UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO DE UN SISTEMA DE CONTROL DE ACCESO PARA PUERTAS UTILIZANDO EQUIPOS CELULARES Y PERIFÉRICO BLUETOOTH

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

ALEX ALBERTO TANTALEAN ANGULO

PROMOCIÓN

2006-I

LIMA – PERÚ
2011

DISEÑO DE UN SISTEMA DE CONTROL DE ACCESO PARA PUERTAS UTILIZANDO EQUIPOS CELULARES Y PERIFÉRICO BLUETOOTH

Dedicado a mi madre
por su cariño, paciencia
y por todo el apoyo brindado.

SUMARIO

El presente informe describe el diseño de un sistema de control de acceso físico utilizando la tecnología Bluetooth y los teléfonos celulares.

Esta solución surge de la necesidad que tienen las empresas por restringir el ingreso de las personas a ciertas áreas. Esta restricción se da por niveles de seguridad, por horarios de ingreso, etc. Las empresas buscan que el acceso a sus recintos sea seguro, cómodo y organizado.

Para cubrir la necesidad expuesta, se plantea el diseño de un sistema moderno y fácil de usar, que permita a los usuarios utilizar sus teléfonos celulares como llaves de acceso a las áreas restringidas. De esta forma se puede sustituir las cerraduras y llaves comunes por sistemas electrónicos.

ÍNDICE

INTRODUCCIÓN

Hoy en día el teléfono móvil se ha convertido en un instrumento casi indispensable para cualquier persona como herramienta de comunicación con su entorno: amigos, familia, negocios, etc. Atrás quedaron aquellos tiempos en los que el teléfono celular era un instrumento que servía simplemente para hablar. Después llegaron los SMS, MMS, correo electrónico, juegos, cámaras de fotos, etc.

Una de las principales aportaciones que ha ofrecido el mundo de la telefonía móvil es la incorporación de la tecnología Bluetooth. Esta tecnología permite la comunicación con otros dispositivos (incluso con otros dispositivos que no son teléfonos), intercambiar archivos y todo ello sin costo. Esta última es una de las características que han hecho que el Bluetooth en los teléfonos celulares haya tenido una gran acogida entre los usuarios.

La solución presentada en este informe se enfoca en el diseño de un sistema de acceso para puertas, el cual nace de la necesidad que tienen las empresas de implementar mecanismos de seguridad en sus instalaciones. Se tiene que considerar que el acceso físico a áreas restringidas es sólo una parte del sistema de seguridad.

El diseño utiliza una señal de radio para identificar al usuario del sistema de acceso, antes de permitirle o negarle el ingreso. De este modo, el Bluetooth encuentra otra forma de uso. La tecnología Bluetooth se presenta como alternativa para controlar el acceso a determinadas zonas.

El presente trabajo abarca la programación de un periférico Bluetooth y un teléfono celular. Al ser parte de un sistema de seguridad la comunicación debería estar encriptada. Sin embargo, en este diseño, no se ha encriptado la comunicación Bluetooth.

El informe de suficiencia está dividido en cuatro capítulos. El primer capítulo hace la descripción del problema y la situación actual del control de acceso en los edificios.

El segundo capítulo describe el concepto de la tecnología inalámbrica Bluetooth. Se explica en modo general los protocolos, perfiles, versiones y seguridad en Bluetooth. Además se menciona la topología de red inalámbrica Bluetooth: Piconet y scatternet.

El tercer capítulo abarca la tecnología Java para dispositivos móviles (J2ME), se describe las máquinas virtuales J2ME, los estados de los MIDlets y el ciclo de vida de los MIDlets. Se explican las configuraciones y perfiles que se implementan sobre las máquinas virtuales de J2ME. También se describe el API Java para Bluetooth que es el

JSR-82. El segundo y tercer capítulo en conjunto forman el fundamento teórico del informe que permite entender la solución propuesta.

El cuarto capítulo es la metodología para la solución del problema, donde se explica el diseño de la solución al problema planteado en el primer capítulo. Se describe los elementos de hardware utilizados en el proyecto.

CAPÍTULO I
PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción del problema

Los edificios deben tener un control de acceso para puertas en sus instalaciones, que permita el ingreso ordenado de personas, además de clasificar algunos ambientes de la construcción como zonas restringidas. Por razones de seguridad, las personas no pueden tener ingreso libre a todas las áreas del edificio.

Los avances tecnológicos han permitido que se puedan sustituir las cerraduras y las llaves comunes por sistemas electrónicos. Para lograr esta tarea existen varias tecnologías como:

1) Tarjetas.
2) Código de barras.
3) Identificadores biométricos.
4) Etc.

Es evidente, que las empresas de seguridad siguen buscando nuevos y variados modos de controlar el ingreso físico de personas, incluyendo nuevas tecnologías.

1.2 Objetivo

Diseñar un sistema de control de acceso físico que utilice Bluetooth y que permita a los usuarios utilizar sus teléfonos celulares como llaves de acceso a las áreas restringidas.

CAPÍTULO II
BLUETOOTH

2.1 Conectividad inalámbrica

El término "Era de la Información" viene del intercambio de datos en cantidades masivas entre dispositivos informáticos usando formas de comunicación inalámbrica y alambrada.

La incrementada dependencia en el Internet y la necesidad de quedarse conectado en cualquier parte, todo el tiempo, han permitido avances en computación móvil y comunicaciones. Las comunicaciones sin alambres se dan por satélites, teléfonos inalámbricos, teléfonos celulares, dispositivos de control remoto, etc. Sin embargo, en recientes años la industria de comunicaciones inalámbricas ha visto un crecimiento explosivo.

La comunicación inalámbrica de largo alcance invariablemente usa radio frecuencia (RF). Típicamente, las comunicaciones de largo alcance usan la parte licenciada del espectro RF. Las comunicaciones de corto alcance pueden usar RF o infrarrojo y típicamente usan las partes no licenciadas (libres) del espectro de frecuencia.

Hay muchos estándares inalámbricos de corto alcance, pero los tres principales son: Infrarrojo de Infrared Data Association (IrDA), tecnología inalámbrica Bluetooth, y redes de área local inalámbrica (WLAN). WLAN es también conocido como IEEE 802.11, y viene en diferentes variantes (802.11b, 802.11g, 802.11a, 802.11n, etc.), los cuales operan en 2.4 gigahertz (GHz) o 5 GHz. El IrDA creó un sistema de comunicaciones inalámbrico que hace uso de la luz infrarroja. Mientras que la comunicación RF puede penetrar muchos objetos, IrDA está limitado a la línea de vista.

La tecnología 802.11b y la tecnología inalámbrica Bluetooth se comunican en los 2.4 GHz de la banda RF pero están apuntados a diferentes segmentos de mercado. La tecnología 802.11 tiene un alcance más largo pero consume sustancialmente más potencia que la tecnología inalámbrica Bluetooth. La Tabla Nº 2.1 provee una comparación de estas tres tecnologías.

Las comunicaciones inalámbricas permiten que la informática y los dispositivos de comunicaciones puedan ser usa dos en casi cualquier parte y en nuevas formas progresivas. El incremento en dispositivos de Internet móvil inalámbricos es prueba que

la conectividad inalámbrica es un fenómeno generalizado.

TABLA Nº 2.1 Comparación de las tecnologías de comunicación inalámbrica (1)

Característica y Función	IrDA	LAN Inalámbrica	Comunicación Bluetooth
Tipo de conexión	Infrarrojo, haz estrecho, línea de vista	Espectro ensanchado, esférico	Espectro ensanchado, esférico
Espectro	Óptico 850-900 nm	RF 2.4 GHz (5 GHz para 802.11a/n)	RF 2.4 GHz
Potencia de transmisión	40-500 mW/Sr	100 mW	10-100 mW
Máxima velocidad de datos	9600 bps-16 Mbps (muy raro)	11 Mbps (54 Mbps para 802.11a, 802.11g)	3 Mbps
Alcance	1 m	100 m	10-100 m
Dispositivos soportados	2	Se conecta a través de un punto de acceso	8 (activo), 200 (pasivo)
Canales de voz	No	No	Si
Direccionamiento	32 bits ID físico	48 bits MAC	48 bits MAC

2.2 Definición

La tecnología inalámbrica Bluetooth es una especificación abierta para la tecnología de radio de corto alcance, bajo costo y baja potencia, para comunicaciones inalámbricas ad hoc de voz y datos en cualquier parte del mundo.

1) Una especificación abierta quiere decir que está disponible públicamente y libre de derechos de autor.

2) Tecnología de radio de corto alcance significa que los dispositivos pueden comunicarse sobre al aire usando ondas de radio en una distancia de 10 metros. Con potencia de transmisión más alta el rango incrementa hasta 100 metros.

3) Como la comunicación está dentro de un corto alcance, los radios son de baja potencia y son apropiados para dispositivos portátiles operados por baterías.

4) La tecnología inalámbrica Bluetooth soporta voz y datos, permitiendo a los dispositivos comunicar cualquier tipo de contenido.

5) La tecnología inalámbrica Bluetooth trabaja en cualquier parte del mundo porque opera en 2.4 GHz en la banda industrial, científica y médica (ISM), que tiene licencia libre y está disponible globalmente.

Como la banda de frecuencia ISM está disponible para uso general, otros dispositivos operan en esta banda (WLAN, teléfonos inalámbricos, hornos microondas). La tecnología

inalámbrica Bluetooth está diseñada para ser muy robusta en interferencia de otros dispositivos.

Clasificación

Los dispositivos Bluetooth se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión, siendo totalmente compatibles los dispositivos de una clase con los dispositivos de las otras, ver la Tabla Nº 2.2.

TABLA Nº 2.2 Potencia de transmisión (8)

Clase	Potencia Máxima Permitida (mW)	Potencia Máxima Permitida (dBm)	Rango
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2. 5 mW	4 dBm	~10 metros
Clase 3	1 mW	0 dBm	~1 metro

Los dispositivos Bluetooth también pueden clasificarse según su ancho de banda, ver la Tabla Nº 2. 3.

TABLA Nº 2.3 Ancho de Banda (8)

Versión	Ancho de Banda
Versión 1.2	1 Mbit/s
Versión 2.0 + EDR	3 Mbit/s
Versión 3.0 + HS	24 Mbit/s

2.3 Historia de Bluetooth

Bluetooth obtuvo su nombre del rey vikingo danés Harald Blåtand (Bluetooth), que reinó en el siglo X (años 940-981). Durante su reinado Harald unió y controló Dinamarca y Noruega.

Las comunicaciones Bluetooth se originaron en 1994, cuando Ericsson inició un estudio para encontrar alternativas a la conexión de teléfonos móviles con sus accesorios. Los ingenieros miraron una interfaz de radio de baja potencia y bajo costo para eliminar cables entre los dispositivos. Sin embargo, los ingenieros también se dieron cuenta que para una tecnología exitosa, ésta tenía que ser un estándar abierto, no una propietaria.

En inicios de 1998, Ericsson reunió a Intel, International Business Machines (IBM), Nokia y Toshiba y formó el Bluetooth Special Interest Group (SIG) para desarrollar una especificación abierta para la tecnología inalámbrica Bluetooth. En julio de 1999, el Bluetooth SIG publicó la versión 1.0 de la especificación Bluetooth. En diciembre de 1999 se unieron a Bluetooth SIG 4 compañías: 3Com, Agere, Microsoft y Motorola (Figura 2.1).

El interés en el Bluetooth SIG ha crecido, y actualmente hay miles de compañías asociadas.

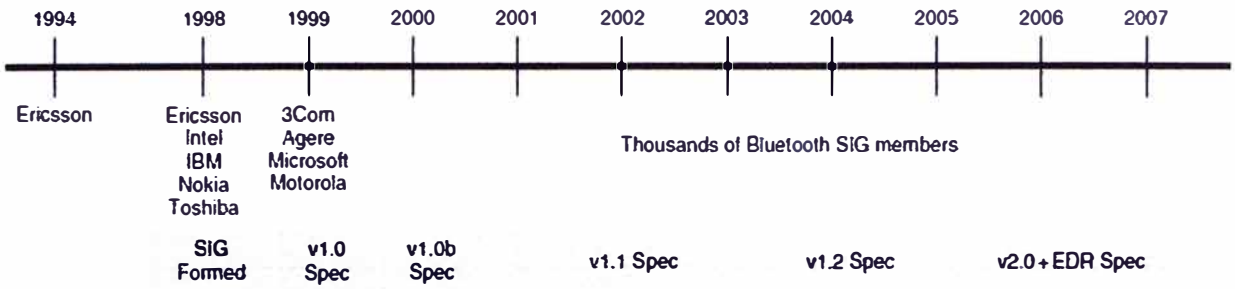

Fig. 2.1 Línea de tiempo del Bluetooth SIG (1)

2.4 Versiones

Bluetooth v.1.1: Ericsson inició un estudio para investigar la viabilidad de una nueva interfaz de bajo costo y consumo para la interconexión vía radio (eliminando así cables) entre dispositivos como teléfonos móviles y otros accesorios. El estudio partía de un largo proyecto que investigaba unos multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado MC link. Conforme este proyecto avanzaba se fue haciendo claro que éste tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, ya que tenía como principal virtud que se basaba en un chip de radio.

Bluetooth v.1.2: A diferencia de la 1.1, provee una solución inalámbrica complementaria para co-existir Bluetooth y Wi-Fi en el espectro de los 2.4 GHz, sin interferencia entre ellos. La versión 1.2 usa la técnica "Adaptive Frequency Hopping (AFH)", que ejecuta una transmisión más eficiente y un cifrado más seguro. Para mejorar las experiencias de los usuarios, la V1.2 ofrece una calidad de voz (Voice Quality – Enhanced Voice Processing) con menor ruido ambiental, y provee una más rápida configuración de la comunicación con los otros dispositivos Bluetooth dentro del rango del alcance, como pueden ser PDAs, HIDs (Human Interface Devices), computadoras portátiles, computadoras de escritorio, Headsets, impresoras y celulares.

Bluetooth v.2.0: Creada para ser una especificación separada, principalmente incorpora la técnica "Enhanced Data Rate" (EDR) que le permite mejorar las velocidades de transmisión en hasta 3Mbps a la vez que intenta solucionar algunos errores de la especificación 1.2.

Bluetooth v.2.1: Simplifica los pasos para crear la conexión entre dispositivos, además el consumo de potencia es 5 veces menor.

Bluetooth v3.0: Aumenta considerablemente la velocidad de transferencia. La idea es que el nuevo Bluetooth trabaje con WiFi, de tal manera que sea posible lograr mayor

velocidad en los smartphones (mediados 2009).

Bluetooth v4.0: Disminuye drásticamente el consumo de energía, ampliando la cantidad de aplicaciones, permitiendo la incorporación de receptores y transmisores Bluetooth en dispositivos pequeños como relojes, reproductores portátiles, instrumental médico, etc. Se mejora la seguridad con encriptadores AES-128 (finales 2010).

2.5 Arquitectura Bluetooth

La pila de protocolos Bluetooth puede dividirse ampliamente en dos componentes: el Bluetooth host y el Bluetooth controller. El Host Controller Interface (HCI) provee una interfaz estandarizada entre el Bluetooth host y el Bluetooth controller (Figura 2.2).

