UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO Y CÁLCULO DE UN SISTEMA DE CONTROL PARA EQUIPOS ELÉCTRICOS UTILIZANDO LA TECNOLOGÍA BLUETOOTH

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

VALENTÍN LÓPEZ FERNÁNDEZ

PROMOCIÓN

2005 - II

LIMA – PERÚ
2011

DISEÑO Y CÁLCULO DE UN SISTEMA DE CONTROL PARA EQUIPOS ELÉCTRICOS UTILIZANDO LA TECNOLOGÍA BLUETOOTH

DEDICADO A MIS PADRES

SUMARIO

En la actualidad, muchos de los aparatos comercializados cuentan con tecnología inalámbrica, como por ejemplo la tecnología Bluetooth la cual viene incluida en la mayoría de los teléfonos celulares modernos. El objetivo de este proyecto es utilizar esta tecnología inalámbrica para controlar equipos eléctricos de uso en domicilios y en la industria mediante comandos vía Bluetooth.

El objetivo se convierte entonces en que las personas puedan controlar equipos eléctricos, pudiendo ser estos: motores eléctricos, luminarias, calentadores eléctricos, equipos de refrigeración, aire acondicionado, Bombas Contra Incendio, etc. utilizando un teléfono celular y un módulo que controlará el funcionamiento de encendido y apagado de dichos equipos eléctricos empleando la tecnología Bluetooth.

Para lograr este objetivo se ha tenido que diseñar (utilizando lenguaje de programación J2ME) el código de control de los equipos eléctricos los cuales son una Bomba BCI, un grupo electrógeno y un motor eléctrico.

El proceso se simula en el entorno JAVA, utilizando el software Netbeans, el código generado se instala en el celular el cual se denomina cliente, mientras la computadora se denomina servidor, el cual hará el papel del equipo eléctrico a controlar.

INDICE

PROLOGO

El objetivo principal del presente informe de suficiencia es la simulación del proceso de control (encendido y apagado) de equipos eléctricos, usando para este cometido la tecnología Bluetooth mediante aplicativos generados en el lenguaje J2ME, utilizando una computadora la que hace el papel del equipo eléctrico e instalando el aplicativo creado en el entorno Netbeans, en la memoria interna del celular, este trabajo comprende 5 capítulos los cuales son:

El capitulo **I** presenta una introducción a la tecnología Bluetooth, se determina las características propias de esta tecnología que pueden ser aprovechadas para aplicaciones de control, dentro de viviendas y de la industria.

El capitulo **II** trata sobre de la descripción de la plataforma Java Microedition, un subconjunto creado para aplicaciones pequeñas, también se realiza los elementos necesarios para la creación de una Midlet y tenerlo listo para instalarlo en el celular.

El capitulo **III** se resume el desarrollo de la aplicación, se analizan las clases principales que se usaron y se muestran partes del código del programa. El programa está analizado separando la interfaz de usuario y la interfaz de comunicación. Además esta el desarrollo del aplicativo para lo que es el servidor la cual se ejecuta en la computadora.

El capitulo **IV** Abarca el diseño de la interfaz de hardware, se estudia el circuito de alimentación y los circuitos acopladores, cabe resaltar que este punto no se llegó a implementar, el cual no es el objetivo de este informe.

El capitulo **V** presenta la simulación del control de equipos eléctricos en imágenes, utilizando un celular y una computadora.

CAPITULO I
DESCRIPCION DE LA TECNOLOGIA BLUETOOTH

En la actualidad los equipos móviles como los celulares cuentan con Tecnología Bluetooth. Las características de esta tecnología permiten usarla como enlace de comunicación entre dispositivos que no deben ser necesariamente del mismo tipo, por ejemplo, una PC con un teléfono o una PDA con unos audífonos, o la PC con los audífonos, etc. Por lo que aprovechando la misma tecnología ya incluida en estos equipos se puede crear nuevas aplicaciones.

El objetivo principal del presente proyecto es realizar una simulación de cómo esta tecnología BLUETOOTH entabla un puente de comunicación entre un teléfono celular y una computadora; cliente y servidor respectivamente, la computadora simulará ser un equipo eléctrico sobre el cual esta montado e instalado un módulo bluetooth; para su control y manejo.

1.1 Uso de la tecnología bluetooth en equipos móviles

Piconets.- Si un equipo se encuentra dentro del radio de cobertura de otro, éstos pueden establecer conexión entre ellos. Cada dispositivo tiene una dirección única de 48 bits, basada en el estándar IEEE 802.11 para WLAN. En principio sólo son necesarias un par de unidades con las mismas características de hardware para establecer un enlace. Dos o más unidades Bluetooth que comparten un mismo canal forman una piconet.

En la Fig.1.1 se muestra 8 unidades participantes, de las cuales una se convertirá en maestra, una piconet consta de un máximo de 8 unidades, 10 piconets como máximo pueden coexistir en una misma área de cobertura.

Están habilitados tres canales de voz de 64 kbit/s por piconet. Las conexiones son uno a uno con un rango máximo de diez metros, aunque utilizando amplificadores se puede llegar hasta los 100 metros, pero en este caso se introduce alguna distorsión. Los datos se pueden intercambiar a velocidades de hasta 1 Mbit/s. El protocolo banda base que utiliza Bluetooth combina las técnicas de circuitos y paquetes para asegurar que los paquetes lleguen en orden.

Bluetooth esta diseñado para operar en un ambiente multi-usuario. Los dispositivos deben habilitarse para comunicarse entre si e intercambiar datos de una forma transparente. Dado que cada enlace es codificado y protegido contra perdida de enlace,

Bluetooth puede considerarse una red inalámbrica de corto alcance y muy segura. En una piconet los participantes podrían intercambiar papeles si un esclavo quisiera asumir el papel de unidad maestra.

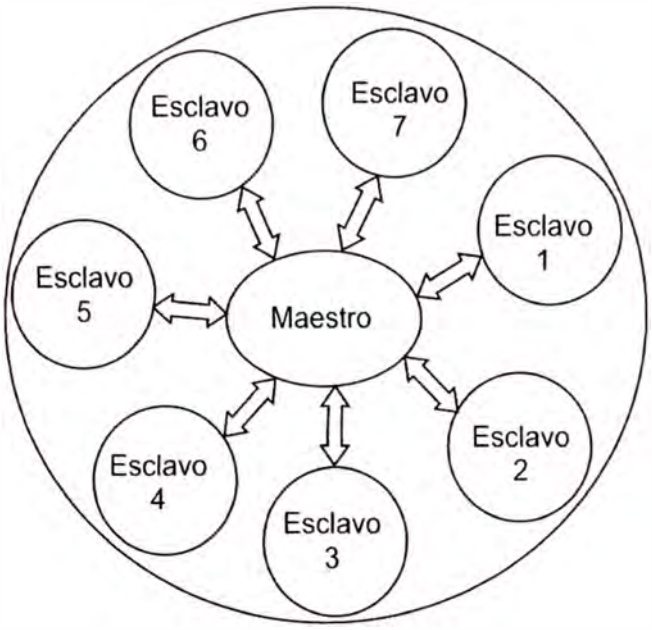


Fig.1.1: Piconet [1]

En el estado conectado el dispositivo Bluetooth puede encontrarse en varios modos de operación:

- *Active mode*: En este modo, el dispositivo Bluetooth participa activamente en el canal.
- *Sniff mode*: En este modo, el tiempo de actividad durante el cual el dispositivo esclavo escucha se reduce. Esto significa que el maestro sólo puede iniciar una transmisión en unos slots de tiempo determinados.
- *Hold mode*: En el estado conectado, el enlace con el esclavo puede ponerse en espera. Durante este modo, el esclavo puede hacer otras cosas, como escanear en busca de otros dispositivos, atender otra piconet, etc.
- *Park mode*: En este estado, el esclavo no necesita participar en la piconet, pero aún quiere seguir sincronizado con el canal. Deja de ser miembro de la piconet. Esto es útil por si hay más de siete dispositivos que necesitan participar ocasionalmente en la piconet.

El uso de la tecnología Bluetooth dentro de los equipos móviles se facilita al utilizar aplicaciones basadas en lenguajes de programación. Las aplicaciones Java para dispositivos móviles se encuentran muy difundidas actualmente.

Sun Microsystems, desarrolló Java Micro Edition para desarrollar aplicaciones para pequeños dispositivos, que tienen como tal, algunas capacidades reducidas como su interfaz gráfico, procesamiento, memoria, etc.

---

[1]Referencia:http://www.hispazone.com/Articulo/72/Bluetooth:-El-Futuro-de-lasComunicaciones-I.html

1.1.1 APIs Java Para Bluetooth

Las interfaces generadas para las aplicaciones a través de Java se conocen como APIs (Application Programming Interfaces), y son la base de las aplicaciones.

JSR 82, estandariza como desarrollar las aplicaciones Bluetooth usando Java, estas APIs para Bluetooth están orientadas para dispositivos que cumplan las características siguientes:

- Al menos 512K de memoria RAM y ROM libre.
- Conectividad a la red inalámbrica Bluetooth.
- Que tengan una implementación de J2ME Y CLDC.

J2ME permite interactuar con otros dispositivos Bluetooth de manera eficiente.

Los APIs JSR 82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones con aplicaciones Java Bluetooth.

El API ofrece 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.
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

Las APIs JSR-82 están divididas en dos partes: el paquete **javax.bluetooth** y el paquete **javax.obex**. Los dos paquetes son totalmente independientes (Fig.1.2). El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. La comunicación a través de javax.bluetooth es a bajo nivel: mediante flujos de datos o mediante la transmisión de arrays de bytes.

Por el contrario el paquete javax.obex permite manejar el protocolo de alto nivel OBEX (OBject EXchange). Este protocolo es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos. El protocolo OBEX es un estándar desarrollado por IrDA y es utilizado también sobre otras tecnologías inalámbricas distintas de Bluetooth.

1.1.2 CLDC (Connected Limited Device Configuration)

La plataforma principal de desarrollo de las APIs JSR-82 es J2ME, Las APIs han sido diseñadas para depender de la configuración CLDC. En J2ME se pueden crear aplicaciones Java conocidas como Midlets. El API de Bluetooth, usado en un MIDlet, posibilita el descubrimiento y registro de dispositivos, descubrimiento y registro de servicios, establecimiento de la comunicación, envío y recepción de datos, acceso y control sobre otros dispositivos que soporten Bluetooth.

La integración de Bluetooth con otras herramientas, como la plataforma Java, abre una

ventana inmensa de posibilidades para la creación de aplicaciones que pueden ser hechas más a la medida de pequeños usuarios.

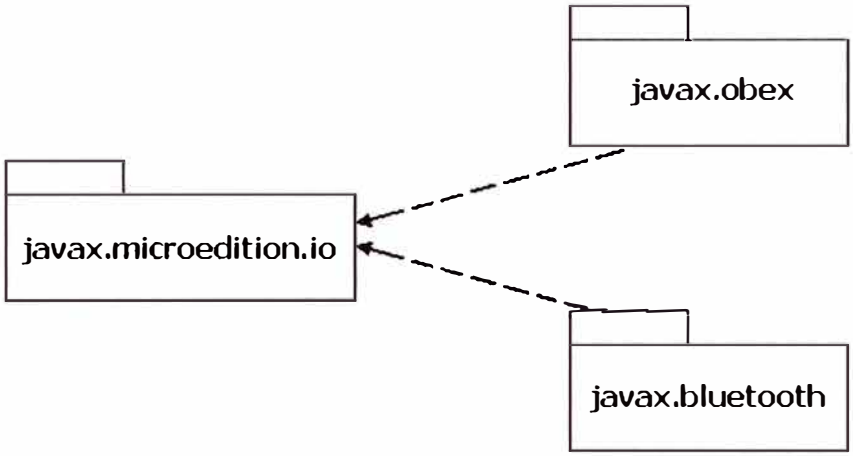


Fig. 1.2: Las APIs JSR-82 [2]

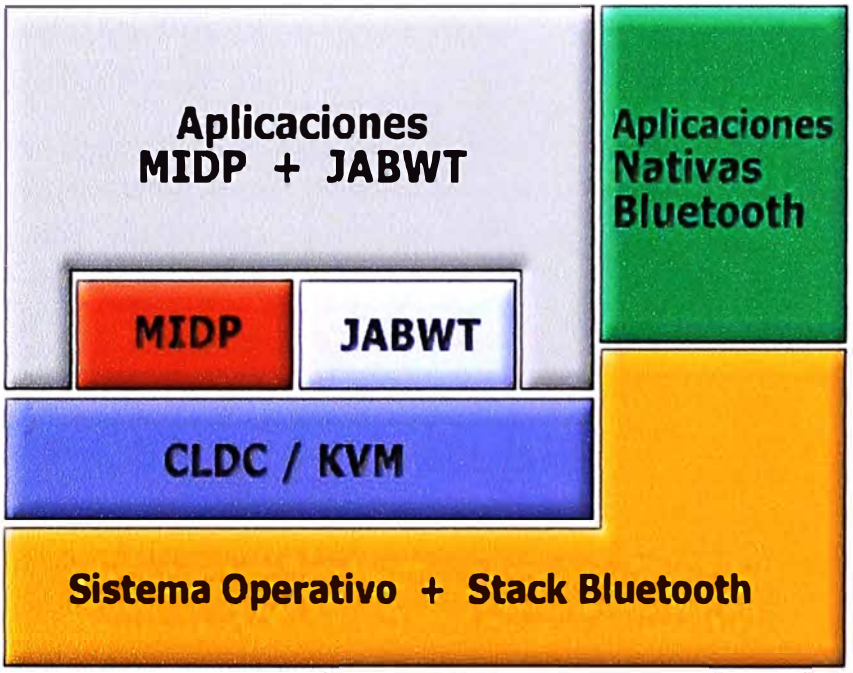


Fig. 1.3: Integración Bluetooth y J2ME[3]

La Fig. 1.3 muestra un ejemplo de cómo JSR 82 se ajusta dentro de la arquitectura de J2ME. Aunque aquí se muestra sobre un dispositivo con soporte para el perfil de Información del Dispositivo Móvil (MIDP, Mobile Information Device Profile).

1.2 Sistema Bluetooth Para Control de Dispositivos

Para hacer posible un control de dispositivos eléctricos usando Bluetooth, es necesario que estos también cuenten con la tecnología, o bien sean adaptados a otros que si lo hagan.

[2 Y 3] http://janoxmoraga.blogspot.com/2010/06/api-java-bluetooth-jsr-82.html

Para el control del acceso (Por ejemplo: un portero eléctrico) y la iluminación (Por ejemplo: una lámpara) de una habitación, se puede emplear módulos Bluetooth que les proporcionen esta característica.

Entonces no será necesario cambiar todos los dispositivos de la casa sino más bien adaptarlos.

El sistema implementado constaría de un dispositivo móvil (teléfono celular), un módulo Bluetooth y un dispositivo eléctrico común. Dentro del teléfono se implementaría una aplicación que establezca la conexión y sea la interfaz entre el usuario y el enlace. El módulo Bluetooth consta de un sistema de radio, una antena integrada, un pequeño procesador y una memoria flash. El módulo puede ser programado para interactuar con el dispositivo que se desea controlar. Una vez establecida la conexión, el equipo móvil será capaz de enviar comandos que controlarán el dispositivo eléctrico a través del módulo Bluetooth.

1.2.1 La pila de protocolos Bluetooth

La pila de protocolos Bluetooth se divide en dos zonas, cada una de las cuales se implementa en distintos procesadores:

- El módulo Bluetooth (hardware), encargado de las tareas relacionadas con el envío de información a través del interfaz de radiofrecuencia.
- El host Bluetooth (software), encargado de la parte relacionada con las capas superiores de enlace y aplicación.

Ambas zonas están comunicadas por el Interfaz de Controlador de Host (HCI). Sobre la capa de protocolos específicos de Bluetooth, cada fabricante puede implementar su capa de protocolos de aplicación propietarios. De esta forma, la especificación abierta de Bluetooth expande enormemente el número de aplicaciones que pueden beneficiarse de las capacidades que ofrece esta tecnología inalámbrica. Sin embargo, la especificación Bluetooth exige que, a pesar de la existencia de diferentes pilas de protocolos de aplicación propietarios, se mantenga la interoperabilidad entre dispositivos que implementen diferentes pilas.

Las pilas de protocolos Bluetooth (Fig 1.4) mas conocidas son Widcomm, Toshiba Bluetooth Stack. Microsoft Windows XP Bluetooth y IVT BlueSoleil Stack Linux dispone de las pilas de protocolos Bluetooth BlueZ, OpenBT y Affix, de Nokia.

a). Capa de banda base e interfaz de radio

En la base de la pila de protocolos Bluetooth se encuentran la capa de banda base y el interfaz de radio. Su función principal es permitir el enlace físico por radiofrecuencia (RF) entre unidades Bluetooth dentro de una piconet realizando tareas de modulación y demodulación de los datos en señales RF que se transmiten por el aire.

El nivel de banda base proporciona los dos tipos de enlace físico:

- Enlace asíncrono sin conexión (ACL, Asynchronous Connection-less):
- Conexiones simétricas o asimétricas punto-multipunto entre maestro y esclavo.
- Conexión utilizada para tráfico de datos.
- Sin garantía de entrega, se retransmiten paquetes.
- La máxima velocidad de envío es de 721 Kbps en una dirección 57.6 Kbps en la otra.
- Enlace síncrono orientado a conexión (SCO, Synchronous Connection-Oriented):
- Conexiones simétricas punto a punto entre maestro y esclavo.
- Conexión capaz de soportar voz en tiempo real y tráfico multimedia.
- Velocidad de transmisión de 64 KB/s.

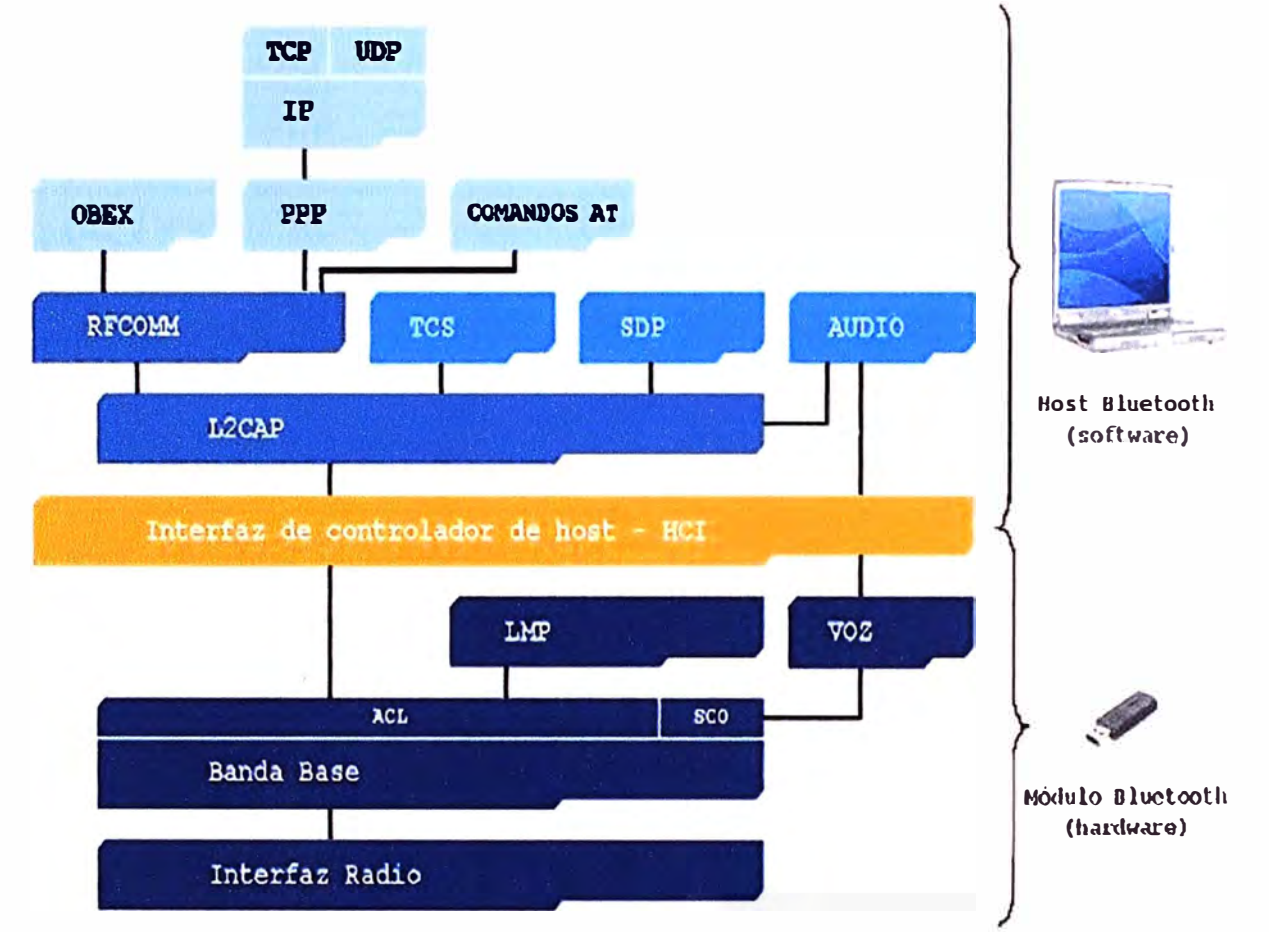


Fig. 1.4: Pila de protocolos Bluetooth[4]

b). Capa de Protocolo de Gestión de Enlace (LMP)

LMP (Link Manager Protocol) es el responsable de la configuración y control de enlace entre dispositivos Bluetooth, incluyendo el control y negociación del tamaño de los paquetes de banda base.

