UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ESTUDIO DE LA TECNOLOGÍA J2ME EN EL DESARROLLO DE APLICACIONES MÓVILES PARA CONEXIONES INTERACTIVAS PC - DISPOSITIVO MÓVIL A TRAVÉS DE LA TECNOLOGÍA DE CONEXIÓN INALÁMBRICA BLUETOOTH

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

IAN MOISES RANGEL VILLAGRAN

Asesor: JESÚS HERNÁNDEZ CABRERA

SEPTIEMBRE 2009

"En el ajedrez, como en la vida, el adversario más peligroso es uno mismo"

Vassily Smyslov

AGRADECIMIENTOS A:

MI PERSONA

Por haber logrado alcanzar esta meta a base de esfuerzo y dedicación. Lo que me brinda gran felicidad, satisfacción y motivación para continuar adelante con mis metas futuras.

MIS PADRES

Juan Carlos Rangel Colinas y María Esther Villagrán Piña por brindarme siempre todo su apoyo, comprensión, confianza, amor y sobre todo el mayor de sus esfuerzos y dedicación para mi superación personal y profesional, son las personas que más admiro y principal motivo de este y todos mis logros que también son logros suyos, muchas gracias por darme la más grande y mejor de las herencias, "mi profesión".

MIS HERMANOS

Hanz, Lua, Miguel, Arturo y Mariana por su cariño, admiración, apoyo, comprensión y confianza que me han brindado durante todo el tiempo que hemos compartido juntos desde nuestra infancia, cada uno de ustedes a formado parte de mi motivación de todos los días para seguir adelante y cumplir mis metas.

MI FAMILIA

Por todo el apoyo y cariño que me han brindado durante todo este tiempo.

MIS AMIGOS

Por su gran amistad y apoyo que me han brindado todo este tiempo, por todos los grandes momentos que hemos compartido, en especial a los cinco de siempre su amistad es invaluable.

MIS MAESTROS

Por toda su labor realizada para mi desarrollo profesional. En especial a mi asesor Jesús Hernández Cabrera por su amistad, paciencia y dedicación para la realización de este trabajo.

LA UNAM Y A LA FES ARAGÓN

Por brindarme la oportunidad de forjarme una profesión.
"Por mi raza hablará el espíritu".

ÍNDICE

INTRODUCCIÓN

JUSTIFICACIÓN Y DELIMITACIÓN

OBJETIVO GENERAL

OBJETIVOS ESPECÍFICOS

METODOLOGÍA

INTRODUCCIÓN

A lo largo de su historia, y con la idea de facilitar sus tareas cotidianas los seres humanos hemos inventado una gran cantidad diversa de técnicas y herramientas para resolver distintos tipos de problemas en diferentes situaciones, algunas por necesidad y otros simplemente por mejorar la calidad de vida. Es de esta manera que se comienza a dar principio a lo que hoy conocemos como *tecnología*.

La palabra *tecnología* tiene su origen griego y esta formada por *tekne* (εχνη): técnica, oficio, y por *logos* (λογος): ciencia, conocimiento. Pero, realmente ¿a qué se refiere esta palabra de tecnología?, seguramente muchas veces hemos escuchado frases como: *La tecnología de punta, Lo último en tecnología, La tecnología a tu alcance* entre otras y al escucharlas nos viene a la mente cualquier cosa relacionada con las computadoras y de manera equivocada se relaciona la palabra *tecnología* con la *computación* como uno mismo.

Para comprender mejor a lo que se refiere esta palabra, de forma general hay que mencionar a la *tecnología* como un conjunto de técnicas y conocimientos que nos van a permitir desarrollar distintos tipos de herramientas para aprovechar de manera práctica el conocimiento obtenido y de esta forma poder satisfacer algún tipo de necesidad para el hombre.

Una de estas herramientas que se han desarrollado son las computadoras, las cuales actualmente son una de las herramientas más utilizadas por el hombre en diversas áreas, convirtiéndose en un recurso indispensable para realizar la mayoría de las actividades y tareas cotidianas.

