Escuela Superior Politécnica de Chimborazo

Facultad de Informática y Electrónica

Escuela de Ingeniería en Electrónica y Tecnología en Computación

Diseño e implementación de un prototipo para el control de usuarios basados en teléfonos móviles con bluetooth

Tesis de Grado previa obtención del título de Ingeniería en Electrónica y Tecnología en Computación

Jaime Rodrigo Vinueza Coba

Riobamba, 2010

Agradezco a Dios por su infinito amor al concederme la vida y brindarme fuerza para los retos de cada día.

A mi madre que con su sacrificio y entrega me ha enseñado el camino correcto, y a Amy que gracias a su apoyo ha sido posible el desarrollo de esta Tesis.

Firmas	**Notas**
Dr. Romeo Rodríguez **DECANO DE LA FACULTAD DE** **INFORMATICA Y ELECTRONICA**	____________________
Ing. Paúl Romero **DIRECTOR DE LA ESCUELA DE** **INGENIERIA ELECTRONICA Y TECNOLOGIA** **EN COMPUTACION**	____________________
Ing. Paul Romero **DIRECTOR DE TESIS**	____________________
Ing. Hugo Moreno **MIEMBRO DE TRIBUNAL**	____________________
Lcdo. Carlos Rodríguez **DIRECTOR DEL CENTRO DE DOCUMENTACION**	____________________

Nota: ____________________

Yo, Jaime Rodrigo Vinueza Coba, soy el responsable de las ideas, doctrinas y resultados expuestos en esta Tesis, y el patrimonio intelectual de la misma pertenecen a la Escuela Superior Politécnica de Chimborazo

____

Jaime Rodrigo Vinueza

Índice General

Capítulo I

MARCO REFERENCIAL ***14***

1.1. Antecedentes *14*

1.2. Justificación *15*

1.3. Objetivos *16*

1.3.1. Objetivo General 16

1.3.2. Objetivos Específicos 16

1.4. Hipótesis *16*

Capítulo II

BLUETOOTH y JAVA ***17***

2.1. Bluetooth *18*

2.1.1. Banda de frecuencia libre 18

2.1.2. Salto de frecuencia 19

2.1.3. Definición de canal 19

2.1.4. Definición de paquete 20

2.1.5. Definición de enlace físico 21

2.1.6. Inmunidad a las interferencias 22

2.1.7. Red inalámbrica 23

2.1.7.1. Piconets 23

2.1.7.1.1. Estableciendo conexión 24

2.1.7.2. Scatternet 26

2.1.7.3. Comunicación inter-piconet 28

2.1.8. Seguridad 29

2.2. Java ***30***

2.2.1. Filosofía 31

2.2.2. Orientado a objetos 31
2.2.3. Independencia de la plataforma 33
2.2.4. Sintaxis 35
2.2.4.1. Aplicaciones autónomas “Hola Mundo” 35
2.2.4.2. Aplicaciones con ventanas 37
2.2.5. Entornos de funcionamiento 39
2.2.5.1. En dispositivos móviles y sistemas empotrados 39
2.2.5.2. En el navegador web 40
2.2.5.3. En sistemas de servidor 40
2.2.5.4. En aplicaciones de escritorio 42
2.2.6. Plataformas soportadas 42
2.2.7. Industria relacionada 43
2.2.8. El lenguaje 43
2.2.9. Recursos 44
2.2.9.1. JRE 44
2.2.9.2. Componentes 45
2.2.10. APIs 46

CAPITULO III

J2ME, J2SE Y BLUETOOTH *48*

3.1. J2ME *50*
3.1.1. Creación de MIDlet 52
3.1.1.1. Diseño 52
3.1.1.2. Código 53
3.1.1.3.Compilar 55
3.1.1.4. Preverificado 56
3.1.1.5.Paquete 57
3.1.1.6. Prueba 58
3.1.1.7. Desplegar 59

3.1.2. Ciclo de vida del MIDlet 59

3.1.3. API JSR-82 63

3.1.3.1. Programación de aplicaciones Bluetooth 64

3.1.3.1.1. Inicialización 65

3.1.3.1.1.1. BCC 65

3.1.3.1.1.2. Inicialización de la pila 65

3.1.3.1.2. Descubrimiento de dispositivos y servicios 66

3.1.3.1.2.1. Descubrir Dispositivos (Device Discovery) 66

3.1.3.1.2.1.1. Clases del Device Discovery 67

3.1.3.1.2.2. Descubrir Servicios (Service Discovery) 67

3.1.3.1.2.2.1. Clases del Service Discovery 68

3.1.3.1.2.2.2. Registro del Servicio (Service Registration) 69

3.1.3.1.2.2.3. Responsabilidades del Registro de Servicio 70

3.1.3.1.2.2.4. Modos conectable y no conectable 71

3.1.3.1.2.2.5. Clases del Service Registration 72

3.1.3.1.3. Manejo del Dispositivo 73

3.1.3.1.3.1. Perfil de Acceso Genérico (GAP) 74

3.1.3.1.3.1.1. Clases del GAP 74

3.1.3.1.4. Comunicación 75

3.1.3.1.4.1. Perfil de Puerto Serie 76

3.1.3.1.4.1.1. Conexiones URL de un cliente y servidor SPP 76

3.1.3.1.4.1.2. Registro del Servicio de Puerto Serie 77

3.1.3.1.4.2. Establecimiento de la conexión 78

3.1.3.1.4.2.1. Establecimiento de la conexión del servidor 78

3.1.3.1.4.2.2. Establecimiento de la conexión del cliente 79

3.1.3.1.4.2.3. Registro de Servicio del SPP 80

CAPITULO IV

CONECTIVIDAD A BASE DE DATOS DESDE JAVA *83*

4.1. JDBC *83*

4.1.1. Fundamentos de los Drivers JDBC 84

4.1.2. Registrar un Driver JDBC 85

4.1.3. URLs de Drivers JDBC 88
4.1.3.1. Drivers del Tipo 1 90
4.1.3.1.1. Codificación para Drivers del Tipo 1 90
4.1.3.2. Drivers del Tipo 2 92
4.1.3.3. Drivers del Tipo 3 93
4.1.3.4. Drivers del Tipo 4 94

CAPITULO V

ANALISIS DE PRUEBAS Y RESULTADOS *98*

5.1. Ejecución de la aplicación Servidor *99*

5.2. Ejecución en teléfono Motorola A1200 y Nokia 5310 ....*102*

5.3. Análisis de resultados....*106*

Índice de Ilustraciones

Capítulo I

Marco Referencial

Capítulo II

Bluetooth y Java

Figura II.1. Módulos bluetooth en USB y tarjetas SD *18*

Figura II.2. Datagrama Bluetooth *21*

Figura II.3. En la piconet, los esclavos solo pueden comunicarse con el dispositivo maestro *24*

Figura II.4. Representación del funcionamiento de una Scatternet *27*

Capítulo III

J2ME, J2SE y Bluetooth

Figura III.1. Estructura básica de la arquitectura del sistema *49*

Figura III.2. Configuración Java para dispositivos con recursos limitados *51*

Figura III.3. Etapas para la creación de un MIDlet *52*

Figura III.4. Pantalla inicial en el emulador antes de cargar el MIDlet *58*

Figura III.5. Resultado de la aplicación *59*

Figura III.6. Ciclo de vida del MIDlet representado en un diagrama de estados *60*

Figura III.7. Ejemplo de cómo interactúa el AMS con los estados de un MIDlet llamado DateTimeApp *62*

Figura III.8. Colaboración entre la implementación y la aplicación servidora para el registro del servicio *71*

Figura III.9. Forma como se integra Bluecove a la JVM *81*

Figura III.10. Librerías Bluecove y JDBC de MySQL integrados al proyecto *82*

CAPITULO IV

Conectividad a Base de Datos desde Java

Figura IV.1. Representación drivers tipo 1 .... *90*

Figura IV.1. Representación drivers tipo 2 .... *92*

Figura IV.1. Representación drivers tipo 3 .... *93*

Figura IV.1. Representación drivers tipo 4 .... *94*

CAPITULO V

Análisis de Pruebas y Resultados

Figura V.1. Esquema general del proyecto *98*

Figura V.2. Esquema detallado del proyecto *99*

Figura V.3. Pantalla de inicio del programa servidor *100*

Figura V.4. Aplicación en modo de espera de conexiones bluetooth *101*

Figura V.5. Momento en que se da la transmisión/recepción de datos *101*

Figura V.6. Base de datos de los usuarios bluetooth conectados, consola de MySQL Query Browser *102*

Figura V.7. Icono de la aplicación en la pantalla de menú principal del teléfono *103*

Figura V.8. Pantalla de bienvenida *103*

Figura V.9. Buscando servidores bluetooth con Protocolo de Puerto Serie (SPP) *104*

Figura V.10. Lista de dispositivos encontrados por la aplicación cliente *104*

Figura V.11. Proceso de intercambio de datos entre el servidor y el cliente *105*

Figura V.12. Finalización del intercambio de datos y cierre de la conexión *105*

Figura V.13. Visualizacion de la base de datos del servidor *106*

Introducción

Bluetooth es la norma que define un estándar global de comunicación inalámbrica, que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia., con lo que se pretende facilitar las comunicaciones entre equipos móviles y fijos, eliminar cables y conectores entre éstos, ofreciendo la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales

La tecnología Bluetooth comprende hardware, software y requerimientos de interoperabilidad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones y la informática, tales como: Ericsson, Nokia, Toshiba, IBM, Intel y otros. Posteriormente se han ido incorporando muchas más compañías, y se prevé que los hagan también empresas de sectores tan variados como: automatización industrial, maquinaria, ocio y entretenimiento, fabricantes de juguetes, electrodomésticos, etc., con lo que en pocos años se nos presentará un panorama de total conectividad de nuestros aparatos tanto en casa como en el trabajo.

Ahora bien, si unimos uno de los sistemas de comunicación inalámbrica más baratos con uno de los lenguajes más potentes, versátiles y por sobre todo gratuito del mercado como lo es Java, obtendremos un sin número de aplicaciones que sin duda simplificarán nuestras tareas en muchos aspectos de nuestra vida. Por ejemplo, controlar desde

nuestro teléfono las luces de nuestra casa, abrir la puerta de entrada, pagar una tarifa transporte sin necesidad de llevar dinero, verificar quien está en nuestra puerta desde la pantalla de nuestro móvil, etc., hoy ya no son cosas inalcanzables.

Aquí podremos ver una ínfima parte de lo que estas dos tecnologías abarcan, dando a entre ver una de las tantas aplicaciones que se podría hacer. A continuación una breve descripción de cada capítulo:

Capítulo I. Justificación y objetivos propuestos para el proyecto de tesis.

Capítulo II. Descripción y funcionamiento de la tecnología de comunicación Bluetooth, y lenguaje de programación Java.

Capítulo III. Explicación más detallada de los elementos constituyentes del desarrollo de tesis con fragmentos del código de las aplicaciones como ejemplos.

Capítulo IV. Teoría e implementación para la conectividad con el gestor de base de datos MySQL.

Capítulo V. Entorno de ejecución del sistema y análisis de los resultados obtenidos.

CAPÍTULO I

MARCO REFERENCIAL

1.1. ANTECEDENTES

En la actualidad, los avances de la tecnología nos han deslumbrado con su impresionante desarrollo y la gran infinidad de lugares en donde se los aplica. Nos simplifican el trabajo en muchos aspectos de nuestra vida diaria y nos afianzan a una mejor calidad de vida.

Como en muchos lugares, el sistema de control de usuarios se lo realiza de manera manual o semiautomática, el hardware utilizado por lo general es relativamente

costoso y la instalación complicada de hacer, a mas de los inconvenientes al momento de su funcionamiento como el uso ineficiente del tiempo y las molestias que ocasionan los métodos actualmente utilizados para el control de los usuarios.

1.2. JUSTIFICACIÓN

De esta forma se propuso mediante el uso de una de las tecnologías de comunicación más esparcidas en nuestro medio como es el caso de los teléfonos móviles y el sistema de transmisión de datos “Bluetooth”, brindar un prototipo para el control de usuarios a un área definida en donde se prevea controlar el acceso y salida de personal.

El prototipo tuvo previsto realizar con un sistema de transmisión y recepción Bluetooth, con una interfaz desarrollada en este caso para teléfonos celulares en el lenguaje de programación JAVA. También se necesita la colaboración de una unidad de procesamiento y almacenamiento de datos que sea compatible con este sistema de comunicación. El prototipo tendrá las siguientes funcionalidades:

- Registro dinámico de usuarios que utilicen el sistema.
- Detección de usuarios en el sistema.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Diseñar e implementar un prototipo para el control de usuarios, basado en teléfonos móviles con Bluetooth.

1.3.2. OBJETIVOS ESPECIFICOS

- Estudiar el funcionamiento del protocolo de comunicaciones Bluetooth.
- Diseñar e implementar una base de datos para el registro de los usuarios.
- Implementar la interfaz hardware que permita la comunicación entre el dispositivo móvil y el servidor de datos.
- Diseñar e implementar el software de control para el sistema.
- Probar y testear la implementación del prototipo funcionando.

1.4. HIPÓTESIS

La implementación de este proyecto demostrará que se puede automatizar el sistema de control unificado de usuarios explotando tecnología de consumo masivo

CAPÍTULO II

BLUETOOTH Y JAVA

Como primer capítulo se expone el corazón de esta tesis investigativa, el modo como se desenvuelve y la evolución de uno de los sistemas de radiocomunicaciones más baratos y con un amplia gama de aplicaciones que ha hecho que la tecnología bluetooth se desarrolle y migre de una manera sorprendente a nuestro vivir diario junto con uno de los lenguajes que ha revolucionado el campo informático por su sencillez, accesibilidad y potencia, como lo es Java.

2.1. Bluetooth

Es la norma que define un estándar global de comunicación inalámbrica, que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretende conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

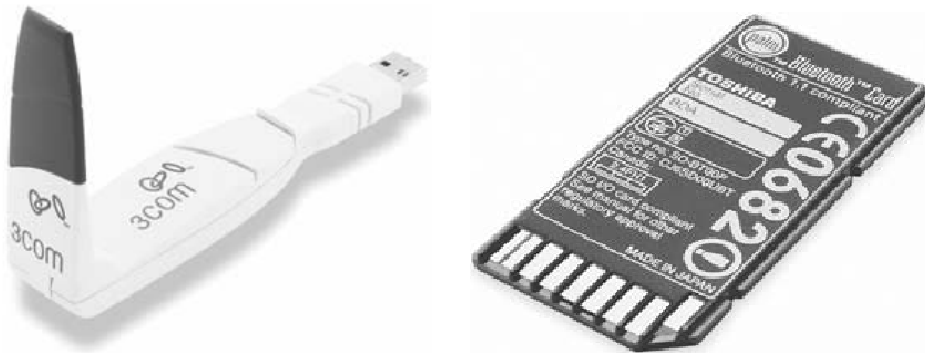


Figura II.1. Módulos bluetooth en USB y tarjetas SD, HOPKINS (1).

2.1.1. Banda de frecuencia libre

Para poder operar en todo el mundo es necesaria una banda de frecuencia abierta a cualquier sistema de radio independientemente del lugar del planeta donde nos

encontremos. Sólo la banda ISM de 2,45 Ghz cumple con éste requisito, con rangos que van de los 2.400 Mhz a los 2.500 Mhz, con algunas restricciones en países como Francia, España y Japón.

2.1.2. Salto de frecuencia

Debido a que la banda ISM está abierta a cualquiera, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que se puedan producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia. Los sistemas de radio Bluetooth utilizan el método de salto de frecuencia debido a que ésta tecnología puede ser integrada en equipos de baja potencia y bajo coste. Éste sistema divide la banda de frecuencia en varios canales de salto, donde, los transceptores, durante la conexión van cambiando de uno a otro de manera pseudo-aleatoria. Con esto se consigue que el ancho de banda instantáneo sea muy pequeño y también una propagación sobre el total de la banda. En conclusión, con el sistema FH, se pueden conseguir transceptores de banda estrecha con máxima inmunidad a las interferencias.

2.1.3. Definición de canal

Bluetooth utiliza un sistema FH/TDD, en el que el canal queda dividido en intervalos de 625 µs, llamados slots, donde cada salto de frecuencia es ocupado por un slot. Esto

da lugar a una frecuencia de salto de 1600 veces por segundo. Un paquete de datos puede ocupar un slot para la emisión y otro para la recepción y pueden ser usados alternativamente, dando lugar a un esquema de tipo TDD.

Dos o más unidades pueden compartir el mismo canal dentro de una piconet , donde una unidad actúa como maestra, controlando el tráfico y las demás como esclavas. El salto de frecuencia del canal es determinado por la secuencia, es decir, el orden en que llegan los saltos y por la fase de ésta secuencia. En Bluetooth, la secuencia está determinada por la identidad de la unidad maestra de la piconet (un código único para cada equipo, y por su frecuencia de reloj. Por lo que, para que una unidad esclava pueda sincronizarse con una unidad maestra, ésta primera debe añadir un ajuste a su propio reloj nativo y así poder compartir la misma portadora de salto.

En países donde la banda está abierta a 80 canales o más, espaciados todos ellos a 1 Mhz, se han definido 79 saltos de portadora, y en aquellos donde la banda es más estrecha se han definido 23 saltos.

2.1.4. Definición de paquete

En cada slot, un paquete de datos puede ser intercambiado entre la unidad maestra y una de las esclavas. 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 o no al de cabeza, puede tener una longitud de 0 a 2745 bits. Cada paquete que se intercambia en el canal es precedido por el código de acceso.

Figura II.2. Datagrama Bluetooth, BORCHES (2).

Los receptores de la piconet 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.

2.1.5. Definición de enlace físico

Se han definido dos tipos de enlace para poder soportar aplicaciones multimedia:

- Enlace de sincronización de conexión orientada (SCO)
- Enlace asíncrono de baja conexión (ACL)

Los enlaces SCO soportan conexiones asimétricas, punto a punto, usadas normalmente en conexiones de voz. Estos enlaces están definidos en el canal, reservándose dos slots consecutivos (envío y retorno) en intervalos fijos. Los enlaces

ACL soportan conmutaciones puntos a punto simétricos o asimétricos, típicamente usadas en la transmisión de datos.

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 voz se envía sin proteger, pero si el intervalo del enlace SCO disminuye, se puede seleccionar una velocidad de corrección de envío de 1/3 o 2/3.
- Para los enlaces 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.

2.1.6. Inmunidad a las interferencias

Como se mencionó anteriormente Bluetooth opera en una banda de frecuencia que está sujeta a considerables interferencias, por lo que el sistema ha sido optimizado para evitar éstas interferencias. Las técnicas de salto de frecuencia son aplicadas a una alta velocidad y una corta longitud de los paquetes (1600 saltos/segundo, para slots-simples), si un paquete se perdiese, sólo una pequeña parte de la información se perdería. Los paquetes de datos están protegidos por un esquema ARQ, en el cual los paquetes perdidos son automáticamente retransmitidos. La voz no se retransmite nunca, sin embargo, se utiliza un esquema de codificación muy robusto. Éste

esquema, que está basado en una modulación variable de declive delta (CSVD), sigue la forma de la onda de audio, y es muy resistente a los errores de bits. Éstos son percibidos como ruido de fondo, que se intensifica si los errores aumentan.

2.1.7. Red inalámbrica

2.1.7.1. Piconets

Si un equipo se encuentra dentro del radio de cobertura de otros, éstos pueden establecer conexión entre ellos. En principio sólo son necesarias un par de unidades con las mismas características de hardware. Dos o más unidades Bluetooth que comparten un mismo canal forman 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 pueden intercambiar los papeles entre ellos, si una unidad esclava quiere 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 un ajuste de reloj se añade al nativo para sincronizar el reloj del esclavo con el del maestro. El reloj nativo mantiene siempre constante su frecuencia, sin embargo los ajustes sólo son válidos mientras dura la conexión.