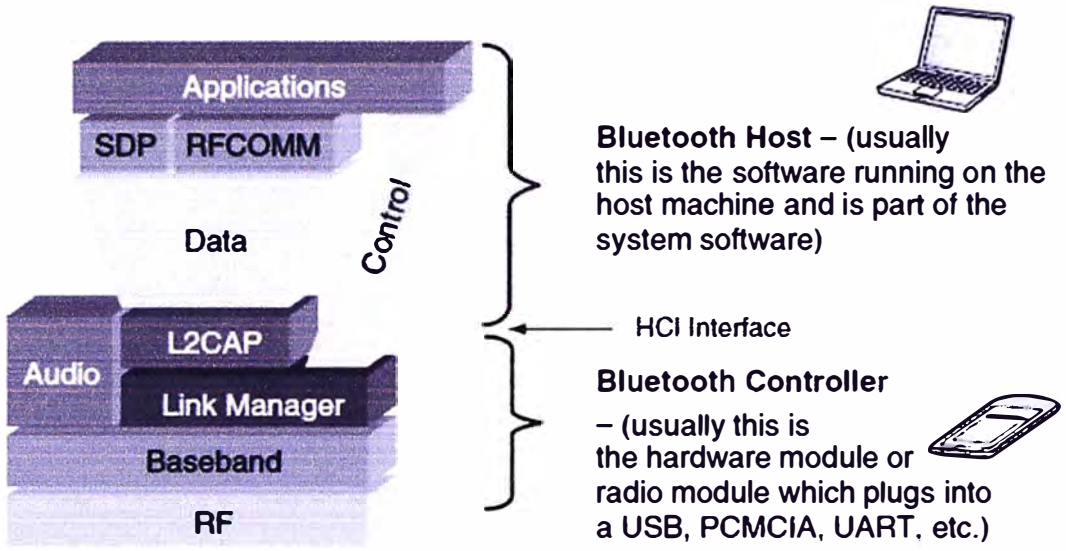

Fig. 2.2 Bluetooth host y Bluetooth controller (1)

El Bluetooth host también es conocido como la capa superior de la pila y usualmente está implementada en software. Está generalmente integrado con el software del sistema o el sistema operativo del host. Los perfiles Bluetooth están construidos sobre los protocolos. Ellos están generalmente en software y corren en el hardware del dispositivo host. Por ejemplo, una laptop sería el dispositivo host. El Bluetooth host estaría integrado con el sistema operativo de la laptop.

El Bluetooth controller usualmente es un módulo hardware como una tarjeta de PC que conecta un dispositivo destino. Muchos dispositivos tienen el Bluetooth controller construido dentro del dispositivo. El Bluetooth controller interactúa con el sistema del host vía un mecanismo estándar de entrada/salida (I/O) como UART y USB.

2.5.1 Protocolos Bluetooth

La Figura 2.3 muestra un diagrama de bloques de la pila de protocolos Bluetooth. Varios protocolos están definidos en la especificación Bluetooth, pero la Figura 2.3 muestra los comunes. Las cajas sombreadas representan los protocolos dirigidos por las

APIs de Java para la tecnología inalámbrica Bluetooth. La pila de protocolos está compuesta de protocolos específicos a Bluetooth, como Service Discovery Protocol (SDP), y otros protocolos adoptados, como el Object Exchange protocol (OBEX).

a) La capa **Bluetooth radio** es la capa más baja definida en la especificación Bluetooth. Ésta define los requerimientos del dispositivo transceptor Bluetooth operando en 2.4 GHz en la banda ISM.

b) La capa **baseband and link control** habilita el enlace RF físico entre las unidades Bluetooth haciendo una conexión. La **baseband** maneja el canal de procesamiento y sincronización, y el **link control** maneja el canal de control de acceso. Hay dos tipos de enlace físico: síncrono orientado a conexión (SCO) y asíncrono no orientado a conexión (ACL). Un enlace ACL lleva paquetes de datos, mientras que un enlace SCO soporta tráfico de audio en tiempo real.

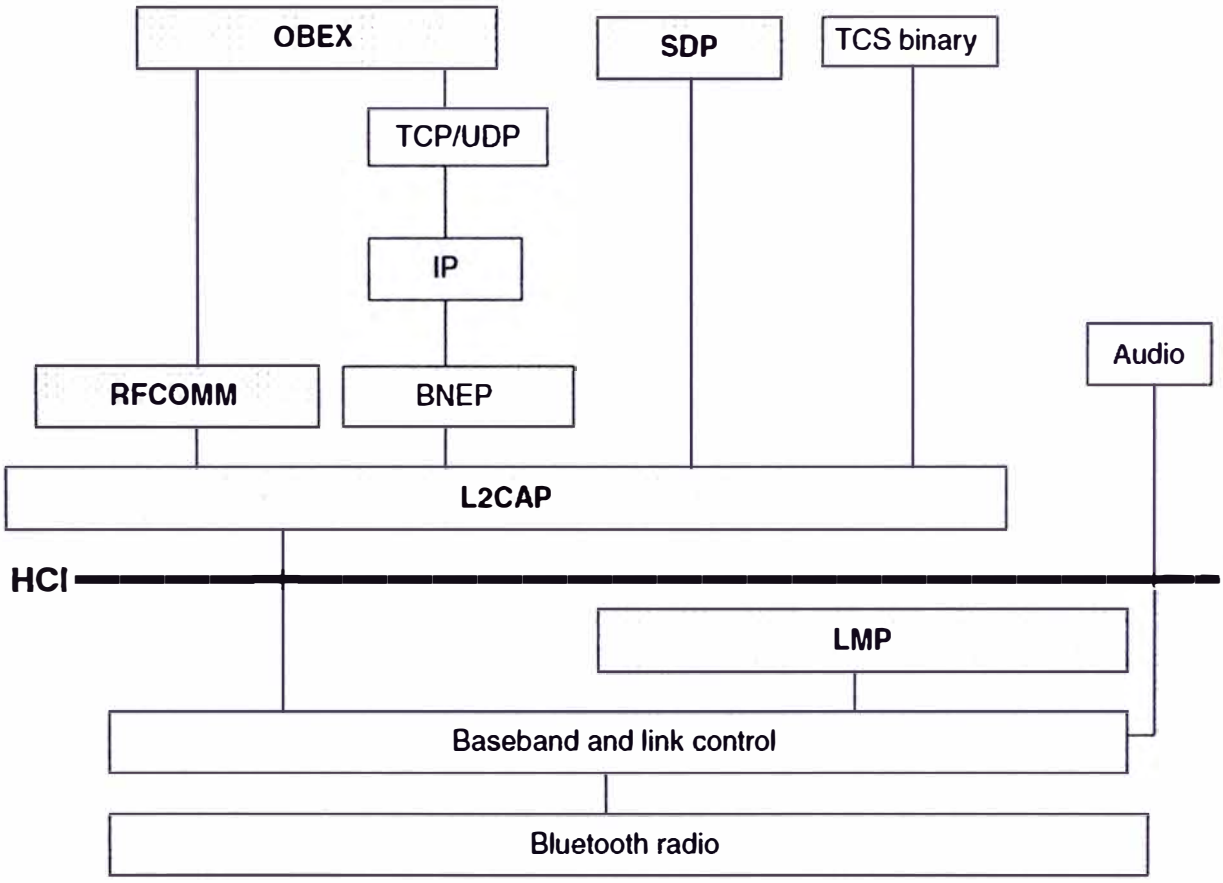

Fig. 2.3 Pila de Protocolos Bluetooth (1)

c) **Audio** no es realmente una capa de la pila de protocolos, pero esto es mostrado aquí porque el audio es tratado únicamente en la comunicación Bluetooth. Los datos de audio son dirigidos directamente hacia y desde la capa **baseband** sobre un enlace SCO. Si un canal de datos es usado (aplicaciones VoIP), los datos de audio serán transmitidos sobre un enlace ACL.

d) El **LMP** (Link Manager Protocol) es responsable de la configuración de enlace entre dispositivos Bluetooth, controlando y negociando el tamaño de los paquetes de banda base. El LMP gestiona los aspectos de seguridad, como autenticación y encriptación.
e) El **HCI** (Host Controller Interface) es una interfaz estándar para acceder a las capacidades de banda base Bluetooth, el estado del hardware, y el control de registros.
f) El **L2CAP** (Logical Link Control and Adaptation Protocol) protege los protocolos de capas superiores de los pormenores de los protocolos de capas inferiores. Ésta multiplexa entre varias conexiones lógicas hechas por las capas superiores.
g) El **SDP** (Service Discovery Protocol) provee un medio para que las aplicaciones consulten servicios y las características de los servicios. A diferencia de una conexión LAN, en la que se conecta a una red y luego se encuentra dispositivos, en un entorno Bluetooth se encuentra los dispositivos antes de encontrar los servicios. El conjunto de servicios disponibles cambia en un entorno cuando los dispositivos están en movimiento. Por eso SDP es bastante diferente del descubrimiento de servicios en tradicionales entornos de red. SDP está construido encima de L2CAP.
h) El puerto serie es uno de los más comunes en informática y dispositivos de comunicación. El protocolo **RFCOMM** provee emulación de puerto serial sobre L2CAP. RFCOMM suministra capacidades de transporte para los servicios de nivel superior que utilizan una interfaz serial como mecanismo de transporte. RFCOMM provee múltiples conexiones concurrentes a un dispositivo y suministra conexiones a múltiples dispositivos.
i) Los dispositivos con Bluetooth activado tendrán la habilidad para formar redes e intercambiar información, un formato de paquete común debe ser definido para encapsular protocolos de red de la capa 3. El **BNEP** (Bluetooth Network Encapsulation Protocol) es un protocolo opcional que encapsula paquetes de varios protocolos de red. Los paquetes son transportados directamente sobre L2CAP.
j) **TCS binary** (Telephony Control Protocol Specification Binary) define la señalización para el control de llamadas, establecimiento de llamadas de voz y datos entre dispositivos Bluetooth. Éste está construido sobre L2CAP.
k) **OBEX** es un protocolo adoptado que permite enviar y recibir objetos.

2.5.2 Perfiles Bluetooth

Los perfiles Bluetooth han sido definidos por el Bluetooth SIG. Un perfil Bluetooth define formas estándar de utilizar protocolos seleccionados y características de protocolos que activan un modelo de uso particular. En otras palabras, éste define cómo diferentes partes de la especificación Bluetooth pueden ser usadas para un caso de uso particular. Un perfil puede ser descrito como una tajada vertical a través de la pila de

protocolos. Dos perfiles pueden usar un conjunto diferente de capas de protocolo y un conjunto diferente de características dentro de la misma capa de protocolo.

Un dispositivo Bluetooth puede soportar uno o más perfiles. Los cuatro perfiles básicos son el Generic Access Profile (GAP), el Serial Port Profile (SPP), el Service Discovery Application Profile (SDAP), y el Generic Object Exchange Profile (GOEP).

a) El **GAP** es la base de todos los otros perfiles. GAP define los procedimientos genéricos relacionados a establecer conexiones entre dos dispositivos, incluyendo el descubrimiento de dispositivos Bluetooth, configuración y gestión del enlace, y procedimientos relacionados al uso de diferentes niveles de seguridad.

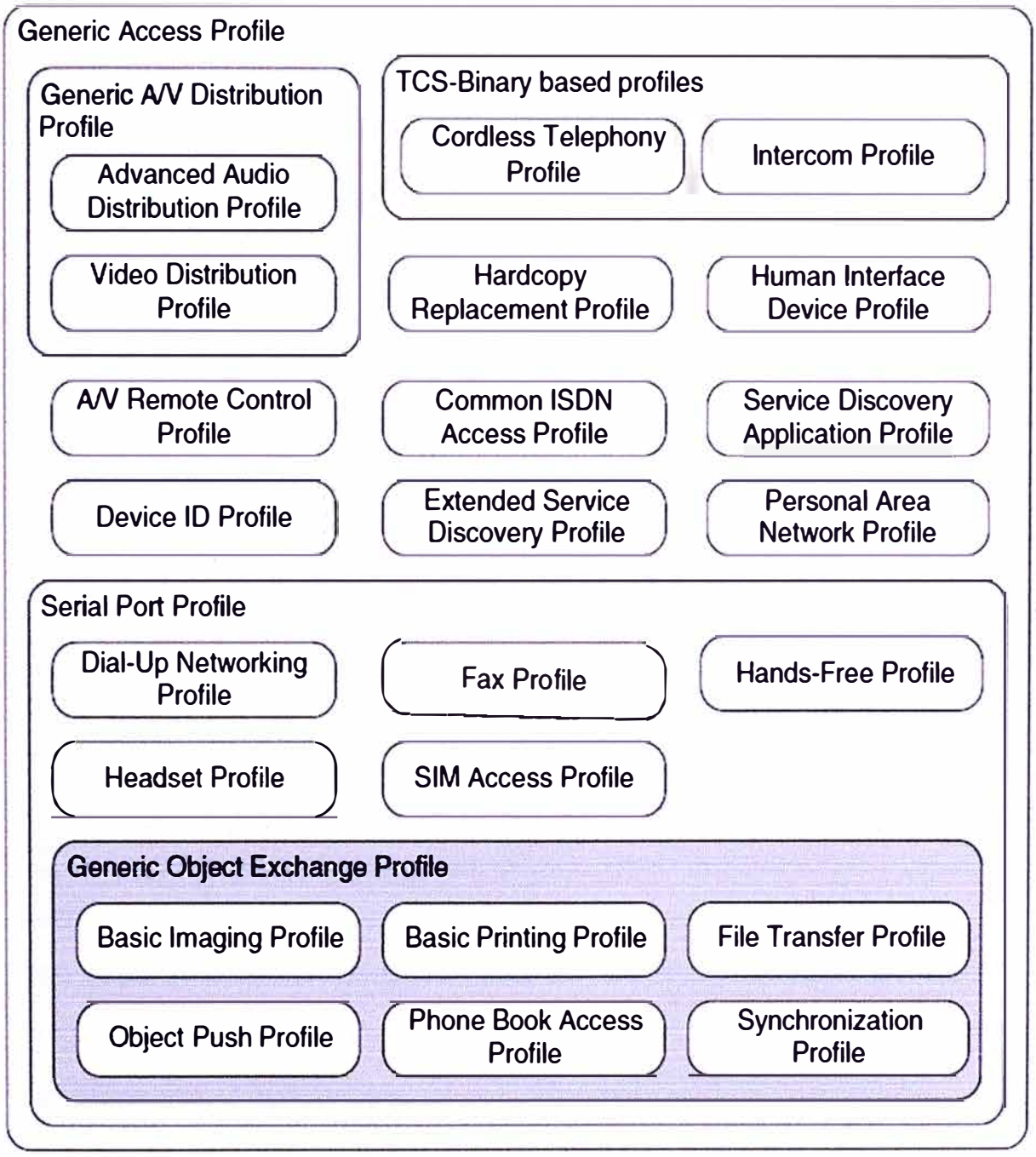

Fig. 2.4 Jerarquía de los Perfiles Bluetooth (1)

b) El **SDAP** describe las operaciones fundamentales necesarias para el descubrimiento de servicios. Este perfil define los protocolos y procedimientos que son usados por aplicaciones para localizar servicios en otros dispositivos con Bluetooth activado.
c) El **SPP** define los requerimientos necesarios para emular conexiones de cable serial usando RFCOMM entre dos dispositivos. SPP mapea el protocolo RFCOMM directamente y habilita aplicaciones antiguas usando Bluetooth como reemplazo de cable.
d) El **GOEP** es un perfil abstracto. Es usado por los perfiles que quieren emplear la funcionalidad del protocolo OBEX.

