD: Punto de conexión del terminal de un dispositivo.

PAGING: servicio para transferencia de señalización o información en un sentido, mediante paquetes, tonos, etc.

PAN: (Personal Area Network). Red de Área Personal.

PATH: es la trayectoria de una pista en un circuito impreso.

PCB: (Printed Circuit Borrad). Tarjeta de Circuito Impreso.

PPP: (Point to Point Protocol). Protocolo Punto a Punto.

QoS: (Quality of Service). Calidad de Servicio.

RF: Radio Frecuencia.

RFCOMM: (Serial Port Emulation). Puerto serial Emulado, basado en el estándar ETSI TS07.10.

RTS: (Request to Send). Indica que el DTE está listo para transmisión y recepción.

SAR: Segmentación y Reensamblado.

SCO: (Synchronous Connection Oriented). Síncrono Orientado a la Conexión.

SDAP: (Service Discovery Aplication Protocol). Perfil de Aplicación de Descubrimiento de Servicio.

SDP: (Service Discovery Protocol). Protocolo de Descubrimiento de Servicio.

SIG: (Special Interest Group). Grupo de Interés Especial.

SOCKET: es una abstracción de red para los terminales de un canal. El Socket está asociado con el protocolo; usualmente, el PF_INET es usado para asociar un socket con el protocolo TCP/IP.

TIME OUT: tiempo de espera excedido.

TIME SLOT: ranura de tiempo. En Bluetooth tiene una duración de 625 us.

TOKEN RING: el anillo de Fichas (token ring), es una red de topología de anillo que se sirve del pase de fichas para el control de acceso. La frase también se aplica a una topología de pase de fichas específica definida por la IBM Corporation (http://www.guiahost.com/servicios/glosario).

TRANSCEIVER: transmisor-receptor.

UART: (Universal Asynchronous Receiver Transmitter). El transmisor receptor universal asíncrono. Es un dispositivo que multicanaliza datos paralelos en serie para ser transmitidos y convierte en paralelos los datos serie recibidos.

USB: (Universal Serial Bus). La característica principal del bus serie universal reside en que los periféricos pueden conectarse y desconectarse con el equipo en marcha, configurándose de forma automática. Conector externo que llega a transferencias de

12 millones de bits por segundo. Totalmente PnP, sustituirá al puerto serie y paralelo, gracias a la posibilidad de conectar 127 dispositivos. (http://www.guiahost.com/servicios/glosario).

WAP: (Wireless Application Protocol). Protocolo para Aplicaciones Inalámbricas.

INTRODUCCIÓN

El trabajo de grado que se presenta a continuación, se elabora con la finalidad de desarrollar un proyecto de telecomunicaciones basado en una tecnología inalámbrica que ha surgido recientemente en el mercado, denominada Bluetooth. Esta es una tecnología de corto alcance que funciona por medio de enlaces de radio frecuencia, a través de los cuales los dispositivos se interconectan entre sí para el intercambio de información. Una vez que se tienen dos o mas dispositivos Bluettoh interconectados y compartiendo el mismo canal de comunicación, se tiene una "Picored" Bluetooth establecida.

El proyecto que se desea realizar, consta de la implementación de una Picored Bluetooth, orientando su funcionamiento a una posterior incorporación dentro de un sistema de seguridad y vigilancia para instalaciones privadas.

Este documento se divide en seis (6) capítulos de desarrollo y adicionalmente se encuentra un capítulo de evaluación del proyecto. De los seis capítulos de desarrollo se tienen dos de marco teórico, un tercer (III) capítulo que plantea el marco metodológico a seguir para la culminación del proyecto y los tres capítulos restantes complementan el desarrollo del proyecto desde que se tienen los equipos Bluetooth en la mano hasta la implementación final.

Se necesita en un principio asentar unas buenas bases teóricas acerca de esta tecnología (capítulo I y II), de la cual se tiene poco conocimiento, para posteriormente adquirir módulos Bluetooth, realizar el estudio de su funcionamiento y realizar pruebas de comunicación real entre ellos (capítulo IV).

Posteriormente se propondrá un diseño para un sistema de seguridad que supervise los accesos de entrada y salida de una instalación privada (capítulo V), que

incluya entre sus equipos los dispositivos Bluetooth adquiridos y como parte final del proyecto se tratará de poner en marcha el sistema de seguridad integrado con los dispositivos Bluetooth y los requerimientos de Hardware y Software necesarios para que se cumplan las funciones propuestas en el diseño (capítulo VI).

Se espera entonces con este trabajo ampliar los conocimientos sobre una nueva tecnología dentro del área de las telecomunicaciones inalámbricas y lograr poner en funcionamiento los equipos Bluetooth de una manera determinada, realizando las investigaciones teóricas y técnicas necesarias para dar los primeros pasos hacia el desarrollo de un producto que pueda, en un futuro, ser expandido y comercializado por la empresa BCCOM Business, lugar donde se elaborará el proyecto y donde se realizarán todas las pruebas de comunicación y primera implementación de la Picored Bluetooth que se desarrolle.

PLANTEAMIENTO DEL PROBLEMA

En los últimos tiempos, ha surgido la necesidad de hacer uso de sistemas de seguridad y vigilancia para el resguardo de las propiedades privadas y de los bienes que en ella se encuentren. Esto se ve reflejado desde el personal de vigilancia que labora en casi cualquier infraestructura privada, bien sea de vivienda o de trabajo, hasta los sistemas de vigilancia y seguridad electrónica más complejos y sofisticados que se pueden ver en algunas grandes empresas. Es por esto que algún sistema de este tipo que sea innovador, sin importar su sencillez, puede tener alguna aplicación beneficiosa dentro de la sociedad.

En el campo de las telecomunicaciones, principalmente en la ingeniería, es importante mantenerse al día con las tecnologías que emergen continuamente, buscando usos y aplicaciones provechosas para los distintos usuarios que lo requieran.

Los dispositivos que incorporan la tecnología Bluetooth se reconocen entre ellos y se hablan de la misma forma que lo hace un ordenador con su impresora; el canal permanece abierto y no requiere la intervención directa y constante del usuario cada vez que se quiere enviar algo.

Luego, tomando en cuenta las ventajas que puede ofrecer una “Picored” inalámbrica Bluetooth, que no es mas que dos o mas dispositivos Bluetooth comunicándose a la vez, ha surgido la necesidad de ampliar su campo de aplicaciones, bien sean domésticas, industriales o de seguridad y vigilancia. Ésta última es la que se encuentra actualmente en pleno desarrollo, integrándose a sensores de seguridad, cámaras de vigilancia, equipos de monitoreo de variables físicas y demás equipos electrónicos utilizados en los sistemas de seguridad para propiedades privadas.

En la actualidad se ha hecho uso de redes locales inalámbricas dentro de sistemas de seguridad y vigilancia para instalaciones privadas, que han garantizado la eficiencia de estos sistemas, ya que es mucho más complejo interrumpir las conexiones entre dispositivos inalámbricos que entre aquellos conectados mediante cables. Mejorar los sistemas de seguridad propietaria es un deber. Los dispositivos tradicionales con capacidades sencillas de encendido-apagado están siendo reemplazados por aplicaciones inalámbricas que incorporan flujos de datos y muestras de cámara fotográfica y de video, lo que conlleva a una nueva era de seguridad y vigilancia inteligente.

Las aplicaciones de seguridad inalámbrica junto con la tecnología Bluetooth, pueden responder a condiciones predefinidas que remotamente bloqueen puertas, disparen alarmas, o envíen comunicación de voz, de datos o visual.

La empresa BCCOM C.A., un operador de servicios de valor agregado en el área de telecomunicaciones interesada en el ofrecimiento de servicios al consumidor, ha propuesto un proyecto que tiene como finalidad poner en funcionamiento una “picored” local inalámbrica apoyada en la tecnología Bluetooth, que pueda posteriormente ser integrada dentro de un sistema de seguridad para el monitoreo de accesos de entrada y salida de una propiedad privada.

BCCOM C.A. considera que la realización de este trabajo puede ofrecer una herramienta más para la implementación de redes de corto alcance destinadas a formar parte de sistemas de seguridad y vigilancia que requieran comunicación en línea. También sugiere que el mismo motivará la implementación de un laboratorio de aplicaciones inalámbricas basadas en la tecnología Bluetooth y además apoyará la toma eficiente de decisiones tecnológicas.

OBJETIVOS

OBJETIVO GENERAL

Desarrollar una aplicación de la tecnología “Bluetooth” implementando una picored local inalámbrica mediante módulos programables compatibles con dicha tecnología, para un sistema de seguridad y vigilancia que monitoree los accesos de entrada y salida a instalaciones privadas.

OBJETIVOS ESPECÍFICOS

- Diseñar un algoritmo para desarrollar una aplicación sobre módulos “Bluetooth” programables.

- Seleccionar y adquirir un mínimo de tres (3) módulos “Bluetooth” programables para implementar una picored inalámbrica.

- Elaborar una propuesta para desarrollar un sistema de seguridad que monitoree los accesos de entrada y salida a propiedades privadas, el cual va a estar constituido por equipos electrónicos soportados por los dispositivos “Bluetooth” programados.

- Implementar una picored inalámbrica, tal que responda a las demandas del sistema de seguridad propuesto.

- Validar la comunicación y efectuar ajustes finales para garantizar el cumplimiento de los requerimientos de aceptación y calidad de la aplicación.

- Evaluación de la operación y costos del proyecto.

CAPÍTULO I

1 DESCRIPCIÓN GENERAL DE LA TECNOLOGÍA INALÁMBRICA BLUETOOTH

1.1 RESEÑA HISTÓRICA

En 1.994, la compañía de telecomunicaciones ERICSSON, comenzó un estudio para investigar la viabilidad de una interfaz de radio de baja potencia y bajo costo entre teléfonos móviles y sus accesorios. El objetivo era eliminar los cables entre los teléfonos móviles y tarjetas de PCs, headsets, dispositivos desktop, etc. El estudio fue parte de otro gran proyecto de investigación que involucraba dispositivos de comunicación múltiple conectados a la red celular por medio de los teléfonos celulares. El ultimo enlace en dicha conexión debería ser un radio enlace de corto rango.

A medida que el proyecto progresaba, se volvió claro que las aplicaciones que envuelven dicho enlace de corto rango serían ilimitadas. A comienzos de 1997, Ericsson se aproxima a otros fabricantes de dispositivos portátiles para incrementar el interés en esta tecnología. El motivo era simple: para que el sistema fuera exitoso y verdaderamente utilizable, una cantidad crítica de dispositivos portátiles deberían utilizar la misma tecnología de radio enlaces de corto alcance. En Febrero de 1998, cinco compañías, Ericsson, Nokia, IBM, Toshiba e Intel, forman un Grupo de Interés Especial (SIG). Dicho grupo contiene la mezcla perfecta en lo que respecta al área de negocios, dos líderes del mercado en telefonía móvil, dos líderes del mercado en computadoras portátiles y un líder del mercado en tecnología de procesamiento de señales digitales [1]. La meta era establecer la creación de especificaciones globales para conectividad sin hilos de corto alcance.

La razón del nombre de esta tecnología, es debido a que en el siglo X el rey Harald II de Dinamarca, apodado "diente azul" (Bluetooth), a causa de una enfermedad que le daba esta coloración a su dentadura, reunificó bajo su reinado numerosos pequeños reinos que existían en Dinamarca y Noruega que funcionaban con reglas distintas, lo mismo que hace la tecnología Bluetooth, promovida por Ericsson (Suecia) y Nokia (Finlandia), dos países escandinavos. El 20 y el 21 de mayo de 1998, el consorcio de Bluetooth se anunció al público en Londres, San José-California y Tokio lo que provocó la adopción de la tecnología por varias compañías. El propósito del consorcio era establecer un dispositivo estándar y un software que lo controle.

Actualmente, más de 1.600 empresas ya pertenecen al SIG (Special Interest Group), los cuales han adoptado esta tecnología para desarrollarla con sus propios productos, que empezaron a salir al mercado a finales del año 2000. Cada nueva compañía miembro del SIG recibe de las otras una licencia para implantar la especificación 1.0 de Bluetooth.

1.2 BLUETOOTH

Bluetooth es un estándar empleado en enlaces de radio de corto alcance, destinado a reemplazar el cableado existente entre dispositivos electrónicos como teléfonos celulares, Asistentes Personales Digitales (o sus siglas en Inglés PDA), computadoras y muchos otros dispositivos, ya sea en el hogar, en la oficina, en el auto, etc. La tecnología empleada permite a los usuarios conexiones instantáneas de voz y datos entre varios dispositivos en tiempo real. El modo de transmisión empleado, asegura protección contra interferencias y seguridad en el envío de datos.

Entre sus principales características, pueden nombrarse su robustez, baja complejidad, bajo consumo y bajo costo. También es importante destacar que cada dispositivo Bluetooth se caracteriza por tener una dirección Bluetooth única, por

medio de la cual puede ser identificado por cualquier otro dispositivo que tenga esta tecnología y que se encuentre dentro de su alcance.

El radio Bluetooth es un pequeño microchip que opera en una banda de frecuencia disponible mundialmente. Pueden realizarse comunicaciones punto a punto y punto multipunto.

La tecnología inalámbrica Bluetooth ofrece una forma de remplazar cables y enlaces infrarrojos que interconectan dispositivos por un enlace de radio universal de corto alcance, con capacidad de crear pequeñas radio *LAN*s.

1.3 ¿CÓMO FUNCIONA BLUETOOTH?

Bluetooth es un sistema de radio basado en la transmisión de paquetes de datos, que opera en la banda de frecuencia libre de *2.4 GH*z. Esta banda de frecuencia está disponible en la mayor parte del mundo y es la llamada banda ISM la cual está destinada a equipos en el área industrial, científica y médica. En **Venezuela** la *Comisión Nacional de Telecomunicaciones* reglamenta su uso así: “De conformidad con el anexo 2 de la Recomendación de la Unión Internacional de Telecomunicaciones N° UIT-R 746, la porción del espectro radioeléctrico comprendida entre 2300 y 2480 MHz es utilizada para la operación de sistemas de microondas” [2]. Por otro lado, esta banda ISM está abierta a todo el mundo, por lo que es necesario que los sistemas de radio puedan comunicarse entre sí aun cuando existan otros dispositivos trabajando a la misma frecuencia, por ejemplo, monitores de bebe, abridores de puertas de garaje, teléfonos inalámbricos, hornos de microondas, etc.

Para superar esta dificultad se usa la técnica de Spread Spectrum o ensanchamiento del espectro, la cual no es más que una doble modulación de las señales transmitidas con saltos de frecuencia o FH (Frecuency Hopping). En la

modulación de espectro expandido la señal ocupa un ancho de banda mayor que el mínimo requerido para que los datos sean transmitidos. El ensanchamiento de la señal transmitida se logra con una modulación en la cual se utiliza una señal de frecuencia mucho mayor que la tasa de bits de los datos, llamada *spreading signal*.

Cuando esta modulación se realiza en la modalidad de saltos de frecuencia, la *spreading signal* es tal que su frecuencia cambia periódicamente; un código binario pseudoaleatorio controla la secuencia de frecuencias de la portadora. En la modalidad de secuencia directa, este código pseudoaleatorio se utiliza como *spreading signal*. En la recepción se utiliza una réplica de la *spreading signal* empleada en la transmisión para demodular la señal de datos deseada.

Luego de mencionar la Spreading signal o señal de ensanchamiento, hay que hacer notar que la técnica de Spread Spectrum tiene dos modulaciones. La segunda modulación es la denominada G-FSK o modulación adaptada de Gauss por codificación de cambio de frecuencias (Gaussian frequency shift keying), donde un "1" binario representa una desviación de frecuencia positiva, y un "0" binario representa una desviación de frecuencia negativa. La desviación máxima de frecuencia está entre 140 KHz y 175 KHz. Gracias a este tipo de modulación la velocidad puede alcanzar magnitudes de hasta 1 Mbps, claro está que ello depende del ancho de banda de cada canal.

La técnica de FH o salto de frecuencia utilizada por la tecnología *Bluetooth*, divide el ancho de banda total en 79 canales de radio frecuencia de 1 MHz de ancho de banda cada uno, por lo que pueden llegar a alcanzar la velocidad antes mencionada. Después de que cada paquete es enviado en una determinada frecuencia de transmisión, ésta cambia a otra de las 79 frecuencias [3].

La unidad básica de red en Bluetooth es la picored, la cual consiste de un dispositivo maestro y de uno a siete dispositivos esclavos activos, donde el

dispositivo Bluetooth que inicia la conexión representa al maestro. El maestro determina también el canal (secuencia de Frecuency Hopping) y la fase.

Los radios Bluetooth se comunican usando TDD (Time Division Duplex), el cual transmite datos en una sola dirección alternando entre las dos direcciones. Debido a que dos o más dispositivos comparten el medio de una picored Bluetooth, la técnica de acceso es TDMA, por lo tanto el acceso a una picored Bluetooth puede caracterizarse como FH-TDD-TDMA.

El rango típico de operación de *Bluetooth* va a depender de la clasificación de los emisores según el nivel de potencia de salida que estos posean. Tal clasificación de los equipos Bluetooth se divide en tres clases como sigue:

- **Clase 1**: Diseñados para dispositivos de largo alcance (hasta 100 m). Potencia de salida máxima de 20 dBm.
- **Clase 2**: Diseñados para dispositivos ordinarios (hasta10m). Potencia de salida máxima de 4 dBm.
- **Clase 3**: Diseñados para dispositivos de corto alcance (hasta10cm). Potencia de salida máxima de 0 dBm.

Como se puede observar en la **Figura 1-1**, la comunicación sobre *Bluetooth* se divide en varias capas. A continuación se presenta una breve descripción de algunas de ellas y se tratarán con mayor detalle posteriormente.

La capa mas baja es la de **Radio Bluetooth**, la cual define la frecuencia, modulación y potencia de transmisión. Estas especificaciones de radio se muestran en la **tabla 1-1.**

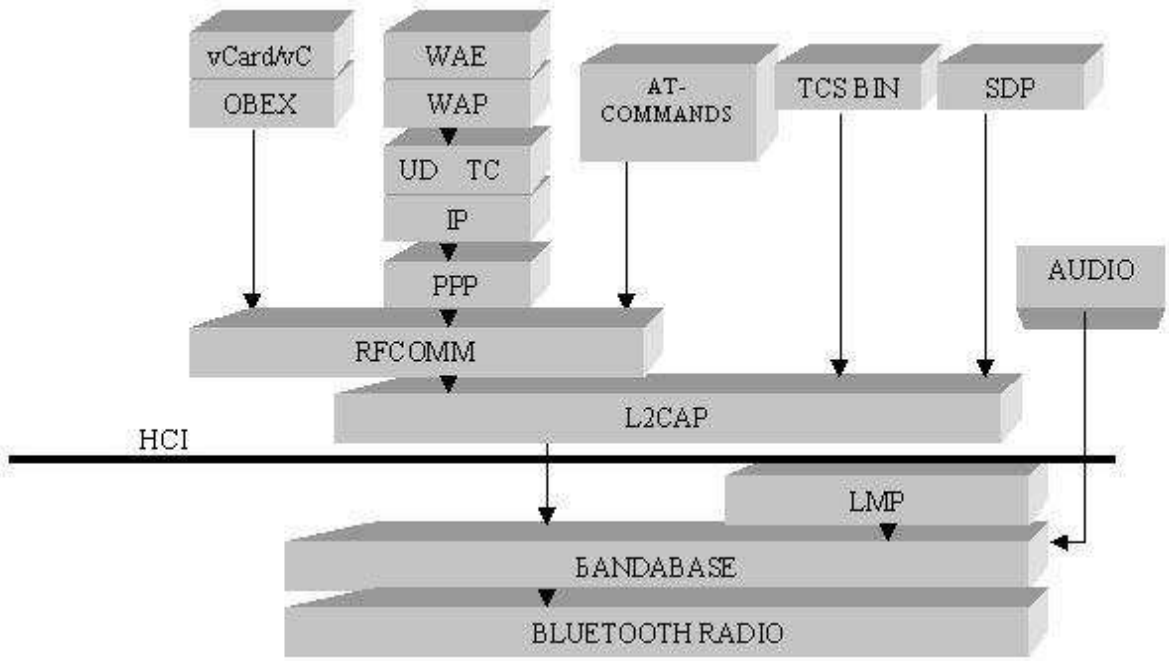

Figura 1-1 Conjunto de Protocolos Bluetooth.

Tabla 1-1. Especificaciones de radio.

Topología	Hasta 7 enlaces en estrella
Modulación	GFSK
Velocidad Pico	1 Mbps
Ancho de Banda RF	1 MHz
Banda RF	2.4 GHz, Banda ISM
Potencia TX	100 mW, 2.5 mW, 1mW
Acceso Picores	FH-TDD-TDMA
Razón de FH	1600 saltos/s

La segunda capa de comunicación es llamada **banda base**. Esta capa implementa el canal físico real. Emplea una secuencia aleatoria de saltos a través de 79 frecuencias de radio diferentes. Los paquetes son enviados sobre el canal físico,

con un salto frecuencia diferente. La *Banda Base* controla la sincronización de las unidades *Bluetooth* y la secuencia de saltos en frecuencia, además es la responsable de la información para el control de enlace a bajo nivel como el reconocimiento, control de flujo y caracterización de carga útil y soporta dos tipos de enlace: *síncrono orientado a la conexión* **(SCO**), principalmente para voz y *asíncrono no orientado a la conexión* **(ACL**), principalmente para datos. Los dos pueden ser multiplexados para usar el mismo enlace *R*F. Usando ancho de banda reservado, los enlaces *SCO* soportan tráfico de voz en tiempo real.

Link Manager Protocol (LMP) o *Protocolo de Gestión de Enlace* es el responsable de la autenticación, encriptación, control y configuración del enlace. El *LMP* también se encarga del manejo de los modos y consumo de potencia, además soporta los procedimientos necesarios para establecer un enlace *SC*O.

Host Controller Interface (HCI) o *Interfaz del Controlador de Enlace* brinda un método de interfaz uniforme para acceder a los recursos de hardware de *Bluetooth*. Éste contiene comandos para el *controlador banda base* y la *gestión de enlace* y para acceder al hardware.

Logical Link Control and Adaptation Protocol (L2CAP) o *Protocolo de Control y Adaptación de Enlace Lógic*o, corresponde a la capa de enlace de datos. Éste protocolo proporciona servicios de datos orientados y no orientados a la conexión a los protocolos de las capas superiores, junto con facilidades de multicanalización y de segmentación y reensamblaje. L2CAP permite que los protocolos de capas superiores puedan transmitir y recibir paquetes de datos L2CAP de hasta 64 kilobytes de longitud.

L2CAP se basa en el concepto de *canales*. Un canal es una conexión lógica que se sitúa sobre la conexión de banda base. Cada canal se asocia a un único

protocolo. Cada paquete L2CAP que se recibe en un canal se redirige al protocolo superior correspondiente. Varios canales pueden operar sobre la misma conexión de banda base, pero un canal no puede tener asociados más de un protocolo de alto nivel.

Con el fin de manipular paquetes de capas superiores más grandes que el máximo tamaño del paquete de banda base, *L2CAP* los segmenta en varios paquetes banda base. La capa *L2CAP* del receptor reensambla los paquetes de banda base en paquetes más grandes para la capa superior. La conexión L2CAP permite el intercambio de información referente a la calidad de la conexión, además maneja grupos, de tal manera que varios dispositivos pueden comunicarse entre sí.

Sevice Discovery Protocol (SDP) o *Protocolo de Descubrimiento de servicio* define cómo actúa una aplicación de un cliente *Bluetooth* para descubrir servicios disponibles de servidores *Bluetooth*, además de proporcionar un método para determinar las características de dichos servicios.

El protocolo **RFCOMM** ofrece emulación de puertos serie sobre el protocolo *L2CAP*. *RFCOMM* emula señales de control y datos *RS-232* sobre la banda base *Bluetooth.* Éste ofrece capacidades de transporte para servicios de capas superiores (por ejemplo *OBE*X, que se explica un poco más adelante) que usan una línea serie como mecanismo de transporte. *RFCOMM* soporta dos tipos de comunicación, directa entre dispositivos actuando como **puntos finales** y **dispositivo-modem-dispositivo**, además tiene un esquema para emulación de **null modem[1].**

El **control de telefonía binario (TCS binario)** es un protocolo que define la señalización de control de llamadas, para el establecimiento y liberación de una

[1] Null modem es básicamente un cable para interconectar dos puertos serie. Normalmente se necesitan dos computadoras o dos puertos físicos en una computadora para lograr esto. es.tldp.org/COMO-INSFLUG/es/pdf/Terminales-Como.pdf

conversación o una llamada de datos entre unidades *Bluetooth.* Además, éste ofrece funcionalidad para intercambiar información de señalización no relacionada con el progreso de llamadas.

La capa de **Audio** es una capa especial, usada sólo para enviar audio sobre *Bluetooth.* Las transmisiones de audio pueden ser ejecutadas entre una o más unidades usando muchos modelos diferentes. Los datos de audio no pasan a través de la capa *L2CAP*, sino directamente a la capa banda base, luego de abrir un enlace y establecer la comunicación entre dos unidades *Bluetooth.*

- **Protocolos Específicos**

- **Control de telefonía – Comandos AT (Atenttion).** *Bluetooth* soporta un número de comandos *AT* para el control de telefonía a través de emulación de puerto serie *(RFCOM*M).

- **Protocolo Punto-a-Punto (PPP).** El *PPP* es un protocolo orientado a paquetes y por lo tanto debe usar su mecanismo serie para convertir un torrente de paquetes de datos en una corriente de datos serie. Este protocolo corre sobre RFCOMM para lograr las conexiones punto-a-punto.

- **Protocolos UDP/TCP – IP.** Los estándares *UDP/TCP e IP* permiten a las unidades *Bluetooth* conectarse, por ejemplo a *Internet*, a través de otras unidades conectadas. Por lo tanto, la unidad puede actuar como un puente para *Interne*t. La configuración *TCP / IP / PPP* está disponible como un transporte para *WA*P.

- **Wireless Aplication Protocol (WAP)** o *Protocolo de Aplicación Inalámbrica.* Es una especificación de protocolo inalámbrica que trabaja con una amplia variedad de tecnologías de redes inalámbricas conectando

dispositivos móviles a *Internet*. *Bluetooth* puede ser usado como portador para ofrecer el transporte de datos entre el cliente *WAP* y su servidor de *WAP* adyacente. Además, las capacidades de red *Bluetooth*, dan a un cliente *WAP* posibilidades únicas, en cuanto a movilidad, comparadas con otros portadores *WAP*. Un ejemplo de *WAP sobre Bluetooth* sería un almacén que transmite ofertas especiales a un cliente *WAP* cuando éste entra en el rango de cobertura.

- **Protocolo OBEX (Object Exchange).** Es un protocolo opcional de nivel de aplicación diseñado para permitir a las unidades *Bluetooth* soportar comunicación infrarroja para intercambiar una gran variedad de datos y comandos. Éste usa un modelo *cliente-servidor* y es independiente del mecanismo de transporte y del *API* **(Application Program Interface)** de transporte. *OBEX* usa *RFCOMM* como principal capa de transporte.

La **Figura 1-2** muestra una comparación del conjunto *Bluetooth* con el modelo de referencia estándar **Open Systems Interconect, OSI,** para pilas de protocolos de comunicaciones. A pesar de que *Bluetooth* **no concuerda** exactamente con el modelo, esta comparación es muy útil para relacionar las diferentes partes del conjunto *Bluetooth* con las capas del modelo *OSI.* Dado que el modelo de referencia es un conjunto ideal y bien particionado, la comparación sirve para resaltar la división de funciones en el conjunto de protocolos *Bluetooth.*

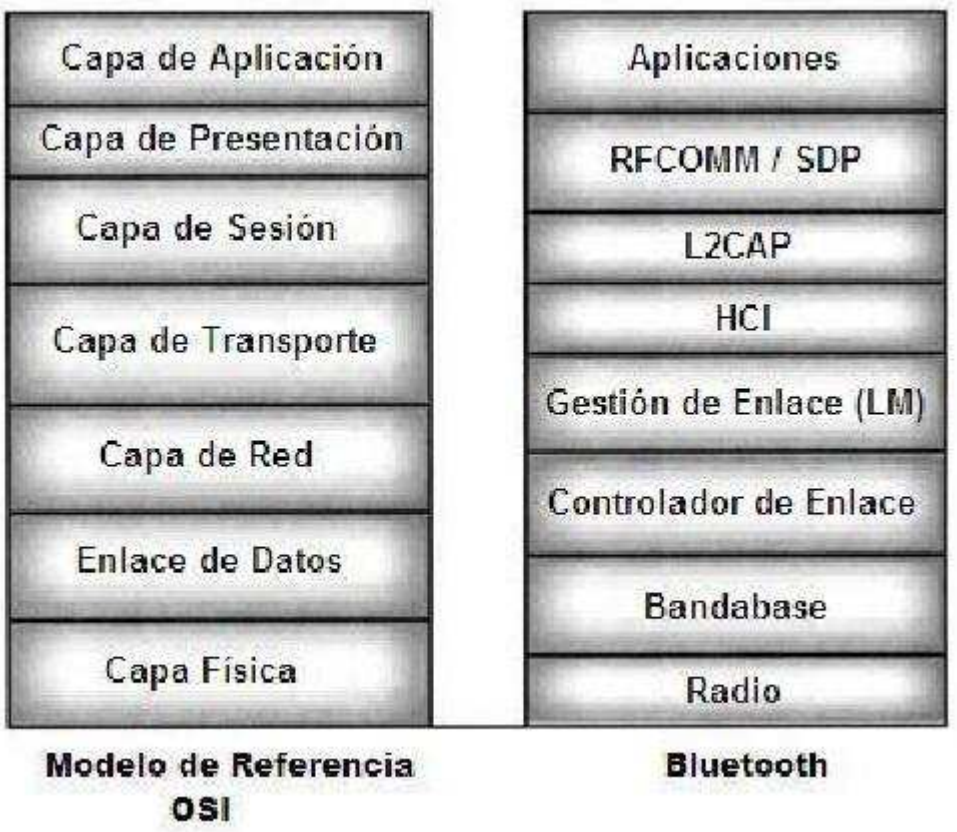

Figura 1-2. Modelo de Referencia OSI y Bluetooth.

La **Figura 1-3** muestra el recorrido de un dato en las capas Bluetooth, para poder ser transferido de un dispositivo a otro. Allí se muestra de manera general como funciona el conjunto de capas Bluetooth haciendo énfasis en las capas más bajas de comunicación, de modo que describe las zonas de Software, Hardware y Firmware que hacen posible el establecimiento de la comunicación Bluetooth. De esta manera se puede tener una idea de todos los pasos que se deben tomar en cuenta para el desarrollo de una aplicación que tenga como finalidad comunicar dos o más Host vía Bluetooth.

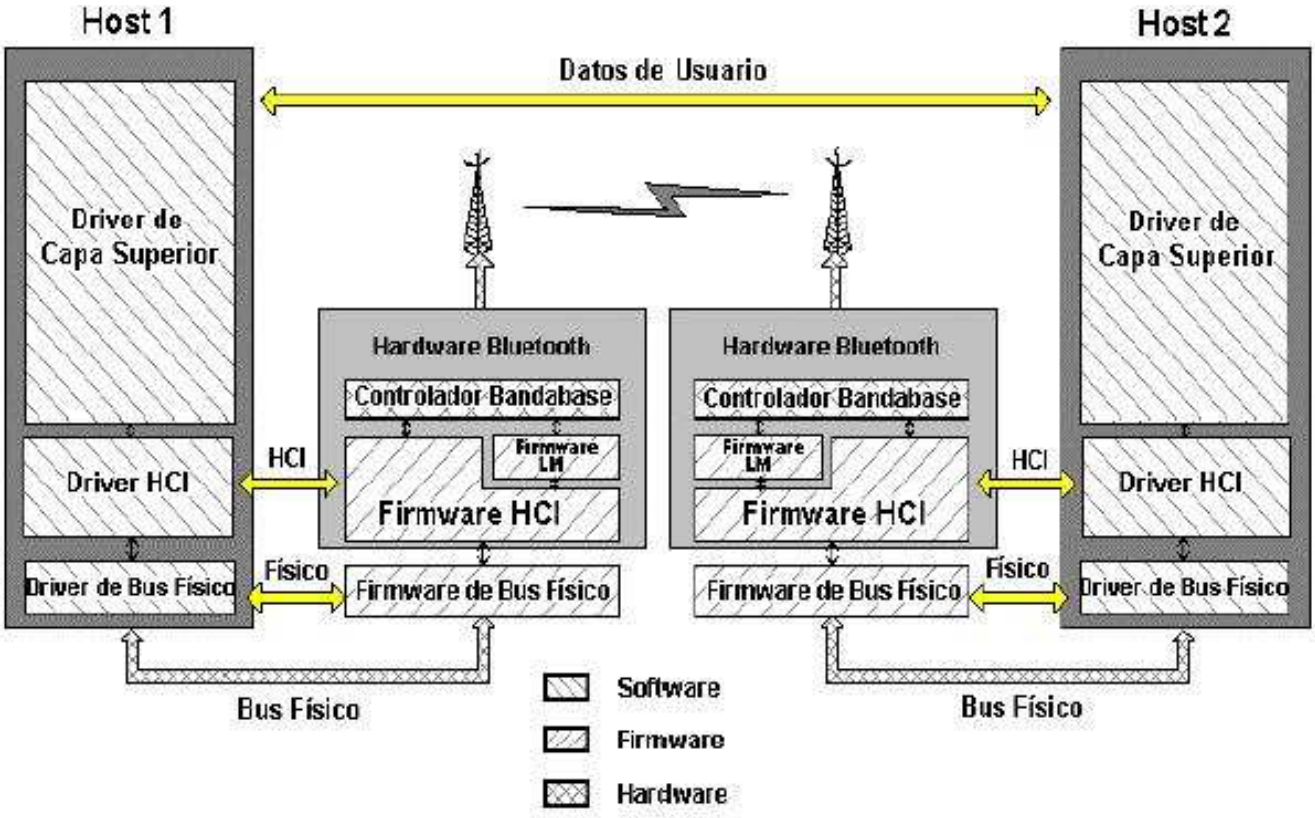

Figura 1- 3. Diagrama general "end to end" de las capas Bluetooth más bajas.

1.3.1 BANDA BASE

1) Descripción general. *Bluetooth* soporta un canal de datos asíncrono de hasta tres canales de voz simultáneos. El canal asíncrono soporta comunicación simétrica y asimétrica. En la comunicación asimétrica pueden ser enviados 723.3 kb/s desde el servidor y 57.6 kb/s hacia el servidor, mientras que en la comunicación simétrica pueden ser enviados 433 kb/s en ambas direcciones. *Bluetooth* brinda conexión *punto-a-punto* o conexión *punto-a-multipunto*. Dos o más unidades compartiendo el mismo canal forman una *picored.* Cada *picored* debe tener un maestro y puede tener hasta siete esclavos activos, además pueden haber muchos más esclavos en estado **parked**. En la **Figura 1-4** se puede observar una *picored* donde el PC actúa como maestro y los otros dispositivos son conectados como esclavos. Los esclavos en estado **parked,** no están activos en el canal sin embargo están

sincronizados con el maestro con el fin de asegurar una rápida iniciación de comunicación.

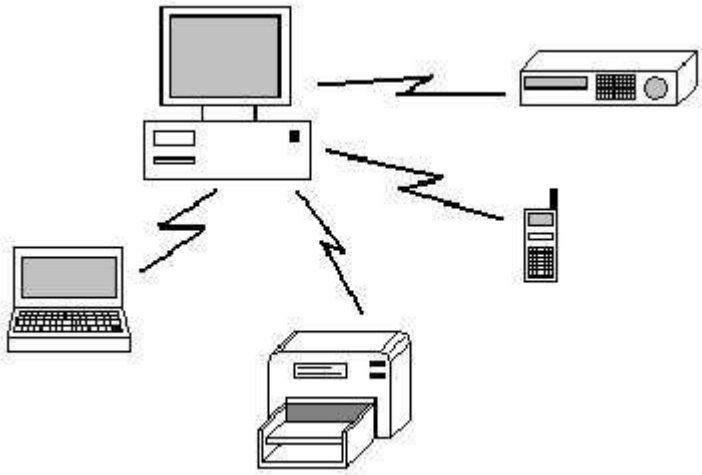

Figura 1-4. Diagrama de una Picored.

La interconexión de varias *picoredes* forma una *scatternet*, con un máximo de diez *picoredes* interconectadas [3].

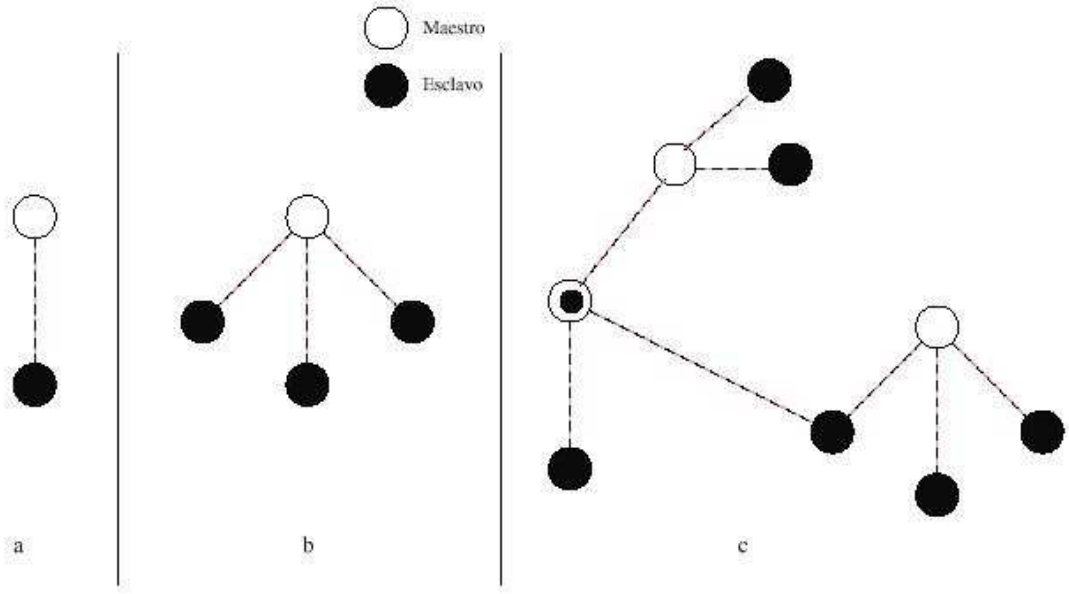

Figura 1-5. Conexión Punto a Punto (a), Picored (b) y Scatternet (c)

En la **Figura 1-5a** se puede observar la Picored más sencilla la cual está constituida por dos dispositivos, mientras que en la **Figura 1-5b** se tiene una Picored

constituida por cuatro de estos dispositivos. En la **Figura 1-5c** está ejemplificada una Scatternet que posee tres Picoredes, una constituida por cuatro unidades, otra por dos y la última por 3 unidades respectivamente.

Los equipos que comparten un mismo canal se dividirán los recursos y la capacidad de éste. Aunque los canales tienen un ancho de banda de un 1Mhz, cuantos más usuarios se incorporen a la Picored, menores serán los recursos adjudicados a cada usuario, es decir, disminuirá la capacidad del ancho de banda de cada dispositivo hasta unos 10 kbit/s o un poco menos.

La Scatternet, fue introducida para solucionar el problema del bajo ancho de banda que le queda a cada usuario de una Picored, si en esta se encuentra gran cantidad de unidades conectadas. El rendimiento, en conjunto e individualmente de los usuarios de una Scatternet es mayor que el que tiene cada usuario cuando participa en un mismo canal de 1 MHz. Además, estadísticamente se obtienen ganancias por multiplexación y rechazo de canales de salto. Debido a que individualmente cada Picored tiene un salto de frecuencia diferente, diferentes Picoredes pueden usar simultáneamente diferentes canales de salto.

2) Canal físico. El canal físico utiliza 79 portadoras de radio diferentes, las cuales son accedidas de acuerdo a una secuencia de saltos aleatoria. La rata de saltos estándar es de *1600 saltos/s.* El canal está dividido en **timeslots** (ranuras de tiempo). Cada ranura corresponde a una frecuencia de salto y tiene una longitud de 625us. Cada secuencia de salto en una *picored* está determinada por la dirección Bluetooth del maestro de la *picored.* Todos los dispositivos conectados a la *picored* están sincronizados con el canal en salto y tiempo. En una transmisión, cada paquete debe estar alineado con el inicio de una ranura. Un paquete ocupa normalmente una única ranura (**figura 1-6**), pero puede ocupar hasta cinco consecutivas (en la **figura 1-7** se muestra el caso de un paquete transmitido en tres ranuras consecutivas)**.** Durante la transmisión de un paquete la frecuencia es fija. Para evitar fallas en la transmisión, el

maestro envía paquetes en las ranuras de tiempo pares y los esclavos en las ranuras de tiempo impares [3].