Cuando dos dispositivos Bluetooth se encuentran dentro del radio de acción del otro, el gestor de enlace (Link Manager) de cada dispositivo se comunica con su homólogo por medio de mensajes a través del protocolo LMP.

Estos mensajes realizan el establecimiento del enlace entre ambos dispositivos,

[4]http://www.seguridadmobile.com/bluetooth/especificacion-bluetooth/arquitectura-de-protocolo/index.html

incluyendo mecanismos de seguridad tales como la autenticación y cifrado, que comprende la generación, intercambio y comprobación de claves de enlace y de cifrado. Por medio de este intercambio de mensajes, LMP también controla los modos de administración de energía y los ciclos de trabajo de los dispositivos de radio Bluetooth, así como los estados de conexión de las unidades Bluetooth situadas dentro de una piconet. El gestor de enlace filtra e interpreta estos mensajes y no los propaga hacia los niveles superiores.

El gestor de enlace es un módulo software que se ejecuta en un microprocesador dentro de la unidad Bluetooth para gestionar la comunicación entre dispositivos. Cada dispositivo Bluetooth tiene su propio gestor de enlace, que se encarga de descubrir otros gestores de enlace remotos y comunicarse con los mismos para gestionar el establecimiento del enlace, la autenticación, la configuración y otras funciones.

Para realizar su papel de proveedor de servicios, el gestor de enlace hace uso de las funciones que ofrece el controlador de enlace (LC, Link Controller) subyacente, un módulo de supervisión que maneja todas las funciones de la banda base de Bluetooth y da soporte al gestor de enlace. El controlador de enlace envía y recibe datos, solicita la identificación del dispositivo emisor, autentica el enlace, establece el tipo de enlace SCO o ACL y determina el tipo de trama a utilizar en cada paquete.

Los mensajes que se intercambian entre los gestores de enlace toman la forma de unidades de datos de protocolo (PDU, Protocol Data Unit). Estos datos tienen una prioridad más alta que los datos del usuario, para que un mensaje que el gestor de enlace necesite enviar no se vea retardado por el tráfico L2CAP.

c). Capa de Interfaz de Controlador de Host (HCI)

La capa HCI (Host Controller Interface) actúa como frontera entre las capas de protocolo relativas al hardware (módulo Bluetooth) y las relativas al software (host Bluetooth). Proporciona una interfaz de comandos para la comunicación entre el dispositivo y el firmware del módulo Bluetooth y permite disponer de una capa de acceso homogénea para todos los módulos Bluetooth de banda base, aunque sean de distintos fabricantes.

Una de las tareas más importantes del interfaz HCI es el descubrimiento de dispositivos Bluetooth que se encuentren dentro del radio de cobertura. Esta operación se denomina consulta o inquiry y funciona del siguiente modo:

- Inicialmente, el dispositivo origen envía paquetes inquiry y se mantiene en espera de recibir respuestas de otros dispositivos presentes en su zona de cobertura.

- Si los dispositivos destino están configurados en modo visible (discoverable) se encontrarán en estado inquiry_scan y en predisposición de atender estos paquetes. En este caso, al recibir un paquete inquiry cambiarán a estado inquiry_response y enviarán

una respuesta al host origen con sus direcciones MAC y otros parámetros.

- Los dispositivos que estén configurados en modo no visible (non discoverable) se encontrarán en modo inquiry_response y, por tanto, no responderán al host origen y permanecerán ocultos.

d). Capa de protocolo de adaptación y control del enlace lógico (L2CAP)

La especificación Bluetooth incluye el protocolo L2CAP (Logical Link Control and Adaptation Protocol), que se encarga de la multiplexación de protocolos, ya que el protocolo de banda base no soporta un campo tipo para identificar el protocolo de nivel superior al que quiere transmitir la información, por ejemplo SDP, RFCOMM y TCS.

Otra función que se realiza en el nivel L2CAP es la segmentación y recomposición de paquetes, necesaria para permitir la utilización de protocolos que utilicen paquetes de mayor tamaño que los soportados por la capa de banda base. Los paquetes L2CAP de gran tamaño se deben segmentar en múltiples paquetes de formato banda base más pequeños antes de su transmisión. En el lado del receptor, los paquetes de banda base se recomponen en paquetes L2CAP más grandes tras comprobar su integridad.

El proceso de establecimiento de la conexión L2CAP también permite el intercambio de información referente a la calidad de servicios (QoS) que se espera entre dos dispositivos Bluetooth. La implementación L2CAP en cada uno de los extremos controla los recursos utilizados por el protocolo y se asegura de que se cumplen los contratos de calidad de servicio.

La especificación L2CAP está definida únicamente para enlaces asíncronos sin conexión (ACL) y no puede existir más que un enlace entre dos dispositivos.

e). Capa de protocolo de descubrimiento de servicios (SDP)

El descubrimiento de servicios hace referencia a la capacidad de buscar y encontrar servicios disponibles en dispositivos Bluetooth. A través de los servicios, dos dispositivos pueden ejecutar aplicaciones comunes e intercambiar datos.

El protocolo SDP (Service Discovery Protocol) permite a una aplicación cliente obtener información sobre servidores SDP disponibles en otros dispositivos Bluetooth cercanos, enumerar los servicios que ofrecen y las características de dichos servicios. Después de haber localizado los servicios disponibles en un dispositivo, el usuario puede elegir aquel de ellos que resulte más apropiado para el tipo de comunicación que desea establecer.

Un servicio es cualquier entidad que puede ofrecer información, ejecutar una acción o controlar un recurso. Un servicio puede estar implementado como hardware, software o una combinación de hardware y software.

f). Capa RFCOMM

El protocolo RFCOMM (Radio Frequency Communication) es un protocolo de

emulación de línea serie basado en el estándar ETSI TS 07.10. Proporciona una emulación de los puertos serie RS-232 sobre el protocolo L2CAP.

Este protocolo de "sustitución de cable serie" emula las señales de control y datos RS-232 sobre la banda base, proporcionando capacidades de transporte a los servicios de niveles superiores que utilizan el cable serie como mecanismo de transporte.

Para los propósitos de RFCOMM, un camino de comunicación directa involucra siempre a dos aplicaciones que se ejecutan en dos dispositivos distintos extremos de la comunicación. Entre ellos existe un segmento que los comunica, en este caso, un enlace Bluetooth desde un dispositivo al otro. RFCOMM pretende soportar aquellas aplicaciones que utilizan los puertos serie de los dispositivos donde se ejecutan.

RFCOMM es un protocolo de transporte sencillo que soporta hasta 9 puertos serie RS-232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos Bluetooth.

g). Comandos AT

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un terminal módem. Los comandos AT se denominan así por la abreviatura de attention.

El juego de comandos AT fue desarrollado en 1977 por Dennis Hayes como un interfaz de comunicación con un módem para poder configurarlo y proporcionarle instrucciones, tales como marcar un número de teléfono. Más adelante, fueron las compañías Microcomm y US Robotics las que siguieron desarrollando y expandiendo el juego de comandos hasta universalizarlo.

La telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales. De esta forma, todos los teléfonos móviles GSM poseen un juego de comandos AT específico que sirve de interfaz para configurar y proporcionar instrucciones a los terminales.

El juego de comandos AT puede encontrarse en la documentación técnica de los terminales GSM y permite acciones tales como realizar llamadas de datos o de voz, leer y escribir entradas en la agenda de contactos y gestión de mensajes SMS, además de muchas otras opciones de configuración del terminal.

1.3 La especificación Bluetooth

Bluetooth es una especificación que define un estándar global de comunicaciones inalámbricas para redes de área personal, que permite la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia en entornos de comunicaciones móviles y fijas. La especificación Bluetooth está recogida por el grupo de trabajo 802.15.1 del IEEE.

1.3.1 Ventajas de Bluetooth

Es fácil reconocer las ventajas sobre otros tipos de comunicación inalámbrica fijándose en la descripción técnica anterior. Además, los objetivos que la tecnología Bluetooth persigue son los que permitieron crear expectativas en la factibilidad de este proyecto, algunos de estos se resumen a continuación.

a). Bluetooth es una tecnología disponible mundialmente

La especificación Bluetooth está disponible de forma gratuita para empresas afiliadas de todo el mundo. Fabricantes de distintos sectores industriales están implantando esta tecnología en sus productos para reducir el número de cables y lograr conexiones sin interrupciones, transmitir sonido estéreo, transferir datos o para establecer comunicaciones de voz.

La tecnología Bluetooth funciona en la banda de 2.4 GHz, una de las bandas de radio industrial, científica y médica (ISM) que no requiere licencia. Por tanto, no existen gastos asociados al uso de la tecnología Bluetooth, excepto aquellos directamente vinculados al dispositivo.

Desde que se publicó la primera especificación Bluetooth en 1999, más de 4.000 empresas se han afiliado al Grupo de Interés Especial, Special Interest Group (SIG), de Bluetooth, y el número de productos Bluetooth disponibles en el mercado se multiplica con rapidez.

b). Bluetooth funciona en una amplia gama de dispositivos

La tecnología Bluetooth está disponible en todo un abanico de dispositivos, desde teléfonos móviles hasta instrumental médico, pasando por automóviles, electrodomésticos, etc. y abarca una gran variedad de usuarios, desde consumidores particulares hasta mercados industriales. Su bajo consumo de energía, reducido tamaño y el escaso costo de los chips permite emplear la tecnología Bluetooth hasta en los dispositivos más pequeños.

c). Bluetooth es fácil de usar

El estándar Bluetooth es una tecnología que no necesita una infraestructura fija y es sencilla de instalar y configurar. La conexión se realiza sin cables. El proceso resulta muy sencillo para los nuevos usuarios: una vez adquirido el producto Bluetooth, basta con comprobar los perfiles disponibles y conectarlo a otro dispositivo Bluetooth con los mismos perfiles. El usuario lleva consigo en todo momento su red de área personal (PAN) e incluso puede conectarse a otras.

d). El emisor de radio Bluetooth consume poca energía y puede integrarse a equipos alimentados por baterías.

Actualmente, la información ya no discurre por cables, y debe ser enviada a su destinatario de forma segura, sin ser interceptada. Los estándares que rigen las comunicaciones inalámbricas están evolucionando y presentan varios formatos para

garantizar seguridad a los usuarios.

Desde su creación, Bluetooth se ha esforzado para que los usuarios de este estándar global puedan sentirse plenamente seguros al conectarse. Cuenta con un grupo de expertos en seguridad formado por ingenieros de las empresas afiliadas. Este grupo suministra información decisiva sobre cuestiones de seguridad y sus consejos se tienen en cuenta en el proceso de desarrollo de las especificaciones inalámbricas de la tecnología Bluetooth.

Al estar disponible en todo el mundo a través de la banda ISM abierta de 2.4 GHz, la fiabilidad fue prioritaria desde un primer momento. Gracias a la función de salto adaptivo de frecuencia (AFH) se limita las interferencias de otras señales.

Bluetooth cuenta con seguridad integrada, como el cifrado de 128 bits y la autenticación mediante código PIN. El número de identificación personal (PIN) es un código alfanumérico compuesto de cuatro o más caracteres que se asocia temporalmente a un producto para realizar un emparejamiento.

CAPITULO II
DESCRIPCIÓN DE JAVA 2 PLATAFORM MICRO EDITION

Actualmente se usa la programación orientada a objetos, el software que se usa en el presente informe de suficiencia es el JAVA2ME, un lenguaje de programación creada por la extinta empresa SUN MICROSYSTEM, es un lenguaje de alto nivel que es de uso libre.

Java2ME es un estándar de lenguaje de programación para pequeños dispositivos como lo son los teléfonos celulares, es por eso que se ha elegido este lenguaje de programación para el desarrollo de este informe de suficiencia.

Se creará una aplicación Java2ME para un teléfono celular y para la PC, que permita realizar una conexión Bluetooth con la PC y sobre la cual se simula el control de los dispositivos eléctricos.

El lenguaje Java es reconocido mundialmente por su capacidad de adaptación a cualquier ambiente. Los desarrolladores de Java optaron por dotar al lenguaje de un conjunto de capacidades orientadas a objetos que permitiesen añadir librerías de apoyo a medida que fuera necesario y en función de las necesidades tecnológicas.

Este informe se enfoca en teléfonos móviles que sin importar la marca pueden correr aplicaciones Java2ME. Las características concretas de este tipo de dispositivos han obligado a los desarrolladores de Java a construir un subconjunto del lenguaje y a reconfigurar sus principales bibliotecas para permitir su adaptación a un entorno con poca capacidad de memoria, poca velocidad de proceso y pantallas de reducidas dimensiones.

Para esto se creó una nueva plataforma de desarrollo y ejecución sobre la que se centra este proyecto: Java ME, conocida inicialmente como Java 2 Platform Micro Edition.

En este capítulo se revisarán las características del lenguaje Java, principalmente de la plataforma Java ME, que será usada para crear una aplicación que maneje tecnología Bluetooth y pueda ser usada por un teléfono celular compatible con Java

2.1 Descripción del lenguaje 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 Standard 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.

2.1.1 Orientada 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 comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de

desarrollo. La decisión de separarlos fue debida a que no todas las características de la Plataforma Java son necesarias para el desarrollo de aplicaciones.

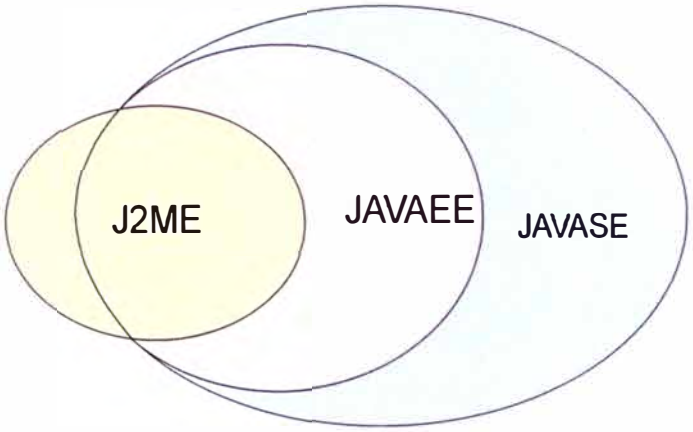


Fig. 2.1 Entorno Java[5]

Los entornos específicos que están definidos para la Plataforma Java (Fig. 2.1) son los que se resumen a continuación:

- Java SE (J2SE), Java Standard Edition, por decirlo de alguna manera, la base de la tecnología Java. Permite el desarrollo de applets (aplicaciones que se ejecutan en un navegador web) y aplicaciones independientes (stand-alone). Java SE es el heredero directo del Java original.
- Java EE (J2EE), Java Enterprise Edition, está basado en Java SE, pero añade una serie de características necesarias en entornos empresariales, relativos a redes, acceso a datos y entrada/salida que requieren mayor capacidad de proceso, almacenamiento y memoria.
- Java ME (J2ME), Java 2 Micro Edition, es un Subconjunto de Java SE para dispositivos con recursos más limitados. Este entorno es el más adecuado para realizar aplicaciones en teléfonos, por tal razón, serán objeto de una revisión más profunda.

Cada entorno usa las librerías apropiadas para su área de desarrollo, estas librerías vienen agrupadas en lo que se conoce como APIs.

2.1.2 APIs (Aplication programming interfaces)

Un API Java es una Interfaz de Programación de Aplicaciones provista por los creadores del lenguaje Java, y que da a los programadores los medios para desarrollar aplicaciones Java. A través de un procedimiento formal, los expertos de la industria desde un concepto general pueden dar forma futura al estándar, un API proporciona un conjunto de funciones de uso general como dibujar ventanas o iconos en la pantalla.

Este proceso se conoce como JCP, Java Community Process y conlleva el uso de documentos oficiales que describen las especificaciones y tecnologías propuestas para añadirse en la plataforma Java, se conocen como Java Specification Request (JSR).

---

[5]http://desarrollomoviles.blogspot.com/2008/11/y-llegamos-al-j2me.htmlgeneral

Existen más de 300 JSR en general y más de 80 para JavaME, algunos ejemplos de ellos se pueden apreciar en la Tabla 2.1

Tabla 2.1 Descripción de algunos JSR[6]

JSR	DESCRIPCIÓN
30	Java METM Connected, Limited Device Configuration
	Esta especificación definirá un estándar para la configuración de JavaTM 2 Platform Micro Edition (Java METM) para dispositivos pequeños con recursos limitados conconexión (CLDC)
36	Connected Device Configuration
	La configuración de dispositivos con conexión (CDC) provee las bases de JavaTM 2 platform Micro Edition para dispositivos que tienen un procesador mínimo de 32-bits amplia memoria.
37	Mobile Information Device Profile for the Java METM Platform
	Esta especificación definirá un perfil que extenderá y ampliará el Java METM Connected, Limited Device Configuration" (JSR-000030), habilitando el desarrollo de aplicaciones para aparatos de información móviles y dispositivos de comunicación por voz.
46	Foundation Profile
	El Foundation Profile es un conjunto de APIs pensando para aplicaciones corriendo en dispositivos pequeños que tienen algún tipo de conexión a red.
62	Personal Profile Specification
	El Java METM Personal Profile provee el ambiente Java ME para aquellos dispositivos que tengan una necesidad de contar con un alto grado de conectividad a Internet y fidelidad Web.
68	Java METM Platform Specification
	Esta especificación definirá la siguiente mejor revisión del JavaTM 2 Platform, Micro Edition.
75	PDA Optional Packages for the Java METM Platform
	Este JSR produce dos paquetes opcionales separados por características comúnmente encontradas en PDAs y otros dispositivos móviles Java ME: uno para acceder a datos PIM y otro para acceder a sistemas de ficheros.
82	JavaTM APIs for Bluetooth
	Bluetooth es un estándar importante que está emergiendo para la integración wireless de dispositivos pequeños. La especificación estandariza

	un conjunto de APIs de Java, para los dispositivos que soportan Java, que permiten la integración a un ambiente Bluetooth
118	Mobile Information Device Profile 2.0
	Esta especificación definirá un perfil que ampliará y extenderá el "Java METM Mobile Information Device Profile" (JSR-000037).
134	JavaTM Game Profile
	Define un perfil Java 2 Micro Edition con fines de desarrollo de juegos dirigidos a los consumidores de dispositivos de juegos computadores de escritorio.
135	Mobile Media API
	Especifica un API multimedia para Java METM, para el sencillo y fácil acceso para el control básico de audio y a los recursos multimedia, al mismo tiempo que hace frente a la escalabilidad y el apoyo de características más sofisticadas.
139	Connected Limited Device Configuration 1.1
	Esta especificación definirá una versión revisada del Java METM Connected, Limited Device Configuration (CLDC).
205	Wireless Messaging API 2.0
	Este JSR extenderá y ampliará el API para mensajes wireless (JSR-000120)
216	Personal Profile 1.1
	Este JSR actualizará la especificación del actual Personal Profile (JSR-62) para reflejar el API J2SETM 1.4.
218	Connected Device Configuration (CDC) 1.1
	Este JSR define una revisión de la especificación Java ME CDC. Este JSR provee actualizaciones para el núcleo existente basadas en el J2SE, v1.4, no gráficas, para dispositivos pequeños.
246	Device Management API
	Habilita a las aplicaciones Java METM el acceso a dispositivos de gestión de implementaciones.
927	Java TV™ API 1.1
	El mantenimiento de la especificación Java TVTM.