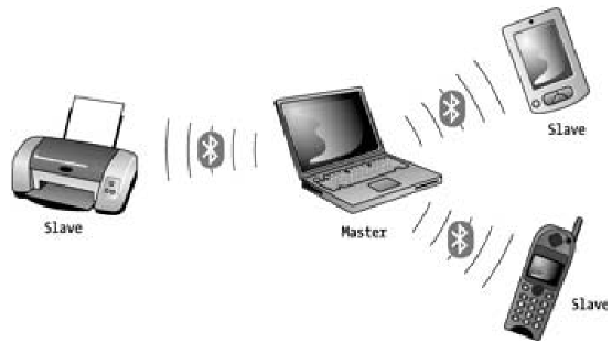


Figura II.3. En la piconet, los esclavos solo pueden comunicarse con el dispositivo maestro, HOPKINS (1).

Las unidades maestras controlan en tráfico del canal. Éstas tienen la capacidad para reservar slots en los enlaces SCO. Para los enlaces ACL, utilizan 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 en el procedimiento de slot maestro-esclavo. Éste tipo de slot implica un sondeo por parte del esclavo, por lo que, un tráfico normal de paquetes 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.

2.1.7.1.1. Estableciendo conexión

Cuando las unidades no están participando en la piconet, entran en modo standby (espera), desde el cual reciben periódicamente las páginas de mensajes. De un

conjunto total de 79 (23) portadoras del salto, un subconjunto de 32(16) portadoras activas han sido definidas. El subconjunto, que es seleccionado pseudo-aleatóriamente, se define por una única identidad.

Acerca de la secuencia de activación de las portadoras, se establece que, cada una de ellas visitará cada salto de portadora una sola vez, con una longitud de la secuencia de 32(16) saltos. En cada uno de los 2.048 (1.028) saltos, las unidades en standby mueven sus saltos de portadora siguiendo la secuencia de las unidades activas. El reloj de la unidad activa determina la secuencia de activación.

Durante la recepción de los intervalos, en los últimos 18 slots o 11,25 ms, las unidades escuchan una simple portadora de salto de activación y correlacionan las señales entrantes con el código de acceso derivado de su propia identidad. Si los triggers correlativos, esto es, si la mayoría de los bits recibidos coinciden con el código de acceso, la unidad se auto-activa e invoca un procedimiento de ajuste de conexión. Sin embargo la unidad vuelve al estado de reposo hasta el siguiente evento activo.

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., el tren de 10 ms. de códigos de acceso de diferentes saltos de portadora se transmite repetidamente hasta que el receptor o 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. De éste modo se establece una piconet en la que la unidad emisora actúa como maestra. 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 comunicarse.

El número máximo de unidades que pueden participar activamente en una simple piconet es de 8, un maestro y siete esclavos. La dirección MAC del paquete de cabecera que se utiliza para distinguir cada unidad se limita a tres bits.

2.1.7.2. Scatternet

Los equipos que comparten un mismo canal sólo pueden utilizar una parte de su capacidad. Aunque los canales tienen un ancho de 1Mhz, cuantos más usuarios se incorporan a la piconet, disminuye la capacidad hasta unos 10 kbit/s más o menos. Teniendo en cuenta que el ancho de banda medio disponible es de unos 80 Mhz en Europa y USA (excepto en España y Francia), éste no puede ser utilizado eficazmente, cuando cada unidad ocupa una parte del mismo canal de salto de 1Mhz. Para poder

solucionar éste problema se adoptó una solución de la que nace el concepto de scatternet.

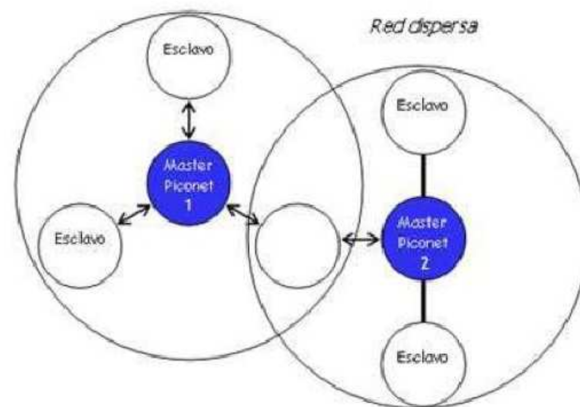


Figura II.4. Representación del funcionamiento de una Scatternet, BORCHES (2).

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. 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 canales salto. Debido a que individualmente cada piconet tiene un salto de frecuencia diferente, diferentes piconets pueden usar simultáneamente diferentes canales de salto.

Hemos de tener en cuenta que cuantas más piconets se añaden a la scatternet el rendimiento del sistema FH disminuye poco a poco, habiendo una reducción por término medio del 10%. Sin embargo el rendimiento que finalmente se obtiene de múltiples piconets supera al de una simple piconet.

2.1.7.3. Comunicación inter-piconet

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.

La sincronización de varias piconets no está permitida en la banda ISM. Sin embargo, las unidades pueden participar en diferentes piconets en base a un sistema TDM. Esto es, una unidad participa secuencialmente en diferentes piconets, a condición de que ésta este sólo activa en una al mismo tiempo. Una unidad al incorporarse a una nueva piconet modifica el offset (ajuste interno) de su reloj para minimizar la deriva entre el reloj del maestro y el de su propio reloj nativo, por lo que gracias a éste sistema puede participar en varias piconets realizando los ajustes correspondientes una vez conocidos los diferentes parámetros de la piconet. Cuando se deja una piconet por otra, una esclava informa el maestro actual que ésta no estará disponible por un

determinado periodo. Durante su ausencia el tráfico en la piconet entre el maestro y otros esclavos continúa igualmente.

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.

2.1.8. Seguridad

Para asegurar la protección de la información se ha definido un nivel básico de encriptación, que se ha incluido en el diseño del clip de radio para proveer de seguridad en equipos que carezcan de capacidad de procesamiento, las principales medidas de seguridad son:

- Una rutina de pregunta-respuesta, para autentificación
- Una corriente cifrada de datos, para encriptación
- Generación de una clave de sesión (que puede ser cambiada durante la conexión)

Tres entidades son utilizadas en los algoritmos de seguridad: la dirección de la unidad Bluetooth, que es una entidad pública; una clave de usuario privada, como una entidad secreta; y un número aleatorio, que es diferente por cada nueva transacción.

Como se ha descrito anteriormente, la dirección Bluetooth se puede obtener a través de un procedimiento de consulta. La clave privada se deriva durante la inicialización y nunca es revelada. El número aleatorio se deriva de un proceso pseudo-aleatorio en la unidad Bluetooth.

2.2. JAVA

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 y casi todo el trabajo de los programadores se dedicó al desarrollo de éstos. Con los años, Java ha progresado enormemente en varios ámbitos como servicios HTTP, servidores de aplicaciones, acceso a bases de datos (JDBC)... Como vemos Java se ha ido adaptando a las necesidades tanto de los usuarios como de las empresas ofreciendo soluciones y servicios tanto a unos como a otros. Debido a la explosión tecnológica de estos últimos años Java ha desarrollado soluciones personalizadas para cada ámbito tecnológico. Sun ha agrupado cada uno de esos ámbitos en una edición distinta de su lenguaje Java. Estas ediciones son Java 2 Stantard Edition, orientada al desarrollo de aplicaciones independientes y de

applets, Java 2 Enterprise Edition, enfocada al entorno empresarial y Java 2 Micro Edition, orientada a la programación de aplicaciones para pequeños dispositivos.

2.2.1. Filosofía

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

2.2.2. Orientado a Objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es

separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de “código abierto” (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

2.2.3. Independencia de la plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, ‘’’write once, run everywhere’’’.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode) —instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para

acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT.

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT, convierte el bytecode a código nativo

cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una recompilación dinámica en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

2.2.4. Sintaxis

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase (recordemos que una clase es un molde a partir del cual pueden crearse varios objetos).

2.2.4.1. Aplicaciones autónomas “Hola Mundo”

```
import java.io*;
public class Hola
{
   public static void main(String[] args)throws IOException {
      System.out.println("¡Hola, mundo!");
   } }
```

El ejemplo anterior necesita una pequeña explicación:

- Todo en Java está dentro de una clase, incluyendo programas autónomos.
- El código fuente se guarda en archivos con el mismo nombre que la clase que contienen y con extensión ".java". Una clase (class) declarada pública (public) debe seguir este convenio. En el ejemplo anterior, la clase es Hola, por lo que el código fuente debe guardarse en el fichero "Hola.java"
- El compilador genera un archivo de clase (con extensión ".class") por cada una de las clases definidas en el archivo fuente. Una clase anónima se trata como si su nombre fuera la concatenación del nombre de la clase que la encierra, el símbolo "$", y un número entero.
- Los programas que se ejecutan de forma independiente y autónoma, deben contener el método "main()".
- La palabra reservada "void" indica que el método main no devuelve nada.
- El método main debe aceptar un array de objetos tipo String. Por acuerdo se referencia como "args", aunque puede emplearse cualquier otro identificador.
- La palabra reservada "static" indica que el método es un método de clase, asociado a la clase en vez de una instancia de la misma. El método main debe ser estático o "de clase''.
- La palabra reservada public significa que un método puede ser llamado desde otras clases, o que la clase puede ser usada por clases fuera de la

jerarquía de la propia clase. Otros tipos de acceso son "private" o "protected".

- La utilidad de impresión (en pantalla por ejemplo) forma parte de la biblioteca estándar de Java: la clase "System" define un campo público estático llamado "out". El objeto out es una instancia de "PrintStream", que ofrece el método "println (String)" para volcar datos en la pantalla (la salida estándar).
- Las aplicaciones autónomas se ejecutan dando al entorno de ejecución de Java el nombre de la clase cuyo método main debe invocarse. Por ejemplo, una línea de comando (en Unix o Windows) de la forma java –cp. Hola ejecutará el programa del ejemplo (previamente compilado y generado "Hola.class"). El nombre de la clase cuyo método main se llama puede especificarse también en el fichero "MANIFEST" del archivo de empaquetamiento de Java (.jar).

2.2.4.2. Aplicaciones con ventanas

Swing es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.

```
// Hola.java
import javax.swing.*;

public class Hola extends JFrame {
   Hola() {
      setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
      add(new JLabel("Hola, mundo!"));
      pack();
   }
    public static void main(String[] args) {
      new Hola().setVisible(true);
   }
}
```

Las instrucciones import indican al compilador de Java que las clases e interfaces del paquete javax.swing se incluyan en la compilación.

La clase Hola extiende (extends) la clase javax.swing.JFrame, que implementa una ventana con una barra de título y un control para cerrarla.

El constructor Hola() inicializa el marco o frame llamando al método setDefaultCloseOperation(int) heredado de JFrame para establecer las operaciones por defecto cuando el control de cierre en la barra de título es seleccionado al valor WindowConstants.DISPOSE_ON_CLOSE. Esto hace que se liberen los recursos tomados por la ventana cuando es cerrada, y no simplemente ocultada, lo que permite a la máquina virtual y al programa acabar su ejecución. A continuación se crea un objeto de tipo JLabel con el texto "Hola, mundo!", y se añade al marco mediante el método add (Component), heredado de la clase Container. El método

pack(), heredado de la clase Window, es invocado para dimensionar la ventana y distribuir su contenido.

El método main() es llamado por la JVM al comienzo del programa. Crea una instancia de la clase Hola y hace la ventana sea mostrada invocando al método setVisible (boolean) de la superclase (clase de la que hereda) con el parámetro a true. Véase que, una vez el marco es dibujado, el programa no termina cuando se sale del método main(), ya que el código del que depende se encuentra en un hilo de ejecución independiente ya lanzado, y que permanecerá activo hasta que todas las ventanas hayan sido destruidas.

2.2.5. Entornos de funcionamiento

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

2.2.5.1. En dispositivos móviles y sistemas empotrados

Desde la creación de la especificación J2ME, una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores específicamente diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos.

El modelo de desarrollo de estas aplicaciones es muy semejante a las *applets* de los navegadores salvo que en este caso se denominan *MIDlets*.

Más adelante hablaremos con mayor profundidad acerca de J2ME, en un capitulo que también tratara su uso con bluetooth y J2SE.

2.2.5.2. En el navegador web

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (*plug-in*) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

2.2.5.3. En sistemas de servidor

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP.

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica).

Los servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es muy sencilla, flexible y extensible.
- Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.
- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

2.2.5.4. En aplicaciones de escritorio

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

2.2.6. Plataformas soportadas

Una versión del entorno de ejecución Java JRE está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de GNU/Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

2.2.7. Industria relacionada

Sun Microsystem, como creador del lenguaje de programación Java y de la plataforma JDK, mantiene fuertes políticas para mantener una especificación del lenguaje así como de la máquina virtual a través del JCP. Es debido a este esfuerzo que se mantiene un estándar de facto.

Son innumerables las compañías que desarrollan aplicaciones para Java y/o están volcadas con esta tecnología:

- La industria de la telefonía móvil está fuertemente influenciada por la tecnología Java.
- El entorno de desarrollo Eclipse ha tomado un lugar importante entre la comunidad de desarrolladores Java.
- La fundación Apache tiene también una presencia importante en el desarrollo de bibliotecas y componentes de servidor basados en Java.
- IBM, BEA, IONA, Oracle,... son empresas con grandes intereses y productos creados en y para Java.

2.2.8. El lenguaje

- En un sentido estricto, Java no es un lenguaje absolutamente orientado a objetos, a diferencia de, por ejemplo, Ruby o Smalltalk. Por motivos de

eficiencia, Java ha relajado en cierta medida el paradigma de orientación a objetos, y así por ejemplo, no todos los valores son objetos.

- El código Java puede ser a veces redundante en comparación con otros lenguajes. Esto es en parte debido a las frecuentes declaraciones de tipos y conversiones de tipo manual (casting). También se debe a que no se dispone de operadores sobrecargados, y a una sintaxis relativamente simple. Sin embargo, J2SE 5.0 introduce elementos para tratar de reducir la redundancia, como una nueva construcción para los bucles '''foreach'''.
- A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener.

2.2.9. Recursos

2.2.9.1. JRE

El **JRE** es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para

ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

2.2.9.2. Componentes

- Bibliotecas de Java, que son el resultado de compilar el código fuente desarrollado por quien implementa la JRE, y que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas bibliotecas son:
    - Las bibliotecas centrales, que incluyen:
        - Una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos.
        - Bibliotecas para análisis de XML.
        - Seguridad.
        - Bibliotecas de internacionalización y localización.
    - Bibliotecas de integración, que permiten la comunicación con sistemas externos. Estas bibliotecas incluyen:
        - La API para acceso a bases de datos JDBC.
        - La interfaz JNDI para servicios de directorio.
        - RMI y CORBA para el desarrollo de aplicaciones distribuidas.
    - Bibliotecas para la interfaz de usuario, que incluyen:

  - El conjunto de herramientas nativas AWT, que ofrece componentes GUI, mecanismos para usarlos y manejar sus eventos asociados.
  - Las Bibliotecas de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT.
  - APIs para la captura, procesamiento y reproducción de audio.
- Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java, que es la encargada de la ejecución del código de las bibliotecas y las aplicaciones externas.
- Plugins o conectores que permiten ejecutar applets en los navegadores Web.
- Java Web Start, para la distribución de aplicaciones Java a través de Internet.
- Documentación y licencia.

2.2.10. APIs

Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus APIs de forma que pertenezcan a cada una de las plataformas:

- Java ME o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs, etc.
- Java SE o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.

- Java EE o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados paquetes. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP. Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

CAPÍTULO III

J2ME, J2SE y BLUETOOTH

En función de la actual proliferación de dispositivos móviles tanto con soporte Bluetooth como de J2ME, se abre un sin fin de aplicaciones posibles que permiten otorgar diversos servicios en todas las áreas comerciales existentes. Esto se potencia más aún, con las tecnologías incluidas con los dispositivos móviles, tales como cámaras de video e imagen de alta resolución, APIs que soportan sistema de pagos, etc.

Ahora con Bluetooth y J2ME es posible diseñar programas que interactúen con el usuario en diversas situaciones, tales como sinopsis de películas en cine, promociones y

descuentos en tiendas, procesamiento digital de imágenes para telemedicina, diagnóstico multidisciplinario, compartir datos en reuniones de trabajo, sistema de pagos automáticos de servicios, etc., las posibilidades son muy amplias.

De tal manera, se convierte primordial el estudiar la programación de celulares con J2ME y estudiar el estándar Bluetooth y el API JSR-82 que da soporte para J2SE y J2ME, además dar a conocer las herramientas de desarrollo que existen para crear programas con estas tecnologías.

Por último, se capitalizará el conocimiento adquirido creando una aplicación cliente-servidor (El cliente será el celular, el servidor será el PC) que permita mostrar las potencialidades de Bluetooth. Esta aplicación constará de dos partes. Primero, se creará una aplicación en el servidor que permita la disponibilidad del servicio requerido para establecer conexión serial con las aplicaciones clientes de los teléfonos celulares y segundo, se diseñará dicha aplicación cliente que permita responder a las peticiones del servidor.

Figura III. 1. Estructura básica de la arquitectura del sistema, APORTE A JAVA (16).

3.1. J2ME

Java 2 Micro Edition combina una Maquina Virtual de Java restringida y un conjunto de APIs de Java que permiten desarrollar aplicaciones para tecnología móvil. En este documento se pretende explicar la creación de la aplicación J2ME en el teléfono celular y las librerías utilizadas para la conexión a través de la tecnología de comunicación inalámbrica de corto alcance Bluetooth con la aplicación servidor escrita en la plataforma J2SE.

J2ME no es más que otro conjunto de APIs como J2SE, pero en una versión más reducida, dado que estas APIs no pueden correr en una JVM tradicional, debido al tamaño limitado de los dispositivos móviles en lo que respecta a la memoria y la disponibilidad de recursos, para este propósito J2ME define una versión limitada de la Máquina virtual, los fabricadores de los dispositivos instalan y pre empaquetan en sus dispositivos a la JVM (y las APIs).

J2ME puede ser dividido en 3 partes (como se muestra en la figura), configuración, perfil y paquetes opcionales. La configuración contiene a la JVM (no la JVM tradicional, sino la versión reducida de ésta) y algunas librerías de clases; el perfil construido sobre las librerías base otorgando un conjunto de APIs; por último, los paquetes opcionales que son APIs que se pueden usar o no para crear aplicaciones. Los paquetes opcionales tradicionalmente no son empaquetados por el fabricador de los dispositivos, por lo tanto se tiene que empaquetar y distribuirlos con la aplicación.

La configuración y el perfil son suplidos por el manufacturador y son embebidos en el dispositivo.

Figura III.2 Configuración Java para dispositivos con recursos limitados, BLUETOOTH Y J2ME (18).

El perfil y configuración más popular que Sun provee es el perfil de dispositivos de información móvil (MIDP) y la configuración de dispositivos limitados en conexión (CLDC), respectivamente. CLDC es para dispositivos que tengan sólo 128 a 1MB de memoria disponible para aplicaciones Java. En ese sentido, la JVM que provee es muy limitada y soporta un número pequeño de las clases de Java tradicionales (lang, io y util). (Esta JVM se llama KVM, k-kilobytes). Su contrapartida, la configuración de dispositivos conectados (CDC) es para dispositivos con al menos 2MB de memoria disponible y soporta un JVM con más clases (pero todavía no es la JVM estándar).