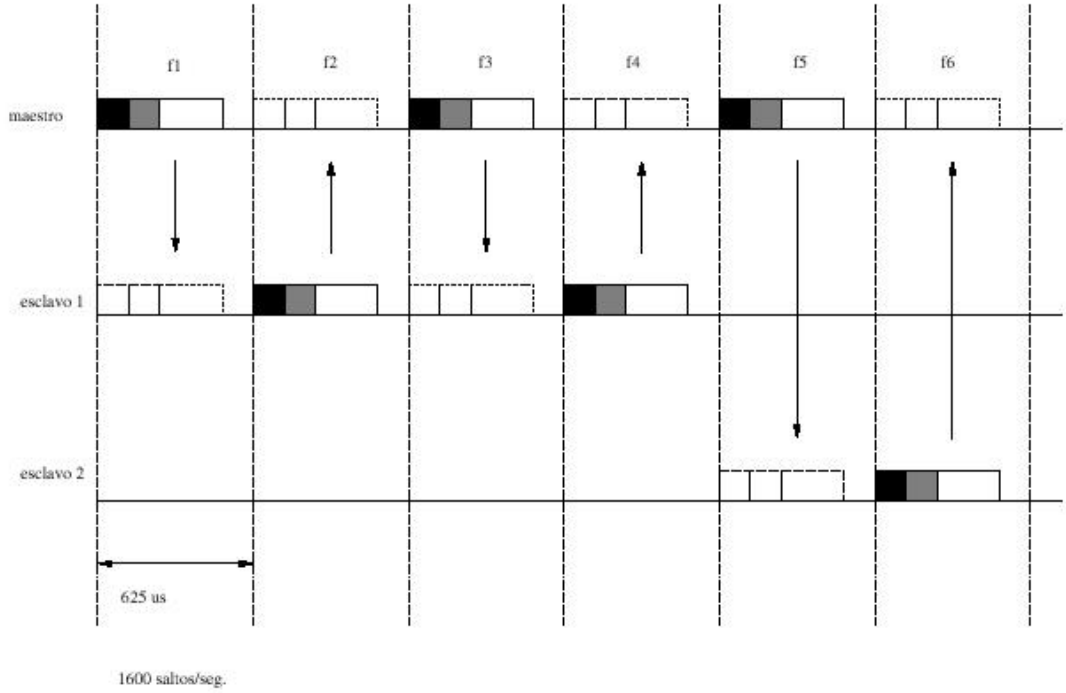

Figura 1-6. Transmisión de paquetes sencillos en una Picored.

3) Enlace físico. La comunicación sobre *Bluetooth* se ha optimizado para enlaces *SCO* o enlaces *AC*L. El enlace *SCO* es una conexión simétrica *punto-a-punto* entre el maestro y un esclavo específico. Para lograr la comunicación, el enlace *SCO* reserva **ranuras** a intervalos regulares en la iniciación, por esto el enlace puede ser considerado como una conexión de conmutación de circuitos. El enlace *ACL* es un enlace *punto- multipunto* entre el maestro y uno o más esclavos activos en la *picored*. Este enlace de comunicación es un tipo de conexión de conmutación de paquetes. Todos los paquetes son retransmitidos para asegurar la integridad de los datos. El maestro puede enviar mensajes de difusión (**broadcast**) a todos los esclavos conectados dejando vacía la dirección del paquete, así todos los esclavos leerán el paquete [3].

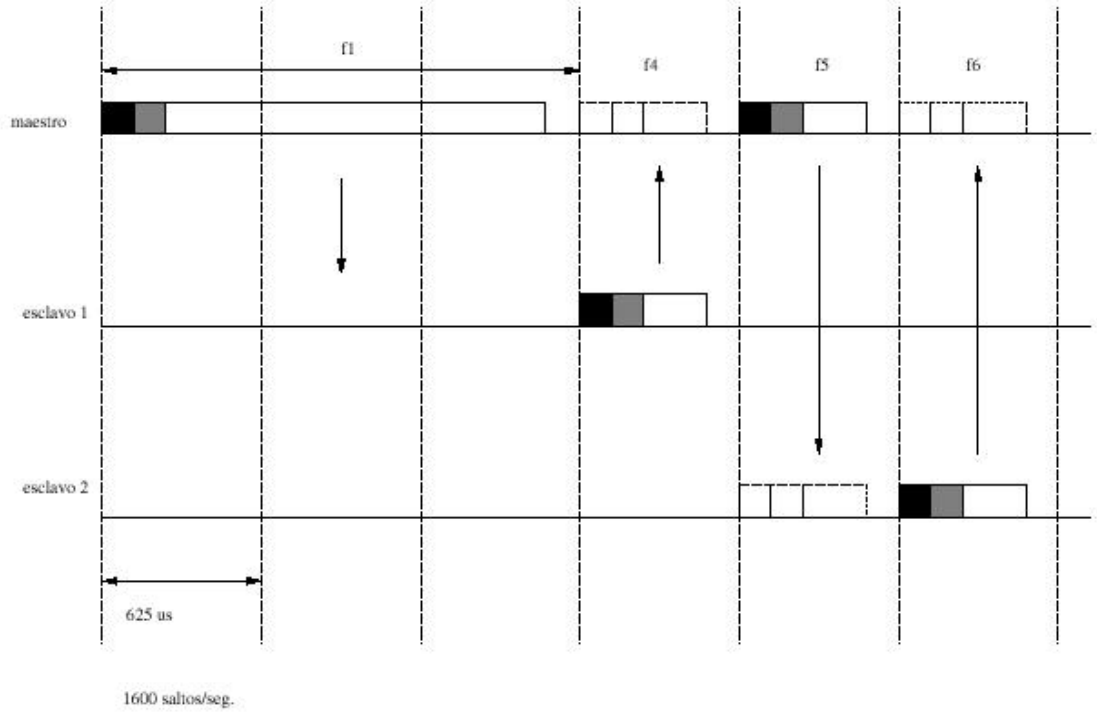

Figura 1-7 Transmisión de paquetes multi-ranura en una Picored

Existen dos tipos de enlaces físicos entre maestros y esclavos como se muestra en la **tabla 1-2**:

Tabla 1-2. Tipos de enlaces físicos Bluetooth.

	**Enlace SCO (Synchronous Connection-Oriented):** El enlace SCO es una conexión simétrica punto-a-punto con un ancho de banda fijo entre el maestro y un esclavo específico (El maestro soporta 3 conexiones SCO con el mismo o diferentes esclavos).Para lograr la comunicación, el enlace SCO reserva **ranuras de tiempo** en intervalos regulares en la iniciación, por esto el enlace puede ser considerado como una conexión de conmutación de circuitos. En este tipo de enlace no es necesario asegurar la entrega y suele ser utilizado para comunicaciones de voz.

	**Enlace ACL (Asynchronous Connection-Less):** El enlace ACL es una conexión simétrica o asimétrica punto-a-multipunto entre el maestro y uno o más esclavos activos en la picored sin reserva de ancho de banda. Este enlace de comunicación es un tipo de conexión de conmutación de paquetes. Aquí, a diferencia del anterior, se necesita asegurar la entrega de datos y es utilizado para transferencia de datos sin requerimientos temporales.

En la **figura 1-8** se da un ejemplo de tráfico SCO y ACL en una picored:

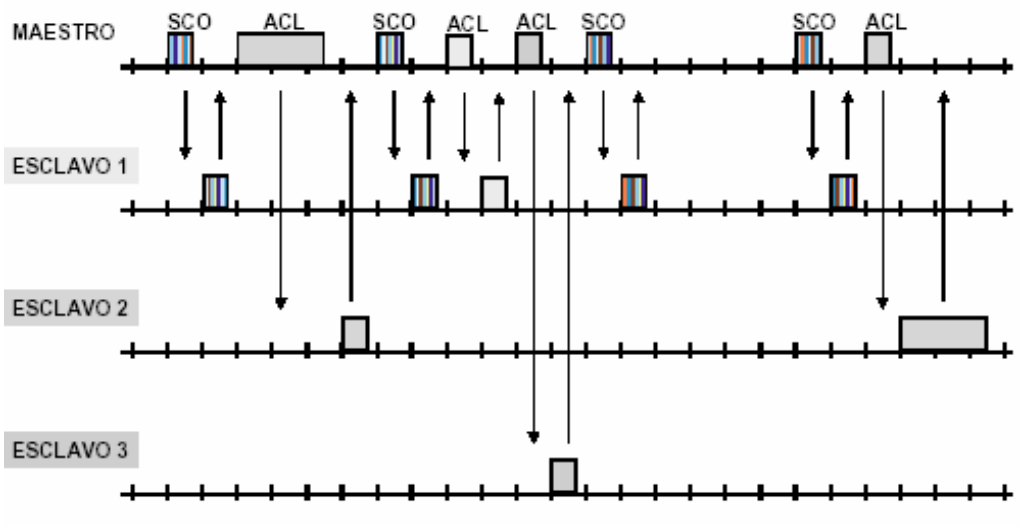

Figura 1-8. Tráfico SCO y ACL.

4) Paquetes. Los datos enviados sobre el canal de la *picored* son convertidos en paquetes, éstos son enviados y el receptor los recibe iniciando por el *bit* menos significativo. Como se observa en la **Figura 1-9**, el formato de paquete general consta de tres campos: *código de acceso, cabecera y carga útil* [3].

Figura 1-9. Formato de Paquete General.

· **Código de acceso.** Es usado para sincronización e identificación. Todos los paquetes comunes que son enviados sobre el canal de la *picored* están precedidos del mismo código de acceso al canal. En la **figura 1-10** se ilustra como esta conformado el código de acceso.

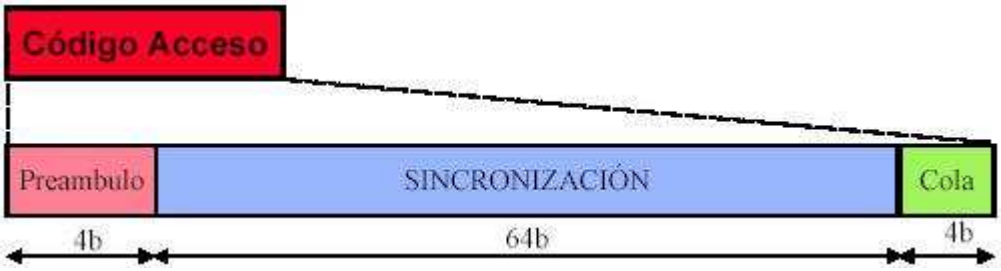

Figura 1-10. Estructura del código de acceso

Existen tres tipos diferentes de código de acceso:

- **Código de acceso al canal (CAC)-** Para identificar los paquetes que tienen acceso al canal de la *picored.*

- **Código de acceso de dispositivo (DAC)** – Usado para identificación de unidades Bluetooth y está definido únicamente por la dirección Bluetooth del mismo. Se usa en procedimientos de **Paging** o sincronización de equipos.

- **Código de Acceso de Búsqueda *(IAC)*** – llamado *IAC general* cuando se quiere descubrir a otras unidades *Bluetooth* dentro del rango, o *IAC dedicado* cuando se desea descubrir unidades de un tipo específico.

El Código de Acceso consta de 72 bits que se dividen en 4 bits de **preámbulo**, encargados de la identificación del código, 64 bits utilizados para la **sincronización** del código y 4 bits de **cola**, los cuales compensan el tamaño del código en caso de ser necesario.

· **Cabecera de paquete.** Como se observa en la **Figura 1-11**, la *cabecera de paquete* consta de seis campos:

- **Dirección o AM_ADDR**– Dirección temporal de 3 bits que se utiliza para distinguir los dispositivos activos en una *picored.*

- **Tipo** – Define qué tipo de paquete es enviado (comandos, respuestas, eventos o datos) y cuantas ranuras de tiempo va a ocupar.

- **Flujo** – El *bit* de control de flujo es usado para notificar al emisor cuándo el buffer del receptor está lleno y por lo tanto se debe dejar de transmitir, en este caso el bit tendrá el valor “0”.

- **ARQN** – **Acknowledge Receive Data** o bit de reconocimiento de datos recibidos.

- **SEQN** – **Sequential Numbering** o numeración secuencial para ordenar los paquetes en el canal.

- **HEC** – Chequeo de redundancia cíclica de cabecera.

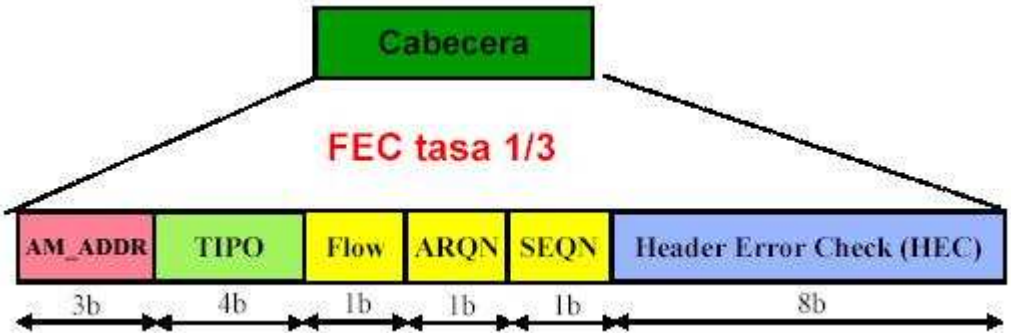

Figura 1-11 Formato de Cabecera de Paquete.

· **Carga útil (Payload).** La carga útil de un paquete puede ser dividida en dos campos:

- **Campo de Voz** – Consta de datos de voz de longitud fija y existe en paquetes de alta calidad de voz y paquetes combinados de datos-voz. No es necesaria ninguna cabecera de carga útil.

- **Campo de Datos** – Consta de tres partes, cabecera de carga útil, datos de carga útil, y código *CRC.*

  La cabecera de carga útil, usada en el campo de datos, consta a su vez de tres partes definidas y una no definida, 3 bits de **L_CH** que define el canal por el que se envían los datos, el cual varía dependiendo del esclavo con el que se está comunicando. El **Flow** indica si el paquete está saliendo o llegando al dispositivo (Flow in, Flow out). **Lengnth** indica la longitud o tamaño del paquete que se envía o que se recibe.

En la **figura 1-12** se muestran los dos campos que forman la carga útil, en la parte superior el campo de datos e inferior a éste, el campo de voz.

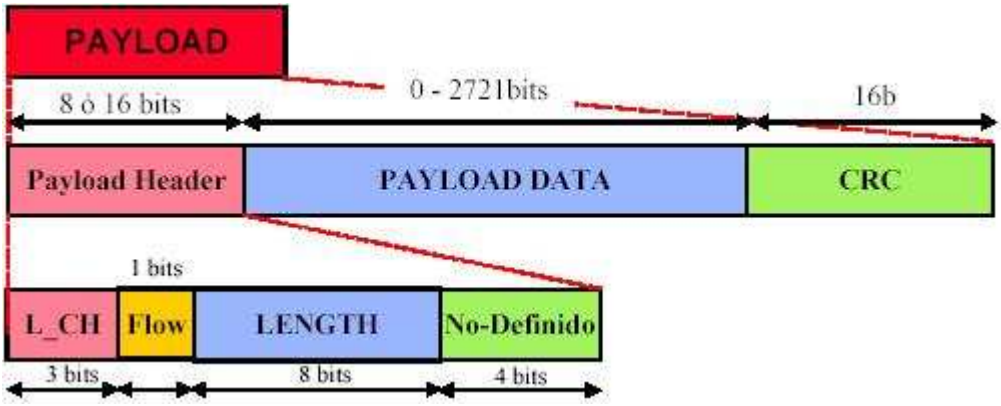

Figura 1-12. Carga útil o Payload

5) Corrección de errores. En una comunicación *Bluetooth* existen varios esquemas diferentes de corrección de errores [3]:

· En la cabecera, cada *bit* es repetido tres veces.

· En la carga útil se usa un esquema de *código Hamming*[2]. Los *bits* de información son agrupados en secuencias de 10 *bits*, éstos son enviados como 15 *bits* y el algoritmo corrige todos los errores en un *bit* y detecta los errores en dos *bits.*

· Para garantizar una recepción correcta, todos los paquetes de datos son retransmitidos hasta que el emisor reciba una confirmación. La confirmación es enviada en la cabecera de los paquetes retornados. Los paquetes **broadcast** son paquetes transmitidos desde el maestro a todos los esclavos. No hay posibilidad de usar confirmación para esta comunicación, sin embargo, para incrementar la posibilidad de recibir correctamente un paquete, cada *bit* en el paquete es repetido un número fijo de veces.

· El chequeo de redundancia cíclica *(CRC*) se usa para detectar errores en la cabecera.

[2] El Código Hamming es un código binario corrector de errores que consiste en añadir varios bits de paridad intercalados entre los bits de datos.

La suma de comprobación *CRC* está contenida en el campo *HEC* de la cabecera del paquete. Los chequeos de redundancia cíclica también se aplican sobre la carga útil en la mayoría de los paquetes.

· Para asegurar que no desaparezcan paquetes completos, *Bluetooth* usa números de secuencia. Actualmente sólo se usa un número de secuencia de un *bit*.

6) Transmisión/Recepción. Como se mencionó antes, el maestro de la *picored* empieza enviando en ranuras de tiempo (**timeslots)** pares y el esclavo en los impares. Cada esclavo espera las oportunidades de conexión dadas por el maestro. Los paquetes pueden ser más grandes que una ranura de tiempo**,** debido a esto el maestro puede continuar enviando en las cuatro ranuras siguientes a la ranura en la cual se comenzó el envío, igualmente, si un esclavo envía un paquete mayor a una ranura de tiempo, seguirá enviando en las cuatro ranuras de tiempo subsecuentes. El sistema de reloj del maestro sincroniza a toda la *picored.* El maestro nunca ajusta su sistema de reloj durante la existencia de una *picored,* son los esclavos quienes adaptan sus relojes con un *offset* de tiempo con el fin de igualarse con el reloj del maestro. Este *offset* es actualizado cada vez que es recibido un paquete desde el maestro.

7) Control de Canal. El control de canal describe cómo se establece el canal de una *picored* y cómo las unidades pueden ser adicionadas o liberadas en la *picored.* La dirección Bluetooth del maestro determina la secuencia de saltos y el código de acceso al canal. La *fase* de la *picored* está determinada por el sistema de reloj del maestro. Por definición, la unidad *Bluetooth* que inicia la conexión representa al maestro.

En *Bluetoot*h, la capa de control de enlace se divide en dos estados principales: **standby** *y conexión.* Además existen siete sub-estados: **page, page scan, inquiry, inquiry scan, master response**, **inquiry response y slave response**. Los

sub-estados son usados para agregar nuevos esclavos a una *picored* [3]. Para moverse de un estado a otro se usan comandos de capas más altas o señales internas. En la **figura 1-13** se muestra la transición de los estados de la capa de control.

El sub-estado **inquiry**, define un procedimiento de búsqueda que se usa en aplicaciones donde la dirección del dispositivo de destino es desconocida para la fuente. Esto puede ser usado para descubrir qué otras unidades *Bluetooth* están dentro del rango. Durante este sub-estado, la unidad de descubrimiento recoge la dirección del dispositivo y el reloj de todas las unidades que respondan a tal mensaje de búsqueda; entonces, la unidad puede iniciar una conexión con alguna de las unidades descubiertas. El mensaje de búsqueda difundido por la fuente no contiene información de ella, sin embargo, puede indicar qué clase de dispositivos deberían responder.

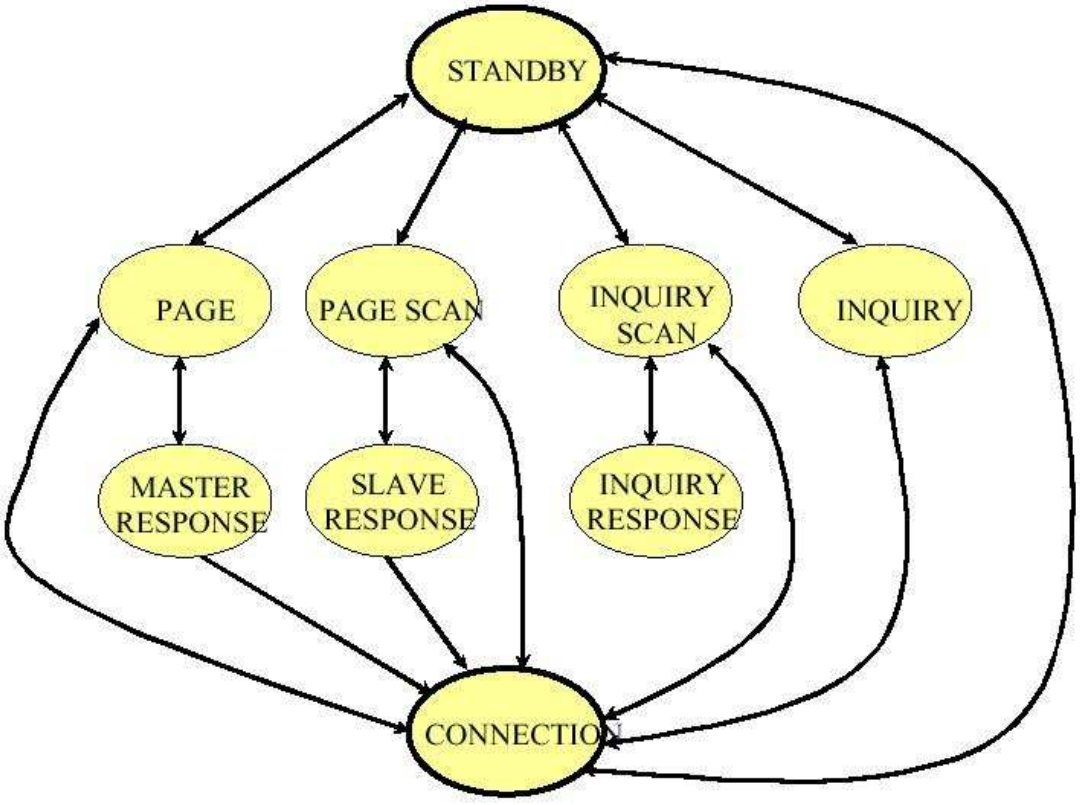

Figura 1-13. Diagrama de transición de estados Bluetooth.

Una unidad que permita ser descubierta, regularmente entra en un sub-estado de **inquiry scan** para responder a los mensajes de búsqueda.

Existen dos formas de detectar otras unidades Bluetooth. La primera, detecta todas las otras unidades en el rango de cobertura, y la segunda detecta un tipo específico de unidades. Los esclavos que se encuentran en el sub-estado de **page scan**, escuchan esperando su propio código de acceso de dispositivo. El maestro en el sub-estado **page** activa y conecta a un esclavo. El maestro trata de capturar al esclavo transmitiendo repetidamente el código de acceso de dispositivo en diferentes canales de salto. Debido a que los relojes del maestro y del esclavo no están sincronizados, el maestro no sabe exactamente cuando y en qué frecuencia de salto se activará el esclavo. Después de haber recibido su propio código de acceso de dispositivo, el esclavo transmite un mensaje de respuesta. Este mensaje de respuesta es simplemente el código de acceso de dispositivo del esclavo. Cuando el maestro ha recibido este paquete, envía un paquete de control con información acerca de su reloj, dirección, clase de dispositivo, etc. El esclavo responde con un nuevo mensaje donde envía su dirección. Si el maestro no obtiene esta respuesta en un determinado tiempo, él reenvía el paquete de control. Si el esclavo excede el tiempo de espera, entonces retorna al sub-estado de **page scan.** Si es el maestro quien lo excede, entonces retorna al sub-estado de **page** e informa a las capas superiores.

Cuando se establece la conexión, la comunicación inicia con un paquete de sondeo desde el maestro hacia el esclavo. Como respuesta se envía un nuevo paquete de sondeo y de esta forma se verifica que la secuencia de salto y la sincronización sean correctas. La **figura 1-14** muestra la inicialización de la comunicación sobre el nivel *banda bas*e. Cada **transceiver** (receptor-transmisor) *Bluetooth* tiene una única dirección de dispositivo de 48 bits asignada, la cual está dividida en tres campos: campo *LAP*, campo *UAP* y campo *NAP* [3]. Los campos *LAP y UAP* forman la parte significativa del código de acceso. En la **figura 1-15** se puede observar el formato de

la dirección para un dispositivo *Bluetooth.* La dirección del dispositivo es conocida públicamente y puede ser obtenida a través de una rutina **inquiry.**

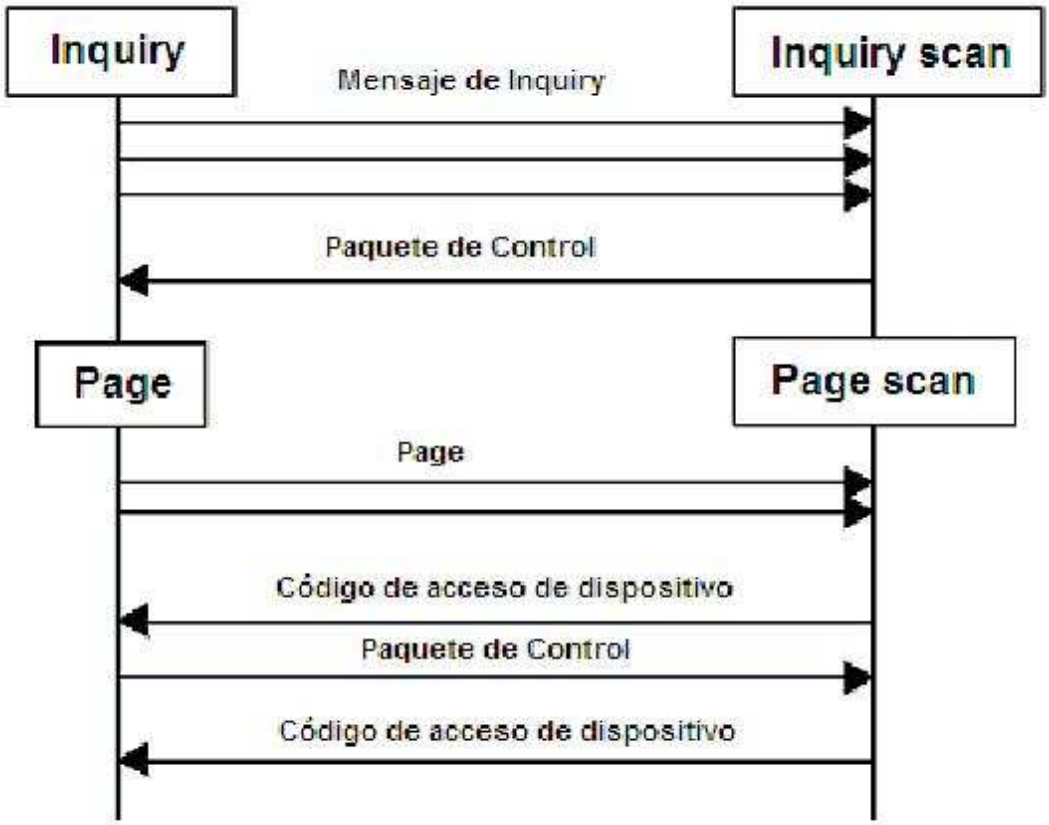

Figura 1-14. Iniciación de Comunicación sobre el nivel banda base

La dirección Bluetooth de los dispositivos, está compuesta por 48 bits dividida en tres campos:

- Non-Significant Address Part (NAP): 16 bits.
- Upper Address Part (UAP): 8 bits.
- Lower Address Part (LAP): 24 bits

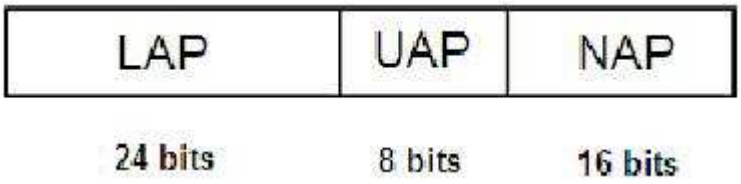

Figura 1-15 Dirección de dispositivo Bluetooth

8) Seguridad en Bluetooth. Con el fin de brindar protección y confidencialidad a la información, el sistema debe ofrecer medidas de seguridad en las dos capas, la de aplicación y de enlace. Todas las unidades *Bluetooth* tienen implementadas las mismas rutinas de autenticación y encriptación. En la capa de enlace, estas rutinas constan de cuatro entidades diferentes: *una única dirección pública, dos llaves secretas y un número aleatorio* el cual es diferente para cada transacción. *Solamente es encriptada la carga útil.* El *código de acceso* y la cabecera de paquete nunca son *encriptados.*

Cada tipo de unidad *Bluetooth* tiene una rutina de *autenticación* común. El maestro genera un número aleatorio y lo envía al esclavo, el esclavo usa este número y su propia identidad para calcular el número de autenticación. Luego, este número es enviado al maestro quien hace el mismo cálculo. Si los dos números generados son iguales entonces la autenticación es concedida.

La *encriptación,* frecuentemente está restringida por leyes de varios países. Para evitar estas limitaciones, en *Bluetooth* la llave de encriptación no es estática, ésta es deducida de la llave de autenticación cada vez que se activa la encriptación [3].

1.3.2 PROTOCOLO DE GESTIÓN DE ENLACE (LMP)

En el *protocolo de gestión de enlace, LMP,* se usan mensajes asociados con el establecimiento, seguridad y control. Los mensajes son enviados en la carga útil y no en los mensajes de datos de *L2CAP*. Los mensajes *LMP* son separados de los demás por medio de un valor reservado en uno de los campos de la cabecera de carga útil.

Todos los mensajes *LMP* son extraídos e interpretados por la capa *LMP* del receptor, esto significa que ningún mensaje es enviado a capas superiores. Los mensajes *LMP* tienen mayor prioridad que los datos de usuario, esto significa que si la *gestión de enlace* necesita enviar un mensaje, éste no debe ser retrasado por otro tráfico. Solamente las retransmisiones de los paquetes del nivel de banda base pueden retrasar los mensajes *LMP*. Además, éstos no necesitan rutinas de reconocimiento ya que la capa banda base asegura un enlace confiable [3]. El *protocolo de gestión enlace* soporta mensajes para:

- Autenticación.
- Paridad.
- Encriptación.
- Temporización y sincronización.
- Versión y características.
- Conmutación para desempeño como maestro o esclavo dependiendo de si el dispositivo es quien inicia (maestro) o no (esclavo) el enlace con otro dispositivo.
- Petición de nombre.
- Desconexión.
- Modo **hold:** el maestro ordena al esclavo entrar en este estado para ahorro de potencia.
- Modo **sniff:** para envío de mensajes en ranuras de tiempo específicas.
- Modo **park:** para que el esclavo permanezca inactivo pero sincronizado en la *picored.*
- Enlaces *SCO.*
- Control de paquetes **multi-slot.**
- Supervisión de enlace.

1) Establecimiento de conexión. Después del procedimiento **paging**, el maestro debe encuestar al esclavo enviando paquetes de sondeo. El otro lado recibe este mensaje y lo acepta o rechaza, si es aceptado, la comunicación incluyendo las capas superiores están disponibles (ver **Figura 1-16**).

Figura 1-16 Establecimiento de la Conexión Bluetooth.

1.3.3 INTERFAZ DEL CONTROLADOR DE HOST (HCI)

La *HCI* proporciona una interfaz de comando al *controlador banda base* y a la *gestión de enlace,* además de acceso al hardware y a los registros de control. Esta interfaz brinda un método estándar para acceder a los recursos de banda base *Bluetooth* [3].

1) Capas mas bajas del conjunto Bluetooth. A continuación se hace una breve descripción de las capas más bajas del conjunto de *software y hardware Bluetooth.* La **Figura 1-17** da una idea de estas capas.

El *firmware HCI* implementa los comandos *HCI* para el *hardware* a través del acceso de comandos banda base, comandos de la *gestión de enlac*e, hardware de registros de estado, registros de control y registros de eventos.

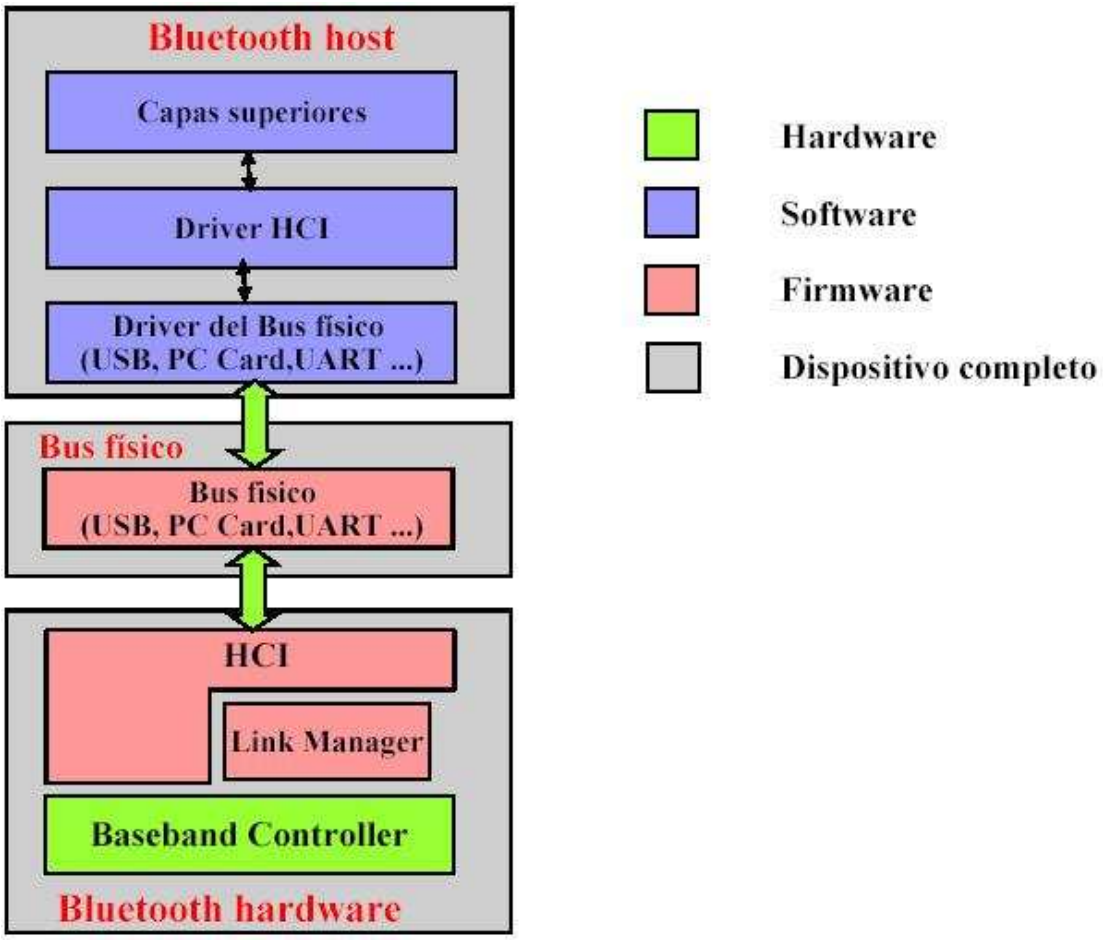

Figura 1-17. Diagrama general de las capas más bajas

El **driver** *HCI* intercambia datos y comandos con el *firmware HCI* (en el hardwar*e*). El **driver** de la capa de transporte, por ejemplo un bus físico, brinda a las dos capas *HCI* la posibilidad de intercambiar información.

2) Posibles arquitecturas de bus físico. Los dispositivos *Bluetooth* tienen varias interfaces de bus físicas que pueden ser usadas para conectar el hardware. Estos buses pueden tener diferentes arquitecturas y parámetros. El *controlador de*

host soporta tres arquitecturas de bus físico, *USB, UART y PC Card.* Todas ellas pueden manejar varios canales lógicos sobre el mismo canal físico simple (a través de **endpoints**), por lo tanto los canales de control, datos y voz no requieren alguna interfaz física adicional.

El objetivo principal de la capa de transporte del *controlador de host* es la transparencia entre el driver del *controlador de host* y el *controlador de host.*

1.3.4 PROTOCOLO DE CONTROL Y ADAPTACIÓN DE ENLACE LÓGICO (L2CAP)

L2CAP es la capa que se encuentra sobre el *protocolo de gestión de enlace (LMP)* y se encarga de adaptar los protocolos superiores al protocolo de banda base. *L2CAP* permite a protocolos de niveles superiores y a aplicaciones, la transmisión y recepción de paquetes de datos *L2CAP* de hasta 64 kilobytes, con capacidad de multicanalización para distintos protocolos, operación de segmentación y reensamble, así como también abstracción de grupos. Para cumplir sus funciones, *L2CAP* espera que la *banda base* suministre paquetes de datos en **full duplex,** que realice el chequeo de integridad de los datos y que reenvíe los datos hasta que hayan sido reconocidos satisfactoriamente. Las capas superiores que se comunican con *L2CAP* son por ejemplo el *protocolo de descubrimiento de servicio (SDP), el RFCOMM y el control de telefonía (TCS)* [3].

1) Canales. L2CAP se basa en el concepto de *canales*. Un canal es una conexión lógica que se sitúa sobre la conexión de banda base. Cada canal se asocia a un único protocolo. Cada paquete L2CAP que se recibe en un canal se redirige al protocolo superior correspondiente. Varios canales pueden operar sobre la misma conexión de banda base, pero un canal no puede tener asociado a más de un protocolo de alto nivel. Los canales de datos orientados a la conexión representan una conexión entre dos dispositivos, donde un *CID (Channel Idedntification)* identifica hacia donde

está conectado cada canal. Los canales no orientados a la conexión limitan el flujo de datos a una sola dirección. La señalización de canal es un ejemplo de un canal reservado. Este canal reservado es usado para crear y establecer canales de datos orientados a la conexión. En la **figura 1-18** se muestra el establecimiento de canales en la capa L2CAP.

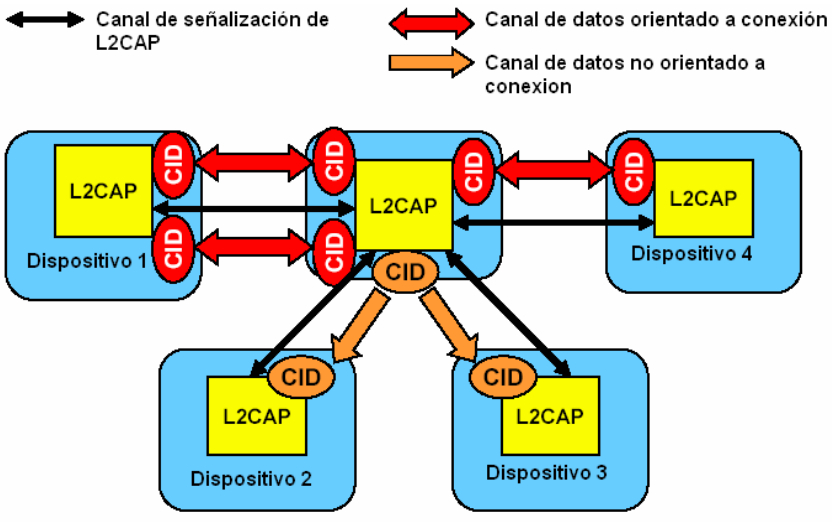

Figura 1-18. Establecimiento de canales en L2CAP.

2) Operaciones entre Capas. Las implementaciones *L2CAP* deben transferir datos entre protocolos de capas superiores e inferiores. Cada implementación debe soportar un grupo de comandos de señalización, además, debe ser capaz de aceptar ciertos tipos de eventos de capas inferiores y generar eventos para capas superiores. En la **Figura 1-19** se muestra esta arquitectura.

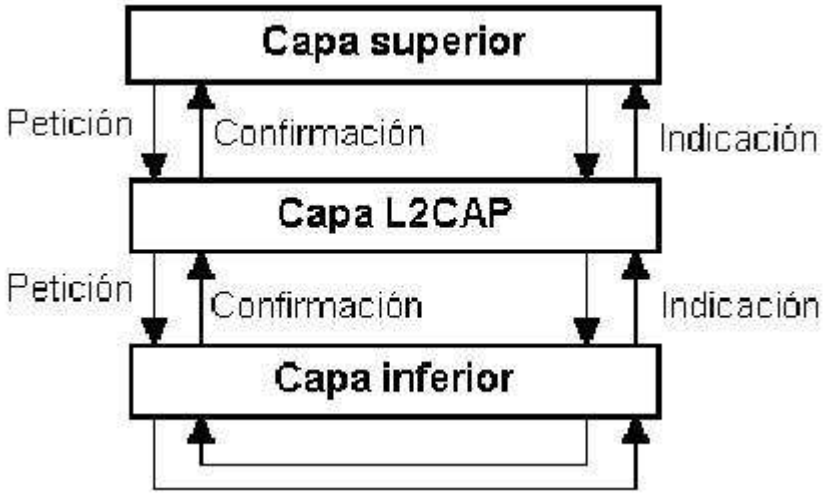

Figura 1-19. Arquitectura L2CAP.

3) Eventos. Todos los mensajes y **timeouts** que entran en la capa *L2CA*P, son llamados eventos. Los eventos se encuentran divididos en cinco categorías: indicaciones y confirmaciones de capas inferiores, peticiones de señal y respuestas de capas *L2CA*P, datos de capas *L2CA*P, peticiones y respuestas de capas superiores, y eventos causados por expiraciones de tiempo.

4) Acciones. Todos los mensajes y **timeouts** enviados desde la capa *L2CAP* son llamados acciones (en el lado del receptor estas acciones son llamadas eventos). Las acciones se encuentran divididas en cinco categorías: peticiones y respuestas a capas inferiores, peticiones y respuestas a capas *L2CA*P, datos a capas *L2CA*P, indicaciones a capas superiores, y configuración de **timers.**

5) Formato del paquete de datos. *L2CAP* está basado en paquetes pero sigue un modelo de comunicación basado en canales. Un canal representa un flujo de datos entre entidades *L2CAP* en dispositivos remotos. Los canales pueden ser o no orientados a la conexión. Como se puede observar en la **Figura 1-20**, los paquetes de canal orientados a la conexión están divididos en tres campos: *longitud de la información, identificador de canal e información*

Longitud (16 bits)	CID (16 bits)	Datos (0-65535 bytes)

Figura 1-20. Paquete L2CAP

Longitud: Especifica la longitud del campo de datos en bytes.