Los teléfonos móviles tienen incluidos algunos de estos APIS. Para ser usados en este proyecto es importante que cuenten principalmente con los siguientes:
JSR82 / Bluetooth; JSR118 / Mobile Information Device Profile 2.0 y JSR139 / Connected

[6] Referencia: http://www.jcp.org/en/jsr/all

Limited Device Configuration 1.1.

Actuales dispositivos Bluetooth incluyen APIs como el JSR135 que incluye el manejo multimedia. Para uso de otras aplicaciones como ejemplo es el JSR-62, con este API se logra aplicaciones para la conectividad a internet.

2.2 Plataforma Java Micro Edition (Java ME)

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.

La elección de un entorno Java específico depende del tipo de aplicación y del tipo de dispositivo en el cual se ejecutará. Java Platform, Micro Edition (Java ME) es una colección de tecnologías y especificaciones para crear una plataforma que se ajuste a las necesidades de dispositivos móviles y que se pueden combinar para crear un completo entorno de ejecución de Java.

El entorno de ejecución estará compuesto por: una máquina virtual, una configuración, un perfil y otros paquetes opcionales.

En la Fig. 2.2 se pueden observar la Plataforma Java y sus diferentes entornos, incluyendo los componentes del entorno Java ME.

2.2.1 Maquinas virtuales para JAVA ME

Una Máquina virtual Java (en inglés Java Virtual Machine, JVM) es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al Hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode, como el sistema sobre el que pretende ejecutar.

Por lo tanto la JVM es un entorno donde no importa donde se realice un código ni la plataforma donde se ejecutará, ya que la JVM será capaz de interpretar y ejecutar, previamente debe estar instalada la JVM. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una máquina virtual Java en concreto, siendo ésta la que en última instancia convierte de código bytecode a código nativo del dispositivo final.

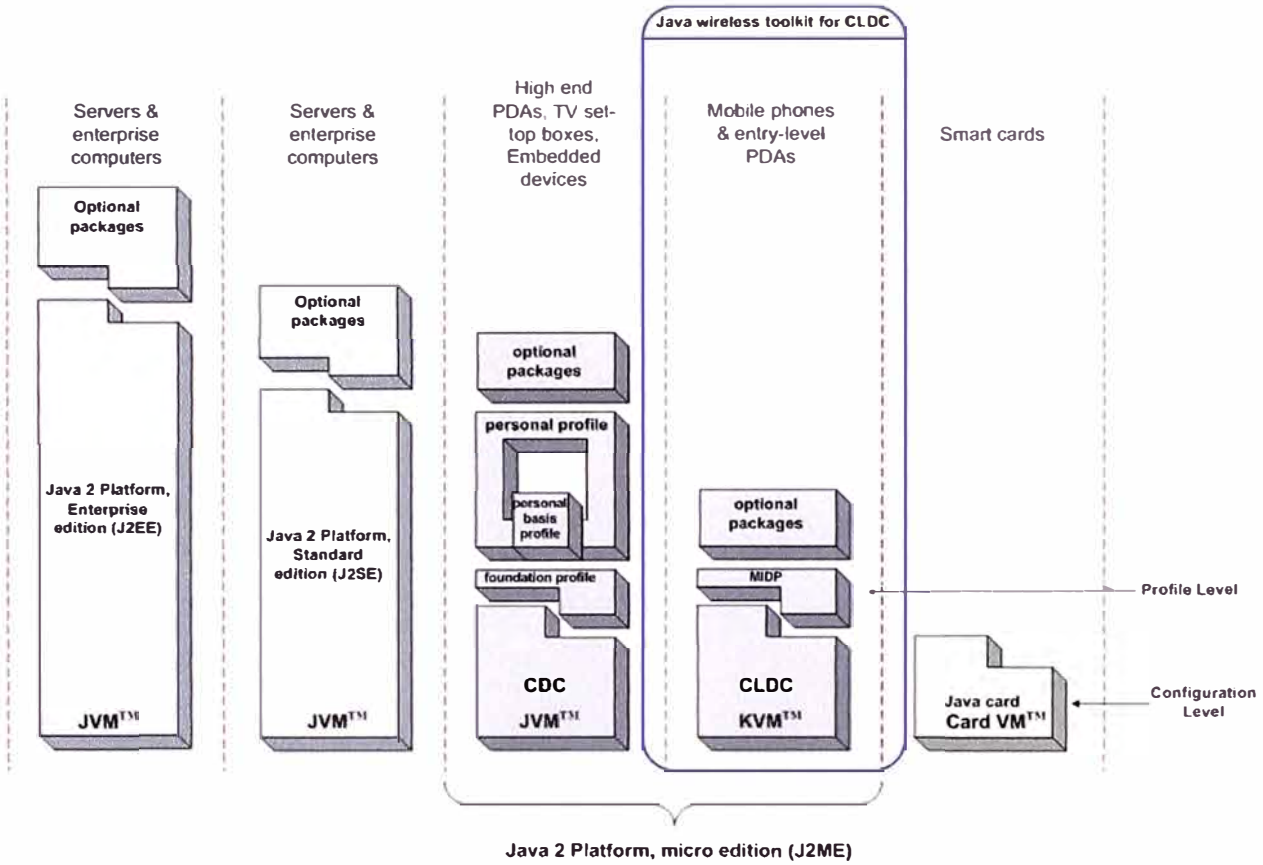


Fig. 2.2: Plataforma Java [7]

La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas. Ente los más usados se tienen:

a). Kilobyte Virtual machine (KVM)

Es una máquina virtual desarrollada por Sun Microsystems, derivados de la máquina virtual Java especificación. El KVM fue escrito desde cero en C. Está diseñado para dispositivos pequeños y cuentan con memoria reducida. Soporta un subconjunto de las características del extremo superior de JVM. Por ejemplo, un KVM no puede apoyar las operaciones de coma. El 'K' está en KVM para kilobytes, lo que significa que se ejecuta en el KVM kilobytes de memoria en lugar de megabytes.

La KVM se utiliza en teléfonos y dispositivos móviles, estos a su vez tienen

---

[7] http://ikt.hia.no/aml/public-projects/J2ME/Rams-J2ME/J2ME.html

memoria y potencia de procesador limitados. El fabricante del dispositivo móvil debe instalar la KVM, el usuario final no tiene la posibilidad de descargarlo e instalarlo.

b). CVM

Corresponde a la máquina virtual con algunas características adicionales a la KVM, CVM significa Compact Virtual Machine. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y con alrededor de 2Mb o más de memoria RAM. Soporta las mismas características que J2SE.

2.3 Configuraciones Java ME

J2ME presenta dos configuraciones: CLDC y CDC. La primera se dedica a dispositivos con estrictas limitaciones de memoria, capacidad de cálculo, consumo y conectividad de red. Por otro lado, CDC se encarga de dispositivos con más potencia. Parte de CLDC es un subconjunto de CDC, por lo que la portabilidad de aplicaciones se puede conseguir cuando nos movemos de un entorno más restringido a otro más rico. De la misma manera, y siguiendo en el hilo de la portabilidad, una aplicación en J2ME podrá ejecutarse en J2SE normalmente, salvo que se utilicen las bibliotecas específicas de J2ME.

Existen dos configuraciones dentro de Java ME:

- CDC, orientada a dispositivos con menos limitaciones.

CLDC, orientada a dispositivos con limitaciones computacionales y de memoria considerables.

2.3.1 CDC

Esta configuración se ha desarrollado para dispositivos con 2 MB o más de memoria disponible para la plataforma, incluyendo RAM y memoria flash o ROM. Estos dispositivos requieren todas las características y funcionalidades de la JVM e incluyen teléfonos móviles de última generación, asistentes personales (PDAs), terminales de punto de venta (TPVs), etc. Permiten tanto redes con conexión intermitente o de alta velocidad, sin cablear (wireless) o cableadas. La CDC incorpora el perfil Foundation (Foundation Profile), aunque están disponibles los perfiles Personal y Personal Basis para aplicación que requieren una interfaz gráfica de usuario compleja o Web. La plataforma CDC para J2ME incorpora:

- Una máquina virtual Java (Java Virtual Machine, JVM) completa compatible con J2SE.
- Las bibliotecas de clase y APIs mínimas para que el sistema funcione.
- Soporte completo para carga de clases, gestión de subprocesos (threads) y mecanismos de seguridad.

2.3.2 CLDC

CLDC es una especificación general para un amplio abanico de dispositivos, que van desde teléfonos móviles hasta PDAs y otros, que se caracterizan por sus

limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Algunas de estas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño.

La configuración CLDC define características incluidas por el lenguaje Java, las funcionalidades de la máquina virtual Java, las APIs (Application Programming Interfaces) necesarias para el desarrollo de aplicaciones y un subconjunto muy pequeño de las librerías de J2SE.

Los dispositivos que usan CLDC se caracterizan por:

- Disponer como mínimo de 128 Kb de memoria para la Máquina Virtual
- Tener un procesador de 16 o 32 bits con al menos 25 MHz de velocidad.
- Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

2.3.3 Librerías incluidas en CLDC

Las librerías en general entregan al programador un conjunto de funciones para realizar tareas comunes, manejar elementos y objetos, funciones y accesos. El numero de librerías incluidas en CLDC es menor que en CDC, esto debido al tipo de dispositivos que usan esta aplicación y su alcance, ver tabla 2.2.

Perfiles Para CDC:

- Foundation Profile: Incluye gran parte de J2SE, pero excluye interfaz gráfica AWT y Swing.
- Personal Profile: Soporte gráfico AWT. Requiere Foundation Profile.
- RMI: Requiere Foundation Profile. Subconjunto de J2SE RMI.

Tabla 2.2: Librerías incluidas en CLDC

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

2.4 Perfiles de JAVA ME

Un perfil especifica las características de un dispositivo según su tipo, tomando en cuenta su uso y las aplicaciones que se pueden ejecutar en estos, por ejemplo: teléfonos celulares, agendas electrónicas, electrodomésticos, audífonos y manos libres, etc. La parte gráfica es un elemento importante a definir en un perfil, no es lo mismo la interfaz gráfico para un teléfono celular que una interfaz táctil, por ejemplo.

El Perfil especifica estas características mediante APIs, mientras que la Configuración hace algo similar a nivel de grupos de dispositivos. Así, un perfil siempre debe estar construido sobre una configuración determinada. En la Figura 2.3 se encuentra un esquema de la arquitectura del entorno Java ME.

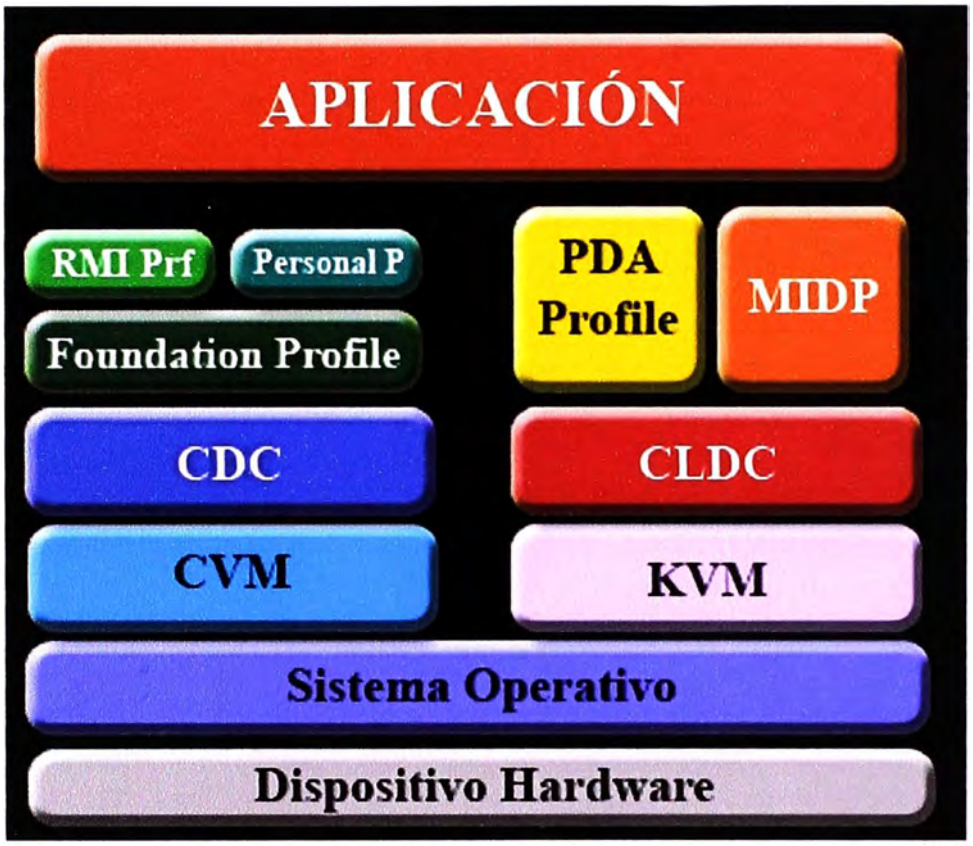


Fig.2.3: Arquitectura J2ME [8]

En resumen, un Perfil no es nada más que un conjunto de APIs que le dan a una configuración un carácter más específico.

Para la configuración CLDC se tienen los siguientes perfiles:

- PDA Profile (PDAP): Está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero.
- Mobile Information Device Profile (MIDP): Este perfil está construido sobre la configuración CLDC y define un entorno Java para teléfonos celulares, buscapersonas y dispositivos similares con recursos limitados.

Los dispositivos que usan MIDP deben tener las siguientes características:

---

[8]http://caraballomaestre.blogspot.com/2009/05/introduccion-j2me.html

a. Suficiente memoria para correr las aplicaciones.

b. Un display de alrededor de 96 por 56 píxeles mínimo.

c. Un teclado o una pantalla táctil.

d. Capacidad de conexión inalámbrica.

El perfil MIDP da a la aplicación la capacidad de controlar la interfaz de usuario, el almacenamiento, las conexiones de red, etc., en la tabla 2.3 se describe las principales paquetes del MIDP.

Tabla 2.3: Paquetes incluidos en el perfil MIDP [9]

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

2.5 Paquetes Opcionales

Los paquetes opcionales son APIs, similares a los que tienen incluidos los Perfiles y las Configuraciones, que pueden ser empaquetados con J2ME conjuntamente para crear una gran pila de software.

Entre los paquetes opcionales para dispositivos móviles se encuentra el API de Bluetooth, JSR 82, que incluye los recursos necesarios para crear aplicaciones que permitan controlar el Bluetooth del dispositivo móvil y por lo tanto, será usado en el presente proyecto.

2.5.1 APIs de Java para Bluetooth JSR-82

JSR-82, Java Specification Request for Bluetooth, es el nombre oficial JCP para el conjunto de APIs para Bluetooth. Los APIs JSR 82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth.

JSR-82 permite, entre otras, las siguientes operaciones:

a. Registro de servicios.

---

[9] Referencia: http://www.jcp.org/en/jsr/all

b. Descubrimiento de dispositivos y servicios.

c. Establecer conexiones entre dispositivos.

d. Usar dichas conexiones para mandar y recibir datos

e. Manejar y controlar las conexiones de comunicación.

f. Ofrecer seguridad a dichas actividades.

2.6 Aplicaciones JAVA ME

Este Informe de suficiencia incluye el desarrollo de una aplicación Java basada en el perfil MIDP sobre la configuración CLDC y que además usa el API de Bluetooth.

Para conseguir una aplicación Java ME se debe pasar por varias fases desde la edición del código hasta poder tener listo el MIDlet que se instalará en el dispositivo móvil. Las fases de desarrollo de un MIDlet se resumen a continuación.

Midlet: Midlet es un programa desarrollado en lenguaje de programación Java para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java 2 MicroEdition (Java 2 ME). Generalmente son aplicaciones que corren en un teléfono móvil. Está desarrollada bajo la especificación MIDP (perfil para información de dispositivo móvil).

2.6.1 Fase de edición

El código fuente o código de programación se puede editar en programas especializados como NetBeans, Eclipse, etc. ampliamente usados para crear aplicaciones de Java a cualquier nivel; o simplemente en un editor de texto cualquiera como Notepad de Windows. El archivo debe llevar la extensión .java que lo identifica como código de programación Java para pasar a las siguientes fases.

2.6.2 Fase de compilación

Para el proceso de compilación también se pueden usar cualquiera de los programas citados anteriormente. Además, se pueden usar otras herramientas creadas específicamente para la programación de equipos móviles, por ejemplo: "Sun One Studio" de Sun Microsystems o "JBuilder" de Borland.

Para usar cualquiera de estas herramientas, es imprescindible tener instalada una Máquina Virtual Java en el computador, la aplicación de java virtual se puede descargar desde la pagina oficial de Oracle, en la sección de downloads.

Sin importar el compilador que se use, las aplicaciones Java estarán organizadas en proyectos, el resultado de un proyecto será el MIDlet.

Los proyectos están organizados con una estructura de directorios donde se puede encontrar los archivos generados.

En la Tabla 2.4 se resumen el contenido de las carpetas generadas en un proyecto.

2.6.3 Fase de preverificación

Cuando se realiza la compilación de un programa Java, se genera un archivo con la extensión .class por cada clase, que contiene el código intermedio que es capaz de ejecutar la Máquina Virtual de Java.

Tabla 2.4: Carpetas generadas

Carpeta	DESCRIPCION
Bin	Aquí se guardan los archivos JAD y JAR cuando el proyecto es empaquetado. Esta carpeta contiene también el archivo de manifiesto
classes	Esta carpeta es usada por el compilador para almacenar los archivos de clases .class.
Lib	Aquí se guardan las librerías usadas en el proyecto
Res	Las imágenes, sonido y más recursos van en esta carpeta. Luego estos son empaquetados dentro del archive JAR.
Src	En esta carpeta se encuentran los códigos fuente, en archivos .java.
tmpclasses	En esta carpeta, el compilador crea archivos temporales usados para la simulación y depuración.
tmpsrc	En esta carpeta, el compilador crea archivos temporales usados para la simulación y depuración.

Estos archivos son almacenados en el directorio classes. En esta fase se realizará la preverificación de clases. Esta fase busca generar dicho fichero .class sin errores.

La compilación y preverificación se hace a través del comando Build en el J2ME Wireless Toolkit o con la tecla F11 en NetBeans.

2.6.4 Fase de depuración y ejecución

Una vez que se han completado los procesos de compilación y preverificación, el MIDlet se puede ejecutar en un simulador. En fases de prueba se deberá ejecutar el MIDlet sobre un emulador. Se lo puede hacer a través del comando Run en el Sun Java Wireless Toolkit, lo mismo en el NetBeans.

2.6.5 Fase de empaquetamiento

Una vez que se han realizado todas las pruebas y simulaciones en el computador, la aplicación deberá ser empaquetada. Los compiladores empaquetan la aplicación y dan como resultado dos tipos de archivos: los archivos JAR y los archivos

JAD. Un archivo JAR es un archivo comprimido (en formato ZIP) que contiene las clases (.class) que ha generado la compilación de un programa, puede contener los recursos necesarios para el MIDlet como sonidos, gráficos, etc. y además, contiene un archivo de tipo manifest.mf.