El perfil MIDP complementa al CLDC muy bien porque minimiza tanto el uso de la memoria como el de la energía requerida por los dispositivos. Provee la API básica que es usada para crear aplicaciones en estos dispositivos. Por ejemplo, provee javax.microedition.lcdui que permite crear elementos de la GUI que pueden ser mostrados en un dispositivo (limitado) corriendo el perfil MIDP sobre una

configuración CLDC. Notar que la MIDP no puede ser usada con dispositivos CDC. Los dispositivos CDC obtienen su propio conjunto de perfiles, como el perfil personal y fundación.Las últimas versiones de MIDP y CLDC son 2.0 y 1.1, respectivamente.

3.1.1. Creación de MIDlet

Son 7 los pasos involucrados en la creación de un MIDlet. Estos pasos son: diseño, código, compilar, preverificar, empaquetar, probar y desplegar. Alguno de estos pasos no son obligatorios (pero cada aplicación necesita ser diseñada, codificada y compilada). WTK permite abstraerse de alguno de estos pasos para que sea más sencillo su manejo.

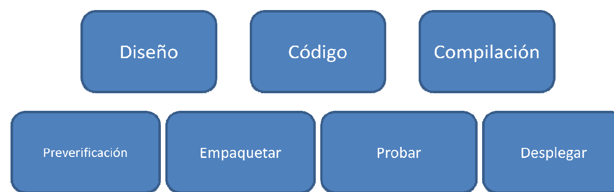


Figura III.3. Etapas para la creación de un MIDlet, BLUETOOTH Y J2ME (18).

3.1.1.1. Diseño

Los MIDlets son diferentes de otras aplicaciones que se podrían haber creado hasta ahora, dado que los MIDlet corren en un ambiente muy diferente a los usuales. Por ejemplo, la interactividad de su MIDlet con el usuario.

Para el ejemplo, el MIDlet no requiere de interactividad con el usuario. Necesita mostrar cierta información correspondiente al teléfono como por ejemplo la hora, tipo de pantalla y disponibilidad de memoria. Para casos simples como éste, es suficiente para esta etapa especificar en un papel el diseño del MIDlet. Para diseños más complejos con múltiples pantallas, sería mejor diseñar las pantallas profesionalmente antes de comenzar a crear el código para el proceso.

3.1.1.2. Código

Cada MIDlet debe extender la clase abstracta MIDlet encontrada en javax.microedition.midlet, parecido a como se crea un applet extendiendo la clase java.applet.Applet. Como mínimo, el MIDlet debe sobrescribir 3 métodos de la clase abstracta, starApp(), pauseApp() y destroyApp(boolean unconditional).

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;
public class Midlet extends MIDlet implements CommandListener{
private Command salir;
private Display display;
private Form pantalla;

public Midlet() {
// Recuperamos el objeto Display
display = Display.getDisplay(this);
// Creamos el comando de salida
salir = new Command("Salir", Command.EXIT, 2);
// Creamos el "Form" de la pantalla principal
pantalla = new Form("InfoDispositivo");
```

```
// Obtenemos la hora
Calendar calendar = Calendar.getInstance();
String hora = Integer.toString(calendar.get(Calendar.HOUR_OF_DAY)) + ":"
+
Integer.toString(calendar.get(Calendar.MINUTE)) + ":" +
Integer.toString(calendar.get(Calendar.SECOND));
// Obtenemos la memoria total y la libre
Runtime runtime = Runtime.getRuntime();
String memTotal = Long.toString(runtime.totalMemory());
String memLibre = Long.toString(runtime.freeMemory());
// Obtenemos las propiedades de la pantalla
String color = display.isColor() ? "Sí" : "No";
String numColores = Integer.toString(display.numColors());
// Creamos las cadenas de información y las añadimos a la pantalla
pantalla.append(new StringItem("", "Hora: " + hora + "\n"));
pantalla.append(new StringItem("", "Mem Total: " + memTotal + " b\n"));
pantalla.append(new StringItem("", "Mem Libre: " + memLibre + " b\n"));
pantalla.append(new StringItem("", "Color: " + color + "\n"));
pantalla.append(new StringItem("", "Colores: " + numColores + "\n"));
// Establecemos el comando de salida
pantalla.addCommand(salir);
pantalla.setCommandListener(this);
}
public void startApp() throws MIDletStateChangeException{
// Establecemos el "Display" actual a nuestra pantalla
display.setCurrent(pantalla);
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
  public void commandAction(Command c, Displayable d) {
    if (c == salir){
destroyApp(false);
notifyDestroyed();
    }
} }
```

En este ejemplo, el constructor MIDlet crea los elementos necesarios para mostrar toda la información necesaria en la pantalla del dispositivo y el método starApp() efectúa la tarea de mostrar los elementos.

3.1.1.3. Compilar

Ahora se necesita compilar el código para que esté listo para incluirlo en el dispositivo móvil. Compilar el MIDlet no es muy diferente de como se compilan comúnmente las aplicaciones Java. Todavía se usa javac como compilador, pero se necesita cambiar el CLASSPATH mientras se compila el MIDlet. Esto tiene como efecto cambiar la base de las clases de Java que el compilador usa en el MIDlet, asegurándose que la compilación se efectué con las APIs de la plataforma J2ME. En vez de compilar en java.util del Java "normal", se quiere que la compilación sea con java.util de J2ME. Esto es hecho apuntando al CLDC y las clases MIDP con la opción de javac -bootclasspath mientras se compila. Por ejemplo:

C:\WTK22\article>javac–bootclasspath ..\lib\cldcapi11.jar;

..\lib\midpapi20.jar com\j2me\part1\MIDlet.java

Gracias al desarrollo de Ambientes de Desarrollo Integrado (IDE), esta tarea se nos facilita enormemente, evitándonos la molestia de escribir comandos al sistema.

Hay varios entornos de desarrollo, desde los que son de paga hasta los gratuitos. Durante el desarrollo de esta tesis, se utilizo en un principio el IDE Eclipse, pero

debido a la difícil integración de las librerías J2ME y el emulador apropiado para simular los MIDlets, se tuvo que descartar, optando por otro IDE mucho más completo y potente como es Netbeans 6.5 programado por Sun, la misma empresa desarrolladora de Java.

3.1.1.4. Preverificado

Antes de desplegar la clase MIDlet, se necesita preverificar. La verificación del bytecode se realiza por la JVM antes de correr cualquier archivo clase con el fin de asegurarse que el archivo esta estructural y conceptualmente correcto tal como corresponde con la especificación de la JVM. Si este proceso falla, se rechaza y la JVM se apaga, indicando ya sea violación de seguridad o integridad del archivo clase. Esta verificación es hecha por todas las JVM, incluso por la pequeña JVM contenida en la configuración CLDC de un dispositivo J2ME. Esto no es un problema para las aplicaciones normales, la verificación en dispositivos J2ME por el grado de memoria y recursos utilizados resulta prohibitivo. Por esto, se necesita la preverificación.

La preverificación es un proceso de dos pasos, especialmente diseñado para dispositivos restringidos, tales como los que corren JVM basados en CLDC. La idea es dejar al desarrollador preverificar sus clases, lo que limita la cantidad de trabajo necesitada a ser realizada cuando las clases son verificadas en el dispositivo. Esta preverificación agrega información especial a las clases que las identifica como preverificadas y hace el proceso en el dispositivo mucho más eficiente. Para el ejemplo:

C:\WTK22\article>..\bin\preverify.exe –classpath ..\lib\cldcapi11.jar;..\lib\midpapi20.jar com.j2me.part1.MIDlet

3.1.1.5. Paquete

Empaquetar el MIDlet tal que esté listo para ser probado y desplegado es un proceso que involucra unos cuantos pasos. Los cuales se deben seguir en una secuencia apropiada.

El primer paso es crear un archivo Manifest. Este archivo Manifest describe el contenido del Archivo Java (JAR) que será creado en el siguiente paso. Existen varios atributos que pueden ir en este archivo, para el ejemplo propuesto es uno sencillo como el que se muestra a continuación:

MIDlet-Name:MIDlet
MIDlet-Version: 1.0.0
MIDlet-Vendor: Vendor

Guarda este archivo como *Manifest.mf* .

Para crear el JAR:

C:\WTK22\article\output>jar cvfm DateTimeApp.jar Manifest.mf .\com

El último paso es crear un archivo que tiene extensión .jad. Un descriptor de aplicación Java (JAD) apunta a la ubicación del MIDlet con el propósito de que un dispositivo J2ME pueda instalarlo. Este archivo puede contener varios atributos para un sólo MIDlet (o para varios MIDlets), para el ejemplo:

MIDlet-1: Midlet, , Midlet
MIDlet-Jar-Size: 2017
MIDlet-Jar-URL: info.jar
MIDlet-Name: info
MIDlet-Vendor: Vendor
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0

Al utilizar NetBeans 6.5, todos este proceso se facilita en muy pocos pasos, para crear el archivo .jar y .jad, debemos dar click derecho sobre el proyecto y marcar la opción de “Construir” o “Build”, creando los respectivos archivos .jar y .jad dentro de la carpeta “dist” en la carpeta del proyecto.

3.1.1.6. Prueba

Antes de desplegar el MIDlet, debe ser probado usando un emulador de dispositivo. Este emulador es parte de NetBeans 6.5 y provee funcionalidades que seguro estarán presentes en la mayor parte de los dispositivos a los cuales los MIDlet serán cargados.

Figura III.4. Pantalla inicial en el emulador antes de cargar el MIDlet, Wireless Toolkit 2.5.2

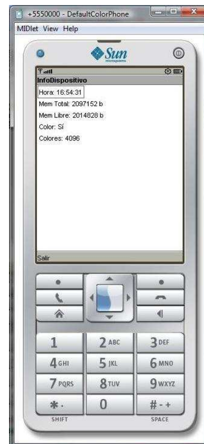


Figura III.5. Resultado de la aplicación, Wireless Toolkit 2.5.2

3.1.1.7. Desplegar

Por ejemplo a través de USB.

3.1.2. Ciclo de Vida del MIDlet

Los dispositivos móviles, sea emulador o real, interactúan con un MIDlet usando su propio software, el cual es llamado Software de Administración de Aplicación (AMS). La AMS es responsable por inicializar, comenzar, pausar, resumir y destruir un MIDlet. (Además de estos servicios, el AMS puede ser responsable de instalar y remover un MIDlet). Para facilitar esta administración, un MIDlet puede verse como un árbol de estados que es controlado a través de los métodos de la clase MIDlet, y todo MIDlet que la extiende y sobrescribe. Estos estados, son activo, pausa y destruye.

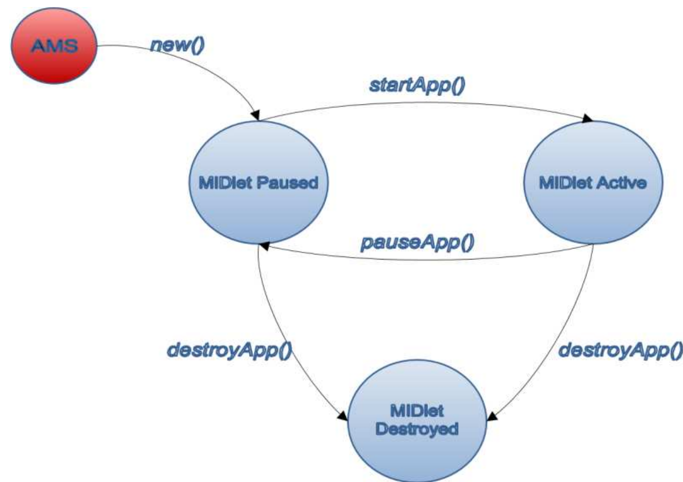


Figura III.6. Ciclo de vida del MIDlet representado en un diagrama de estados, BLUETOOTH Y J2ME (18).

Un MIDlet instalado es puesto en pausa por el AMS creando una instancia de él, llamando a un constructor sin argumentos. Este no es el único medio en que puede ser pausado. Puede entrar en este estado cuando el AMS invoca el método pauseApp() sobre un MIDlet activo. El MIDlet puede provocar el ingreso en este estado invocando el método notifyPaused(), en contrario a pauseApp() que es llamado por el AMS.

En un estado de pausa, el MIDlet está esperando por entrar en estado activo. Teóricamente, en este estado, no se debería estar usando ninguno de los recursos del dispositivo. Una vez el MIDlet es creado, este es el estado antes de pasar a activo. También se entra al estado pausa cuando el dispositivo requiere consumir pocos recursos, porque estos recursos podrían ser requeridos para mantener otras funciones del dispositivo, como manejar una llamada entrante. En estos casos es cuando los dispositivos invocan al método pauseApp() a través del AMS.

Otra forma en que el MIDlet puede entrar en estado pausado es con el método startApp() del MIDlet, el cual es llamado cuando el AMS invoca a iniciar el MIDlet (ya sea para la primera vez o desde el estado de pausa), tira un MIDletStateChangeException. Esencialmente, en caso de un error, el MIDlet toma el camino seguro de establecer un estado de pausa.

El estado activo es donde cada MIDlet quiere estar. Esto es cuando el MIDlet puede hacer sus funciones, mantener los recursos del dispositivo y generalmente, hacer lo que supuestamente hace. Como se dijo previamente, un MIDlet está en un estado activo cuando el AMS llama al método starApp() en un MIDlet pausado. Un MIDlet pausado puede pedir ir al estado activo llamando al método resumeRequest(), el cual informa al AMS que el MIDlet desea estar activo. El AMS podría por supuesto, elegir ignorar esta petición o, alternativamente, encolarla si hay otros MIDlet solicitando lo mismo.

El estado destruido se entra cuando el método destroyApp(boolean unconditional) del MIDlet es llamado y retorna exitosamente, ya sea desde un estado activo o pausado. Este método es llamado por el AMS cuando siente que no hay necesidad de seguir corriendo el MIDlet y es tiempo de que el MIDlet pueda realizar limpieza y otras actividades de último minuto. El MIDlet puede entrar en este estado, llamando al método notifyDestroy(), el cual informa al AMS que el MIDlet ha limpiado sus recursos y está listo para su destrucción. Por supuesto, en este caso, el método

destroyApp(boolean unconditional) no es llamado por el AMS, cualquier actividad de último minuto debe ser hecha antes de que este método sea invocado.

¿Qué sucede si el AMS llama al método destroyApp(boolean unconditional) en medio de un importante paso que el MIDlet esté haciendo? El flag unconditional si es seteado a true, el MIDlet será destruido, sin importar que esté haciendo el MIDlet. Sin embargo, si es falso, efectivamente, el AMS generara una excepción MIDletStateChangeException, y el AMS no lo destruirá todavía. Sin embargo, no hay garantías de que el MIDlet no será destruido, y cada dispositivo decide como manejarlo.

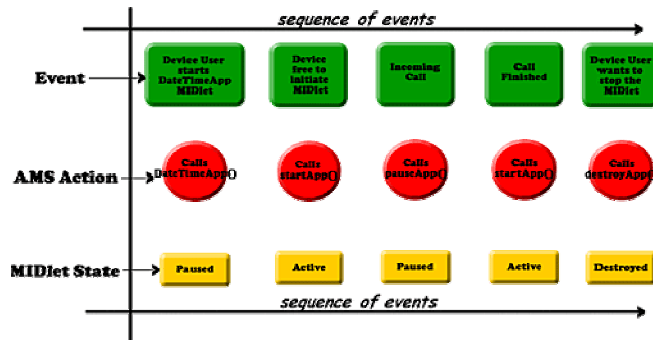


Figura III.7. Ejemplo de cómo interactúa el AMS con los estados de un MIDlet llamado DateTimeApp, BLUETOOTH Y J2ME (18).

Como otras APIS que extienden a J2ME tenemos la JSR 82 para Bluetooth. El JSR-82 especifica un API de alto nivel para la programación de dispositivos Bluetooth. Depende de la configuración CLDC de J2ME, y se divide en dos paquetes: javax.bluetooth y javax.obex. El primer paquete provee la funcionalidad para la realización de búsquedas de dispositivos, búsquedas de servicios y comunicación mediante flujos de datos (streams) o arrays de bytes. Por otro lado el paquete

javax.obex permite la comunicación mediante el protocolo OBEX (OBject Exchange); se trata de un protocolo de alto nivel muy similar a HTTP.

3.1.3. API JSR-82

Mientras que el hardware Bluetooth había avanzado mucho, hasta hace relativamente poco no había manera de desarrollar aplicaciones java Bluetooth, hasta que apareció JSR- 82, que estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java. Ésta esconde la complejidad del protocolo Bluetooth detrás de unos APIs que permiten centrarse en el desarrollo en vez de los detalles de bajo nivel del Bluetooth.

Estos APIs para Bluetooth están orientados para dispositivos que cumplan las siguientes características:

- Al menos 512K de memoria libre (ROM y RAM) (las aplicaciones necesitan memoria adicional).
- Conectividad a la red inalámbrica Bluetooth.
- Que tengan una implementación del J2ME CLDC.

El objetivo de ésta especificación era definir un API estándar abierto, no propietario que pudiera ser usado en todos los dispositivos que implementen J2ME. Por consiguiente fue diseñado usando los APIs J2ME y el entorno de trabajo CLDC/MIDP.

Los APIs JSR 82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth. El API intenta ofrecer las siguientes capacidades:

- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones RFCOMM, L2CAP y OBEX entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades. Los APIs Java para Bluetooth definen dos paquetes que dependen del paquete CLDC

Los APIs Java para Bluetooth definen dos paquetes que dependen del paquete CLDC javax.microedition.io:

- javax.bluetooth
- javax.obex

3.1.3.1. Programación de aplicaciones Bluetooth:

La anatomía de una aplicación Bluetooth está dividida en cuatro partes:

- Inicialización de la pila.
- Descubrimiento de dispositivos y servicios.
- Manejo del dispositivo.
- Comunicación.

3.1.3.1.1. Inicialización

3.1.3.1.1.1. BCC

Los dispositivos Bluetooth que implementen este API pueden permitir que múltiples aplicaciones se estén ejecutando concurrentemente. El BCC previene que una aplicación pueda perjudicar a otra. El BCC es un conjunto de capacidades que permiten al usuario o al OEM resolver peticiones conflictivas de aplicaciones definiendo unos valores específicos para ciertos parámetros de la pila Bluetooth.

El BCC puede ser una aplicación nativa, una aplicación en un API separado, o sencillamente un grupo de parámetros fijados por el proveedor que no pueden ser cambiados por el usuario. Hay que destacar, que el BCC no es una clase o un interfaz definido en esta especificación, pero es una parte importante de su arquitectura de seguridad.

3.1.3.1.1.2. Inicialización de la pila

La pila Bluetooth es la responsable de controlar el dispositivo Bluetooth, por lo que es necesario inicializarla antes de hacer cualquier otra cosa. El proceso de inicialización consiste en un número de pasos cuyo propósito es dejar el dispositivo listo para la comunicación inalámbrica.

Desafortunadamente, la especificación deja la implementación del BCC a los vendedores, y cada vendedor maneja la inicialización de una manera diferente. En un dispositivo puede haber una aplicación con un interfaz GUI, y en otra puede ser una

serie de configuraciones que no pueden ser cambiados por el usuario. Un ejemplo sería:

```
// Configuramos el puerto
BCC.setPortNumber("COM1");
// Configuramos la velocidad de la conexión
BCC.setBausRate(50000);
 //Configuramos el modo conectable
BCC.setConnectable(true);
//Configuramos el modo discovery a LIAC
BCC.setDiscoverable(DiscoveryAgent.LIAC); ...
```

3.1.3.1.2. Descubrimiento de dispositivos y servicios

Dado que los dispositivos inalámbricos son móviles, necesitan un mecanismo que permita encontrar, conectar, y obtener información sobre las características de dichos dispositivos. En esta sección veremos como el API de Bluetooth permite realizar estas tareas.

3.1.3.1.2.1. Descubrir Dispositivos (Device discovery)

Cualquier aplicación puede obtener una lista de dispositivos a los que es capaz de encontrar, usando, o bien startInquiry() (no bloqueante) o retrieveDevices() (bloqueante).startInquiry() requiere que la aplicación tenga especificado un listener, el cual es notificado cuando un nuevo dispositivo es encontrado después de haber lanzado un proceso de búsqueda. Por otra parte, si la aplicación no quiere esperar a descubrir dispositivos (o a ser descubierta por otro dispositivo) para comenzar, puede utilizar retrieveDevices(), que devuelve una lista de dispositivos encontrados en una búsqueda previa o bien unos que ya conozca por defecto.

3.1.3.1.2.1.1. Clases del Device Discovery

***interface javax.bluetooth.DiscoveryListener**

Esta interfaz permite a una aplicación especificar un evento en el listener que reaccione ante eventos de búsqueda. También se usa para encontrar dispositivos. El método deviceDiscovered () se llama cada vez que se encuentra un dispositivo en un proceso de búsqueda. Cuando el proceso de búsqueda se ha completado o cancelado, se llama al método inquiryCompleted(). Este método recibe un argumento, que puede ser INQUIRY_COMPLETED, INQUIRY_ERROR o INQUIRY_TERMINATED, dependiendo de cada caso.

***interface javax.bluetooth.DiscoveryAgent**

Esta interfaz provee métodos para descubrir dispositivos y servicios. Para descubrir dispositivos, esta clase provee del método startInquiry() para poner al dispositivo en modo de búsqueda, y el método retrieveDevices() para obtener la información de dispositivos previamente encontrados. Además provee del método cancelInquiry() que permite cancelar una operación de búsqueda.

3.1.3.1.2.2. Descubrir Servicios (Service Discovery)

En este capítulo vamos a ver la parte del API que usa el cliente para descubrir servicios disponibles en los dispositivos servidores encontrados. La clase DiscoveryAgent provee de métodos para buscar servicios en un dispositivo servidor Bluetooth e iniciar transacciones entre el dispositivo y el servicio. Este API no da soporte para buscar servicios que estén ubicados en el propio dispositivo.

Para descubrir los servicios disponibles en un dispositivo servidor, el cliente primero debe recuperar un objeto que encapsule funcionalidad SDAP. Este objeto es del tipo DiscoveryAgent, cuyo pseudocódigo viene dado por:

DiscoveryAgent da = LocalDevice.getLocalDevice().getDiscoveryAgent();

3.1.3.1.2.2.1. Clases del Service Discovery

***class javax.bluetooth.UUID**

Esta clase encapsula enteros sin signo que pueden ser de 16, 32 ó 128 bits de longitud. Estos enteros se usan como un identificador universal cuyo valor representa un atributo del servicio. Sólo los atributos de un servicio representados con UUID están representados en la Bluetooth SDP.

***class javax.bluetooth.DataElement**

Esta clase contiene varios tipos de datos que un atributo de servicio Bluetooth puede usar. Algunos de estos son:

- String
- boolean
- UUID
- Enteros con signo y sin signo, de uno, dos, cuatro o seis bytes de longitud

- Secuencias de cualquiera de los tipos anteriores. Esta clase además presenta una interfaz que permite construir y recuperar valores de un atributo de servicio.

***class javax.bluetooth.DiscoveryAgent**

Esta clase provee métodos para descubrir servicios y dispositivos.

***interface javax.bluetooth.ServiceRecord**

Esta interfaz define el Service Record de Bluetooth, que contiene los pares (atributo ID, valor). El atributo ID es un entero sin signo de 16 bits, y valor es de tipo DataElement. Además, esta interfaz tiene un método populateRecord() para recuperar los atributos de servicio deseados (pasando como parámetro al método el ID del atributo deseado).

***interface javax.bluetooth.DiscoveryListener**

Esta interfaz permite a una aplicación especificar un listener que responda a un evento del tipo Service Discovery o Device Discovery. Cuando un nuevo servicio es descubierto, se llama al método servicesDiscovered(), y cuando la transacción ha sido completada o cancelada se llama a serviceSearchCompleted().

3.1.3.1.2.2.2. Registro del Servicio (Service Registration)

Las responsabilidades de una aplicación servidora de Bluetooth son:

1. Crear un Service Record que describa el servicio ofrecido por la aplicación.

2. Añadir el Service Record al SDDB del servidor para avisar a los clientes potenciales de este servicio.
3. Registrar las medidas de seguridad Bluetooth asociadas a un servicio.
4. Aceptar conexiones de clientes que requieran el servicio ofrecido por la aplicación.
5. Actualizar el Service Record en el SDDB del servidor si las características del servicio cambian.
6. Quitar o deshabilitar el Service Record en el SDDB del servidor cuando el servicio no está disponible.

A las tareas 1, 2, 5 y 6 se las denominan registro del servicio (Service Registration), que comprenden unas tareas relacionadas con advertir al cliente de los servicios disponibles.

3.1.3.1.2.2.3. Responsabilidades del Registro de Servicio:

En la figura vemos, que cuando la aplicación llama a Connector.open() con un String conexión URL, la implementación crea un ServiceRecord. El correspondiente registro del servicio es añadido a la SDDB por la implementación cuando la aplicación servidora llama a acceptAndOpen(). La aplicación servidora puede acceder a dicho ServiceRecord llamando a getRecord() y hacer las modificaciones pertinentes. Las modificaciones se hacen también en el ServiceRecord de la SDDB cuando la aplicación llama a updateRecord(). Finalmente el ServiceRecord es eliminado de la SDDB cuando la aplicación servidora manda un close al *notifier*.

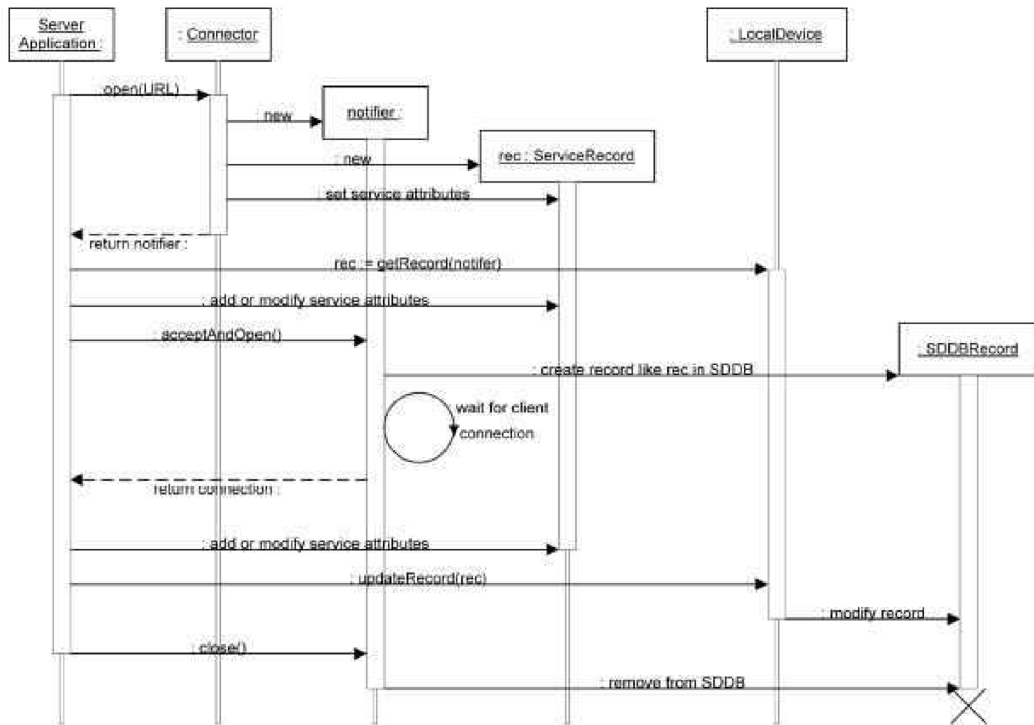


Figura III.8. Colaboración entre la implementación y la aplicación servidora para el registro del servicio, BORCHES (2).

3.1.3.1.2.2.4. Modos conectable y no conectable

- Modo conectable: un dispositivo en este modo escucha periódicamente intentos de iniciar una conexión de un dispositivo remoto.
- Modo no-conectable: un dispositivo en este modo no escucha intentos de iniciar una conexión de un dispositivo remoto.

Para el correcto funcionamiento de una aplicación servidora, es necesario que el dispositivo servidor esté en modo conectable. Es por esto, que en la implementación de acceptAndOpen(), ésta debe asegurarse que el dispositivo local esté en modo conectable (dado que depende del usuario el tener o no el dispositivo en un modo u

otro). La implementación hace una petición al BCC para hacer al dispositivo local conectable, si ésta no es posible, se lanzará una excepción BluetoothStateException.

Cuando todos los servicios en la SDDB han sido eliminados o deshabilitados, la implementación puede decidir opcionalmente pedir al dispositivo servidor que pase al modo no-conectable.

Aunque un dispositivo esté en el modo no-conectable (no responde a intentos de conexión), puede iniciar un intento de conexión. Por esto, un dispositivo en modo no-conectable puede ser un cliente, pero no un servidor. Por lo tanto la implementación no necesita pedir al dispositivo que se ponga en modo conectable si no tiene ningún ServiceRecord en su SDDB.

3.1.3.1.2.2.5. Clases del Service Registration

***interfaz javax.bluetooth.ServiceRecord**

Un ServiceRecord describe un servicio Bluetooth a los clientes. Está compuesto de unos “cuantos” atributos de servicio. El SDP del servidor mantiene una base de datos de los ServiceRecords. Un servicio *run-before-connect* (la aplicación se ejecuta sin esperar a que haya una conexión establecida) añade su ServiceRecord a la SDDB llamando a acceptAndOpen(). El ServiceRecord provee suficiente información a un cliente SDP para poder conectarse al servicio Bluetooth del dispositivo servidor.La aplicación servidor puede usar el método setDeviceClasses() para activar alguno de los bits de la clase servidora para reflejar la incorporación de un nuevo servicio.

***class javax.bluetooth.LocalDevice**

Esta clase provee un método getRecord() que la aplicación servidora puede usar para obtener el ServiceRecord. Una vez modificado, puede ser puesto en la SDDB llamando al método notifier.acceptAndOpen() o updateRecord() de LocalDevice.

***class javax.bluetooth.ServiceRegistrationException extends java.io.IOException**

La excepción ServiceRegistrationException se lanza cuando se intenta añadir o modificar un ServiceRecord en la SDDB y hay algún error. Estos errores pueden ocurrir:

- Durante la ejecución de Connector.open().
- Cuando un servicio *run-before-connect* invoca a acceptAndOpen() y la implementación intenta añadir el ServiceRecord asociado al *notifier* en la SDDB.
- Después de la creación del ServiceRecord, cuando la aplicación servidora intenta modificar el ServiceRecord en la SDDB usando updateRecord().

3.1.3.1.3. Manejo del Dispositivo

Los dispositivos inalámbricos, son más vulnerables a ataques del tipo spoofing y eavesdropping que los demás dispositivos. Es por ello, que la tecnología Bluetooth incluye una serie de medidas para evitar estas vulnerabilidades, como es por ejemplo el salto de frecuencia; más aún, Bluetooth provee además de otros mecanismos opcionales como son la encriptación y autenticación.

3.1.3.1.1.1. Perfil de acceso genérico (GAP)

3.1.3.1.1.1.1. Clases del GAP

Ahora veremos las clases que representan los objetos Bluetooth esenciales, como son LocalDevice y RemoteDevice. Las clases DeviceClass y BluetoothStateException dan soporte a la clase LocalDevice. La clase RemoteDevice representa un dispositivo remoto y provee métodos para obtener información del dispositivo remoto.

***class javax.bluetooth.LocalDevice**

Esta clase provee acceso y control sobre el dispositivo local Bluetooth. Está diseñada para cumplir con los requerimientos del GAP definidos para Bluetooth.

***class javax.bluetooth.RemoteDevice**

Esta clase representa al dispositivo Bluetooth remoto. De ella se obtiene la información básica acerca de un dispositivo remoto incluyendo su dirección Bluetooth y su *friendlyname* (nombre Bluetooth del dispositivo).

***class javax.bluetooth.BluetoothStateException extends java.io.IOException**

Esta excepción ocurre cuando un dispositivo no puede atender una petición que normalmente atendería por culpa de las características de la conexión radio (Ej: en ocasiones algunos dispositivos no permiten a otro conectarse cuando ya están conectados a otro dispositivo).

***class javax.bluetooth.DeviceClass**

Esta clase define los valores del tipo de dispositivo y los tipos de servicios de un dispositivo.

3.1.3.1.4. Comunicación

Para usar un servicio en un dispositivo Bluetooth remoto, el dispositivo local debe comunicarse usando el mismo protocolo que el servicio remoto. Los APIs permiten usar RFCOMM, L2CAP u OBEX como protocolo de nivel superior

El Generic Connection Framework del CLDC provee la conexión base para la implementación de protocolos de comunicación. CLDC provee de los siguientes métodos para abrir una conexión:

- Connection Connector.open(String name);
- Connection Connector.open(String name, int mode);
- Connection Connector.open(String name, int mode, boolean timeouts);

La implementación debe soportar abrir una conexión con una conexión URL servidora o con una conexión URL cliente, con el modo por defecto READ_WRITE.

El desarrollo del presente trabajo escrito se lo realizó en base a la comunicación RFCOMM o SPP, que no es más que una emulación de un puerto serial a través de la conexión bluetooth, es por eso que detallaremos más SPP que la conexión a través de L2CAP u OBEX.

3.1.3.1.4.1. Perfil del puerto serie

El protocolo RFCOMM provee múltiples emulaciones de los puertos serie RS-232 entre dos dispositivos Bluetooth. Las direcciones Bluetooth de los dos puntos terminales definen una sesión RFCOMM. Una sesión puede tener más de una conexión, el número de conexiones dependerá de la implementación. Un dispositivo podrá tener más de una sesión RFCOMM en tanto que esté conectado a más de un dispositivo.

3.1.3.1.4.1.1. Conexiones URL de un cliente y servidor SPP

A continuación vamos a mostrar algunos de los argumentos (ABNF) necesarios para la conexión URL entre clientes y servidores.

*srvString = protocol colon slashes srvHost 0*5(srvParams)*
*cliString = protocol colon slashes cliHost 0*3(cliParams)*

protocol = btspp

btspp = %d98.116.115.112.112 // define el literal btspp

cliHost = address colon channel
srvHost = "localhost" colon uuid

channel = %d1 -30
*uidd = 1*32(HEXDIG)*
colon = ":"
slashes = "//"
bool = "true" / "false"
*address = 12*12(HEXDIG)*
text = 1(ALPHA/ DIGIT / SP / "-" / "_")*

name = ";name=" text

master = “;master=”bool
encrypt = “;encrypt=” bool
authorize = “;authorize=” bool
authenticate = “;authenticate=” bool
cliParams = master / encrypt / authenticate
servParams = name / master / encrypt / authorize / authenticate

SP se usa para espacios, ALPHA para letras alfabéticas mayúsculas y minúsculas, DIGIT se usa para números de cero a nueve y HEXDIG para números hexadecimales (0-9, a-f,A-F).

3.1.3.1.4.1.2. Registro del servicio del puerto serie

Un SPP debe inicializar los servicios que ofrece y registrarlos en el SDDB. Un servicio de puerto serie viene representado por un par de objetos emparentados:

- Un objeto que implementa el interfaz javax.microedition.io.StreamConnectorNotifier Este objeto escucha conexiones clientes que demanden este servicio.
- Un objeto que implemente el interfaz javax.bluetooth.ServiceRecord. Este objeto describe el servicio y como puede ser accedido por dispositivos remotos.

Para crear estos objetos la aplicación servidora usa el método Connector.open() con un argumento de conexión URL, del siguiente modo:

StreamConnectionNotifier service = (StreamConnectionNotifier)Connector.open (btspp://localhost: 1101, true;name=VatoServer);

Invocando Connector.open() con un argumento conexión URL, éste devuelve un StreamConnectionNotifier que representa el servicio SPP. La implementación de Connector.open() además crea un nuevo registro de servicio (*service record*) que representa el servicio SPP. Una implementación de un SPP debe realizar los siguientes pasos cuando crea el registro de servicio.

- Crear un identificador de un canal servidor RFCOMM, chanN y asignarlo.
- chanN es añadido al ProtocolDescritorList en el registro de servicio.
- El UUID (1101) usado en el c*onnection string* para describir el tipo de servicio ofrecido es añadido al ServiceClassIDList.
- El atributo ServiceName es añadido al registro de servicio con el valor "VatoServer".

En el caso de un servicio *run-before-connect*, el registro de servicio es añadido a la SDDB la primera vez que la aplicación servidora llama a acceptAndOpen() en el StreamConnectionNotifier asociado. El registro de servicio se hace visible a potenciales clientes SPP cuando es añadida a la SDDB.

3.1.3.1.4.2. Establecimiento de la conexión

3.1.3.1.4.2.1. Establecimiento de la conexión del servidor

Un servidor SPP crea un objeto StreamConnectionNotifier del siguiente modo:

- Usando el apropiado string para un servidor SPP como argumento de Connector.open()

- Haciendo un *casting* del resultado de Connector.open() al interfaz StreamConnectionNotifier.

StreamConnectionNotifier service = (StreamConnectionNotifier)Connector.open (“btspp://localhost: 1101, true;name=VatoServer”);
StreamConnection con = (StreamConnection) service.acceptAndOpen();

Un servicio SPP puede aceptar múltiples conexiones de diferentes clientes llamando a acceptAndOpen() repetidamente. Cada cliente accede al mismo registro de servicio y se conecta al servicio usando el mismo canal servidor RFCOMM. Si el sistema Bluetooth no soporta múltiples conexiones, el acceptAndOpen() lanzará un BluetoothStateException.

El método close() en el objeto StreamConnection representa que se ha usado una conexión SPP servidora para cerrar la conexión.

Cuando un servicio *run-before-connect* manda un mensaje close() al StreamConnectionNotifier, el registro de servicio asociado a ese *notifier* se vuelve inaccesible a los clientes que estén usando el servicio *discovery*. La implementación debe eliminar el registro de servicio de la SDDB. El mensaje close() además hace que la implementación desactive los bits de clase que fueron activados por setServiceClasses() (excepto si otro *notifier* activó esos bits y aún está activo).

3.1.3.1.4.2.2. Establecimiento de la conexión del cliente

Antes de que un cliente SPP pueda establecer una conexión con un servicio SPP, éste debe previamente haber descubierto el servicio mediante el servicio *discovery*. Una

conexión URL del cliente incluye la dirección Bluetooth del dispositivo servidor y el identificador de canal del servidor. El método getConnectionURL() en el interfaz ServiceRecord se usa para obtener la conexión URL del cliente.

Invocando el método Connector.open() con una conexión URL del cliente, devuelve un objeto StreamConnection que representa la conexión SPP del lado del cliente.

StreamConnection con = (StreamConnection)
Connector.open("btspp://dirección:identificador_canal");

3.1.3.1.4.2.3. Registro de Servicio del SPP

Los registros de servicio consisten en una colección de pares (attrID, attrValue). Cada par describe un atributo del servicio. La aplicación servidora puede opcionalmente añadir otros atributos de servicio al ServiceRecord. Es posible incluso añadir atributos definidos por el usuario.

Con el método updateServiceAvailability(), la aplicación servidora puede obtener el ServiceRecord que fue creado por:

ServiceRecord record = localDev.getRecord(notifier);

La aplicación servidora modificará el atributo ServiceAviability basándose en el número de conexiones de cliente actuales. Las modificaciones que la aplicación servidora hace al ServiceRecord no se reflejan inmediatamente en la SDDB, si no que para ello se usará:

localDev.updateRecord(record);

Hasta ahora hemos visto el código y la implementación correspondiente desde el lado del teléfono, como hemos dicho, J2ME tiene implementado las APIs necesarias para el control y la comunicación de dispositivos móviles a través de Bluetooth, pero un punto importante a tomar en cuenta, es el hecho que J2SE no cuenta con esas librerías.

Este aspecto fue desde un principio del desarrollo de la tesis un gran inconveniente, ya que podíamos desarrollar la aplicación Java en el servidor pero no teníamos forma alguna de manipular el hardware bluetooth de la misma, y por ende la transmisión de datos desde y hacia el teléfono era imposible.

Gracias al desarrollo de librerías de terceros, este problema fue resuelto. Para el desarrollo de la aplicación se utilizó el paquete “Bluecove**”** que da soporte al API JSR-82 para la plataforma Windows.

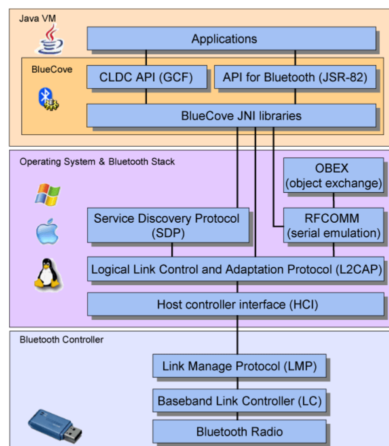


Figura III.9. Forma como se integra Bluecove a la JVM, APORTE A JAVA (16).

De esta manera, teníamos la posibilidad de transmitir y recibir datos desde el teléfono, para esto tenemos que añadir las librerías a nuestro proyecto.

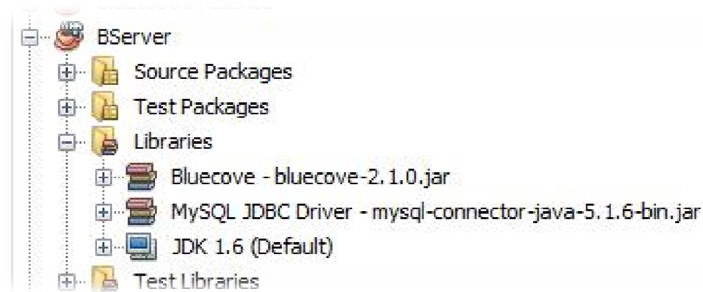


Figura III.10. Librerías Bluecove y JDBC de MySQL integrados al proyecto

Este archivo fue compilado y ejecutado sobre el IDE de NetBeans agregando al proyecto las librerías necesarias que se citaron anteriormente.

Gracias a estas librerías, podemos utilizar el código de J2ME en J2SE, tal y cual se tratase de cualquier otro MIDlet a excepción de que este no cuenta con los estados de inicio, pausa y destrucción de un MIDlet normal.

Con los elementos necesarios ya expuestos sobre la mesa, podíamos ya empezar a desarrollar ambas aplicaciones de forma paralela, tanto del teléfono móvil como en el computador que actuará de servidor, pero aún no contamos con un aspecto de vital importancia, que es la del registro de los usuarios hacia una base de datos.

El gestor de base de datos elegida para el proyecto por su flexibilidad y sencillez, fue MySQL. La conexión y registro lo veremos en el siguiente capítulo.

CAPÍTULO IV

CONECTIVIDAD A BASE DE DATOS DESDE JAVA

Como aspectos finales de la aplicación en el servidor, era la de poder registrar a los usuarios que se conectaban desde los teléfonos celulares a una base de datos, dando de esta manera una correcta administración de la información obtenida.

Como era de suponerse, tal y como se necesitó una librería adicional para el soporte JSR-82, también se necesita para la conectividad con la base de datos. Ahora veremos aspectos básicos de cómo se da el enlace desde Java hacia MySql.

4.1. JDBC

JDBC es un conjunto de clases e interfaces escritos en Java que ofrecen una API completa para la programación de bases de datos de diferentes proveedores (Microsoft SQL Server, Oracle, MySQL, Interbase, Microsoft Access, IBM DB2, PostgreSQL, etc...) usando instrucciones SQL.

JDBC realiza varias funciones:

- Conecta con la base de datos; la BD puede ser local (en nuestro PC) o remota (en otro PC)
- Envía las sentencias SQL
- Manipula los registros de la BD
- Recoge el resultado de la ejecución de las sentencias SQL

4.1.1. Fundamentos de los Drivers JDBC

Una inspección casual del API JDBC muestra rápidamente la dominación de las interfaces dentro del API, lo que podría llevar al usuario a preguntarse dónde se realiza el trabajo. Realmente esta es sólo una aproximación que tocan los desarrolladores JDBC porque la implementación real es la proporcionada por los vendedores de Drivers JDBC, que a su vez proporcionan las clases que implementan las interfaces necesarias. Esta aproximación presenta la competición que proporciona al consumidor más opciones, y para la mayor parte, produce mejor software. Con todos los drivers disponibles elegir uno puede ser difícil. Afortunadamente, Sun Microsystems mantiene una base de datos con más de 150 drivers JDBC de una amplia variedad de vendedores. Esta debería ser la primera parada después de seleccionar una base de datos.

Desde una perspectiva de programación, hay dos clases principales responsables para el establecimiento de una conexión con una base de datos. La primera clase es DriverManager, que es una de las clases que realmente proporciona el API JDBC. DriverManager es responsable de manejar un almacén de drivers registrados, esencialmente abstrayendo los detalles del uso de un driver para que el programador no tenga que tratar con ellos directamente. La segunda clase es la clase real del Driver JDBC. Estas son proporcionadas por vendedores independientes. La clase Driver JDBC es la responsable de establecer la conexión con la base de datos y de manejar todas las comunicaciones con la base de datos.

4.1.2. Registrar un Driver JDBC

El primer paso en el proceso de crear una conexión entre una aplicación Java y una base de datos es el registro de un driver JDBC con la JVM en la que se está ejecutando la aplicación Java. En el mecanismo de conexión tradicional (en oposición al mecanismo de conexión del DataSource) la conexión y todas las comunicación con la base de datos son controladas por un objeto DriverManager. Para establecer una conexión, se debe registrar un driver JDBC adecuado para la base de datos objetivo con el objeto DriverManager. La especificación JDBC, dice que se supone que los drivers JDBC se registran automáticamente a sí mismos con el objeto DriverManager cuando se cargan en la JVM. Por ejemplo, el siguiente fragmento de código usa un

inicializador estático para primero crear un ejemplar del driver JDBC persistentjava y luego registrarlo con el DriverManager :