CID: Como L2CAP esta basado en el concepto de canales necesitamos identificar a cada uno de ellos. Por ello utilizamos el CID o identificador de canal que nos permite identificar el canal a través del cual el paquete será entregado. Un identificador de un canal tiene un ámbito local, por lo que un dispositivo pude asignar identificadores de canal de forma independiente de otros dispositivos, excepto que necesite usar identificadores reservados. El mismo CID no puede utilizarse simultáneamente para identificar múltiples canales simultáneos entre un dispositivo local y uno remoto.

Datos: Contendrá los datos recibidos y enviados a la capa de red, además señalar que el la MTU[3] (Unidad Máxima de Transmisión) limitará el tamaño de este campo de carga útil.

6) Segmentación y Reensamblado. Los paquetes de datos definidos por el protocolo *banda base* están limitados en tamaño. Los paquetes *L2CAP* grandes deben ser segmentados en varios paquetes de *banda base* más pequeños antes de transmitirse y luego deben ser enviados a la *gestión de enlace.* En el receptor los pequeños paquetes recibidos de la banda base son reensamblados en paquetes *L2CAP* más grandes. Varios paquetes de *banda base* recibidos pueden ser reensamblados en un solo paquete *L2CAP* seguido de un simple chequeo de integridad. La segmentación y reensamblado, *SAR*, funcionalmente es absolutamente necesaria para soportar protocolos usando paquetes más grandes que los soportados por la *banda*

[3] La mínima MTU (Máxima Unidad de Transmisión) de 1691 se seleccionó con base en el máximo paquete de carga útil de Ethernet (1500 bytes más 15 bytes de encabezado de BNEP y otros de extensión). 1691=5*339 (Tamaño de un DH5)- 4(Encabezado de L2CAP).

*bas*e. La **Figura 1-21** muestra un ejemplo de segmentación y reensamblado de un paquete *L2CAP*.

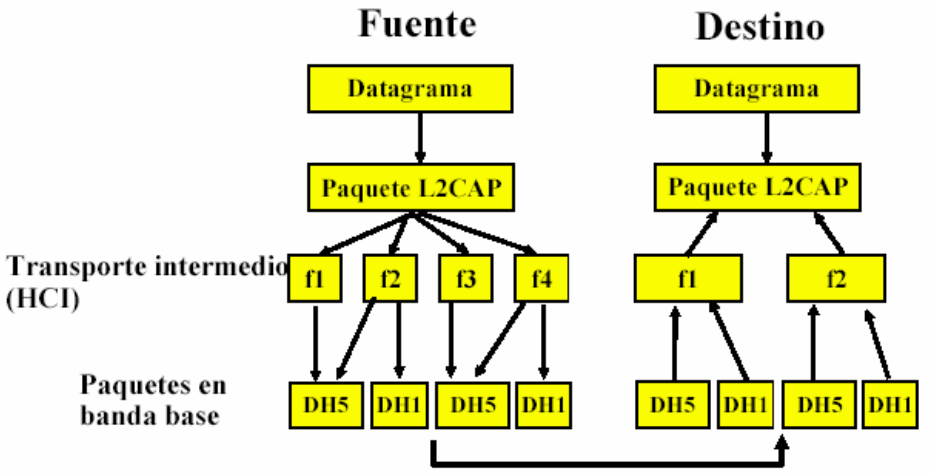

Figura 1-21. Segmentación y reensamblado de paquete L2CAP.

7) Calidad de servicio (QoS). La capa *L2CAP* transporta la información de calidad de servicio a través de los canales y brinda control de admisión para evitar que canales adicionales violen contratos de calidad de servicio existentes.

Algunos esclavos pueden requerir un alto rendimiento o una respuesta rápida. Antes de que un esclavo con grandes peticiones sea conectado a una *picored*, el esclavo trata de obtener una garantía a sus demandas. Puede solicitar una determinada rata de transmisión, tamaño del buffer de tráfico, ancho de banda, tiempo de recuperación de datos, etc. Por lo tanto, antes de que el maestro conecte a un nuevo esclavo o actualice la configuración de calidad, debe chequear si posee **ranuras de tiempo** y otros recursos libres.

1.3.5 PROTOCOLO DE DESCUBRIMIENTO DE SERVICIO (SDP)

El *protocolo de descubrimiento de servicio, SDP,* brinda a las aplicaciones recursos para descubrir qué servicios están disponibles y determinar las características de dichos servicios.

1) Descripción General. Un servicio es una entidad que puede brindar información, ejecutar una acción o controlar un recurso a nombre de otra entidad. El *SDP* ofrece a los clientes la facilidad de averiguar sobre servicios que sean requeridos, basándose en la clase de servicio o propiedades específicas de estos servicios. Para hacer más fácil la búsqueda, el *SDP* la habilita sin un previo conocimiento de las características específicas de los servicios. Las unidades *Bluetooth* que usan el *SDP* pueden ser vistas como un servidor y un cliente. El servidor posee los servicios y el cliente es quien desea acceder a ellos. En el SDP esto es posible ya que el cliente envía una petición al servidor y el servidor responde con un mensaje. El *SDP* solamente soporta el descubrimiento del servicio, no la llamada del servicio.

2) Registros de servicio. Los registros de servicio contienen propiedades que describen un servicio determinado. Cada propiedad de un registro de servicio consta de dos partes, un identificador de propiedad y un valor de propiedad. El identificador de propiedad es un número único de 16 bits que distingue cada propiedad de servicio de otro dentro de un registro. El valor de propiedad es un campo de longitud variable que contiene la información.

3) El protocolo. El *protocolo de descubrimiento de servicio (SDP)* usa un modelo petición/respuesta.

- **Petición de búsqueda de servicio:** se genera por el cliente para localizar registros de servicio que concuerden con un patrón de búsqueda dado como parámetro. Aquí el servidor examina los registros en su base de datos y responde con una *respuesta a búsqueda de servicio.*

- **Respuesta a búsqueda de servicio:** se genera por el servidor después de recibir una *petición de búsqueda de servicio* válida.

- **Petición de propiedad de servicio:** una vez que el cliente ya ha recibido los servicios solicitados, puede obtener mayor información de uno de ellos, dando como parámetros el registro de servicio y una lista de propiedades deseadas.

- **Respuesta a propiedad de servicio:** El *SDP* genera una respuesta a una *petición de propiedad de servicio.* Ésta contiene una lista de propiedades del registro requerido.

- **Petición de búsqueda y propiedad de servicio:** se suministran un patrón de servicio con servicios requeridos y una lista de propiedades deseadas que concuerden con la búsqueda.

- **Respuesta de búsqueda y propiedad de servicio:** como resultado se puede obtener una lista de servicios que concuerden con un patrón dado y las propiedades deseadas en estos servicios.

1.3.6 RFCOMM

El protocolo *RFCOMM* brinda emulación de puertos serie sobre el protocolo *L2CA*P. La capa *RFCOMM* es una simple capa de transporte provista adicionalmente de emulación de circuitos de puerto serie *RS-232*. El protocolo *RFCOMM* soporta hasta 60 puertos emulados simultáneamente. Dos unidades *Bluetooth* que usen *RFCOMM* en su comunicación pueden abrir varios puertos serie emulados, los cuales son multiplexados. La **Figura 1-22** muestra el esquema de emulación para varios puertos serie.

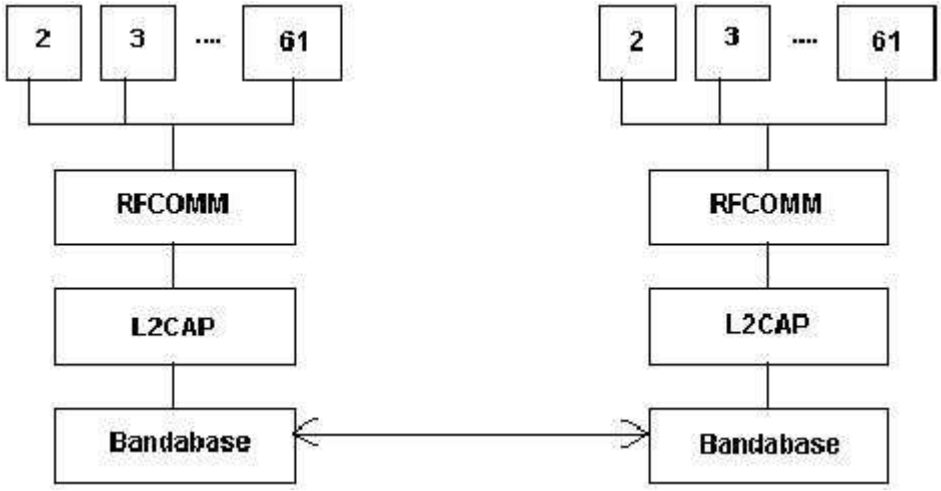

Figura 1- 22. Puertos Serie emulados mediante RFCOMM.

Muchas aplicaciones hacen uso de puertos serie. El *RFCOMM* está orientado a hacer más flexibles estos dispositivos, soportando fácil adaptación de comunicación *Bluetoot*h. Un ejemplo de una aplicación de comunicación serie es el protocolo *punto-a-punto (PPP).* El RFCOMM tiene construido un esquema para emulación de null modem y usa a *L2CAP* para cumplir con el control de flujo requerido por alguna aplicación [3].

1.3.7 PERFILES BLUETOOTH

El estándar *Bluetooth* fue creado para ser usado por un gran número de fabricantes e implementarlo en áreas ilimitadas. Para asegurar que todos los dispositivos que usen *Bluetooth* sean compatibles entre sí son necesarios esquemas estándar de comunicación en las principales áreas. Para evitar diferentes interpretaciones del estándar *Bluetooth* acerca de cómo un tipo específico de aplicación debería ser implementado, el *Bluetooth* **Special Interest Group** *(SIG),* ha definido modelos de usuario y perfiles de protocolo. Un perfil define una selección de mensajes y procedimientos de las especificaciones *Bluetooth* y ofrece una descripción clara de la interfaz de aire para servicios específicos. Un perfil puede ser descrito como una “rebanada” completa del protocolo.

Existen cuatro perfiles generales definidos, en los cuales están basados directamente algunos de los modelos de usuario más importantes y sus perfiles. Estos cuatro modelos son *Perfil Genérico de Acceso (GAP), Perfil de Puerto Serie, Perfil de Aplicación de Descubrimiento de Servicio (SDAP) y Perfil Genérico de Intercambio de Objetos (GOEP).* A continuación se hace una breve descripción de estos y algunos otros perfiles *Bluetooth* [4]. La **Figura 1-23** muestra el esquema de los perfiles *Bluetooth.* En ella se puede observar la jerarquía de los perfiles, como por ejemplo que todos los perfiles están contenidos en el *Perfil Genérico de Acceso (GAP).*

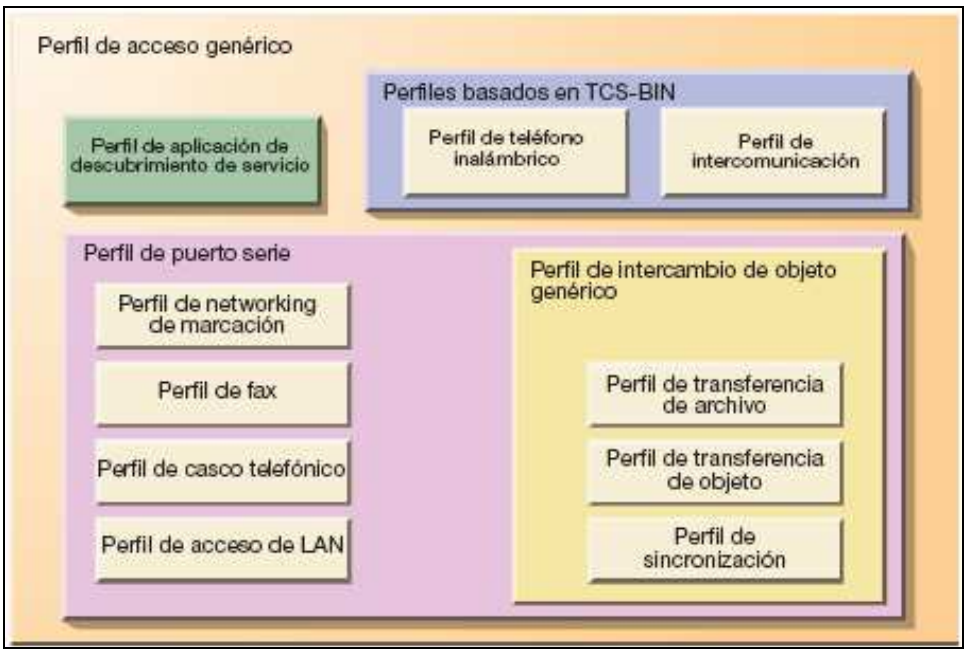

Figura 1- 23. Perfiles Bluetooth

1) Perfil Genérico de Acceso (GAP). Este perfil define los procedimientos generales para el descubrimiento y establecimiento de conexión entre dispositivos *Bluetoot*h. El *GAP* maneja el descubrimiento y establecimiento de conexión entre unidades que no están conectadas y asegura que cualquier par de unidades *Bluetoot*h,

sin importar su fabricante o aplicación, puedan intercambiar información a través de *Bluetooth* para descubrir qué tipo de aplicaciones soportan las unidades.

2) Perfil de Puerto Serie. Este perfil define los requerimientos para dispositivos *Bluetoot*h, necesarios para establecer una conexión serial emulada usando RFCOMM entre dos dispositivos similares. Pueden ser usadas ratas de datos de hasta 128 kbps. El soporte para ratas más altas es opcional. RFCOMM es usado para transportar los datos de usuario, señales de control de *modem* y comandos de configuración. *El perfil de puerto serie* es dependiente del *GAP*.

3) Perfil de Aplicación de Descubrimiento de Servicio (SDAP). Este perfil define los protocolos y procedimientos para una aplicación en un dispositivo *Bluetooth* donde se desea descubrir y recuperar información relacionada con servicios localizados en otros dispositivos. El *SDAP* es dependiente del *GA*P.

4) Perfil Genérico de Intercambio de Objetos (GOEP). Este perfil define protocolos y procedimientos usados por aplicaciones para ofrecer características de intercambio de objetos. Los usos pueden ser, por ejemplo, sincronización, transferencia de archivos o modelo **Object Push**. Los dispositivos más comunes que usan este modelo son agendas electrónicas, *PDA*s, teléfonos celulares y teléfonos móviles. El *GOEP* es dependiente del *perfil de puerto serial*.

5) Perfil de Telefonía Inalámbrica. Este perfil define cómo un teléfono móvil, puede ser usado para acceder a un servicio de telefonía de red fija a través de una estación base, para telefonía inalámbrica de hogares u oficinas pequeñas. El perfil incluye comunicación a través de una estación base, haciendo llamadas de intercomunicación directa entre dos terminales y accediendo adicionalmente a redes externas. Es usado por dispositivos que implementan el llamado “teléfono 3-en-1”.

6) Perfil de Intercomunicador. Este perfil define usos de teléfonos móviles los cuales establecen enlaces de conversación directa entre dos dispositivos. El enlace directo es establecido usando señalización de telefonía sobre *Bluetooth.* Los teléfonos móviles que usan enlaces directos funcionan como **walkie-talkies.**

7) Perfil de Manos Libres. Este perfil define los requerimientos, para dispositivos *Bluetooth,* necesarios para soportar el uso de manos libres. En este caso el dispositivo puede ser usado como unidad de audio inalámbrico de entrada/salida. El perfil soporta comunicación segura y no segura.

8) Perfil Dial-up Networking. Este perfil define los protocolos y procedimientos que deben ser usados por dispositivos que implementen el uso del modelo llamado *Puente Internet*. Este perfil es aplicado cuando un teléfono celular o *modem* es usado como un *modem* inalámbrico.

9) Perfil de Fax. Este perfil define los protocolos y procedimientos que deben ser usados por dispositivos que implementen el uso de fax. Describe como un ordenador puede hacer uso de un teléfono móvil o un MODEM para llevar a cabo el envío y recepción de un fax.

10) Perfil de Acceso LAN. Este perfil define el acceso a una *red de área local, LAN,* usando el protocolo punto-a-punto, *PPP*, sobre *RFCOMM. PPP* es ampliamente usado para lograr acceder a redes soportando varios protocolos de red. El perfil soporta acceso *LAN* para un dispositivo *Bluetooth* sencillo, acceso LAN para varios dispositivos Bluetooth y PC-a-PC (usando interconexión *PPP* con emulación de cable serial).

11) Perfil Object Push. Este perfil define protocolos y procedimientos usados en el modelo **object push**. Este perfil usa el *GOEP*. En el modelo **object push** hay

procedimientos para introducir en el **inbox**, sacar e intercambiar objetos con otro dispositivo *Bluetooth.*

12) Perfil de Transferencia de Archivos. Este perfil define protocolos y procedimientos usados en el modelo de transferencia de archivos. El perfil usa el *GOEP*. En el modelo de transferencia de archivos hay procedimientos para chequear un grupo de objetos de otro dispositivo *Bluetooth*, transferir objetos entre dos dispositivos y manipular objetos de otro dispositivo. Los objetos podrían ser archivos o fólderes de un grupo de objetos tal como un sistema de archivos.

13) Perfil de Sincronización. Este perfil define protocolos y procedimientos usados en el modelo de sincronización. Éste usa el *GOEP*. El modelo soporta intercambios de información, por ejemplo para sincronizar calendarios de diferentes dispositivos.

CAPITULO II

2 CONCEPTOS BÁSICOS PARA LA IMPLEMENTACIÓN DE UNA RED DE ÁREA PERSONAL BLUETOOTH.

La especificación *Bluetooth* define perfiles o esquemas estándar de los escenarios de desarrollo para sus aplicaciones; varios de ellos ya se han mencionado en el capítulo 1. Uno de estos escenarios corresponde al *perfil de red de área personal o perfil PAN* **(Personal Area Networking Profil**e). El perfil PAN brinda capacidades de red a los dispositivos *Bluetooth* para lo cual utiliza el *Protocolo de Encapsulamiento de Red BNEP* ***(Bluetooth*** **Network Encapsulation Protoco**l), de gran importancia ya que encapsula los paquetes provenientes de varios protocolos de red *(IPv4, IPv6 e IPX* entre otros) y los transporta directamente sobre la capa de protocolo *L2CAP* de *Bluetooth,* haciendo posible que la red *Bluetooth* se comporte y forme parte de una red TCP/IP.

2.1 PROTOCOLO DE ENCAPSULAMIENTO DE RED BNEP

El protocolo de encapsulamiento de red de *Bluetooth* envuelve los paquetes de los protocolos de red más utilizados, para ser transportados sobre la capa *L2CA*P. *BNEP* soporta los mismos protocolos de red sostenidos por el estándar ***IEEE 802.3*** para ecapsulamiento Ethernet *(IPv4, IPv6, IPX entre otro*s). *BNEP* requiere además un formato de encapsulamiento que optimice los bits de cabecera de tal manera que se ofrezca un manejo eficiente del ancho de banda. Los siguientes apartes son tomados de la especificación del protocolo *BNEP* publicada por el *SIG Bluetooth* [4].

2.1.1 CONSIDERACIONES

1. *BNEP* está implementado usando canales *L2CAP* orientados a conexión.

2. *Bluetooth* se considera un medio de transmisión al mismo nivel *OSI* de *Ethernet, Token Ring, ATM*, etc.

3. Se considera a *L2CAP* como la capa de control de acceso al medio *MAC* **(Media Access Contro**l) de *Bluetooth.*

4. *BNEP* especifica una ***MTU*** *(Unidad máxima de transmisión)* para *L2CAP* de 1691 bytes.

5. Se deben aplicar a *Bluetooth* las reglas de conectividad y las topologías de red definidas por el estándar *IEEE 802.3* [6].

6. El espacio de dirección *Bluetooth BD_ADDR* es administrado por IEEE, y es asignado desde el espacio de dirección de *Ethernet*. De esta manera es posible construir un punto de acceso de red *Bluetooth* como un puente entre los dispositivos *Bluetooth* y una red *Ethernet.*

La **Figura 2-1** muestra la ubicación del *BNEP* dentro del conjunto de protocolos de *Bluetooth.*

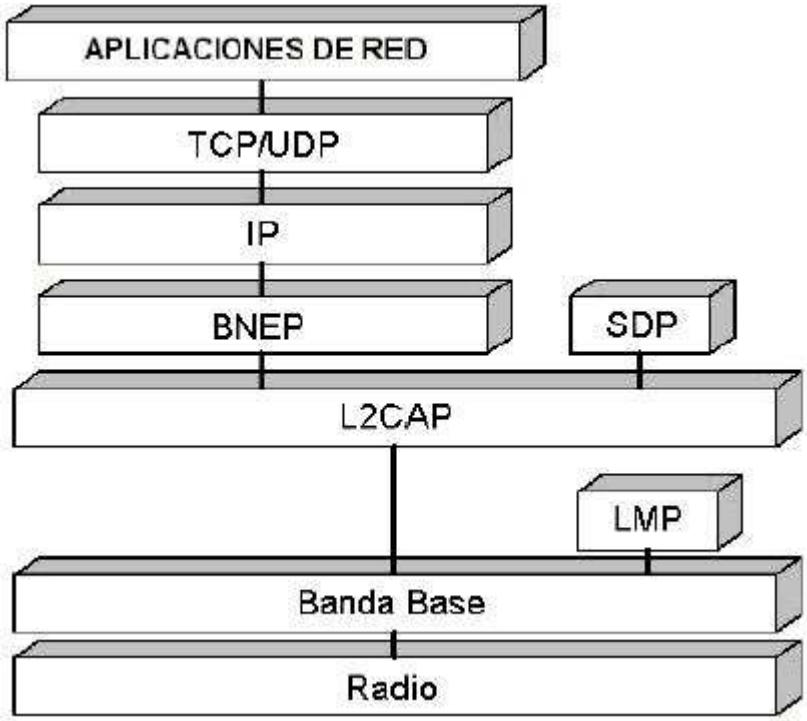

Figura 2-1. Ubicación del BNEP dentro del conjunto de Protocolos Bluetooth.

2.2 PICORED Y SCATTERNET.

Dos o más dispositivos *Bluetooth* que comparten el mismo canal de conexión conforman una ***picored***. Esta se establece a través de enlaces *punto-multipunto*, en donde uno de los dispositivos cumple el rol de ***maestro*** mientras los demás actúan como ***esclavos***. El número máximo de unidades que pueden participar activamente en una picored es de 8, un maestro y siete esclavos, por lo que la dirección o AM_ADDR (Capítulo I, pág. 24) del paquete de cabecera que se utiliza para distinguir a cada unidad dentro de la picored, se limita a tres bits. Algunos de estos esclavos activos, pueden pasar a un estado de *Stand By*, cuando no se encuentra intercambiando información con el maestro, sin embargo el canal de comunicación entre ellos sigue establecido. Adicionalmente, puede haber hasta un máximo de 255 dispositivos en estado *Park* asociados a la picored, que podrían pasar a formar parte de la conexión sustituyendo a un esclavo activo que finalice su conexión con la

picored. Las **Figuras 2-2** y **2-3** ilustran respectivamente el esquema y un ejemplo de una *picored*.

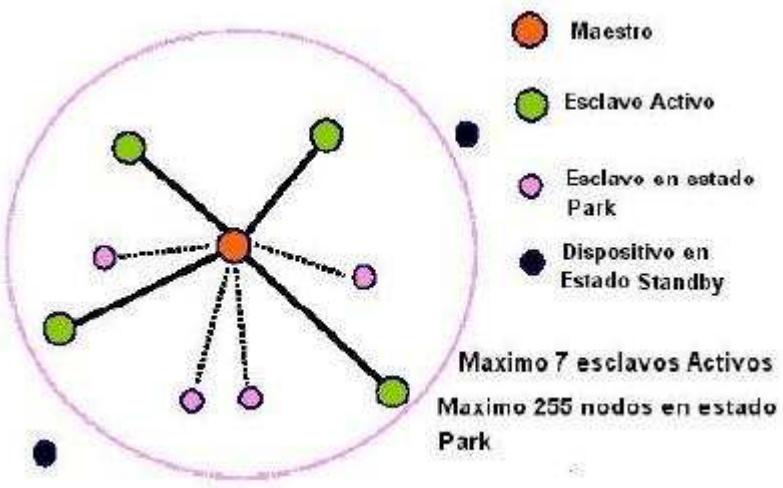

Figura 2-2. Esquema de una Picored.

Figura 2-3. Ejemplo de una Picored.

En una *picored* los enlaces se establecen sobre todas las capas de protocolo *Bluetooth* desde *banda base* hasta *L2CAP*. *BNEP* utiliza estos enlaces *L2CAP* para encapsular los protocolos de red de capas superiores de tal manera que el medio de transporte *(Bluetooth)* es transparente para las *aplicaciones*. El perfil *PAN* describe

los mecanismos mediante los cuales se brindan propiedades de red a los dispositivos *Bluetooth* que forman parte de una picored.

Múltiples *picoredes* en donde sus rangos de cobertura se traslapan conforman una ***scatternet***. Cada *picored* debe tener un único *maestro*, sin embargo, los *esclavos* pueden participar en diferentes *picoredes*. Además el *maestro* de una *picored* puede ser *esclavo* en otra. Las **Figuras 2-4** y **2-5** ilustran respectivamente el esquema y un ejemplo de una *scatternet*.

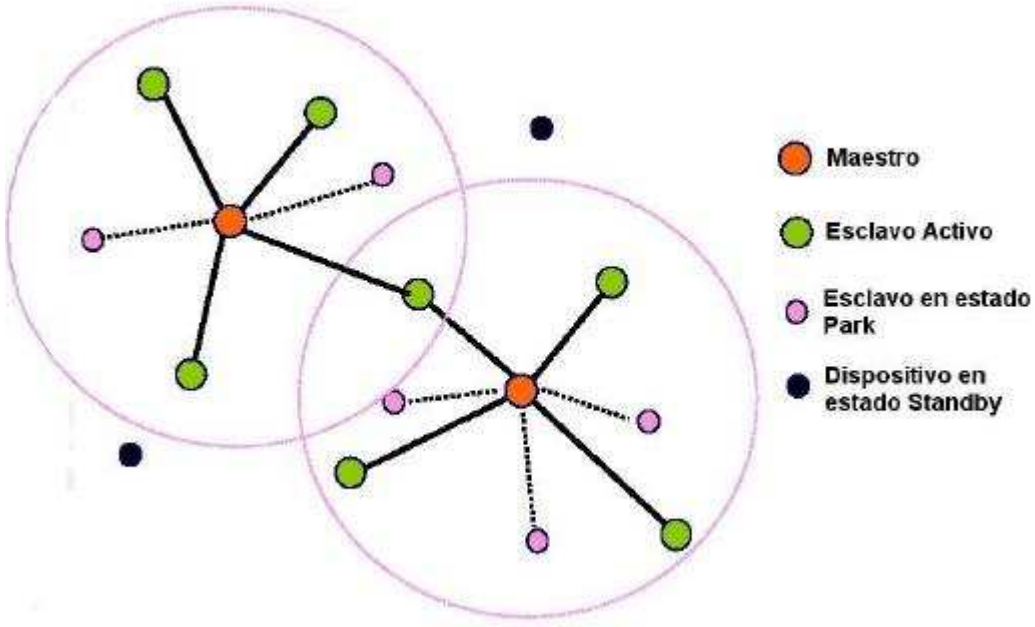

Figura 2-4. Esquema de una Scatternet.

Figura 2-5. Ejemplo de una Scatternet.

2.3 PERFIL DE RED DE AREA PERSONAL (PAN PROFILE)

El *perfil PAN* (**Personal Area Networking Profile**) describe como usar el protocolo ***BNEP*** para brindar capacidades de red a los dispositivos Bluetooth. Estos apartes son tomados del documento **Personal Area Networking Profile** publicado por el *SIG. Bluetooth* [4]. El *perfil PAN* presenta los siguientes requerimientos funcionales:

- Define una red IP ad-hoc, dinámica y personal.
- Debe ser independiente del sistema operativo, lenguaje y dispositivo.
- Brinda soporte para los protocolos de red más comunes como IPv4 e IPv6.
- Brinda soporte para puntos de acceso en donde la red puede ser una LAN corporativa, GSM u otro tipo de red de datos.
- Debe acomodarse a los recursos reducidos disponibles en los dispositivos limitados respecto a memoria, capacidad de procesamiento y uso de interfaces.

2.3.1 CONSIDERACIONES

- El perfil *PAN* debe soportar *IPv4* e *IPv6*. Los otros protocolos pueden estar o no habilitados.
- En una red generalizada, la trayectoria del tráfico originado desde un dispositivo hacia otro puede estar conformada por uno o varios medios de transporte, por ejemplo, *Bluetooth, Ethernet, Token Ring, PSTN, ISDN, ATM, GSM*, etc.

Son tres escenarios propuestos para el *perfil PAN**: Puntos de acceso a una red* (**Network Access Points),** *Grupo de red Ad-hoc* **(Group Ad-hoc Networks)** y *PANU-PANU **(PAN USER).*** Cada uno de estos escenarios define el rol y el servicio que deben asumir los dispositivos involucrados en una de estas arquitecturas.

2.3.2 PUNTOS DE ACCESO A UNA RED (NAP)

Un punto de acceso a una red **(NAP)** corresponde a una unidad que contiene uno o más dispositivos Bluetooth y actúa como *puente, proxy o enrutador,* entre una red Bluetooth y otro tipo de tecnología de red (*Ethernet, GSM, ISDN, Home PNA, Cable Módem y Celular*). La **Figura 2-6** ilustra la arquitectura para dos puntos de acceso a red.

Para la Fase I del perfil PAN, el dispositivo que soporta el servicio **NAP (Network Access Point)** debe cumplir con las características de un *Puente Ethernet* para soportar de esta manera los servicios de red. El dispositivo *NAP* distribuye los paquetes *Ethernet* entre los dispositivos *Bluetooth* conectados o **PAN Users** (***PANU***). Los *NAP* pueden requerir características adicionales en los casos en que el puente sea hacia otro tipo de redes, por ejemplo *GPRS*. La **Figura 2-7** muestra la interacción de las capas de protocolo del modelo *Bluetooth* para el rol NAP.

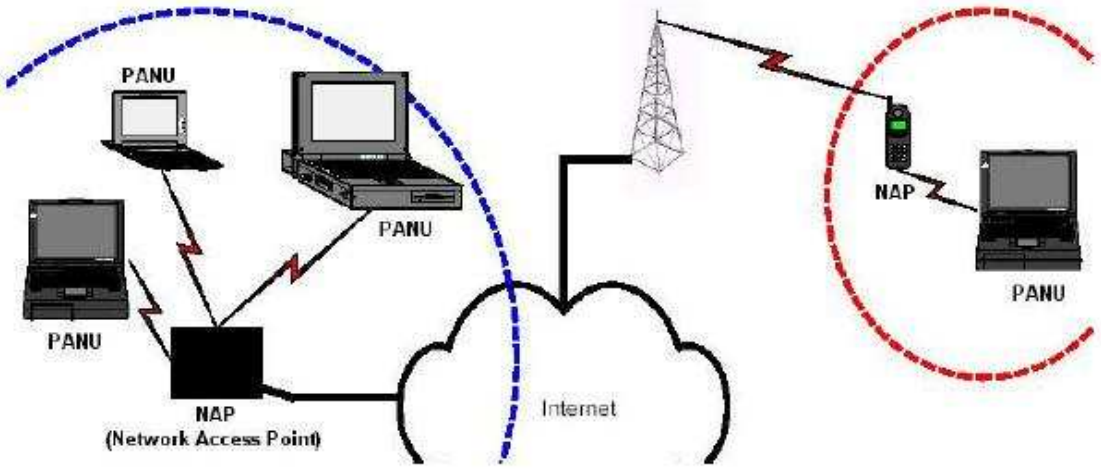

Figura 2-6. Dos puntos de Acceso a Red.

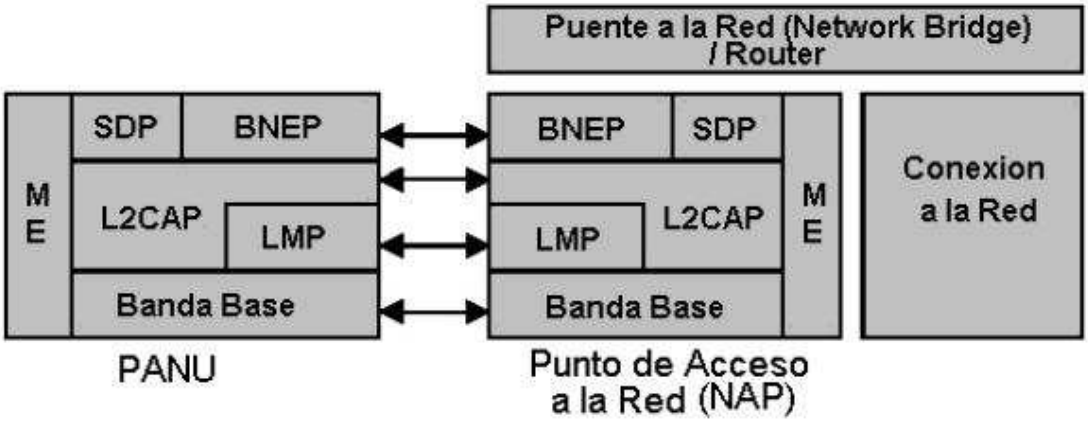

Figura 2-7. Conjunto de protocolos para el rol NAP.

2.3.3 GRUPO DE RED AD-HOC

La *versión 1.0 del Perfil PAN* [3], especifica el escenario para una red personal ad-hoc el cual consiste en una simple ***picored*** con conexiones entre dos o más *dispositivos Bluetooth*. Un maestro y un máximo de siete esclavos conforman esta red. El límite de siete esclavos se debe al esquema activo de direccionamiento de Bluetooth. La **Figura 2-8** es un esquema para una red Ad-hoc.

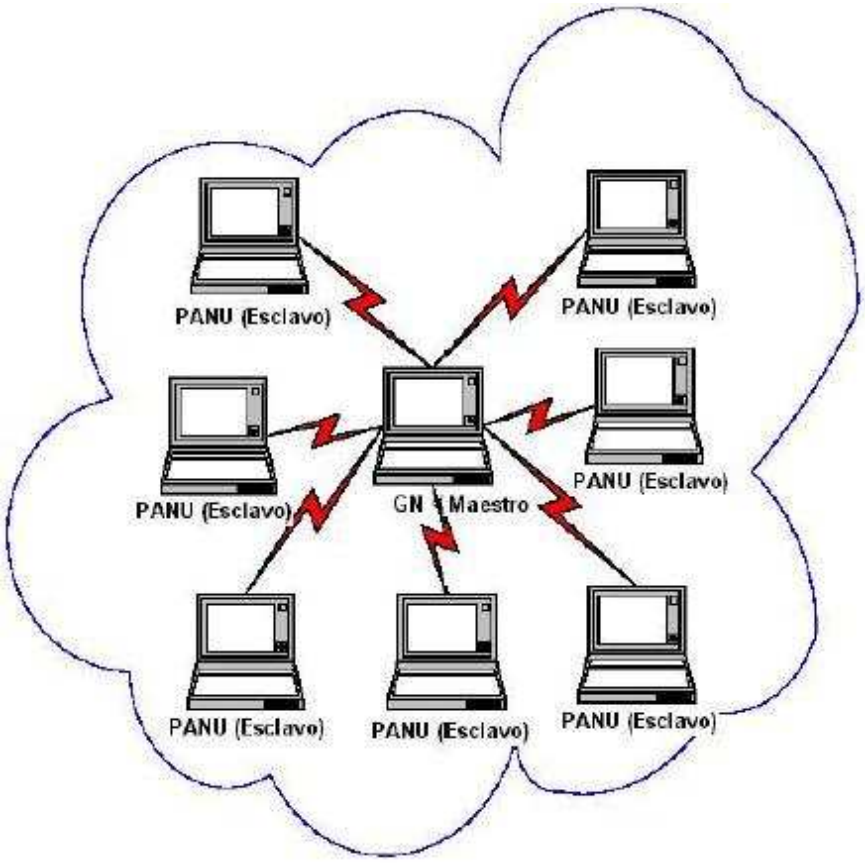

Figura 2-8. Esquema para un Grupo de Red Ad-hoc (GN).

Un grupo de red ad-hoc se establece para que un conjunto de dispositivos conforme una red temporal e intercambien información. El dispositivo cuyo rol es ***GN (Group Ad-hoc Network)*** soporta el servicio *GN***.** La **Figura 2-9** ilustra a nivel de las capas de protocolo un enlace entre un *GN* y un *PANU*.

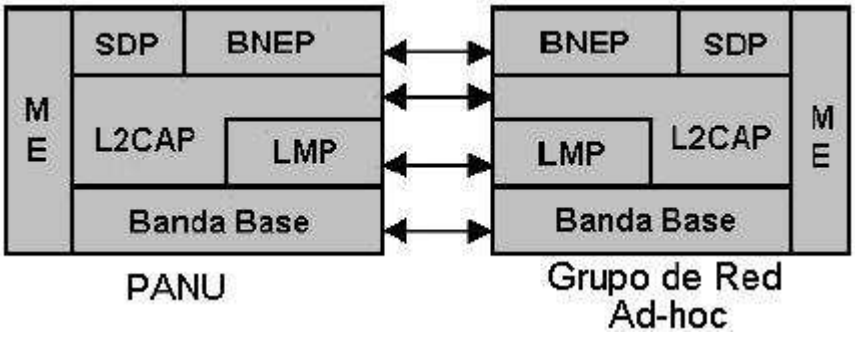

Figura 2-9. Protocolos para un enlace en una red Ad-hoc Bluetooth.

CAPITULO III

3 DISEÑO DE UN ALGORITMO PARA DESARROLLAR UNA APLICACIÓN SOBRE MÓDULOS BLUETOOTH PROGRAMABLES.

Dado que el objetivo principal de este trabajo de grado es el desarrollo de una aplicación basada en la tecnología Bluetooth, anteriormente descrita, y como esta tecnología ha surgido recientemente, se realizó un diseño que consiste en seguir una serie de pasos con el fin de lograr el correcto y completo desarrollo de nuestra aplicación.

Partiendo del concepto de algoritmo que es "un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. De un modo más formal, un algoritmo es una secuencia finita de operaciones realizables, no ambiguas, cuya ejecución da una solución de un problema en un tiempo finito" [5], se procede entonces a hacer una breve descripción de todos los pasos a seguir para el desarrollo de nuestra aplicación basada en una picored Bluetooth y orientada hacia un sistema de seguridad y vigilancia para instalaciones privadas.

Los pasos que conforman nuestro algoritmo son los siguientes:

Paso 1:

Estudiar el *stack* de protocolos utilizados dentro del estándar Bluetooth. Tal y como se expresa en los capítulos I y II de este trabajo, se realizó una investigación a fondo de la tecnología Bluetooth, destacando principalmente sus características y servicios disponibles, con el fin de tener una buena base teórica previa al desarrollo práctico del proyecto.

Paso 2:

Explorar los dispositivos “Bluetooth” programables existentes en el mercado especificando lenguajes, interfaces de comunicación, puertos de entrada/salida, sus bondades y servicios que pueden prestar, arquitectura, estándares involucrados en los mismos, espectro de operación y precios. En la primera parte del capítulo IV se desarrolla este paso.

Paso 3:

Seleccionar los dispositivos “Bluetooth” explorados, aptos y rentables para el diseño de una picored inalámbrica y adquisición de aquellos que requiera la empresa. Este paso forma parte del capítulo IV de este trabajo.

Paso 4:

Desarrollar una propuesta de diseño para un sistema de seguridad y vigilancia que integre los dispositivos “Bluetooth”, para supervisar los accesos de entrada y salida de una propiedad privada. La propuesta es desarrollada en el capítulo V.

Paso 5:

Estudiar la compatibilidad y métodos de adaptación de los dispositivos “Bluetooth” adquiridos, con los equipos electrónicos utilizados en sistemas de seguridad y vigilancia. Este paso también se ilustra en el capítuloV.

Paso 6:

Estudiar el comportamiento de los dispositivos electrónicos con dispositivos Bluetooth incorporados y de cómo podría afectar los factores externos a la seguridad

de la información transmitida. En el capítulo VI se realiza el estudio antes mencionado.

Paso 7:

Programar una aplicación elaborando el Software necesario para controlar los dispositivos Bluetooth adquiridos y realizar pruebas de comunicación entre las unidades, hasta garantizar una correcta operatividad de la picored inalámbrica a implementar. Este Software es el requisito de más peso para lograr la implementación deseada en el trabajo de grado y se desarrolla en el capítulo VI.

Paso 8:

Implementar una picored inalámbrica basada en la tecnología Bluetooth, que cumpla con las demandas del sistema de seguridad y vigilancia propuesto. En el capítulo VI se describe dicha implementación paso a paso.

Paso 9:

Probar la operatividad del sistema completo, conformado por una picored local de al menos tres dispositivos Bluetooth incorporados a tres equipos electrónicos de seguridad que ofrezcan la mayor confiabilidad posible en cuanto a la comunicación entre ellos. En la parte final del capítulo VI se comenta sobre pruebas de operación del sistema.

Paso 10:

Evaluar los costos de la aplicación integrada, la cual se encuentra en la parte final del proyecto.

CAPITULO IV

4 DESCRIPCIÓN DE LOS EQUIPOS BLUETOOTH SELECCIONADOS PARA EL DESARROLLO DEL PROYECTO

4.1 PROCESO DE SELECCIÓN Y ADQUISICIÓN DE EQUIPOS BLUETOOTH.

Luego del estudio realizado con respecto a la tecnología Bluetooth, se procedió a explorar los diferentes dispositivos programables, que trabajan con dicha tecnología, existentes en el mercado, especificando lenguajes, interfaces de comunicación, puertos de entrada/salida, bondades y servicios que pueden prestar, arquitectura, estándares involucrados en los mismos, espectro de operación y precios.