Este archivo de manifiesto contiene información resumida sobre las clases contenidas en el archivo JAR.

El segundo archivo necesario para la instalación de MIDlets son los archivos JAD. El archivo JAD contiene información necesaria para la instalación de los MIDlets contenidos en el archivo JAR. Un archivo puede contener más de un MIDlet. Cuando ocurre esto, se estaría hablando de un MIDlet suite.

CAPITULO III
DESARROLLO DE LAS APLICACIÓNES BLUETOOTH PARA UN CELULAR Y UNA COMPUTADORA

Las aplicaciones del cliente (celular) y la del servidor (computadora) son generadas en el software NetBeans versión 6.9.1. En la Fig. 3.1 se observa el diagrama de flujo de la aplicación Bluetooth.

3.1 Aplicación bluetooth cliente

3.1.1 Programación de la interfaz de usuario

La aplicación Java ME estará sustentada principalmente en dos APIs, por un lado CLDC que hereda algunas de las clases de J2SE, y MIDP que añade nuevas clases que permitirán crear interfaces de usuario.

Las clases más importantes de J2SE que ofrece CLDC son las siguientes:

a. java.lang
b. java.util
c. java.io

Además MIDP añade los siguientes paquetes:

a. javax.microedition.midlet
b. javax.microedition.lcdui
c. javax.microedition.io
d. javax.microedition.rms

El paquete javax.microedition.midlet, es el más importante de todos. Sólo contiene a la clase MIDlet, que ofrece un marco de ejecución para aplicaciones sobre dispositivos móviles. El paquete javax.microedition.lcdui ofrece una serie de clases e interfaces de usuario, en la Fig. 3.2 se muestra el inicio de la implementación del código para el cliente (celular).

a). Elementos de la Interfaz de usuario

Comand es un elemento que permite interaccionar con el usuario y le permite introducir comandos. Están disponibles los siguientes tipos de comandos, en la tabla 3.1.

Se agrega los comandos al código del programa pero antes se tiene que implementar la interface de CommandListener al Midlet para que pueda leer los comandos, en la Fig 3.3 se muestra en el NETBEANS parte del código.

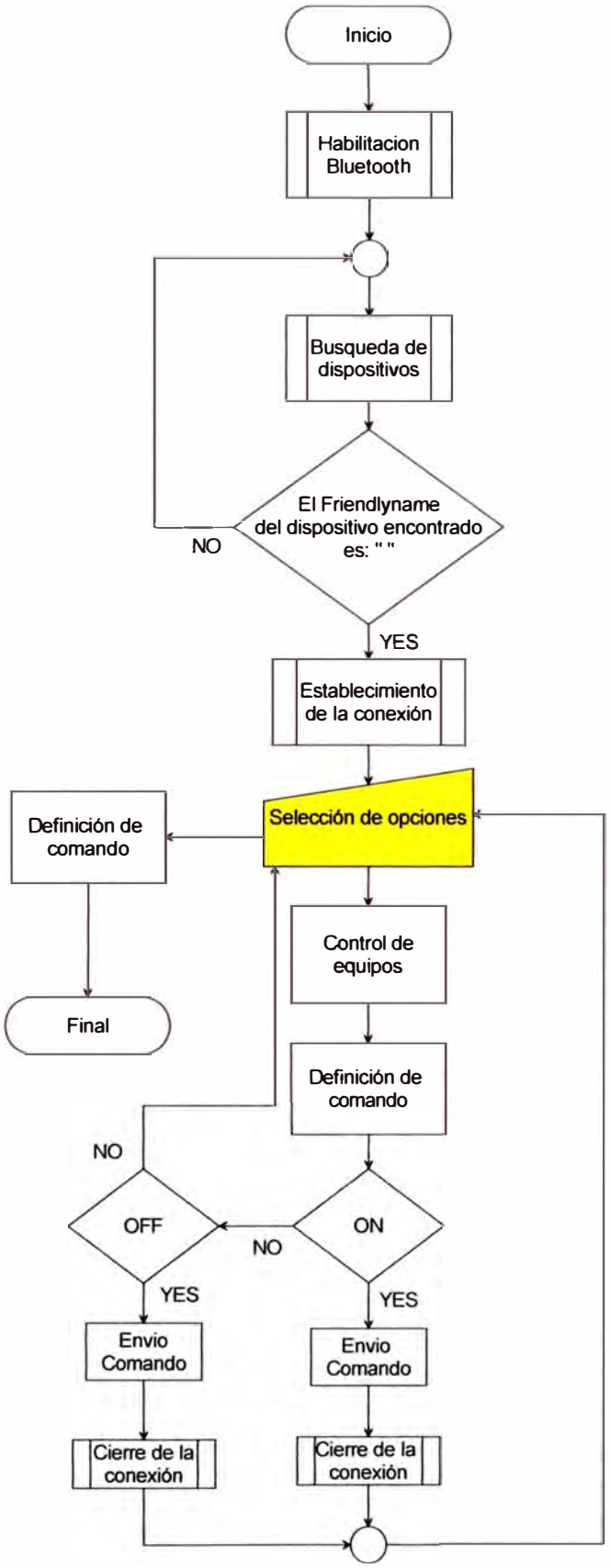


Fig 3.1: Diagrama de flujo de la aplicación Bluetooth

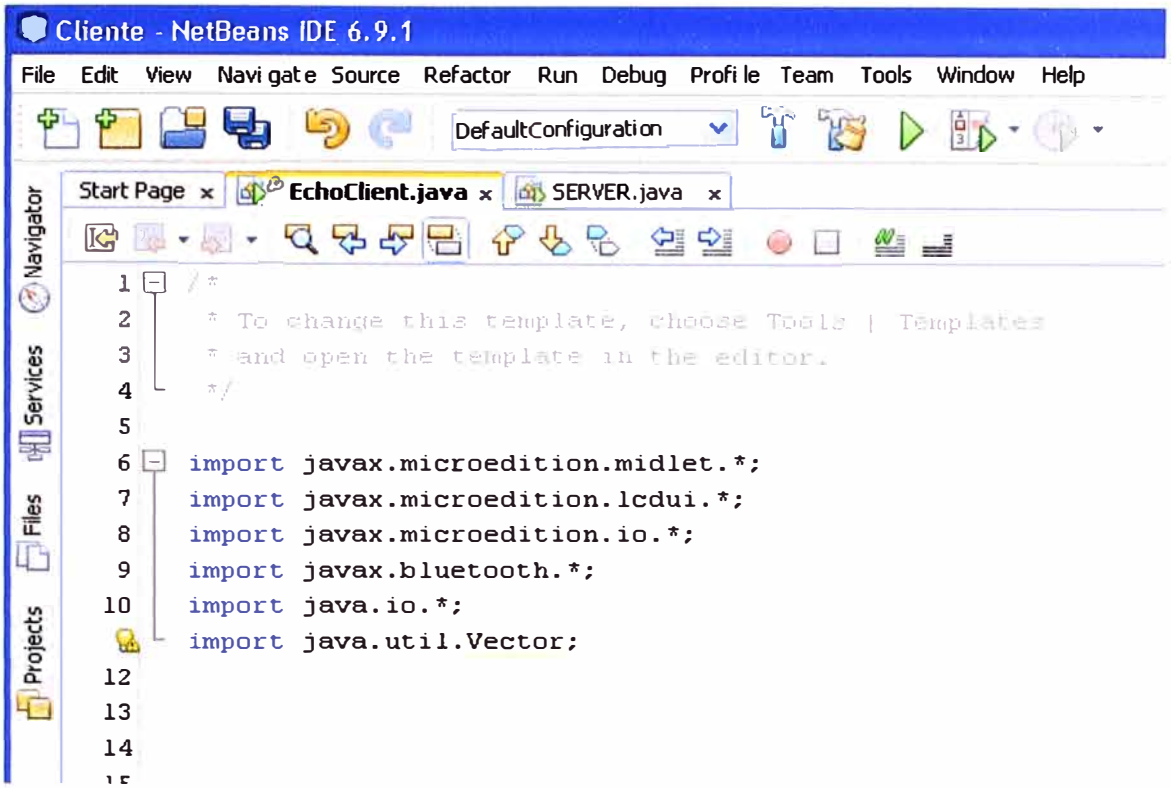


Fig. 3.2: Entorno NetBeans

Tabla 3.1: comandos en NetBeans

COMANDO	DESCRIPCIÓN
OK	Confirma una selección
CANCEL	Cancela la acción actual
BACK	Traslada al usuario a la pantalla anterior
STOP	Detiene una operación
HELP	Muestra una ayuda
SCREEN	Tipo genérico referente a la pantalla actual
ITEM	Tipo genérico referente a un elemento de la pantalla actual

A veces, y dependiendo del modelo y marca del dispositivo, sólo se pueden mostrar un número limitado de comandos en la pantalla. Al resto se accederá mediante un menú. En la Fig. 3.3 se puede observar una parte del código de este programa.

La interfaz DiscoveryListener es usada para que el dispositivo notifique eventos a la aplicación cada vez que se descubre un dispositivo, un servicio o se finaliza una búsqueda, en la Fig 3.3 se muestra como se implementan las dos interfaces juntas el CommandListener y DiscoveryListener, cuando el proceso de búsqueda ha completado o cancelado se llama al método inquiryCompleted().

```
14  public class EchoClient extends MIDlet implements CommandListener,DiscoveryListener{
15      List main_list,dev_list, menu;
16      Command exit,ok, volver, help , bombaON, bombaOFF, grupoOFF, grupoON, motorOFF,
17              motorON, commandOptionSelected;
18      TextBox cmd;
19      Display display;
        Vector devices,services;
21      LocalDevice local;
22      DiscoveryAgent agent;
23      DataOutputStream dout;
24      int currentDevice = 0;
25
26      String opcionSelect = "";
27      String bombaLabel = "BOMBA BCI 2";
28      String grupoLabel = "GRUPO ELECTROGENO 1";
29      String motorLabel = "MOTOR 1";
30      String helpLabel = "Ayuda";
31      String outLabel = "Salir";
32      String backLabel = "Volver";
33
```

Fig. 3.3 Código del programa en Netbeans

El inicio del código define la clase EchoClient, así como también la definición de los atributos de la clase EchoClient (TextBox, Display, Vector, Localdevice y Dataouptstream), cabe mencionar que la línea correspondiente a "int currentDevice" es un indicador para que el servidor lo pueda identificar y por ultimo la definición de constantes string.

Seguidamente se inicializa las instancias de los atributos como muestra parte del código. public void startApp() {

Se crea un comando para salir

volver = new Command(backLabel, Command.BACK, 0);

commandOptionSelected = null;

La lista de los comandos de prendido y apagado de los equipos eléctricos

bombaON = new Command("bombaON", Command.ITEM, 1);

bombaOFF = new Command("bombaOFF", Command.ITEM, 2);

grupoOFF = new Command("grupoOFF", Command.ITEM, 3);

grupoON = new Command("grupoON", Command.ITEM, 4);

motorOFF = new Command("motorOFF", Command.ITEM, 5);

motorON = new Command("motorON", Command.ITEM, 6);

Se muestra la ventana del menú principal

main_list = new List("EQUIPOS ELECTRICOS", List.IMPLICIT);

menu = new List("EQUIPOS ELECTRICOS", List.IMPLICIT);

menu.append(bombaLabel, null);

menu.append(grupoLabel, null);

menu.append(motorLabel, null);

```
menu.append(helpLabel, null);
menu.append(outLabel, null);
```

En el dispositivo móvil se mostrará un menú para poder elegir de la lista de equipos eléctricos con las opciones OK, EXIT, básicamente es elegir opciones.

```
dev_list  = new List("Select Device",Choice.IMPLICIT);
cmd       = new TextBox("Text to echo","",120,TextField.ANY);
exit      = new Command("Exit",Command.EXIT,1);
ok        = new Command("Send",Command.OK,1);
display   = Display.getDisplay(this);
main_list.addCommand(exit);
main_list.setCommandListener(this);
menu.addCommand(exit);
menu.setCommandListener(this);
dev_list.addCommand(exit);
dev_list.setCommandListener(this);
cmd.addCommand(ok);
cmd.setCommandListener(this);
```

Aquí se determina cual es la pantalla inicial del aplicativo, para luego mostrar la lista principal del menú principal.

```
main_list.append("Find Echo Server",null);
display.setCurrent(main_list);
}
```

Una vez elegido del menú principal uno de los equipos eléctricos a controlar, se ingresa a un submenú, el cual mostrará las opciones de prendido apagado y volver

```
public void commandAction(Command com, Displayable dis) {
    if (com == exit){
       destroyApp(false);
       notifyDestroyed();
    }
        if(volver == com){
       display.setCurrent(menu);
    }
    if(menu == dis){
       List nivel = null;
       switch(menu.getSelectedIndex()){
          case 0:
             nivel = new List(bombaLabel, List.EXCLUSIVE);
```

```
            nivel.setCommandListener(this);
            nivel.addCommand(bombaON);
            nivel.addCommand(bombaOFF);
            nivel.addCommand(volver);
            display.setCurrent(nivel);
            //showOptionsByEquip(bombaLabel, bombaON, bombaOFF );
            break;
```

En la siguiente parte del código se muestra como se realiza la conexión entre cliente – servidor y como se ejecuta la parte del código anterior. El cliente busca los dispositivos.

```
if (com == List.SELECT_COMMAND){
        if (dis == main_list){
           if (main_list.getSelectedIndex() >= 0){
              FindDevices();
              do_alert("Searching for devices...", Alert.FOREVER);
           }
        }
```

Establecimiento de la conexión cliente-servidor

```
 if (dis == dev_list){
            StreamConnection conn = null;
            ServiceRecord service =
            (ServiceRecord)services.elementAt(dev_list.getSelectedIndex());
            String url =
    service.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
            try {
           conn = (StreamConnection) Connector.open(url);
```

Muestra el menú en pantalla del celular

```
          dout = new DataOutputStream(conn.openOutputStream());
           display.setCurrent(menu);
            } catch (Exception e) {
               this.do_alert("Error Connecting" , 4000);
            }
```

El usuario envía una orden (comando) a la parte del servidor que básicamente es el prendido y apagado del equipo.

```
    if(com == ok || bombaON == com || bombaOFF == com || grupoOFF == com ||
grupoON == com || motorOFF == com || motorON == com){
```

```
//el cliente envía un comando
        try{
           do_alert("Opcion Seleccionada:"+com.getLabel(), 4100);
           commandOptionSelected = com;
           dout.writeChars(commandOptionSelected.getLabel() + "\n");
           dout.flush();
           commandOptionSelected = null;
        } catch (Exception e) {
           this.do_alert("Error sending data" , 4200);}
 public void showOptionsByEquip(String nombreEquip, Command cON, Command cOFF)
{
      List nivel = new List(nombreEquip, List.EXCLUSIVE);
      nivel.setCommandListener(this);
      nivel.addCommand(cON);
      nivel.addCommand(cOFF);
      nivel.addCommand(volver);
      display.setCurrent(nivel);
   }
```

3.1.2 Programación de la comunicación

Un objeto LocalDevice representa al dispositivo local. Este objeto será el punto de partida de prácticamente cualquier operación que vayamos a llevar a cabo en este API.

Alguna información de interés que podemos obtener a través de este objeto es, por ejemplo, la dirección Bluetooth de nuestro dispositivo, el apodo o "friendly-name" (también llamado "Bluetooth device name" o "user-friendly name").

Los posibles valores que admite setDiscoverable() están definidos en la clase DiscoveryAgent como campos estáticos. Estos son:

• DiscoveryAgent.GIAC (General/Unlimited Inquiry Access Code): Conectividad ilimitada.

• DiscoveryAgent.LIAC (Limited Dedicated Inquiry Access Code): Conectividad limitada.

• DiscoveryAgent.NOT_DISCOVERABLE: El dispositivo no será visible para el resto de dispositivos.

Las búsquedas de dispositivos y servicios Bluetooth las realizaremos a través del objeto DiscoveryAgent. Este objeto es único y lo obtendremos a través del método getDiscoveryAgent() del objeto LocalDevice.

a). Excepción BluetoothStateException

Al llamar al método getLocalDevice() se puede producir una excepción del tipo BluetoothStateException. Esto significa que no se pudo inicializar el sistema Bluetooth. La

excepción BluetoothStateException extiende de java.io.IOException y no añade ningún método adicional.

Para comenzar una nueva búsqueda de dispositivos llamaremos al método **startInquiry().** Este método requiere dos argumentos. El primer argumento es un entero que especifica el modo de conectividad que deben tener los dispositivos a buscar.

Este valor deberá ser DiscoveryAgent.GIAC o bien DiscoveryAgent.LIAC. El segundo argumento es un objeto que implemente DiscoveryListener.

La parte del código del programa es el siguiente:

```
public void FindDevices(){
    try{
       devices            = new Vector();
       LocalDevice local    = LocalDevice.getLocalDevice();
       DiscoveryAgent agent = local.getDiscoveryAgent();
       agent.startInquiry(DiscoveryAgent.GIAC,this);
    }catch(Exception e){
       this.do_alert("Erron in initiating search" , 4300);
    }
  }
```

b). Clase UUID

La clase UUID (universally unique identifier) representa indentificadores únicos universales.

Se trata de enteros de 128 bits que identifican protocolos y servicios. Como veremos más adelante un dispositivo puede ofrecer varios servicios. Los UUID servirán para identificarlos.

Se muestra en el codigo del programa como sigue:

```
  public void FindServices(RemoteDevice device){
    try{
       UUID [ ] uuids  = new UUID[1];
       uuids[0]      = new UUID("27012f0c68af4fbf8dbe6bbaf7aa432a",false);
       local         = LocalDevice.getLocalDevice();
       agent         = local.getDiscoveryAgent();
       agent.searchServices(null,uuids,device,this);
    }catch(Exception e){this.do_alert("Erron in initiating search" , 4400);}
  }
```

c). La interfaz DiscoveryListener

La interfaz DiscoveryListener tiene los siguientes métodos:

- public void deviceDiscovered(RemoteDevice rd, DeviceClass c)

- public void inquiryCompleted(int c)
- public void servicesDiscovered(int transID, ServiceRecord[] sr)
- public void serviceSearchCompleted(int transID, int respCode)

Los dos primeros métodos serán llamados durante el proceso de búsqueda de dispositivos.

Los otros dos métodos serán llamados durante un proceso de búsqueda de servicios.

public void deviceDiscovered(RemoteDevice rd, DeviceClass c)

Cada vez que se descubre un dispositivo se llama a este método. Nos pasa dos argumentos.

El primero es un objeto de la clase RemoteDevice que representa el dispositivo encontrado.

El segundo argumento nos permitirá determinar el tipo de dispositivo encontrado.

public void inquiryCompleted(int c)

Este método es llamado cuando la búsqueda de dispositivos ha finalizado. Nos pasa un argumento entero indicando el motivo de la finalización. Este argumento podrá tomar los valores: DiscoveryListener.INQUIRY_COMPLETED si la búsqueda ha concluido con normalidad,

DiscoveryListener.INQUIRY_ERROR si se ha producido un error en el proceso de búsqueda, o DiscoveryListener.INQUIRY_TERMINATED si la búsqueda fue cancelada.

So obtendrá la URL necesaria para realizar la conexión a través del método getConnectionURL() de un objeto ServiceRecord. Un objeto ServiceRecord representa un servicio, es decir una vez encontrado el servicio deseado (un objeto ServiceRecord), él mismo proveerá la URL necesaria para conectarnos a él.

Este método requiere dos argumentos, el primero de los cuales indica si se debe autenticar y/o cifrar la conexión. Los posibles valores de este primer argumento son:

• ServiceRecord.NOAUTHENTICATE_NOENCRYPT: No se requiere ni autenticación, ni cifrado.

• ServiceRecord.AUTHENTICATE_NOENCRYPT: Se requiere autenticación, pero no cifrado

• ServiceRecord.AUTHENTICATE_ENCRYPT: Se requiere tanto autenticación como cifrado.