```
static {
java.sql.DriverManager.registerDriver(new com.persistentjava.JdbcDriver()) ;
}
```

Registrar un driver es tan simple como cargar la clase del driver en la JVM, lo que puede hacerse de muchas maneras. Una forma es con el ClassLoader *Class.forName(com.persistentjava.JdbcDriver)*. Otro método, que no es tan bien conocido, usa la propiedad del sistema jdbc.drivers . Este método se puede usar de tres formas distintas:

Desde la línea de comandos:

```
java -Djdbc.drivers=com.persistentjava.JdbcDriver Connect
```

Dentro de la aplicación Java:

```
System.setProperty("jdbc.drivers","com.persistentjava.JdbcDriver") ;
```

Seleccionando la propiedad jdbc.drivers en el fichero de propiedades del sistema, que generalmente depende del sistema. Separando los drivers con una coma, se pueden registrar varios drivers usando la técnica de la propiedad del sistema mostrada arriba. Uno de los beneficios de usar la técnica de la propiedad del sistema es que los drivers

se pueden intercambiar fácilmente sin modificar ningún código (o al menos con unos mínimos cambios). Si se registran varios drivers, su orden de precedencia es:

Drivers JDBC registrados por la propiedad jdbc.drivers en la inicialización de la JVM, y Drivers JDBC cargados dinámicamente.

Como la propiedad jdbc.drivers sólo se chequea una vez durante la primera invocación del método DriverManager(), es importante asegurarse de que todos los drivers están registrados correctamente antes de establecer la conexión con la base de datos. Sin embargo, no todas las JVM están creadas igual, y algunas de ellas no siguen la especificación JVM. Como resultado, los inicializadores estáticos no siempre funcionan. Esto resulta en múltiples formas de registrar un driver JDBC, incluyendo:

Class.forName("com.persistentjava.JdbcDriver").newInstance();

DriverManager.registerDriver(new com.persistentjava.JdbcDriver()) ;

Estas alternativas deberían funcionar bien en todas las JVMs, por eso deberíamos sentirnos agusto usándolas a lo largo del amplio conjunto de JVM disponibles. Un problema final es que Class.forname() puede lanzar una ClassNotFoundException , por eso debemos envolver el código de registro en un manejador de excepción apropiado.

4.1.3. URLs de Drivers JDBC

Una vez que un Driver JDBC se ha registrado con el DriverManager, puede usarse para establecer una conexión a una base de datos. ¿Pero cómo selecciona DriverManager el driver correcto, dado que puede haber realmente registrados cualquier número de drivers? (Una sola JVM podría soportar múltiples aplicaciones concurrentes, que podrían conectarse con diferentes bases de datos con diferentes drivers). La técnica es bastante simple: cada driver JDBC usa una URL JDBC específica (que tiene el mismo formato que una dirección Web) como un significado de auto-identificación. El formato de la URL es correcto y probablemente parece familiar: jdbc:sub-protocol:database locator. El sub-protocol es específico del driver JDBC y puede ser odbc , oracle , db2 , etc., dependiendo del vendedor del driver real. El localizador de la base de datos es un indicador específico del driver para especificar de forma única la base de datos con la que una aplicación quiere interactuar. Dependiendo del tipo de driver, este localizador incluye un nombre de host, un puerto, y un nombre de sistema de base de datos.

Cuando se presenta con una URL específica, el DriverManager itera sobre la colección de drivers registrados hasta que uno de ellos reconoce la URL especificada. Si no se encuentra ningún driver adecuado, se lanza una SQLException . La siguiente lista presenta varios ejemplos específicos de URLs JDBC reales:

jdbc:odbc:jdbc

jdbc:oracle:thin:@persistentjava.com:1521:jdbc";

jdbc:db2:jdbc

Muchos drivers, incluyendo el driver puente JDBC-ODBC**,** acepta parámetros adicionales al final de la URL como un nombre de usuario y un password. El método para obtener una conexión a una base de datos, dando una URL JDBC específica, es llamar a getConnection() sobre el objeto DriverManager . Este método tiene varias formas:

DriverManager.getConnection(url) ;

DriverManager.getConnection(url, username, password) ;

DriverManager.getConnection(url, dbproperties) ;

Aquí url es un objeto String que es la URL JDBC, username y password son objetos String que representan el nombre de usuario y el password que la aplicación JDBC debería usar para conectar con la fuente de datos; y dbproperties es un objeto Properties de Java que encapsula todos los parámetros (posiblemente incluyendo nombre de usuario y el password) que requiere un driver JDBC para hacer una conexión con éxito. Ahora que que conocemos el driver básico, podemos examinar los tipos de drivers individuales:

4.1.3.1. Drivers del Tipo 1

Los drivers del tipo uno tienen algo en común: todos usan el puente JDBC-ODBC**,** que está incluido como parte estándar del JDK. Los drivers del tipo uno son diferentes al driver ODBC adjunto al puente JDBC-ODBC**.** Para conectar con una fuente de datos diferente, simplemente tenemos que registrar (o unir efectivamente) una fuente de datos ODBC diferente, usando el Administrador ODBC, al nombre de la fuente de datos apropiada.

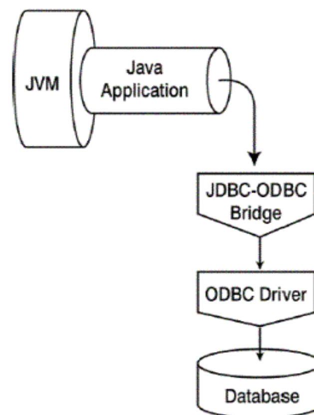


Figura IV.1. Representación drivers tipo 1, WEBTALLER (21).

4.1.3.1.1. Codificación para Drivers del Tipo 1

El nombre de clase para el driver puente JDBC-ODBC es sun.jdbc.odbc.JdbcOdbcDriver y al URL JDBC toma la forma jdbc:odbc:dsn , dsn es usado para registrar la base de datos con el Administrador ADBC. Por ejemplo, si una base de datos se registra con una fuente de datos ODBC llamada jdbc; un nombre de usuario de java y un password de sun, se puede usar el siguiente fragmento de código para establecer una conexión.

```
String url = "jdbc:odbc:jdbc" ;
Connection con ;
try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;
} catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ") ;
        System.err.println(e.getMessage()) ;
        return ;
} try {
        con = DriverManager.getConnection(url, "java", "sun");
        } catch(SQLException ex) {
        System.err.println("SQLException: " + ex.getMessage());
} finally {
        try{
                con.close ;
        } catch(SQLException ex) {
                System.err.println(SQLException: " + ex.getMessage()) ;
        } }
```

4.1.3.2. Drivers del Tipo 2

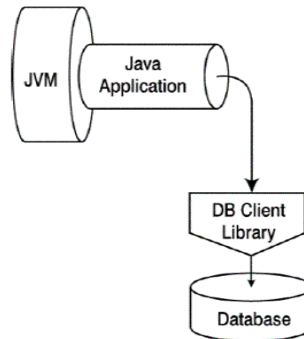


Figura IV.2. Representación drivers tipo 2, WEBTALLER (21).

Los drivers del tipo dos también son conocidos como drivers Java parciales, porque traducen directamente el API JDBC en un API específico de la base de datos. La aplicación cliente de base de datos debe tener las librerías cliente apropiadas para la base de datos, que podrían incluir código binario instalado y posiblemente ejecutándose. Para una aplicación distribuida, este requerimiento puede introducir problemas extra con las licencias, así como posibles pesadillas con los problemas de distribución de código. Por ejemplo, usar un modelo del tipo dos restringe a los desarrolladores a utilizar plataformas y sistemas operativos soportados por la librería cliente de la base de datos.

Sin embargo, este modelo puede funcionar eficientemente, cuando la base cliente está fuertemente controlada. Esto ocurre típicamente en LANs corporativas. Un ejemplo de driver del tipo dos es el driver de aplicación JDBC para DB2. El siguiente ejemplo demuestra cómo establecer una conexión usando un driver DB2:

```
String url = "jdbc:db2:jdbc" ;
try {
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver") ;
} catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ") ;
        System.err.println(e.getMessage()) ; return ;
}
```

Se puede observar la similitud que tiene este fragmento de código con el del ejemplo 1. Esta es la principal característica del modelo del tipo 2: la curva de aprendizaje para un programador que se mueve de un modelo a otro es prácticamente inexistente.

4.1.3.3. Drivers del Tipo 3

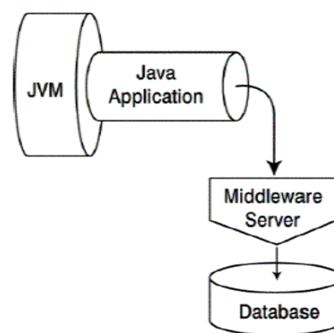


Figura IV.3. Representación drivers tipo 3, WEBTALLER (21).

Los drivers del tipo tres son drivers puro Java que transforman el API JDBC en un protocolo independiente de la base de datos. El driver JDBC no comunica directamente con la base de datos; comunica con un servidor de capa media, que a su vez comunica con la base de datos. Este nivel extra de dirección proporciona

flexibilidad en que se puede acceder a diferentes bases de datos desde el mismo código porque el servidor de la capa media oculta las especificidades a la aplicación Java. Para cambiar a una base de datos diferente, sólo necesitamos cambiar los parámetros en el servidor de la capa media. (Un punto a observar: el formato de la base de datos a la que estamos accediendo debe ser soportado por el servidor de la capa media).

El lado negativo de los drivers del tipo tres es que el nivel extra de indirección puede perjudicar el rendimiento general del sistema. Por otro lado, si una aplicación necesita interactuar con una variedad de formatos de bases de datos, un driver del tipo tres en una aproximación adecuada debido al hecho de que se usa el mismo driver JDBC sin importar la base de datos subyacente. Además, como el servidor de la capa media se puede instalar sobre una plataforma hardware específico, se pueden realizar ciertas optimizaciones para capitalizar los resultados perfilados.

4.1.3.4. Drivers del Tipo 4

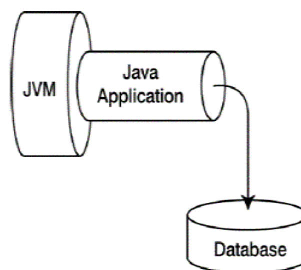


Figura IV.4. Representación drivers tipo 4, WEBTALLER (21).

Los drivers del tipo cuatro son drivers puro Java que se comunican directamente con la base de datos. Muchos programadores consideran éste el mejor tipo de driver, ya que normalmente proporciona un rendimiento óptimo y permite al desarrollador utilizar las funcionalidades específicas de la base de datos. Por supuesto este acoplamiento puede reducir la flexibilidad, especialmente si necesitamos cambiar la base de datos subyacente en una aplicación. Este tipo de driver se usa frecuentemente en applets y otras aplicaciones altamente distribuidas. El siguiente fragmento de código muestra cómo usar un driver DB2 del tipo cuatro:

```
String url = "jdbc:db2://persistentjava.com:50000/jdbc" ;
try {
        Class.forName("COM.ibm.db2.jdbc.net.DB2Driver") ;
} catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ") ;
        System.err.println(e.getMessage()) ;
        return ;
}
```

Este ultimo tipo de driver, por su flexibilidad y potencia es el que usaremos en el desarrollo de la aplicación servidora. Habiendo revisado la teoría acerca de los JDBC, ahora revisaremos las sentencias necesarias para la conexión. Lo primero que tenemos que saber para realizar la conexión a la base de datos es donde se encuentra dicha base de datos

Lo primero que debemos de hacer es instanciar la clase del driver, como se muestra con el siguiente código de la aplicación servidora.

```
Class.forName(com.mysql.jdbc.Driver).newInstance();
```

Dicha clase será la que nos proporcione una conexión a la base de datos, y como bien se ha dicho, la conexión se guardará en una instancia de la clase Connection.

```
String url="jdbc:mysql://localhost/usuariosbt?user=root&password=root";
con = DriverManager.getConnection(url);
```

Un vez se ha conseguido una conexión a la base de datos lo siguiente será el preparar una sentencia SQL en un createStatement.

```
String sql = "select * from users";
Statement stm = con.createStatement();
```

Construida nuestra sentencia, en el cual puede constar una consulta, inserción, eliminación o actualización de la base de datos, realizamos la ejecución de la misma. Y el resultado al ResultSet.

```
resultSet=stm.executeQuery(sql);
```

Un ResultSet no deja de ser una especie de matriz (filas x columnas) que deberemos de recorrer mediante el movimiento de un cursor. Y la forma más fácil en Java es mediante un bucle while. Y para acceder a las columnas bastará con utilizar los

métodos getXXX del ResultSet: getString() para las cadenas de texto, getDouble() para los decimales, getDate() para las fechas,......

```
while(resultSet.next()){
        mac[i]=resultSet.getString("MAC");
        name[i]=resultSet.getString("friendlyName");
        date[i]=resultSet.getString("date");
        i++;
    }
```

Una vez que hayamos finalizado el cometido, deberemos de cerrar las conexiones a la base de datos. Para ello invocaremos el método close() sobre los tres objetos mencionados. En todo este proceso, las excepciones. No nos debemos de olvidar de ellas. A tal respecto deberemos de ejecutar nuestro código en un bloque try-catch que controle la SQLException. Excepción común que se produce en el acceso a la base de datos.