Entre los fabricantes más populares estudiados, que ofrecen módulos Bluetooth programables con características idóneas para el desarrollo de una aplicación comercial se encuentran: BLUERADIOS INC, STOLLMAN, TDK SYSTEMS, ST MICROELECTRONICS, TAIYO YUDEN, NATIONAL SEMICONDUCTOR, entre otros.

Debido a que los equipos desarrollados por los fabricantes nombrados no poseen compañías distribuidoras en Venezuela, la empresa BCCOM Business se veía en la necesidad de adquirir los dispositivos en algún distribuidor en el extranjero, razón por la cual uno de los principales parámetros que se tomaron en cuenta para la elección de los equipos a ser comprados para el desarrollo del proyecto, fue el tiempo necesario para tener los equipos en el país y que tipo de asesoramiento técnico ofrecían los fabricantes o en su defecto la compañía distribuidora de los equipos en el extranjero.

Por las razones antes expuestas, se estableció contacto con los distintos fabricantes y distribuidores de equipos vía correo electrónico, visitando páginas web y estudiando los manuales de los equipos que podían ser descargados de forma gratuita (que no en todos los casos se podía), con el fin de tomar la mejor decisión posible para la compra a realizar.

El parámetro que marcó la pauta para depurar la lista de posibles proveedores y fabricantes de equipos Bluetooth, fue tomar en cuenta únicamente aquellos equipos que pudieran ser controlados utilizando el protocolo estándar de comandos AT[4]. Este estándar es altamente conocido y común en múltiples equipos del ámbito industrial, además de esto, en la empresa BCCOM Business se desarrolló recientemente un proyecto de telecomunicaciones, en el cual se usaron módulos celulares que podían ser controlados remotamente a través de los comandos AT; por tal razón, se tenía en la empresa un conocimiento previo de ellos, así como también de la sencillez con la cual resulta trabajar con éste tipo de comandos.

Los equipos que cumplían con todas estas características fueron los TDK SYSTEMS y los BLUERADIO INC. Vale destacar que la diferencia de precios entre estos equipos era muy poca y en comparación con los precios de todas las marcas en general, se encontraban entre los más accesibles.

Se decidió finalmente por los productos TDK SYSTEMS, fabricante que ofrecía mayor material de estudio sobre sus equipos y muy buena disposición en cuanto a asesoría técnica se refiere.

Los equipos adquiridos para el desarrollo del proyecto fueron: tres módulos serie inteligentes Bluetooth (*Blu*^{2i}), un adaptador USB GO BLUE y una tarjeta de pruebas o de desarrollo para *Blu*^{2i}.

[4] Comandos desarrollados por HAYES™ que permiten al usuario controlar un MODEM a través del puerto serie.

4.2 MÓDULO SERIE INTELIGENTE BLUETOOTH DE TDK - BLU2I -

4.2.1 INTRODUCCIÓN

El Módulo Serie Inteligente Bluetooth de TDK (conocido también como *Blu*^{2i}), es un módulo que se diseña para proporcionar una interfaz Bluetooth simple y de bajo costo. El módulo se diseña para integrarse dentro de una amplia gama de aplicaciones y de plataformas con una interfaz simple de software usando comandos AT. En la **figura 4-1** se observa el modulo serie *Blu*^{2i}.

Figura 4-1. Módulo *Blu*^{2i}.

4.2.2 FUNCIONAMIENTO

El *Blu*^{2i} contiene una interfaz completa Bluetooth y no requiere de ningún otro hardware para implementar una comunicación Bluetooth completa. El módulo tiene integrada una antena de alto rendimiento junto con la circuitería completa de RF y Bandabase. Se interconecta con el sistema microprocesado programable (Host) mediante un puerto serie utilizando los comandos AT. El módulo ejecuta un firmware específico dentro del procesador virtual que incluye un perfil de puerto serie y un interpretador de comandos AT. El *Blu*^{2i} se puede configurar para ser unido a un terminal no inteligente o para incorporarlo a un PC o PDA para aplicaciones de reemplazo de cables. El módulo proporciona el acceso a 5 líneas generales de I/O y a 2 líneas analógicas para proveer de conexiones Bluetooth a dispositivos simples tales como interruptores o LED´s sin requerir de ningún proceso en el extremo del módulo.

En la **tabla 4-1** se muestran las principales características que presentan los *Blu*^{2i} con sus respectivas implementaciones.

Tabla 4-1. Módulo Serie Inteligente de TDK.

CARACTERÍSTICA	**IMPLEMENTACIÓN**
Transmisión Bluetooth	Clase 1
Frecuencia	2.400 – 2.485 GHz
Potencia máxima de transmisión	+6 dBm
Potencia mínima de transmisión	0 dBm
Sensibilidad de recepción	Mayor o igual a -85 dB
Ganancia de la antena	+2 dBi
Alcance	Hasta 200 m (en espacio libre)
Velocidad de transferencia de datos	Hasta 200 Kbps
Interfaz Serie	RS- 232
Baud rate	Configurable desde 9600 bps
Tamaño físico	24 x 69 x 5 mm
Calificación completa Bluetooth	Bluetooth 1.1 PRODUCT
Consumo de Corriente	Menos de 36 mA durante transferencia de datos
Rango de temperatura	Operación normal -20 °C hasta +75°C
Niveles de interfaz	3.3V
Audio	El audio puede ser transferido sobre canales SCO a través de la interfaz PCM a 64 Kbps

4.2.3 INTERFAZ

El módulo serie está equipado con un conector de tarjeta a tarjeta de 40 terminales que es el que se conecta a la plataforma de aplicación. Este conector incluye las siguientes partes:

- Interfaz serie.
- Fuente de poder.
- Interfaz eléctrica.

1) Interfaz Serie: UART_TX, UART_RX, UART_RTS Y UART_CTS forman un puerto serie asíncrono convencional de datos [7]. Esta interfaz es diseñada para funcionar correctamente cuando está conectada con otros dispositivos UART. Los niveles de señalización son 0V y 3.3 V nominales y se invierten con respecto a la señalización sobre un cable RS232. El control de flujo de dos vías es puesto en ejecución por UART_RTS y UART_CTS. UART_RTS es una salida y es activa en nivel bajo. UART_CTS es una entrada y es activa en nivel bajo. Estas señales funcionan según la convención normal de la industria. La señal de UART_RI sirve para indicar llamadas entrantes. UART_DSR es una entrada activa en nivel bajo y debe ser conectada con la salida de DTR del anfitrión. El módulo se diseña para ser usado como DCE, de acuerdo con las convenciones para el DCE-DTE.

El módulo se comunica con el Host usando las siguientes señales:

- Port/ TXD la aplicación envía datos al UART_RX del módulo.

- Port/ RXD la aplicación recibe datos del UART_TX del módulo.

En la **figura 4-2** se ilustra la interfaz UART entre el módulo serie Bluetooth y un Host cualquiera.

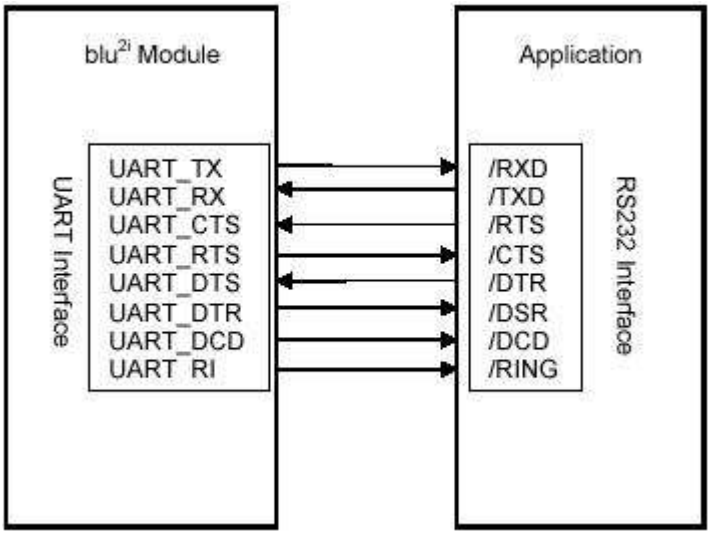

Figura 4-2. Interfaz UART entre el *Blu^{2i}* y una aplicación.

2) Fuente de Poder: La fuente de alimentación para el módulo serie tiene que ser una fuente de voltaje sencilla de Vcc = 3.6V - 6V que debe poder proporcionar la suficiente corriente en una transmisión exigida que puede llegar a consumir 65mA. El módulo incluye reguladores para proporcionar 3.3V y 1.8V locales.

3) Especificaciones Eléctricas de la Interfaz: La asignación de los terminales es la que se indica en la **tabla 4-2.**

Tabla 4-2. Especificaciones de la interfaz eléctrica.

Terminal	Señal	Descripción		Terminal	Señal	Descripción
1	Analógica 0	1.8 V Max		2	GPIO1	I/O for Host
3	Analógica 1	1.8 V Max		4	GPIO2	I/O for Host
5	SPI_MISO	SPI Bus Serie O/P		6	UART_RI	'Ring'entrada o salida
7	SPI_CSB	SPI Chip Select I/P		8	UART_DCD	Input or Output
9	SPI_CLK	SPI Bus		10	UART_DSR	Input

		Clock I/P				
11	GND			12	GPIO3/UART_DTR	I/ O for Host
13	RESET	Reset I/P		14	GPIO4	I/ O for Host
15	GND			16	GPIO5	I/ O for Host
17	SPI_MOSI	SPI Bus serie I/P		18	GND	
19	UART_CTS	Clear to Send I/P		20	PCM_CLK	PCM Clock I/P
21	UART_TX	Transmit Data O/P		22	PCM_IN	PCM Data I/P
23	UART_RTS	Request to Send O/P		24	PCM_SYNC	PCM Sync I/P
25	UART_RX	Receive Data I/P		26	PCM_OUT	PCM Data O/P
27	VCC_3V3	3.3 V Output		28	N/C	
29	VCC_5V	3.6V<Vin< 6.0V		30	GND	
31	N/C			32	RESERVED	No conectar
33	N/C			34	RESERVED	No conectar
35	N/C			36	GND	
37	N/C			38	GND	
39	VCC_1V8	1.8V Output		40	N/C	

4.2.4 RANGOS MÁXIMOS ABSOLUTOS

Los valores máximos absolutos para la fuente de voltaje y los voltajes en los terminales digitales del módulo se enumeran en la **tabla 4-3**. En caso de exceder estos valores, ocurrirán daños permanentes en los módulos.

Tabla 4-3. Rangos máximos absolutos.

Parámetro	**Valor Mínimo**	**Valor Máximo**
Corriente pico de la fuente de poder	0 mA	100 mA
Voltaje en los terminales digitales	-0.3 V	3.7 V
Voltaje en el terminal "POWER"	3.3 V	7 V

El módulo *Blu*^{2i} tiene dos LED's que pueden ser configurados para mostrar el estado de la conexión. Un LED es usado para indicar el estado "Ring Indicate" o para seguir el estado de entrada de la línea DSR sobre la interfaz UART. El otro LED indica que la conexión ha sido establecida. El modo de operación del módulo puede ser descrito mejor por indicación de los comandos AT requeridos para entrar en el modo deseado.

4.2.5 ALCANCE DEL *BLU* ^{2i}

En la **figura 4-3** se puede observar la tasa de transferencia de datos en función de la distancia a la cual se encuentren dos módulos serie conectados. El rendimiento de procesamiento del *Blu*^{2i} está limitado a 200Kbps debido a la cantidad de datos que son transferidos a través del procesador de comandos AT. En la **figura 4-3** se aprecia también el mejor caso de transferencia de datos, con y sin el procesamiento de comandos AT. Las distancias son medidas en espacio libre entre dos *Blu*^{2i}.

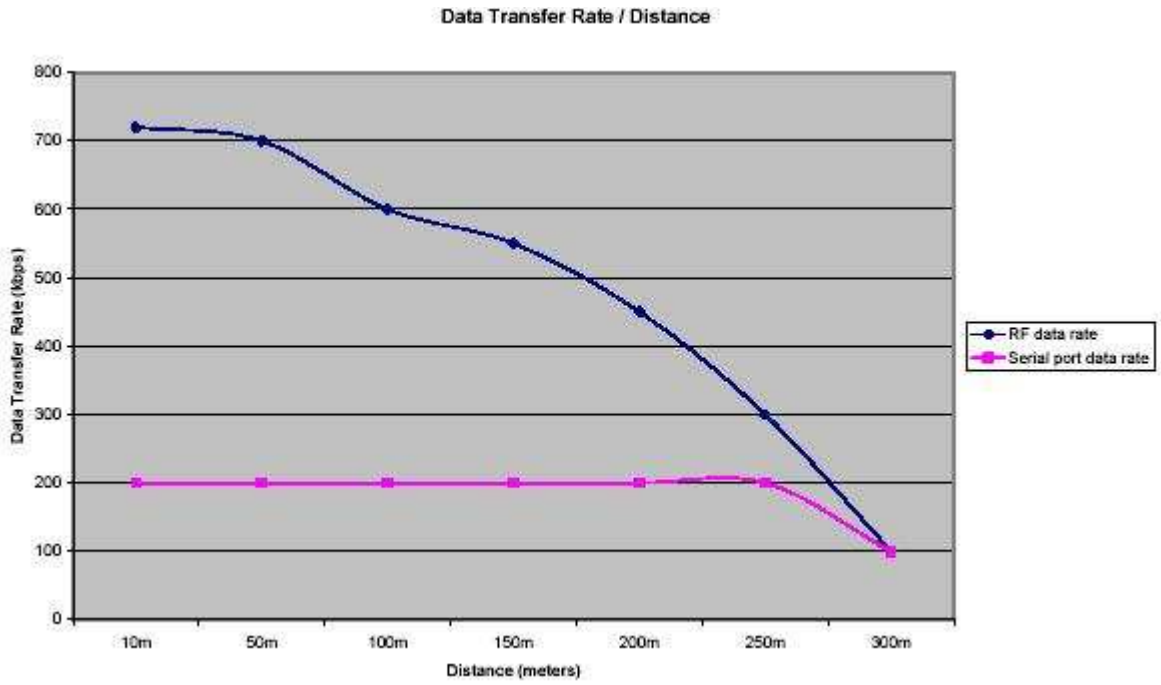

Figura 4-3. Tasa de Transferencia de datos Vs distancia entre módulos.

4.3 ADAPTADOR USB GO BLUE DE TDK

El adaptador USB *Go Blue* es un dispositivo que ofrece conectividad Bluetooth con diversos equipos equipados con esta tecnología. Este adaptador tiene la capacidad de interactuar con otros adaptadores USB *Go Blue,* con módulos serie *Blu*^{2i}, con *PC Card* Bluetooth y adaptadores Bluetooth TDK RS232 entre otros, los cuales son varios de los productos del mismo fabricante.

La aplicación más común del *Go Blue,* es la de establecer conexión entre un PC de escritorio o un PC portátil, y teléfonos celulares, con el fin de cargar y descargar datos sin necesidad de utilizar un cable. Algunos de los equipos celulares compatibles con el *Go Blue TDK,* son los modelos Ericsson R520m, Ericsson T39m, Ericsson T68, Nokia 6210 y Nokia 6310 entre los más conocidos.

El adaptador USB *Go Blue* a diferencia del *Blu*^{2i}, no puede ser controlado a través de la interfaz de comandos AT debido al tipo de puerto que éste maneja. En lugar de esto se complementa con un CD-ROM de instalación de Software diseñado exclusivamente para este tipo de dispositivos y por medio del cual se administran todas las funciones de un dispositivo *Go Blue* conectado a un PC. En la **figura 4-4** se puede observar el adaptador USB *Go Blue* de TDK.

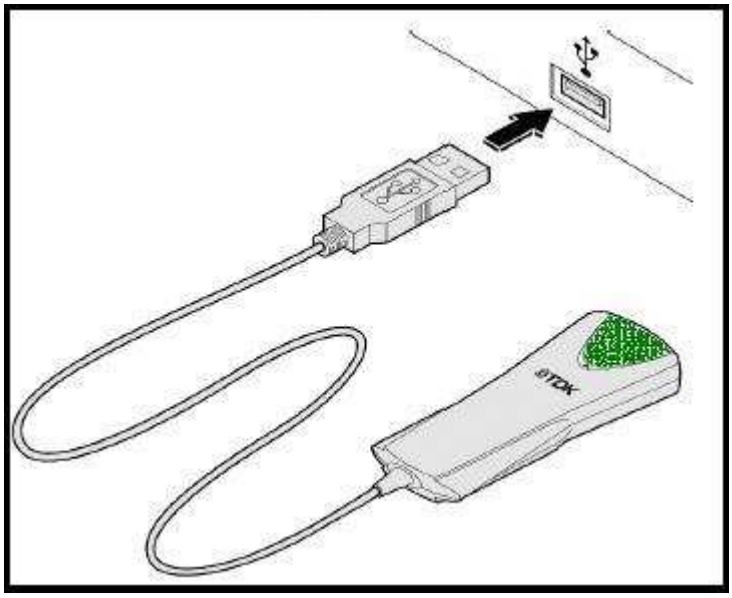

Figura 4-4. Adaptador USB Go Blue.

4.4 TARJETA DE PRUEBAS O DESARROLLO PARA BLU ^{2I} DE TDK.

La tarjeta de pruebas para el *Blu*^{2i} de TDK, permite al módulo serie ser conectado de manera sencilla a un PC. Esta tarjeta provee conversión de niveles a RS232 y un conector estándar DB9 hembra. Adicionalmente tiene un conector tipo DB25 que da el acceso al bus SPI en el módulo para las actualizaciones del Firmware. Este conector también permite la experimentación con las capacidades de audio de los módulos. La tarjeta de desarrollo esta equipada con una interfaz serie RS232 y una interfaz de audio. Un módulo *Blu*^{2i} montado sobre una tarjeta de pruebas

o desarrollo se muestra en la **figura 4-5**. Se indican también los conectores, switches y elementos más resaltantes de las tarjetas.

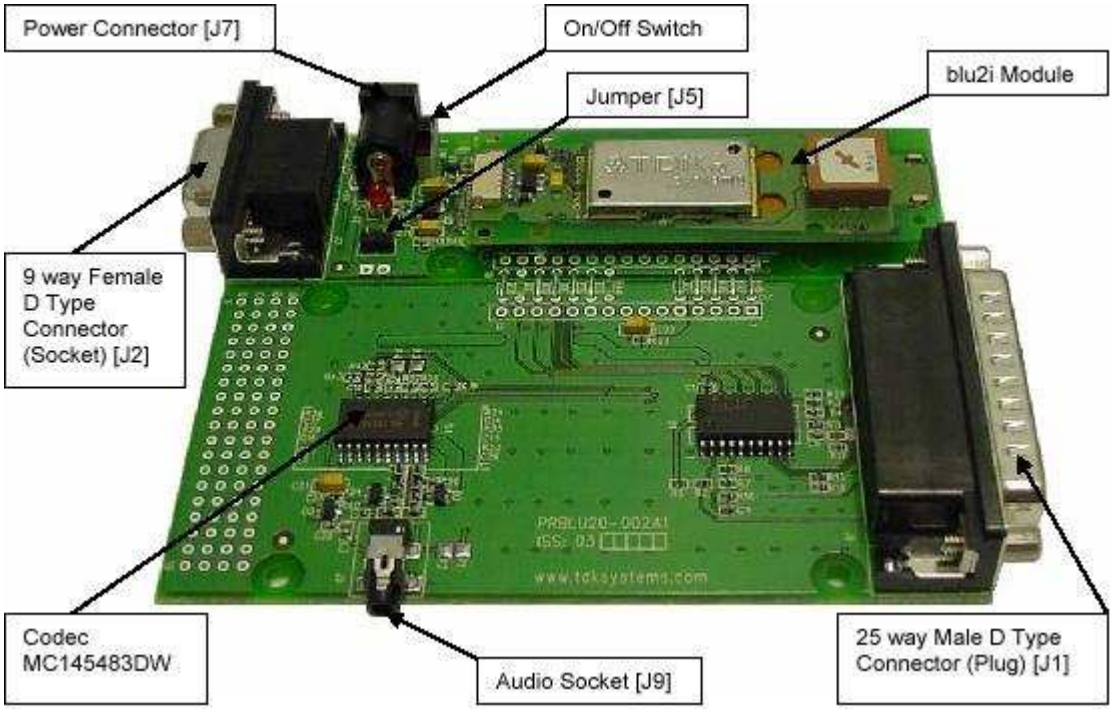

Figura 4-5. Tarjeta de pruebas para *Blu^{2i}*.

CAPITULO V

5 PROPUESTA PARA EL DESARROLLO DE UN SISTEMA DE SEGURIDAD QUE SUPERVISE LOS ACCESOS DE ENTRADA Y SALIDA A INSTALACIONES PRIVADAS HACIENDO USO DE LA TECNOLOGÍA BLUETOOTH.

Partiendo del estudio realizado sobre la tecnología inalámbrica Bluetooth y de los manuales de los equipos TDK adquiridos por la empresa BCCOM Business, se procede a realizar una propuesta para el diseño de un sistema de seguridad y vigilancia que incorpore la tecnología Bluetooth, mediante el uso de los módulos *Blu*^{2i} de TDK Systems.

La empresa BCCOM Business, ha desarrollado recientemente una aplicación de seguridad para instalaciones privadas la cual está basada en dispositivos de seguridad, tales como sensores de movimiento y de sonido conectados a un sistema autónomo de procesamiento de datos que envía un mensaje a uno o varios usuarios, a través de la plataforma SMS de una operadora de telefonía móvil celular del país. Ahora bien, a continuación se propone el diseño de un sistema de equipos de supervisión remotos, con la intención de incorporar la tecnología Bluetooth a un sistema de seguridad que sea compatible y que pueda formar parte en un futuro del sistema que actualmente se está introduciendo en el mercado por la empresa BCCOM Business.

5.1 SISTEMA DE SEGURIDAD Y VIGILANCIA FORMADO POR EQUIPOS DE SUPERVISIÓN REMOTA INALÁMBRICOS SOPORTADOS POR MÓDULOS BLUETOOTH.

El sistema de comunicación inalámbrica que se propone para desarrollar una aplicación de seguridad y vigilancia que tenga como soporte una picored Bluetooth, consta de dos equipos remotos de monitoreo o supervisión, a los que se les incorporan módulos *Blu*^{2i} conectados a través del puerto serie, los cuales se encargarán de dar la información solicitada vía Bluetooth por un maestro conectado a una Unidad de Procesamiento Central, en la cual se reciben las alarmas y reportes de los dispositivos de supervisión de accesos de entrada y salida de una instalación privada. Luego, para darle autonomía a los dispositivos remotos, en el sentido de que puedan recibir y enviar algún tipo de información cuando el Host conectado al dispositivo maestro lo pida, se les debe incorporar una unidad de procesamiento de datos que sirva de intermediario entre un *Blu*^{2i} esclavo y su respectivo Host. Esta unidad de procesamiento puede estar formada por Microcontroladores PIC, los cuales manejan comunicación del tipo serie y han formado parte de aplicaciones desarrolladas con anterioridad dentro de la empresa BCCOM Business, obteniéndose resultados satisfactorios.

Todo esto traería como trabajo adicional, realizar una extensión de la Unidad de Procesamiento Central de datos, que forma parte principal del sistema de seguridad y vigilancia que ya se encuentra implementado en la empresa, lo que luego se traduciría en un sistema de seguridad mucho más amplio. Vale destacar que la Unidad de Procesamiento antes mencionada, maneja el protocolo de comunicación serie, por lo cual se le podría conectar un módulo *Blu*^{2i}, ya que, como se vio en la sección 4.2.3, estos dispositivos poseen una interfaz de comunicación serie. En la **figura 5-1** se muestra un esquema del sistema propuesto.

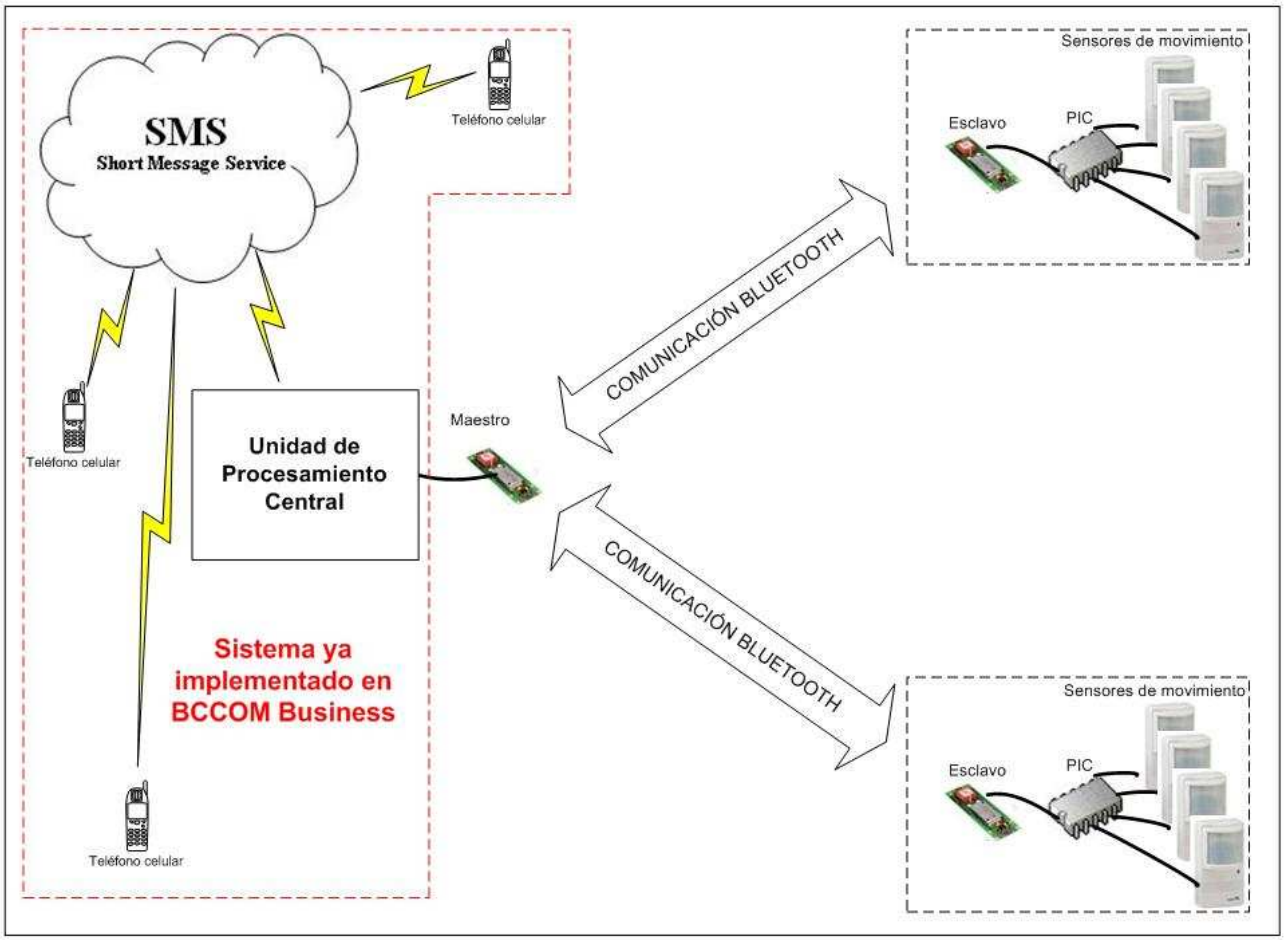

Figura 5-1. Sistema de equipos de supervisión remotos incorporados al sistema ya implementado en la empresa.

Para este sistema propuesto se cuenta únicamente con los módulos Bluetooth que se tienen actualmente en la empresa. Claro está que a la hora de adquirir un mayor número de módulos *Blu*^{2i}, se pueden incorporar a dicho sistema hasta cinco dispositivos *Blu*^{2i} más, de modo que se tenga una picored de un maestro con siete esclavos, la cual es la capacidad máxima de una picored activa Bluetooth.

Debido a las características de comunicación y rangos de cobertura de la tecnología Bluetooth, un sistema de seguridad y vigilancia como este, podría ser muy útil dentro de un sistema central de supervisión de los accesos de entrada y salida de un edificio comercial o de viviendas, así como en un recinto universitario o una

hacienda campestre donde los portones de acceso se encuentren dentro del rango de cobertura de los módulos, de modo que se pueda tener un registro contabilizado, con hora y fecha, de todos los movimientos de personal en una instalación privada cuyo acceso sea restringido.

Este sistema propuesto, había formado parte de las recomendaciones finales del proyecto de grado que dio como resultado el sistema de seguridad implementado en la empresa BCCOM Business [8].

5.2 DELIMITACIÓN DE LA PROPUESTA A DESARROLLAR.

Tomando en cuenta que el objetivo principal de este trabajo de grado es implementar y desarrollar una picored local inalámbrica basada en la tecnología Bluetooth, que se adapte a las necesidades de un sistema de seguridad y vigilancia, la propuesta a desarrollar debe enfocarse en mayor grado al estudio, funcionamiento y puesta en marcha de una picored Bluetooth, sin necesidad de modificar por el momento el Sistema de seguridad que ya está implementado en la empresa BCCOM, ya que esto requeriría ahondar y hacer modificaciones en el funcionamiento interno de la Unidad de Procesamiento Central del sistema ya implementado. Ahora bien, como no se tiene contemplado por el momento en la empresa realizar este tipo de cambios en el sistema recién desarrollado, y apoyándonos en el hecho de que la Unidad de Procesamiento Central permite comunicación a través de un puerto serie, en lugar de hacer las modificaciones antes mencionadas, el dispositivo Bluetooth maestro del sistema que se propone en la sección 5.1, será conectado al puerto serie de un PC. Se deberá desarrollar entonces un software de gestión para el sistema de seguridad y vigilancia formado por un módulo maestro y dos dispositivos de supervisión remotos, conectados a los módulos Bluetooth esclavos. Un esquema general de un sistema propuesto delimitado se muestra en la **figura 5-2.**

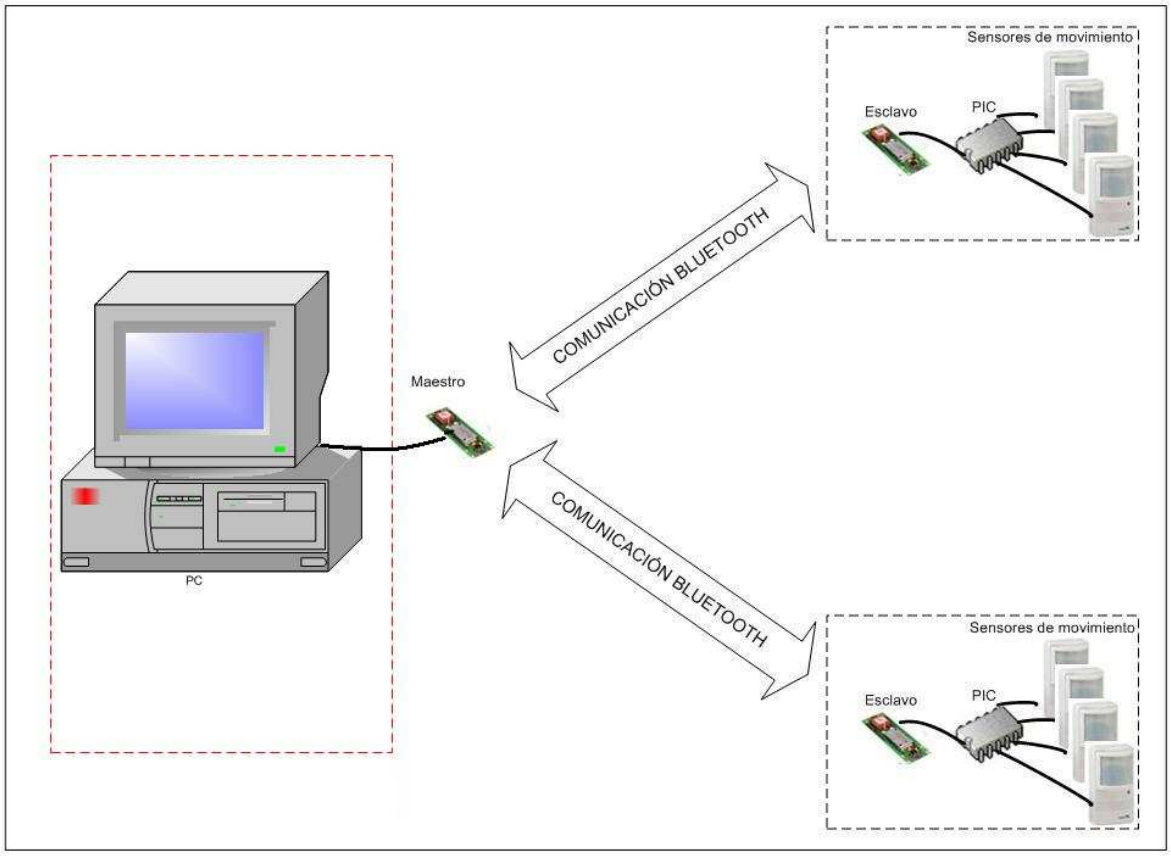

Figura 5-2. Esquema del sistema de seguridad y vigilancia delimitado.

La mayor ventaja que podría ofrecer un sistema apoyado en una picored Bluetooth con estas características, sería la movilidad de cada uno de los dispositivos remotos, los cuales se pueden cambiar de ubicación sin mayor esfuerzo de instalación. Igualmente el módulo *Blu*^{2i} maestro podría ser conectado a cualquier otro computador que se encuentre en el mismo recinto, ya que solo bastaría con instalar el software de gestión del sistema en el computador desde el cual se desee supervisar los accesos y hacer la conexión del módulo Bluetooth al PC a través del puerto serie.

CAPITULO VI

6 IMPLEMENTACIÓN Y DESARROLLO DE UNA PICORED BLUETOOTH QUE RESPONDA A LAS DEMANDAS DEL SISTEMA DE SEGURIDAD PROPUESTO.

En este capítulo se procede a realizar la implementación de una picored inalámbrica local basada en la tecnología Bluetooth, tal que cumpla con los requerimientos totales del sistema de seguridad y vigilancia propuesto en el capítulo V, integrarla a dicho sistema y verificar que todo este desarrollo funcione correctamente luego que sea instalado.

Como punto de partida, vamos a retomar el concepto de Picored, que no es más que "Dos o más dispositivos Bluetooth compartiendo el mismo canal inalámbrico de conexión" [9]. Ahora bien, necesitamos entonces cubrir las necesidades de Hardware y Software para el desarrollo de nuestra aplicación de seguridad para instalaciones privadas que se base en una Picored Bluetooth, pasando también por el protocolo de comunicación que será utilizado para poner en funcionamiento los módulos Bluetooth.

6.1 ELABORACIÓN DE UNA INTERFAZ FÍSICA PARA LA COMUNICACIÓN ENTRE EL BLU ^{2I} Y EL PC (HARDWARE).

Para la realización de las primeras pruebas de comunicación entre un *Blu*^{2i} y un Host, en este caso un PC, se estableció la conexión entre ellos utilizando la Tarjeta de Desarrollo (sección 4.4) que fue adquirida junto con los demás equipos TDK, que posee la interfaz necesaria para que el PC se comunique con el módulo a través del puerto serie utilizando el estándar RS232. Ahora bien, en el momento que se quiere

establecer conexión inalámbrica entre dos o más equipos electrónicos por medio de los *Blu*^{2i}, se necesita una interfaz de comunicación entre el equipo electrónico y su respectivo módulo *Blu*^{2i}, ya que únicamente se cuenta con una sola Tarjeta de Desarrollo. Luego, para hacer esta comunicación posible se elaboró una interfaz de comunicación que permite la conversión de los niveles lógicos de tensión del estándar RS232 del conector DB9 del PC, a los niveles lógicos de tensión TTL del módulo *Blu*^{2i}. Para realizar la implementación de nuestra Picored, se diseñaron dos tarjetas de circuito impreso o PCB, que se muestran en la **figura 6-1**. En la **figura 6-2** se muestra el esquema de dicha interfaz, en la cual se utilizó como elemento principal el chip Max232, el cual tiene la facultad de realizar la conversión los niveles de tensión antes mencionados.

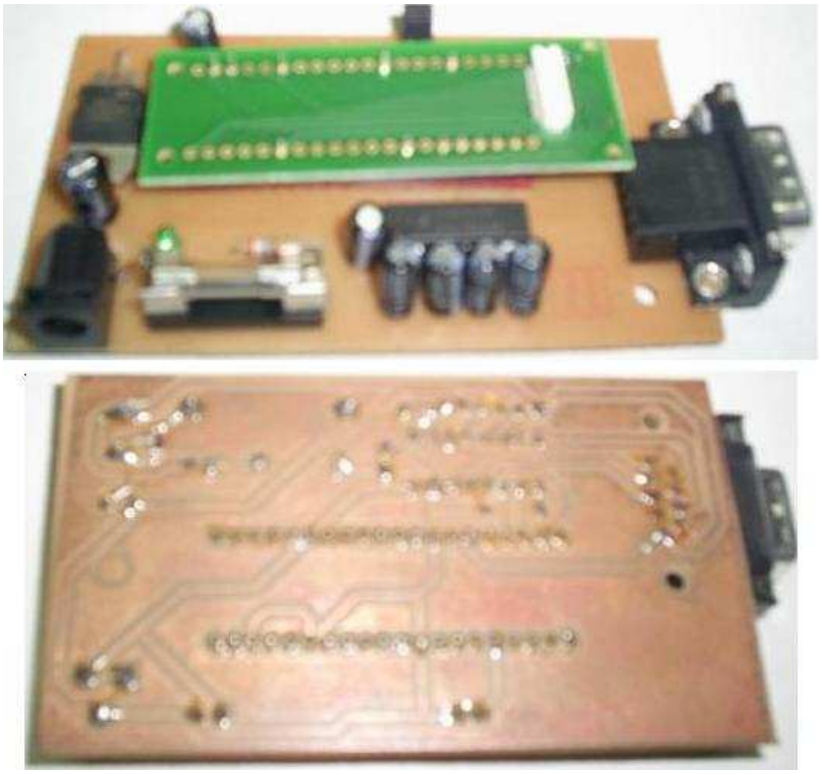

Figura 6-1. Interfaz para comunicar un PC con el *Blu*^{2i} en tarjeta de circuito impreso.

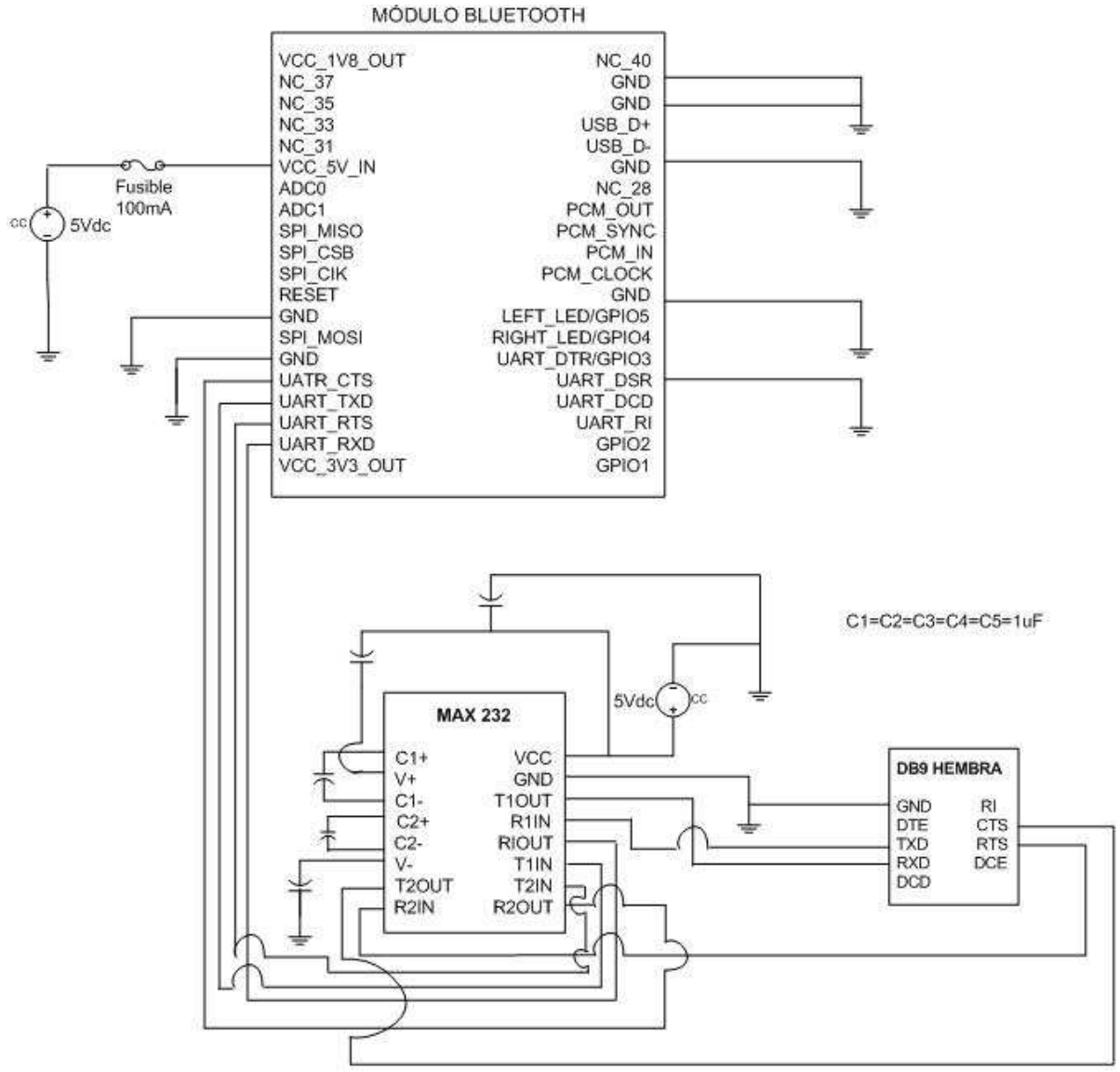

Figura 6-2. Esquema de la interfaz para comunicar el PC con el *Blu^{2i}*.