La computadora es un conjunto de dispositivos electrónicos y/ó mecánicos capaces de realizar gran cantidad de cálculos de manera rápida, eficaz y autónoma, obedeciendo a un conjunto de instrucciones específicas de un usuario ó programa. A través de dispositivos de entrada reciben tipos de datos específicos los cuales de forma ordenada y sistematizada son procesados para posteriormente ser mostrados como resultado a través de los dispositivos de salida

Las computadoras han tenido un desarrollo sorprendente y acelerado. En sus inicios estas tenían grandes tamaños, consumían gran cantidad de energía y eran demasiado costosas para su fabricación; su propósito principal era el de realizar gran cantidad de operaciones matemáticas y facilitar la labor de los científicos; conforme continuaron evolucionando y mejorando su capacidad de procesamiento surgieron los circuitos de transistores los cuales permitieron reducir sus tamaños y abaratar sus costos; pero no fue si no hasta la creación de los circuitos integrados que se dio un avance significativo ya que permitieron una mayor reducción en tamaño, una mayor velocidad de procesamiento y su comercialización a una mayor cantidad de usuarios. De esta forma las computadoras dejan de ser de uso exclusivo para usuarios expertos, y comienzan a utilizarse con propósitos múltiples.

Al transcurrir el tiempo y gracias a su evolución surge la *computación móvil* la cual hace uso de la computación para ser utilizada en diferentes tipos de dispositivos y equipos portátiles para lograr que estos realicen sus propias funciones. En la actualidad el desarrollo de este tipo de dispositivos ha crecido de manera gradual teniendo como resultado gran variedad de dispositivos móviles con diferentes funciones; entre ellos destacan Laptop, Pocket PC, reproductores, teléfonos celulares, entre otros.

A pesar de su gran evolución las computadoras por si solas no son capaces de operar ni efectuar algún tipo de tarea, son simplemente un conjunto de cables y circuitos conectados entre si que están a la espera de recibir una instrucción por parte de los humanos para poder llevar acabo alguna operación. Entonces: ¿cómo es qué vamos a poder comunicarnos con el conjunto de cables y circuitos para darles instrucciones y que estos entreguen los resultados deseados?.

Es de esta forma como surgen los lenguajes de programación de alto nivel, los cuales nos permiten escribir programas con el fin de establecer una comunicación con las computadoras para poder transmitir nuestras peticiones y obtener resultados. Pero por otro lado sabemos que un conjunto de circuitos no sería capaz de entender ningún lenguaje que nosotros conozcamos, por muy sencillo y que este pueda ser. Los circuitos solo son capaces de entender el lenguaje binario (1,0), que es el representativo de presencia y ausencia de energía, por lo que es necesario hablarle a la computadora en su propio lenguaje, ó traducir nuestro lenguaje a lenguaje binario, todo esto a través de herramientas desarrolladas especialmente para llevar acabo esta labor, las cuales reciben el nombre de

traductores. A estos traductores se les conoce comúnmente como compiladores ó intérpretes.

Son varios los lenguajes de programación de los que actualmente se dispone, cada uno de ellos con características específicas para el desarrollo de programas para diversos tipos de computadoras y dispositivos móviles. Entre los más populares destacan: Pascal, Basic, C, C++, Java, C#, Php, Perl entre otros.

En el trabajo de investigación que se realizará, se hará un estudio de la tecnología J2ME del lenguaje de programación Java con la especificación JSR-82 la cual nos permite el desarrollo de aplicaciones para dispositivos móviles haciendo uso del dispositivo de conexión inalámbrica Bluetooth que estos incluyan; y con esto poder desarrollar aplicaciones móviles para comunicaciones interactivas entre la PC y dispositivo móvil vía Bluetooth.

El primer capítulo presentará los antecedentes teóricos los cuales nos brindarán una mejor perspectiva para abordar el tema de nuestro estudio. Aquí se mencionarán la evolución de los teléfonos celulares desde sus inicios principalmente por ser uno de los dispositivos móviles en los que la tecnología J2ME se ha extendido ampliamente, también abarcará los orígenes y funcionamiento de la tecnología Bluetooth así como la descripción de algunas tecnologías existentes para el desarrollo de aplicaciones móviles.

En el segundo capítulo se hará una revisión más profunda del funcionamiento de la tecnología J2ME así como de algunas de las principales API´s existentes para el desarrollo de aplicaciones móviles.

En el tercer capítulo se mostrará a detalle la especificación JSR–82 en J2ME y su implementación Bluecove en J2SE para el desarrollo de aplicaciones para el manejo del radio Bluetooth en la PC ó el dispositivo móvil.

En el cuarto y último capítulo se desarrollará una aplicación para evaluar con un caso práctico la tecnología J2ME con uso de la especificación JSR–82 y su potencialidad. El sistema que se va a desarrollar permitirá interactuar una PC con un teléfono celular vía Bluetooth, esto para la difusión de eventos escolares.