La Figura 2.4 muestra la relación entre varios perfiles Bluetooth. Los perfiles Bluetooth son jerárquicos. Por ejemplo, el File Transfer Profile está construido encima de GOEP, que depende de SPP, el cual está construido sobre GAP.

2.5.3 Calificación Bluetooth

Es el proceso de certificación requerido para cualquier producto que use Bluetooth. El proceso de calificación asegura que los productos cumplan con la especificación Bluetooth. Sólo productos calificados están autorizados para usar la licencia libre de patentes requerida para implementar la tecnología inalámbrica Bluetooth, la marca Bluetooth y el logo Bluetooth. Existen las siguientes pruebas de calificación Bluetooth:
1) Conformidad con la especificación de núcleo.
2) Pruebas de interoperabilidad para asegurar que los dispositivos trabajan con otros en el nivel de perfil.

2.6 Funcionamiento Bluetooth

2.6.1 Salto de frecuencia

Todos los dispositivos Bluetooth operan en la banda de frecuencia de 2.4 GHz. Esto significa que ellos usan la misma frecuencia de radio como microondas, 802.11 y algunos teléfonos inalámbricos (la clase que está anexado a las líneas de tierra, no teléfonos móviles). Lo que hace a Bluetooth diferente de las otras tecnologías es que éste divide la banda de 2.4 GHz en 79 canales y emplea técnicas de salto de canal, así que los dispositivos Bluetooth siempre están cambiando la frecuencia de recepción y transmisión. De esta forma, la interferencia en un canal no es mucho problema, pues no debe haber interferencia después de saltar a un nuevo canal.

Para comparación, considerar la forma como opera WiFi (802.11b y 802.11g). La banda de 2.4 GHz es dividida en 14 canales que tienen 5 MHz de ancho. Cuando una red inalámbrica es configurada el administrador de red escoge uno de estos canales y todos los dispositivos 802.11 en la red inalámbrica siempre transmitirán en la frecuencia de radio para este canal, con la posibilidad de colisión entre redes inalámbricas.

Bluetooth también divide la banda de 2.4 GHz en canales, pero tiene 79 canales en

vez de 14, y los canales son más estrechos (1 MHz de ancho en vez de 5 MHz). La gran diferencia es que los dispositivos Bluetooth nunca se quedan en el mismo canal. Una comunicación activa entre dispositivos Bluetooth cambia de canal cada 625 μs (1600 veces por segundo) en orden aleatorio. Dos dispositivos Bluetooth que se están comunicando deben saltar canales juntos de modo que ellos siempre están transmitiendo y recibiendo en la misma frecuencia. Esto es ilustrado en la Figura 2.5.

Presuntamente, todos estos saltos hacen a Bluetooth más robusto frente a interferencia de fuentes cercanas de malas ondas de radio, y permite a varias redes Bluetooth coexistir en el mismo lugar.

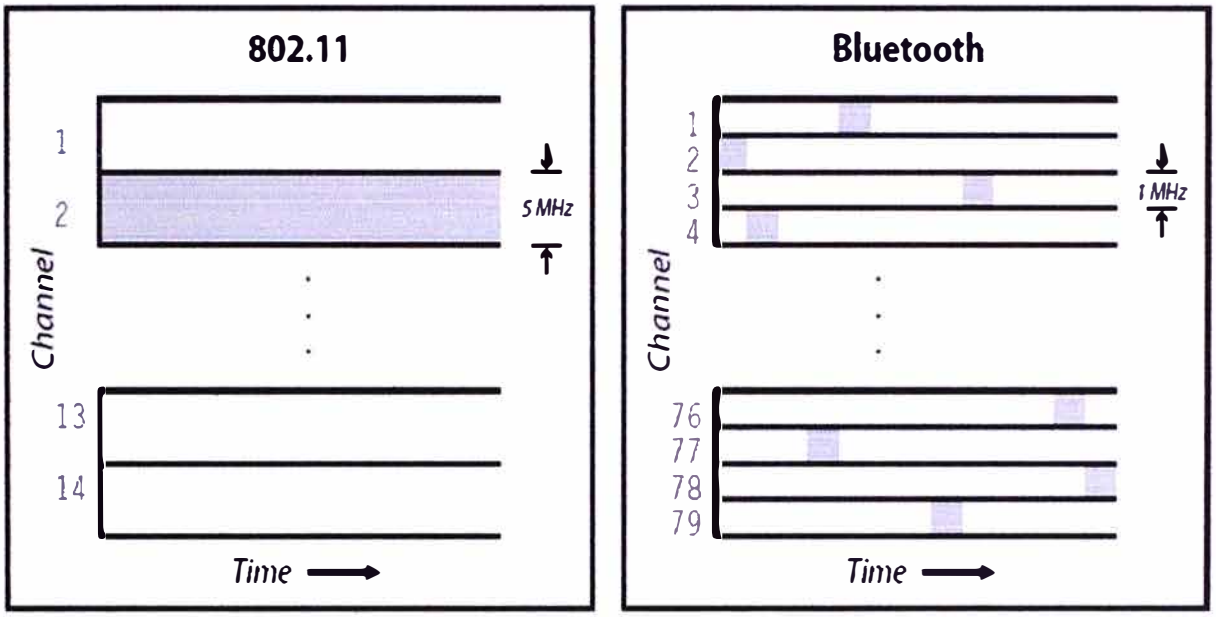

Fig. 2.5 Uso de la Banda de 2.4GHz por 802.11b y Bluetooth (4)

2.6.2 Definición de paquete

La información que se intercambia entre dos unidades Bluetooth se realiza mediante un conjunto de slots que forman un paquete de datos. Cada paquete comienza con un código de acceso de 72 bits, que se deriva de la identidad maestra, seguido de un paquete de datos de cabecera de 54 bits. Éste contiene importante información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits para la retransmisión automática de la pregunta, y chequeo de errores de campos de cabeza. Finalmente, el paquete que contiene la información, que puede seguir al de cabeza, tiene una longitud de 0 a 2745 bits. En cualquier caso, cada paquete que se intercambia en el canal está precedido por el código de acceso. La Figura 2.6 muestra un paquete Bluetooth.

Los receptores en la red comparan las señales que reciben con el código de acceso, si éstas no coinciden, el paquete recibido no es considerado como válido en el canal y el

resto de su contenido es ignorado. En la especificación Bluetooth se han definido dos tipos de enlace:

1) Enlace síncrono orientado a conexión (SCO).

2) Enlace asíncrono no orientado a conexión (ACL).

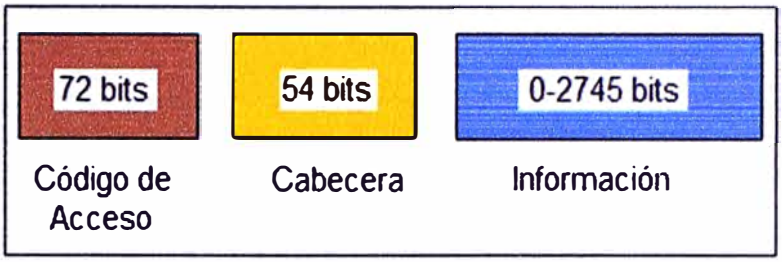

Fig. 2.6 Paquete Bluetooth (7)

Un conjunto de paquetes se han definido para cada tipo de enlace físico:

Para los enlaces SCO, existen tres tipos de slot simple, cada uno con una portadora a una velocidad de 64 kbit/s. La transmisión de voz se realiza sin ningún mecanismo de protección, pero si el intervalo de las señales en el enlace SCO disminuye, se puede seleccionar una velocidad de corrección de envío de 1/3 o 2/3.

Para los paquetes ACL, se han definido el slot-1, slot-3, slot-5. Cualquiera de los datos pueden ser enviados protegidos o sin proteger con una velocidad de corrección de 2/3. La máxima velocidad de envío es de 721 kbit/s en una dirección y 57.6 kbit/s en la otra.

2.6.3 Piconet y scatternet

a) Piconet

Si un equipo se encuentra dentro del radio de cobertura de otro, éstos pueden establecer conexión entre ellos. En principio sólo son necesarias un par de unidades con las mismas características de hardware para establecer un enlace. Dos o más unidades Bluetooth que comparten un mismo canal forman una piconet. Una piconet tiene un maestro y hasta 7 esclavos. Los esclavos sólo pueden comunicarse con el maestro. En la Figura 2.7 se muestra dispositivos Bluetooth en una piconet.

Para regular el tráfico en el canal, una de las unidades participantes se convertirá en maestra, pero por definición, la unidad que establece la piconet asume éste papel y todos los demás serán esclavos. Los participantes podrían intercambiar los papeles si una unidad esclava quisiera asumir el papel de maestra. Sin embargo sólo puede haber un maestro en la piconet al mismo tiempo.

Cada unidad de la piconet utiliza su identidad maestra y reloj nativo para seguir en el canal de salto. Cuando se establece la conexión, se añade un ajuste de reloj a la propia frecuencia de reloj nativa de la unidad esclava para poder sincronizarse con el reloj nativo

del maestro. El reloj nativo mantiene siempre constante su frecuencia, sin embargo los ajustes producidos por las unidades esclavas para sincronizarse con el maestro, sólo son válidos mientras dura la conexión.

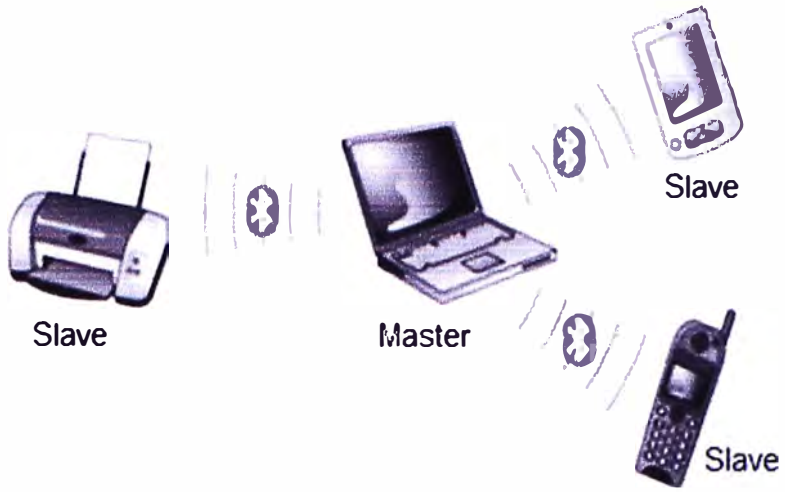

Fig. 2.7 Piconet (2)

Las unidades maestras controlan en tráfico del canal, por lo que estas tienen la capacidad para reservar slots en los enlaces SCO. Para los enlaces ACL, se utiliza un esquema de sondeo. A una esclava sólo se le permite enviar un slot a un maestro cuando ésta se ha dirigido por su dirección MAC (medio de control de acceso) en el procedimiento de slot maestro-esclavo. Éste tipo de slot implica un sondeo por parte del esclavo, por lo que, en un tráfico normal de paquetes, este es enviado a una urna del esclavo automáticamente. Si la información del esclavo no está disponible, el maestro puede utilizar un paquete de sondeo para sondear al esclavo explícitamente. Los paquetes de sondeo consisten únicamente en uno de acceso y otro de cabecera. Éste esquema de sondeo central elimina las colisiones entre las transmisiones de los esclavos.

Para establecer la piconet, la unidad maestra debe conocer la identidad del resto de unidades que están en modo standby en su radio de cobertura. El maestro o aquella unidad que inicia la piconet transmite el código de acceso continuamente en periodos de 10 ms, que son recibidas por el resto de unidades que se encuentran en standby. El tren de 10 ms. de códigos de acceso de diferentes saltos de portadora, se transmite repetidamente hasta que el receptor responde o bien se excede el tiempo de respuesta.

Cuando una unidad emisora y una receptora seleccionan la misma portadora de salto, la receptora recibe el código de acceso y devuelve una confirmación de recibo de la señal, es entonces cuando la unidad emisora envía un paquete de datos que contiene su identidad y frecuencia de reloj actual. Después de que el receptor acepta éste paquete,

ajustará su reloj para seleccionar el canal de salto correcto determinado por emisor. De éste modo se establece una piconet en la que la unidad emisora actúa como maestra y la receptora como esclava. Después de haber recibido los paquetes de datos con los códigos de acceso, la unidad maestra debe esperar un procedimiento de requerimiento por parte de las esclavas, diferente al proceso de activación, para poder seleccionar una unidad específica con la que quiere comunicarse.

b) Scatternet

Los equipos que comparten un mismo canal sólo pueden utilizar una parte de su capacidad. Aunque los canales tienen un ancho de banda de un 1Mhz, cuantos más usuarios se incorporan a la piconet, disminuye la capacidad hasta unos 10 kbit/s más o menos.

Las unidades que se encuentran en el mismo radio de cobertura pueden establecer potencialmente comunicaciones entre ellas. Sin embargo, sólo aquellas unidades que realmente quieran intercambiar información comparten un mismo canal creando la piconet. Éste hecho permite que se creen varias piconets en áreas de cobertura superpuestas.

A un grupo de piconets se le llama scatternet (Figura 2.8). 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 multiplexión y rechazo de salto de canales. Debido a que individualmente cada piconet tiene un salto de frecuencia diferente, diferentes piconets pueden usar simultáneamente diferentes canales de salto.

Fig. 2.8 Scatternet (2)

En un conjunto de varias piconets, éstas seleccionan diferentes saltos de frecuencia y están controladas por diferentes maestros, por lo que si un mismo canal de salto es

compartido temporalmente por piconets independientes, los paquetes de datos podrán ser distinguidos por el código de acceso que les precede, que es único en cada piconet.

De la misma manera que una esclava puede cambiar de una piconet a otra, una maestra también lo puede hacer, con la diferencia de que el tráfico de la piconet se suspende hasta la vuelta de la unidad maestra. La maestra que entra en una nueva piconet, en principio, lo hace como esclava, a no ser que posteriormente ésta solicite actuar como maestra.

2.7 Seguridad en Bluetooth

a) Tipos de clave

La seguridad proporcionada por el núcleo Bluetooth está construida sobre el uso de mecanismos criptográficos de clave simétrica para autenticación, encriptación de enlace y generación de clave. Un número de diferentes tipos de claves son usados en conexión con estos mecanismos. En Bluetooth, un enlace es un canal de comunicación que es establecido entre dos dispositivos Bluetooth.

Para verificar que un enlace está establecido entre los dispositivos correctos, un procedimiento de autenticación entre dos dispositivos ha sido introducido. El mecanismo de autenticación en este procedimiento usa la llamada clave de enlace. Hay diferentes tipos de claves de enlace. Las claves de enlace no sólo son usadas para autenticación. Ellas también son usadas para derivación de la clave que controla la encriptación de los datos enviados por un enlace. A través de esta encriptación, la confidencialidad de los datos transmitidos está realizada. El correspondiente mecanismo de encriptación usa la clave de encriptación de enlace. Una clave de enlace es creada durante el emparejamiento de dos dispositivos.

Es importante distinguir dos estados importantes. En primer lugar, el estado en que un dispositivo quiere establecer una conexión con un dispositivo que no ha sido emparejado. En segundo lugar, el estado donde un dispositivo quiere comunicarse con un dispositivo emparejado. Un dispositivo puede perder la información de emparejamiento asociada con otro dispositivo. En esta situación, el dispositivo tiene el estado no emparejado.

b) Emparejamiento

El emparejamiento de dos dispositivos es el procedimiento por el cual dos dispositivos establecen un secreto compartido que ellos pueden usar cuando se reúnan otra vez. El emparejamiento requiere interacción del usuario, por ejemplo, el ingreso de una clave de paso (pass-key) o número de identificación personal (PIN).