6.2 INTERFAZ PARA COMUNICAR LOS MÓDULOS BLU ^{2I} CON MICROCONTROLADORES PIC (HARDWARE).

El Hardware necesario para los dispositivos remotos fueron proporcionados por la empresa, puesto que ya se habían diseñado tarjetas de circuito impreso en proyectos anteriores, donde se coloca el Microcontrolador PIC, que maneja niveles lógicos de tensión TTL, para realizar la comunicación del tipo Serial, a través de un

conector macho DB9. Se conectó entonces, la tarjeta que se desarrolló en la sección 6.1 con la tarjeta que ya había sido diseñada anteriormente en la empresa para la comunicación de Microcontroladores PIC con otros dispositivos electrónicos a través de un puerto serie. La tarjeta para Microcontroladores PIC se muestra en la **figura 6-3**.

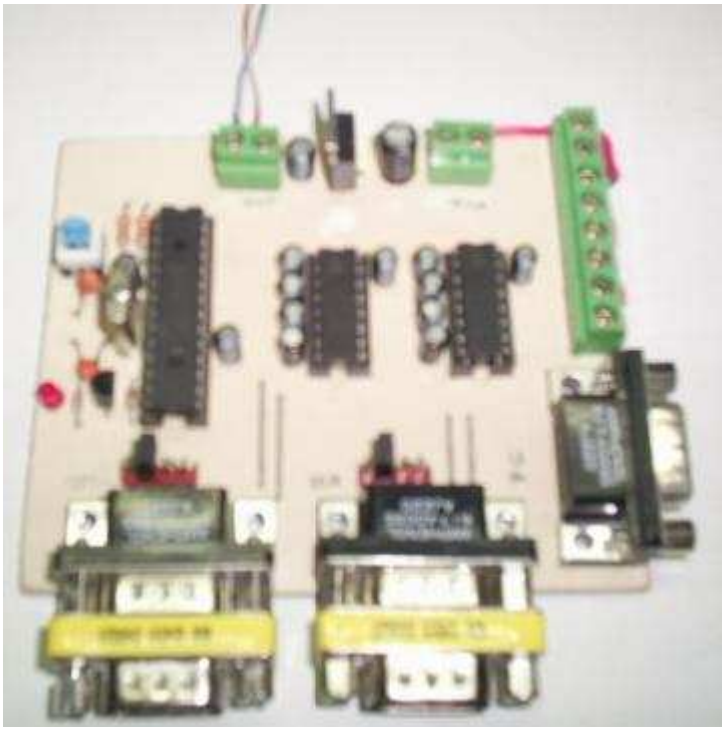

Figura 6-3. Tarjeta para Microcontroladores PIC.

Haciendo uso de esta tarjeta para Microcontroladores, se pueden conectar posteriormente uno o dos detectores de movimiento a un terminal del PIC, de esta manera al momento de que un sensor de movimiento sea activado, emite un cambio brusco de tensión a través de su cable de salida, de manera que el PIC, una vez programado, reconocerá el cambió de tensión en su terminal.

6.3 ESTABLECIMIENTO DE LA COMUNICACIÓN INALÁMBRICA BLUETOOTH.

Para establecer una comunicación Bluetooth entre dos o más dispositivos, por ejemplo computadoras, a través de módulos *Blu*^{2i}, con la finalidad de establecer una picored de la manera más sencilla posible, se procede de la siguiente manera: tomamos dos computadoras que se encuentren en un mismo recinto y a cada una se le conecta en su puerto serie, un módulo *Blu*^{2i} utilizando la interfaz diseñada en la sección 6.1.

Es importante destacar, que los equipos *Blu*^{2i} pueden trabajar bajo dos modalidades diferentes. Para ser más específicos, pueden trabajar usando dos Firmware diferentes, los cuales son descargados a los módulos utilizando dos Software (uno para cada modalidad), que son suministrados por el fabricante.

Los *Blu*^{2i} vienen de fábrica con el ***Firmware AT*** incorporado, el cual permite la comunicación con los módulos a través del puerto serie utilizando el protocolo de Comandos AT para tal fin. Con el uso de comandos AT, se puede configurar y controlar el funcionamiento de los *Blu*^{2i}. El fabricante ha proporcionado los comandos AT apropiados para hacer que el *Blu*^{2i} realice las dos acciones básicas de un dispositivo Bluetooth, las cuales son establecer/romper conexiones y búsqueda de dispositivos Bluetooth dentro del rango de cobertura. Muchos otros comandos también son proporcionados para realizar funciones auxiliares, entre las que se destacan la gestión de la base de datos de dispositivos Bluetooth confiables y el mantenimiento de los registros S[5].

Este Firmware de comandos AT presenta la gran limitación de que únicamente permite establecer conexión entre dos módulos *Blu*^{2i}, un maestro y un

[5] Los registros S son usados para almacenar una gran cantidad de parámetros que poseen los *Blu*^{2i} y estos pueden ser modificados mediante comandos AT.

esclavo, lo cual funciona muy bien para hacer reemplazo de cables de datos serie y muchas otras aplicaciones que requieran conexiones punto a punto, pero no permite desarrollar una picored, que conste de un maestro y hasta siete esclavos, tal y como lo ofrece el estándar Bluetooth.

Por otro lado, se puede cambiar la modalidad de operación de los *Blu*^{2i}, descargándoles el ***Firmware Multipunto***, el cual utiliza una interfaz de mensajes basada en paquetes de datos serie, la cual permite a un Host enviar comandos, recibir respuestas e intercambiar datos multiplexados con el módulo *Blu*^{2i}. Esta modalidad permite conexiones a uno o más esclavos, por lo que utilizando los módulos *Blu*^{2i} bajo este Firmware, se puede desarrollar una picored Bluetooth tal y como se propone en este trabajo de grado.

Tenemos entonces que para controlar todas las funciones que pueda realizar un módulo *Blu*^{2i}, necesitamos usar los Comandos Bluetooth predeterminados para cada función, y tales comandos deben ser enviados a través de un puerto serie en forma de paquetes de datos al dispositivo *Blu*^{2i}.

Se estableció luego la comunicación vía puerto serie entre los módulos *Blu*^{2i} y el puerto COM de un computador, haciendo uso de un pequeño software codificado en lenguaje de programación C++, el cual manipula el puerto COM serie seleccionado y a través del cual se enviaron comandos de comunicación Bluetooth, en forma de paquetes de datos, recibiendo las respuestas y/o eventos correspondientes al comando enviado. En la **figura 6-4,** se muestra la primera prueba realizada a través del programa en lenguaje C++, para comunicar el computador y el *Blu*^{2i}, en el que fue enviado el comando CMD_READ_BDADDR, representado por el paquete de datos 04 00 02 7F, y el cual solicita la dirección Bluetooth del dispositivo. Se recibió la respuesta RSP_READ_BDADDR representada por el paquete de datos 0B 00 02 00 00 00 80 98 74 40 56, que indica que la dirección Bluetooth del dispositivo es 008098744056.

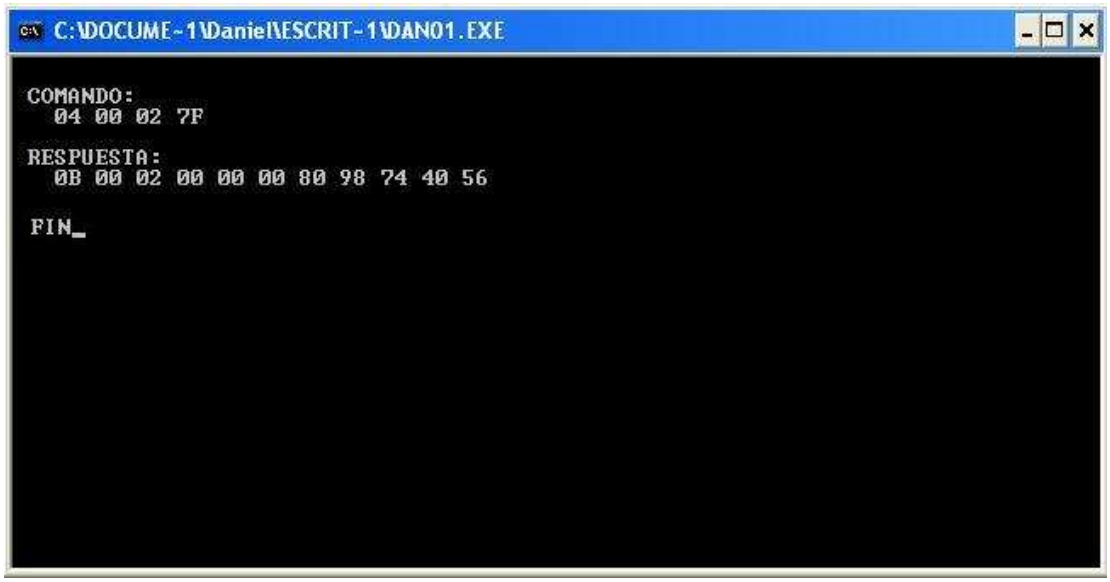

Figura 6-4. Comunicación entre el computador y el *Blu*^{2i}.

Al ya saber como controlar los módulos *Blu*^{2i} a través de comandos predeterminados y teniendo una herramienta computacional para enviarle dichos comandos a los módulos, establecemos ahora la conexión inalámbrica entre dos dispositivos Bluetooth. En la **figura 6-5** se muestra paso a paso cuales son los paquetes de datos que se intercambian entre los módulos *Blu*^{2i} y sus respectivos Host para que ellos se logren comunicar.

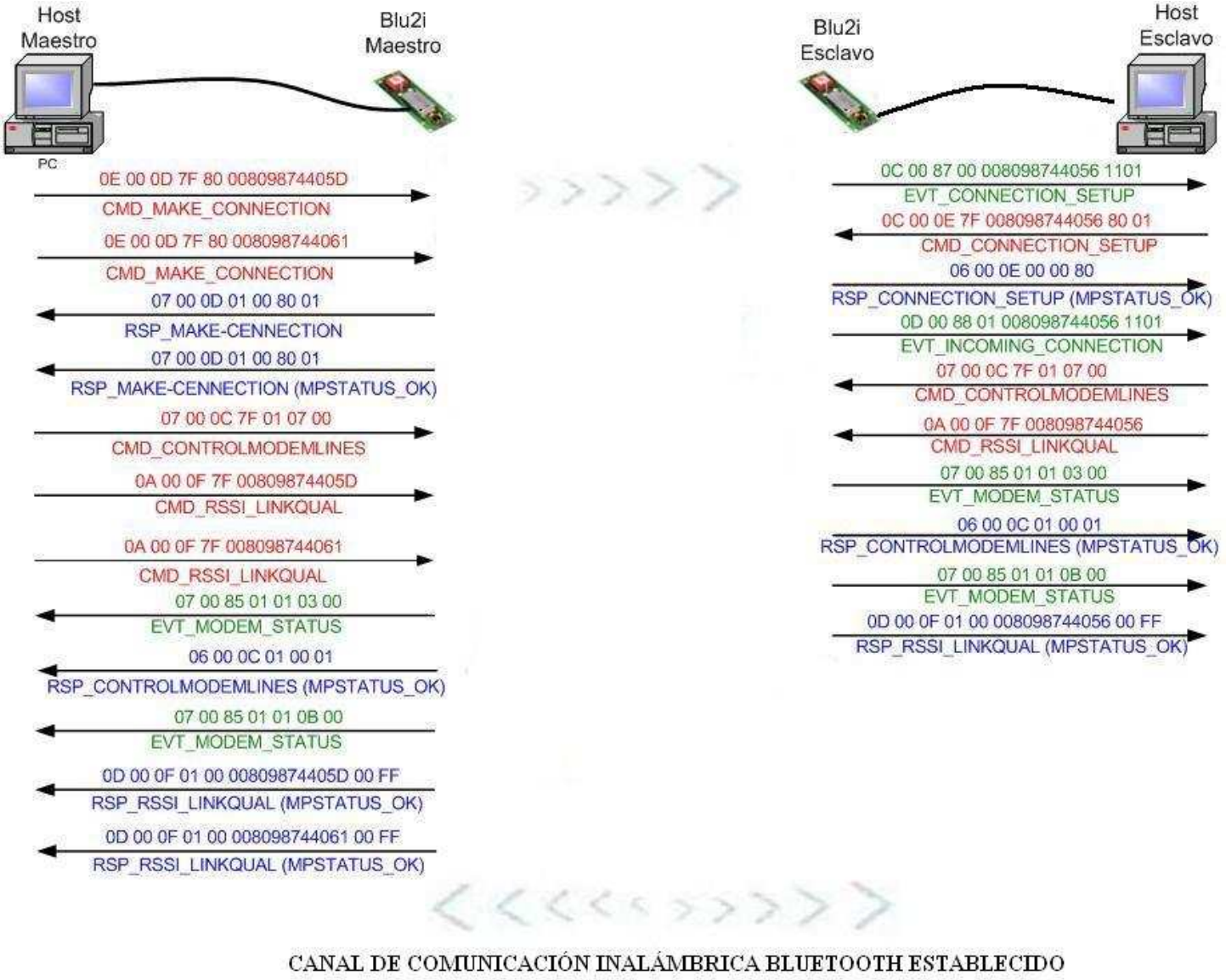

Figura 6-5. Paquetes de datos para establecer comunicación Bluetooth.

Una vez que se establece la comunicación inalámbrica Bluetooth, se pueden transmitir paquetes de datos entre los dos computadores, de la misma forma a como si estuvieran conectados por medio de un cable de comunicación a través de sus puertos serie. La prueba mas sencilla que se realizó fue el envío de cadenas de caracteres introducidos por el teclado del computador y convertidos en forma de paquetes de datos serie, tal y como se muestra en la **figura 6-6**.

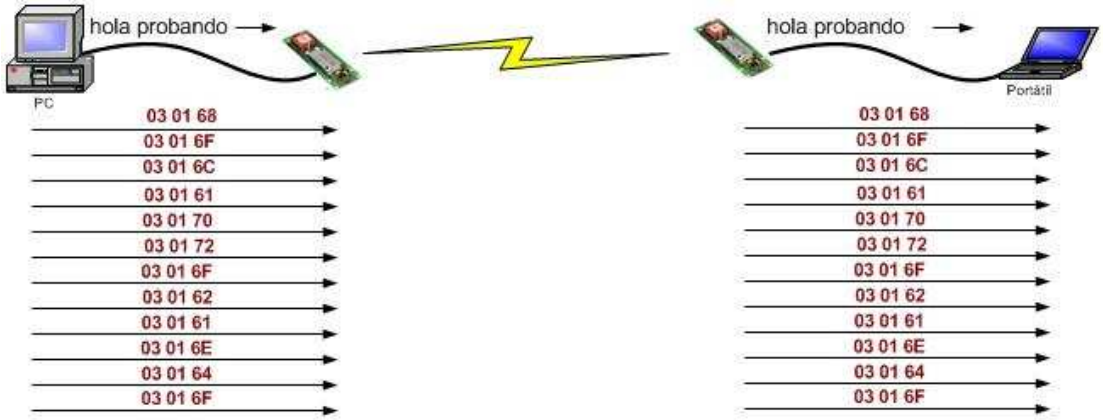

Figura 6-6. Paquetes para envío de caracteres.

Estos paquetes del tipo “03 01 68”, indican lo siguiente:

03: Longitud del paquete.

01: Canal de comunicación asignado al esclavo.

68: Código ASCII del caracter en valor hexadecimal, en este caso el caracter ‘h’.

Se efectuó también la comunicación entre un módulo *Blu*^{2i} y un Microcontrolador PIC, marca Microchip, modelo 16F876A, en forma de paquetes de datos serie. Se descargó en el PIC un software, similar al realizado en C++ para realizar la comunicación entre el puerto COM del computador y el *Blu*^{2i}, elaborado y compilado en el lenguaje de programación CCS para Microcontroladores Microchip, que se asemeja bastante al C++. Para probar la correcta comunicación, el software descargado al PIC, funciona básicamente de la siguiente manera: al recibir el PIC un cambio de tensión en uno de sus terminales, que puede representar la alarma emitida por un sensor de movimiento, debe enviar un comando determinado al *Blu*^{2i} y luego evaluar la respuesta recibida por el módulo. En caso de ser la respuesta y/o evento correspondiente al comando enviado, se percibirá un cambio de tensión en un terminal determinado del PIC, que encenderá un diodo emisor de luz para indicar que el resultado fue correcto.

Se tiene también, que la Unidad de Procesamiento Central del sistema de seguridad existente en la empresa BCCOM Business, maneja la comunicación del tipo serie, a través de un Microcontrolador PIC de la misma marca y modelo del PIC con el cual se realizaron las pruebas anteriores, por lo tanto tenemos que el módulo Bluetooth adquirido por la empresa también es compatible con esta Unidad de Procesamiento Central, que forma parte de uno de los sistemas propuestos.

6.4 CONFIGURACION DE LOS BLU [2I] PARA LA IMPLEMENTACIÓN DE UNA PICORED.

Para establecer la comunicación entre un módulo maestro y varios esclavos al mismo tiempo, se le debe enviar al maestro comandos de solicitud de conexión desde el Host, que en nuestro caso es un PC. Cada comando de solicitud de conexión, debe incluir la dirección Bluetooth del esclavo al que se le pide conexión. Esta dirección se puede obtener haciendo que el maestro realice un proceso de búsqueda (Inquiry) de cuales dispositivos Bluetooth están dentro del rango de cobertura de este dispositivo maestro y cuales se encuentran en modo reconocible para posteriormente hacer la solicitud de conexión. De igual modo si se conoce previamente la dirección Bluetooth de los equipos esclavos con los que se desea establecer comunicación, simplemente se manda desde el Host al maestro el comando de solicitud de conexión adjuntándole la dirección Bluetooth del equipo conocido y se espera la confirmación del establecimiento del canal de comunicación. Este último es el caso que se va a implementar, pues se tiene la dirección Bluetooth de los dos módulos que estarán conectados a los dispositivos remotos y que representarán los dispositivos esclavos, y no se desea que el maestro establezca conexión con algún otro dispositivo Bluetooth que se encuentre dentro de su rango de cobertura que no forme parte del sistema de seguridad que se quiere desarrollar.

También cabe mencionar que al establecer una conexión Bluetooth múltiple, el dispositivo maestro va asignando canales de comunicación a cada esclavo en el orden en que estos van confirmando la aceptación de conexión, es decir se tienen 7 canales de comunicación disponibles para el establecimiento de la picored, de los cuales se asigna el canal 1 al primer esclavo que responde, el canal 2 al segundo y así sucesivamente hasta que todos estén comunicados con el maestro.

Para configurar estos equipos, de modo tal que cumplan las funciones requeridas por nuestra picored formada por un maestro y dos esclavos, se deben manipular ciertos parámetros y registros que poseen los módulos para adecuar su comportamiento conforme lo necesite el sistema a desarrollar. Estos parámetros y registros se configuran enviando los comandos adecuados, en forma de paquetes de datos serie. La manera más sencilla de configurar los equipos, es haciendo uso del pequeño software que se utilizó en la sección 6.3, para establecer la comunicación entre los *Blu*^{2i} y un computador a través de su puerto serie.

En la tabla 6-1 se muestran los parámetros y registros S configurados en los equipos, con sus respectivos comandos, mostrando su nombre y paquete de datos respectivo.

Tabla 6-1. Parámetros y Registros configurados en los Blu2i.

Parámetro o Registro S	**Comando**	**Paquete de Datos**
Modo Conectable	CMD_CONNECTABLE_MODE 1= Enable	06 00 0A 7F 01 01
Modo No Descubrible	CMD_DISCOVERABLE_MODE 0= Disable	05 00 09 7F 00
Modo Seguridad (No Encriptado)	CMD_SECURITY_MODE 0= No Encryption.	05 00 18 7F 00
Coloca todos los	CMD_DEFAULT_SREG	04 00 1C 7F

Registros S por defecto		
S2(Tamaño máximo de los paquetes de datos)	CMD_WRITE_SREG	09 00 04 7F 02 00 02 55 00
S3(Perfil de Puerto Serie)	CMD_WRITE_SREG 0= Serial Port Profile	09 00 04 7F 03 00 00 00 00
S13(Auto aceptar los canales RFCOMM entrantes)	CMD_WRITE_SREG	09 00 04 7F 13 00 00 00 01
S14(Auto aceptar las solicitudes de conexión)	CMD_WRITE_SREG	09 00 04 7F 14 00 00 00 01
Guarda los cambios en la memoria No Volátil de los Blu2i	CMD_STORE_SREG	04 00 05 01

6.5 ELABORACIÓN DE UNA INTERFAZ GRÁFICA PARA LA COMUNICACIÓN ENTRE EL USUARIO Y LOS MÓDULOS BLUETOOTH PARA CONTROLAR LA PICORED INALÁMBRICA.

Se desarrolló una interfaz gráfica, en ambiente Windows, que controla a través del puerto serie de un computador, la configuración y establecimiento de conexiones entre los módulos *Blu*^{2i}, y que adicionalmente permite realizar transferencias de varios tipos de archivos entre varias computadoras de manera inalámbrica, a través de un enlace de radio Bluetooth (Ver Anexos).

Esta interfaz fue elaborada en lenguaje de programación BUILDER C++, en el cual se codifica de igual forma que en el tradicional C++, solo que este lenguaje no se desempeña en ambiente DOS sino es de ambiente Windows.

La interfaz gráfica permite el manejo de los puertos COM de un PC, configurándolo y habilitándolo para la transmisión y recepción de paquetes de datos por medio de rutinas y algoritmos de programación. En la **figura 6-7** se muestra el menú principal de la interfaz elaborada para la transferencia de archivos, vía Bluetooth, entre computadores haciendo uso de los módulos *Blu*^{2i}.

Figura 6-7. Menú principal de la interfaz gráfica elaborada.

Botón Abrir Puerto: al hacer clic en este botón, se habilita la comunicación a través del puerto serie del computador, por medio del cual se realizará la transferencia de archivos. Para la configuración del puerto se debe hacer clic en el botón configurar.

Botón Configurar: se utiliza para elegir el puerto COM que se va a utilizar, así como también para fijar los parámetros de comunicación del puerto, tales como velocidad del puerto, paridad, bits de parada y tipo de control de flujo. Esto se aprecia en la **figura 6-8**.

Figura 6-8. Configuración del puerto COM.

Botón Conectar: Una vez que se abre la comunicación con el puerto COM seleccionado, se habilita el botón de Conectar. Al hacer clic en Conectar, se debe elegir si el módulo se actuará como maestro o esclavo (**ver figura 6-9**). En caso de ser maestro se debe indicar la dirección Bluetooth del dispositivo con el cual se desea conectar. Luego se envía al *Blu*^{2i} los paquetes de comandos necesarios para establecer la comunicación Bluetooth con los esclavos, como se mostró en la sección 6.3, ya que como sabemos el maestro siempre es el que establece la conexión.

Figura 6-9. Configuración del *Blu^{2i}*.

Botón Enviar: Una vez que los módulos esclavos han establecido la comunicación con el maestro, se habilita el botón Enviar. Al hacer clic en este botón, se despliega una ventana o caja de selección, donde se ubica el archivo que se desea transferir para que el programa realice la rutina de dividir el archivo en paquetes de datos y, al hacer clic en Abrir, mandar estos paquetes a través del puerto COM seleccionado.

Transferencia de archivos de texto: .txt, .doc, .xls. Para la transmisión y recepción de archivos de texto, se utilizó el Software desarrollado para tal fin, de modo que al seleccionar el documento que se quiere transmitir y luego hacer clic en el botón Enviar, dicho documento se envíe en forma de paquetes de datos desde el PC hasta el Blu2i, para que este lo mande vía Bluetooth hasta el módulo esclavo al que se

le desee transmitir el archivo. El proceso de selección, envío y recepción de archivos usando el Software desarrollado se ilustra en las **figuras 6-10 y 6-11**.

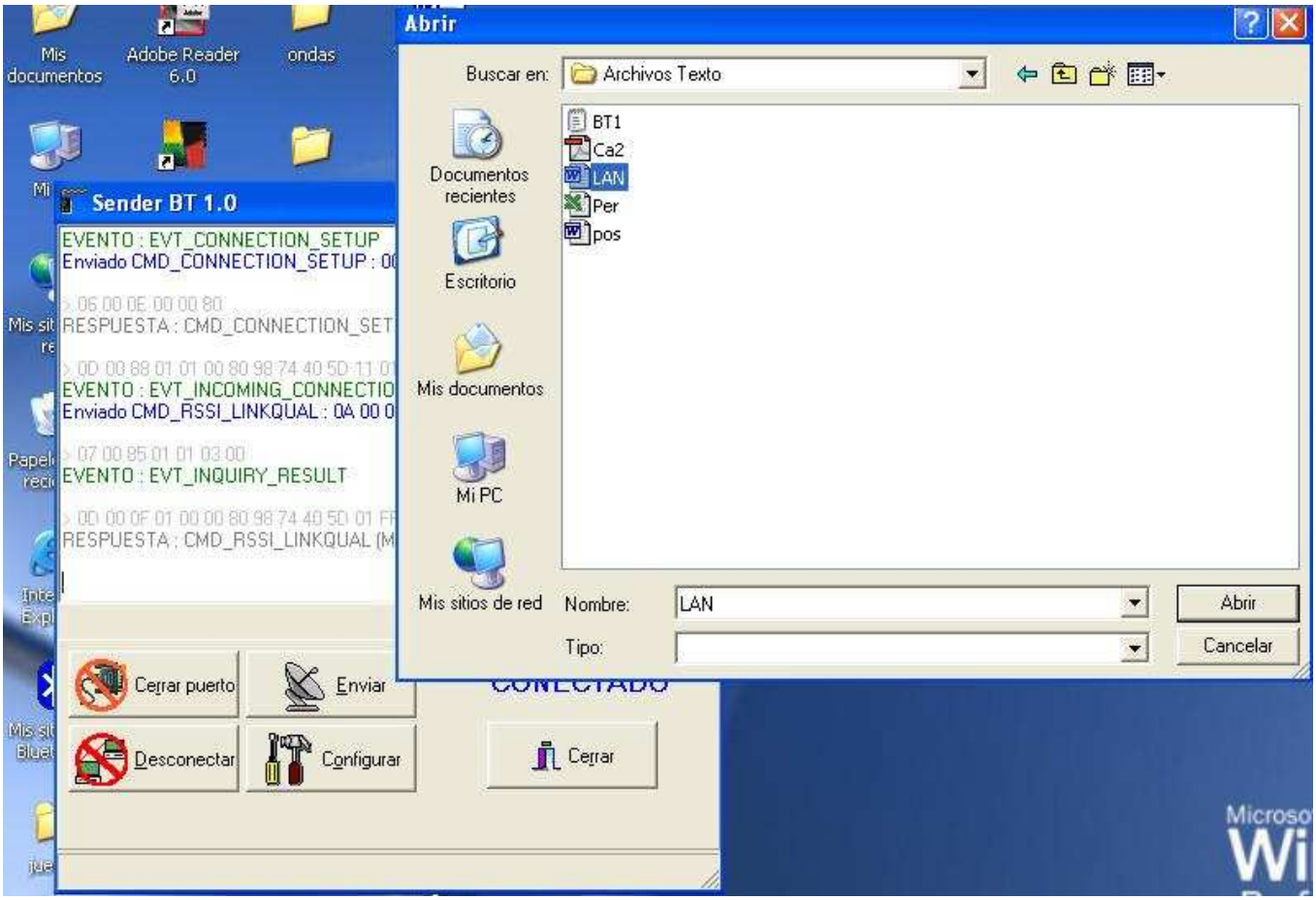

Figura 6-10. Selección y envío de archivos de texto.

Figura 6-11. Recepción de archivos de texto.

Para comprobar la calidad y el tiempo de envío de un archivo de texto, se realizaron varias pruebas de transferencia como se muestra en la **tabla 6-2**.

Tabla 6-2. Pruebas de transferencia de archivos de texto vía Bluetooth.

Tipo de Archivo	Tamaño del archivo enviado (KB)	Tiempo de transferencia (minutos)	Tamaño del archivo recibido (KB)	Fidelidad del archivo recibido respecto al enviado
.txt	5	0.4	5	Copia fiel
.doc	30	0.6	30	Copia fiel
.xls	50	0.9	49	Copia fiel
.pdf	100	1.1	96	Copia incompleta

De los datos que se tienen en la tabla 6-2 se puede concluir que las transmisiones de archivos de texto a través de los módulos *Blu*^{2i} utilizando el puerto serie como interfaz con el computador tienen tiempos de envío un poco largos, pero lo suficientemente buenos para la aplicación que se desea, ya que los mensajes de alarma siempre serán muy cortos, y por ello su tamaño es, por lo general, de 2 a 3 KB.

Transferencia de archivos de imágenes. Utilizando nuevamente la herramienta desarrollada para la transmisión y recepción de archivos, se procedió de manera similar a la expuesta anteriormente, solo que esta vez; luego de hacer clic en el botón enviar, se seleccionó un archivo tipo JPG guardado en el disco duro del PC para ser transmitido en forma de paquetes de datos desde el PC hasta el módulo y de ahí al módulo *Blu*^{2i} de destino, de manera inalámbrica. El proceso de selección, envío y recepción de archivos usando el Software desarrollado se ilustra en las figuras **6-12 y 6-13**.

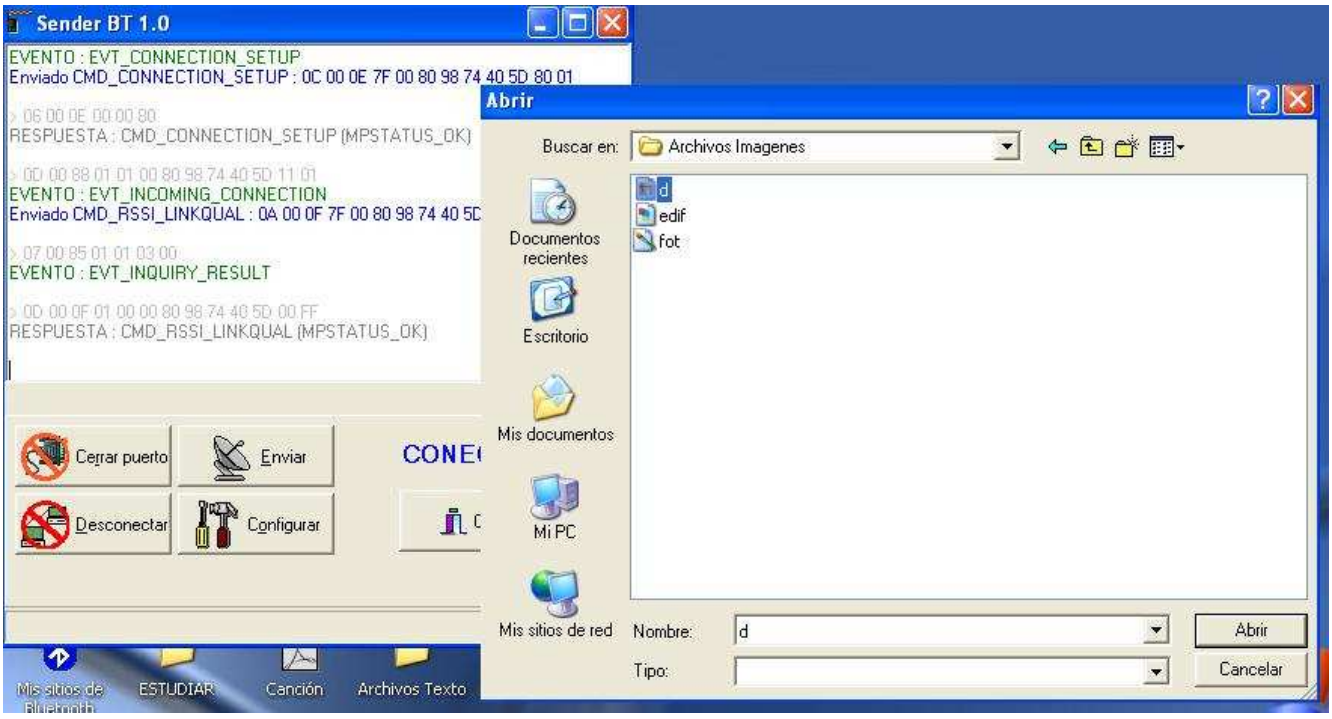

Figura 6-12. Selección y envío de archivos de imagen.

Figura 6-13. Recepción de archivos de imagen.

Se realizaron varias pruebas de transferencia de archivos de imágenes y se descargaron en la tabla 6-3.

Tabla 6-3. Pruebas de transferencia de archivos de imágenes vía Bluetooth.

Tipo de Archivo	Tamaño del archivo enviado (KB)	Tiempo de transferencia (minutos)	Tamaño del archivo recibido (KB)	Fidelidad del archivo recibido respecto al enviado
.jpg	8	0.4	8	Copia fiel
.jpg	60	0.9	59	Copia fiel
.gif	95	1	91	Incompleta
.jpg	100	1.1	98	Copia aceptable

De los datos que se tienen en la tabla 6-3, al igual que en la transferencia de archivos de datos, el tiempo que tarda en completarse la transferencia de una imagen, es un poco mayor en comparación a una transferencia de imágenes hecha a través de un cable. Existe la posibilidad de que el software desarrollado para realizar la transferencia de archivos no se encuentre completamente optimizado o contenga rutinas de programación que retrasen el proceso de envío de paquetes a través del puerto serie, razón por la cual seguirá siendo depurado hasta descartar esta posibilidad. Igual que en el caso anterior, estos tiempos de envío son lo suficientemente buenos para la aplicación de seguridad que se desea implementar.

Botón Desconectar: Al hacer clic en el este botón, se envía al *Blu*^{2i} los paquetes de comandos necesarios para finalizar la comunicación. La comunicación puede ser finalizada desde el maestro hacia uno o ambos esclavos, así como también desde un esclavo hacia el maestro.

Actualmente, la empresa no cuenta con cámaras fotográficas o de video para integrarlas al nuevo sistema que se desea implementar. En tal caso no debería haber problema con alguna que sea capaz de descargar sus imágenes a través de un puerto serie, aunque es muy poco común, ya que las que más abundan en el mercado poseen un cable de comunicación del tipo USB. Por tal razón, si se adquiere un equipo con esta última característica, será necesario tener o diseñar, un procesador de imágenes que las reciba, por ejemplo en formato JPG, a través de un puerto USB, y luego sea convertida en paquetes de datos serie y sea enviada al módulo *Blu*^{2i} a través de un puerto serie. La factibilidad de esta aplicación se ve comprobada en el software de pruebas realizado para la validación de las propuestas; que se describe más adelante, el cual se debe ejecutar en un computador y es capaz de tomar un archivo de imagen del tipo JPG, dividirlo en paquetes de datos serie y enviarlo a través de un puerto COM al que se encuentra conectado un módulo *Blu*^{2i}, que a su vez lo enviará a través de un enlace de radio Bluetooth hasta el módulo *Blu*^{2i} deseado, que se encuentra conectado a otro computador.

6.6 DESARROLLO DE SOFTWARE PARA EL ESTABLECIMIENTO Y CONTROL DE LA PICORED BLUETOOTH.

Para poder implementar la Picored Bluetooth deseada, necesitamos realizar, de manera automatizada, las funciones básicas de la picored que son: establecer, mantener y finalizar la conexión entre los módulos *Blu*^{2i}, en el momento que sea necesario. También sabemos que en la tecnología Bluetooth, el dispositivo maestro es quien se encarga de efectuar las funciones básicas antes mencionadas, salvo la función de finalizar la conexión, que también la puede realizar un dispositivo esclavo, pero para la aplicación a desarrollar, nuestro dispositivo maestro será el único autorizado para finalizar una conexión hacia sus esclavos.

Los requerimientos de Software para la aplicación, serán distintos para el maestro y los esclavos. Nuestro dispositivo maestro estará conectado a un computador y cada esclavo a un microcontrolador PIC distinto, los cuales se encargarán de recibir las activaciones de los detectores o sensores de movimiento que se encuentren conectados a dichos PIC. Por estos motivos se necesita un software principal, que será ejecutado desde el PC donde se encuentra conectado el módulo maestro y que deberá entonces, tener la capacidad de realizar las funciones de conexión, mantenimiento y desconexión de los módulos de manera inalámbrica, a través de la capa de radio que posee el conjunto de protocolos Bluetooth. El maestro debe también poder recibir mientras la conexión Bluetooth esté establecida, recibir mensajes en forma de archivos de texto, que contengan la identificación del sensor de movimiento que haya sido activado, y almacenar el contenido de este archivo de texto, agregándole la fecha y la hora en la cual fue recibido por el dispositivo maestro, en un archivo principal que se encontrará en el computador que controla el sistema de seguridad.

6.6.1 PROGRAMA ELABORADO PARA LOS DISPOSITIVOS ESCLAVOS.

En el caso de los esclavos, como su Host es un Microcontrolador PIC, se desarrolló un pequeño programa, codificado en un computador en lenguaje de programación CCS para Microcontroladores, que es muy similar al lenguaje C++, y descargado posteriormente al PIC, cuyo funcionamiento consta de lo siguiente: en primer lugar, al recibir el evento que indica que el dispositivo maestro (lo reconoce por la dirección Bluetooth que el maestro envía junto al evento) y el esclavo intentan establecer la conexión, empieza a enviar los comandos necesarios, en forma de paquetes de datos serie, al módulo *Blu*^{2i} conectado a él, para aceptar el establecimiento del canal de comunicación. Luego de tener una comunicación establecida, se encuentra a la espera de que los terminales del microcontrolador que se encuentran unidos a un sensor de movimiento, sufran un cambio de tensión. En este momento se envía la cabecera de un archivo de texto establecido con anterioridad, ya segmentado en paquetes de datos del tipo serie, que contiene un mensaje de identificación acerca del sensor que ha sido activado. Una vez enviados todos los paquetes, se espera un tiempo determinado para recibir un mensaje o palabra clave que indica la recepción satisfactoria del archivo y entra nuevamente en el estado de espera de activación de algún sensor conectado a él. En el caso en que se interrumpa el enlace de radio con el módulo maestro, el PIC realiza un proceso de Power on Reset, que devuelve el programa al estado de espera de solicitud de conexión por el módulo maestro. En la **figura 6-13** se muestra un pequeño diagrama de cómo funciona el programa descargado sobre los PIC.

6.6.2 SOFTWARE DESARROLLADO PARA EL MÓDULO MAESTRO.

Se realizó una extensión de la interfaz gráfica elaborada en la sección 6.3 (Ver anexos), codificado en lenguaje de programación BUILDER C++, que contiene

rutinas de programación que nos permiten la conexión, envío de archivos a través del enlace Bluetooth establecido y finalizar la conexión. En esta parte del trabajo se explicará con más detalle el funcionamiento completo del software (incluyendo las rutinas de programación que fueron agregadas) que forma parte primordial en la implementación de nuestra Picored Bluetooth.

Como este software automatiza las acciones a realizar por el dispositivo maestro, lo primero que debe hacer es establecer la comunicación con los dos dispositivos esclavos que se tienen, para así abrir los canales de comunicación inalámbrica Bluetooth y tener la Picored en funcionamiento.

Una vez establecida la comunicación Bluetooth, el programa se prepara para recibir la cabecera de un archivo de texto, que proporciona la información del tamaño total que ocupa en el disco duro tal archivo, la cantidad de paquetes de una longitud máxima de 255 bytes en los cuales fue dividido el archivo, y posteriormente se dispone a recibir estos paquetes de datos, va reensamblando el archivo de texto y lo guarda en un directorio del disco duro predeterminado. Luego de haber recibido el archivo completo, el software lo abre, le adjunta al final del texto la fecha y la hora que tiene la computadora en ese momento y vuelve a cerrar al archivo. Una vez completada esta operación, el programa envía un conjunto de caracteres clave al esclavo que le mandó el archivo, con lo cual le confirma que el archivo se recibió correctamente y fue guardado satisfactoriamente. En caso de que el esclavo no reciba esta palabra clave de envío satisfactorio después de un tiempo determinado, reenviará el archivo de texto una y otra vez hasta recibir esta respuesta de confirmación. Esta palabra clave de confirmación sirve a su vez para el caso poco común de que se activen dos sensores de movimiento al mismo tiempo, ya que el maestro recibirá uno primero, cumpliendo exactamente con el procedimiento anterior, y el segundo esclavo, al no recibir la palabra clave en el período de tiempo establecido, reenviará el archivo de manera repetida, mientras que el maestro termina de recibir el primer archivo. Una vez que la recepción se complete y el maestro vuelva al estado de

recepción de archivos, comenzará la recepción del segundo archivo, que se estaba enviando de manera reincidente, y una vez culminado el proceso mandará la palabra clave para que el esclavo no siga reenviando el archivo. Claro está que el archivo que llegó de segundo, tendrá la hora de recibido con un cierto retraso de tiempo que no supera los 90 segundos.

El software también controla o supervisa el estado de la conexión comparando todos los paquetes de datos que recibe el Host maestro provenientes del *Blu*^{2i} conectado a él, con el evento llamado EVT_DISCONNECT que no es más que un paquete de datos preestablecido que indica el número del canal que ha sido interrumpido. En caso de ocurrir esto, se llama nuevamente al procedimiento que establece la conexión con un módulo esclavo, pasándole como parámetro el número del canal que fue suministrado por el evento de desconexión.