```
resultSet.close();
      stm.close();
    }catch (SQLException ex) {
      ex.printStackTrace();
    }finally{
 try {
        if (con != null) {
          con.close();
        }
      }catch (SQLException ex1) {
        ex1.printStackTrace();
    }
  }
```

CAPÍTULO V

ANÁLISIS DE PRUEBAS Y RESULTADOS

Como aspecto fundamental e importante se presentan las pruebas realizadas a las aplicaciones cliente y servidor, y los resultados obtenidos de dichas pruebas, que serán los precedentes para determinar importantes conclusiones sobre el proyecto.

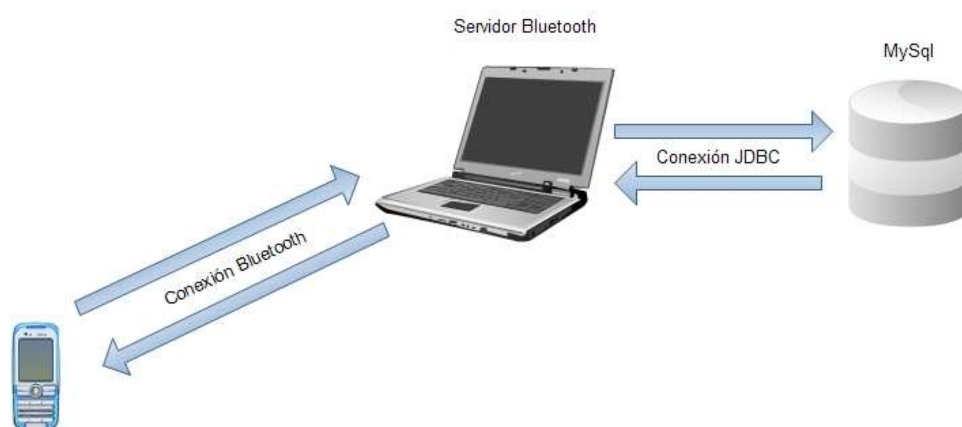


Figura V.1. Esquema general del proyecto

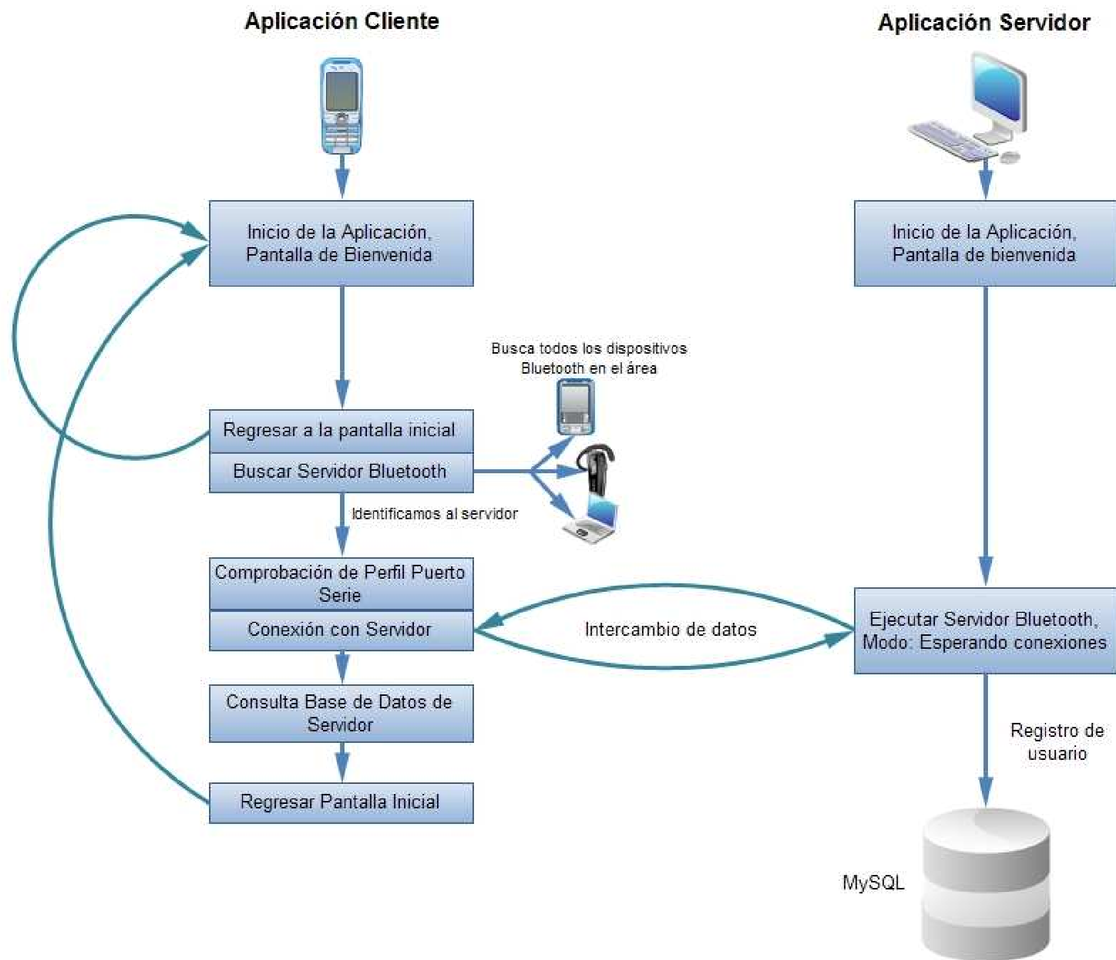


Figura V.2. Esquema detallado del proyecto

5.1. Ejecución de la aplicación Servidor

Luego de cierto tiempo de codificación del programa, estamos listos para llevar a cabo las pruebas respectivas del caso. Podemos ejecutar la aplicación de dos maneras, la primera ejecutándola directamente desde el archivo ".jar" generado por NetBeans, o desde el propio IDE.

La ventana de bienvenida podemos observarla en el siguiente grafico, en donde la aplicación se encuentra lista para la recepción de conexiones bluetooth.

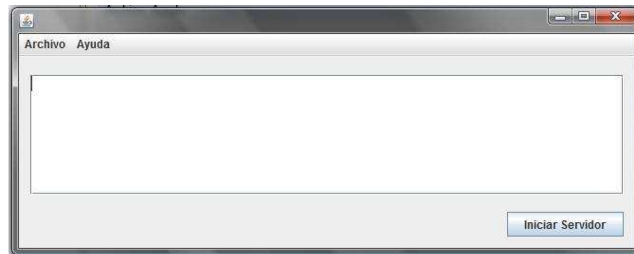


Figura V.3. Pantalla de inicio del programa servidor

Al momento de iniciar la aplicación, podemos notar algo importante que es la inicialización de la pila bluetooth de nuestro equipo. En nuestro caso en específico, quien se encarga de inicializar el “stack”, es el paquete Bluecove, que reconoce el hardware bluetooth instalado en el equipo y también es encargado de manejar correctamente los drivers del fabricante, en este caso particular “Widcomm”. El modulo con el que trabajamos viene integrado en la laptop utilizada como servidor, la Dell XPS M1530. El modulo Bluetooth es un “Dell TrueMobile 355 (2.0 + EDR Technology)” clase 2, con un alcance teórico de 10 metros. Este módulo es compatible con Bluetooth 2.0 Enhanced Data Rate (hasta 3 Mbps), que permite el uso de múltiples dispositivos Bluetooth para aplicaciones de banda ancha tales como la transferencia de grandes imágenes, vídeos y archivos de datos. Backward compatible con Bluetooth 1.2 y 1.1 que garantiza la compatibilidad con más dispositivos Bluetooth.

A continuación podemos observar en el momento cuando la aplicación se encuentra lista para la recepción de las conexiones clientes.

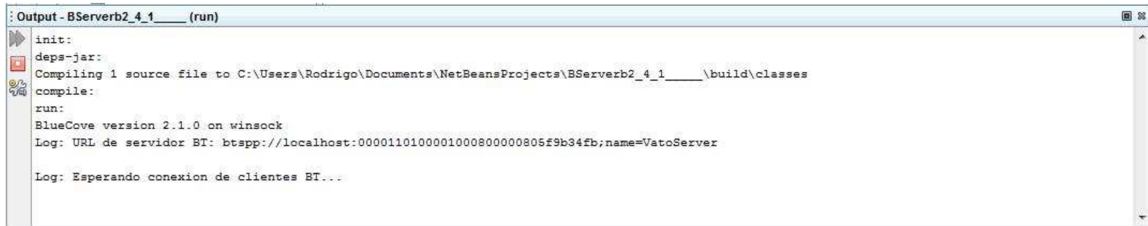


Figura V.4. Aplicación en modo de espera de conexiones bluetooth

Luego, procedemos a la inicialización de la aplicación cliente y la correspondiente conexión con el equipo servidor.

Después del emparejamiento y pasar los sistemas de autenticación, podemos intercambiar los datos tanto del teléfono como del Pc, como son la dirección bluetooth del teléfono móvil (podríamos hacer una analogía con las direcciones MAC de las tarjetas de red), su friendlyName, hora y fecha, etc.

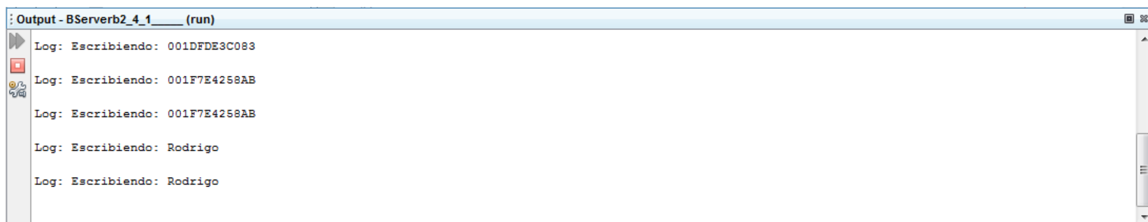


Figura V.5. Momento en que se da la transmisión/recepción de datos

Luego de intercambiar la información necesaria, procedemos a registrar los datos correspondientes del teléfono bluetooth en la base de datos. Podemos verificarlo a través de MySQL Query Browser, que los datos se encuentren correctamente registrados.

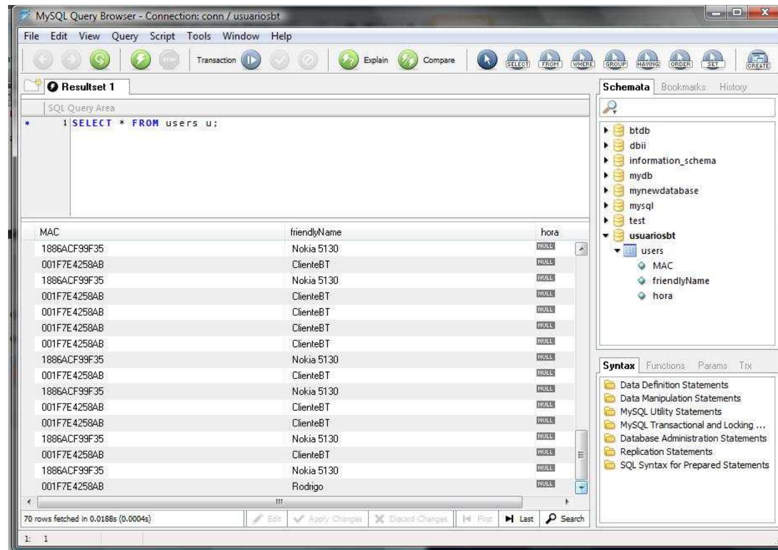


Figura V.6. Base de datos de los usuarios bluetooth conectados, consola de MySQL Query Browser

Hasta ahora hemos visto la interfaz por parte del servidor, veamos ahora la aplicación desde el punto de vista del teléfono móvil. Para el desarrollo de este trabajo y gracias a la independencia de plataforma de Java, se ha probado y testeado el programa Java con éxito en los siguientes modelos de teléfonos:

- Motorola A1200
- Nokia 5130
- Nokia 5310

Cada uno de estos teléfonos cuenta con el MIDP 2.0 y CLDC 1.0, necesarios para la ejecución del programa.

5.2. Ejecución en teléfono Motorola A1200 y Nokia 5310

Estos teléfonos cumplen con los requisitos necesarios para la ejecución del programa cliente denominado “BClient”. Cuentan con MID Profile 2.0 y CLDC 1.0, indispensables para la ejecución ya que tienen las librerías bluetooth que nos interesa.

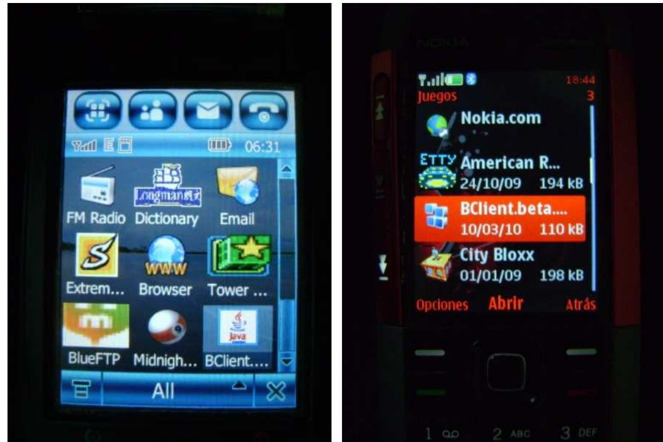


Figura V.7. Icono de la aplicación en la pantalla de menú principal del teléfono

Figura V.8. Pantalla de bienvenida

Tenemos como entrada la pantalla de bienvenida en el cual tenemos la opción de iniciar la búsqueda de servidores bluetooth, iniciaremos la acción presionando en el botón “Server”. Algo importante que notar es el hecho de la gran ventaja que nos brinda Java al momento de migrar la aplicación de un teléfono a otro. Como ejemplo podemos anotar que el Motorola A1200 es pantalla táctil y el Nokia 5310 de teclado

físico, y en ningún momento la aplicación lanzo algún tipo de error, con lo que podemos deducir lo fiables de las librerías de la KVM.

Figura V.9. Buscando servidores bluetooth con Protocolo de Puerto Serie (SPP)

Esperando un tiempo prudente, la aplicación reconoce todos los dispositivos bluetooth a su alrededor. Ahora deberemos conectarnos con el equipo servidor, para ello la aplicación cliente deberá verificar los servicios ofertantes del servidor y verificar si encuentra uno con el UUID correcto, en nuestro caso “0x1101” que corresponde al Perfil de Puerto Serie.

Figura V.10. Lista de dispositivos encontrados por la aplicación cliente

Figura V.11. Proceso de intercambio de datos entre el servidor y el cliente

Al verificar si el UUID es el correcto, se procede a la autenticación. Para la aplicación la autenticación está abierta y no requiere de un PIN previamente establecido. A razón seguida, se comienza con el intercambio de datos de ambos dispositivos para posteriormente registrarlos en la base de datos.

Figura. V.12. Finalización del intercambio de datos y cierre de la conexión

Figura V.13. Visualizacion de la base de datos del servidor

Como etapa final, tenemos la opcion de visualizar los ultimos cinco registros de la tabla “users” del servidor, acorde a la pantalla, la tabla se visualizara de mejor o peor manera.

5.3. Analisis de Resultados

Después de realizar el intercambio de datos, y comprobar la multiplataforma de Java en algunos celulares, podemos determinar que el trabajo de tesis ha culminado. Se encontró algunos problemas con teléfonos celulares de gama alta como también en teléfonos de gama baja que eran los pioneros en implementar java. El problema residía fundamentalmente en la incompatibilidad de MIDP con el cual fue desarrollado el programa cliente y el MIDP del teléfono residente.

Podemos destacar como otro punto importante el radio con el que cuenta el modulo utilizado, en la práctica, luego de realizar algunas pruebas, obteníamos un alcance máximo de 8.6 metros, sin pérdida de datos. Esta prueba fue realizada con el modulo

integrado a la laptop Dell XPS M1530 y el teléfono móvil Motorola A1200. Podemos añadir a lo anteriormente expuesto, que en el momento de establecer la conexión a una distancia cercana al emisor bluetooth, y una vez procedíamos al intercambio de datos, no se perdía la conexión, aun a pesar de alejarnos más allá del límite de 8.6 metros ya establecido anteriormente como la distancia máxima de conexión.

Cabe destacar, que el API que da soporte para bluetooth en Java, desafortunadamente no es muy versátil, por ejemplo, no podemos saber la intensidad de la señal Bluetooth, tampoco podemos realizar aplicaciones de voz, o un sistema que pueda determinar la distancia entre dos dispositivos conectados, para este propósito deberíamos explorar otros escenarios con lenguajes nativos del teléfono como puede ser en C o C++.

Debido a esta limitación derivada del API, se hace difícil obtener resultados cuantitativos del proyecto, reservándonos más al campo de programación.

CONCLUSIONES

1. El control se logra de manera exitosa cumpliendo con los parámetros impuestos por el radio de alcance de Bluetooth
2. Se ha logrado comprender y asimilar el funcionamiento del protocolo de comunicaciones Bluetooth.
3. Se consiguió una exitosa conectividad de la base de datos de MySQL con Java.
4. El software es muy sencillo de utilizar, reservándose para el usuario la administración de los usuarios contenidos en la base de datos.
5. Las pruebas se han realizado de manera exitosa, previendo mejoras y actualizaciones del software en futuros trabajos.
6. La tecnología de comunicación Bluetooth abre un amplio abanico de posibilidades para la implementación de proyectos donde no sea un factor importante la distancia y la velocidad de transmisión.
7. El sistema servidor admite una conexión bluetooth a la vez, esto debido a las limitaciones del API JSR-82 sobre el dispositivo.
8. El radio de cobertura del transmisor se reduce sustancialmente cuando se lo ejecuta en ambientes interiores con paredes y pasillos de por medio.
9. Algunos teléfonos móviles no podían ejecutar la aplicación cliente, aun a pesar de contar con el perfil correcto, siendo el error común la librería que controla el hardware bluetooth.

10. Java es un lenguaje donde todo es un objeto, la migración a este tipo de lenguajes es complicado y un tiempo considerable de aprendizaje.
11. Los Threads son segmentos de código muy útiles en java, ya que permiten la ejecución de programas concurrentes en el programa principal, dándonos una herramienta poderosa para recibir y enviar información al mismo tiempo.
12. Bluecove administra de manera eficiente los manejadores bluetooth acoplándose muy fácilmente a las aplicaciones Java.
13. Instalar un driver para la conexión con MySQL es muy sencillo y rápido, a más de brindar seguridad con la conexión.
14. Debido a la gran proliferación de teléfonos celulares con hardware bluetooth integrado, el desarrollo de este tipo de aplicaciones permiten ver un futuro de manejo de una gran cantidad de dispositivos por tan solo un componente.
15. Los teléfonos celulares se han convertido en un objetivo para los desarrolladores de software, en donde han visto un campo donde explotar por mucho tiempo.
16. Gracias al desarrollo de tecnologías de comunicación inalámbricas baratas como Bluetooth y ZigBee, se puede vislumbrar a la domotica como una alternativa muy accesible para los hogares.
17. Es necesario realizar una pausa en los intervalos de escritura de datos, esto ayuda a la sincronización entre el teléfono y la Pc.

18. El proyecto carece de características avanzadas debido a la escasa información y pocas características de las herramientas de desarrollo.
19. RFCOMM nos facilita de gran manera la comunicación, ya que este simula la conexión Bluetooth como si se tratara de un puerto serial conectado vía cable al computador, ocultando la complejidad de la tecnología inalámbrica.

Campos de Aplicación

Este sistema de control de usuarios, podría tener una gran variedad de aplicaciones. Podría ayudarnos a controlar con rapidez y facilidad la entrada y salida de usuarios de un determinado lugar de trabajo, o mediante el teléfono celular controlar periódicamente la información de una base de datos con tan solo pasar cerca de un servidor bluetooth.

El proyecto también podría ser migrado con gran aceptación al sector del transporte público, en donde los usuarios estarían registrados previamente en la base de datos del servidor con un saldo de dinero disponible, y al pasar por un punto de control, simplemente ejecutarían la aplicación cliente y se les descontaría automáticamente el valor del pasaje.

Un campo de aplicación novedoso e interesante podría ser en el del automovilismo. Podríamos integrar este trabajo a algún automotor para manipularlo remotamente sin necesidad de otro aparato más que el teléfono celular.

Trabajos a futuro

El sistema de control podría mejorarse en muchos aspectos, aun quedaría mucho por investigar e implementar. Un aspecto ideal de mejora pudiera ser la manipulación de algún objeto mecánico remotamente, como el hecho de registrarse en la base de datos para poder abrir una puerta por ejemplo.

RESUMEN

El presente trabajo de tesis desarrolló el sistema de control de usuarios basados en teléfonos móviles con bluetooth con el objetivo de determinar la fiabilidad del control y registro automático de usuarios.

El sistema denominado “JBluetooth” consta de dos aplicaciones. La aplicación servidor se desarrolló en Java 2 Standard Edition utilizando el API JSR-82 a través de la librería Bluecove para el control de hardware Bluetooth y con conexión a base de datos de MySQL a través de un JDBC. Por otra parte y la aplicación cliente se desarrolló en Java 2 Micro Edition que se probó sobre un celular Motorola A1200. La comunicación utiliza la técnica de programas concurrentes o Threads que logra sincronizar y registrar de manera automática a los usuarios.

Con la depuración del software, se inició la fase de pruebas en un escenario de 10 x 4 metros, con una distribución uniforme de decoración interior se logró alcanzar una distancia máxima y óptima de intercambio de datos de 8.6 metros de radio perdiendo conectividad a partir de esta distancia. El registro automático se logra sin problema alguno logrando el 100% de fiabilidad en la transmisión dentro del rango óptimo, perdiendo información solo cuando la persona se desplaza hacia afuera del radio Bluetooth. Cabe indicar que el radio de alcance se acorta drásticamente en un ambiente con paredes y columnas llegando a alcanzar el 50% de la distancia máxima alcanzada en un ambiente óptimo.

De esta manera se comprobó el correcto funcionamiento acorde a las especificaciones establecidas al inicio del proyecto.

SUMARY

The present thesis work developed the control system based users with Bluetooth mobile phones in order to determine the reliability of automatic control and registration of users.

The system called "JBluetooth" consists of two applications. The application server was developed in Java 2 Standard Edition using the API JSR-82 through the library Bluecove to control Bluetooth hardware connected to the MySQL database through a JDBC. Moreover, the client application developed in Java 2 Micro Edition, which was tested on a Motorola A1200 cell phone. The communication uses the technique of concurrent programs or Threads that achieves synchronize and automatically register users.

With debugging purposes, began the testing phase in a setting of 10 x 4 meters, with a uniform distribution of interior decoration may attain a maximum range and optimal data exchange of 8.6 m radius losing connectivity from this distance. Automatic registration is achieved without any problem, achieving 100% reliability in the transmission within the optimum range, losing information only when the person moves out of the Bluetooth radio. It is noted that the radio range is reduced drastically in an environment with walls and columns reaching up to 50% of the maximum distance reached in an optimal environment.

In this way verified the correct functioning according to specifications set at the beginning of the project

GLOSARIO

DSN	Data Source Name (Nombre de la Fuente De Datos)
ISM	Médico-Científica Internacional
FH	Salto de Frecuencia
FH/TDD	Salto de Frecuencia/División de Tiempo Duplex
TDD	Time Division Duplexing
SCO	Enlace de sincronización de conexión orientada
ACL	Enlace asíncrono de baja conexión
ARQ	Repetición Automática de consulta
CSVD	Modulación Variable de Declive Delta
MAC	Medio de Control de Acceso
TDM	División de Tiempo Múltiplexada
HTTP	HyperText Transfer Protocol
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
JIT	Just In Time
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
J2EE	Java 2 Enterprise Edition
JSP	Java Server Pages
CGI	Common Gateway Interface
JRE	Entorno de Ejecución Java
AWT	Abstract Window Toolkit
SWT	Standard Widget Toolkit

JRE Java Runtime Environment

IEEE Instituto de Ingenieros Eléctricos y Electrónicos

GUI Graphical User Interface

JNI Java Native Interfaces

JDK Java Development Kit

XML Extensible Markup Language

JNDI Java Naming and Directory Interface

RMI Remote Method Invocation

API Interfaz de Programación de Aplicaciones

PDA Personal Digital Assistant

JCP Java Community Process

MIDP Mobile Information Device Profiles

CLDC Connected Limited Device Configuration

CDC Configuración de Dispositivos Conectados

WTK Wireless Toolkit

IDE Entorno de Desarrollo Integrado

AMS Software de Administración de Aplicación

ROM Read Only Memory

RAM Read Access Memory

RFCOMM Radio Frequency Communication

L2CAP Logical Link Control and Adaptation Protocol

OBEX OBject EXchange

BCC Bluetooth Control Center

UUID Universally Unique Identifier

SDDB Service Discovery DataBase

GAP	Perfil de acceso genérico
GFC	Generic Connection Framework
SPP	Perfil de Puerto Serie
URL	Uniform Resource Locator
PIN	Personal Identification Number

ANEXOS

ANEXO A: CODIGO FUENTE – APLICACIÓN SERVIDOR

Paquete appMain-Controller.java