Justificación y Delimitación

Se eligió el estudio de la tecnología J2ME porque es de distribución libre, lo cual no implica gastos por usarla, también esta va acaparando más el mercado para el desarrollo de aplicaciones móviles, siendo implementada cada vez en un mayor número de dispositivos móviles. Actualmente nos permite tener acceso para controlar hardware Bluetooth implementando la especificación JSR-82, la cual ya esta disponible en gran parte los dispositivos móviles.

J2ME es seguro, ya que el hecho de que las aplicaciones sean ejecutadas dentro del confinamiento establecido por la máquina virtual Java es una seguridad de ejecución del código fuente a las demás aplicaciones, por lo que no puede afectar a ninguna otra aplicación fuera de la máquina virtual del dispositivo móvil.

La Tecnología J2ME ofrece cierta portabilidad de código ya que a través de la máquina virtual que implemente el dispositivo móvil es posible escribir una aplicación y ejecutarla en diversos dispositivos móviles. Por lo que las aplicaciones desarrolladas con esta tecnología pueden ser distribuidas de manera fácil de un dispositivo móvil a otro. Todo esto con la única limitante de saber la configuración y perfil base en el que se escribirá la aplicación.

La tecnología J2ME es robusta y lleva gran tiempo en el mercado para el desarrollo de aplicaciones móviles, además de contar con una amplia comunidad de desarrolladores lo que presenta una mayor facilidad al momento de buscar ayuda ó soporte acerca de su funcionamiento.

El beneficio es para distintos tipos de personas, en las que su estudio se vea enfocado en la computación, permite que puedan elegir entre más opciones en las cuales deseen desarrollarse profesionalmente, también brinda grandes beneficios a los usuarios de los dispositivos móviles ya que son los que finalmente demandan las aplicaciones que desean tener instaladas para cubrir alguna necesidad.

Objetivo General

Describir la tecnología J2ME con la especificación JSR-82 para desarrollar aplicaciones móviles para conexiones interactivas PC – Dispositivo Móvil vía Bluetooth, para así exponer su funcionamiento eficaz que brinda en esta área.

Objetivos Específicos

- Vislumbrar la potencialidad en el mercado del desarrollo de aplicaciones móviles, ya que este va en gran aumento.
- Revisar y utilizar los recursos que nos brinda la Ingeniería de Software para el desarrollo de aplicaciones móviles.
- Desarrollar un sistema, el cual a través de su proceso de desarrollo permitirá aplicar y evaluar de forma práctica todo lo estudiado acerca de la tecnología J2ME con especificación JSR-82.

Metodología

La parte principal del estudio es la tecnología J2ME con la especificación JSR – 82, por lo que se estudiará y utilizará el API proporcionada por Sun Microsystems, demostrando su funcionamiento con distintos ejemplos prácticos, así también las pruebas que deban desarrollarse se realizarán en equipo propio para así determinar el funcionamiento correcto de lo revisado.

Para esto se utilizarán herramientas IDE las cuales ofrecen un entorno de desarrollo para la programación, en esta caso emplearemos el IDE NetBeans 6.1 proporcionado de forma gratuita por Sun Microsystems, el cual tiene el entorno de desarrollo para aplicaciones J2ME.

Se hará uso de distintos artículos proporcionados por Sun Microsystems los cuales dan seguimiento, describen el funcionamiento y uso de la tecnología J2ME. También se recopilará información de distintos libros y tutoriales orientados al desarrollo de aplicaciones móviles bajo esta tecnología.

CAPÍTULO 1: ANTECEDENTES

PARTE 1

EVOLUCIÓN DEL TELÉFONO CELULAR

PARTE 2

TECNOLOGÍA BLUETOOTH

PARTE 3

DESARROLLO DE APLICACIONES MÓVILES

PARTE 1
EVOLUCIÓN DEL TELÉFONO CELULAR

Introducción

Los teléfonos celulares forman ya parte esencial en nuestra forma de vida. Cada día encontramos que son más los usuarios que emplean estos dispositivos, con el transcurso del tiempo se vuelven cada vez más pequeños y sofisticados. Aunque su principal uso radica en la comunicación telefónica, con el paso del tiempo se han desarrollado nuevas formas de comunicación y otras capacidades entorno a ellos, como lo son: teléfonos celulares con reproductores de audio, video, y capacidad de ejecución de diversas aplicaciones como juegos, agendas, entre otras. Pero ¿Cómo fue que sucedió esto?

En la naturaleza cualquier ser vivo de alguna u otra forma tiene la necesidad de comunicarse con el medio que lo rodea, a través de la comunicación el hombre ha podido intercambiar y expresar ideas, conocimientos y pensamientos logrando relacionarse con otros y obtener un mejor desarrollo.