La función final del software es la de finalizar la conexión de manera correcta, a través de un botón colocado en el formulario principal, para culminar la actividad de la Picored, una vez que se vaya a dejar de utilizar el sistema de seguridad. En la **figura 6-14** se muestra un diagrama de flujo sencillo para visualizar el funcionamiento de este software.

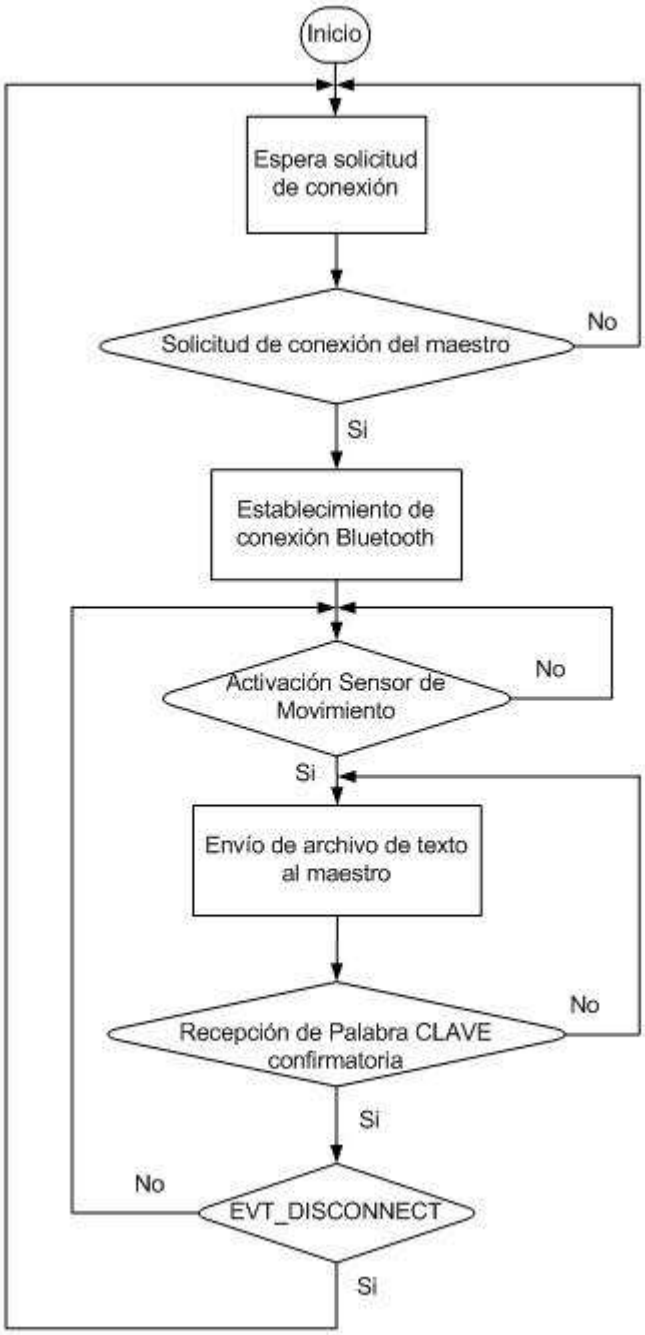

Figura 6-13. Programa que controla los dispositivos esclavos.

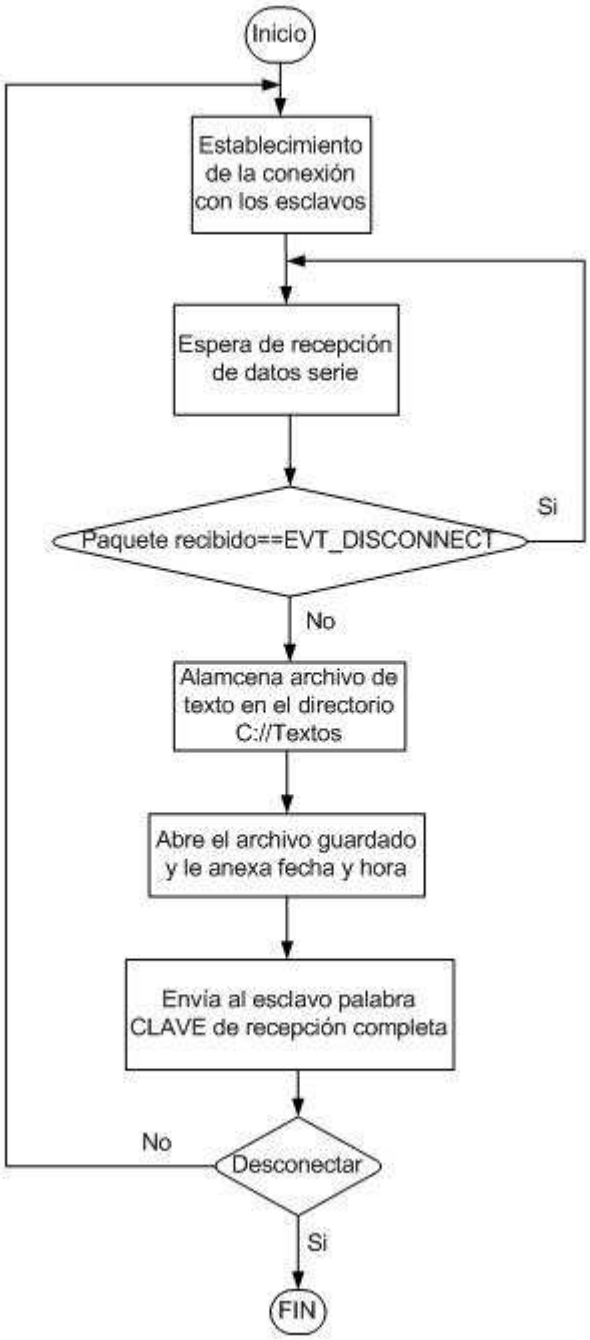

Figura 6-14. Diagrama de flujo del Software para el módulo maestro.

6.7 IMPLEMENTACIÓN DE LA PICORED BLUETOOTH INTEGRANDO LOS DISPOSITIVOS DEL SISTEMA DE SEGURIDAD PROPUESTO.

Luego de satisfacer los requerimientos de Hardware y Software para el establecimiento de una Picored Bluetooth, nos disponemos a integrar todos los dispositivos que formarán parte del sistema de seguridad a ser instalado. A continuación se muestra por separado los tres módulos Bluetooth, con los respectivos equipos a los cuales se encuentran conectados para conformar el sistema de seguridad completo.

6.7.1 DISPOSITIVO MAESTRO INCORPORADO AL SISTEMA DE SEGURIDAD.

Se tiene que, el módulo Bluetooth maestro y su respectivo Host se encuentran integrados de la siguiente forma:

- Primero tenemos un computador Pentium 4, con el software elaborado en la sección 6.6.2 instalado en su disco duro, para ser ejecutado desde el sistema operativo Windows XP.

- Conectado al puerto COM 1 del computador, a través de un cable serie con conector DB9, se tiene la tarjeta de desarrollo que funciona de interfaz entre el computador y el *Blu*^{2i} maestro.

- La tarjeta de desarrollo se encuentra energizada con una fuente de 6 V DC, que ofrece una corriente máxima de salida de 2A.

En la **figura 6-15** se aprecia el sistema maestro integrado.

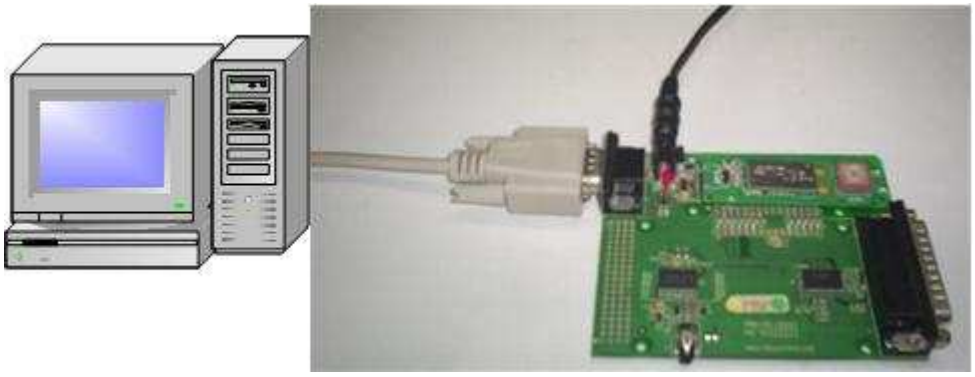

Figura 6-15. Sistema maestro integrado.

6.7.2 DISPOSITIVOS ESCLAVOS INCORPORADOS AL SISTEMA DE SEGURIDAD.

Los dos módulos esclavos que se tienen, se integran al sistema como dispositivos remotos de supervisión de los accesos de entrada y salida para una propiedad privada. Estos se encuentran integrados como se describe a continuación:

- En primer lugar se cuenta con tres detectores de movimiento, de los cuales se conectaron dos a un dispositivo esclavo y uno al segundo dispositivo esclavo.

- Los PIC, que han sido programados como se indica en la sección 6.6.1, poseen un puerto de comunicación serie que, por medio de la interfaz mostrada en la sección 6.2, conecta sus terminales correspondientes a un conector del tipo DB9.

- Se conectó luego la interfaz elaborada en la sección 6.1 donde se coloca el *Blu*^{2i} esclavo, a través de un conector serie DB9, a la interfaz donde se encuentra el PIC programado, y ambas tarjetas se energizaron con una fuente de 6V DC, con una corriente máxima de salida de 500mA.

En la **figura 6-16** se visualiza uno de los dispositivos esclavos de supervisión remota con todas sus partes incorporadas.

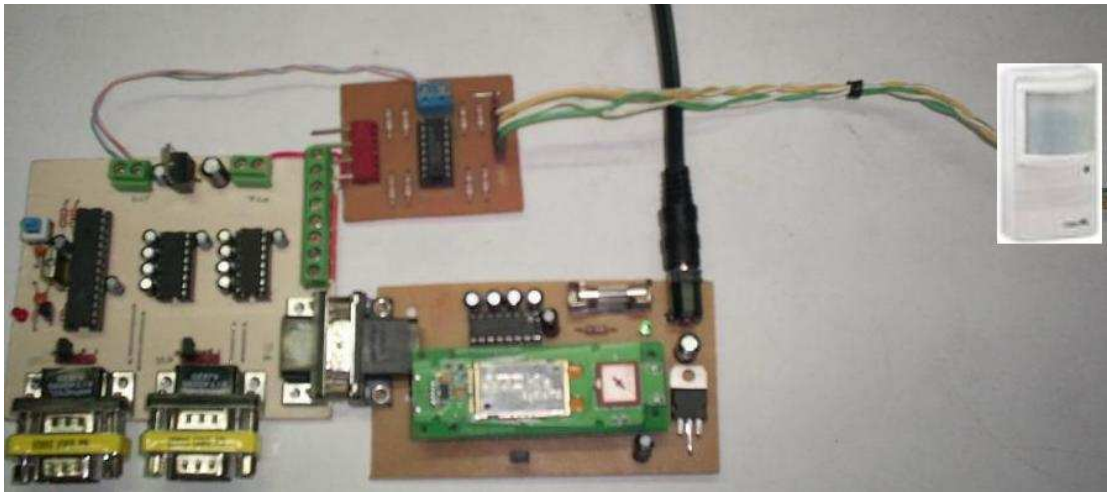

Figura 6-16. Dispositivos esclavos de supervisión remota

6.8 PRUEBAS DE OPERATIVIDAD DEL SISTEMA COMPLETO, CONFORMADO POR UNA PICORED LOCAL BLUETOOTH.

Como parte final de este trabajo de grado, se instaló el sistema de seguridad completo, conformado por una picored local inalámbrica Bluetooth que consta de un maestro y dos esclavos, donde el dispositivo maestro se encuentra conectado a un computador que contiene el software de gestión del sistema de seguridad, y los dispositivos esclavos se encuentran en localidades remotas, conectados a microcontroladores PIC. Éstos reciben las activaciones de sensores de movimiento, ubicados en los accesos de entrada y salida de la instalación privada donde se desee colocar el sistema diseñado.

Para realizar las pruebas de aceptación del sistema completo y la validación de la comunicación, se instaló el sistema prototipo en la empresa BCCOM Business, ejecutando el software del dispositivo maestro desde un computador ubicado en la oficina principal de la empresa y colocando detectores de movimiento, junto al

acceso de entrada y salida de vehículos, y otros dos en la puerta de entrada y salida de personal. En la **figura 6-17** se visualiza un plano de vista superior de la empresa, resaltando los puntos donde se encuentran ubicados los módulos *Blu*^{2i}.

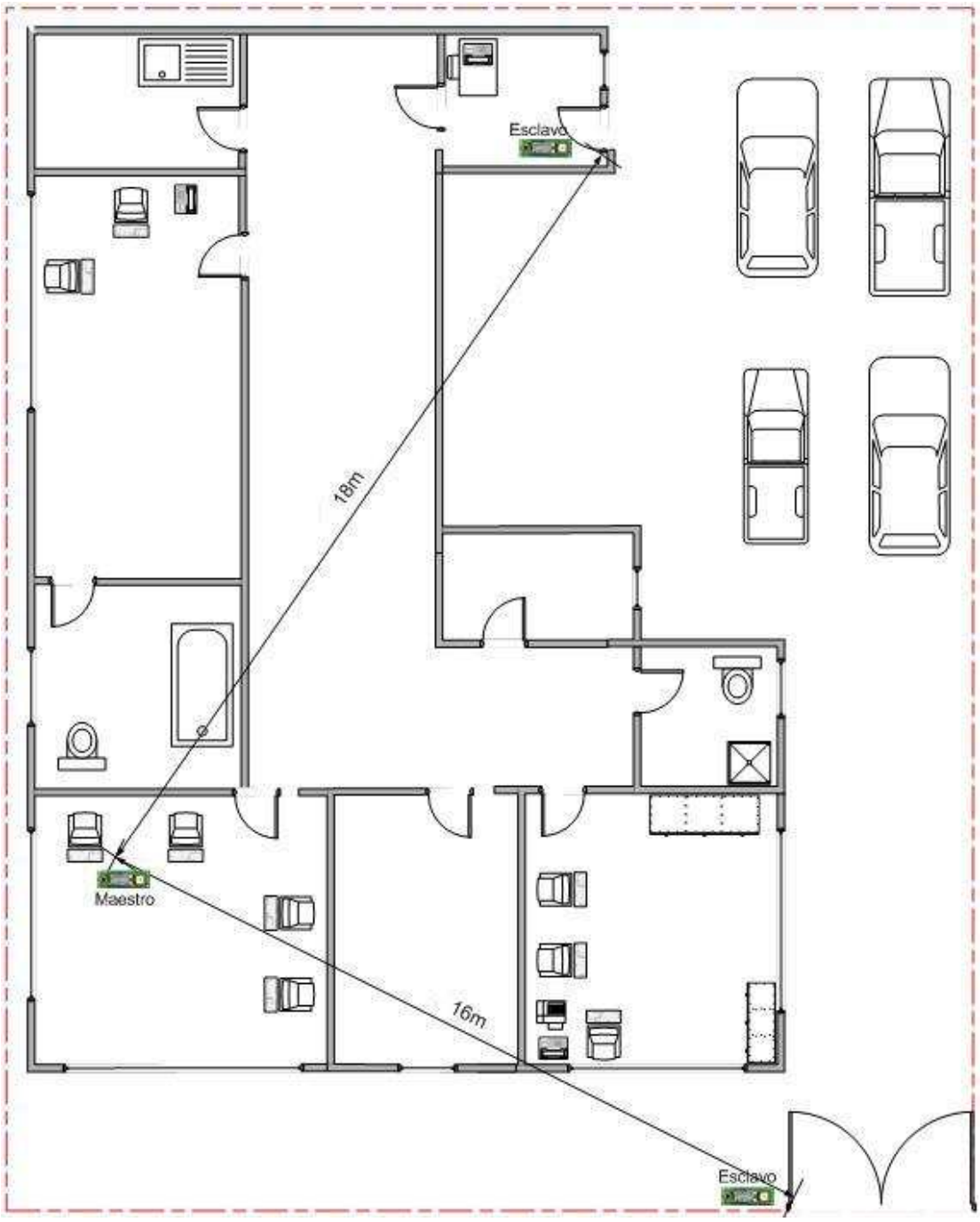

Figura 6-17. Plano vista superior de BCCOM Business con la picored implementada.

Se tiene la ventaja en este caso de que la distancia entre el lugar donde se encuentra el PC al cual está conectado el dispositivo maestro y los accesos de entrada y salida de la empresa, son bastante cerca, por lo que el rango de cobertura de los módulos no se ve afectado, sino por el contrario el sistema funciona de manera correcta y poco forzado.

6.8.1 VALIDACIÓN DE LA COMUNICACIÓN INALÁMBRICA A TRAVÉS DE LA PICORED.

Recordando que los dispositivos conectados a sensores de movimiento en el sistema implementado, al activarse de ellos, éste sufre un cambio de estado en su salida que es enviado al Microcontrolador PIC, que tiene un software, diseñado con antelación, y grabado sobre él con la finalidad de que, al percibir el cambio de estado en uno de sus terminales que se encuentra conectado a algún sensor, este realice una rutina encargada de identificar cual sensor sufrió un cambio de estado y enviar, a través de sus terminales de transmisión y recepción de datos bajo el estándar RS232, un mensaje o un archivo de texto almacenado en su memoria, al módulo *Blu*^{2i} esclavo para que este transfiera tal archivo al módulo maestro conectado al PC que se encarga en ese momento de administrar el sistema de seguridad.

Para realizar entonces la validación de este dispositivo de supervisión remoto, con un computador portátil con el software desarrollado instalado y conectado a un *Blu*^{2i} esclavo, se transmitió un archivo de texto con un mensaje que identifica el dispositivo que fue activado y el acceso que ha sido utilizado. Se probó entonces el envío de este archivo cambiando de ubicación el dispositivo maestro, observando que en lugares con línea de vista entre los módulos, se obtuvo una buena y rápida comunicación, hasta cercanos los 90 metros de distancia. Luego en recintos cerrados, entre dos pisos de distancia, donde la línea de vista entre los módulos se encuentra altamente obstaculizada, la comunicación es eficiente hasta unos 25 metros de distancia.

Luego, se concluye que, dependiendo del lugar donde se instalen los dispositivos remotos a distancias considerables de su módulo maestro, se deben probar varios puntos de ubicación de los módulos donde se tenga la menor cantidad de obstáculos posible, desechando por completo objetos metálicos grandes, que puedan encontrarse entre la línea directa del maestro y uno de sus esclavos, ya que la proximidad de los *Blu*^{2i} a este tipo de objetos puede afectar el rango y el funcionamiento del sistema, puesto que interfiere con las ondas electromagnéticas que se emiten desde la antena del dispositivo Bluetooth, así como las ondas que quieren llegar hasta él, provenientes de otro dispositivo Bluetooth.

Después de realizar estas pruebas, los ajustes finales se realizaron principalmente en la ubicación exacta de los dispositivos esclavos y del dispositivo maestro, tomando en cuenta la línea de vista entre los equipos y la distancia prudente con respecto a objetos largos metálicos.

En el software de gestión del sistema también se realizaron varios ajustes de subrutinas de control del equipo maestro para mejorar la comunicación y tomar precauciones en caso de que los *Blu*^{2i} pierdan la configuración inicial, como se dio el caso en una oportunidad.

EVALUACIÓN DEL PROYECTO

Para recomendar la aprobación de este proyecto, donde es necesario realizar una inversión monetaria, vamos a hacer el estudio de tres viabilidades principales, entendiendo por viabilidad la "posibilidad de" o la "conveniencia de" realizar el proyecto: la viabilidad técnica, legal y económica.

VIABILIDAD TÉCNICA

La viabilidad técnica, que siempre debe establecerse con la ayuda de los técnicos especializados en la materia, busca determinar si es posible física o materialmente hacer un proyecto. Esto se toma en cuenta antes de empezar el proyecto de forma generalizada, pero no es sino en el transcurso del desarrollo donde se puede comprobar y verificar que el proyecto sí se puede elaborar con éxito, por lo menos a nivel de instalación.

En nuestro caso esta viabilidad se ve reflejada principalmente en el capítulo V, donde se realizan las primeras pruebas físicas con los dispositivos Bluetooth adquiridos. Se puede decir que esto empieza a verse claramente en el momento en que se logra establecer comunicación entre un módulo Blu^{2i} y un computador a través del puerto COM serie, utilizando la interfaz adecuada para ello. Claro está que al momento de hacer el estudio de cuales equipos serán adquiridos para el desarrollo del proyecto (capítulo IV), se tomaron en cuenta todos los datos que proporciona el fabricante, haciendo énfasis en los puertos de comunicación que los equipos manejan y los productos que ellos ofrecen para facilitar esto, como por ejemplo la tarjeta de desarrollo mostrada en la sección 4.4.

Finalmente, teniendo unos buenos resultados en las pruebas de operatividad del sistema de seguridad y vigilancia propuesto, así como el desarrollo exitoso de una interfaz para la comunicación entre los dispositivos esclavos y sus respectivos Host

(sección 6.1), se puede confirmar que, con todos los equipos que se tienen y la compatibilidad que hay entre ellos, es posible realizar físicamente una implementación de una Picored Local Inalámbrica basada en la tecnología Bluetooth que se adapte a las necesidades de un sistema de seguridad y vigilancia propuesto, y que para ello se necesita el desarrollo de un software de aplicación para el control del dispositivo maestro de la picored y otro software menor para el control de los dispositivos esclavos, que en ambos casos se sabe que es posible lograrlo creando rutinas y algoritmos de programación que manipulen y controlen el intercambio de información entre los equipos a través del puerto de comunicación serie.

VIABILIDAD LEGAL

La viabilidad legal se refiere a la necesidad de determinar la inexistencia de trabas legales para la instalación y operación del proyecto, por lo que se deben definir el marco de restricciones legales que enfrentará dicho proyecto.

Para el desarrollo de nuestro proyecto, que se basa en la implementación de una picored inalámbrica, la única restricción legal que se podría presentar es el uso del espectro de frecuencia, para lo cual, a nivel mundial se requiere autorización de un ente legal que se encarga de la administración de espectro. En nuestro país el ente encargado es la Comisión Nacional de Telecomunicaciones (CONATEL), y son quienes otorgan los permisos legales para el uso de una determinada frecuencia en una determinada zona geográfica del territorio nacional.

Ahora bien, esta restricción y posible traba se encuentra solucionada en nuestro caso gracias a que la tecnología Bluetooth trabaja en la banda de frecuencia libre de 2.4GHz, la llamada ISM (Industrial, Científica y Médica), que se encuentra disponible en la mayor parte del mundo, como se menciona en la sección 1.3, donde también se cita textualmente como CONATEL reglamenta el uso de la frecuencia a la

cual los equipos Bluetooth trabajan, y por consiguiente la que se utilizará en la implementación del proyecto.

VIABILIDAD ECONÓMICA

En el estudio de la viabilidad económica se pretende definir, mediante la comprobación de beneficios y costos estimados de un proyecto, si es recomendable su implementación y posterior operación. Esto incluye también la evaluación económica que identifica los méritos propios del proyecto, independientemente de la manera como se obtengan y se paguen los recursos financieros que se necesiten y del modo como se distribuyan los excedentes o utilidades que genera. Los costos y beneficios constituyen el flujo económico.

Los indicadores más utilizados para realizar la evaluación económica de un proyecto son: valor actual neto, tasa interna de retorno, coeficiente beneficio costo y periodo de recuperación.

VALOR ACTUAL NETO (VAN): Consiste en actualizar a valor presente los flujos de caja futuros que va a generar el proyecto, descontados a un cierto tipo de interés (“la tasa de descuento”), y compararlos con el importe inicial de la inversión. Como tasa de descuento se utiliza normalmente el costo de oportunidad del capital (COK) de la empresa que hace la inversión.

$$VAN = -A + [FC1 / (1+r)^1] + [FC2 / (1+r)^2] +….+[FCn / (1+r)^n]$$

Siendo:
A: desembolso inicial
FC: flujos de caja
n: número de años (1,2,...,n)
r: tipo de interés ("la tasa de descuento")

1/(1+r)^n: factor de descuento para ese tipo de interés y ese número de años

Si VAN > 0: El proyecto es rentable.

Si VAN = 0: El proyecto es postergado.

Si VAN < 0: El proyecto no es rentable.

A la hora de elegir entre dos proyectos, elegiremos aquel que tenga el mayor VAN. Este método se considera el más apropiado a la hora de analizar la rentabilidad de un proyecto.

El valor de A o desembolso inicial en nuestro proyecto lo calculamos en la **tabla 7.1.**

Tabla 7.1. Valor de desembolso inicial del proyecto

CANTIDAD	CONCEPTO	MONTO (2150 Bs/$)
1	TDK Blu2i serie + MDK+ USB Go Blue	225$ (483.750 Bs.)
2	Módulos Blu2i	212$ (455.800 Bs.)
3	Expansión Boards	52.5$ (112.875 Bs.)
2	Tarjeta de circuito impreso	69.76$ (150.000 Bs.)
3	Sensor movimiento	104.65$(225000)
2	PIC 16F87A	23.25$(50000)
	TOTAL	**687.17$ (1.477.425 Bs.)**

El valor de los flujos de caja será para nosotros, el valor estimado en bolívares al que se aspira obtener como ganancia anualmente por el uso de un sistema de seguridad y vigilancia conformado por una picored Bluetooth, al salir al mercado. Esta ganancia estimada o flujo de caja que tomaremos es de 1.000.000 bolívares anuales.

Siendo un poco ambiciosos, tomaremos el valor de n = 4 años, aspirando a que la tecnología Bluetooth se innovadora todavía para esa época. Luego nuestra tasa de descuento será el costo de oportunidad del capital (COK) de la empresa BCCOM

Business que oscila entre el 35%. En la tabla 7.2 se aprecia el cálculo del VAN para nuestro proyecto.

Tabla 7.2 Cálculo del Valor Actual Neto

Año	Costos y Beneficios	Factor de Descuento	Flujo Actualizado
0	-1.477.425	$1 / (1+0.35)^0$	-1.477.425
1	1.000.000	$1 / (1+0.35)^1$	740.741
2	1.000.000	$1 / (1+0.35)^2$	548.697
3	1.000.000	$1 / (1+0.35)^3$	406.454
4	1.000.000	$1 / (1+0.35)^4$	301.067
VAN			**519.534**

TASA INTERNA DE RETORNO (TIR): Se define como la tasa de descuento o tipo de interés que iguala el VAN a cero, es decir, se efectúan tanteos con diferentes tasas de descuento consecutivas hasta que el VAN sea cercano o igual a cero y obtengamos un VAN positivo y uno negativo.

Si TIR > tasa de descuento (r): El proyecto es aceptable.
Si TIR = r: El proyecto es postergado.
Si TIR < tasa de descuento (r): El proyecto no es aceptable.
Este método presenta más dificultades y es menos fiable que el anterior, por eso suele usarse como complementario al VAN.

Tabla 7.3 Tasa Interna de Retorno

Año	Costos y Beneficios	Factor Actualizado r = 56%	Flujo Actualizado r = 57%
0	-1.477.425	-1.477.425	-1.477.425
1	1.000.000	641026	636943
2	1.000.000	410914	405696
3	1.000.000	263406	258405
4	1.000.000	168850	164589
VAN		6.771	-11.792
TIR = 56% + 6771 / (6771 + 11792) = 56.36%			

COEFICIENTE BENEFICIO COSTO (BC): Se obtiene con los datos del VAN; cuando se divide la sumatoria de todos los beneficios entre la sumatoria de los costos.

Si BC > 1: El proyecto es aceptable.

Si BC = ó cercano a 1: El proyecto es postergado.

Si BC < 1: El proyecto no es aceptable.

Tabla 7.4 Coeficiente Beneficio Costo

Año	Costos y Beneficios	Beneficios Actualizados r = 35%	Costos Actualizados r = 35%
0	-1.477.425		1.477.425

1	1.000.000	740.741	
2	1.000.000	548.697	
3	1.000.000	406.454	
4	1.000.000	301.067	
VAN		1996959	1.477.425
BC = 1.996.959 / 1.477.425 = 1,35			

PERIODO DE RECUPERACION (PR): Se define como el período que tarda en recuperarse la inversión inicial a través de los flujos de caja generados por el proyecto. La inversión se recupera en el año en el cual los flujos de caja acumulados superan a la inversión inicial. Se efectúa por tanteos utilizando los valores del VAN hasta obtener un valor negativo y uno positivo. No se considera un método adecuado si se toma como criterio único. Pero, de la misma forma que el método anterior, puede ser utilizado complementariamente con el VAN.

Tabla 7.5 Período de Recuperación

Año	**Costos y Beneficios**	**Valores Actualizados t = 2 años**	**Valores Actualizados t = 3 años**
0	-1.477.425	-1.477.425	-1.477.425
1	1.000.000	740.741	740.741
2	1.000.000	548.697	548.697
3	1.000.000		406.454
4	1.000.000		
VAN		-187.987	218.467
PR = 2 + 187.987 / (187.987 + 218.467) = 2.46 años			

RECOMENDACIONES

- Hacer un estudio detallado del funcionamiento de una Scatternet Bluetooth y de los pasos que se deben seguir para su implementación.
- Investigar sobre los avances regulares de la tecnología Bluetooth, tal como la recién aparición del Bluetooth versión 2.
- Estudiar el funcionamiento de los módulos *Blu*^{2i} de TDK Systems en cuanto a transmisiones de voz se refiere usando los terminales para modulación PCM que poseen estos módulos.
- Intentar desarrollar la interfaz necesaria para establecer una conexión con el *Blu*^{2i}, a través de un puerto USB de un computador, estudiando los terminales que poseen estos módulos para este tipo de comunicación.
- Explorar los terminales GPIO (General Port In Out) y SPI (Serial Port Interface) del módulo *Blu*^{2i} y realizar un estudio de las posibles aplicaciones reales que con ellos se pueden implementar.
- Realizar pruebas de transmisión de video a través de los dispositivos *Blu*^{2i}, evaluando su eficiencia y limitaciones, con el fin de poder integrarse cámaras de video dentro de un sistema de seguridad y vigilancia que maneje comunicación inalámbrica Bluetooth. .

CONCLUSIONES

- Bluetooth es una solución inalámbrica que ha sido lo suficientemente prometedora como para obtener el apoyo de una amplia base de fabricantes que representen todos los segmentos de los mercados de informática y comunicaciones.
- A través del algoritmo diseñado para el desarrollo de una aplicación basada en la tecnología Bluetooth, se siguió un orden correcto para llegar a la culminación del proyecto, ayudándonos a ver de manera clara y desglosada las etapas que se necesitaban superar en la realización del trabajo, y cuando se presentaron problemas, encontrar el posible punto de traba donde se originó la dificultad.
- Los módulos *Blu*^{2i} de TDK, poseen gran facilidad para manipular sus terminales y por consiguiente es sencillo establecer las conexiones físicas con otros dispositivos electrónicos, así como también el fabricante nos ofrece una gran cantidad de información técnica acerca de los módulos e igualmente un buen asesoramiento técnico por parte de sus especialistas.
- Luego de elaborar y validar la factibilidad de la propuesta para un sistema de seguridad y vigilancia para instalaciones privadas basada en una picored Bluetooth, se concluye que esta nueva tecnología inalámbrica admite usos alternativos, distintos a aquellos para los que fue concebida inicialmente y se podría sacar un buen provecho dentro de aplicaciones industriales y domésticas.
- En la implementación de una Picored Bluetooth, adaptada hacia una utilidad determinada, se presentan una gran cantidad de detalles en los requerimientos de Hardware y especialmente de Software que deben ser solventados minuciosamente y con gran dedicación de tiempo, para lograr su correcto funcionamiento.

- La evaluación del proyecto realizada, arrojó teóricamente muy buenos resultados en cuanto a su viabilidad técnica, legal y económica, concluyendo de esta última, por tres métodos diferentes de evaluación, que el proyecto es económicamente rentable.

REFERENCIAS BIBLIOGRÁFICAS

[1] BLUETOOTH SPECIAL INTEREST GROUP. Disponible en Internet: <URL: http://www.bluetooth.com>.

[2] COMISIÓN NACIONAL DE TELECOMUNICACIONES. Cuadro Nacional de Atribución de Bandas de Frecuencias.doc [Archivo de Word], Disponible en Internet: <URL: http://www.conatel.gov.ve/ >.

[3] BLUETOOTH SPECIAL INTEREST GROUP. "Bluetooth Core", Specification of the Bluetooth System, Versión 1.1, 22 de Febrero de 2003. Disponible en Internet:< URL: http:// www.bluetooth.com/dev/specifications.asp >.

[4] BLUETOOTH SPECIAL INTEREST GROUP. "Bluetooth Profiles", |Specification of the Bluetooth System, Versión 1.1, 22 de Febrero de 2003. Disponible en Internet: < URL: http:// www.bluetooth.com/dev/specifications.asp >.

[5] BONANATA, Maximiliano. "Programación y Algoritmos". M P Ediciones S.A. Buemos Aires- Argentina. Agosto 2003.

[6] DIGITAL EQUIPMENT CORPORATION, INTEL CORPORATION, XEROX CORPORATION. The Ethernet – A Local Area Network, Versión 1.0. Septiembre de 1980.

[7] LEJED, María. "Interfaces RS232 y V.35", guía de estudio de la asignatura Tecnologías de Acceso de Última Milla. UCV. Caracas, Noviembre de 2004.

[8] PAREDES, Alejandro. Trabajo de Grado Titulado “Implementación y Desarrollo de una Aplicación de Seguridad mediante el Servicio de Mensajería Corta de Operadoras Celulares”. Escuela de Ingeniería Eléctrica UCV. Caracas, Julio 2005.

[9] PÉREZ, Joffre. “BLUETOOTH DESCRIPCIÓN DEL ESTÁNDAR”, Revista Electrónica. Facultad de Ingeniería UCV, Diciembre 2000.

BIBLIOGRAFÍA

ANDER-EQQ, Ezequiel. “Introducción a la planificación “. El Cid Editor. Buenos Aires – Argentina. Abril 1978

ARIAS ODON, Fidias. “Tesis & Proyectos de Investigación” Editorial Episteme. Caracas – Venezuela. Mayo 1998

BAKKER, Dee y MCMICHAEL Diane. “Bluetooth End to End”. Primera edición. Enero 2002

BONANATA, Maximiliano. “Programación y Algoritmos”. M P Ediciones S.A. Buenos Aires- Argentina. Agosto 2003.

HARTE, Lawrence. “Introduction to Bluetooth Technology: Market, Operation, Profiles, and Services”. Enero 2004.

MORROW, Robert. “Bluetooth: Operación y Uso”. McGraw-Hill. Primera edición. Junio 2002.

PÉREZ, Joffre. “Bluetooth Descripción del Estándar”, Revista Electrónica. Facultad de Ingeniería UCV, Diciembre 2000.

ANEXO

CÓDIGO FUENTE DEL SOFTWRE QUE CONTROLA LA PICORED BLUETOOTH.

El Software elaborado consta de varios módulos programados por separado que luego son llamados o vinculados al módulo principal o **main.** Se muestran a continuación cada uno de los módulos programados.

Sender BT.cpp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("uMain.cpp", frmPrincipal);
USEFORM("uConfigurar.cpp", frmConfigurar);
USEFORM("uMS.cpp", frmMS);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->Title = "SenderBT";
                Application->CreateForm(__classid(TfrmPrincipal), &frmPrincipal);
                Application->CreateForm(__classid(TfrmConfigurar), &frmConfigurar);
                Application->CreateForm(__classid(TfrmMS), &frmMS);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        catch (...)
        {
                try
                {
                        throw Exception("");
                }
                catch (Exception &exception)
                {
                        Application->ShowException(&exception);
                }
        }
        return 0;
}
//---------------------------------------------------------------------------
```

Serie.cpp :

```
/*******************************************************
*                COMUNICA.CPP                          *
********************************************************
********************************************************
* Programa para comunicaci¢n de datos entre dos P.C.     *
* Se utilizara los puertos en serie COM1 y COM2.        *
*                                                        *
* Autor; Wilman de J. Menco Silva                       *
********************************************************/

/*******************************************************
*              Definicion de librerias              *
********************************************************/

/* Rutinas del menu instanatanteo para operar en el modo texto */

/* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   Este programa utiliza las interrupciones del 8086, por
   lo tanto, para ejecutarlo debes hacerlo en un computador con este
   procesador o que soporte dichas interrupciones.
  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */

#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <bios.h>

#define  PUERTO    0
#define  ARRIBA    0
#define  ABAJO     1
#define  BOT     4
#define  BORDE          1
#define  ESC      27
#define  VID_INV  0x70
#define  NORM_VID 7
#define  BKSP     8
#define  MAX_BASE 10

void guarda_video () ;
void restaura_video () ;
void pd_driver () ;
void vete_xy () ;
void borpan () ;
void escribe_texto () ;
```

```
void escribe_car () ;
void muestra_cabecera () ;
void dibuja_borde () ;
void desplaza_ventana () ;
void ventana_leet () ;
void redimen () ;
void mover () ;
void ventana_borra () ;
void ventana_borrestol () ;
void ventana () ;
void visualiza_menu () ;
void decahex () ;
void block_notas () ;
void calc () ;
void salva_video () ;
void receptor () ;
void finaliza (void) ;
void lisdir (void) ;
void trans_arch () ;
void recibe_archivo () ;
unsigned int longarchivo () ;
void epuerto () ;
void envia_archivo () ;
void rec_archivo () ;
void envia_nombre_archivo () ;
void lee_nombre_archivo () ;
void inic_puerto () ;
void espera () ;

 char *opcion [] =
  {
   "1. TRANSMISION  ARCHIVO",
   "2. RECEPCION ARCHIVO",
   "3. TRANSMISION TEXTO",
   "4. RECEPCION TEXTO",
   "5. SALIR"
  } ;

 char far *mem_video ;
 int opc ;

 struct base_ventana
  {
   int comienzax, finx, comienzay, finy ;          /* Posicion ventana */
   int curx, cury ;         /* Posicion actual del cursor en la ventana */
   unsigned char *p ;                          /* Puntero a buffer */
   char *cabecera ;                           /* Mensaje a cabecera */
   int borde ;                                 /* Borde si o no */
   int activa ;                              /* En pantalla si o no */
  } base [MAX_BASE] ;
```

```
main ()
{
 union inkey
  {
   char car [2] ;
   int i ;
  } c ;

 int i ;
 char car ;
 clrscr();
 vete_xy (0, 0) ;

 /* Primero crea las estructuras de las ventanas */

 vete_xy (0,0) ;
 crear_ventana (0, " VENTANA DE TRANSMISION ", 12, 0, 23, 78, BORDE)
;
 crear_ventana (1, " VENTANA DE RECEPCION ", 0, 0, 12, 78, BORDE) ;
 crear_ventana (2, " SELECCION ARCHIVO ", 0, 25, 5, 51, BORDE) ;
 crear_ventana (4, " MENU ",9, 25, 17, 51, BORDE) ;

 do
  {
   ventana (4) ;
   opc = menu_instantaneo (opcion, "12345", 5, 10, 27) ;

   switch (opc)
    {
     case 0:
              {
               trans_arch () ;
               break ;
              }
     case 1:
              {
               recibe_archivo () ;
               break ;
              }
     case 2:
              {
               trans_arch () ;
               break ;
              }
     case 3:
              {
               recibe_archivo () ;
               break ;
              }
```

```
    }
  } while (opc != 4) ;

 /* Desactiva (0) ;  Borra la ventana */

  restaura_video (4) ;
  vete_xy (24,0) ;

 }

/*============================================================
===========*/
 /*            FUNCIONES     DE    VENTANA                    */

/*============================================================
===========*/

 menu_instantaneo (char *menu[], char *teclas, int contador,int x, int y)
                  /* menu  ->                    Texto del Menu */
                  /* teclas ->                  Teclas Claves  */
                  /* contador ->            Numero de items del Menu */
                  /* x, y   ->   Coordenadas de la esquina sup. izquierda*/
  {
  register int i, len ;
  int finx, finy, modov, eleccion ;
  unsigned int *p ;

  /* Visualiza el Menu  */
    visualiza_menu (menu, x+1, y+1, contador) ;
  /* Obtiene la Respuesta del Usuario */
    eleccion = obtiene_resp(x+1, y, contador, menu, teclas) ;

  /* Restaura la Pantalla Original*/
    restaura_video (4) ;
    return (eleccion) ;

  }

 /*******  Aparicion en Pantalla de una Ventana *********/

  void ventana (int num)
               /* num  ->   Numero de la Ventana */
  {
  int modov, selecc ;
  int x, y ;

  modov = modo_video () ;
   if ((modov != 2) && (modov != 3) && (modov != 7))
    {
     printf (" VIDEO DEBE SER EN MODO TEXTO DE 80 COLUMNAS") ;
```

```
        exit (1) ;
       }

    /*  Selecciona la direccion adecuada de la RAM  de Video  */
       if (modov == 7)
        mem_video = (char far *) 0xB0000000 ;

       else
        mem_video = (char far *) 0xB8000000 ;

    /* Situa en activa la Ventana */
      if (!base [num].activa)                 /* No esta Activa */
       {
        guarda_video (num) ;       /* Graba la pantalla a la vista */
        base [num].activa = 1 ;       /* Activa la marca de Activa */
       }

      if (base [num].borde)
       dibuja_borde (num) ;
      muestra_cabecera (num) ;              /* Muestra la Ventana */

     x = base [num].comienzax + base [num].curx+1 ;
     y = base [num].comienzay + base [num].cury+1 ;
     vete_xy (x, y) ;

    }


/*============================================================
===========*/
    /* Construye una base de ventana 1 es retornado si el marco de la ventana */
    /* debe ser construido, otro caso 0 es retornado.                  */