El código en el programa NetBeans es el siguiente:

```
public void deviceDiscovered(RemoteDevice remoteDevice,DeviceClass deviceClass) {
    devices.addElement(remoteDevice);
}
public void servicesDiscovered(int transID,ServiceRecord[] serviceRecord) {
    for (int x = 0; x < serviceRecord.length; x++ )
```

```
            services.addElement(serviceRecord[x]);
        try{
            dev_list.append(((RemoteDevice)devices.elementAt(currentDevice)).
                                    getFriendlyName(false),null);
        }catch(Exception e){this.do_alert("Erron in initiating search" , 4500);}
    }
public void inquiryCompleted(int param){
        switch (param) {
            case DiscoveryListener.INQUIRY_COMPLETED:
                if (devices.size() > 0){
                    services = new Vector();
                    this.FindServices((RemoteDevice)
                            devices.elementAt(0));
                }else
                  do_alert("No device found in range",4600);
            break;
            case DiscoveryListener.INQUIRY_ERROR:
                this.do_alert("Inqury error" , 4700);
            break;
            case DiscoveryListener.INQUIRY_TERMINATED:
  agent.cancelInquiry()
                 this.do_alert("Inqury Canceled" , 4800);
            break;
           }
    }
```

Para realizar una búsqueda de servicios también se usará la clase DiscoveryAgent y se implementará la interfaz DiscoveryListener. En este caso es de interés los métodos servicesDiscovered() y serviceSearchCompleted() de la interfaz DiscoveryListener.

Para comenzar la búsqueda se usará searchServices() de la clase DiscoveryAgent.

Es posible que queramos hacer diversas búsquedas de servicios al mismo tiempo. El método searchServices() nos devolverá un entero que identificará la búsqueda. Este valor entero nos servirá para saber a qué búsqueda pertenecen los eventos servicesDiscovered() y serviceSearchCompleted().

Public void serviceSearchCompleted(int transID, int respCode), este método es

llamado cuando se finaliza un proceso de búsqueda. El primer argumento identifica el proceso de búsqueda (que recordemos que es el valor devuelto al invocar el método searchServices() de la clase DiscoveryAgent). El segundo argumento indica el motivo de finalización de la búsqueda. Este último argumento puede tomar los siguientes valores:

- DiscoveryListener.SERVICE_SEARCH_COMPLETED: El proceso de búsqueda ha finalizado con normalidad
- DiscoveryListener.SERVICE_SEARCH_TERMINATED: El proceso de búsqueda ha sido cancelado
- DiscoveryListener.SERVICE_SEARCH_NO_RECORDS: No se han encontrado registros en el proceso de búsqueda
- DiscoveryListener.SERVICE_SEARCH_ERROR: Hubo un error durante el proceso de búsqueda
- DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE: No se pudo conectar al dispositivo sobre el que se quería realizar la búsqueda.

Podemos cancelar un proceso de búsqueda de servicios llamando al método cancel ServiceSearch() pasándole como argumento el identificador de proceso de búsqueda, que es el número entero devuelto cuando se comenzó la búsqueda con searchServices().

El código generado es el siguiente:

```
public void serviceSearchCompleted(int transID, int respCode) {
      switch(respCode) {
         case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
            if(currentDevice == devices.size() -1){ //all devices have been searched
               if(services.size() > 0){
                  display.setCurrent(dev_list);
               }else
                  do_alert("The service was not found",4900);
            }else{
               currentDevice++;
               this.FindServices((RemoteDevice)devices.elementAt(currentDevice));
            }
         break;
         case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
            this.do_alert("Device not Reachable" , 5000);
         break;
         case DiscoveryListener.SERVICE_SEARCH_ERROR:
            this.do_alert("Service serch error" , 5100);
         break;
```

```
        case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
          this.do_alert("No records returned" , 5200);
        break;
        case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
          this.do_alert("Inqury Cancled" , 5300);
        break;
      }
  }
```

3.2 Aplicación bluetooth servidor

3.2.1 Programación del código para el servidor

Lo primero de todo, para ofrecer un servicio a través de Bluetooth necesitaremos poner nuestro dispositivo en modo visible. Esto se hace, a través de la clase LocalDevice: Para crear una conexión servidora es necesario pasarle una URL al método Connector.open() para indicar que es un servidor indicaremos "localhost" como host en la URL. URL. De este modo la URL deberá comenzar por "btspp://localhost:" o por "btl2cap://localhost:".

Además del host de la URL deberemos indicar el UUID que identifica el servicio. Posteriormente indicaremos el nombre del servicio y otros parámetros, como por ejemplo el requerimiento o no de autenticación.

Al pasar una URL de este tipo al método Connector.open(), éste devolverá un "notifier", que en el caso de SPP será un objeto StreamConnectionNotifier y en el caso de L2CAP será un objeto L2CAPConnectionNotifier. Estos objetos permitirán escuchar conexiones entrantes de los clientes.

El codigo es el siguiente:

```
public class SERVER {
public final UUID uuid = new UUID( "27012f0c68af4fbf8dbe6bbaf7aa432a", false);
   public final String name = "Echo Server";
   public final String url  =  "btspp://localhost:" + uuid
                        + ";name=" + name
                        + ";authenticate=false;encrypt=false;";
   LocalDevice local = null;
   StreamConnectionNotifier server = null;
   StreamConnection conn = null;
```

En las siguientes líneas del código se muestra como el servidor detecta el dispositivo en este caso el celular, apertura el servicio, e indica que el dispositivo esta conectado.

```
public SERVER() {
      try {
```

```
            System.out.println("DISPOSITIVO HACER DETECTADO...");
            local = LocalDevice.getLocalDevice();
            local.setDiscoverable(DiscoveryAgent.GIAC);
            System.out.println("EMPEZANDO EL SERVICIO...");
            server = (StreamConnectionNotifier)Connector.open(url);
            System.out.println("ESPERANDO POR CONEXION CLIENTE...");
            conn = server.acceptAndOpen();
            System.out.println("CLIENTE CONECTADO");
            DataInputStream din   = new DataInputStream(conn.openInputStream());
            while(true){
               String cmd = "";
               char c;
               while (((c = din.readChar()) > 0) && (c!='\n') ){
                  cmd = cmd + c;
               }
               System.out.println("RECIBIDO" + cmd);
               createWindow(cmd);
            }
         } catch (Exception  e) {System.out.println("Exception Occured: " + e.toString());}
      }
      public static void main (String args[]){
         SERVER echoserver = new SERVER();
      }
```

Una vez entablado la comunicación con el cliente, el servidor esta listo a recibir los comandos que son básicamente el prendido y apagado, esto se hace a traves de ventanas emergentes, cada vez que el cliente envie una orden el servidor ejecutara una ventana emergente tipo pop up, indicando que la orden llego.

```
      public static void createWindow(String codigoEquipo){
         JFrame ventana = new JFrame("Acciòn Realizada");
         ventana.setDefaultCloseOperation(1);
        String mensaje = obtenerMensajeByCodigo(codigoEquipo);
         JComponent comp = new JTextArea(mensaje);
         ventana.getContentPane().add(comp, BorderLayout.CENTER);
         ventana.setSize(300, 200);
         ventana.setVisible(true);
      }
      public static String obtenerMensajeByCodigo(String codigo){
```

```
    String mensaje = "";
    if(codigo.equals("bombaON")){
       mensaje = "BOMBA BCI 2, Encendido";
    }else if(codigo.equals("bombaOFF")){
      mensaje = "BOMBA BCI 2, Apagado";
    }else if(codigo.equals("grupoON")){
      mensaje = "GRUPO ELECTROGENO 1, Encendido";
    }else if(codigo.equals("grupoOFF")){
      mensaje = "GRUPO ELECTROGENO 1, Apagado";
    }else if(codigo.equals("motorON")){
      mensaje = "MOTOR 1, Encendido";
    }else if(codigo.equals("motorOFF")){
      mensaje = "MOTOR 1, Apagado";
    }else{
      mensaje = "No se realizò ninguna acción";
    }
    return mensaje;
  }
```

En el anexo A y B se muestran los códigos completos del cliente y del servidor.

CAPITULO IV

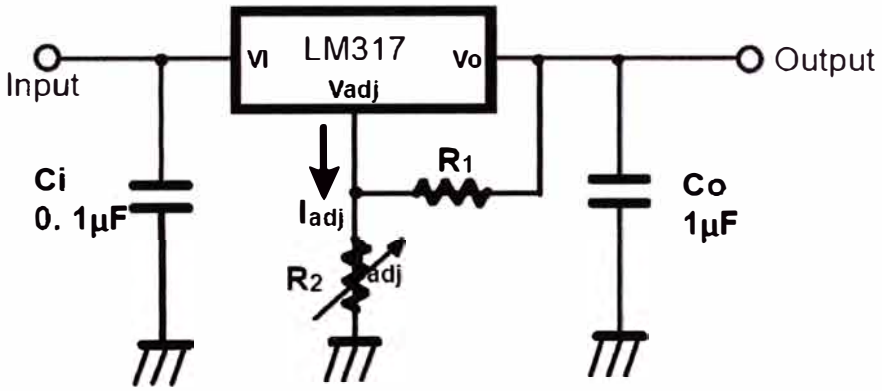
DISEÑO Y CÁLCULO DE LA INTERFAZ DE HARDWARE DEL SISTEMA BLUETOOTH

Hasta este punto se ha desarrollado la aplicación Java en el teléfono celular. El dispositivo móvil está listo para conectarse vía Bluetooth con el módulo de control.

El circuito electrónico se divide en dos partes: Circuitos de Alimentación y Circuitos Acopladores de Potencia.

4.1 Circuito de Alimentación

El chip de un modulo Bluetooth. Según especificaciones del fabricante, necesitan un voltaje de polarización de 3.3V y usan niveles lógicos TTL, en la figura 4.1 se muestra una configuración típica del regulador de voltaje LM317



$$V_O = 1.25V\ (1 + R_2/R_1) + I_{adj}\,R_2$$

Fig. 4.1: Configuración típica LM317 (ver anexo C)

Ci . 0.1uF se podría obviar, este capacitor es requerido cuando el regulador está a una distancia considerable del filtro de la fuente de poder.

Co . 1uF no es necesario para la estabilidad del regulador pero mejora la respuesta transitoria.

Mientras la Iadj sea menor de 100uA, el error asociado al voltaje de salida Iadj R_2 es casi insignificante en la mayoría de aplicaciones.

El módulo KC-21 (ver anexo 2) necesita un voltaje de entrada de 3.3 V a 3.6V entonces para un voltaje 3.4V y considerando Iadj=0; se obtiene:

- Del datasheet del fabricante ver anexo C.

$$Vo = 1.25\left(1 + \frac{R_2}{R_1}\right) + I_{adj}R_2 \quad \text{.....(4.1)}$$

$$3.4 = 1.25\left(1 + \frac{R_2}{R_1}\right) + I_{adj}R_2 \approx 1.25\left(1 + \frac{R_2}{R_1}\right) \quad \text{.....(4.2)}$$

$$\frac{3.4}{1.25} - 1 = \frac{R_2}{R_1}$$

$$\frac{R_2}{R_1} = 1.72$$

$$R_2 = 1.72R_1 \quad \text{.....(4.3)}$$

Se usará una resistencia variable

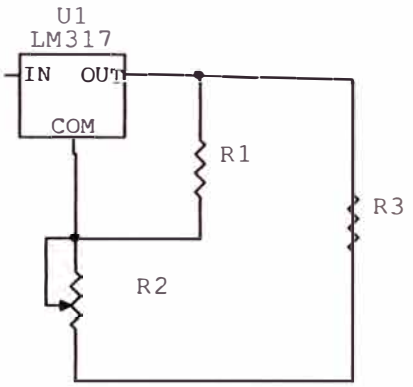


Figura 4.2 Elementos resistivos en la salida del regulado

De la figura 4.2, se requiere hallar las resistencias R1 Y R2. Del voltaje de referencia proporcionado por el fabricante se observa de Vin y Vout del chip LM317.

$$3 \le Vin - Vout \le 40 \quad \text{.....(4.4)}$$

$$3 \le 13 \le Vin - 3.4 \le 40$$

$$Vin = 16.4 \approx 16$$

Vin es el valor mínimo que debería haber en la entrada del LM317

Vinmin= 16 Voltios en la entrada del LM317

En la figura 4.3 se muestra el esquema de la fuente regulada. Para la simulación se esta usando el software de simulación Circuitmaker.

$I_1 = I_L + Iadj$

$Iadj = 100\mu A$ por las especificaciones del LM317

$I_L = 1Amp$

$I_1 \approx 1Amp$

$Ic \approx I_1 = 1Amp$

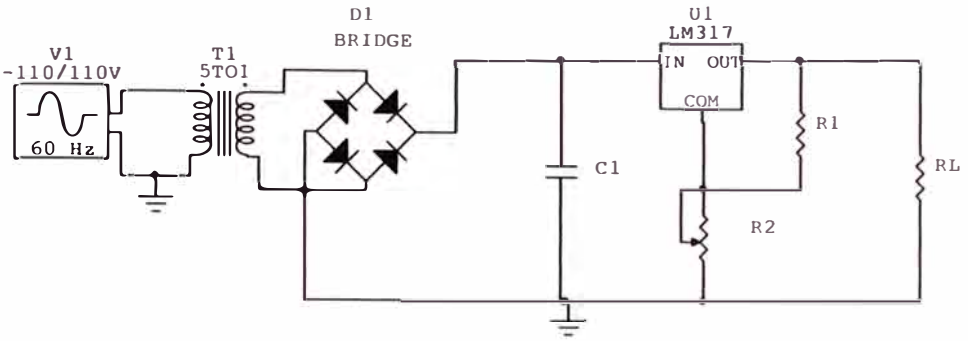


Figura 4.3: fuente regulada

$Ic \approx I_1 = 1Amp$

si R2 =0 entonces Vout=1.25voltios

$Vo\min = 1.25 = I_1 R_1$

$$R_1 = \frac{1.25}{100\mu A} = 12.5K\Omega$$

De la ecuación (4.3) se tiene:

$R_2=1.72R_1 =21.5K\Omega$

Asumiendo condición de diseño:

$Vin_{\min} = 0.9Vin_{\max}$

$Vin_{\max} = 17.7777 \approx 18 \ Voltios$

El voltaje rizo viene dado por:

$Vrizo = Vin_{\max} - Vin_{\min} = 18 - 16 = 2 \ Voltios$

$$Vr = \frac{I_{DC}}{2fC}$$

$$C = \frac{Idc}{2fcVr} = \frac{1}{2x60x2} = 4166.7\mu F$$

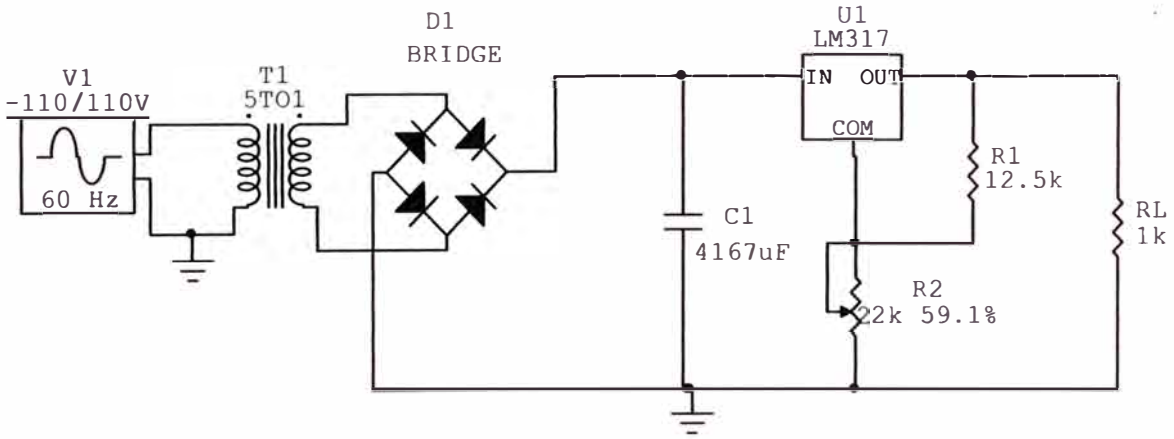


Figura 4.5: fuente regulada.

La figura 4.5 muestra la fuente regulada con sus elementos hallados. En la Fig.4.6

se observa la forma de onda a la salida del regulador (Vout). El cual tiene un valor de 3.27 Voltios, lo que viene hacer 3.3 voltios.el voltaje de funcionamiento del modulo bluetooth.

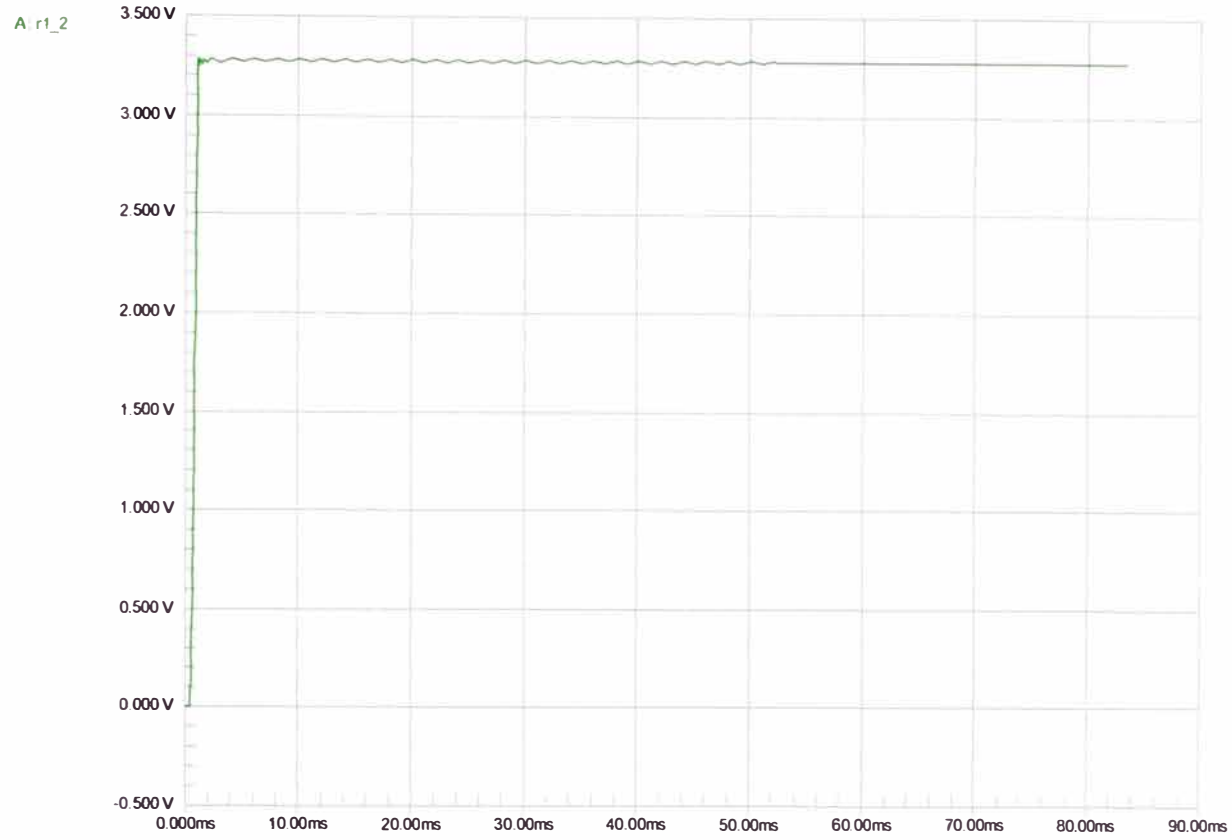


Figura 4.6: Simulación en la salida de la carga R_L fuente regulada.
En el software Circuitmaker

4.2 Circuitos acopladores de potencia

Cada GPIO del módulo Bluetooth se conectará a un circuito acoplador de potencia diferente. Cada Circuito Acoplador está compuesto básicamente por un optotriac y un triac. El optotriac usado en este proyecto es el MOC3021.

El triac puede ser un triac genérico, de 8A a 600V, puede ser el BTB0841.