"Genéricamente, la comunicación es la transferencia de información con sentido desde un lugar (llamado remitente, fuente, origen o transmisor) a otro (que recibe el nombre destino o receptor). Por otra parte, información se define como un patrón físico al cual se le ha asignado un significado comúnmente acordado (mensaje). El patrón debe ser único, separado y distinto (de otro modo no sería posible determinar unívocamente el significado del mensaje) y debe poder ser enviado por el transmisor y ser detectado y entendido por el receptor".[1]

En el párrafo anterior el autor muestra de forma general lo que es la comunicación y como se lleva acabo. Podemos distinguir tres elementos fundamentales que la componen: transmisor, canal o medio de transmisión y receptor. Una de las principales formas de comunicarse del hombre es a través del habla y la escritura, la cual por medio de significados, palabras y símbolos representa información que es transmitida y recibida a

[1]ROLDAN MARTINEZ, David, "Comunicaciones Inalámbricas" p.3.

través del uso de sus sentidos (oído, vista y tacto) y el medio que lo rodea (En el caso del habla: voz-transmisor, aire-medio de transmisión, oído-receptor).

En el transcurso del tiempo y con el avance de la tecnología se han desarrollado diferentes tipos de medios de comunicación como el teléfono, radio, televisión, periódico, entre otros, todo con el objetivo de poder mantener la comunicación entre las personas y a través de grandes distancias a lo que se denomina como telecomunicaciones.

El teléfono es uno de los medios de comunicación de mayor importancia y más utilizado en el mundo, con él se puede transmitir la voz humana a grandes distancias lo que permite a las personas establecer una conversación desde cualquier ubicación en la que se encuentre un teléfono conectado al cableado de sistema de red telefónica que brinde el servicio (Telefonía fija).

La necesidad de poder estar comunicado desde cualquier ubicación sin tener que depender de un teléfono conectado al cableado de la red telefónica, se volvió necesidad entre varias personas, principalmente aquellas que se dedicaban a los negocios, necesitaban estar en contacto con sus clientes, proveedores y oficinas. Para resolver este tipo de necesidad es que surge la telefonía móvil, la cual por medio de teléfonos celulares permite realizar llamadas y establecer conversaciones desde cualquier ubicación sin tener que estar conectado al cableado de la red telefónica.

La comunicación inalámbrica fue algo fundamental que logró permitir el desarrollo de la telefonía celular, este tipo de comunicación permite el empleo del aire y ondas de radio para la comunicación, por lo que no requiere de un medio físico (cableado) para poder establecer la comunicación entre 2 puntos.

Red de Telefonía Celular

La forma en que opera la red de telefonía celular es por medio de la división del área de cobertura en celdas o células, cada una de ellas cuenta con su estación base que transmite un conjunto de canales de tráfico y señalización por lo que cada estación se encarga de darle servicio a la célula o celda a la que pertenece. Cada zona de cobertura necesita tener

su oficina central para controlar todas las conexiones a las estaciones de cada celda y a las de la red telefónica fija.

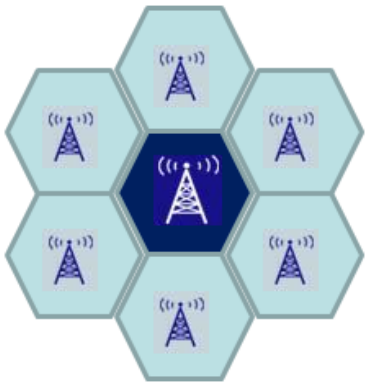


Figura 1.1

Operación de celdas de la Red de Telefonía Celular

Generaciones en la Red de Telefonía Celular

El avance en la red de telefonía celular se ha caracterizado por el desarrollo de nuevas tecnologías, las cuales surgen para cumplir las distintas demandas y necesidades del mercado comercial. Estos avances tecnológicos se clasifican en 3 generaciones que representan los avances más significativos.

Primera generación (1G): Esta generación aparece comercialmente en 1982, se caracteriza por ser un sistema totalmente analógico especialmente utilizado para la transmisión de voz.

Segunda generación (2G): Aparece en 1992 se caracteriza por migrar los sistemas analógicos a sistemas digitales, lo que permite tener mayor cantidad de usuarios conectados a la red celular, así como tener una mayor seguridad en la comunicación. Algo importante que surgió en esta generación es la mayor velocidad y calidad en la transmisión de voz, así como la incorporación de envió de mensajes (SMS).

Tercera generación (3G): Esta generación comienza a finales de 1999 y principios del 2000. La demanda de más acho de banda, y la demanda de manejo de mayor cantidad de datos influye para que surja esta generación, la cual permite el envío a mayor velocidad de video, imágenes, datos para la navegación en Internet entre otros servicios.