```
package appMain;

import java.sql.SQLException;
import javax.bluetooth.BluetoothStateException;
import server.serverBT;
import base.data;

public class Controller {
   public Controller(){
   }

   private static appJFrame frame;
   private static Controller instance;
   private static serverBT server;
   private static data db=null;

   public static Controller getInstance() throws BluetoothStateException {
      if(instance==null){
         instance=new Controller();
         frame=new appJFrame();
         server=new serverBT();
         db=new data();
      }
      return instance;
   }

   public void start(){
      frame.setVisible(true);
   }

  public void comenzar() throws SQLException {
     server.initSearch();
   }

   public void presentarTexto(String txt){
      frame.mostrarTexto(txt);
   }
}
```

Paquete appMain-appJFrame.java

```
package appMain;

import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.bluetooth.BluetoothStateException;

public class appJFrame extends javax.swing.JFrame {

   public Controller cont=null;

   /** Crea el form de la aplicacion*/
   public appJFrame() throws BluetoothStateException {
      initComponents();
      this.cont=Controller.getInstance();
   }

   public void mostrarTexto(String txt){
      areaTexto.setText(areaTexto.getText()+txt+"\n");
   }

   private void initComponents() {

      jScrollPane1 = new javax.swing.JScrollPane();
      areaTexto = new javax.swing.JTextArea();
      jButton1 = new javax.swing.JButton();
      jMenuBar1 = new javax.swing.JMenuBar();
      jMenu1 = new javax.swing.JMenu();
      jMenuItem1 = new javax.swing.JMenuItem();
      jMenu2 = new javax.swing.JMenu();
      jMenuItem2 = new javax.swing.JMenuItem();

      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
      setForeground(java.awt.Color.white);

      areaTexto.setColumns(20);
      areaTexto.setRows(5);
      jScrollPane1.setViewportView(areaTexto);

      jButton1.setText("Iniciar Servidor");
```

```
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
      }
    });
    jMenu1.setText("Archivo");

jMenuItem1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.
VK_X, java.awt.event.InputEvent.ALT_MASK));
    jMenuItem1.setText("Salir");
    jMenu1.add(jMenuItem1);
    jMenuBar1.add(jMenu1);
    jMenu2.setText("Ayuda");
    jMenuItem2.setText("Acerca de...");
    jMenu2.add(jMenuItem2);
    jMenuBar1.add(jMenu2);
    setJMenuBar(jMenuBar1);
    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 626,
Short.MAX_VALUE)
          .addComponent(jButton1, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addGap(20, 20, 20)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(jButton1)
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
```

```
        pack();
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            cont.comenzar();
        } catch (SQLException ex) {
            Logger.getLogger(appJFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    new appJFrame().setVisible(true);
                } catch (BluetoothStateException ex) {
                    Logger.getLogger(appJFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    }

    private javax.swing.JTextArea areaTexto;
    private javax.swing.JButton jButton1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JScrollPane jScrollPane1;
    // End of variables declaration
}
```

Paquete appMain-Main.java

```
package appMain;

import base.data;
import appMain.Controller;

import javax.bluetooth.BluetoothStateException;
import javax.swing.UIManager;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

   public static data base;
   private static Controller cont;

   public Main(){}

   public static void main(String[] args) {
      java.awt.EventQueue.invokeLater(new Runnable(){
         public void run(){
            try{
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
            }catch(Exception e){
               e.printStackTrace();
            }
            try {
               cont = Controller.getInstance();
            } catch (BluetoothStateException ex) {
               Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            }
            cont.start();
            }
      });
   }
  }
```

Paquete base-data.java

```
package base;

import java.sql.*;
import server.serverBT;

public class data {

   public static serverBT server=null;
   public static Connection con=null;
   public static ResultSet resultSet=null;
   //public static String campo1;
   //public static String campo2;
   public static String[] mac=new String[100];
   public static String[] name=new String[100];
   public static final String DriverClass = "com.mysql.jdbc.Driver";

   public data(){
   }

   static{
   try {
      Class.forName(DriverClass);
   } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      System.out.println("No puedo cargar el driver JDBC de la BD");
   }
}

    public static void newUser(String mac,String name) throws SQLException{
   try {
      Class.forName(DriverClass);
   } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
      System.out.println("No puedo cargar el driver JDBC de la BD");
   }
   try {
      String url="jdbc:mysql://localhost/usuariosbt?user=root&password=root";
      con = DriverManager.getConnection(url);
      //con.setAutoCommit(false);
```

```
        String sql = "INSERT INTO users(MAC,friendlyName,hora)
VALUES('"+mac+"','"+name+"',null)";
       Statement stm = con.createStatement();
       stm.executeUpdate(sql);
       stm.close();
    }catch (SQLException ex) {
        ex.printStackTrace();
        throw new SQLException(ex);
    }finally{
       try {
          if (con != null) {
             con.close();
          }
       }catch (SQLException ex1) {
       ex1.printStackTrace();
       }
   }
}

     public static void leerData() throws SQLException{
       int i=0;
       try {
             Class.forName(DriverClass);
       }catch (ClassNotFoundException ex) {
             ex.printStackTrace();
             System.out.println("No puedo cargar el driver JDBC de la BD");
       }
       try {
          String url="jdbc:mysql://localhost/usuariosbt?user=root&password=root";
          con = DriverManager.getConnection(url);
          String sql = "select * from users";
          Statement stm = con.createStatement();
          resultSet=stm.executeQuery(sql);
          while(resultSet.next()){
              mac[i]=resultSet.getString("MAC");
              name[i]=resultSet.getString("friendlyName");
             date[i]=resultSet.getString("date");
             i++;
          }
          resultSet.close();
          //stm.executeUpdate(sql);
          stm.close();
```

```
        }catch (SQLException ex) {
           ex.printStackTrace();
        }finally{
           try {
              if (con != null) {
                con.close();
              }
           }catch (SQLException ex1) {
               ex1.printStackTrace();
        }

    }
        serverBT.leer(mac, name);
}
}
```

Paquete logging-log.java

```
package logging;

import appMain.Controller;
import javax.bluetooth.BluetoothStateException;

public class log {
   public log(){

   }

   public static void Log(String texto) throws BluetoothStateException {
      System.out.println("Log: "+texto+"\n");
      Controller controller=Controller.getInstance();
      controller.presentarTexto(texto);
   }

   public static void Error(Throwable e){
      System.err.println("Error: "+e+"\n");
   }

}
```

Paquete server-serverBT.java

```
package server;

import java.sql.SQLException;
import javax.bluetooth.*;
import javax.microedition.io.*;

import java.util.Vector;
import java.io.*;

import logging.log;
import base.data;

public class serverBT {

   private static data db=null;
   public DiscoveryAgent da=null;
   public LocalDevice device=null;
   public RemoteDevice rd=null;
   public static String name=null;
   public static String mac=null;
   public static String[] campo1=new String[100];
   public static String[] campo2=new String[100];
   public static String[] ca1=new String[5];
   public static String[] ca2=new String[5];
   protected String UUID = new UUID("1101", true).toString();
   protected int discoveryMode = DiscoveryAgent.GIAC;
   protected static String endToken = "alto";
   protected static String[] datos=new String[2];
   //protected static String[] tokens = new String[13];
   //public static Vector datos=new Vector();
   public static final Vector token= new Vector();
   protected int j=0;


   public serverBT() throws BluetoothStateException{
      device=LocalDevice.getLocalDevice();
      da=device.getDiscoveryAgent();

   }
```

```
public void initSearch(){
   try {
      device.setDiscoverable(DiscoveryAgent.GIAC);
      String url = "btspp://localhost:" + UUID + ";name=VatoServer";
                        log.Log("URL de servidor BT: " + url);
                        StreamConnectionNotifier    notifier    =    (StreamConnectionNotifier)
                        Connector.open(url);
                        serverLoop(notifier);
                } catch (Throwable e) {
                        log.Error(e);
                }
}

private void serverLoop(StreamConnectionNotifier notifier) {
            try {
             while (true) { // loop infinito para aceptar conexiones
               log.Log("Esperando conexion de clientes BT...");
               queryDB();
               handleConnection(notifier.acceptAndOpen());
            }
               } catch (Exception e) {
               log.Error(e);
               }
        }

private void handleConnection(StreamConnection conn) throws IOException,
SQLException {
                DataOutputStream out = conn.openDataOutputStream();
                DataInputStream in = conn.openDataInputStream();
                startReadThread(in);
                try {
                        for (int i = 0; i < token.size(); i++) {
                                out.writeUTF((String) token.elementAt(i));
                                out.flush();
                                log.Log("Escribiendo: "+token.elementAt(i).toString());
                                // espera un tiempo antes de escribir otra vez
                                Thread.sleep(1 * 500);
                        }
                } catch (Exception e) {
                        log.Error(e);
                } finally {
   log.Log("Conexion cerrada.");
```

```
if (conn != null) {
try {
conn.close();
} catch (IOException e) {
}
}
}
    }

private void startReadThread(final DataInputStream in) {
                Thread reader = new Thread() {
                        public void run() {
        try {
      outer: while (true) {
           String s=in.readUTF();
           datos[j]=s;
           j++;
          log.Log("Leyendo:" + in.readUTF().toString());
          if (s.equals(endToken)){
          mac=datos[0];
          name=datos[1];
          data.newUser(mac,name);
          break outer;
       }
       }
       } catch (Throwable e) {
             log.Error(e);
       } finally {
       if (in != null) {
             try {
              in.close();
       } catch (IOException e) {
       }
       }
       }
       try {
          guardar();
       } catch (SQLException ex) {
          Logger.getLogger(serverBT.class.getName()).log(Level.SEVERE, null, ex);
       }

                        }
```

```
                    };
                    reader.start();
            }

    public void guardar() throws SQLException{
    mac=datos[0];
    name=datos[1];
    data.newUser(mac,name);
    }

public static void leer(final String[] c1,final String[] c2){
        tokens[0]="m";
        tokens[1]="n";
        tokens[12]=endToken;
        for(int i=2;i<(c1.length+2);i++){
           tokens[i]=c1[i-2];
        }
        for(int k=7;k<(c2.length+7);k++){
           tokens[k]=c2[k-7];
        }
    }

 public void queryDB() throws SQLException{
        data.leerData();
    }
}
```

ANEXO B: CODIGO FUENTE-APLICACIÓN CLIENTE

Paquete Cliente-Cliente.java

```
package Cliente;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;

import java.io.IOException.*;
import java.io.*;
import java.util.Vector;

import org.netbeans.microedition.lcdui.SimpleTableModel;
import org.netbeans.microedition.lcdui.SplashScreen;
import org.netbeans.microedition.lcdui.TableItem;

/**
 * @author Rodrigo
 */
public class Cliente extends MIDlet implements CommandListener, DiscoveryListener {

    private boolean midletPaused = false;
    public static String[] datos=new String[100];
    static boolean debug=false;
    static final String debugAddress="002269BE62BC";//SERVERBT
    protected UUID uuid=new UUID(0x1101);
    protected int inquiryMode=DiscoveryAgent.GIAC;
    protected int connectionOptions=ServiceRecord.NOAUTHENTICATE_NOENCRYPT;
    protected String stop="alto";
    protected String macAddress=null;
    protected String friendlyName=null;
    protected LocalDevice ld=null;
    protected Vector deviceList=new Vector();
    protected static final Vector data=new Vector();


    private Form bienvenido;
    private StringItem stringItem;
    private StringItem stringItem1;
```

```
private StringItem stringItem2;
private StringItem stringItem3;
private Form conexion;
private Form consultaBasedeDatos;
private TableItem tableItem;
private Alert Alerta;
private SplashScreen splashScreen;
private Command exitCommand;
private Command Server;
private Command Consultar;
private Command exitCommand1;
private Command backCommand;
private Command exitCommand2;
private Command backCommand1;
private Command salir;
private Command regresar;
private SimpleTableModel tableModel1;
private Ticker ticker;
private Image image1;
private Font font;

/**
 * constructor
 */

public Cliente() {
}

private void initialize() {
}
public void startMIDlet() {
   switchDisplayable(null, getSplashScreen());
}

public void resumeMIDlet() {

}
```

```
public void switchDisplayable(Alert alert, Displayable nextDisplayable) {
   Display display = getDisplay();
   if (alert == null) {
      display.setCurrent(nextDisplayable);
   } else {
      display.setCurrent(alert, nextDisplayable);
   }
}

public void commandAction(Command command, Displayable displayable) {
   if (displayable == Alerta) {
      if (command == regresar) {
         switchDisplayable(null, getBienvenido());
      } else if (command == salir) {
         exitMIDlet();
      }
   } else if (displayable == bienvenido) {
      if (command == Server) {
         switchDisplayable(null, getConexion());
         makeInfoAreaGUI();
         if(debug){
            startServiceSearch(new RemoteDevice(debugAddress){});
         }else
            try{
               //debug=true;
               deviceList.removeAllElements();
               startDeviceInquiry();
            }catch(Throwable t){
         log(t);
         }
      } else if (command == exitCommand) {
         switchDisplayable(getAlerta(), getBienvenido());
      }
   } else if (displayable == conexion) {
      if (command == Consultar) {
         switchDisplayable(null, getConsultaBasedeDatos());
         int k=0;
         for(int i=0;i<2;i++){
            for(int j=0;j<5;j++,k++){
               tableModel1.setValue(i, j,(String) data.elementAt(k+2));
               tableModel1.fireTableModelChanged();
               }
```