4.2.1 El optotriac

Es un triac controlado a través de la luz que le emite un diodo LED integrado. Al tener este LED encendido, el triac conduce y cuando este LED está apagado no conduce. De esta manera se independiza la parte de control y la parte de potencia, es decir, el circuito de control solo estará conectado a un LED, que se activará con voltajes pequeños DC, y aislado del alto voltaje AC que usa la carga. El funcionamiento de la carga estará dado prácticamente por el estado del Optotriac.

4.2.2 El triac

Es un dispositivo semiconductor de tres pines que permite controlar el paso de

corriente en dos sentidos, entre dos terminales (ánodos), usando el tercer pin(compuerta).

Para los valores de las resistencias R_1, R_2 y R_3 , que se muestran en la Fig 4.8. El valor es de 360Ω según Fig. 4.7 que es una recomendación de diseño de la casa fabricante de este dispositivo. Las resistencias R4, R5 y R6 se calcularon de la siguiente manera

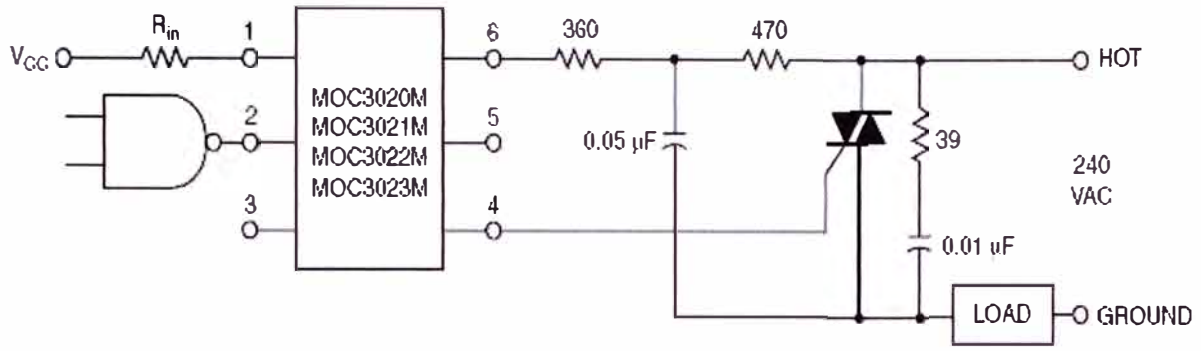


Figura 4.7: Configuración típica del MOC3020M (ver anexo c):

Según las especificaciones del fabricante del modulo KC-21 (ver anexo c), las salidas IO manejan un voltaje de V_{OL}=0,4V para nivel bajo y V_{OH}=2.4V para nivel alto, en ambas condiciones I_{OL}= I_{OH}=2mA y el máximo valor que puede tomar las corrientes I_{OL}= I_{OH} = I=3.1mA.

El voltaje de entrada V_F de este optotriac puede estar alrededor de 1.2V típicamente, según el gráfico que se encuentra en el datasheet (anexo c). Para un valor de I=2mA se tendrá un V_F=1,07 V y para un valor de I=3.1mA se tendrá un V_F=1,09 V. Por lo tanto, para ambas condiciones se tiene:

$I = 2mA$

$Vf = 1.07$

$$R = \frac{2.4 - 1.07}{2} = 0.665K\Omega$$

$I = 3.1mA$

$Vf = 1.09$

$$R = \frac{2.4 - 1.09}{3.1} = 0.422K\Omega$$

Entonces los valores de resistencia a la entrada del MOC3020 deben estar entre 422 y 665. Se escogió R4=R5=R6=470Ω. El Diagrama del circuito se muestra en la Fig 4.8.

Se puede observar como el módulo KC-21 queda aislado de la línea de 220V, protegiendo de esta manera ante cualquier falla que pueda suceder en la línea de alterna., siendo estos elementos de protección el triac y el optotriac.

A continuación se muestra el esquema Fig 4.8 de los circuitos acopladores de potencia.

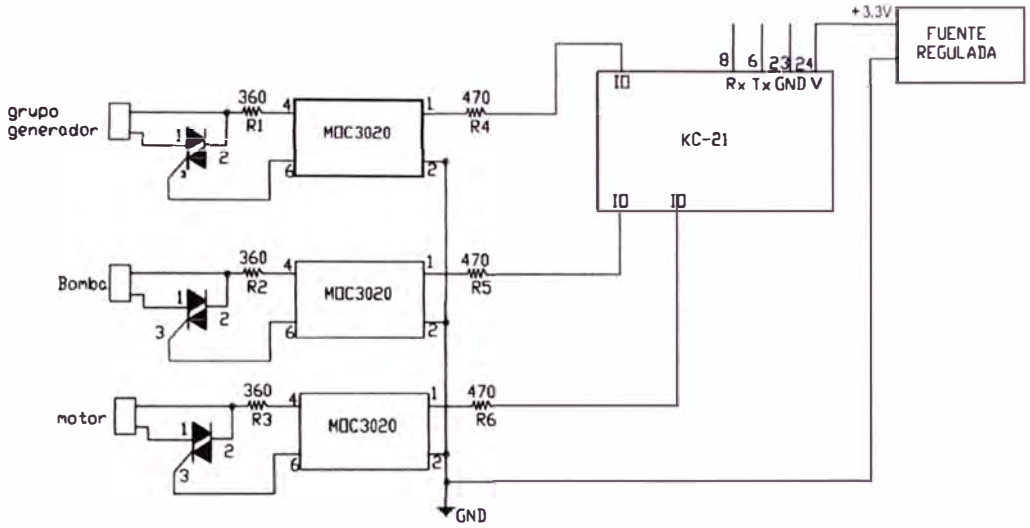


Figura 4.8: Esquema de los circuitos acopladores de potencia

CAPITULO V
SIMULACIÓN

5.1 Imágenes de la simulación celular (cliente) - computadora (servidor)

A continuación se muestra, la simulación del programa. Cliente y servidor:

Se inicializa el servidor y muestra la siguiente pantalla esperando la conexión con el cliente (celular), observar la figura 6.1

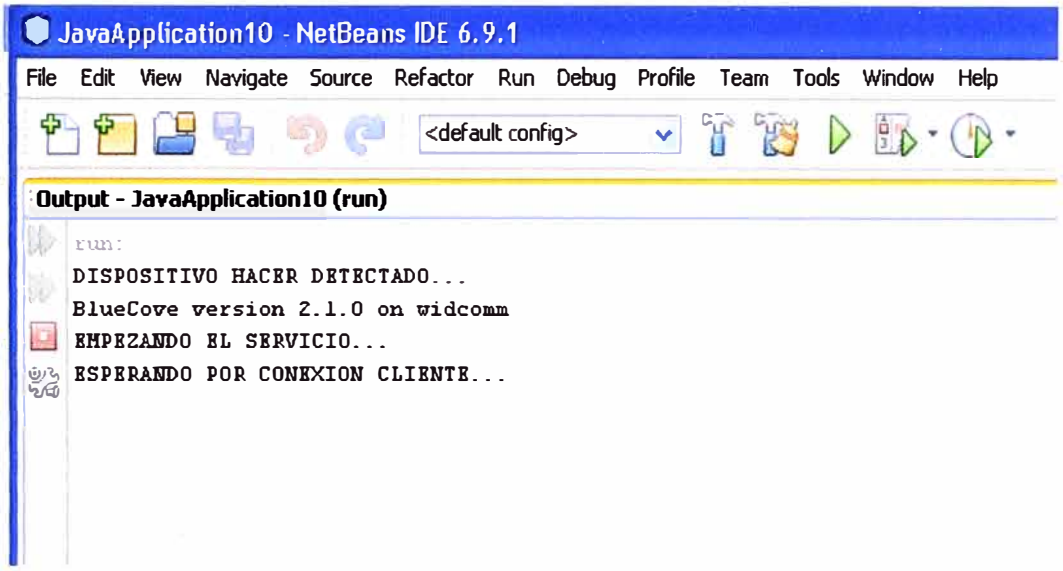


Figura 6.1: Servidor esperando la conexión del cliente (celular)

El Bluecove [10] es una librería que se añade al programa y es de distribución libre. BlueCove es una implementación de código abierto de la API Bluetooth JSR-82 para aplicaciones java J2SE. Aunque BlueCove no implementa íntegramente la API JSR-82, permite de manera simple migrar un código escrito para J2ME a una aplicación J2SE.

Bluecove provee para la API JSR-82 los siguientes perfiles Bluetooth: SDAP que se usa para el descubrimiento de dispositivos, RFCOMM es un protocolo de emulación del puerto serie, L2CAP es el control de enlace lógico y el protocolo de adaptación y OBEX intercambio de objetos genéricos.

Se inicializa el Cliente (celular), para el desarrollo de la aplicación se ha utilizado un teléfono celular de la marca Samsung GT-S5233T en el anexo c, se da detalles de este equipo celular. En la Fig. 6.2 se observa el inicio del aplicativo en el celular.

---

[10] referencias: http://bluecove.org/

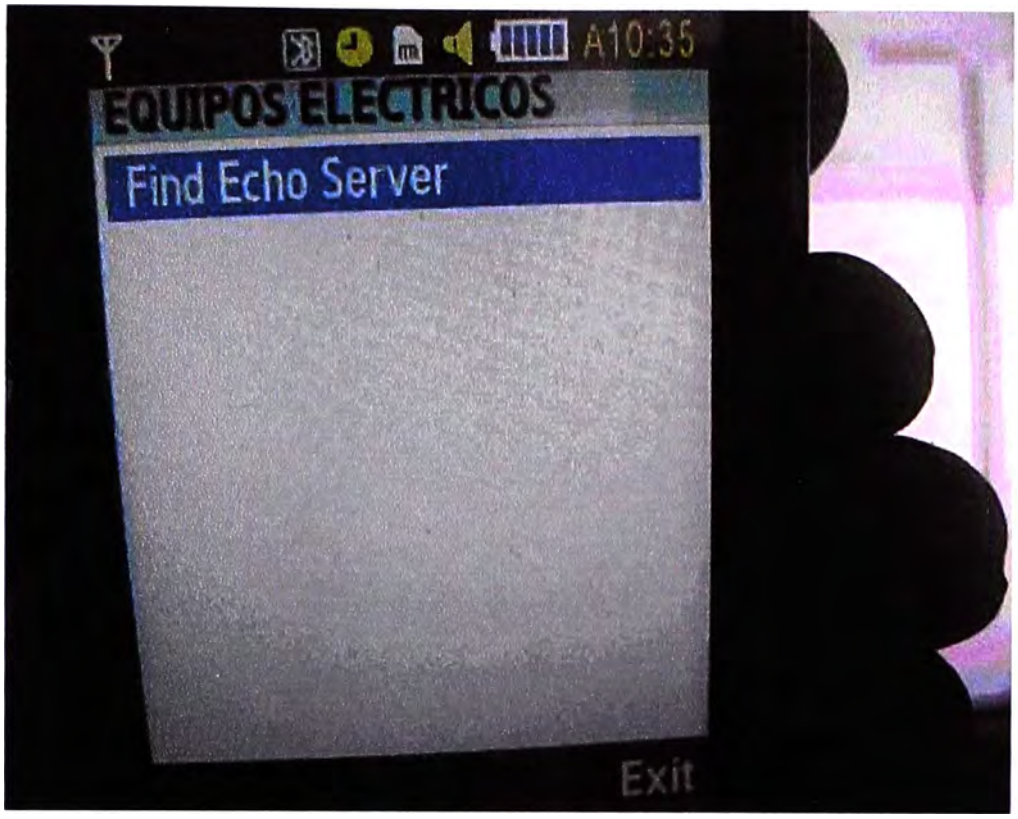


Figura 6.2: Inicio del aplicativo en el teléfono celular

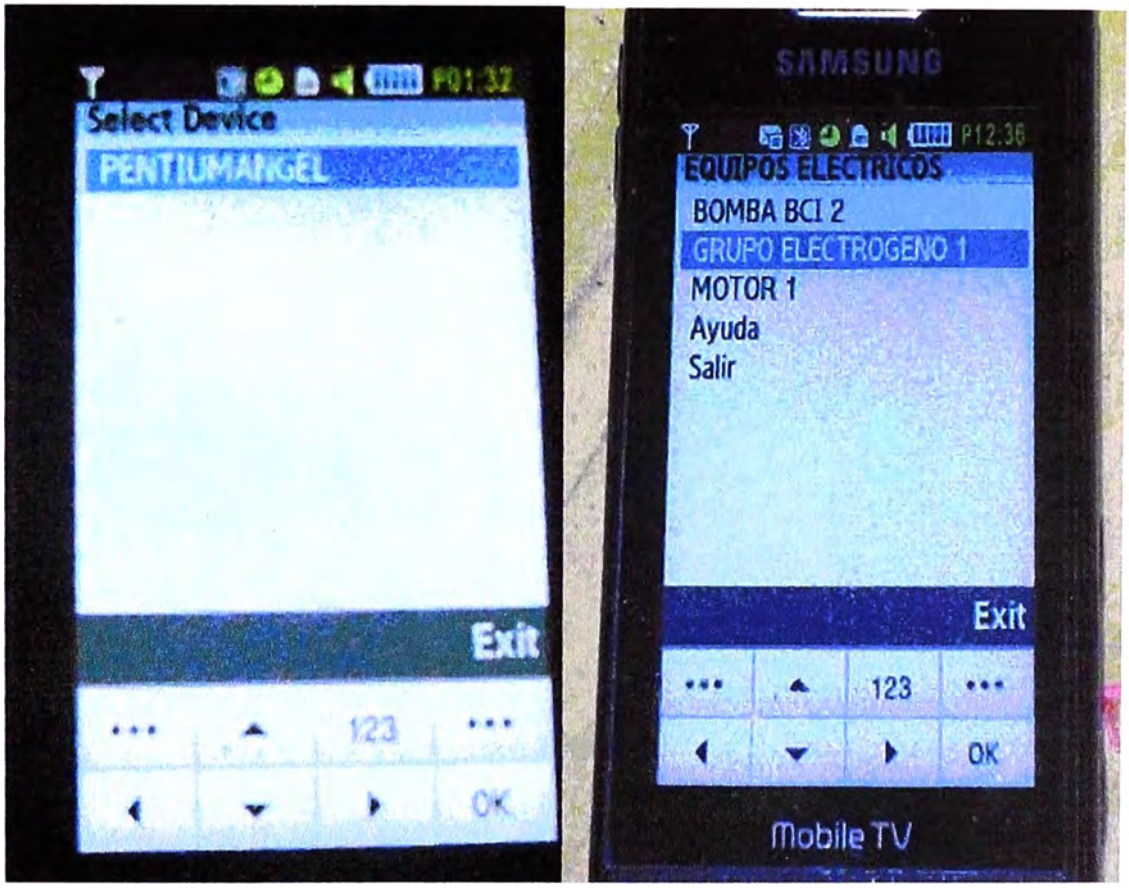


Figura 6.3: Detección de el servidor (pentiumAngel) por parte del cliente y el posterior ingreso al menú principal

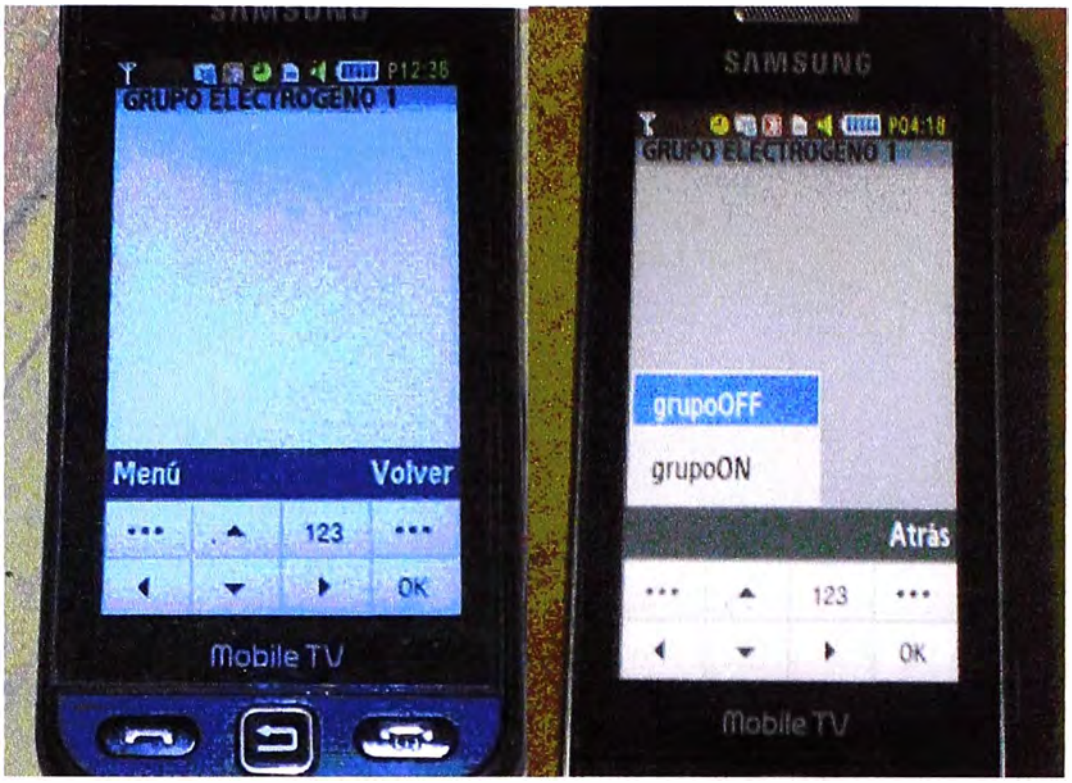


Figura 6.4: Ingreso a la opción Grupo Electrógeno y despliegue del submenú de comandos de encendido y apagado

Una vez entablada la conexión se puede enviar comandos de encender y apagar de los equipos eléctricos. En el servidor aparecen las órdenes del cliente, en la figura 6.5 se puede observar como el servidor detecta al cliente y que esta listo a recibir los comandos prender y apagar de los equipos

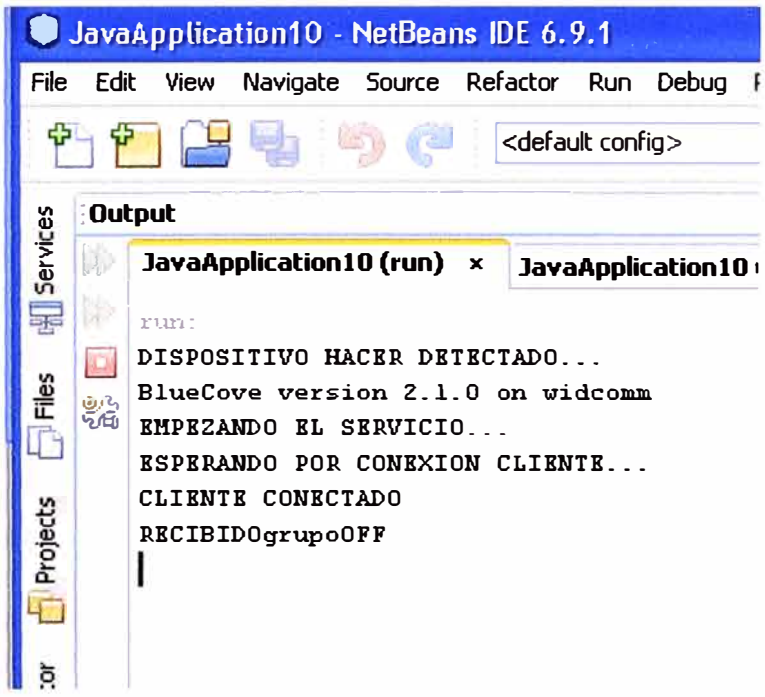


Figura 6.5: detección del cliente e inicio de recepción de los comandos

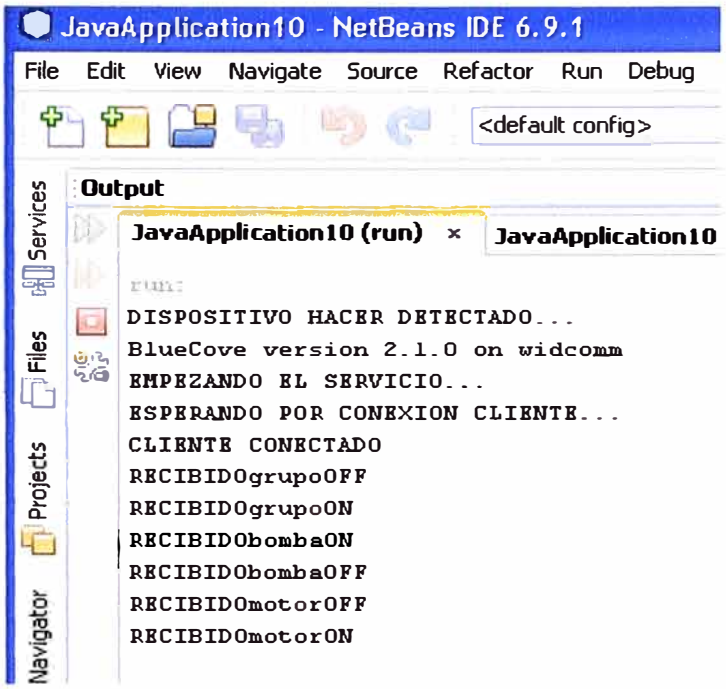


Figura 6.6: Recepción por parte del servidor de los comandos (encender y apagar) de los 3 equipos