El sistema Bluetooth permite claves de paso de 128 bits. Esta característica permite el uso de un esquema automático de alto nivel llamado acuerdo de clave (key agreement),

que puede ser una red o protocolo de seguridad en la capa de transporte (TLS). Ver la Figura 2.9.

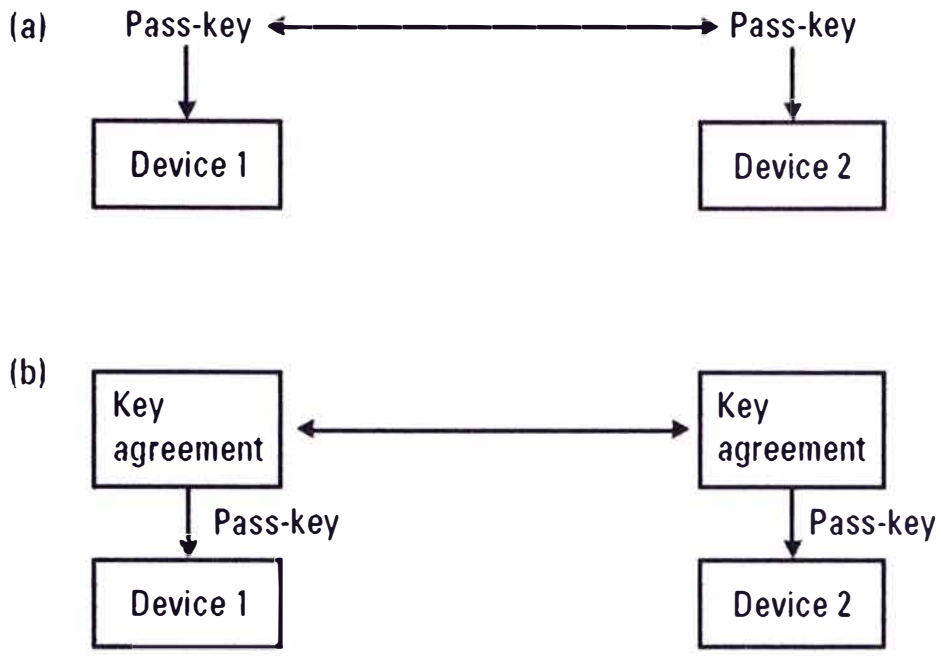

Fig. 2.9 Emparejamiento (5)

c) Autenticación

Un dispositivo Bluetooth en estado conectable acepta solicitudes de conexión de otros dispositivos. Esto significa que hay riesgo que un dispositivo conectable sea conectado y atacado por un dispositivo malicioso. Obvio, esto puede ser evitado si un dispositivo nunca entra en estado conectable.

Todas las conexiones Bluetooth no pueden ser establecidas. Existe la necesidad de identificar con seguridad el otro punto de comunicación, de esta forma las conexiones de dispositivos desconocidos pueden ser rechazados. La identificación del dispositivo es proporcionado a través del mecanismo de autenticación Bluetooth.

El proceso de autenticación es llamado esquema desafío-respuesta (challenge-response), donde el dispositivo verificador envía un desafío aleatorio al dispositivo demandante y espera un valor de respuesta válida de retorno. Si la autenticación mutua es necesitada el procedimiento debe ser repetido con los roles del verificador y demandante intercambiados.

d) Autorización

Autorización es un proceso de dar permiso para hacer o tener acceso a alguna cosa. Para Bluetooth esto significa decidir si un dispositivo remoto tiene derecho de acceso a un servicio en el host local y que privilegios gana por esto. Usualmente esto implica alguna forma de interacción con el usuario.

e) Modos de seguridad

El Generic Access Profile (GAP) define el procedimiento genérico relacionado al descubrimiento de dispositivos Bluetooth y a los aspectos de gestión de enlace para conectar dispositivos Bluetooth. El GAP también define los diferentes procedimientos de seguridad básicos de un dispositivo Bluetooth. Un dispositivo conectable puede operar en tres modos de seguridad.

Modo de seguridad 1: Es el modo inseguro en Bluetooth. Una unidad que ofrece su servicio a todos los dispositivos conectados opera en modo de seguridad 1. Esto implica que la unidad no demanda autenticación o encriptación en el establecimiento de la conexión. Por ejemplo, un punto de acceso que ofrece servicios de información a cualquiera es un posible escenario de uso para el modo de seguridad 1. El soporte de autenticación es mandatario y la unidad debe responder a cualquier desafío de autenticación. No obstante, la unidad nunca enviará un desafío de autenticación por sí mismo y la autenticación mutua nunca es ejecutada.

Modo de seguridad 2: Los procedimientos de seguridad no son iniciados hasta que una solicitud de canal o conexión haya sido recibida. Sólo cuando la aplicación o servicio lo requiera, se activarán los mecanismos de autenticación y/o encriptación.

Modo de seguridad 3: Las políticas de seguridad siempre están activadas. La unidad no será generalmente accesible. Todas las unidades deben ser autenticadas. Dos diferentes políticas de seguridad son posibles: siempre demanda autenticación o siempre demanda autenticación y encriptación.

CAPÍTULO III
JAVA 2 MICRO EDITION (J2ME)

3.1 Introducción

La empresa Sun Microsystems lanzó a mediados de los años 90 el lenguaje de programación Java que, aunque en un principio fue diseñado para generar aplicaciones que controlaran electrodomésticos como lavadoras, frigoríficos, etc, debido a su gran robustez e independencia de la plataforma donde se ejecutase el código, desde sus comienzos se utilizó para la creación de componentes interactivos integrados en páginas Web y programación de aplicaciones independientes. Estos componentes se denominaron applets.

Sun Microsystems, dispuesto a proporcionar las herramientas necesarias para cubrir las necesidades de todos los usuarios, creó distintas ediciones de Java. J2SE (Java Standard Edition) orientada al desarrollo de aplicaciones independientes de la plataforma, J2EE (Java Enterprise Edition) orientada al entorno empresarial y J2ME (Java Micro Edition) orientada a dispositivos con capacidades restringidas.

Java 2 Standard Edition (J2SE)

Esta edición de Java es la que en cierta forma recoge la iniciativa original del lenguaje Java. Tiene las siguientes características:

1) Inspirado inicialmente en C++, pero con componentes de alto nivel, como soporte nativo de strings y recolector de basura.

2) Código independiente de la plataforma, precompilado a bytecodes intermedio y ejecutado en el cliente por una JVM (Java Virtual Machine).

3) Modelo de seguridad tipo sandbox proporcionado por la JVM.

4) Abstracción del sistema operativo subyacente mediante un juego completo de APIs de programación.

Esta versión de Java contiene el conjunto básico de herramientas usadas para desarrollar Java Applets, así como las APIs orientadas a la programación de aplicaciones de usuario final: Interfaz gráfica de usuario, multimedia, redes de comunicación, etc.

Java 2 Enterprise Edition (J2EE)

Esta versión está orientada al entorno empresarial. El software empresarial tiene unas características propias marcadas: está pensado no para ser ejecutado en un equipo, sino para ejecutarse sobre una red de ordenadores de manera distribuida y remota mediante

EJBs (Enterprise Java Beans). De hecho, el sistema se monta sobre varias unidades o aplicaciones. En muchos casos, además, el software empresarial requiere que se sea capaz de integrar datos provenientes de entornos heterogéneos. Esta edición está orientada especialmente al desarrollo de servicios web, servicios de nombres, persistencia de objetos, XML, autenticación, APIs para la gestión de transacciones, etc. El cometido de esta especificación es ampliar la J2SE para dar soporte a los requisitos de las aplicaciones de empresa.

Java 2 Micro Edition (J2ME)

Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica. La Figura 3.1 muestra la arquitectura de la plataforma Java 2.

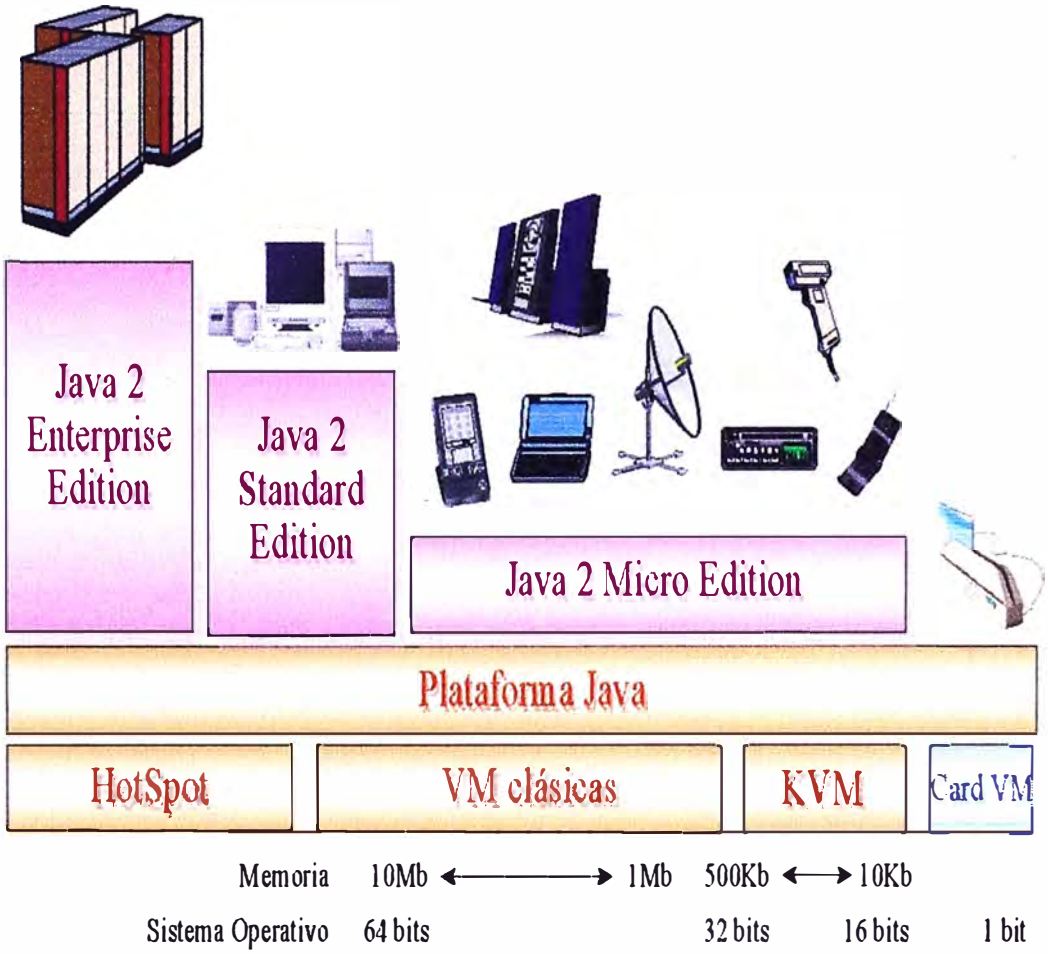

Fig. 3.1 Arquitectura de la plataforma Java 2 de Sun (3)

Java es un conjunto de tecnologías que abarca a todos los ámbitos de la computación con dos elementos en común:

1) El código fuente en lenguaje Java es compilado a código intermedio interpretado por una Java Virtual Machine (JVM), por lo que el código ya compilado es independiente de la plataforma.

2) Todas las tecnologías comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes java.lang y java.io.

Sun separó los productos por razones de eficiencia, J2EE es un superconjunto de J2SE pues contiene toda la funcionalidad de éste y más características, así como J2ME es un subconjunto de J2SE (excepto por el paquete javax.microedition). La Figura 3.2 muestra esta relación. En realidad cada una de las ediciones está enfocada a ámbitos de aplicación muy distintos.

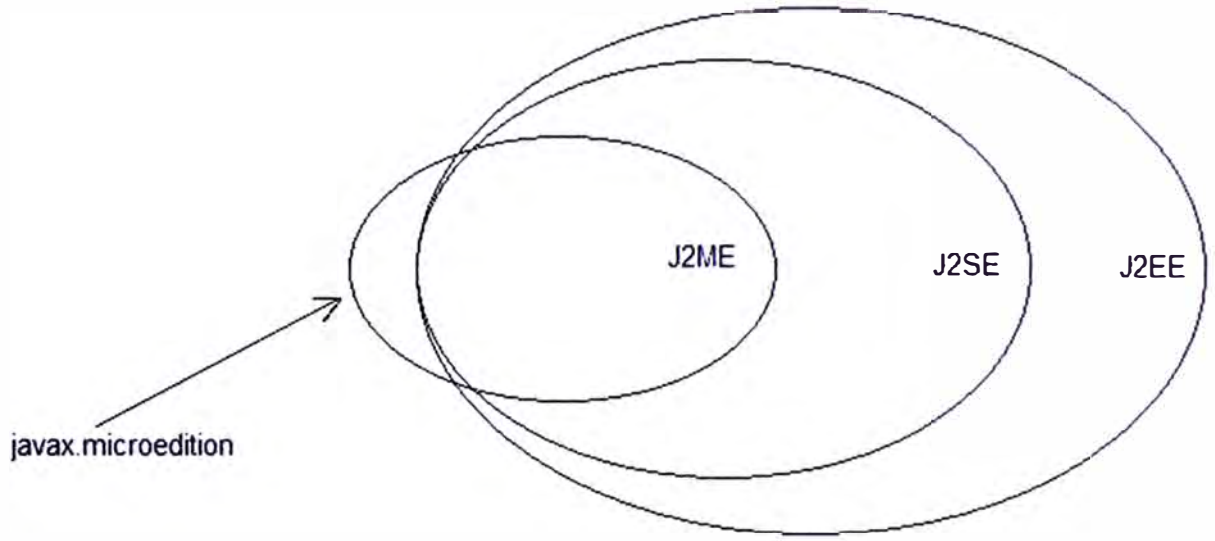

Fig. 3.2 Relación entre las APIs de la plataforma Java (3)

3.2 Nociones básicas de J2ME

Los componentes que forman parte de esta tecnología son:

1) Una serie de máquinas virtuales Java con diferentes requisitos, cada una para diferentes tipos de pequeños dispositivos.

2) Configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas.

3) Perfiles, que son unas bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

Un entorno de ejecución determinado de J2ME se compone de una selección de:

1) Máquina virtual.

2) Configuración.

3) Perfil.

4) Paquetes Opcionales.

La arquitectura de un entorno de ejecución se puede ver en la Figura 3.3.

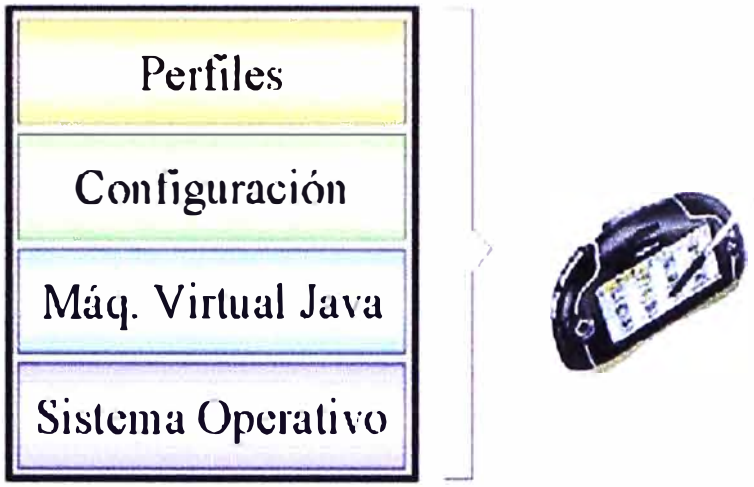

Fig. 3.3 Entorno de ejecución (3)

3.2.1 Máquinas virtuales J2ME

Una máquina virtual de Java (JVM) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo subyacente. Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a memoria ocupada y requerimientos computacionales. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

La VM (Virtual Machine) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM.

a) KVM

Se corresponde con la Máquina Virtual más pequeña desarrollada por Sun. Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código, y fue diseñada para ser:

1) Pequeña, con una carga de memoria entre los 40Kb y los 80 Kb, dependiendo de la

plataforma y las opciones de compilación.