/*============================================================
===========*/

    crear_ventana (int num, char *cabecera, int comienzax,
                   int comienzay, int finx, int finy, int borde )
                                        /* num ->   Numero de la Ventana */
                                     /* cabecera ->  Texto de la cabecera */
              /* comienzax, comienzay -> X, Y  Coordenadas esquina sup. izq */
                /* finx, finy -> Coordenadas de la esquina inferior derecha */
                                              /* borde -> no borde si 0 */
    {
     unsigned char *p ;

     if (num > MAX_BASE)
      {
       printf ("DEMASIADAS VENTANAS\n") ;
       return (0) ;
```

```
  }

if((comienzax >24) || (comienzax < 0) || (comienzay >78) || (comienzay <0))
 {
  printf ("ERROR DE RANGO") ;
  return (0) ;
 }

if((finx > 24) || (finy > 79))
 {
   printf ("VENTANA NO ADECUADA") ;
   return (0) ;
 }

/* Localiza suficiente memoria para almacenar */
p = (unsigned char *) malloc (2 *(finx-comienzax+1) * (finy-comienzay+1)) ;
if (!p)
 exit (1) ;

/* Construccion del Marco */
   base [num].comienzax = comienzax ;
   base [num].finx = finx ;
   base [num].comienzay = comienzay ;
   base [num].finy = finy ;
   base [num].p = p ;
   base [num].cabecera = cabecera ;
   base [num].borde = borde ;
   base [num].activa = 0 ;
   base [num].curx = 0 ;
   base [num].cury = 0 ;
   return (1) ;
 }

/* Desactiva la ventana y la quita de la pantalla */
 desactiva (int num)
 {
  /* Inicializa el cursor en la esquina superior izquierda */
   base [num].curx = 0 ;
   base [num].cury = 0 ;
   restaura_video(num) ;
 }

/* Visualiza menu en su posicion */
 void visualiza_menu (char *menu[], int x, int y, int contador)
 {
  register int i ;

  for (i = 0 ; i < contador ; i++, x++)
   escribe_texto (x, y, menu[i], NORM_VID) ;
 }
```

```
/* Obtiene seleccion del Usuario */
 obtiene_resp ( int x, int y,int contador, char *menu [], char *teclas)
 {
  union inkey
  {
   char ch [2] ;
   int i ;
  } c ;
 int flecha = 0 ;
 int tecla_elegida ;
  y++ ;

 /* Sobre-ilumina la primera seccion */
  vete_xy (x, y) ;
  salva_video (x, y, menu [0], VID_INV) ; /* Video Inverso */
  for ( ; ; )  /* Bucle Infinito */
   {
    while (!bioskey(1)) ; /* Espera hasta que se pulse una Tecla */
    c.i = bioskey (0) ;   /* Lee una Tecla */

 /* Pone la seccion en Video Normal */
  vete_xy (x+flecha, y) ;
  salva_video (x+flecha, y, menu [flecha], NORM_VID) ; /* Revisualiza */
   if (c.ch [0])   /* Es tecla normal */
    {
   /* Valida si es tecla clave */
    tecla_elegida = esta_en (teclas, tolower (c.ch [0])) ;
    if (tecla_elegida)
     return (tecla_elegida - 1) ;
  /* Compruebe ENTER o barra espaciadora */

   switch (c.ch [0])
     {
         case '\r' : return (flecha) ;
         case ' '  : flecha++ ;
                     break ;
         case ESC  : return -1 ;
     }
    }
   else                    /* Es una tecla especial */
    {
     switch (c.ch [1])
     {
         case 72 : flecha-- ;    /* Flecha Arriba */
                   break ;
         case 80 : flecha++ ;    /* Flecha Abajo */
                   break ;
     }
    }
```

```
   if (flecha == contador)
    flecha = 0 ;
   if (flecha < 0 )
    flecha = contador - 1 ;

  /* Sobre-ilumina la siguiente seccion */
   vete_xy (x+flecha, y) ;
   salva_video (x+flecha, y, menu [flecha], VID_INV) ;
   }
}
esta_en (char *s, char c)
 {
  register int i ;

   for ( i = 0 ; *s ; i++)
    if (*s++ == c)
     return (i + 1) ;
     return (0) ;
 }

/* Cambia interactivamente el tama¤o de una ventana */
 void redimen (int num)
 {
  char car ;
  int x, y ;
  int comienzax, comienzay ;

 /* Activa si es necesario */
  if (!base [num].activa)
   ventana (num) ;

  comienzax = x = base [num].comienzax ;
  comienzay = y = base [num].comienzay ;
  ventana_xy (num, 0 , 0) ;

  do
   {
    car = lec_especial () ;
    switch (car)
     {
      case 75   : comienzay-- ;      /* Izquierda */
                  break ;

      case 77   : comienzay++ ;      /* Derecha */
                  break ;

      case 72   : comienzax-- ;      /* Arriba */
                  break ;
```

```
    case 80   : comienzax++ ;      /* Abajo */
                break ;

    case 71   : comienzax-- ;      /* Arriba Izquierda */
                comienzay-- ;
                break ;

    case 73   : comienzax-- ;      /* Arriba Derecha */
                comienzay++ ;
                break ;

    case 79   : comienzax++ ;      /* Abajo Izquierda */
                comienzay-- ;
                break ;

    case 81   : comienzax++ ;      /* Abajo Derecha */
                comienzay++ ;
                break ;

    case 27   : comienzax = x ;  /* ESC cancela y vuelve a tama¤o orig.*/
                comienzay = y ;
                car = 60 ;
   }

 /* Comprueba si esta fuera de rango */
  if (comienzax < 0 )
    comienzax++ ;
  if (comienzax >= base[num].finx)
    comienzax-- ;
  if (comienzay < 0 )
    comienzay++ ;
  if (comienzay >= base[num].finy)
    comienzay-- ;
   desactiva (num) ;
   base [num].comienzax = comienzax ;
   base [num].comienzay = comienzay ;
   ventana (num) ;

  } while (car != 60) ;
 /*  desactiva (num) ;      Borra la Ventana */
 }

/* Mueve interactivamente una ventana */
 void mover (int num)
 {
  char car ;
  int x, y ;
  int ex, ey ;
  int finx, finy ;
  int comienzax, comienzay ;
```

```
/* Activa si es necesario */
 if (!base [num].activa)
  ventana (num) ;

 comienzax = x = base [num].comienzax ;
 comienzay = y = base [num].comienzay ;
 finx = ex = base [num].finx ;
 finy = ey = base [num].finy ;
 ventana_xy (num, 0 , 0) ;

 do
  {
   car = lec_especial () ;
   switch (car)
    {
     case 75   : comienzay-- ;      /* Izquierda */
                 finy-- ;
                 break ;

     case 77   : comienzay++ ;      /* Derecha */
                 finy++ ;
                 break ;

     case 72   : comienzax-- ;      /* Arriba */
                 finx-- ;
                 break ;

     case 80   : comienzax++ ;      /* Abajo */
                 finx++ ;
                 break ;

     case 71   : comienzax-- ;      /* Arriba Izquierda */
                 finx-- ;
                 comienzay-- ;
                 finy-- ;
                 break ;

     case 73   : comienzax-- ;      /* Arriba Derecha */
                 finx-- ;
                 comienzay++ ;
                 finy++ ;
                 break ;

     case 79   : comienzax++ ;      /* Abajo Izquierda */
                 finx++ ;
                 comienzay-- ;
                 finy-- ;
                 break ;
```

```
    case 81   : comienzax++ ;      /* Abajo Derecha */
                finx++ ;
                comienzay++ ;
                finy++ ;
                break ;

    case 27   : comienzax = x ;  /* ESC cancela y vuelve a tama¤o orig.*/
                comienzay = y ;
                finx = ex ;
                finy = ey ;
                car = 60 ;
   }

 /* Comprueba si esta fuera de rango */
  if (comienzax < 0 )
   {
    comienzax++ ;
    finx++ ;
   }
  if (finx >= 24)
   {
    comienzax-- ;
    finx-- ;
   }
  if (comienzay < 0 )
   {
    comienzay++ ;
    finy++ ;
   }
  if (comienzay >= 79)
   {
    comienzay-- ;
    finy-- ;
   }
    desactiva (num) ;
    base [num].comienzax = comienzax ;
    base [num].comienzay = comienzay ;
    base [num].finx = finx ;
    base [num].finy = finy ;
    ventana (num) ;
   } while (car != 60) ;
  /* desactiva (num) ;     Borra la Ventana */
 }

/* Visualiza la cabecera en su posicion */
 void muestra_cabecera (int num)
 {
  register int y, lon ;
  y = base [num].comienzay ;
```

```
/*=============================================================
=========*/
          /* Calcula la posicion correcta de comienzo del mensaje de cabecera     */
          /* si es negativo el mensaje no es adecuado.                            */

/*=============================================================
=========*/
           lon = strlen (base[num].cabecera) ;
           lon = (base[num].finy - y - lon) / 2 ;
            if (lon < 0 )
               return ;       /* No lo visualiza */
              y = y + lon ;
           escribe_texto (base [num].comienzax, y, base [num].cabecera, NORM_VID ) ;

          }

          void dibuja_borde (int num)
          {
           register int i ;
           char far *v, far *t ;

           v = mem_video ;
           t = v ;

           for (i = base [num].comienzax+1 ; i < base [num].finx ; i++)
            {
              v += (i*160) + base [num].comienzay * 2 ;
              *v++ = 186 ;
              *v = VID_INV ;
              v = t ;
              v += (i*160) + base [num].finy *2 ;
              *v++ = 186 ;
              *v = VID_INV ;
              v = t  ;
            }
           for (i = base [num].comienzay+1 ; i < base [num].finy ; i++)
            {
              v += (base [num].comienzax * 160) + i * 2 ;
              *v++ = 205 ;
              *v = VID_INV ;
              v = t ;
              v += (base [num].finx * 160) + i * 2 ;
              *v++ = 205 ;
              *v = VID_INV ;
              v = t  ;
            }

           escribe_car (base [num].comienzax, base [num].comienzay, 201, VID_INV) ;
           escribe_car (base [num].comienzax, base [num].finy, 187, VID_INV) ;
```

```
 escribe_car (base [num].finx, base [num].comienzay, 200, VID_INV) ;
 escribe_car (base [num].finx, base [num].finy, 188, VID_INV) ;
}


/*=================================================================
===========*/
/*          FUNCIONES DE  E/S  VENTANA                      */

/*=================================================================
===========*/

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º Escribe una cadena en la posicion actual del cursor en determinada    º
 º  ventana. Retorna 0 si la ventana no esta activa, 1 en otro caso.     º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 ventana_escribet (int num, char *str)
 {
   /* Comprueba ventana esta activa */
    if (!base [num].activa)
     return (0) ;

    for ( ; *str ; str++)
     ventana_escribecar (num, *str) ;
     return (1) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º Escribe un caracter en la posicion actual del cursor en determinada   º
 º  ventana. Retorna 0 si la ventana no esta activa, 1 en otro caso.     º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 ventana_escribecar (int num, char car)
 {
  register int x, y ;
  char far *v ;

  /* Comprueba ventana esta activa */
   if (!base [num].activa)
      return (0) ;

   x = base [num].curx + base [num].comienzax+1 ;
   y = base [num].cury + base [num].comienzay+1 ;

   v = mem_video ;
   v += (x * 160) + y * 2 ;   /* Calcula la direccion */
   if (y >= base [num].finy)
    {
     return (1) ;
    }
   if (x >= base[num].finx)
```

```
    {
     return (1) ;
    }

   if (car == '\n')        /* Nueva Linea */
    {
     x++ ;
     y = base[num].comienzax + 1 ;
     v = mem_video ;
     v += (x*160) + y*2 ;  /* Calcula la direccion */
     base [num].curx++ ;    /* Incrementa x */
     base [num].cury = 0 ; /* Inicializa y */
    }
  else
   {
    base [num].cury++ ;
    *v++ = car ;          /* Escribe el caracter */
    *v++ = NORM_VID ;      /* Atributo normal de video */
   }
  ventana_xy (num, base [num].curx, base [num].cury) ;
  return (1) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º Posiciona el cursor en una determinada posicion de la ventana.        º
 º Retorna 0 si sale fuera del rango, no-cero en otro caso.              º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 ventana_xy (int num, int  x, int y)
 {
  if (x < 0 || x + base [num].comienzax >= base [num].finx - 1)
     return (0) ;
  if (y < 0 || y + base [num].comienzay >= base [num].finy - 1)
     return (0) ;
  base [num].curx = x ;
  base [num].cury = y ;
  vete_xy (base [num].comienzax + x + 1, base [num].comienzay + y + 1) ;
  return (1) ;
 }

/* Lee una cadena desde una ventana */
 void ventana_leet (int num, char *s)
 {
  char car, *temp ;
  temp = s ;

  for ( ; ; )
   {
    car = ventana_leecar (num) ;
    switch (car)
     {
```

```
     case '\r' :              /* La tecla ENTER es pulsada */
      *s = '\0' ;
         return  ;

     case BKSP :              /* Backspace */
          if (s > temp)
           {
            s-- ;
            base [num].cury-- ;
            if (base [num].cury < 0)
            base [num].cury = 0 ;
            ventana_xy (num, base [num].curx, base [num].cury) ;
            escribe_car (base [num].comienzax + base [num].curx + 1,
            base [num].comienzay + base [num].cury + 1, ' ', NORM_VID) ;
           }
     break ;
     default :
             *s = car ;
             s++ ;
    }
  }
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º           Entran las teclas pulsadas en la ventana.                º
 º           Retorna el codigo de exploracion de 16 bit.              º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 ventana_leecar (int num)
 {
  union inkey
   {
    char car [2] ;
    int i ;
   } c ;
  if (!base [num].activa)      /* Ventana no activa */
     return (0) ;
  ventana_xy (num, base [num].curx, base [num].cury) ;

  c.i = bioskey (0) ;          /* Lee tecla */
  if (c.car [0])
   {
    switch (c.car [0])
     {
      case '\r' :              /* La tecla ENTER es pulsada */
            break ;

      case BKSP :              /* Backspace */
            break ;

          default :
```

```
          if (base [num].cury + base [num].comienzay < base [num].finy - 1)
           {
            escribe_car (base [num].comienzax + base [num].curx + 1,
            base [num].comienzay + base [num].cury + 1, c.car[0], NORM_VID) ;
            base [num].cury++ ;
           }
    }
   if (base [num].curx < 0)
    base [num].curx = 0 ;
   if (base [num].curx + base [num].comienzax >= base [num].finx-2)
    {
     desplaza_ventana (base [0].comienzay + 1, base [0].comienzax + 1,
     base [0].finy - 1, base [0].finx - 1, 1, ARRIBA) ;
     base [num].curx-- ;
    }
   ventana_xy (num, base [num].curx, base [num].cury) ;
  }
 return (c.i) ;
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º          Desplaza una ventana hacia arriba o abajo.              º
 º        comienzax, comienzay  -> Esquina superior izquierda          º
 º        finx, finy  ->  Esquina inferior derecha                    º
 º        linea  ->  Numero de lineas a desplazar                     º
 º        direcc ->  Arriba o Abajo                                 º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void desplaza_ventana (char comienzax, char comienzay, char finx,
                        char finy, char linea, char direcc)
 {
  union REGS r ;

  if (direcc == ARRIBA)
     r.h.ah = 6 ;        /* Desplaza hacia Arriba */
  else
     r.h.ah = 7 ;        /* Desplaza hacia Abajo */

  r.h.al = linea ;
  r.h.ch = comienzay ;
  r.h.cl = comienzax ;
  r.h.dh = finy ;
  r.h.dl = finx ;
  r.h.bh = 0 ;          /* Visualiza Atributo */
  int86 (0x10, &r, &r) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º          Limpia el final de la linea.                            º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void ventana_borra (int num)
```

```
{
 register int i, j ;
 char far *v, far *t ;

 v = mem_video ;
 t = v ;
 for (i = base [num].comienzay + 1 ; i < base [num].finy ; i++)
  for (j = base [num].comienzax + 1 ; j < base [num].finx ; j++)
   {
    v = t ;
    v += (j * 160) + i * 2 ;
    *v++ = ' ' ;               /* Escribe un espacio */
    *v = NORM_VID ;
   }
 base [num].curx = 0 ;
 base [num].cury = 0 ;
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º           Borra hasta el final de una linea.                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void ventana_borrestol (int num)
{
 register int i, x, y ;

 x = base [num].curx ;
 y = base [num].cury ;
 ventana_xy (num, base [num].curx, base [num].cury) ;

 for (i = base [num].cury ; i < base [num].finy - 1 ; i++)
   ventana_escribecar (num, ' ') ;
 ventana_xy (num, x, y) ;
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                Mueve el cursor una linea arriba.                      º
º              Retorna no-cero si prospera, 0 otro caso.                º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
ventana_arriba (int num)
{
 if (base [num].curx > 0)
  {
   base [num].curx-- ;
   ventana_xy (num, base [num].curx, base [num].cury) ;
   return (1) ;
  }
 return (0) ;
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
```

```
º                  Mueve el cursor una linea abajo.                   º
º               Retorna no-cero si prospera, 0 otro caso.            º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
ventana_abajo (int num)
{
 if (base [num].curx < base [num].finx - base [num].comienzax - 1)
  {
   base [num].curx++ ;
   ventana_xy (num, base [num].curx, base [num].cury) ;
   return (1) ;
  }
 return (1) ;
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                      Back up un caracter.                          º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
ventana_retro (int num)
{
 if (base [num].cury > 0)
  {
   base [num].cury-- ;
   ventana_xy (num, base [num].curx, base [num].cury) ;
   ventana_escribecar (num, ' ') ;
   base [num].cury-- ;
   ventana_xy (num, base [num].curx, base [num].cury) ;
   return (1) ;
  }
}


/*============================================================================
===========*/
/*                    MISCELANEO  DE  FUNCIONES                         */

/*============================================================================
===========*/

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º             Visualiza una cadena con determinado atributo.           º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void escribe_texto (int x, int y, char *p, int atrib)
{
 register int i ;
 char far *v ;

 v = mem_video ;
 v += (x * 160) + y * 2 ;      /* Calcula la direccion */

 for ( i = y ; *p ; i++)
```

```
  {
   *v++ = *p++ ;               /* Escribe el caracter */
   *v++ = atrib ;              /* Escribe el atributo */
  }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º            Escribe un caracter con determinado atributo.             º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void escribe_car (int x, int y, char car, int atrib)
 {
  register int i ;
  char far *v ;

  v = mem_video ;
  v += (x * 160) + y * 2 ;      /* Calcula la direccion */
  *v++ = car ;                  /* Escribe el caracter */
  *v = atrib ;                  /* Escribe el atributo */
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º            Salva una parte de la pantalla.                       º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void guarda_video (int num)
 {
  register int i, j ;
  char  *buf_ptr ;
  char far *v, far *t ;

  buf_ptr = base [num].p ;
  v = mem_video ;
  for (i = base [num].comienzay ; i < base [num].finy + 1 ; i++)
   for (j = base [num].comienzax ; j < base [num].finx + 1 ; j++)
    {
     t = (v + (j * 160) + i * 2) ;
     *buf_ptr++ = *t++ ;
     *buf_ptr++ = *t ;
     *(t-1) = ' ' ;             /* Limpia la ventana */

    }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                   Salva una parte de la pantalla.                º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void salva_video (int x, int y, char *p, int atrib)
 {
  register int i, j ;
  union REGS r ;
```

```
 for ( i = y ; *p ; i++)
  {
   vete_xy (x, i) ;
   r.h.ah = 9 ;
   r.h.bh = 0 ;
   r.x.cx = 1 ;
   r.h.al = *p++ ;
   r.h.bl = atrib ;
   int86 (0x10, &r, &r) ;
  }
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º               Restaura una parte de la pantalla.                  º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void restaura_video (int num)
{
 register int i, j ;
 char  *buf_ptr ;
 char far *v, far *t ;

 buf_ptr = base [num].p ;
 v = mem_video ;
 t = v ;
 for (i = base [num].comienzay ; i < base [num].finy + 1 ; i++)
  for (j = base [num].comienzax ; j < base [num].finx + 1 ; j++)
   {
    v = t ;
    v += (j * 160) + i * 2 ;
    *v++ = *buf_ptr++ ;            /* Escribe el caracter */
    *v = *buf_ptr++ ;              /* Escribe el caracter */

   }
 base [num].activa = 0 ;           /* Restaura video */
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                     Limpia  pantalla.                             º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void borpan ()
{
 union REGS r ;

   r.h.ah = 6 ;        /* Codigo de scroll de pantalla */
   r.h.al = 0 ;        /* Codigo de limpia la pantalla */
   r.h.ch = 0 ;        /* Fila de comienzo */
   r.h.cl = 0 ;        /* Columna de comienzo */
   r.h.dh = 24 ;       /* Fila final */
   r.h.dl = 79 ;       /* Columna final */
   r.h.bh = 7 ;        /* Linea en blanco */
```

```
   int86 (0x10, &r, &r) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º        Envia el cursor a una posicion especificada por x,y.          º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void vete_xy (int x, int y)
 {
  union REGS r ;

   r.h.ah = 2 ;        /* Funcion de direccion del cursor */
   r.h.dl = y ;        /* Coordenada de la columna */
   r.h.dh = x ;        /* Coordenada de la fila */
   r.h.bh = 0 ;        /* Pagina de video */
   int86 (0x10, &r, &r) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º    Devuelve el codigo de posicion de las teclas de flecha y funcion    º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 lec_especial ()
 {
  union inkey
   {
   char car [2] ;
   int i ;
   } c ;

  while (!bioskey (1)) ;          /* Espera una pulsacion */
  c.i = bioskey (0) ;
  return (c.car [1]) ;
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º              Devuelve el modo de video en operacion.                º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 modo_video ()
 {
  union REGS r ;

   r.h.ah = 15 ;        /* Lee el modo de video */
   return (int86 (0x10, &r, &r) & 255) ;
 }

 is_in (char *s, char c)
 {
  register int i ;
  for ( i = 0 ; *s ; i++)
   if (*s++ == c)
      return (i + 1) ;
```

```
     return (0) ;
    }


/*===============================================================
===========*/
   /*           FUNCIONES DE VENTANA INSTANTANEA               */

/*===============================================================
===========*/

   /* Block de notas instantaneo */
    #define NOTAS_MAX 50
    #define BKPS       8

    char notas [NOTAS_MAX] [80] ;

   /*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
    ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
    void block_notas ()
    {
     static primero = 1 ;
     register int i, j ;

     union inkey
      {
       char car [2] ;
       int i ;
      } c ;
     char car ;

   /* Inicializa el array notas si es necesario */
    if (primero)
     {
      for (i = 0 ; i < NOTAS_MAX ; i++)
       *notas [i] = '\0' ;
      primero = !primero ;
     }

     ventana (0) ;
   /* Visualiza las notas existentes */
    for (i = 0 ; i < NOTAS_MAX ; i++)
     {
      if (*notas [i])
       ventana_escribet (0, notas [i]) ;
      ventana_escribecar (0, '\n') ;
     }

     i = 0 ;
     ventana_xy (0, 0, 0) ;
```

```
 for ( ; ; )
  {
   c.i = ventana_leecar (0) ;       /* Lee tecla */
   if (tolower (c.car [1]) == 59)   /* F1 para salir */
    {
     desactiva (0) ;
     break ;
    }
   if (tolower (c.car [1]) == 60)    /* F2 para salir */
    {
     redimen (0) ;
     mover (0) ;
    /* Visualiza las notas existentes */
     for (i = 0 ; i < NOTAS_MAX ; i++)
      {
          if (*notas [i])
          ventana_escribet (0, notas [i]) ;
          ventana_escribecar (0, '\n') ;
         }
    }
   /* Si tecla normal */
    if (isprint (c.car [0]) || c.car [0] == BKSP)
     {
         ventana_borrestol (0) ;
         notas [i] [0] = c.car [0] ;
         j = i ;
/*****     ventana_escribecar (0, notas [i] [0]) ; ************/

          do
          {
           car = ventana_leecar (0) ;
           if (car == BKSP)
            {
             if (j > 0)
              {
               j-- ;
               ventana_retro (0) ;
              }
            }
           else
            {
             notas [i] [j] = car ;
             j++ ;
            }
          } while (notas [i] [j-1] != '\r') ;
         notas [i] [j] = '\0';
         i++ ;
         ventana_escribecar (0, '\n') ;
     }
```

```
    else                   /* Es tecla especial */
     {
         switch (c.car [1])
          {
          case 72 :                    /* Flecha arriba */
                     if (i > 0)
                      {
                       i-- ;
                       ventana_arriba (0) ;
                      }
                     break ;
          case 80 :                    /* Flecha abajo */
                     if (i < NOTAS_MAX - 1)
                      {
                       i++ ;
                       ventana_abajo (0) ;
                      }
                     break ;
          }
     }
  }
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void recibe_archivo ()
 {
  register int i, j ;
  union inkey
   {
    char car [2] ;
    int i ;
   } c ;
  char car ;
  ventana (1) ;
  i = 0 ;
  ventana_xy (1, 0, 0) ;
  vete_xy (24, 15) ;
  printf ("PF1 [SALIR]  PF2 [REDIMEN]  PF3 [CONTINUAR]") ;
  for ( ; ; )
   {
    c.i = ventana_leecar (1) ;       /* Lee la tecla */
    if (tolower (c.car [1]) == 59)    /* F1 para salir */
     {
      desactiva (1) ;
      break ;
     }
    if (tolower (c.car [1]) == 60)    /* F2 para redimensionar */
     {
      redimen (1) ;
```

```
    mover (1) ;
   /* Visualiza las notas existentes */
    for (i = 0 ; i < NOTAS_MAX ; i++)
     {
          if (*notas [i])
          ventana_escribet (1, notas [i]) ;
          ventana_escribecar (1, '\n') ;
         }
   }
  if (tolower (c.car [1]) == 62)    /* F4 para Transmision */
         rec_archivo () ;
 }
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                Transmitir archivo.                              º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void trans_arch ()
{
 register int i, j ;
 union inkey
  {
   char car [2] ;
   int i ;
  } c ;
 static char nomarch [12] ;
 char car ;
 ventana (0) ;
 i = 0 ;
 ventana_xy (0, 0, 0) ;
 vete_xy (24, 7) ;
 printf  ("PF1  [SALIR]    PF2  [REDIMEN]    PF3  [DIRECTORIO]    PF4
[CONTINUAR]") ;
 for ( ; ; )
  {
   c.i = ventana_leecar (0) ;        /* Lee la tecla */
   if (tolower (c.car [1]) == 59)    /* F1 para salir */
    {
    desactiva (0) ;
    break ;
    }
   if (tolower (c.car [1]) == 60)    /* F2 para redimensionar */
    {
    redimen (0) ;
    mover (0) ;
   /* Visualiza las notas existentes */
    for (i = 0 ; i < NOTAS_MAX ; i++)
     {
          if (*notas [i])
          ventana_escribet (0, notas [i]) ;
```

```
          ventana_escribecar (0, '\n') ;
         }
     }
    if (tolower (c.car [1]) == 61)    /* F3 para directorio */
          lisdir () ;
    if (tolower (c.car [1]) == 62)    /* F4 para Transmision */
     {
      if (opc == 0 )
         {
          ventana (2) ;
          inic_puerto (PUERTO, 231) ; /* Inicializacion del puerto serie */
          vete_xy (base [2].comienzax + 1, base [2].comienzay + 2) ;
          scanf ("%s", nomarch) ;
          desactiva (2) ;
         }
      envia_archivo (nomarch) ;
     }
   }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void lisdir ()
 {
  int numcol, conta ;
  int j = base [0].comienzax + 1 ;
  int i = base [0].comienzay + 2 ;
  static unsigned char pp [44] ;  /* Vector del tama¤o del buffer         */
  bdos (0x1A, (unsigned) pp, 0) ; /* Direccionando al buffer para transfe */
  bdos (0x4E, (unsigned)"*.*", 0) ; /* Mandando los comodines de busqueda */

  numcol = 0 ;
  conta = 1 ;
  numcol = ((base [0].finy - 2) - (base [0].comienzay + 2)) / 14 ;
  vete_xy (j, i) ;
  printf ("%s \n", &pp [30]) ;  /* Imprimiendo desde el buffer el archivo */
  for ( ; ; )
   {
    if (18 == bdos (0x4F, 0, 0))     /* No mas archivos en el directorio */
     break ;
    if ((j > base [0].finx - 1) && (conta >= numcol))
     {
      getch () ;
      ventana_borra (0) ;
      conta = 1 ;
      i = base [0].comienzay + 2 ;
      j = base [0].comienzax ;
     }
    if (j >= base [0].finx - 1)
     {
```

```
      conta = conta + 1 ;
      i = i + 14 ;
      j = base [0].comienzax ;
     }
    j++ ;
    vete_xy (j, i) ;
    printf ("%s \n", &pp [30]) ;
   }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                        Envia un archivo dado.                          º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void envia_archivo (char *nombref)
 {
  FILE *fp ;
  char ch ;
  int aa1, bb1, aa2, bb2 ;

  union
   {
    char c [2] ;
    unsigned int cuenta ;
   } cnt ;
  if (opc == 0)
   {
    if (!(fp = fopen (nombref,"rb")))
     {
      printf ("NO PUEDE ABRIR EL ARCHIVO DE ENTRADA \n") ;
      exit (1) ;
     }
    else
     {
      vete_xy (base [0].comienzax + 1, base [0].comienzay + 1) ;
      printf ("ARCHIVO DE ENTRADA ABIERTO %s", nombref) ;
     }
    envia_nombre_archivo (nombref) ;     /* Envia el nombre del archivo */
    espera (PUERTO) ;          /* Espera el reconocimiento del receptor */

   /* Calcula el tama¤o del archivo */
    cnt.cuenta = longarchivo (fp) ;

   /* Envia el tama¤o */
    epuerto (PUERTO, cnt.c [0] ) ;
    espera (PUERTO) ;
    epuerto (PUERTO, cnt.c [1] ) ;
    aa1 = base [0].comienzax + 4 ;
    bb1 = base [0].comienzay + 1 ;
   }
  else
```

```
 {
  aa1 = base [0].comienzax + 1 ;
  bb1 = base [0].comienzay + 1 ;
 }
aa2 = base [0].finx - 2 ;
bb2 = base [0].finy - 2 ;
if (opc == 0)
 do
  {
   ch = getc (fp) ;
   if (ferror (fp))
    {
        printf ("ERROR DE LECTURA EN EL ARCHIVO DE ENTRADA") ;
        break ;
    }
   if (aa1 <= aa2)
    {
        if ((bb1 > bb2) || (ch == '\n') || (ch == '\r'))
         {
          bb1 = base [0].comienzay + 1 ;
          aa1 = aa1 + 1 ;
         }
        vete_xy (aa1, bb1) ;
        printf ("%c", ch) ;
        bb1 = bb1 + 1 ;
    }
   else
    {
        desplaza_ventana (base [0].comienzay + 1, base [0].comienzax + 1,
                          base [0].finy - 1, base [0].finx - 1, 1, ARRIBA) ;
        bb1 = base [0].comienzay + 1 ;
        aa1 = aa1 - 1 ;
    }

 /* Espera hasta que el receptor este preparado */
  if (!feof (fp))
   {
    espera (PUERTO) ;
    epuerto (PUERTO, ch) ;
   }
  } while (!feof (fp)) ;
else
 do
  {
   if (opc == 2)
    ch = getch () ;
   if (aa1 <= aa2)
    {
        if ((bb1 > bb2) || (ch == '\n') || (ch == '\r'))
         {
```

```
          bb1 = base [0].comienzay + 1 ;
          aa1 = aa1 + 1 ;
         }
        vete_xy (aa1, bb1) ;
        printf ("%c", ch) ;
        bb1 = bb1 + 1 ;
     }
    else
     {
        desplaza_ventana (base [0].comienzay + 1, base [0].comienzax + 1,
                           base [0].finy - 1, base [0].finx - 1, 1, ARRIBA) ;
        bb1 = base [0].comienzay + 1 ;
        aa1 = aa1 - 1 ;
     }

   /* Espera hasta que el receptor este preparado */
    if (ch != EOF)
     {
      espera (PUERTO) ;
      epuerto (PUERTO, ch) ;
     }
    } while (ch != EOF) ;
  epuerto (PUERTO, ch) ;
  if (opc == 0 )
   {
    espera (PUERTO) ;     /* Lee el ultimo punto del puerto */
    fclose (fp) ;
   }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                  Recepcion de un archivo.                    º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void rec_archivo ()
 {
  FILE *fp ;
  char ch ;
  char nombref [14] ;
  int xx1, yy1, xx2, yy2 ;

  union
   {
    char c [2] ;
    unsigned int cuenta ;
   } cnt ;
  if (opc == 1)
   {
    lee_nombre_archivo (nombref) ;     /* Lee el nombre del archivo */
    vete_xy (base [1].comienzax + 2, base [1].comienzay + 2) ;
    printf ("RECIBE EL ARCHIVO %s\n", nombref) ;
```

```
 remove (nombref) ;
 if (!(fp = fopen (nombref, "wb")))
  {
   printf ("NO PUEDE ABRIR EL ARCHIVO DE SALIDA \n") ;
   exit (1) ;
  }

/* Lee la longitud del archivo */
 epuerto (PUERTO, '.') ;          /* Envia reconocimiento */
 cnt.c [0] = lpuerto (PUERTO) ;
 epuerto (PUERTO, '.') ;          /* Envia reconocimiento */
 cnt.c [1] = lpuerto (PUERTO) ;
}
epuerto (PUERTO, ',') ;           /* Envia reconocimiento */
if (opc == 1)
 {
  xx1 = base [1].comienzax + 3 ;
  yy1 = base [1].comienzay + 1 ;
 }
else
 {
  xx1 = base [1].comienzax + 1 ;
  yy1 = base [1].comienzay + 1 ;
 }
xx2 = base [1].finx - 2 ;
yy2 = base [1].finy - 2 ;
if (opc == 1)
 {
  for (  ; cnt.cuenta ; cnt.cuenta--)
   {
    ch = lpuerto (PUERTO) ;
    if (xx1 <= xx2)
        {
         if ((yy1 > yy2) || (ch == '\n') || (ch == '\r'))
       {
            yy1 = base [1].comienzay + 1 ;
            xx1 = xx1 + 1 ;
           }
            vete_xy (xx1, yy1) ;
            printf ("%c", ch) ;
            yy1 = yy1 + 1 ;
           }
    else
        {
         desplaza_ventana (base [1].comienzay + 1, base [1].comienzax + 1,
                           base [1].finy - 1, base [1].finx - 1, 1, ARRIBA) ;
         yy1 = base [1].comienzay + 1 ;
         xx1 = xx1 - 1 ;
        }
    putc (ch, fp) ;
```

```
     if (ferror (fp))
          {
           printf ("ERROR AL ESCRIBIR  EL ARCHIVO ") ;
           exit (1) ;
          }
     epuerto (PUERTO, '.') ;   /* Envia reconocimiento */
    }
   fclose (fp) ;
  }
 else
  do
   {
    ch = lpuerto (PUERTO) ;
    if (xx1 <= xx2)
     {
          if ((yy1 > yy2) || (ch == '\n') || (ch == '\r'))
           {
            yy1 = base [1].comienzay + 1 ;
            xx1 = xx1 + 1 ;
           }
          vete_xy (xx1, yy1) ;
          printf ("%c", ch) ;
          yy1 = yy1 + 1 ;
     }
    else
     {
          desplaza_ventana (base [1].comienzay + 1, base [1].comienzax + 1,
                            base [1].finy - 1, base [1].finx - 1, 1, ARRIBA) ;
          yy1 = base [1].comienzay + 1 ;
          xx1 = xx1 - 1 ;
     }
    epuerto (PUERTO, '.') ;     /* Envia reconocimiento */

   } while (ch != EOF) ;

}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º           Devuelve la longitud, en bytes de un archivo.             º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 unsigned int longarchivo (FILE *fp)
 {
  unsigned long i ;

  i = 0 ;
  do
   {
    getc (fp) ;
    i++ ;
   } while (!feof (fp)) ;
```

```
 rewind (fp) ;
 return (i - 1) ;       /* No cuenta el caracter EOF */
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                Envia el nombre del archivo.                    º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void envia_nombre_archivo (char *f)
{
 vete_xy (base [0].comienzax + 2, base [0].comienzay + 1) ;
 printf ("TRANSMISOR EN ESPERA....") ;
 do
  {
   epuerto (PUERTO, '?') ;
  } while (!kbhit () && ! (chequea_estado (PUERTO) & 256)) ;
 if (kbhit ())
  {
   getch () ;
   exit (1) ;
  }
 espera (PUERTO) ;     /* Espera el reconocimiento del receptor */
 vete_xy (base [0].comienzax + 3, base [0].comienzay + 1) ;
 printf ("ENVIA %s", f) ;

/* Envia el nombre */
 while (*f)
  {
   epuerto (PUERTO, *f++) ;
   espera (PUERTO) ;   /* Espera el reconocimiento del receptor */
  }
 epuerto (PUERTO, '\0') ; /* Terminador nulo */
}

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                Recibe el nombre del archivo.                   º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
void lee_nombre_archivo (char *f)
{
 printf ("ESPERA DEL RECEPTOR....\n") ;
 while (lpuerto (PUERTO) != '?') ;
 epuerto (PUERTO, '.') ;             /* Envia reconocimiento */
 while ((*f = lpuerto (PUERTO)))
  {
   if (*f != '?')
    {
     f++ ;
     epuerto (PUERTO, '.') ;         /* Envia reconocimiento */
    }
  }
}
```

```
/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                          Espera una respuesta.                          º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void espera (int puerto)
 {
  if (lpuerto (puerto) != '.')
   {
    printf ("ERROR DE COMUNICACION\n") ;
    exit (1) ;
   }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                 Envia un caracter al puerto serie.                      º
 º                  int puerto  ->  puerto de E/S.                         º
 º                  char c  -> Caracter a enviar.                          º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
 void epuerto (int puerto, char c)
 {
  union REGS r ;

  r.x.dx = puerto ;      /* Puerto Serie */
  r.h.al = c ;           /* Caracter a enviar */
  r.h.ah = 1 ;           /* Funcion que envia un caracter */
  int86 (0x14, &r, &r) ;
  if (r.h.ah & 128)      /* Comprueba el bit 7 */
   {
    printf ("DETECTADO ERROR DE TRANSMISION EN EL PUERTO SERIE") ;
    exit (1) ;
   }
 }

/*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
 º                    Lee un caracter del puerto.                          º
 ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
  lpuerto (int puerto)
  {
   union REGS r ;

 /* Espera por el caracter del puerto */
  while (!(chequea_estado (PUERTO) & 256))

    if ((opc == 0) || (opc == 1))
     if (kbhit ())        /* Aborta en caso de pulsar una tecla */
      {
          getch () ;
          exit (1) ;
      }
```

```
    r.x.dx = puerto ;       /* Puerto serie */
    r.h.ah = 2 ;            /* funcion de lectura de un caracter */
    int86 (0x14, &r, &r) ;
    if (r.h.ah & 128)
      printf ("DETECTADO ERROR DE LECTURA EN EL PUERTO SERIE") ;
    return (r.h.al) ;
  }

 /*ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
  º           Chequea el estado del puerto serie.                     º
  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ*/
  chequea_estado (int puerto)
  {
   union REGS r ;

   r.x.dx = puerto ;      /* Puerto Serie */
   r.h.ah = 3 ;
```

uConfigurar.cpp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <IniFiles.hpp>
#pragma hdrstop

#include "uConfigurar.h"
#include "uMain.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmConfigurar *frmConfigurar;
//---------------------------------------------------------------------------
__fastcall TfrmConfigurar::TfrmConfigurar(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmConfigurar::BitBtn1Click(TObject *Sender)
{
        if(cbGuardar->Checked)
  {
                TIniFile *ini;
           ini = new TIniFile(ChangeFileExt( Application->ExeName, ".INI"));
        ini->WriteInteger( "Acceso", "Puerto", cbPuerto->ItemIndex);
        ini->WriteInteger( "Acceso", "Velocidad", cbVelocidad->ItemIndex);
        ini->WriteInteger( "Acceso", "Paridad", cbParidad->ItemIndex);
        ini->WriteInteger( "Acceso", "BitsParada", cbBParada->ItemIndex);
        ini->WriteInteger( "Acceso", "ControlFlujo", cbCFlujo->ItemIndex);
```

```
        delete ini;
   }
        frmPrincipal->SetConfig();
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigurar::BitBtn2Click(TObject *Sender)
{
        cbPuerto->ItemIndex=frmPrincipal->cpSerial->SerialPort;
        cbVelocidad->ItemIndex=frmPrincipal->cpSerial->BaudRate;
        cbParidad->ItemIndex=frmPrincipal->cpSerial->Parity;
        cbBParada->ItemIndex=frmPrincipal->cpSerial->StopBits;
  cbCFlujo->ItemIndex=frmPrincipal->cpSerial->FlowControl;
}
//---------------------------------------------------------------------------
```

uConfigurar.h:

```
//---------------------------------------------------------------------------

#ifndef uConfigurarH
#define uConfigurarH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TfrmConfigurar : public TForm
{
__published:   // IDE-managed Components
        TComboBox *cbVelocidad;
        TComboBox *cbParidad;
        TComboBox *cbBParada;
        TComboBox *cbCFlujo;
        TCheckBox *cbGuardar;
        TBitBtn *BitBtn1;
        TBitBtn *BitBtn2;
        TComboBox *cbPuerto;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall BitBtn2Click(TObject *Sender);
private:        // User declarations
```

```
public:         // User declarations
        __fastcall TfrmConfigurar(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmConfigurar *frmConfigurar;
//---------------------------------------------------------------------------
#endif
```

uMain.cpp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdio.h>
#include <IniFiles.hpp>
#pragma hdrstop