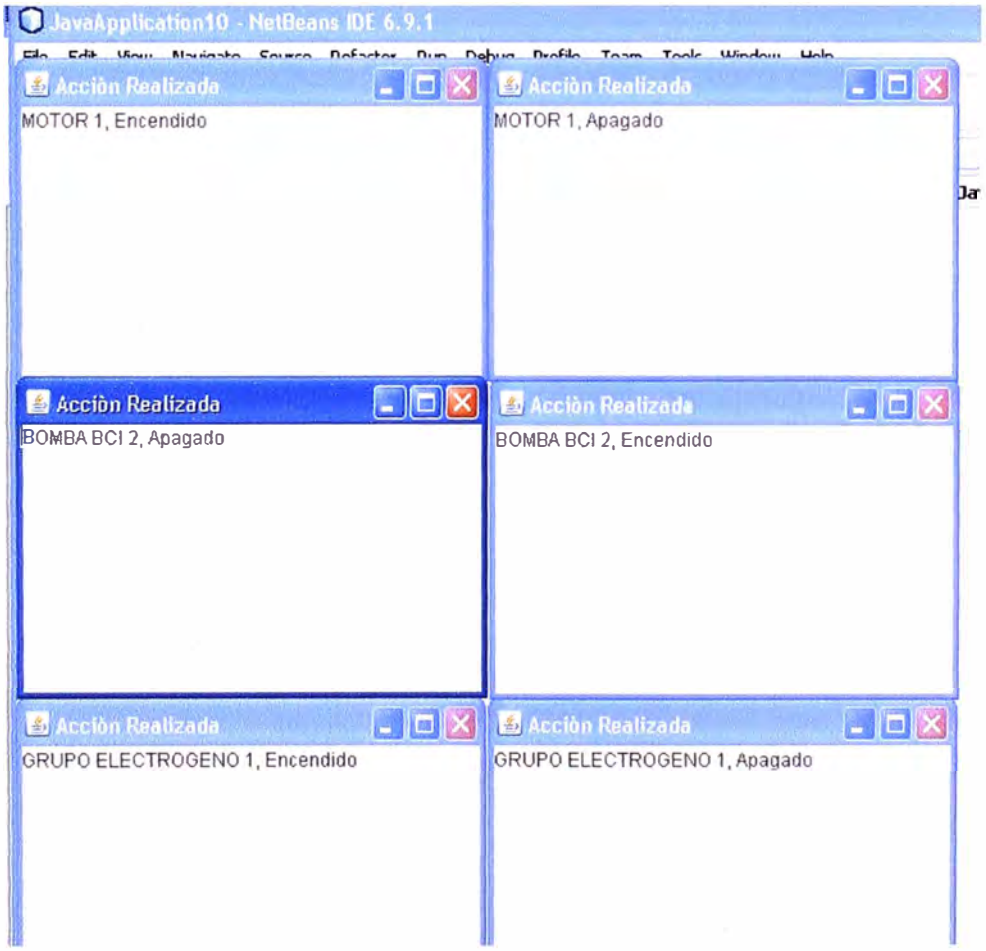


Figura 6.7: El servidor una vez que recepciona un comando ya sea encender o apagar, muestra en pantalla una ventana tipo pop up mostrando que equipo eléctrico se ha encendido o apagado.

De esta manera concluye la simulación del presente informe de suficiencia que es el control de equipos eléctricos utilizando la tecnología Bluetooth.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Este proyecto ha permitido dar un uso adicional a la tecnología Bluetooth integrada en los teléfonos celulares. También se ha demostrado que la Plataforma Java ME creada para dispositivos móviles se puede usar para controlar equipos eléctricos usando la tecnología Bluetooth.

2. Se ha conseguido desarrollar un sistema Bluetooth sencillo y a la vez muy útil, mucho más para personas con algún tipo de discapacidad o que por algún motivo no pueden usar los controles convencionales dentro de una vivienda.

3. Se puede concluir que Bluetooth es un ejemplo del avance acelerado de la tecnología para integrar las telecomunicaciones globalmente y proveer acceso a más personas día a día.

4. Los equipos celulares al incluir la tecnología Bluetooth, les proporciona cierta ventaja con respecto a otras tecnologías como por ejemplo el wifi; el cual necesita ser configurado tanto en hardware como en software para su uso y su implementación es mas costosa, la tecnología Bluetooth se puede usar sin haber instalado una infraestructura de red, se puede concluir que el usar un teléfono celular como dispositivo de control es muy ventajoso, puesto que ya tiene incluida esta tecnología.

5. Al usar el teléfono celular y adaptado el modulo Bluetooth a un equipo eléctrico, simultáneamente se pueden manipular los controles tanto manualmente como vía Bluetooth. Por lo tanto, se puede adaptar cualquier equipo eléctrico para que sea controlado desde un teléfono celular.

6. La piconet que se forma al realizar la conexión, puede contar con más de un cliente que se conecta al servidor. Por lo que, se concluye que un equipo eléctrico conectado a este sistema puede ser controlado desde varios puntos a la vez y al mismo tiempo sin necesidad de implementar algo adicional.

7. El módulo de control solo puede controlar dispositivos en dos estados, generalmente encendido y apagado.

8. La simulación del sistema de control es satisfactorio y se cumplió el objetivo del presente Informe de Suficiencia. Por lo tanto, se puede concluir, que Bluetooth es una

tecnología alternativa para este tipo de aplicaciones por su capacidad de funcionamiento en equipos móviles, de reducido tamaño y bajo consumo de potencia, su estabilidad y seguridad.

RECOMENDACIONES

1. Se recomienda el uso del lenguaje Java, para la creación de aplicaciones para móviles, por la versatilidad y gran cantidad de equipos que la soportan.

2. El presente informe de suficiencia puede ser referencia y guía para quienes deseen continuar desarrollando este tipo de aplicaciones. Por ejemplo, se recomienda mejorar este sistema para que realice control y monitoreo simultáneamente, que este control se pueda realizar desde cualquier lugar e inclusive corregir un determinado proceso de control, para esto integrándose a la Internet, con el API JSR-62 se puede lograr lo mencionado anteriormente.

3. Se recomienda tomar como referencia este informe de suficiencia para realizar aplicaciones dirigidas por ejemplo al área salud, monitorear el estado de un paciente desde el lugar de trabajo o casa, para esto se recomienda usar el JSR-135 en la aplicación, este API especifica el control multimedia.

4. Se recomienda adquirir módulos Bluetooth mas completos que el modulo KC-21, que incluyan entrada de 5V y posean entradas para programar los comandos AT, algunos incluyen entrada serial para la comunicación con la PC.

5. El módulo KC-21; es de clase 2, diseñado para ser usado a distancias máximas de 10m., para aplicaciones de control de distancias mayores, se recomienda adquirir un módulo Bluetooth de clase 1, que permite distancias máximas de 100m.

6. Para el diseño de sistemas Bluetooth se recomienda usar módulos que cuenten con una interfaz serial, de esta manera se podrán ver los datos recibidos usando el HyperTerminal de la plataforma Windows.

7. Si se desea usar este Módulo de Control Bluetooth para realizar un control en más de dos estados se recomienda usar más terminales GPIO del módulo, estas entradas figuran usualmente en los datasheets de los fabricantes.

8. Se recomienda aprovechar los 14 GPIOs del módulo KC-21 para aumentar el número de dispositivos controlados dentro de una de vivienda, una fábrica, etc. Evitando el uso de un módulo de control adicional y de esta manera disminuir los costos.

ANEXO A
CÓDIGO FUENTE CLIENTE (Celular)

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.Vector;
public class EchoClient extends MIDlet implements CommandListener,DiscoveryListener{
   List main_list,dev_list, menu;
   Command exit,ok, volver, help , bombaON, bombaOFF, grupoOFF, grupoON,
motorOFF, motorON, commandOptionSelected;
   TextBox cmd;
   Display display;
   Vector devices,services;
   LocalDevice local;
   DiscoveryAgent agent;
   DataOutputStream dout;
   int currentDevice = 0;
   String opcionSelect = "";
   String bombaLabel = "BOMBA BCI 2";
   String grupoLabel = "GRUPO ELECTROGENO 1";
   String motorLabel = "MOTOR 1";
   String helpLabel = "Ayuda";
   String outLabel = "Salir";
   String backLabel = "Volver";
   public void startApp() {
      volver = new Command(backLabel, Command.BACK, 0);
      commandOptionSelected = null;
      bombaON = new Command("bombaON", Command.ITEM, 1);
      bombaOFF = new Command("bombaOFF", Command.ITEM, 2);
      grupoOFF = new Command("grupoOFF", Command.ITEM, 3);
      grupoON = new Command("grupoON", Command.ITEM, 4);
      motorOFF = new Command("motorOFF", Command.ITEM, 5);
      motorON = new Command("motorON", Command.ITEM, 6);
      main  list = new List("EQUIPOS ELECTRICOS", List.IMPLICIT)
      menu = new List("EQUIPOS ELECTRICOS", List.IMPLICIT);
      menu.append(bombaLabel, null);
```

```
    menu.append(grupoLabel, null);
    menu.append(motorLabel, null);
    menu.append(helpLabel, null);
    menu.append(outLabel, null);
    dev_list  = new List("Select Device",Choice.IMPLICIT);
    cmd  = new TextBox("Text to echo","",120,TextField.ANY);
    exit  = new Command("Exit",Command.EXIT,1);
    ok  = new Command("Send",Command.OK,1);
    display = Display.getDisplay(this);
    main_list.addCommand(exit);
    main_list.setCommandListener(this);
    menu.addCommand(exit);
    menu.setCommandListener(this);
    dev_list.addCommand(exit);
    dev_list.setCommandListener(this);
    cmd.addCommand(ok);
    cmd.setCommandListener(this);
    main_list.append("Find Echo Server",null);
    display.setCurrent(main_list);
}
public void commandAction(Command com, Displayable dis) {
    if (com == exit){
        destroyApp(false);
        notifyDestroyed();
    }
    if(volver == com){
        display.setCurrent(menu);
    }
    if(menu == dis){
        //do_alert("llega al menu:"+com.getLabel(), 300);
        List nivel = null;
        switch(menu.getSelectedIndex()){
            case 0:
                nivel = new List(bombaLabel, List.EXCLUSIVE);
                nivel.setCommandListener(this);
                nivel.addCommand(bombaON);
                nivel.addCommand(bombaOFF);
```

```
            nivel.addCommand(volver);
            display.setCurrent(nivel);
            //muestra opciones de submenú (grupoLabel, grupoON, grupoOFF)
            break;
        case 1:
            nivel = new List(grupoLabel, List.EXCLUSIVE);
            nivel.setCommandListener(this);
            nivel.addCommand(grupoON);
            nivel.addCommand(grupoOFF);
            nivel.addCommand(volver);
            display.setCurrent(nivel);
            //muestra opciones de submenú (grupoLabel, grupoON, grupoOFF)
            break;
        case 2:
            nivel = new List(motorLabel, List.EXCLUSIVE);
            nivel.setCommandListener(this);
            nivel.addCommand(motorON);
            nivel.addCommand(motorOFF);
            nivel.addCommand(volver);
            display.setCurrent(nivel);
            //muestra opciones de submenú (grupoLabel, grupoON, grupoOFF)
            break;
        case 3:
            //pantalla.
            break;
        case 4:
            // Salimos de la Aplicacion
            notifyDestroyed();
            break;
    }
}
if (com == List.SELECT_COMMAND){
    if (dis == main_list){
        if (main_list.getSelectedIndex() >= 0){
            FindDevices();
            do_alert("Searching for devices...", Alert.FOREVER);
```

```
            }
        }
        if (dis == dev_list){
            StreamConnection conn = null;
            ServiceRecord service =
            (ServiceRecord)services.elementAt(dev_list.getSelectedIndex());
String url =
    service.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
            try {
                conn = (StreamConnection) Connector.open(url);
                dout = new DataOutputStream(conn.openOutputStream());
                display.setCurrent(menu);
            } catch (Exception e) {
                this.do_alert("Error Connecting" , 4000);
            }
        }
    }
    if(com == ok || bombaON == com || bombaOFF == com || grupoOFF == com ||
grupoON -- com || motorOFF -- com || motorON -- com){
//el cliente envia un comando
        try{
            do_alert("Opcion Selecionada:"+com.getLabel(), 4100);
            commandOptionSelected = com;
            dout.writeChars(commandOptionSelected.getLabel() + "\n");
            dout.flush();
            commandOptionSelected = null;
        } catch (Exception e) {
            this.do_alert("Error sending data" , 4200);
        }
    }
  }
public void showOptionsByEquip(String nombreEquip, Command cON, Command cOFF)
{
    List nivel = new List(nombreEquip, List.EXCLUSIVE);
    nivel.setCommandListener(this);
```

```
    nivel.addCommand(cON);
    nivel.addCommand(cOFF);
    nivel.addCommand(volver);
    display.setCurrent(nivel);
 }
 public void FindDevices(){
    try{
       devices = new Vector();
       LocalDevice local    = LocalDevice.getLocalDevice();
       DiscoveryAgent agent = local.getDiscoveryAgent();
       agent.startInquiry(DiscoveryAgent.GIAC,this);
    }catch(Exception e){
       this.do_alert("Erron in initiating search" , 4300);
    }
 }
public void FindServices(RemoteDevice device){
    try{
       UUID[] uuids  = new UUID[1];
       uuids[0]      = new UUID("27012f0c68af4fbf8dbe6bbaf7aa432a",false);
       local         = LocalDevice.getLocalDevice();
       agent         = local.getDiscoveryAgent();
       agent.searchServices(null,uuids,device,this);
    }catch(Exception e){this.do_alert("Error in initiating search" , 4400);}
 }
 public void deviceDiscovered(RemoteDevice remoteDevice,DeviceClass deviceClass) {
    devices.addElement(remoteDevice);
 }
 public void servicesDiscovered(int transID,ServiceRecord[] serviceRecord) {
    for (int x = 0; x < serviceRecord.length; x++ )
       services.addElement(serviceRecord[x]);
    try{
       dev_list.append(((RemoteDevice)devices.elementAt(currentDevice)).
                                    getFriendlyName(false),null);
    }catch(Exception e){this.do_alert("Error in initiating search" , 4500);}
 }
 public void inquiryCompleted(int param){switch (param) {
         case DiscoveryListener.INQUIRY_COMPLETED:
```

```
            if (devices.size() > 0){
                services = new Vector();
                this.FindServices((RemoteDevice)
                        devices.elementAt(0));
            }else
                do_alert("No device found in range",4600);
        break;
        case DiscoveryListener.INQUIRY_ERROR:
        this.do_alert("Inquiry error" , 4700);
        break;
        case DiscoveryListener.INQUIRY_TERMINATED:
            this.do_alert("Inquiry Canceled" , 4800);
        break;
       }
}
public void serviceSearchCompleted(int transID, int respCode) {
    switch(respCode) {
        case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
            if(currentDevice == devices.size() -1){
                if(services.size() > 0){
                    display.setCurrent(dev_list);
                }else
                    do_alert("The service was not found",4900);
            }else{
                currentDevice++;
                this.FindServices((RemoteDevice)devices.elementAt(currentDevice));
            }
        break;
        case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
            this.do_alert("Device not Reachable" , 5000);
        break;
        case DiscoveryListener.SERVICE_SEARCH_ERROR:
            this.do_alert("Service serch error" , 5100);
        break;
        case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
            this.do_alert("No records returned" , 5200);
```

```
            break;
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
               this.do_alert("Inqury Cancled" , 5300);
            break;
         }
      }
      public void do_alert(String msg,int time_out){
         if (display.getCurrent() instanceof Alert ){
            ((Alert)display.getCurrent()).setString(msg);
            ((Alert)display.getCurrent()).setTimeout(time_out);
         }else
{
            Alert alert = new Alert("Bluetooth");
            alert.setString(msg);
            alert.setTimeout(time_out);
            display.setCurrent(alert);
         }
      }
      public void pauseApp() {
      }
      public void destroyApp(boolean unconditional) {
      }
}
```

ANEXO B
CÓDIGO FUENTE SERVIDOR (PC)

```
import java.awt.BorderLayout;
import java.io.*;
import javax.bluetooth.*;
import javax.microedition.io.*;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class SERVER {
public final UUID uuid = new UUID(
                          "27012f0c68af4fbf8dbe6bbaf7aa432a", false);
   public final String name = "Echo Server";
   public final String url  =  "btspp://localhost:" + uuid
                          + ";name=" + name
                          + ";authenticate=false;encrypt=false;";
   LocalDevice local = null;
   StreamConnectionNotifier server = null;
   StreamConnection conn = null;
   public SERVER() {
     try{
      try {
         System.out.println("DETECTANDO DISPOSITIVO...");
         local = LocalDevice.getLocalDevice();
         local.setDiscoverable(DiscoveryAgent.GIAC);
         System.out.println("EMPEZANDO EL SERVICIO...");
         server = (StreamConnectionNotifier)Connector.open(url);
         System.out.println("ESPERANDO POR CONEXION CLIENTE...");
         conn = server.acceptAndOpen();
         System.out.println("CLIENTE CONECTADO");
         DataInputStream din   = new DataInputStream(conn.openInputStream());
         while(true){
            String cmd = "";
            char c;
            while (((c = din.readChar()) > 0) && (c!='\n') ){
               cmd = cmd + c;
            }
            System.out.println("RECIBIDO" + cmd);
            createWindow(cmd);
```

```
            }
        } finally {
                try{
                    if(server != null)
                    server.close();
                    if(conn != null)
                    conn.close();
                    } catch(IOException e) {}
        }
    }
catch (Exception  e) {System.out.println("Exception Occured: " + e.toString());}
    }
    public static void main (String args[]){
        SERVER echoserver = new SERVER();
    }
    public static void createWindow(String codigoEquipo){
        JFrame ventana = new JFrame("Acciòn Realizada");
        ventana.setDefaultCloseOperation(1);
        String mensaje = obtenerMensajeByCodigo(codigoEquipo);
        JComponent comp = new JTextArea(mensaje);
        ventana.getContentPane().add(comp, BorderLayout.CENTER);
        ventana.setSize(300, 200);
        ventana.setVisible(true);
    }
    public static String obtenerMensajeByCodigo(String codigo){
        String mensaje = "";
        if(codigo.equals("bombaON")){
            mensaje = "BOMBA BCI 2, Encendido";
        }else if(codigo.equals("bombaOFF")){
            mensaje = "BOMBA BCI 2, Apagado";
        }else if(codigo.equals("grupoON")){
            mensaje = "GRUPO ELECTROGENO 1, Encendido";
        }else if(codigo.equals("grupoOFF")){
            mensaje = "GRUPO ELECTROGENO 1, Apagado";
        }else if(codigo.equals("motorON")){
            mensaje = "MOTOR 1, Encendido";
        }else if(codigo.equals("motorOFF")){
```