2) Alta portabilidad.

3) Modulable.

4) Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada.

La baja ocupación de memoria hace que posea algunas limitaciones con respecto a la clásica Java Virtual Machine (JVM):

1) No hay soporte para tipos en coma flotante. No existen por tanto los tipos double ni float.

2) No existe soporte para JNI (Java Native Interface) debido a los recursos limitados de memoria.

3) No existen cargadores de clases (class loaders) definidos por el usuario. Sólo existen los predefinidos.

4) No se permiten los grupos de hilos o hilos daemon. Cuando queramos utilizar grupos de hilos utilizaremos los objetos Colección para almacenar cada hilo en el ámbito de la aplicación.

5) No existe la finalización de instancias de clases. No existe el método Object.finalize().

6) No hay referencias débiles.

7) Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.

8) Reflexión.

Un objeto que está siendo apuntado mediante una referencia débil es un candidato para la recolección de basura. Estas referencias están permitidas en J2SE, pero no en J2ME.

La reflexión es el mecanismo por el cual los objetos pueden obtener información de otros objetos en tiempo de ejecución como, por ejemplo, los archivos de clases cargados o sus campos y métodos.

b) CVM

La CVM (Compact Virtual Machine) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en tomo a 2MB o más de memoria RAM. Las características que presenta esta Máquina Virtual son:

1) Sistema de memoria avanzado.

2) Tiempo de espera bajo para el recolector de basura.

3) Separación completa de la VM del sistema de memoria.

4) Recolector de basura modularizado.

5) Portabilidad.

6) Rápida sincronización.

7) Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).

8) Soporte nativo de hilos.

9) Baja ocupación en memoria de las clases.

10) Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.

11) Conversión de hilos Java a hilos nativos.

12) Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

3.2.2 Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Éstas APIs describen las características básicas, comunes a todos los dispositivos:

1) Características soportadas del lenguaje de programación Java.

2) Características soportadas por la Máquina Virtual Java.

3) Bibliotecas básicas de Java y APIs soportadas.

a) Configuración de dispositivos con conexión, CDC (Connected Device Configuration)

La CDC está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Esta máquina virtual es la CVM. La CDC está enfocada a dispositivos con las siguientes capacidades:

1) Procesador de 32 bits.

2) Disponer de 2MB o más de memoria total, incluyendo memoria RAM y ROM.

3) Poseer la funcionalidad completa de la Máquina Virtual Java2.

4) Conectividad a algún tipo de red.

Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones http y basadas en datagramas.

La Tabla Nº 3.1 muestra las librerías incluidas en la CDC.

TABLA Nº 3.1 Librerías de configuración CDC (3)

Nombre de Paquete CDC	Descripción
java.io	Clases e interfaces estándar de E/S.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfaces de reflexión.
java.math	Paquete de matemáticas.
java.net	Clases e interfaces de red.
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de seguridad.
java.text	Paquete de texto.
java.util	Clases de utilidades estándar.
java.util.jar	Clases y utilidades para archivos JAR.
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos.
javax.microedition.io	Clases e interfaces para conexión genérica CDC.

b) Configuración de dispositivos limitados con conexión, CLDC (Connected Limited Device Configuration)

La CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de estos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc. Algunas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

1) Disponer entre 160 KB y 512 KB de memoria total disponible. Como mínimo se debe disponer de 128 KB de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 KB de memoria volátil para la Máquina Virtual en tiempo de ejecución.

2) Procesador de 16 o 32 bits con al menos 25 MHz de velocidad.

3) Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, normalmente baterías.

4) Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

La Tabla Nº 3.2 muestra las librerías incluidas en la CLDC. La CLDC aporta las siguientes funcionalidades a los dispositivos:

1) Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).

2) Un subconjunto de las bibliotecas Java del núcleo.

3) Soporte para E/S básica.

4) Soporte para acceso a redes.

5) Seguridad.

TABLA Nº 3.2 Librerías de configuración CLDC (3)

Nombre de Paquete CLDC	Descripción
java.io	Clases y paquetes estándar de E/S. Subconjunto de J2SE.
java.lang	Clases e interfaces de la Máquina Virtual. Subconjunto de J2SE.
java.util	Clases, interfaces y utilidades estándar. Subconjunto de J2SE.
java.microedition.io	Clases e interfaces de conexión genérica CLDC.

3.2.3 Perfiles

El perfil es el que define las APIs que controlan el ciclo de vida de la aplicación, interfaz de usuario, etc. Un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil.

El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Un perfil siempre se construye sobre una configuración determinada. De este modo, un perfil es un conjunto de APIs que dotan a una configuración de funcionalidad específica.

La configuración CDC tiene los siguientes perfiles:

1) Foundation Profile.

2) Pesonal Profile.

3) RMI Profile.

Y para la configuración CLDC se tiene:

1) PDA Profile.

2) Mobile Information Device Profile (MIDP).

Un perfil puede ser construido sobre cualquier otro. Sin embargo, una plataforma J2ME sólo puede contener una configuración. En la Figura 3.4 se puede ver el esquema del entorno de ejecución.

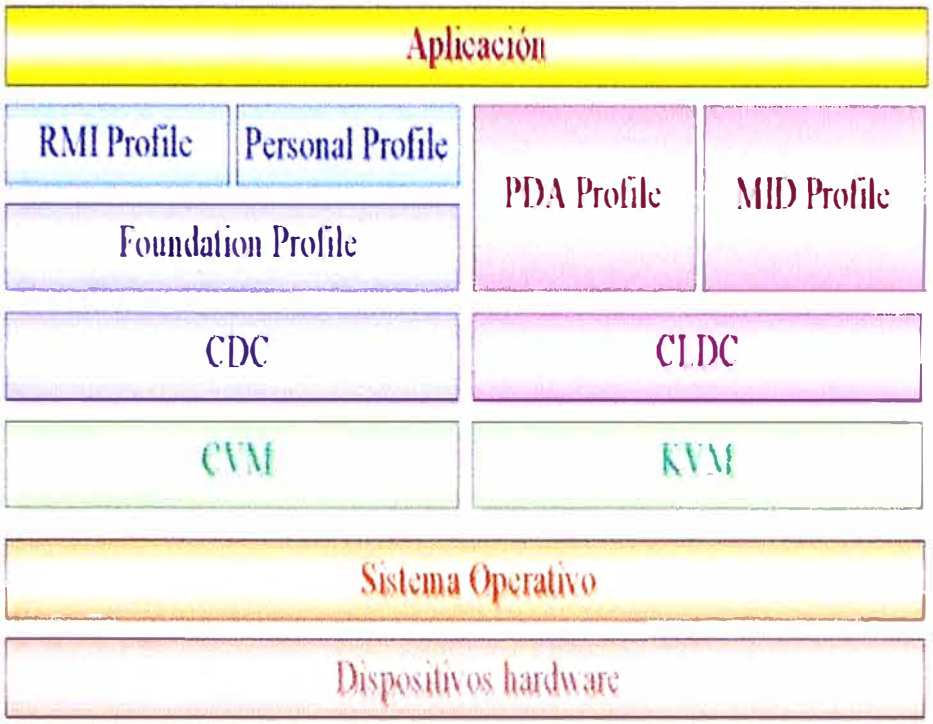

Fig. 3.4 Arquitectura del entorno de ejecución de J2ME (3)

Foundation Profile

Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" Abstract Windows Toolkit (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiere una GUI, entonces sería necesario un perfil adicional. Los paquetes que forman parte del Foundation Profile se muestran en la Tabla Nº 3.3.

TABLA Nº 3.3 Librerías del Foundation Profile (3)

Paquete del Foundation Profile	Descripción
java.lang	Soporte del lenguaje Java.
java.util	Añade soporte completo para zip.
java.net	Incluye sockets TCP/IP y conexiones HTTP.
java.io	Clases Reader y Writer de J2SE.
java.text	Incluye soporte para internacionalización.
java.security	Incluye códigos y certificados.

Personal Profile

Es un subconjunto de la plataforma J2SE v1.3, y proporciona un entorno con un

completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere una implementación del Foundation Profile. La Tabla Nº 3.4 muestra los paquetes que conforman el Personal Profile v1.0.

TABLA Nº 3.4 Librerías del Personal Profile (3)

Paquete del Personal Profile	Descripción
java.applet	Clases necesitadas para crear applets.
java.awt	Clases para crear GUIscon AWT.
java.awt.datatransfer	Clases e interfaces para transmitir datos entre aplicaciones.
java.awt.event	Clases e interfaces para manejar eventos AWT.
java.awt.font	Clases e interfaces para la manipulación de fuentes.
java.awt.im	Clases e interfaces para definir métodos editores de entrada.
java.awt.im.spi	Interfaces que añaden el desarrollo de métodos editores de entrada para cualquier entorno de ejecución Java.
java.awt.image	Clases para crear y modificar imágenes.
java.beans	Clases que soportan JavaBeans.
javax.microedition.xlet	Interfaces que usan el Personal Profile para la comunicación.

RMI Profile

Este perfil requiere una implementación del Foundation Profile. El perfil RMI soporta un subconjunto de las APIs J2SE v1.3 RMI.

PDA Profile

Está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixeles (al menos 200x100 pixeles) con un factor 2:1.

Mobile Information Device Profile (MIDP)

Este perfil está construido sobre la configuración CLDC. Este perfil está orientado para dispositivos con las siguientes características:

1) Reducida capacidad computacional y de memoria.
2) Conectividad limitada (en torno a 9600 bps).
3) Capacidad gráfica muy reducida (mínimo un display de 96x54 pixeles monocromo).
4) Entrada de datos alfanumérica reducida.
5) 128 KB de memoria no volátil para componentes MIDP.
6) 8 KB de memoria no volátil para datos persistentes de aplicaciones.

7) 32 KB de memoria volátil en tiempo de ejecución para la pila Java.

Los tipos de dispositivos que se adaptan a estas características son: teléfonos móviles, buscapersonas (pagers) o PDAs de gama baja con conectividad. El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las APIs relacionadas con:

1) La aplicación (semántica y control de la aplicación MIDP).
2) Interfaz de usuario.
3) Almacenamiento persistente.
4) Trabajo en red.
5) Temporizadores.

En la Tabla Nº 3.5 se puede ver cuáles son los paquetes que están incluidos en el perfil MIDP.

TABLA Nº 3.5 Librerías del perfil MIDP (3)

Paquetes del MIDP	Descripción
javax.microedition.lcdui	Clases e interfaces para GUIs.
javax.microedition.rms	Record Management Storage. Soporte para el almacenamiento persistente del dispositivo.
javax.microedition.midlet	Clases de definición de la aplicación.
javax.microedition.io	Clases e interfaces de conexión genérica.
java.io	Clases e interfaces de E/S básica.
java.lang	Clases e interfaces de la Máquina Virtual.
java.util	Clases e interfaces de utilidades estándar.

3.3 MIDlet

Un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC. Están diseñados para ser ejecutados en dispositivos con poca capacidad gráfica, de cómputo y de memoria. En estos dispositivos no hay línea de comandos para ejecutar las aplicaciones, pero reside en él un software que es el encargado de ejecutar los MIDlets y gestionar los recursos que éstos ocupan.

El Gestor de Aplicaciones

El gestor de aplicaciones o AMS (Application Management System) es el software encargado de gestionar los MIDlets. Este software reside en el dispositivo y es el que permite ejecutar, pausar o destruir aplicaciones J2ME. El AMS realiza dos grandes funciones:

1) Gestiona el ciclo de vida de los MIDlets.
2) Es el encargado de controlar los estados por los que pasa el MIDlet mientras está en la memoria del dispositivo, es decir, en ejecución.

3.3.1 Ciclo de vida de un MIDlet

En la Figura 3.5 se puede ver las 5 fases del ciclo de vida de un MIDlet: descubrimiento, instalación, ejecución, actualización y borrado.

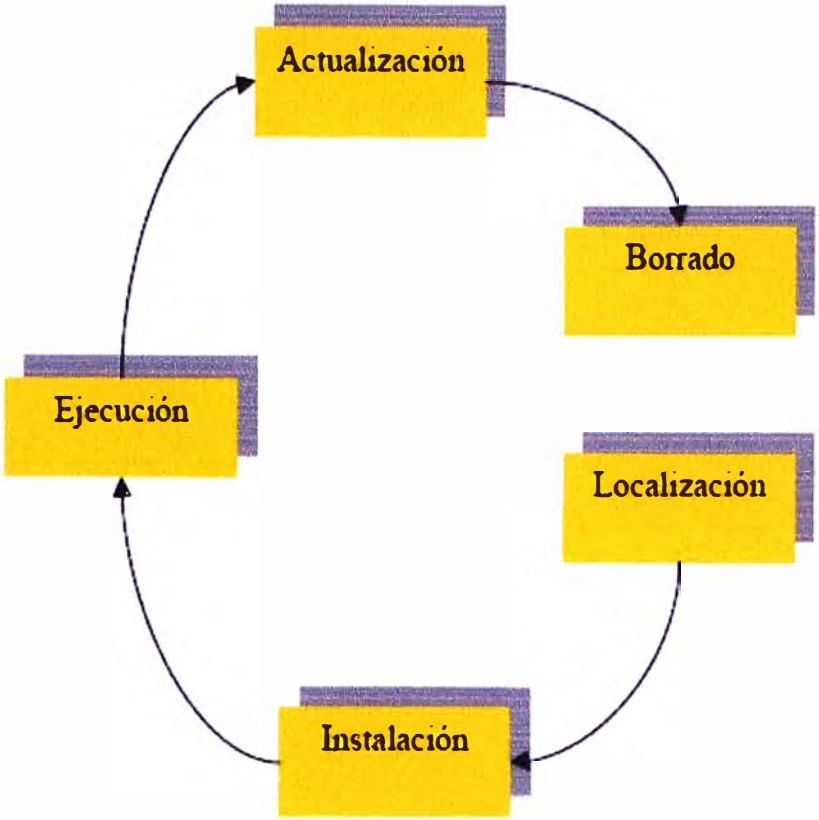

Fig. 3.5 Ciclo de vida de un MIDlet (3)

El AMS es el encargado de gestionar cada una de estas fases de la siguiente manera:
1) **Descubrimiento**: Esta fase es la etapa previa a la instalación del MIDlet y es donde se selecciona a través del AMS la aplicación a descargar. Por tanto, el gestor de aplicaciones tiene que proporcionar los mecanismos necesarios para realizar la elección del MIDlet a descargar. El AMS puede ser capaz de realizar la descarga de aplicaciones de diferentes maneras, dependiendo de las capacidades del dispositivo. Por ejemplo, la descarga se puede realizar mediante un cable conectado a un ordenador o mediante una conexión inalámbrica.
2) **Instalación**: Una vez descargado el MIDlet en el dispositivo, comienza el proceso de instalación. En esta fase el gestor de aplicaciones controla todo el proceso informando al usuario tanto de la evolución de la instalación como de si existiese algún problema durante ésta. Cuando un MIDlet está instalado en el dispositivo, todas sus clases, archivos y almacenamiento persistente están preparados y listos para su uso.
3) **Ejecución**: Mediante el gestor de aplicaciones vamos a ser capaces de iniciar la ejecución de los MIDlets. En esta fase, el AMS tiene la función de gestionar los estados

del MIDlet en función de los eventos que se produzcan durante esta ejecución.