```
            }
        } else if (command == backCommand) {
            switchDisplayable(null, getBienvenido());
        }
    } else if (displayable == consultaBasedeDatos) {
        if (command == backCommand1) {
            switchDisplayable(null, getBienvenido());
        }
    } else if (displayable == splashScreen) {
        if (command == SplashScreen.DISMISS_COMMAND) {
            switchDisplayable(null, getBienvenido());
        }
    }
}

public Form getBienvenido() {
    if (bienvenido == null) {
        bienvenido = new Form("Bienvenido", new Item[] { getStringItem(),getStringItem1(),
        getStringItem2(), getStringItem3() });
        bienvenido.setTicker(getTicker());
        bienvenido.addCommand(getServer());
        bienvenido.addCommand(getExitCommand());
        bienvenido.setCommandListener(this);
    }
    return bienvenido;
}

public Form getConexion() {
    if (conexion == null) {
        conexion = new Form("Conexion");
        conexion.addCommand(getConsultar());
        conexion.addCommand(getBackCommand());
        conexion.setCommandListener(this);
    }
    return conexion;
}

public Form getConsultaBasedeDatos() {
    if (consultaBasedeDatos == null) {
        // write pre-init user code here
        consultaBasedeDatos = new Form("Ultimos 5 registros de la base de datos", new
Item[] { getTableItem() });
```

```
            consultaBasedeDatos.addCommand(getBackCommand1());
            consultaBasedeDatos.setCommandListener(this);
        }
        return consultaBasedeDatos;
    }

    public Alert getAlerta() {
        if (Alerta == null) {
            Alerta = new Alert("Alerta", "Esta seguro en salir de la aplicacion?", null, null);
            Alerta.addCommand(getRegresar());
            Alerta.addCommand(getSalir());
            Alerta.setCommandListener(this);
            Alerta.setTimeout(Alert.FOREVER);
        }
        return Alerta;
    }

    public Command getExitCommand() {
        if (exitCommand == null) {
            exitCommand = new Command("Salir", Command.EXIT, 0);
        }
        return exitCommand;
    }

    public Command getServer() {
        if (Server == null) {
            Server = new Command("Server", Command.OK, 0);
        }
        return Server;
    }

    public Command getConsultar() {
        if (Consultar == null) {
            Consultar = new Command("Consultar", Command.OK, 0);
        }
        return Consultar;
    }

    public Command getExitCommand1() {
        if (exitCommand1 == null) {
            exitCommand1 = new Command("Exit", Command.EXIT, 0);
        }
```

```
    return exitCommand1;
}

public Command getBackCommand() {
    if (backCommand == null) {
        backCommand = new Command("Regresar", Command.BACK, 0);
    }
    return backCommand;
}

public Command getBackCommand1() {
    if (backCommand1 == null) {
        backCommand1 = new Command("Regresar ", Command.BACK, 0);
    }
    return backCommand1;
}

public Command getExitCommand2() {
    if (exitCommand2 == null) {
        exitCommand2 = new Command("Exit", Command.EXIT, 0);
    }
    return exitCommand2;
}

public Command getRegresar() {
    if (regresar == null) {
        regresar = new Command("Regresar", Command.OK, 0);
    }
    return regresar;
}

public Command getSalir() {
    if (salir == null) {
        salir = new Command("Salir", Command.EXIT, 0);
    }
    return salir;
}
```

```
public TableItem getTableItem() {
   if (tableItem == null) {
      tableItem = new TableItem(getDisplay(), "Tabla: SERVERBT//usuariosBT/users");
      tableItem.setModel(getTableModel1());
   }
   return tableItem;
}

public SimpleTableModel getTableModel1() {
   if (tableModel1 == null) {
      tableModel1 = new SimpleTableModel(new java.lang.String[][] {
         new java.lang.String[] { "", "" },
         new java.lang.String[] { "", "" },
         new java.lang.String[] { "", "" },
         new java.lang.String[] { "", "" },
         new java.lang.String[] { "", "" }}, new java.lang.String[] { "mac", "name" });
   }
   return tableModel1;
}

public StringItem getStringItem() {
   if (stringItem == null) {
      stringItem = new StringItem("Jaime Rodrigo Vinueza Coba", null);
   }
   return stringItem;
}

public StringItem getStringItem1() {
   if (stringItem1 == null) {
      stringItem1 = new StringItem("Ingenieria Electronica", null);
   }
   return stringItem1;
}

public StringItem getStringItem2() {
   if (stringItem2 == null) {
      stringItem2 = new StringItem("Cliente Bluetooth", null);
   }
   return stringItem2;
}
```

```
public StringItem getStringItem3() {
   if (stringItem3 == null) {
       stringItem3 = new StringItem("Presione \"Server\" para iniciar la búsqueda de
       servidores bluetooth", null);
   }
   return stringItem3;
}


public Ticker getTicker() {
   if (ticker == null) {
      ticker = new Ticker("  ");
   }
   return ticker;
}

public SplashScreen getSplashScreen() {
   if (splashScreen == null) {
      splashScreen = new SplashScreen(getDisplay());
      splashScreen.setTitle("");
      splashScreen.setCommandListener(this);
      splashScreen.setImage(getImage1());
      splashScreen.setText("Bluetooth Client V1.0");
      splashScreen.setTextFont(getFont());
      splashScreen.setTimeout(2000);
   }
   return splashScreen;
}


public Image getImage1() {
   if (image1 == null) {
      try {
         image1 = Image.createImage("/IMAGE 241.jpg");
      } catch (java.io.IOException e) {
         e.printStackTrace();
      }
   }
   return image1;
}
```

```
public Font getFont() {
   if (font == null) {
        font = Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD,
       Font.SIZE_LARGE);
      // write post-init user code here
   }
   return font;
}

public Display getDisplay () {
   return Display.getDisplay(this);
}

public void exitMIDlet() {
   switchDisplayable (null, null);
   destroyApp(true);
   notifyDestroyed();
}

public void startApp() {
   if (midletPaused) {
      resumeMIDlet ();
   } else {
      initialize ();
      startMIDlet ();
   }
   midletPaused = false;
}

public void pauseApp() {
   midletPaused = true;
}

public void destroyApp(boolean unconditional) {
}

public void deviceDiscovered(RemoteDevice btDevice, DeviceClass dev) {
   log("Dispositivo descubierto ("+getDeviceStr(btDevice)+")");
   deviceList.addElement(btDevice);
}
```

```
public void servicesDiscovered(int transID, ServiceRecord[] records) {
   log("Servicio descubierto.");
   for(int i=0;i<records.length;i++){
      ServiceRecord rec=records[i];
      String url=rec.getConnectionURL(connectionOptions, false);
      handleConnection(url);
   }
}

public void serviceSearchCompleted(int transID, int respCode) {
   String msg=null;
   switch(respCode){
       case SERVICE_SEARCH_COMPLETED:msg="El servicio de busqueda termino
       correctamente";break;
      case SERVICE_SEARCH_TERMINATED:msg="El servicio de busqueda fue
       cancelado";break;
       case SERVICE_SEARCH_ERROR:msg="Un error se produjo mientras se procesaba la
       busqueda";break;
       case SERVICE_SEARCH_NO_RECORDS:msg="no fueron encontrados registros de
       servicios durante la busqqueda";break;
        case SERVICE_SEARCH_DEVICE_NOT_REACHABLE:msg="El dispositivo especificado
       en la busqueda no pudo ser encontrado o el dispositivo local no puede establecer la
       conexion con el dispositivo remoto";break;
   }
   log("Busqueda de servicios completada - "+msg);
}

public void inquiryCompleted(int arg0) {
   log("Busqueda completada. Por favor seleccione un dispositivo");
   makeDeviceSelectionGUI();
}

private void startServiceSearch(RemoteDevice device){
   try{
      log("Inicio de busqueda de perfil de puerto serie de: "+getDeviceStr(device));
      UUID uuids[]=new UUID [] {uuid};
      getAgent().searchServices(null,uuids,device,this);

   }catch(Exception e){
      log(e);
   }
}
```

```
private void startDeviceInquiry(){
   try{
      //*************************
      ld=LocalDevice.getLocalDevice();
      macAddress=ld.getBluetoothAddress();
      friendlyName=ld.getFriendlyName();
      //*************************
      log("Iniciando modo busqueda - espere por favor...");
      DiscoveryAgent da=getAgent();
      da.startInquiry(inquiryMode, this);

   }catch(Exception e){
      log(e);
   }
}

private void handleConnection(final String url){
   Thread echo=new Thread(){
     public void run(){
        //int i=0;
        data.removeAllElements();
        StreamConnection stream=null;
        try{
           log("Conectandose al servidor "+url);
           stream=(StreamConnection)Connector.open(url);
           log("Conexion Bluetooth abierta");
           DataInputStream in=stream.openDataInputStream();
           DataOutputStream out=stream.openDataOutputStream();
           log("Intercambiando datos...");
      outer: while(true){
              String r=in.readUTF();
              data.addElement(r);
              log("Leido \n"+r+", escribiendo de regreso...");
              if(r.equals("m")){
                 out.writeUTF(macAddress);
                 out.flush();
              }
              if(r.equals("n")){
                 out.writeUTF(friendlyName);
                 out.flush();
              }
```

```
                if(r.equals(stop)){
                    out.writeUTF(stop);
                    out.flush();
                    log("Fin de ciclo repetitivo");
                    break outer;
                }
            }
        }catch(IOException e){
            log(e);
        }finally{
        log("Conexion bluetooth cerrada");
        if(stream!=null){
        try{
        stream.close();
        }catch(IOException e){
        log(e);
        }
        }
        }
      }
    };
    echo.start();
  }

  private void makeInfoAreaGUI() {
        conexion.deleteAll();
        Display.getDisplay(this).setCurrent(conexion);
  }

  private void makeDeviceSelectionGUI(){
    final List devices=new List("Seleccione un dispositivo:",List.IMPLICIT);
    for(int i=0;i<deviceList.size();i++)
        devices.append(getDeviceStr((RemoteDevice)deviceList.elementAt(i)),null);
        devices.setCommandListener(new CommandListener(){
            public void commandAction(Command arg0,Displayable arg1){
                makeInfoAreaGUI();

startServiceSearch((RemoteDevice)deviceList.elementAt(devices.getSelectedIndex()));
            }
        });
        Display.getDisplay(this).setCurrent(devices);
  }
```

```
    synchronized private void log(String msg){
        conexion.append(msg);
        conexion.append("\n");
    }

    private void log(Throwable e){
        log(e.getMessage());
    }

    private DiscoveryAgent getAgent(){
        try{
            return LocalDevice.getLocalDevice().getDiscoveryAgent();

        }catch(BluetoothStateException be){
            throw  new Error(be.getMessage());
        }
    }

    private String getDeviceStr(RemoteDevice btDevice){
        return getFriendlyName(btDevice)+" -0x"+btDevice.getBluetoothAddress();
    }

    private String getFriendlyName(RemoteDevice btDevice){
        try{
            return btDevice.getFriendlyName(false);
        }catch(IOException e){
            return "Incapaz de encontrar nombre";
        }
    }

    private void arrange(){
        data.removeElementAt(0);
        data.removeElementAt(1);
        data.removeElementAt(12);
    }

}
```

ANEXO C: MANUAL DE USUARIO

El sistema denominado como “JBluetooth”, es intuitivo y muy fácil de usar. Aspectos importantes a tomar en cuenta antes del uso del mismo serían:

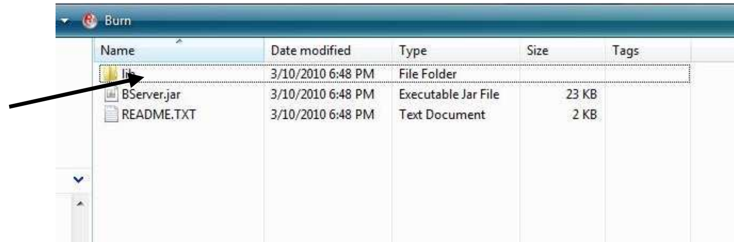
- Hardware Bluetooth en el computador y en los teléfonos que actuarán de servidor y clientes respectivamente.
- Perfil MIDP 2.0 y configuración CLDC 1.0 en el teléfono a instalarse la aplicación cliente.
- En el caso del servidor, deberá tener de preferencia drivers para el hardware bluetooth de Microsoft o IVT Bluesoleil, para que no existan ningún tipo de conflictos con la librería Bluecove incluida en la aplicación.
- El servidor deberá tener instalado y previamente configurado la versión más reciente de la Maquina Virtual de Java.

Gracias a la versatilidad de la multiplataforma de java, la instalación tanto del programa cliente como servidor son muy fáciles de realizar. Tal vez un poco de inconvenientes se lo puede encontrar por el lado del teléfono.

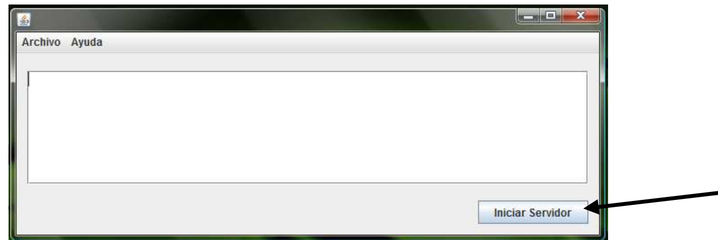
Funcionamiento del sistema

Servidor

- Como primer paso, preparamos el hardware bluetooth.
- Ejecutamos el archivo BServer.jar



- Cuando se esté ejecutando el sistema, presionamos el botón "Iniciar Server"



- Esperamos a que los clientes se registren automáticamente.

Cliente

- Ejecutamos el programa desde nuestro teléfono celular. Por lo general lo encontramos en la sección de juegos o aplicativos.
- Al arrancar, esperamos hasta que esté operable la aplicación. Presionamos el botón "Server". Nota: La posición del botón varía de acuerdo al tipo de teléfono celular que se use.

- La aplicación automáticamente buscara todos los dispositivos bluetooth a su alrededor. Escogemos el servidor asignado.

- Depende de la configuración del teléfono, éste le pedirá o no permiso para el intercambio de datos, en tal caso debe aceptar.
- Después de esto, el usuario quedará registrado, y si gusta podrá ver un listado de los últimos cinco usuarios registrados por el sistema.

Ultimos 5 registros de la base
SERVERBT//usuariosBT/users

mac	name
001DFDE3C083	SKAR
001F7E4258AB	Rodrigo
001F7E4258AB	Rodrigo
Rodrigo	n
Rodrigo	m

Regresar

Nos apoyaremos del software de administración de MySQL para llevar un control de todos los usuarios que el sistema a registrado por u lapso de tiempo.

Recomendaciones

- Se recomienda tener un respaldo de la información que se obtenga con el transcurso del uso del sistema.
- Debido a las limitaciones de distancia que impone la tecnología bluetooth, no es recomendable utilizar este sistema en lugares que se requiera el control que sobrepase los 10 metros de radio.
- Cuando el cliente establezca la conexión, es recomendable que permanezca estático, evitando de esta manera que se aleje del radio de cobertura impidiendo que pueda registrase correctamente y tal vez posibles caídas del sistema.
- Debido a las limitaciones del API JSR 82, no es recomendable que dos o más aplicaciones clientes accedan de manera simultánea al servidor.

BIBLIOGRAFIA

(1) HOPKINS, B. y RANJITH, A. Bluetooth for Java. [s.l.]: Apress, 2003. pp. 11-83.

[Ebook]

http://www.ebookcomputer.com/Bluetooth_for_Java_by_Bruce_Hopkins_pdf_ebook_download-ebook---.html

2009-08

(2) BORCHES JUZGADO, P.D. Java 2 Micro Edition: soporte Bluetooth. Madrid: Universidad Carlos III, 2004. 51 p. [Ebook]

http://grupos.emagister.com/documento/j2me_soporte_bluetooth/1034-105108

2009-08

(3) GALVEZ ROJAS, S. y ORTEGA DIAZ, L. Java a Tope: J2ME. Málaga: Universidad, [s.f.]

200p. [Ebook]

http://www.lcc.uma.es/~galvez/ftp/libros/J2ME.pdf

2009-08

(4) BRIEBA, A.G. JSR-82: Bluetooth desde Java. [s.l.]: [s.e.], 2004. 36 p. [Ebook]

http://www.javahispano.org/contenidos/es/jsr82_bluetooth_desde_java/;jsessionid=86B38F0973EF4A27C363D4081C259FAF

2009-08

(5) TOPLEY, K. J2ME in a Nutshell A Desktop Quick Reference. [s.l.]: O'Reilly, 2002

pp. 16-226. [Ebook]

http://www.chilanti.com/node/1282

2009-08

(6) CUADRA, D. CASTRO, E. y MARTINEZ, P. Diseño de Bases de Datos. [s.l.]: [s.e.], [s.f.]

pp. 20-102. [Ebook]

http://basesdatos.uc3m.es/fileadmin/_temp_/DBD.PDF

2009-09

(7) FEUERSTEIN, S. y HARRISON, G. MySQL Stored Procedure Programming. [s.l.]:

O'Reilly, 2006. pp. 202-337. [Ebook]

http://www.scribd.com/doc/15490808/MySQL-Stored-Procedure-Programming-by-OReilly-Media

2009-09

(8) O'DONAHUE, J. Java Database Programming Bible. [s.l.]: Wiley, 2002. pp. 6-309 [Ebook]
http://www.giuciao.com/books/book.php?id=1030&by=Java&ord=id
2009-10

(9) SILBERSCHATZ, A. KORTH, H.F. y SUDARSHAN, S. Fundamentos de Bases de Datos. Traducido del inglés por Database System Concepts. España: McGraw-Hill, 4ta. Ed, 2002. pp. 19-343. [Ebook]
http://www.librospdf.net/Silberschatz.-Korth.-Sudarshan.-Fundamentos-de-bases-de/1/
2009-10

(10) PRIETO, M. Introducción a J2ME. [s.l.]: [s.e.], 2002. 60 p. [Ebook]
http://www.scribd.com/doc/7136526/Manual-Programacion-Java-Curso-J2ME
2009-10

(11) PARRAGA, I. Curso de Java. [s.l.]: [s.e.], 2003. 139p. [Ebook]
http://www.scribd.com/doc/4026689/muy-buen-curso-de-java
2009-10

(12) OTERO, A. Java. [s.l.]: [s.e.], 2003. 119 p. [Ebook]
http://www.javahispano.org/contenidos/es/curso_de_programacion_java_v__abraham_otero/
2009-10

(13) KLINGSHEIM, A. J2ME Bluetooth Programing. [s.l.]: [s.e.], 2004. 26 p. [Ebook]

www.nowires.org/Presentations-PDF/AndreKpresentasjon.pdf

2009-10

(14) SUN DEVELOPER NETWORK. The Java APIs for Bluetooth Wireless Technology

http://developers.sun.com/mobility/midp/articles/bluetooth2/

2009-10

(15) SUN DEVELOPER NETWORK. Application Programming with J2ME and Bluetooth

http://developers.sun.com/mobility/midp/articles/bluetooth1/

2009-10

(16) APORTE A JAVA. Un pequeño aporte, para los que inician en Java

http://ungranoparajava.blogspot.com/2009/05/una-aplicacion-j2me-bluetooth-y-j2se.html

2009-10

(17) WIRELESS PRO NEWS. Introduction to Bluetooth and J2ME

http://archive.wirelesspronews.com/wirelesspronews-14-20041213IntroductiontoBluetoothandJ2ME.html

2009-10

(18) BLUETOOTH y J2ME. Estudio sobre J2ME, J2SE y Bluetooth JSR-82

http://profesores.elo.utfsm.cl/~agv/elo326/1s07/projects/DiegoGonzalez/Presentacion1Elo326.html

2009-10

(19) CODEGURU. Simple Bluetooth Communication in J2ME

http://profesores.elo.utfsm.cl/~agv/elo326/1s07/projects/

DiegoGonzalez/Presentacion1Elo326.html

2009-10

(20) WIKIPEDIA. Lenguaje de Programación Java

http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java

2009-10

(21) WEBTALLER. Conexiones a Base de Datos en Java

http://www.webtaller.com/construccion/lenguajes/info/lecciones/java/

2009-10