Cambios en el Teléfono Celular

El avance y cambios acelerados en las redes de telefonía celular han permitido en gran medida la evolución de los teléfonos celulares, ya que estos han tenido que adaptarse a las necesidades de los usuarios y funcionamiento de las diferentes tecnologías empleadas en la red celular.

A lo largo de su evolución se ha logrado reducir tamaño y peso, así como mejorar el rendimiento en sus baterías, pero una de las características principales que hay que destacar en la evolución de este dispositivo es la adaptación que ha podido aplicar de otras tecnologías para incorporar nuevas funcionalidades.

Primeramente se adaptan 2 tipos de tecnologías, una es la tecnología telefónica y la otra la tecnología de comunicación inalámbrica lo cual le permite cumplir su principal objetivo de establecer conversaciones telefónicas de forma inalámbrica. Al momento de migrarse los sistemas análogos a digitales y al ser los teléfonos celulares adaptados a estos cambios se incrementan más las posibilidades de ampliar sus funciones. Aparece el soporte para envío de mensajes, el teléfono celular deja de ser utilizado exclusivamente para realizar llamadas telefónicas. Pronto se descubre que este dispositivo puede emplear más funcionalidades basándose en la tecnología de la computación, la cual pasaba por un desarrollo acelerado. Conforme la computación seguía prosperando, los cambios y avances que se generaban permitían incorporar nuevas funcionalidades a los teléfonos celulares. Entre estas funcionalidades surgieron teléfonos celulares con pequeños sistemas operativos que incluían: agendas, calculadoras, calendarios, juegos entre otras aplicaciones.

Al paso del tiempo los teléfonos celulares incrementaron sus capacidades, principalmente se obtiene mejor procesamiento, mayor capacidad de almacenamiento de datos, mejor resolución en pantallas e incorporación de algunos dispositivos hardware como cámaras, reproductores de audio, intercomunicadores con tecnología Bluetooth entre otros, permitiendo ampliar aún más su funcionalidad. De entre estas nuevas incorporaciones Bluetooth es de las más expandidas en los teléfonos celulares, permite interactuar con otros dispositivos intercambiando datos de forma inalámbrica a cortas distancias, en la siguiente parte describiremos esta tecnología con más detalle.

PARTE 2
TECNOLOGÍA BLUETOOTH

Introducción

En un principio la tecnología Bluetooth fue desarrollada por la empresa Ericsson en el año de 1994. Para el año de 1998 se reúnen varias empresas para forman un grupo llamado Bluetooth Special Interest Group (Bluetooth SIG) el cual se encargaría del desarrollo de las especificaciones para la tecnología Bluetooth.

¿Qué es Bluetooth?

Bluetooth pertenece al tipo de redes WPAN (WIRELESS PAN) y es la tecnología que permite la conectividad inalámbrica entre dispositivos a cortas distancias. Esta tecnología fue diseñada con el propósito principal de conectar y comunicar dispositivos entre si, sin la necesidad de utilizar algún tipo de cables. Es de tamaño pequeño, bajo consumo de energía, y un bajo precio, por lo que ha podido ser adaptado en diferentes tipos de dispositivos electrónicos como: teléfonos móviles, auriculares inalámbricos, impresoras, cámaras fotográficas, PDA, Laptop, PC, teclado, ratón entre otros.

El nombre de esta tecnología surge en memoria al rey danés Harald I el cual era apodado Blåtand (en inglés "blue-toothed") quien en el siglo X logró la unificación varios pueblos.

Características Principales

La tecnología Bluetooth opera a 2.4Ghz sobre la banda de radio ISM (Industrial, Scientific and Medical) que internacionalmente esta reservada para un uso no comercial, en la que puede transmitir datos hasta una velocidad máxima de 1Mbps con un alcance variado que depende de la potencia radiada del transmisor. Existen definidos tres clases de transmisores: Clase I, Clase II y Clase III.

En la Clase I el alcance es de hasta 100 metros, utiliza una potencia de 100 mW, y se produce una perdida de señal de 20dBm. La Clase II tiene un alcance de 15 a 20 metros, utiliza una potencia de 2,5 mW con una perdida de señal de 4 dBm. Para la Clase III el

alcance es 10 metros, la potencia utilizada es de 1 mW y la perdida de la señal es de 0dBm, por lo que es el tipo de transmisor usado por la mayor parte de los dispositivos que incorporan la tecnología Bluetooth.