4) **Actualización**: El AMS tiene que ser capaz de detectar después de una descarga si el MIDlet descargado es una actualización de un MIDlet ya presente en el dispositivo. Si es así, tiene que informar de ello y dar la oportunidad de decidir por actualizar o no.

5) **Borrado**: En esta fase el AMS es el encargado de borrar el MIDlet seleccionado del dispositivo. El AMS pedirá confirmación antes de proceder a su borrado e informará de cualquier circunstancia que se produzca.

El MIDlet puede permanecer en el dispositivo todo el tiempo que sea útil. Después de la fase de instalación, el MIDlet queda almacenado en una zona de memoria persistente del dispositivo. El usuario de este dispositivo es el encargado de decidir en qué momento quiere eliminar la aplicación y así se lo hará saber al AMS mediante alguna opción que éste suministre.

3.3.2 Estados de un MIDlet

El AMS es el encargado de controlar los estados del MIDlet durante su ejecución. El MIDlet es cargado en la memoria volátil del dispositivo y es aquí donde puede transitar entre tres estados diferentes: Activo, en pausa y destruido. La Figura 3.6 muestra el diagrama de estados de un MIDlet en ejecución.

1) Activo: El MIDlet está actualmente en ejecución.

2) Pausa: El MIDlet no está actualmente en ejecución. En este estado el MIDlet no debe usar ningún recurso compartido. Para volver a pasar a ejecución tiene que cambiar su estado a Activo.

3) Destruido: El MIDlet no está en ejecución ni puede transitar a otro estado. Además se liberan todos los recursos ocupados por el MIDlet.

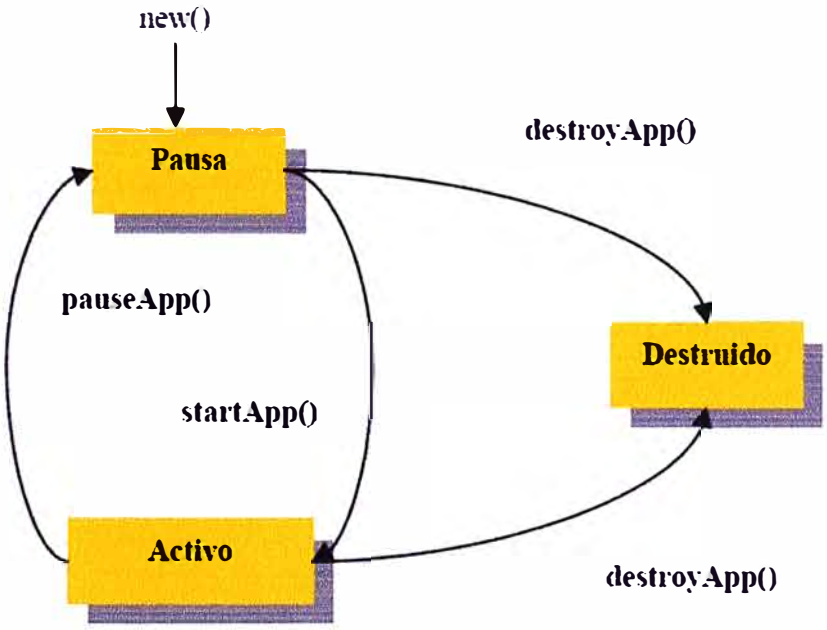

Fig. 3.6 Estados de un MIDlet (3)

Cuando el MIDlet comienza su ejecución, está en el estado "Activo". El gestor de aplicaciones debe ser capaz de cambiar el estado de la aplicación en función de los eventos externos al ámbito de ejecución de la aplicación que se van produciendo. El AMS interrumpiría la ejecución del MIDlet sin que se vea afectada la ejecución de éste y lo pasaría al estado "Pausa" para atender una llamada o un mensaje. Cuando se finaliza la ejecución del MIDlet, éste pasaría al estado de "Destruido" donde sería eliminado de la memoria volátil del dispositivo.

Una vez finalizada la ejecución del MIDlet se puede volver a invocarlo ya que éste permanece en la zona de memoria persistente del dispositivo.

Un MIDlet puede cambiar de estado mediante una llamada a los métodos MIDlet.startApp(), MIDlet.pauseApp() o MIDlet.destroyApp(). El gestor de aplicaciones cambia el estado de los MIDlets haciendo una llamada a cualquiera de los métodos anteriores. Un MIDlet también puede cambiar de estado por sí mismo.

Los estados por los que pasa un MIDlet durante una ejecución típica y las acciones que realizan tanto el AMS como el MIDlet, son los siguientes: En primer lugar se realiza la llamada al constructor del MIDlet pasando éste al estado de "Pausa" durante un corto periodo de tiempo. El AMS por su parte crea una nueva instancia del MIDlet. Cuando el dispositivo está preparado para ejecutar el MIDlet, el AMS invoca al método MIDlet.startApp() para entrar en el estado de "Activo". El MIDlet entonces, ocupa todos los recursos que necesita para su ejecución. Durante este estado, el MIDlet puede pasar al estado de "Pausa" por una acción del usuario o por el AMS. Tanto en el estado "Activo" como en el de "Pausa", el MIDlet puede pasar al estado "Destruido" realizando una llamada al método MIDlet.destroyApp(). Esto puede ocurrir porque el MIDlet haya finalizado su ejecución o porque una aplicación prioritaria necesite ser ejecutada en memoria en lugar del MIDlet. Una vez destruido el MIDlet, éste libera todos los recursos ocupados.

MIDP provee de APIs para crear una aplicación de dos formas distintas:

1) Usando una interfaz a "bajo nivel".
2) Usando una interfaz a "alto nivel".

Una aplicación de **bajo nivel** se caracteriza porque para crearla se utiliza la pantalla como un lienzo en el que se tienen que ir dibujando píxel a píxel todos los elementos que la van a formar. Esta forma de trabajar es más pesada, pero consigue efectos visuales más logrados. Las aplicaciones de "bajo nivel" son ideales para el desarrollo de juegos y aplicaciones gráficas, ya que se obtiene un control total de todos los elementos de una pantalla.

Una aplicación de **alto nivel** se caracteriza porque para su creación se utilizan

componentes definidos en MIDP y que cualquier dispositivo MIDP soporta. Estos componentes son cajas de texto, grupos de opciones, listas, etc. Estos componentes tienen la desventaja de que no son personalizables, ya que cada dispositivo los implementa de una forma diferente, no obstante, tiene la gran ventaja de que son aplicaciones de un alto grado de estandarización.

3.3.3 El paquete javax.microedition.midlet

El paquete javax.microedition.midlet define las aplicaciones MIDP y su comportamiento con respecto al entorno de ejecución. En la Tabla Nº 3.6 se muestran las clases que están incluidas en este paquete.

TABLA Nº 3.6 Clases del paquete javax.microedition.midlet (3)

Clases	Descripción
MIDlet	Aplicación MIDP.
MIDletstateChangeException	Indica que el cambio de estado ha fallado.

a) Clase MIDlet

public abstract class MIDlet

La aplicación debe extender a esta clase para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades. El MIDlet puede por sí mismo realizar cambios de estado invocando a los métodos apropiados.

protected MIDlet()

Constructor de clase sin argumentos. Si la llamada a este constructor falla, se lanzaría la excepción SecurityException.

public final int checkPermission**(String** permiso**)**

Consigue el estado del permiso especificado. Este permiso está descrito en el atributo MIDlet-Permission del archivo JAD. En caso de no existir el permiso por el que se pregunta, el método devolverá un cero. En caso de no conocer el estado del permiso en ese momento debido a que sea necesaria alguna acción por parte del usuario, el método devolverá un -1. Los valores devueltos por el método se corresponden con la siguiente descripción:

a) 0 si el permiso es denegado.

b) 1 si el permiso es permitido.

c) -1 si el estado es desconocido.

protected abstract void destroyApp(**boolean** incondicional) **throws** MIDletstateChangeEception

Indica la terminación del MIDlet y su paso al estado "Destruido". En el estado de "Destruido" el MIDlet debe liberar todos los recursos y salvar cualquier dato en el

almacenamiento persistente que deba ser guardado. Este método puede ser llamado desde los estados "Pausa" o "Activo".

Si el parámetro 'incondicional' es false, el MIDlet puede lanzar la excepción MIDletstateChangeEception para indicar que no puede ser destruido en ese momento. Si es true, el MIDlet asume su estado de destruido independientemente de cómo finalice el método.

public final String getAppProperty(**String** key)

Este método proporciona al MIDlet un mecanismo que le permite recuperar el valor de las propiedades desde el AMS. Las propiedades se consiguen por medio de los archivos manifest y JAD. El nombre de la propiedad a recuperar debe ir indicado en el parámetro key. El método devuelve un String con el valor de la propiedad o null si no existe ningún valor asociado al parámetro key. Si key es null se lanzará la excepción NullPointerException.

public final void notifyDestroyed()

Este método es utilizado por un MIDlet para indicar al AMS que ha entrado en el estado de "Destruido". En este caso, todos los recursos ocupados por el MIDlet deben ser liberados por éste de la misma forma que si se hubiera llamado al método MIDlet.destroyApp(). El AMS considerará que todos los recursos que ocupaba el MIDlet están libres para su uso.

public final void notifyPaused()

Se notifica al AMS que el MIDlet no quiere estar "Activo" y que ha entrado en el estado de "Pausa". Este método sólo debe ser invocado cuando el MIDlet esté en el estado "Activo". Una vez invocado este método, el MIDlet puede volver al estado "Activo" llamando al método MIDlet.startApp(), o ser destruido llamando al método MIDlet.destroyApp(). Si la aplicación es pausada por sí misma, es necesario llamar al método MIDlet.resumeRequest() para volver al estado "Activo".

protected abstract void pauseApp()

Indica al MIDlet que entre en el estado de "Pausa". Este método sólo debe ser llamado cuando el MIDlet esté en estado "Activo". Si ocurre una excepción RuntimeException durante la llamada a MIDlet.pauseApp(), el MIDlet será destruido inmediatamente. Se llamará a su método MIDlet.destroyApp() para liberar los recursos ocupados.

public final boolean platformRequest(String url)

Establece una conexión entre el MIDlet y la dirección URL. Dependiendo del contenido de la URL, el dispositivo ejecutará una determinada aplicación que sea capaz de leer el contenido y dejar al usuario que interactúe con él.

public final void resumeRequest()

Este método proporciona un mecanismo a los MIDlets mediante el cual pueden indicar al AMS su interés en pasar al estado de "Activo". El AMS, en consecuencia, es el encargado de determinar qué aplicaciones han de pasar a este estado llamando al método MIDlet.startApp().

protected abstract void startApp() **throws** MIDletstateChangeEception

Este método indica al MIDlet que ha entrado en el estado "Activo". Este método sólo puede ser invocado cuando el MIDlet está en el estado de "Pausa". En el caso de que el MIDlet no pueda pasar al estado "Activo" en este momento pero si pueda hacerlo en un momento posterior, se lanzaría la excepción MIDletstateChangeEception.

A través de los métodos anteriores se establece una comunicación entre el AMS y el MIDlet. Por un lado se tiene que los métodos startApp(), pauseApp() y destroyApp() los utiliza el AMS para comunicarse con el MIDlet, mientras que los métodos resumeRequest(), notifyPaused() y notifyDestroyed() los utiliza el MIDlet para comunicarse con el AMS.

En la Tabla Nº 3.7 se puede ver un resumen de los métodos que dispone la clase MIDlet.

TABLA Nº 3.7 Métodos de la clase MIDlet (3)

Resumen	
Constructores	
protected	MIDlet()
Métodos	
int	checkPermission(String permiso)
protected abstract void	destroyApp(_olean unconditional)
String	getAppProperty(String key)
void	notifyDestroyed()
void	notifyPaused()
protected abstract void	pauseApp()
boolean	platformRequest()
void	resumeRequest()
protected abstract void	startApp()

b) Clase MIDletChangeStateException

public class MIDletstateChangeException **extends** Exception

Esta excepción es lanzada cuando ocurre un fallo en el cambio de estado de un MIDlet.

3.4 JSR-82

Este API está dividido en dos partes: El paquete javax.bluetooth y el paquete javax.obex. Los dos paquetes son totalmente independientes. El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. Por el contrario el paquete javax.obex permite manejar el protocolo de alto nivel OBEX (OBject EXchange). Este protocolo es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos. El protocolo OBEX es un estándar desarrollado por IrDA y es utilizado también sobre otras tecnologías inalámbricas distintas de Bluetooth.

La plataforma principal de desarrollo del API JSR-82 es J2ME. El API ha sido diseñado para depender de la configuración CLDC. Sin embargo existen implementaciones para usar este API en J2SE.

El paquete javax.bluetooth

En una comunicación Bluetooth existe un dispositivo que ofrece un servicio (servidor) y otros dispositivos acceden a él (cliente). Un cliente Bluetooth deberá realizar las siguientes acciones:

1) Búsqueda de dispositivos. La aplicación realizará una búsqueda de los dispositivos Bluetooth a su alcance que estén en modo conectable.

2) Búsqueda de servicios. La aplicación realizará una búsqueda de servicios por cada dispositivo.

3) Establecimiento de la conexión. Una vez encontrado un dispositivo que ofrece el servicio deseado, la aplicación se conectará a él.

4) Comunicación. Ya establecida la conexión, la aplicación puede leer y escribir en ella.

Por otro lado, un servidor Bluetooth deberá hacer las siguientes operaciones:

1) Crear una conexión servidora.

2) Especificar los atributos de servicio.

3) Abrir las conexiones clientes.

La clase UUID (universally unique identifier) representa identificadores únicos universales. Se trata de enteros de 128 bits que identifican protocolos y servicios. El API javax.bluetooth permite usar dos mecanismos de conexión: SPP y L2CAP. Mediante SPP se obtiene un InputStream y un OutputStream. Mediante L2CAP se envía y recibe arrays de bytes.

El paquete javax.obex

Este paquete es totalmente independiente del paquete javax.bluetooth, es decir, cuando se hace una aplicación utilizando el protocolo OBEX no se usa ninguna de las clases del paquete javax.bluetooth.

OBEX es un protocolo que se usa generalmente para la transferencia de archivos. Consiste en el intercambio de mensajes entre el cliente y el servidor. Tales mensajes consisten en un conjunto de cabeceras de mensaje y opcionalmente un cuerpo de mensaje.

CAPÍTULO IV
METODOLOGÍA PARA LA SOLUCION DEL PROBLEMA

4.1 Alternativas de solución

Un sistema de control de acceso físico para puertas involucra hardware y software. Consiste en implementar algún mecanismo electrónico mecánico que identifique si la persona que quiere entrar está autorizada y en base a ello otorgar acceso a alguna dependencia (permitir el ingreso). Entre las alternativas más comunes se tiene:

1) Identificación de huella digital (biometría).
2) Tarjetas de código de barras.
3) Tarjetas de banda magnética.
4) Tarjetas de proximidad.
5) Teclado para ingresar número de usuario y contraseña.