```
            mensaje = "MOTOR 1, Apagado";
          }else{
            mensaje = "No se realizò ninguna acción";
          }
          return mensaje;
      }
}
```

ANEXO C
DATASHEETS

Hardware Specifications

General Conditions (V_{DD}= 3.3V and 25°C)

Recommended Operating Conditions

Rating	Min	Typical	Max	Unit
Temperature Range 3.0v – 3.3v	-30	-	85	°C
Temperature Range 2.7v – 3.6v	-20	-	80	°C
Supply Voltage V_{DD}	2.7	3.3	3.6	Volts
Signal Pin Voltage	-	3.3	-	Volts
RF Frequency	2400	-	2483.5	MHz

Absolute Maximum Ratings

Rating	Min	Typical	Max	Unit
Storage temperature range	-40	-	+150	°C
Supply voltage, V_{DD}	-0.3	-	+ 3.6	Volts
RF input power	-	-	-5	dBm

Current Consumption

Modes	Avg	Unit
Typical Power Consumption		
ACL data 115K Baud UART at max throughput (Master)	35.0	mA
ACL data 115K Baud UART at max throughput (Slave)	35.0	mA
Connection, no data traffic, master	18.0	mA
Connection, no data traffic, slave	28.0	mA
Connection in sniff (Tsniff=100ms), no data traffic, master	10.2	mA
Connection in sniff (Tsniff=100ms), no data traffic, slave	10.8	mA
Connection in sniff (Tsniff=375ms), no data traffic, master	2.75	mA
Connection in sniff (Tsniff=375ms), no data traffic, slave	3.50	mA
Standby, without deep sleep	16.5	mA
Standby, with deep sleep	0.16	mA
Page/Inquiry scan, deep sleep	2.1	mA
Peak current	90	mA

I/O Operating Characteristics

Symbol	Parameter	Min	Max	Unit	Conditions
V_{IL}	Low-Level Input Voltage	-	0.8	Volts	
V_{IH}	High-Level Input Voltage	2.0	-	Volts	
V_{OL}	Low-Level Output Voltage	-	0.4	Volts	I_{OL} = 2mA
V_{OH}	High-Level Output Voltage	2.4	-	Volts	I_{OH} = 2mA
I_{OL}	Low -Level Output Current	-	2.2	mA	V_{OL} = 0.4 V
I_{OH}	High-Level Output Current	-	3.1	mA	V_{OH} = 2.4 V
I_I	Input Leakage Current	-1	+1	µA	@V_I = 3.3V or 0V
V_{T+}	Schmitt Trigger Low-High	1.47	1.50	Volts	
V_{T-}	Schmitt Trigger High-Low	0.89	0.95	Volts	
R_{PU}	Pull-up Resistor	53	113	KΩ	Resistor Turned On
R_{PD}	Pull-down Resistor	43	118	KΩ	Resistor Turned On
C_I	Input Capacitance		7.5	pF	

Selected RF Characteristics

Parameters	Conditions	BT Spec	Typical	Unit
Antenna load			50	ohm
Radio Receiver				
Sensitivity level	BER < .001 with DH5	≤ -70	-85	dBm
Maximum usable level	BER < .001 with DH1	≥ -20	-5	dBm
Input VSWR			2.5:1	
Radio Transmitter				
Maximum output power	50 Ω load	-6 to +4	+1	dBm
Power control range		≥ 16	30	dB
Power control resolution		2 to 8	4	dB
Initial Carrier Frequency Tolerance		± 75	18	kHz
20 dB Bandwidth for modulated carrier		≤ 1000	930	kHz

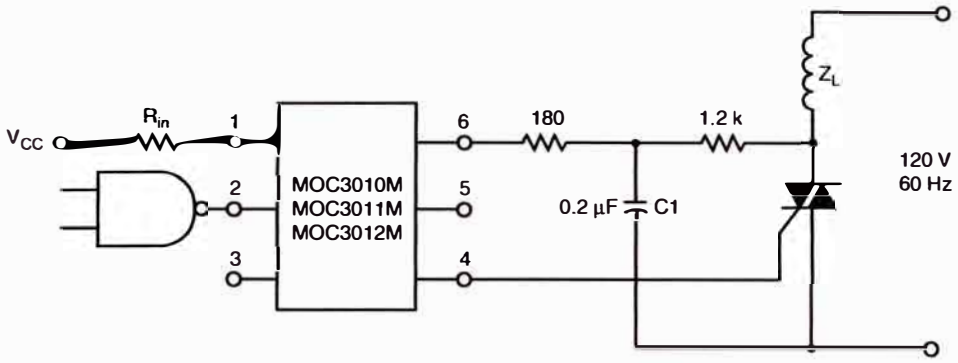
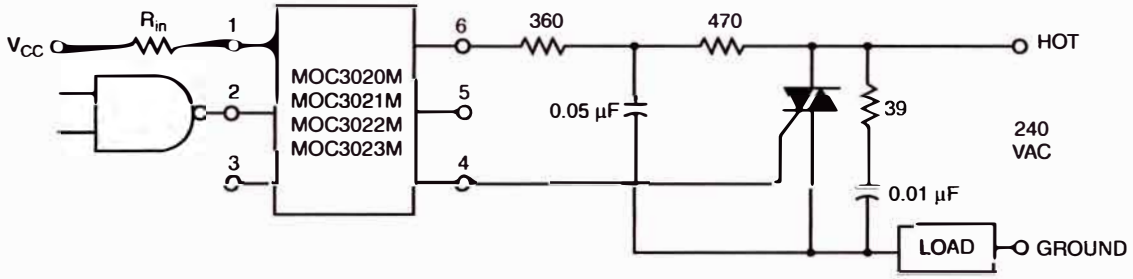


Figure 8. Inductive Load with Sensitive Gate Triac ($I_{GT} \leq 15$ mA)



In this circuit the "hot" side of the line is switched and the load connected to the cold or ground side.

The 39 ohm resistor and 0.01µF capacitor are for snubbing of the triac, and the 470 ohm resistor and 0.05 µF capacitor are for snubbing the coupler. These components may or may not be necessary depending upon the particular and load used.

Figure 9. Typical Application Circuit

FAIRCHILD
SEMICONDUCTOR®

6-PIN DIP RANDOM-PHASE OPTOISOLATORS TRIAC DRIVER OUTPUT (250/400 VOLT PEAK)

MOC3010M MOC3011M MOC3012M MOC3020M MOC3021M MOC3022M MOC3023M

Fig. 1 LED Forward Voltage vs. Forward Current

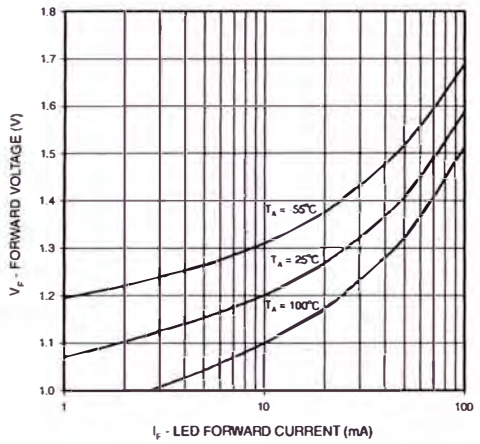


Fig. 2 On-State Characteristics

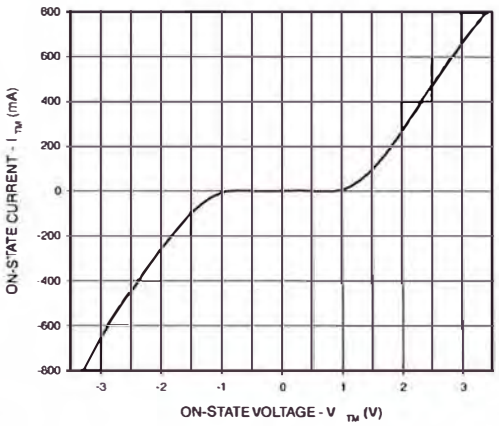


Fig. 3 Trigger Current vs. Ambient Temperature

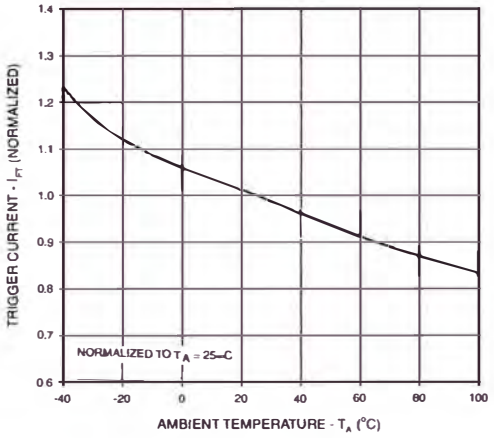


Fig. 4 LED Current Required to Trigger vs. LED Pulse Width

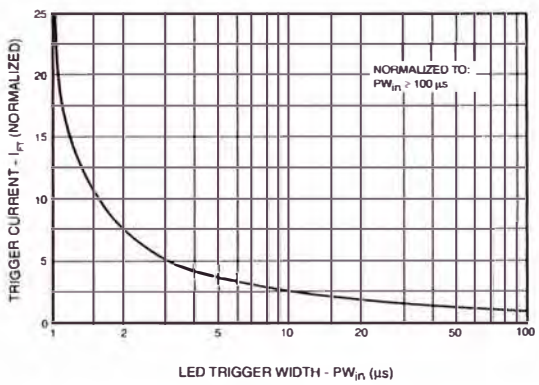


Fig. 5 dv/dt vs. Temperature

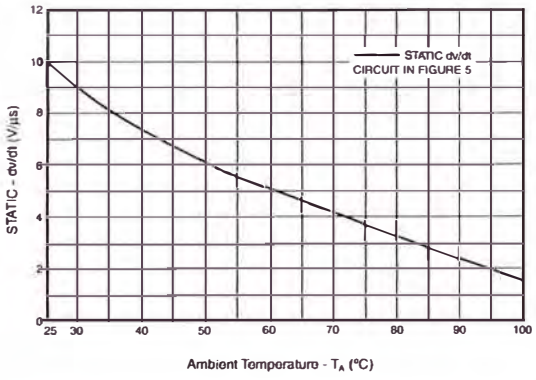
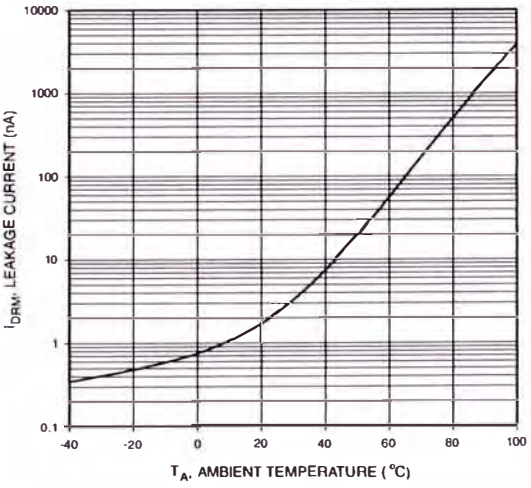
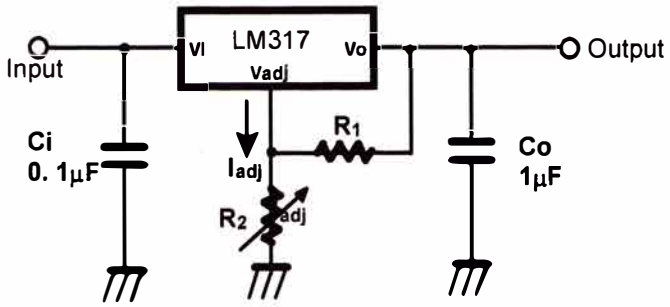


Fig. 6 Leakage Current, I_{DRM} vs. Temperature



Typical Application



$V_O = 1.25V\ (1 + R_2/R_1) + I_{adj} R_2$

Figure 5. Programmable Regulator

- C_i is required when regulator is located an appreciable distance from power supply filter.
  C_O is not needed for stability, however, it does improve transient response.
  Since I_{ADJ} is controlled to less than 100μA, the error associated with this term is negligible in most applications.

Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Input-Output Voltage Differential	$V_I - V_O$	40	V
Lead Temperature	T_{LEAD}	230	°C
Power Dissipation	P_D	Internally limited	W
Operating Junction Temperature Range	T_j	0 ~ +125	°C
Storage Temperature Range	T_{STG}	-65 ~ +125	°C
Temperature Coefficient of Output Voltage	$\Delta V_O/\Delta T$	±0.02	%/°C

Note 1: Absolute Maximum Ratings: are those values beyond which the safety of the device cannot be guaranteed. The device should not be operated at these limits. The parametric values defined in the Electrical Characteristics tables are not guaranteed at the absolute maximum ratings. The "Recommended Operating Conditions" table will define the conditions for actual device operation.

Electrical Characteristics

(V_I-V_O = 5V, I_O = 0.5A, 0°C ≤ T_J ≤ +125°C, I_{MAX} = 1.5A, P_{DMAX} = 20W, unless otherwise specified)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Line Regulation (Note2)	Rline	T_A = +25°C 3V ≤ V_I - V_O ≤ 40V	-	0.01	0.04	%/ V
		3V ≤ V_I - V_O ≤ 40V	-	0.02	0.07	%/ V
Load Regulation (Note2)	Rload	T_A = +25°C, 10mA ≤ I_O ≤ I_{MAX} V_O< 5V V_O ≥ 5V	-	18 0.4	25 0.5	mV%/V_O
		10mA ≤ I_O ≤ I_{MAX} V_O < 5V V_O ≥ 5V	-	40 0.8	70 1.5	mV%/V_O
Adjustable Pin Current	I_{ADJ}	-	-	46	100	µA
Adjustable Pin Current Change	ΔI_{ADJ}	3V ≤ V_I - V_O ≤ 40V 10mA ≤ I_O ≤ I_{MAX} P_D ≤ P_{MAX}	-	2.0	5	µA
Reference Voltage	V_{REF}	3V ≤ V_{IN} - V_O ≤ 40V 10mA ≤ I_O ≤ I_{MAX} P_D ≤ P_{MAX}	1.20	1.25	1.30	V
Temperature Stability	S_{TT}	-	-	0.7	-	%/V_O
Minimum Load Current to Maintain Regulation	$I_{L(MIN)}$	V_I - V_O = 40V	-	3.5	12	mA
Maximum Output Current	$I_{O(MAX)}$	V_I - V_O ≤ 15V, P_D ≤ P_{MAX} V_I - V_O ≤ 40V, P_D ≤ P_{MAX} T_A=25°C	1.5 -	2.2 0.3	- -	A
RMS Noise, % of V_{OUT}	e_N	T_A= +25°C, 10Hz ≤ f ≤ 10kHz	-	0.003	0.01	%/V_O
Ripple Rejection	RR	V_O = 10V, f = 120Hz without C_{ADJ} C_{ADJ} = 10µF (Note3)	66	60 75	-	dB
Long-Term Stability, T_J = T_{HIGH}	ST	T_A = +25°C for end point measurements, 1000HR	-	0.3	1	%
Thermal Resistance Junction to Case	$R_{\theta JC}$	-	-	5	-	°C/W

Note 2: Load and line regulation are specified at constant junction temperature. Change in V_D due to heating effects must be taken into account separately. Pulse testing with low duty is used. (P_{MAX} = 20W)

Note 3: C_{ADJ}, when used, is connected between the adjustment pin and ground.

Samsung GT-S5233t Specifications

Manufacturer/Brand:	Samsung
Model:	GT-S5233t
Main Display Size:	240 x 400 pixels
Since Year:	2009

The Samsung GT-S5233t supports Java applications.

Java/J2ME/JVM Developer Specifications

MIDP:	2.0
CLDC:	1.1
Platform:	GT-S5233T/S5233TUBJA1
Java Canvas Size:	240 x 262 pixels
Java Fullscreen Canvas Size:	240 x 320 pixels

APIs/JSRs:
Touch screen
JSR-75 - File System
JSR-75 - PIM
JSR-82 - Bluetooth
JSR-82 - OBEX
JSR-120 - Wireless Messaging
JSR-135 - Mobile Media
JSR-135 - Mobile Media (Camera)
JSR-172 - Web Services (JAXP)
JSR-172 - Web Services (JAX-RPC)
JSR-177 - Security And Trust (APDU)
JSR-177 - Security And Trust (Crypto)
JSR-177 - Security And Trust (PKI)
JSR-184 - Mobile 3D Graphics
JSR-185 - JTWI
JSR-205 - Wireless Messaging 2
JSR-211 - Content Handling
JSR-226 - 2D Graphics/SVG
JSR-229 - Payments
JSR-238 - Internationalization
JSR-248 - MSA - 1.0 Subset
JSR-256 - Sensors

Other Features

microedition.commports:	COM1, USB
supports.mixing:	false
supports.audio.capture:	true
supports.video.capture:	true
supports.recording:	true
Audio Encodings:	encoding=audio/amr
Video Encodings:	encoding=jpeg
Video Snapshot Encodings:	encoding=jpeg
Display.numColors():	65536
Display.isAlphaLevels():	256
Display.flashBacklight():	true
Display.vibrate():	true

Browser Specifications

User Agent:	SAMSUNG-GT-S5233T/1.0 SHP/VPP/R5 Jasmine/0.8 NexPlayer/2.12 SMM-MMS/1.2.0 profile/MIDP-2.1 configuration/CLDC-1.1
WAP Profile:	"http://wap.samsungmobile.com/uaprof/GT-S5233T.xml"
Accept:	text/html, application/xml, image/vnd.wap.wbmp, image/png, image/jpeg, image/gif, image/bmp, application/vnd.wap.xhtml+xml, application/xhtml+xml, application/vnd.wap.multipart.mixed, multipart/mixed, text/vnd.wap.wml, application/vnd.wap.wmlc, application/vnd.oma.dd+xml, text/vnd.sun.j2me.app-descriptor, application/java-archive, */*
Accept_Encoding:	gzip

Most Popular Software

View All

Free Samsung GT-S5233t Games
Free Samsung GT-S5233t Apps
Free Samsung Games
Free Samsung Apps
Samsung Specs

Mobile Device Forum

Game Reviews by Other Samsung GT-S5233t Owners

8.2 - Lordmancer Ahmadreza from Kashan/Iran - August 13, 2011

9.4 - "Bluevibe" darwin from costa rica - September 3, 2010

Very funny

BIBLIOGRAFIA

[1]. El futuro de las comunicaciones:
http://www.hispazone.com/Articulo/72/Bluetooth:-El-Futuro-de-lasComunicaciones-I.html

[2]. Api Java Jsr82:
http://janoxmoraga.blogspot.com/2010/06/api-java-bluetooth-jsr-82.html

[3]. Arquitectura de Protocolos Bluetooth:
http://www.seguridadmobile.com/bluetooth/especificacion-bluetooth.

[4]. Moviles:
http://desarrollomoviles.blogspot.com/2008/11/y-llegamos-al-j2me.htmlgeneral.

[5]. JSRs: http://www.jcp.org/en/jsr/all

[6]. The Java Plataform :
http://ikt.hia.no/aml/public-projects/J2ME/Rams-J2ME/J2ME.html

[7]. Introducción al J2ME:
http://caraballomaestre.blogspot.com/2009/05/introduccion-j2me.html

[8]. Bluecove:
http://bluecove.org/

[9]. Miguel Angel Lozano "Programación de Dispositivos Moviles con J2ME"
Universidad de Alicante 2004

[10]. Sergio Galvez Rojas y Lucas Ortega Diaz "Java a Tope"
Universidad de Malaga, 2003.

[11]. Bruce Hopkins and Ranjith Anthony, "Bluetooth For Java"
Apress- Estados Unidos, 2003.

[12]. Pedro Daniel Borches Jurado "SOPORTE BLUETOOTH", Universidad Carlos III de Madrid, 2004

[13]. Miguel Angel Lozano "Programación de Dispositivos Moviles con J2ME"
Universidad de Alicante 2004

[14]. C.J. Savant - Jr. Martin S. "Diseño Electrónico" 3ra Edición
Prentice Hall, 2000.