Funcionamiento

La banda de frecuencia ISM (Industrial, Scientific and Medical) esta abierta para ser utilizada por cualquiera sin necesidad de tener una licencia, lo que puede originar problemas de interferencia entre los dispositivos que se conectan a través de esta. Para evitar las interferencias se utiliza la técnica FHSS (*Frequency Hopping Spread Spectrum*), esta técnica de salto de frecuencia permite dividir la banda de frecuencia de 2.4Ghz en varios canales denominados saltos. Los dispositivos Bluetooth transmiten utilizando una secuencia de canales conocida por el emisor y receptor que al cambiar de canales a una velocidad de 1600 saltos/segundo en modo full dúplex (transmisión y recepción simultanea) de manera pseudo-aleatoria logran evitar interferencias con otras ondas de radio.

Cada canal es dividido en slots que son intervalos de 625 μs, un slot ocupa cada salto de frecuencia. La tecnología Bluetooth permite realizar conexiones punto a punto y punto a multipunto en ambos casos un dispositivo actuará como maestro y se encargará de controlar el tráfico de los datos que se genera entre los otros dispositivos (1 o más) que actuarán como esclavos recibiendo y enviando señales al dispositivo maestro. Cada salto de frecuencia lo determina la secuencia de la señal que es establecida por el dispositivo maestro y la frecuencia de reloj del mismo. Para que el dispositivo esclavo pueda sincronizarse debe ajustar su propia frecuencia de reloj para compartir el mismo salto de frecuencia.

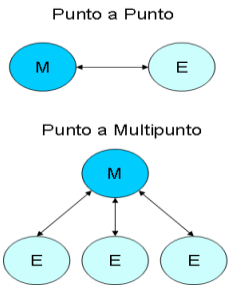


Figura 1.2

Tipos de Conexiones

Toda la información que se intercambia a través de una conexión entre dos dispositivos Bluetooth se realiza a través de paquetes, cada paquete esta formado por un conjunto de slots. Primeramente cada paquete inicia con un código de acceso de 72 bits, derivado del dispositivo maestro, continúa con el paquete de cabecera de 54 bits que contiene información de control, por último viene el paquete que contiene la información de 0 a 2745 bits.

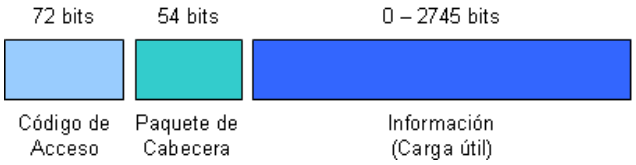


Figura 1.3

Paquetes de Intercambio

Todos los paquetes que se intercambian a través de un canal son evaluados por el receptor, que verifica el código de acceso del canal con las señales que recibe y si no es válido el resto del paquete es ignorado.

Piconet

Se ha mencionado anteriormente que la tecnología Bluetooth puede soportar comunicaciones punto a punto y punto a multipunto basado en el modo maestro – esclavo. Cuando un dispositivo Bluetooth se encuentra dentro del radio de cobertura de otro, estos pueden establecer una conexión entre sí compartiendo el mismo canal; si dos o más dispositivos comparten el mismo canal, forman una red denominada piconet. Cada piconet tiene una secuencia de salto de frecuencia diferente. Para poder regular el tráfico del canal en la red, uno de los dispositivos opera como maestro; usualmente el dispositivo que establece la piconet asume este rol, dejando a los demás dispositivos como esclavos. En una piconet un maestro puede establecer una conexión hasta con un máximo de siete esclavos en modo activo a los que se les asigna un número de 3 bits que recibe el nombre de dirección AMA (Active Member Address), a su vez el mismo maestro puede tener hasta 255 esclavos aparcados (parked) asignando un número de dirección de 8 bits con el nombre de PMA (Parked Member Address). Los esclavos aparcados no intervienen en el tráfico de la piconet, pero permanecen sincronizados con el maestro en todo momento, de manera que en cualquier instante pueden entrar en funcionamiento a la piconet.