En la Figura 4.1 se puede ver un ejemplo de control de acceso en puerta mediante la identificación biométrica.

Fig. 4.1 Identificación de huella digital

Sin embargo, el presente informe abarca el control de acceso para puertas utilizando la tecnología Bluetooth y teléfonos celulares.

4.2 Solución del problema

Para la solución del problema se utilizó varias tecnologías y productos: Bluetooth, Java 2 Micro Edition (J2ME), Módulo Bluegiga, placa Arduino, teléfono celular. El sistema de control es de tipo ON/OFF. Los elementos que forman el sistema de comunicación son:

1) Un teléfono celular.

2) Un periférico Bluetooth.

El periférico Bluetooth forma parte del sistema empotrado que se encuentra detrás de la puerta. El sistema empotrado tiene la capacidad de abrir la puerta. El proceso de comunicación es el siguiente:

1) El teléfono celular se comunica mediante Bluetooth al sistema empotrado asociado a una puerta. Una vez establecida la comunicación, el dispositivo móvil envía la contraseña del usuario para intentar acceder a dicha puerta.

2) El sistema empotrado abre o no la puerta en función de la validez de la contraseña de usuario.

4.2.1 Hardware utilizado

Arduino BT

La plataforma Arduino tiene las siguientes características:

1) Es una plataforma de desarrollo de computación física de código abierto, basada en una placa con un sencillo microcontrolador y un entorno de desarrollo para crear software (programas) para la placa.

2) Simplifica el proceso de trabajar con microcontroladores.

3) El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux.

4) El entorno de programación es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados.

5) El software Arduino está publicado bajo una licencia libre y puede ampliarse a través de librerías C++.

6) Está basado en los microcontroladores ATMEGA168, ATMEGA328 y ATMEGA1280.

El Arduino BT es una placa Arduino con el módulo Bluetooth incorporado, que permite la comunicación inalámbrica con computadoras, teléfonos y otros dispositivos. Es una placa basada en el microcontrolador ATMEGA328 y el módulo Bluetooth Bluegiga WT11. El microcontrolador puede ser programado inalámbricamente sobre la conexión Bluetooth.

El WT11 se comunica con el ATMEGA por medio del serial, la comunicación está configurada a 115200 baudios. El módulo Bluetooth WT11 se puede configurar con

comandos AT enviados a través del puerto serie del ATMEGA. El Arduino BT se muestra en la Figura 4.2.

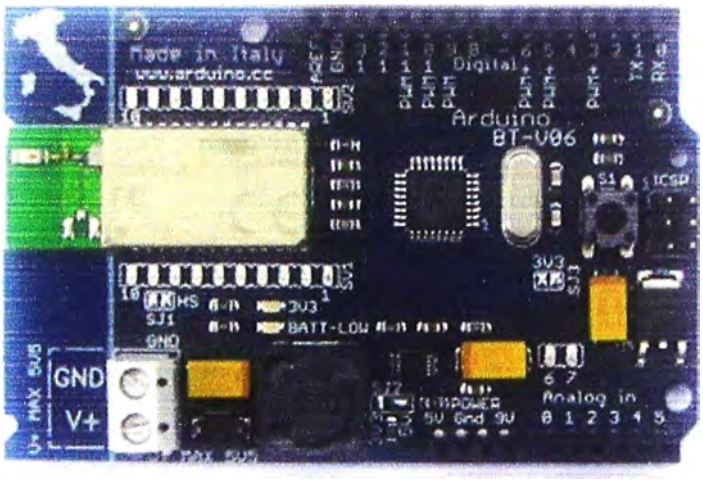

Fig. 4.2 Arduino BT (10)

Bluegiga WT11

Provee una solución ideal para desarrolladores que quieren integrar en sus diseños la tecnología inalámbrica Bluetooth. Este módulo se muestra en la Figura 4.3.

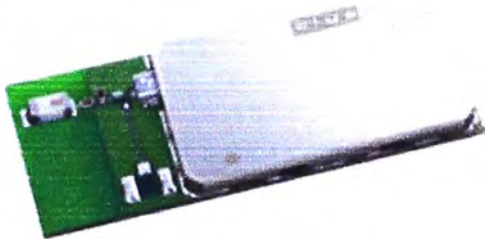

Fig. 4.3 Bluegiga WT11 (11)

Principales características:

1) Bluetooth clase 1.
2) Antena chip integrada.
3) Bluetooth 2.0 + EDR.
4) Tamaño: 35 x 4 x 2.3 mm.
5) Firmware iWRAP.
6) Chipset CSR BlueCore 04.

4.2.2 Desarrollo

Se desarrolló una aplicación para el teléfono celular utilizando J2ME y el entorno de programación NetBeans. También se creó una aplicación para el periférico Bluetooth, que fue programado con el software Arduino. Es importante resaltar que se utilizó software libre en la programación de las dos aplicaciones, de esta forma no se necesita pagar por

licencias y la implementación del sistema de acceso resulta más barata.

a) Dispositivo móvil

La aplicación para el teléfono celular sigue los siguientes pasos:

1) Al lanzar la aplicación aparecerá en el dispositivo móvil la ventana principal con las opciones: Busqueda, Enviar y Salir.

2) Al seleccionar la opción Busqueda, comenzará una búsqueda general que consiste en la detección de todos los dispositivos Bluetooth (puertas) que estén cerca del dispositivo móvil.

3) Al finalizar dicha búsqueda aparecerá en pantalla una lista con todas las puertas disponibles.

4) El siguiente paso será seleccionar aquella puerta a la que se desea ingresar.

5) Una vez seleccionada la puerta elegida, presionar la opción Enviar, aparecerá en la pantalla un campo de texto para introducir la contraseña. Presionar OK y después de unos segundos se mostrará una alerta de confirmación que el mensaje ha sido enviado.

La Figura 4.4 muestra el diagrama de flujo de la aplicación para el teléfono celular.

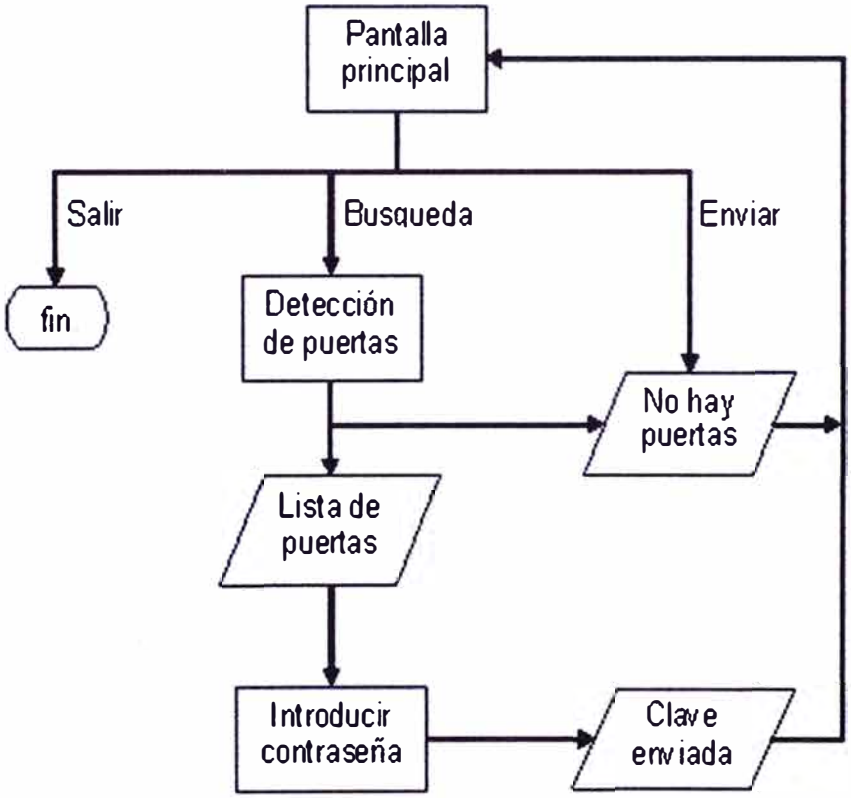

Fig. 4.4 Diagrama de flujo de la aplicación móvil

Antes de instalar el programa en el celular se hicieron pruebas cliente servidor en el entorno NetBeans. El presente informe no explica el funcionamiento del programa servidor porque no es parte del sistema de control. La aplicación del cliente (celular) usa

tres clases:

1) SPPClienteMIDlet.

2) SPPCliente.

3) Mensaje.

La clase SPPClienteMIDlet hereda de la clase MIDlet, es aquí donde se realiza la búsqueda de dispositivos Bluetooth. En la Figura 4.5 se muestra la implementación del comando "Busqueda".

```
//Se limpia la lista
dispositivos_encontrados.removeAllElements();
servicios_encontrados.removeAllElements();
try {
    dispositivoLocal = LocalDevice.getLocalDevice();
    dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
    da = dispositivoLocal.getDiscoveryAgent();
    da.startInquiry(DiscoveryAgent.GIAC, new Listener());
    c.escribirMensaje("Por favor espere...");
} catch (BluetoothStateException be) {
    mostrarAlarma(be, c, 0);
}
```

Fig. 4.5 Código fuente para buscar dispositivos

Después de encontrar dispositivos Bluetooth, se tiene que buscar los servicios que ofrecen dichos dispositivos. Puede haber muchos servicios disponibles en los dispositivos cercanos. Sin embargo, el sistema de acceso solamente usa el puerto serie, con el perfil SPP. Los demás servicios no se consideran para el presente diseño. El código para buscar el servicio puerto serie se muestra en la Figura 4.6.

```
            try {

                //Se busca el puerto serie 0x1101
                da.searchServices(null, new UUID[]{new UUID(0x1101)},
dispositivo_remoto, new Listener());
            } catch (BluetoothStateException be) {
                mostrarAlarma(be, c, 0);
            }
```

Fig. 4.6 Código fuente para buscar el servicio 0x1101

Si la aplicación Bluetooth del teléfono celular no encuentra el servicio 0x1101, no habrá comunicación con el periférico Bluetooth. Si la aplicación Bluetooth encuentra el servicio, el usuario tiene que enviar la clave como mensaje. En la Figura 4.7 se muestra

el método para enviar mensajes.

```
    public void enviarMensaje(String msg) {
        ServiceRecord sr = (ServiceRecord) servicios_encontrados.elementAt(0);

        //La URL asociada a este servicio en el dispositivo remoto
        String URL = sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
        try {
            //Se obtene la conexion y el stream de este servicio
            StreamConnection con = (StreamConnection) Connector.open(URL);
            DataOutputStream out = con.openDataOutputStream();

            //Escribir datos en el stream
            out.writeUTF(msg);
            out.flush();

            //Cerrar la conexion
            out.close();
            con.close();
            mostrarAlarma(null, c, 2);
        } catch (Exception e) {
            mostrarAlarma(e, c, 0);
        }
    }
```

Fig. 4.7 Código fuente para enviar mensajes

Para la emulación de la aplicación se usó el Wireless Toolkit 2.5.2 que permite emular varios teléfonos a la vez. En la Figura 4.8 se muestra la ventana principal de la aplicación y en la Figura 4.9 se ve el campo de texto para escribir la clave.

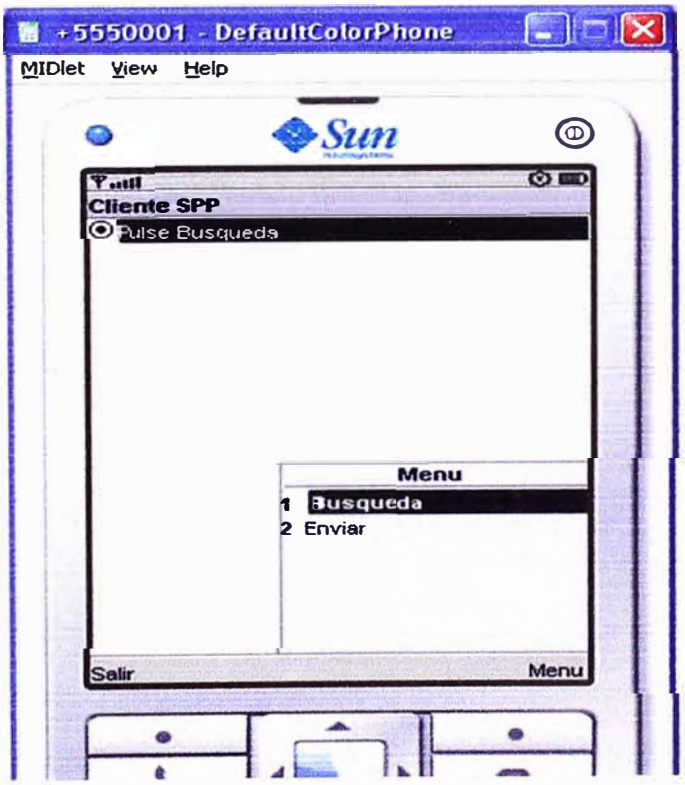

Fig. 4.8 Ventana principal

Fig. 4.9 Introducir clave

b) Aplicación para el periférico Bluetooth

En la Figura 4.10 se muestra el entorno de desarrollo del software Arduino. Se puede ver un ejemplo que viene con el software.

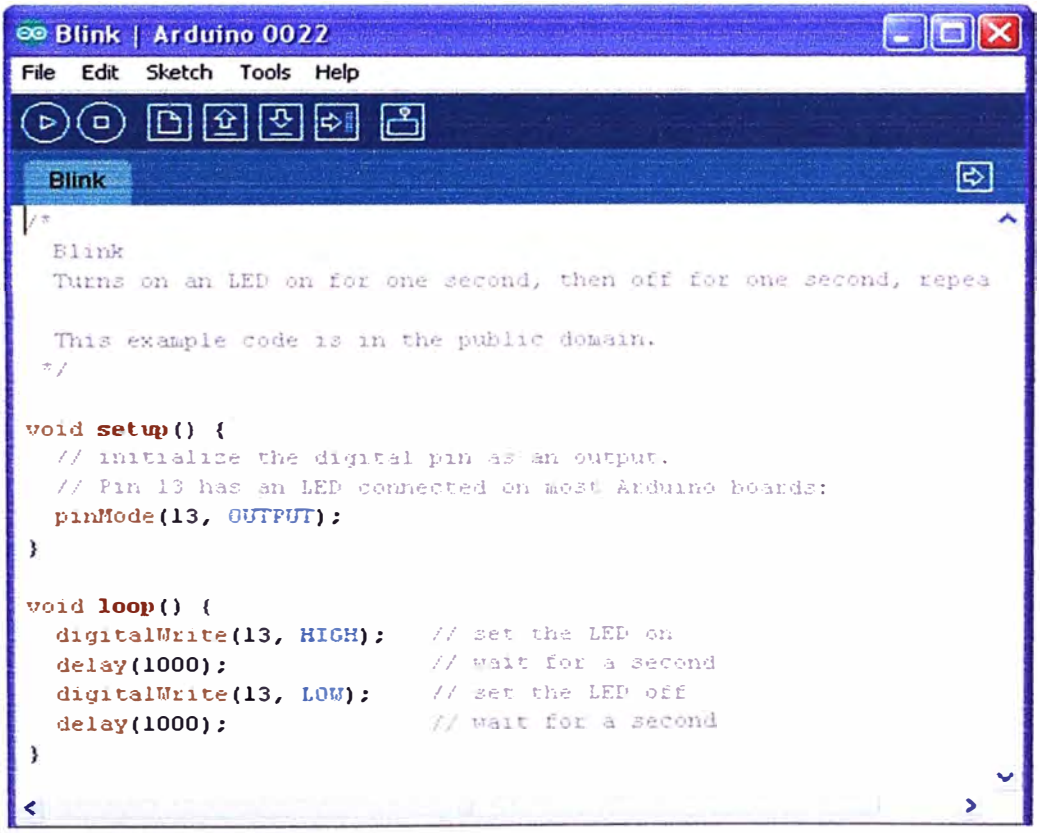

Fig. 4.10 Entorno de desarrollo Arduino

El bucle principal que se está ejecutando en el periférico Bluetooth tiene un retardo de 100 milisegundos en cada iteración para tener un control sobre el tiempo. El programa tiene dos estados:

Estado 1 - Espera/Lectura: Este es el estado habitual del sistema, el programa espera una conexión entrante por Bluetooth y lee los caracteres recibidos (contraseña). Pasa al estado 2.