#include "uMain.h"
#include "uConfigurar.h"
#include "uMS.h"
#include "uPaquetes.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CommPort"
#pragma resource "*.dfm"
TfrmPrincipal *frmPrincipal;
// 00809874405D
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void TfrmPrincipal::AbrePuerto()
{
        if(!cpSerial->Connected)
        {
                try
    {
        cpSerial->Open();
                }
        catch(...)
        {
      PublicaEstado("ERROR : No se pudo abrir el puerto", tmError);
      btnConectarBT->Enabled=false;
      btnEnviar->Enabled=false;
      conPS=false;
    }
          if(cpSerial->Connected)
    {
                PublicaEstado("ESTADO : Se abrió el puerto " + frmConfigurar-
>cbPuerto->Text+ " con éxito", tmEstado);
```

```
            btnConectarPS->Caption="Ce&rrar puerto";
        btnConectarPS->Glyph->LoadFromFile(ImgsDir+"Cerrar.bmp");
        btnConectarBT->Enabled=true;
//        btnEnviar->Enabled=true;
        conPS=true;
      }
    }
    else
    {
          if(conBT)
          btnConectarBT->Click();
      cpSerial->Close();
      PublicaEstado("ESTADO : Se cerró el puerto con éxito", tmEstado);
      btnEnviar->Enabled=false;
                  btnConectarBT->Enabled=false;
                  btnConectarPS->Caption="&Conectar";
      btnConectarPS->Glyph->LoadFromFile(ImgsDir+"Abrir.bmp");
    }
}

//---------------------------------------------------------------------------
void TfrmPrincipal::SetConfig()
{
          bool eActivo;
    eActivo=cpSerial->Connected;
          if(cpSerial->Connected)
      AbrePuerto();
          cpSerial->SerialPort=frmConfigurar->cbPuerto->ItemIndex;
          cpSerial->BaudRate=frmConfigurar->cbVelocidad->ItemIndex;
          cpSerial->Parity=frmConfigurar->cbParidad->ItemIndex;
          cpSerial->StopBits=frmConfigurar->cbBParada->ItemIndex;
          cpSerial->FlowControl=frmConfigurar->cbCFlujo->ItemIndex;

          if(eActivo)
          AbrePuerto();
}

//---------------------------------------------------------------------------
void TfrmPrincipal::PublicaEstado(AnsiString Mensaje, TMsj Tipo)
{
          switch(Tipo)
    {
            case tmEvento:
                        reEstado->SelAttributes->Color=clGreen;
          break;

            case tmEstado:
                        reEstado->SelAttributes->Color=clBlue;
          break;
```

```
                case tmError:
                        reEstado->SelAttributes->Color=clRed;
                break;

                case tmRespuesta:
                        reEstado->SelAttributes->Color=clGray;
                break;

            case tmPaquete:
                        reEstado->SelAttributes->Color=clSilver;
                break;

                    default:
                        reEstado->SelAttributes->Color=clBlack;
              }
                reEstado->Lines->Add(Mensaje);
        }

        //---------------------------------------------------------------------------
        __fastcall TfrmPrincipal::TfrmPrincipal(TComponent* Owner)
                : TForm(Owner), conBT(false), conPS(false), enviando(false), fin(true)
        {
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::btnConectarBTClick(TObject *Sender)
        {
                if(!conBT)
                {
                        frmMS->ShowModal();
                  if(frmMS->rgConexion->ItemIndex==0)
            {
                DirE=InputBox("Destino","Escriba la dirección del esclavo","");
                if(StrADir(DirE, bDirE))
                {
                                        PublicaEstado("ESTADO : Conectado como
MAESTRO con : " + DirHex(bDirE), tmEstado);
                                EnviaCmd(CMD_MAKE_CONNECTION);
                }
                else
                                        PublicaEstado("ERROR : Dirección inválida
"+DirE, tmError);
                }
                    else
                                PublicaEstado("ESTADO : Conectado como ESCLAVO",
tmEstado);
                conBT=true;
                btnConectarBT->Caption="&Desconectar";
                btnConectarBT->Glyph->LoadFromFile(ImgsDir+"PCNOCOMM.bmp");
                btnEnviar->Enabled=true;
```

```
            lblEstado->Caption="CONECTADO";
            lblEstado->Font->Color=clBlue;
          }
          else
          {
                  EnviaCmd(CMD_DROP_CONNECTION);
            conBT=false;
            btnConectarBT->Caption="&Conectar";
            btnConectarBT->Glyph->LoadFromFile(ImgsDir+"PCCOMM.bmp");
            btnEnviar->Enabled=false;
            lblEstado->Caption="DESCONECTADO";
            lblEstado->Font->Color=clRed;
          }
                reEstado->SetFocus();
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::FormClose(TObject *Sender,
            TCloseAction &Action)
        {
          if(cpSerial->Connected)
            AbrePuerto();
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::btnConfigurarClick(TObject *Sender)
        {
                frmConfigurar->ShowModal();
                reEstado->SetFocus();
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::btnEnviarClick(TObject *Sender)
        {
        //        if(conBT)
          {
                        byte Data[255];
                if(!enviando && fin)
            {
              if(odAbrir->Execute())
              {
                NPaq=CreaCabecera(ExtractFileName(odAbrir->FileName),
tam_archivo(odAbrir->FileName), 0x01, Data);
                PublicaEstado("ESTADO : Se envio la cabecera del archivo
"+ExtractFileName(odAbrir->FileName), tmEstado);
                cpSerial->Write(Data, Data[0]);
                PublicaEstado("Tamaño  "+IntToStr(tam_archivo(odAbrir->FileName)),
tmEstado);
                Nombre=odAbrir->FileName;
                enviando=true;
```

```
                fin=false;
                pbProgreso->Visible=true;
                pbProgreso->Position=0;
                cont=0;
                btnEnviar->Caption="Ca&ncelar";
                                btnEnviar->Glyph-
>LoadFromFile(ImgsDir+"Eliminar.bmp");
              }
            }
            else
            {
                                CreaPaqResp(tdCancelar, 0x01, Data);
                    cpSerial->Write(Data, Data[0]);
              btnEnviar->Caption="&Enviar";
              btnEnviar->Glyph->LoadFromFile(ImgsDir+"Satlite.bmp");
              PublicaEstado("ESTADO : El transmisor cancelo la transferencia",
tmError);
                                enviando=false;
              fin=true;
                  }


          }
                reEstado->SetFocus();
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::cpSerialDataReceived(TObject *Sender,
            const char *Buffer, unsigned Length)
        {
                byte *Data;
          unsigned i;
          Data = new byte[Length];
          for(i=0; i<Length; ++i)
                Data[i]=Buffer[i];
          cpSerial->FlushQueue();
          if(TipoPaquete(Data)!=PACKETTYPE_DATA)
                  PublicaEstado("> " + PaqueteHex(Data), tmPaquete);
                switch(TipoPaquete(Data))
          {
            case PACKETTYPE_RESPONSE:
                        PublicaEstado("RESPUESTA : " + MsjRespuesta(Data),
tmRespuesta);
              break;

            case PACKETTYPE_EVENT:
                        PublicaEstado("EVENTO : " + MsjEvento(Data), tmEvento);
              EvtAccion(Data);
              if(EVT_DISCONNECT==Data[2])
                cpSerial->FlushQueue();
```

```
            break;

          case PACKETTYPE_DATA:
        //                PublicaEstado("DATA", tmPaquete);
            PaqueteData(Data, Length);
            break;

              case PACKETTYPE_UNKNOWN:
                    PublicaEstado("ERROR : Paquete desconocido", tmError);
            break;
          }
          if(TipoPaquete(Data)!=PACKETTYPE_DATA)
                  reEstado->Lines->Add("");
        }

        //---------------------------------------------------------------------------
        AnsiString TfrmPrincipal::MsjRespuesta(const byte *Data)
        {
              switch(Data[2])
          {
              case CMD_MAKE_CONNECTION:
              if(Data[4]!=MPSTATUS_OK)
              btnEnviar->Enabled=false;
              return "CMD_MAKE_CONNECTION ("+MsjTipoRespuesta(Data)+")";

            case CMD_CONTROLMODEMLINES:
              return                                    "CMD_CONTROLMODEMLINES
("+MsjTipoRespuesta(Data)+")";

            case CMD_RSSI_LINKQUAL:
                return "CMD_RSSI_LINKQUAL ("+MsjTipoRespuesta(Data)+")";

            case CMD_DROP_CONNECTION:
                cpSerial->FlushQueue();
                          btnEnviar->Enabled=false;
                return "CMD_DROP_CONNECTION ("+MsjTipoRespuesta(Data)+")";

            case CMD_READ_BDADDR:
                    PublicaEstado("ESTADO : Direccion Recibida : ", tmEstado);
                return "CMD_READ_BDADDR ("+MsjTipoRespuesta(Data)+")";

            case CMD_INFORMATION:
              return "CMD_INFORMATION  ("+MsjTipoRespuesta(Data)+")";

            case CMD_CONNECTION_SETUP:
                return "CMD_CONNECTION_SETUP ("+MsjTipoRespuesta(Data)+")";

           default:
                  return IntToHex(Data[2], 2);
          }
```

```
}

//---------------------------------------------------------------------------
void TfrmPrincipal::PaqueteData(byte *Data, unsigned length)
{
  byte i, j;
        byte Resp[5];
        switch(Data[2])
  {
        case tdCabecera:
        if(!enviando)
       {
        Nombre="";
        for(i=3; (Data[i]!='|')&&(i<length); ++i)
          Nombre=Nombre+char(Data[i]);
        if(i!=length)
        {
          PublicaEstado("Cabecera recibida : "+Nombre, tmEstado);
          destino.open(Nombre.c_str(), ios::binary);
          for(j=0, ++i; j<3 ; ++i, ++j)
            ptrNPaq[j]=Data[i];
          PublicaEstado("Numero de paquetes a recibir : "+IntToStr(NPaq),
tmEstado);
          pbProgreso->Position=0;
          pbProgreso->Visible=true;
          CreaPaqResp(tdRecibido, 0x01, Resp);
          cpSerial->Write(Resp, Resp[0]);
          pbProgreso->Position=0;
          pbProgreso->Visible=true;
          enviando=false;
          fin=false;
          cont=0;
          btnEnviar->Caption="Ca&ncelar";
        }
        else
        {
          CreaPaqResp(tdCancelar, 0x01, Data);
          cpSerial->Write(Data, Data[0]);
          PublicaEstado("ERROR : Cabecera recibida corrupta ", tmError);
          CreaPaqResp(tdCancelar, 0x01, Data);
                        cpSerial->Write(Data, Data[0]);
        }
       }
       else
       {
                                PublicaEstado("ERROR : Error en la transmision",
tmError);
        CreaPaqResp(tdCancelar, 0x01, Data);
        cpSerial->Write(Data, Data[0]);
```

```
        }
          break;

      case tdPaquete:
          if(!enviando)
        {
                  for(j=0, i=3; j<3 ; ++i, ++j)
                        ptrPaq[j]=Data[i];
                  for(i=7; i<Data[0]; ++i)
                  destino.write(&Data[i], sizeof(Data[i]));
                pbProgreso->Position=(Paq*100)/NPaq;
             CreaPaqResp(tdRecibido, 0x01, Resp);
                cpSerial->Write(Resp, Resp[0]);
              sbEstado->SimpleText="Paquete recibido : "+IntToStr(Paq);
              enviando=false;
        }
        else
        {
                                    PublicaEstado("ERROR : Error en la transmision",
tmError);
          CreaPaqResp(tdCancelar, 0x01, Data);
          cpSerial->Write(Data, Data[0]);
        }
          break;

      case tdCancelar:
          if(destino.is_open())
          destino.close();
        DeleteFile(Nombre);
        pbProgreso->Visible=false;
        CreaPaqResp(tdRecibido, 0x01, Resp);
        cpSerial->Write(Resp, Resp[0]);
        PublicaEstado("ESTADO : El transmisor canceló la transferencia",
tmError);
        reEstado->Lines->Add(" :: "+PaqueteHex(Resp));
        sbEstado->SimpleText="";
        fin=true;
        enviando=false;
        pbProgreso->Visible=false;
        btnEnviar->Caption="&Enviar";
          break;

      case tdRecibido:
          if(enviando && !fin)
        {
              sbEstado->SimpleText="Paquete enviado : "+IntToStr(cont);
          byte data[255], i, *ptrD;
                pbProgreso->Position=(cont*100)/NPaq;
          ++cont;
          data[0]=7; data[1]=0x01; data[2]=tdPaquete;
```

```
                data[3]=ptrCont[0]; data[4]=ptrCont[1];
                data[5]=ptrCont[2]; data[6]=ptrCont[3];
                        if(!origen.is_open())
                        origen.open(Nombre.c_str(), ios::binary);
                i=7;
                do
                {
                  origen.read(&data[i], 1);
                  data[0]++;
                  ++i;
                }
                while((i<TAM_PAQUETE)&&(!origen.eof()));
                if(origen.eof())
                {
                        origen.close();
                  fin=true;
                  --data[0];
                }
                cpSerial->Write(data, data[0]);
        //              reEstado->Lines->Add(" << "+PaqueteHex(data));
              }
              else
              {
                    CreaPaqResp(tdFin, 0x01, Resp);
                      cpSerial->Write(Resp, Resp[0]);
                      reEstado->Lines->Add(" :: "+PaqueteHex(Resp));
                cont=0;
                        if(origen.is_open())
                        origen.close();
                                        PublicaEstado("ESTADO : Transmision finalizada
"+Nombre, tmEstado);
                        pbProgreso->Visible=false;
              }
                break;

            case tdFin:
                fin=true;
              enviando=false;
                                pbProgreso->Visible=false;
                                PublicaEstado("ESTADO : Transmisión finalizada
"+Nombre, tmEstado);
              sbEstado->SimpleText="";
                                if(origen.is_open())
                origen.close();
                                if(destino.is_open())
                destino.close();
              sdGuardar->FileName=Nombre;
              if(sdGuardar->Execute())
                if(!MoveFile(Nombre.c_str(), sdGuardar->FileName.c_str()))
```

```
                    MessageDlg(" No se pudo mover el archivo ", mtError,
TMsgDlgButtons() << mbOK, 0);
                break;
          }
        }

        //---------------------------------------------------------------------------
        void __fastcall TfrmPrincipal::FormCreate(TObject *Sender)
        {
                ptrNPaq=reinterpret_cast<unsigned char *>(&NPaq);
          ptrCont=reinterpret_cast<unsigned char *>(&cont);
                ptrPaq=reinterpret_cast<unsigned char *>(&Paq);
        }

        //---------------------------------------------------------------------------
        void TfrmPrincipal::EnviaCmd(byte Cmd)
        {
                byte Data[20];
          int tam;
                switch(Cmd)
          {
                case CMD_MAKE_CONNECTION:
                Data[0]=0x0E;      Data[1]=0x00;      Data[2]=Cmd;      Data[3]=0x7F;
Data[4]=0x80;
              for(byte i=0; i<6; ++i)
                Data[5+i]=bDirE[i];
              Data[11]=0x11; Data[12]=0x01; Data[13]=0x00;
              tam=14;
                                PublicaEstado("Enviado CMD_MAKE_CONNECTION :
" + PaqueteHex(Data), tmEstado);
                break;

            case CMD_CONTROLMODEMLINES:
                Data[0]=0x07; Data[1]=0x00;
              Data[2]=CMD_CONTROLMODEMLINES; Data[3]=0x7F;
              Data[4]=0x01; Data[5]=0x07; Data[0]=0x00;
              tam=7;
                                PublicaEstado("Enviado
CMD_CONTROLMODEMLINES : " + PaqueteHex(Data), tmEstado);
                break;

            case CMD_RSSI_LINKQUAL:
                Data[0]=0x0A; Data[1]=0x00;
              Data[2]=CMD_RSSI_LINKQUAL; Data[3]=0x7F;
              for(byte i=0; i<6; ++i)
                Data[4+i]=bDirM[i];
              tam=10;
                                PublicaEstado("Enviado CMD_RSSI_LINKQUAL : " +
PaqueteHex(Data), tmEstado);
                break;
```

```
            case CMD_DROP_CONNECTION:
                Data[0]=0x06; Data[1]=0x00;
              Data[2]=CMD_DROP_CONNECTION; Data[3]=0x7F;
              Data[4]=0x81; Data[5]=0x01;
              tam=6;
                                PublicaEstado("Enviado  CMD_DROP_CONNECTION  :
" + PaqueteHex(Data), tmEstado);
                break;

                        case CMD_READ_BDADDR:
              Data[0]=0x04; Data[1]=0x00;
              Data[2]=0x02; Data[1]=0x7F;
              tam=4;
                                PublicaEstado("Enviado  CMD_READ_BDADDR  :  "  +
PaqueteHex(Data), tmEstado);
                break;

            case CMD_INFORMATION:
                Data[0]=0x05; Data[1]=0x00;
              Data[2]=0x17; Data[3]=0x7F;
              Data[4]=0x00;
              tam=5;
                                PublicaEstado("Enviado  CMD_INFORMATION  :  "  +
PaqueteHex(Data), tmEstado);
              break;

                        case CMD_CONNECTION_SETUP:
              Data[0]=0x0C; Data[1]=0x00;
              Data[2]=0x0E; Data[3]=0x7f;
              for(byte i=0; i<6; ++i)
                Data[4+i]=bDirM[i];
        /*      Data[4]=0x00; Data[5]=0x80;
              Data[6]=0x98; Data[7]=0x74;
              Data[8]=0x40; Data[9]=0x56; */
              Data[10]=0x80; Data[11]=0x01;
              tam=12;
                                PublicaEstado("Enviado CMD_CONNECTION_SETUP :
" + PaqueteHex(Data), tmEstado);
                break;

            default:
                tam=-1;
                }

          if(tam!=-1)
          {
                  cpSerial->FlushQueue();
                cpSerial->Write(Data, tam);
          }
```

```
}

//---------------------------------------------------------------------------
void TfrmPrincipal::EvtAccion(const byte *Data)
{
/*EVT_STATUS
EVT_INVALID_PKTSIZE
EVT_UNKNOWN_COMMAND
EVT_INQUIRY_RESULT
EVT_MODEM_STATUS
EVT_DISCONNECT
EVT_CONNECTION_SETUP*/
        switch(Data[2])
  {
                case EVT_INCOMING_CONNECTION:
        for(byte i=0; i<6; ++i)
        bDirM[i]=Data[5+i];
      EnviaCmd(CMD_RSSI_LINKQUAL);
        break;

                case EVT_CONNECTION_SETUP:
        for(byte i=0; i<6; ++i)
        bDirM[i]=Data[4+i];
      cpSerial->FlushQueue();
        EnviaCmd(CMD_CONNECTION_SETUP);
        break;
   }
/*EVT_LINK_KEY
EVT_LCL_FNAME
EVT_REM_FNAME
EVT_DEBUG_PACKET
EVT_SCO_CONNECT
EVT_SCO_DISCONNECT
EVT_SCO_INCOMING_SETUP
EVT_LOWPOWER_MODE
EVT_PINCODE_REQUEST
EVT_ADC*/

}
//---------------------------------------------------------------------------


/*
EVT_STATUS
EVT_INVALID_PKTSIZE
EVT_UNKNOWN_COMMAND
EVT_INQUIRY_RESULT
EVT_MODEM_STATUS
EVT_DISCONNECT
EVT_CONNECTION_SETUP
```

```
EVT_INCOMING_CONNECTION
EVT_LINK_KEY
EVT_LCL_FNAME
EVT_REM_FNAME
EVT_DEBUG_PACKET
EVT_SCO_CONNECT
EVT_SCO_DISCONNECT
EVT_SCO_INCOMING_SETUP
EVT_LOWPOWER_MODE
EVT_PINCODE_REQUEST
EVT_ADC
*/


/*
CMD_NO_OPERATION
CMD_READ_BDADDR
CMD_READ_SREG
CMD_WRITE_SREG
CMD_STORE_SREG
CMD_READ_COMMSPARM
CMD_WRITE_COMMSPARM
CMD_INQUIRY_REQ
CMD_DISCOVERABLE_MODE
CMD_CONNECTABLE_MODE
CMD_CONNECTION_SETUP
CMD_PAIR_INITIATE
CMD_PAIR_ACCEPT
CMD_TRUSTED_DB_COUNT
CMD_TRUSTED_DB_READ
CMD_TRUSTED_DB_DELETE
CMD_TRUSTED_DB_CHANGETYPE
CMD_TRUSTED_DB_ISTRUSTED
CMD_INFORMATION
CMD_SECURITY_MODE
CMD_GET_REM_FNAME
CMD_SET_LCL_FNAME
CMD_GET_LCL_FNAME
CMD_DEFAULT_SREG
CMD_GET_MODES
CMD_SCO_CONNECT
CMD_SCO_DISCONNECT
CMD_SCO_INCOMING_SETUP
CMD_SNIFF_REQUEST
CMD_PARK_REQUEST
CMD_PINCODE
CMD_GET_IO
*/
void __fastcall TfrmPrincipal::FormActivate(TObject *Sender)
```

```
        {
                TIniFile *ini;
                ini = new TIniFile(ChangeFileExt( Application->ExeName, ".INI"));
                frmConfigurar->cbPuerto->ItemIndex=ini->ReadInteger("Acceso",
"Puerto", frmConfigurar->cbPuerto->ItemIndex);
          frmConfigurar->cbVelocidad->ItemIndex=ini->ReadInteger("Acceso",
"Velocidad", frmConfigurar->cbVelocidad->ItemIndex);
          frmConfigurar->cbParidad->ItemIndex=ini->ReadInteger("Acceso",
"Paridad", frmConfigurar->cbParidad->ItemIndex);
          frmConfigurar->cbBParada->ItemIndex=ini->ReadInteger("Acceso",
"BitsParada", frmConfigurar->cbBParada->ItemIndex);
          frmConfigurar->cbCFlujo->ItemIndex=ini->ReadInteger("Acceso",
"ControlFlujo", frmConfigurar->cbCFlujo->ItemIndex);
                SetConfig();
          delete ini;
          ImgsDir=ExtractFilePath(Application->ExeName)+"Imgs\\";
          reEstado->SelAttributes->Color=clSilver;
          reEstado->Lines->Add("// Fecha : " + DateToStr(Date()));
          reEstado->SelAttributes->Color=clSilver;
          reEstado->Lines->Add("// Hora  : " + TimeToStr(Time()));
          reEstado->Lines->Add(""); reEstado->Lines->Add("");
        }
        //---------------------------------------------------------------------------

        void __fastcall TfrmPrincipal::btnSalirClick(TObject *Sender)
        {
        if(conPS)
                cpSerial->Close();
        }
        //---------------------------------------------------------------------------



        void __fastcall TfrmPrincipal::btnConectarPSClick(TObject *Sender)
        {
                AbrePuerto();
                reEstado->SetFocus();
        }
        //---------------------------------------------------------------------------

        void __fastcall TfrmPrincipal::cpSerialQueueEmpty(TObject *Sender)
        {
        //        PublicaEstado("Vacio...Queue", tmEstado);
        }
        //---------------------------------------------------------------------------



        void __fastcall TfrmPrincipal::cpSerialTxEmpty(TObject *Sender)
        {
```

```
        PublicaEstado("Vacio...TX", tmEstado);
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::cpSerialRxChar(TObject *Sender)
{
//        PublicaEstado("Caracter...RX", tmEstado);
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::cpSerialRxEventChar(TObject *Sender)
{
//        PublicaEstado("Event char...RX", tmEstado);
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::cpSerialSendBroken(TObject *Sender)
{
        PublicaEstado("Send broken", tmEstado);
}
//---------------------------------------------------------------------------


void __fastcall TfrmPrincipal::Guardar1Click(TObject *Sender)
{
        reEstado->Lines->SaveToFile("EstadoConnBT.rtf");
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::Limpiar1Click(TObject *Sender)
{
        reEstado->Clear();
  reEstado->SelAttributes->Color=clSilver;
  reEstado->Lines->Add("// Fecha : " + DateToStr(Date()));
  reEstado->SelAttributes->Color=clSilver;
  reEstado->Lines->Add("// Hora  : " + TimeToStr(Time()));
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::Seleccionartodo1Click(TObject *Sender)
{
        reEstado->SelectAll();
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::Copiar1Click(TObject *Sender)
{
        reEstado->CopyToClipboard();
}
//---------------------------------------------------------------------------
```

uMain.h:

```
//---------------------------------------------------------------------------

#ifndef uMainH
#define uMainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "CommPort.h"
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <fstream.h>
#include "uTipos.h"
#include <ImgList.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
class TfrmPrincipal : public TForm
{
__published:   // IDE-managed Components
        TPanel *Panel1;
        TCommPort *cpSerial;
        TRichEdit *reEstado;
        TStatusBar *sbEstado;
        TSplitter *Splitter1;
        TOpenDialog *odAbrir;
        TSaveDialog *sdGuardar;
        TBitBtn *btnConectarBT;
        TBitBtn *btnEnviar;
        TBitBtn *btnSalir;
        TBitBtn *btnConfigurar;
        TProgressBar *pbProgreso;
        TBitBtn *btnConectarPS;
        TImageList *imImagenes;
        TLabel *lblEstado;
        TPopupMenu *pmEstado;
        TMenuItem *Guardar1;
        TMenuItem *Limpiar1;
        TMenuItem *Seleccionartodo1;
        TMenuItem *N1;
        TMenuItem *Copiar1;
        void __fastcall btnConectarBTClick(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall btnConfigurarClick(TObject *Sender);
        void __fastcall btnEnviarClick(TObject *Sender);
```

```
        void __fastcall cpSerialDataReceived(TObject *Sender, const char
*Buffer,
        unsigned Length);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall btnSalirClick(TObject *Sender);
        void __fastcall btnConectarPSClick(TObject *Sender);
        void __fastcall cpSerialQueueEmpty(TObject *Sender);
        void __fastcall cpSerialTxEmpty(TObject *Sender);
        void __fastcall cpSerialRxChar(TObject *Sender);
        void __fastcall cpSerialRxEventChar(TObject *Sender);
        void __fastcall cpSerialSendBroken(TObject *Sender);
        void __fastcall Guardar1Click(TObject *Sender);
        void __fastcall Limpiar1Click(TObject *Sender);
        void __fastcall Seleccionartodo1Click(TObject *Sender);
        void __fastcall Copiar1Click(TObject *Sender);
private:        // User declarations
public:         // User declarations
        unsigned NPaq, Paq, cont;
  byte *ptrNPaq, *ptrPaq, *ptrCont;
  ofstream destino;
  ifstream origen;
        AnsiString Nombre;
        AnsiString ImgsDir;
  AnsiString DirM, DirE;
  byte bDirM[6], bDirE[6];
  bool enviando;
  bool conBT, conPS;
  bool fin;

        __fastcall TfrmPrincipal(TComponent* Owner);
        void AbrePuerto();
        void SetConfig();
        void PublicaEstado(AnsiString Mensaje, TMsj Tipo);
  void PaqueteData(byte *Data, unsigned length);
  void EnviaCmd(byte CMD);
        void EvtAccion(const byte *Data);
        AnsiString MsjRespuesta(const byte *Data);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmPrincipal *frmPrincipal;
//---------------------------------------------------------------------------
#endif
```

uMs.cpp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
```

```
#pragma hdrstop

#include "uMS.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmMS *frmMS;
//---------------------------------------------------------------------------
__fastcall TfrmMS::TfrmMS(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

uMs.h:

```
//---------------------------------------------------------------------------

#ifndef uMSH
#define uMSH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TfrmMS : public TForm
{
__published:   // IDE-managed Components
        TRadioGroup *rgConexion;
        TBitBtn *BitBtn1;
private:        // User declarations
public:         // User declarations
        __fastcall TfrmMS(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmMS *frmMS;
//---------------------------------------------------------------------------
#endif
```

uPaquetes.cpp:

```
//---------------------------------------------------------------------------


#include "Math.h"
#pragma hdrstop
```

```
#include "uPaquetes.h"
#include "uMain.h"

//---------------------------------------------------------------------------
byte StrAByte(AnsiString Cad, bool &bien)
{
        byte pos, lon;
  byte valor=0;
  pos=Cad.Length();
  lon=pos;
        do
  {
        if((Cad[pos]>='0')&&(Cad[pos]<='9'))
        valor+=pow(16, lon-pos)*(Cad[pos]-48);
        else if((Cad[pos]>='A')&&(Cad[pos]<='F'))
        valor+=pow(16, lon-pos)*(Cad[pos]-55);
    else  if((Cad[pos]>='a')&&(Cad[pos]<='f'))
        valor+=pow(16, lon-pos)*(Cad[pos]-87);
    else
    {
        bien=false;
        return 0;
    }
                --pos;
  }
  while(pos>=1);
  bien=true;
  return valor;
}

//---------------------------------------------------------------------------
bool StrADir(AnsiString asDir, byte *bDir)
{
        if(asDir.Length()==12)
  {
        AnsiString Num;
    bool bien;
        for(byte i=0, j=0; i<12; i+=2, ++j)
    {
        bDir[j]=StrAByte(asDir.SubString(i+1, 2), bien);
      if(!bien)
        return false;
    }
    return true;
  }
  return false;
}

//---------------------------------------------------------------------------
byte TipoPaquete(const byte *Data)
```

```
        {
                byte nRetVal=PACKETTYPE_UNKNOWN;
                byte nTemp;
                if(Data[1]==0)
                {
                        nTemp = Data[2] & 0x80;
                        if( (Data[0] >= 5) && (nTemp == 0) )
                nRetVal = PACKETTYPE_RESPONSE;
                else if((Data[0] >= 4) && (nTemp != 0))
                        nRetVal = PACKETTYPE_EVENT;
                }
                else
                        nRetVal = PACKETTYPE_DATA;
                return nRetVal;
        }

        //---------------------------------------------------------------------------
        AnsiString MsjEvento(const byte *Data)
        {
                switch(Data[2])
          {
                case EVT_STATUS:                        return "EVT_STATUS";
            case EVT_INVALID_PKTSIZE:  return "EVT_INVALID_PKTSIZE";
            case EVT_UNKNOWN_COMMAND: return
"EVT_UNKNOWN_COMMAND";
            case EVT_INQUIRY_RESULT:return "EVT_INQUIRY_RESULT";
            case EVT_MODEM_STATUS:          return "EVT_INQUIRY_RESULT";
            case EVT_DISCONNECT:                    return
"EVT_DISCONNECT";
            case EVT_CONNECTION_SETUP:      return
"EVT_CONNECTION_SETUP";
            case EVT_INCOMING_CONNECTION:       return
"EVT_INCOMING_CONNECTION";
            case EVT_LINK_KEY:                  return "EVT_LINK_KEY";
            case EVT_LCL_FNAME:                     return
"EVT_LCL_FNAME";
            case EVT_REM_FNAME:                     return
"EVT_REM_FNAME";
            case EVT_DEBUG_PACKET:          return "EVT_DEBUG_PACKET";
            case EVT_SCO_CONNECT:           return "EVT_SCO_CONNECT";
            case EVT_SCO_DISCONNECT:        return
"EVT_SCO_DISCONNECT";
            case EVT_SCO_INCOMING_SETUP: return
"EVT_SCO_INCOMING_SETUP";
            case EVT_LOWPOWER_MODE:             return
"EVT_LOWPOWER_MODE";
            case EVT_PINCODE_REQUEST:       return
"EVT_PINCODE_REQUEST";
            case EVT_ADC:                           return "EVT_ADC";
          }
```

```
  return "";
}

//---------------------------------------------------------------------------
void CreaPaqResp(TData Respuesta, byte canal, byte *Data)
{
        Data[0]=0x03; Data[1]=canal; Data[2]=Respuesta;
}

//---------------------------------------------------------------------------
unsigned CreaCabecera(AnsiString Nombre, unsigned Longitud, byte canal,
byte *Data)
{
        byte i, *ptrLon;
  unsigned lon;
        Data[0]=3; Data[1]=canal; Data[2]=tdCabecera;
        for(i=0; i<Nombre.Length(); ++i, ++Data[0])
        Data[3+i]=Nombre[i+1];
  Data[Nombre.Length()+3]='|';
  lon=Longitud/(TAM_PAQUETE-3)+1;
        ptrLon=reinterpret_cast<byte *>(&lon);
  for(i=0; i<4; ++i)
        Data[Nombre.Length()+4+i]=ptrLon[i];
  Data[0]+=5;
  return lon;
}

//---------------------------------------------------------------------------
unsigned tam_archivo(String arch)
{
  unsigned longitud, posic;
  FILE *f;
        f = fopen (arch.c_str(), "rb");
  posic = ftell (f);
  fseek (f, 0, SEEK_END);
  longitud = ftell (f);
  fseek (f, posic, SEEK_SET);
  fclose(f);
  return longitud;
}

//---------------------------------------------------------------------------
AnsiString PaqueteHex(const byte *Data)
{
        byte i;
  AnsiString Ret="";
        for(i=0; i<Data[0]; ++i)
        Ret=Ret+IntToHex(Data[i], 2)+" ";
  return Ret;
}
```

```
//---------------------------------------------------------------------------
AnsiString DirHex(const byte *Data)
{
        byte i;
  AnsiString Ret="";
        for(i=0; i<6; ++i)
        Ret=Ret+IntToHex(Data[i], 2)+" ";
  return Ret;
}

//---------------------------------------------------------------------------
AnsiString MsjTipoRespuesta(const byte *Data)
{
        switch(Data[4])
  {
        case MPSTATUS_OK:
        return "MPSTATUS_OK";
    case MPSTATUS_ILLEGAL_COMMAND:
        return "MPSTATUS_ILLEGAL_COMMAND";
    case MPSTATUS_NO_CONNECTION:
        return "MPSTATUS_NO_CONNECTION";
    case MPSTATUS_HARDWARE_FAIL:
        return "MPSTATUS_HARDWARE_FAIL";
    case MPSTATUS_PAGE_TIMEOUT:
        return "MPSTATUS_PAGE_TIMEOUT";
    case MPSTATUS_INVALID_BDADDR:
        return "MPSTATUS_INVALID_BDADDR";
    case MPSTATUS_INVALID_UUID:
        return "MPSTATUS_INVALID_UUID";
    case MPSTATUS_SDPSEARCH_FAIL:
        return "MPSTATUS_SDPSEARCH_FAIL";
    case MPSTATUS_CHANNEL_NOT_OPEN:
        return "MPSTATUS_CHANNEL_NOT_OPEN";
    case MPSTATUS_ESTABLISH_FAIL:
        return "MPSTATUS_ESTABLISH_FAIL";
    case MPSTATUS_BUSY_TRY_LATER:
        return "MPSTATUS_BUSY_TRY_LATER";
        }
        return IntToHex(Data[4], 2);
}

/*
MPSTATUS_OK
MPSTATUS_ILLEGAL_COMMAND
MPSTATUS_NO_CONNECTION
MPSTATUS_HARDWARE_FAIL
MPSTATUS_PAGE_TIMEOUT
MPSTATUS_AUTH_FAIL
MPSTATUS_KEY_MISSING
```

MPSTATUS_MEMORY_FULL
MPSTATUS_CONN_TIMEOUT
MPSTATUS_MAX_NR_OF_CONNS
MPSTATUS_MAX_NR_OF_SCO
MPSTATUS_MAX_NR_OF_ACL
MPSTATUS_COMMAND_DISALLOWED
MPSTATUS_REJ_BY_REMOTE_NO_RES
MPSTATUS_REJ_BY_REMOTE_SEC
MPSTATUS_REJ_BY_REMOTE_PERS
MPSTATUS_HOST_TIMEOUT
MPSTATUS_UNSUPPORTED_FEATURE
MPSTATUS_ILLEGAL_FORMAT
MPSTATUS_OETC_USER
MPSTATUS_OETC_LOW_RESOURCE
MPSTATUS_OETC_POWERING_OFF
MPSTATUS_CONN_TERM_LOCAL_HOST
MPSTATUS_AUTH_REPEATED
MPSTATUS_PAIRING_NOT_ALLOWED
MPSTATUS_UNKNOWN_LMP_PDU
MPSTATUS_UNSUPPORTED_REM_FEATURE
MPSTATUS_SCO_OFFSET_REJECTED
MPSTATUS_SCO_INTERVAL_REJECTED
MPSTATUS_SCO_AIR_MODE_REJECTED
MPSTATUS_INVALID_LMP_PARAMETERS
MPSTATUS_UNSPECIFIED
MPSTATUS_UNSUPP_LMP_PARAM
MPSTATUS_ROLE_CHANGE_NOT_ALLOWED
MPSTATUS_LMP_RESPONSE_TIMEOUT
MPSTATUS_LMP_TRANSACTION_COLLISION
MPSTATUS_LMP_PDU_NOT_ALLOWED
MPSTATUS_ENC_MODE_NOT_ACCEPTABLE
MPSTATUS_UNIT_KEY_USED
MPSTATUS_QOS_NOT_SUPPORTED
MPSTATUS_INSTANT_PASSED
MPSTATUS_PAIR_UNIT_KEY_NO_SUPPORT
MPSTATUS_CHANNEL_CLASS_NO_SUPPORT
MPSTATUS_PACKET_INVALID_SIZE
MPSTATUS_INVALID_REGISTER
MPSTATUS_INVALID_REGVALUE
MPSTATUS_NONVOL_ERROR
MPSTATUS_INVALID_COMMPARMS
MPSTATUS_INVALID_PARAMETER
MPSTATUS_INVALID_UUID
MPSTATUS_MAX_CONNECTIONS_ACTIVE
MPSTATUS_INVALID_CHANNELID
MPSTATUS_CHANNEL_NOT_OPEN
MPSTATUS_NO_CONNECTION_PENDING
MPSTATUS_NOT_IDLE
MPSTATUS_INVALID_BDADDR
MPSTATUS_DB_PSWRITE_FAIL_KEY

MPSTATUS_DB_PSWRITE_FAIL_INDEX
MPSTATUS_DB_PSDELETE_FAIL_KEY
MPSTATUS_DB_LINKKEY_NOT_PRESENT
MPSTATUS_DB_PERSIST_FULL
MPSTATUS_DB_UNKNOWN_TYPE
MPSTATUS_DB_INVALID_ITEMNO
MPSTATUS_DB_NOT_TRUSTED
MPSTATUS_INVALID_FNAME_LENGTH
MPSTATUS_DB_PSWRITE_FAIL_FNAME
MPSTATUS_INVALID_LICENSE
MPSTATUS_FAIL
MPSTATUS_BUSY_TRY_LATER
MPSTATUS_SDP_REGISTER_BUSY
MPSTATUS_RFC_REGISTER_FAIL
MPSTATUS_TOO_MANY_PROFILES
MPSTATUS_MALLOC_FAILED
MPSTATUS_NULL_POINTER
MPSTATUS_INQUIRY_IN_PROGRESS
MPSTATUS_INQUIRY_NOT_IN_PROGRESS
MPSTATUS_TIMEOUT
MPSTATUS_CANCELLED
MPSTATUS_WRITESCAN_IN_PROGRESS
MPSTATUS_PAIR_IN_PROGRESS
MPSTATUS_PAIR_NOT_IN_PROGRESS
MPSTATUS_UNKNOWN_CHANNEL
MPSTATUS_UNKNOWN_MUX
MPSTATUS_GET_RSSI_PENDING
MPSTATUS_SDPSEARCH_FAIL
MPSTATUS_SDP_INVALID_RESPONSE_SIZE
MPSTATUS_SDPSERVICE_FAIL
MPSTATUS_SDP_INSTANCE_MISSING
MPSTATUS_SDP_ATTRIBUTE_FAIL
MPSTATUS_REMOTE_REFUSAL
MPSTATUS_ESTABLISH_FAIL
MPSTATUS_INCORRECT_STATE
MPSTATUS_NOT_INITIALISED
MPSTATUS_MUX_FULL
MPSTATUS_REGISTER_FAIL
MPSTATUS_UNKNOWN_CONFIG
MPSTATUS_NOT_IMPLEMENTED
MPSTATUS_SCO_BANDWIDTH_FULL
MPSTATUS_PEER_NOT_SCO_CAPABLE
MPSTATUS_INVALID_SCO_HANDLE
MPSTATUS_SCO_NOT_OPEN
MPSTATUS_INVALID_ADDRESS
MPSTATUS_INVALID_TIMEOUT
MPSTATUS_INVALID_SERVER_CHANNEL
MPSTATUS_INVALID_FRAMESIZE
MPSTATUS_INVALID_SCAN_VALUE
MPSTATUS_INVALID_PINCODE

```
MPSTATUS_INVALID_SECURITYMODE
MPSTATUS_CONTACT_TDK


*/
#pragma package(smart_init)
```

uPaquetes.H:

```
//---------------------------------------------------------------------------
#include "uTipos.h"
#include "MpStatus.h"
#include <system.hpp>
#ifndef uPaquetesH
#define uPaquetesH
#define TAM_PAQUETE 255


byte TipoPaquete(const byte *Data);
AnsiString MsjEvento(const byte *Data);
AnsiString MsjTipoRespuesta(const byte *Data);
AnsiString PaqueteHex(const byte *Data);
AnsiString DirHex(const byte *Data);
bool StrADir(AnsiString asDir, byte *bDir);
void CreaPaqResp(TData Respuesta, byte canal, byte *Data);
unsigned tam_archivo(String arch);
unsigned CreaCabecera(AnsiString Nombre, unsigned Longitud, byte canal,
byte *Data);

#endif
```

uTipos.h:

```
#ifndef Tipos_H
#define Tipos_H
#include "BMHostProtocol.h"
//---------------------------------------------------------------------------
enum TMsj {tmEvento, tmEstado, tmError, tmRespuesta, tmPaquete};
enum TData {tdCabecera, tdPaquete, tdCancelar, tdRecibido, tdFin};
enum TPaquete {PACKETTYPE_UNKNOWN, PACKETTYPE_RESPONSE,
PACKETTYPE_EVENT, PACKETTYPE_DATA};
typedef unsigned char byte;
//---------------------------------------------------------------------------
#endif
```