Estado 2 - Validación: El programa verifica la contraseña, si es correcta envía una señal para abrir la puerta. En caso contrario no envía ninguna señal. Pasa al estado 1.

En la Figura 4.11 se ve el diagrama de flujo de la aplicación para el periférico Bluetooth. Lo que se programa es el microcontrolador ATMEGA328, con el software Arduino esta tarea es sencilla. El módulo Bluegiga WT11 ya está configurado para prestar el servicio de puerto serie.

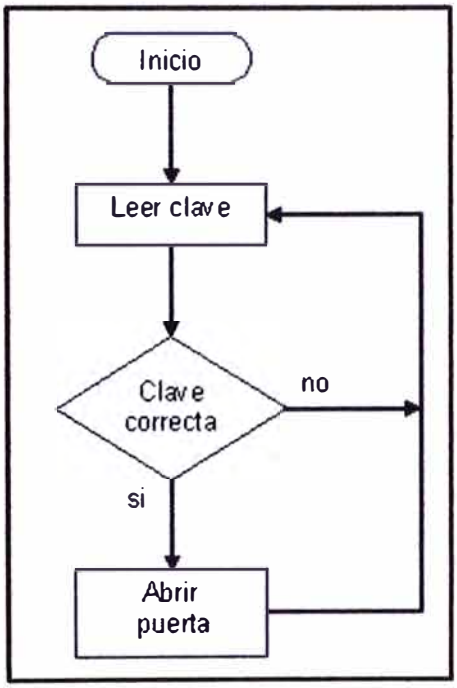

Fig. 4.11 Diagrama de flujo de la aplicación para el periférico

CONCLUSIONES Y RECOMENDACIONES

1. Se demostró que se puede usar la tecnología Bluetooth como medio de comunicación en un sistema de control de acceso para puertas. También queda demostrado que los usuarios de este diseño pueden utilizar sus teléfonos celulares para identificarse en la puerta a la que desean ingresar.
2. Este diseño no pretende imponerse frente a los otros sistemas de acceso físico, como son tarjetas de proximidad, botoneras, identificadores biométricos, bandas magnéticas, códigos de barras, etc. Lo que se busca es aportar una alternativa al conjunto de tecnologías mencionado.
3. Una ventaja del diseño con tecnología Bluetooth, es que el periférico Bluetooth se encuentra detrás de la puerta. Esto no permite que sea manipulado desde afuera por personas malintencionadas.
4. El sistema de control de acceso presentado en este informe, resultaría más barato de implementar que un sistema de identificación biométrica. Considerando que los usuarios ya tienen un teléfono celular que soporte al API JSR-82.
5. Se recomienda que la clave de acceso tenga más dígitos. El presente diseño de sistema de acceso sólo utiliza un dígito para la clave de acceso.
6. También se recomienda, que el mensaje que se envíe a través de la comunicación Bluetooth tiene que estar encriptado. De esta manera, el sistema de considera más seguro. Los teléfonos celulares tienen que soportar el API JSR-177 para la encriptación de mensajes.

ANEXO A
GLOSARIO DE TÉRMINOS Y SIGLAS

Término o Sigla	Descripción
ACL	Asynchronous Connectionless
AFH	Adaptive Frequency Hopping
API	Application Programming Interface
CVM	Compact Virtual Machine
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
EDR	Enhanced Data Rate
GAP	Generic Access Profile
GOEP	Generic Object Exchange Profile
HCI	Host Controller Interface
HID	Human Interface Devices
IrDA	Infrared Data Association
ISM	Industrial, Scientific and Medical
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kilo Virtual Machine
L2CAP	Logical Link Control and Adaptation Protocol
LMP	Link Manager Protocol
MIDlet	es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC
MIDP	Mobile Information Device Profile
OBEX	Object Exchange
PDA	Personal Digital Assistant
RF	Radio Frecuency
SCO	Synchronous Connection Oriented
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol
SPP	Serial Port Profile
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

ANEXO B
CÓDIGO FUENTE EN EL PERIFÉRICO BLUETOOTH

Código fuente para la placa Arduino BT

```
//global variables
int password = 0;
boolean abrirPuerta = false;
//

void setup() {
  Serial.begin(115200);
  pinMode(6, OUTPUT);
}

void loop() {
  delay(100);
  if (Serial.available() > 0) {
    password = Serial.read();
    Serial.print("password: ");
    Serial.println(password, DEC);
  }
  if(password == 50) {//numero 2 - ASCII 50
    abrirPuerta = true;
    digitalWrite(6,HIGH);
    delay(2000);
    password = 0;
  }
  if(password == 0) {
    abrirPuerta = false;
    digitalWrite(6,LOW);
    password = 1;
  }

}
```

ANEXO C
CÓDIGO FUENTE DEL CLIENTE BLUETOOTH

C.1 La clase SPPClienteMIDlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;

public class SPPClienteMIDlet extends MIDlet implements CommandListener {

   //Se crean las variables necesarias
   public static SPPClienteMIDlet SPPc = null;
   public static Display display;
   private SPPCliente c = null;
   private Mensaje msg = null;

   //Objetos Bluetooth necesarios
   public LocalDevice dispositivoLocal;
   public DiscoveryAgent da;

   //Lista de dispositivos y servicios encontrados
   public static Vector dispositivos_encontrados = new Vector();
   public static Vector servicios_encontrados = new Vector();
   public static int dispositivo_seleccionado = -1;

   //Constructor
   public SPPClienteMIDlet() {
      SPPc = this;
   }

   //Se implementa el ciclo de vida del MIDlet
   public void startApp() {
      display = Display.getDisplay(this);
      c = new SPPCliente();
      msg = new Mensaje();

      //Se muestra la lista de dispositivos(vacia al principio)
      c.mostrarDispositivos();
      display.setCurrent(c);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }

   //Este metodo se encarga de las tareas necesarias para salir del MIDlet
   public static void salir() {
      SPPc.destroyApp(true);
      SPPc.notifyDestroyed();
      SPPc = null;
   }
```

```
    //Este metodo se encarga de dar un aviso de alarma cuando se produce una excepcion
    public void mostrarAlarma(Exception e, Screen s, int tipo) {
        Alert alerta = null;
        if (tipo == 0) {
            alerta = new Alert("Excepcion", "Se ha producido la excepcion " +
e.getClass().getName(), null,
                    AlertType.ERROR);
        } else if (tipo == 1) {
            alerta = new Alert("Error", "No ha seleccionado un dispositivo ", null,
AlertType.ERROR);
        } else if (tipo == 2) {
            alerta = new Alert("Informacion", "La clave ha sido enviada ", null,
AlertType.INFO);
        }
        alerta.setTimeout(Alert.FOREVER);
        display.setCurrent(alerta, s);
    }

    //Interaccion del usuario
    public void commandAction(Command co, Displayable d) {
        if (d == c && co.getLabel().equals("Busqueda")) {

            //Se limpia la lista
            dispositivos_encontrados.removeAllElements();
            servicios_encontrados.removeAllElements();
            try {
                dispositivoLocal = LocalDevice.getLocalDevice();
                dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
                da = dispositivoLocal.getDiscoveryAgent();
                da.startInquiry(DiscoveryAgent.GIAC, new Listener());
                c.escribirMensaje("Por favor espere...");
            } catch (BluetoothStateException be) {
                mostrarAlarma(be, c, 0);
            }
        } else if (d == c && co.getLabel().equals("Enviar")) {
            dispositivo_seleccionado = c.getSelectedIndex();

            //El usuario selecciona un dispositivo
            if (dispositivo_seleccionado == -1 || dispositivo_seleccionado >=
dispositivos_encontrados.size()) {
                mostrarAlarma(null, c, 1);
                return;
            }
            display.setCurrent(msg);
        } else if (d == c && co.getLabel().equals("Salir")) {
            salir();
        } else if (d == msg && co.getLabel().equals("OK")) {
            servicios_encontrados.removeAllElements();

            //Se busca el servicio de puerto serie en el dispositivo seleccionado
            RemoteDevice dispositivo_remoto = (RemoteDevice)
dispositivos_encontrados.elementAt(dispositivo_seleccionado);
            try {
```

```
            //Se busca el puerto serie 0x1101
            da.searchServices(null, new UUID[]{new UUID(0x1101)}, dispositivo_remoto,
new Listener());
        } catch (BluetoothStateException be) {
            mostrarAlarma(be, c, 0);
        }
    }
  }

  //Este metodo se va a encargar de enviar un mensaje al primer ServiceRecord usando
  //el Serial Port Profile
  public void enviarMensaje(String msg) {
    ServiceRecord sr = (ServiceRecord) servicios_encontrados.elementAt(0);

    //La URL asociada a este servicio en el dispositivo remoto
    String URL =
sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
    try {
      //Se obtene la conexion y el stream de este servicio
      StreamConnection con = (StreamConnection) Connector.open(URL);
      DataOutputStream out = con.openDataOutputStream();

      //Escribir datos en el stream
      out.writeUTF(msg);
      out.flush();

      //Cerrar la conexion
      out.close();
      con.close();
      mostrarAlarma(null, c, 2);
    } catch (Exception e) {
      mostrarAlarma(e, c, 0);
    }
  }

  //Se implementa el DiscoveryListener
  public class Listener implements DiscoveryListener {

    //Se implementa los metodos del interfaz DiscoveryListener
    public void deviceDiscovered(RemoteDevice dispositivoRemoto, DeviceClass clase)
{
      System.out.println("Se ha encontrado un dspositivo Bluetooth");
      dispositivos_encontrados.addElement(dispositivoRemoto);
    }

    public void inquiryCompleted(int completado) {
      System.out.println("Se ha completado la busqueda de dispositivos");
      if (dispositivos_encontrados.isEmpty()) {
        Alert alerta = new Alert("Problema", "No se ha encontrado dispositivos", null,
AlertType.INFO);
        alerta.setTimeout(3000);
        c.escribirMensaje("Presione descubrir dispositivos");
        display.setCurrent(alerta, c);
      } else {
```

```
            c.mostrarDispositivos();
            display.setCurrent(c);
        }
    }

    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
        System.out.println("Se ha encontrado un servicio remoto");
        for (int i = 0; i < servRecord.length; i++) {
            ServiceRecord record = servRecord[i];
            servicios_encontrados.addElement(record);
        }
    }

    public void serviceSearchCompleted(int transID, int respCode) {
        System.out.println("Terminada la busqueda de servicios");

        //Si hay un servicio puerto serie, se usa para mandar el mensaje
        if (servicios_encontrados.size() > 0) {
            enviarMensaje(msg.getString());
        } else {

            //Si no hay ningun servicio de puerto serie
            c.mostrarDispositivos();
            display.setCurrent(c);
        }
    }
  }
}
```

C.2 La clase SPPCliente

```
import javax.microedition.lcdui.*;
import javax.bluetooth.*;

public class SPPCliente extends List {

  public SPPCliente() {
    super("Cliente SPP", List.EXCLUSIVE);
    addCommand(new Command("Busqueda", Command.SCREEN, 1));
    addCommand(new Command("Enviar", Command.SCREEN, 1));
    addCommand(new Command("Salir", Command.EXIT, 1));
    this.setCommandListener(SPPClienteMIDlet.SPPc);
  }
//Este metodo se encarga de limpiar la pantalla y de mostrar un mensaje
  public void escribirMensaje(String str) {
    for (int i = 0; i < this.size(); i++) {
      delete(i);
    }
    append(str, null);
  }
//Este metodo muestra los "friendly names" de los dispositivos remotos
  public void mostrarDispositivos() {
    for (int i = 0; i < this.size(); i++) {
      delete(i);
```

```
        }
        if (SPPClienteMIDlet.dispositivos_encontrados.size() > 0) {
            for (int i = 0; i < SPPClienteMIDlet.dispositivos_encontrados.size(); i++) {
                try {
                    RemoteDevice dispositivoRemoto = (RemoteDevice)
SPPClienteMIDlet.dispositivos_encontrados.elementAt(i);
                    append(dispositivoRemoto.getFriendlyName(false), null);
                } catch (Exception e) {
                    System.out.println("Se ha producido una excepcion");
                }
            }
        } else {
            append("Pulse Busqueda", null);
        }
    }
}
```

C.3 La clase Mensaje

```
import javax.microedition.lcdui.*;

public class Mensaje extends TextBox {

    public Mensaje() {
        super("Introducir clave", "", 50, TextField.ANY);
        addCommand(new Command("OK", Command.SCREEN, 1));
        this.setCommandListener(SPPClienteMIDlet.SPPc);
    }
}
```

ANEXO D
RADIO FRECUENCIAS COMUNES

Ítem	Rango de Frecuencia
Radio AM	535 KHz – 1.6 MHz
Abre puertas para garaje	40 MHz
Monitores para bebés	49 MHz
Canales de TV 2-6	54 MHz – 88 MHz
Radio FM	88 MHz – 108 MHz
Canales de TV 7-13	174 MHz – 216MHz
Canales de TV 14-83	512 MHz – 806 MHz
CDMA teléfono celular	824 MHz – 894 MHz
GSM teléfono celular	880 MHz – 960 MHz
Teléfonos inalámbricos	900 MHz
GPS	1.227 GHZ – 1.575 GHz
PCS teléfono celular	1.85 GHz – 1.99 GHz
802.11b	2.4 GHz – 2.483 GHz
802.11g	2.4 GHz – 2.483 GHz
Bluetooth	2.4 GHz – 2.483 GHz
Teléfonos inalámbricos	2.4 GHz
802.11a	5.15 GHz – 5.35 GHz

BIBLIOGRAFÍA

[1] Timothy J. Thompson, "Bluetooth Application Programming with the Java APIs Essentials Edition", Morgan Kaufmann, 2008.

[2] Bruce Hopkins, "Bluetooth for Java", Apress, 2003.

[3] Sergio Gálvez, "Java a Tope: J2ME", Universidad de Málaga, 2003.

[4] Albert S. Huang, "Bluetooth Essentials for Programmers", Cambridge, 2007.

[5] Christian Gehrmann, "Bluetooth Security", Artech House, 2004.

[6] Alberto Gimeno, "JSR-82: Bluetooth desde Java", Creative Commons, 2004.

[7] http://www.bluezona.com/index.php?option=com_content&task=view&id=25&Itemid=50/

[8] http://es.wikipedia.org/wiki/Bluetooth

[9] http://bucefalo.com.mx/diferencia-entre-las-versiones-de-bluetooth

[10] http://www.arduino.cc/en/Main/ArduinoBoardBluetooth

[11] http://www.bluegiga.com/WT11_Class_1_Bluetooth_Module