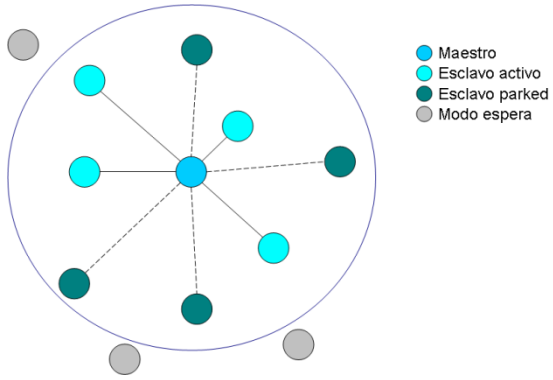


Figura 1.4

Esquema de una Piconet

Scatternet

Todos los dispositivos que participan en una piconet utilizan parte de la capacidad del canal es decir comparten el mismo canal que tiene un ancho de banda de 1 Mbit, mientras más dispositivos se incorporen a la piconet se reduce más la capacidad del canal hasta 10 Kbit/s. Para poder enfrentar esta dificultad surge un tipo de red denominada scatternet que permite conectar hasta 10 piconet. Un dispositivo que se encuentra en el área de cobertura de dos piconet sirve de puente para unir ambas piconet, formando una scatternet que permite tener una red más amplia y mayor rendimiento. En una piconet y scatternet en cualquier momento un maestro y esclavo pueden cambiar de rol sin afectar el funcionamiento con los demás participantes.

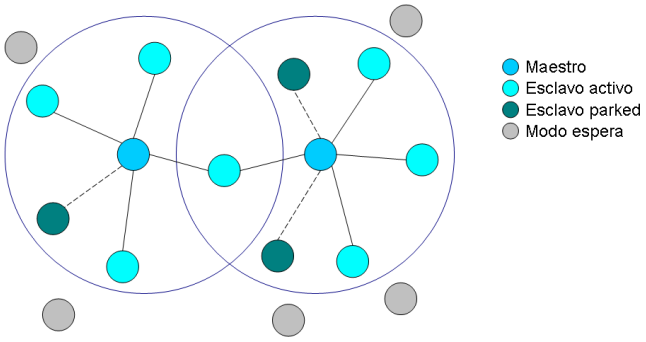


Figura 1.5

Esquema de una Scatternet

Establecimiento de las Conexiones

Antes de establecer alguna conexión los dispositivos Bluetooth se encuentran en modo de espera (standby mode) escuchando a la red. Para poder establecer conexiones se utilizan procedimientos de acceso de búsqueda y pregunta (paging/inquiry).

"El procedimiento de "inquiry" permite a un dispositivo descubrir qué dispositivos están en su zona de cobertura, determinando sus direcciones y el reloj de todos aquellos que respondan al mensaje de búsqueda."[2]

"El procedimiento de "paging" sigue al de "inquiry". El procedimiento de paging pregunta por la dirección de un dispositivo Bluetooth con el que queremos establecer la conexión."[3]

Un dispositivo en modo de espera (standby mode) tratará de encontrar mensajes paging/inquiry cada 1.28 segundos en este tiempo escucha alguna de las 32 frecuencias de salto definidas para ese dispositivo. Al detectarse un dispositivo con otro cualquiera de estos puede iniciar el procedimiento de conexión, convirtiéndose en el dispositivo maestro de la piconet. En el momento que se establece la conexión el dispositivo maestro ó esclavo puede operar en los siguientes modos:

Active mode: El dispositivo se encuentra activo en el canal de comunicación.

Sniff mode: El dispositivo esclavo reduce su tiempo de actividad de escucha, por lo que el dispositivo maestro transmite en slots de tiempos determinados.

Hold mode: El dispositivo esclavo es puesto en espera, mientras tanto puede buscar otros dispositivos o atender alguna otra piconet en la que participe.

Park mode: El dispositivo esclavo deja de participar en la piconet, pero sigue sincronizado con el canal del maestro para retomar de nuevo la comunicación.

[2]AYALA, Danny et. al. "Tecnología Bluetooth" p.13.
[3]Ibidem, p.14

Protocolos de Bluetooth

"Uno de los principales objetivos de la tecnología Bluetooth es conseguir que aplicaciones de dispositivos diferentes mantengan un dialogo fluido. Para conseguirlo, ambos, deben ejecutarse sobre la misma pila de protocolos.

Para conocer está tecnología comenzaremos describiendo la arquitectura de protocolos que podemos observar en la siguiente figura:

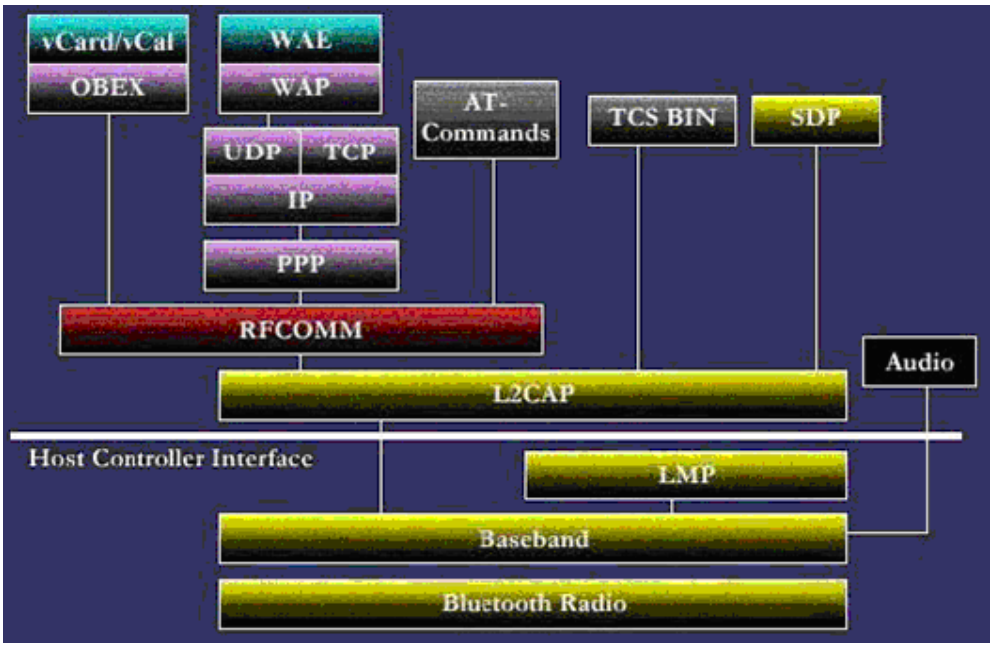


Figura 1.6

Protocolos Bluetooth

La pila esta constituida por dos clases de protocolos. Una primera clase llamada de protocolos específicos que implementa los protocolos propios de Bluetooth. Y una segunda clase formada por el conjunto de protocolos adoptados de otras especificaciones. Esta división en clases en el diseño de la pila de protocolos de Bluetooth permite aprovechar un conjunto muy amplio de ventajas de ambas. Por un lado, al implementar protocolos específicos de Bluetooth permite utilizar los beneficios que aporta la adopción de la tecnología Bluetooth. Por otro lado la utilización de protocolos no específicos ofrece la ventaja de la interacción de esta tecnología con protocolos comerciales ya existentes. Así como la posibilidad de que Bluetooth este abierto a implementaciones libres o nuevos

protocolos de aplicación de uso común. La pila de protocolos se puede dividir en cuatro capas lógicas:

- Núcleo de Bluetooth : Radio, Banda Base, LMP, L2CAP, SDP
- Sustitución de cable: RFCOMM
- Protocolos adoptados: PPP, UDP, TCP, IP, OBEX, WAP, IRMC, WAE
- Control de telefonía: TCS-binary, AT-Commands

El llamado de núcleo de Bluetooth, ha sido implementado en su totalidad por el SIG, no obstante otros como RFCOMM y TCS-binary pese a ser desarrollados por el propio SIG, los han desarrollado siguiendo las recomendaciones de otras instituciones de telecomunicaciones."[4]

Perfiles Bluetooth

Un perfil Bluetooth define el tipo de servicio que ofrece un dispositivo, que va ligado con el tipo de aplicaciones en los que va a ser utilizado. Es así que un perfil determinado permite que el dispositivo no tenga que implementar toda la pila de protocolos, simplemente utilizará los protocolos necesarios para realizar su función. Los principales perfiles Bluetooth utilizados son:

- Perfil de distribución de audio avanzado (A2DP)
- Perfil de control remoto de audio y vídeo (AVRCP)
- Perfil básico de imagen (BIP)
- Perfil básico de impresión (BPP)
- Perfil de telefonía inalámbrica (CTP)
- Perfil de red de marcado (DUNP)
- Perfil de fax (FAX)
- Perfil de transferencia de archivos (FTP)
- Perfil de acceso genérico (GAP)
- Perfil genérico de intercambio de objetos (GOEP)
- Perfil de sustitución de cable de copia impresa (HCRP)

[4]Ibidem, pp. 6 - 7

- Perfil manos libres (HFP)
- Perfil de dispositivo de interfaz humana (HID)
- Perfil de auricular (HSP)
- Perfil de intercomunicador (IP)
- Perfil de acceso LAN (LAP)
- Perfil de objeto push (OPP)
- Perfil de redes de área personal (PAN)
- Perfil de acceso SIM (SAP)
- Perfil de aplicación de descubrimiento de servicio (SDAP)
- Perfil de sincronización (SP): se utiliza para sincronizar el dispositivo con un administrador de información personal (abreviado *PIM*).
- Perfil de puerto de serie (SPP)

Como podemos observar la tecnología Bluetooth es de gran utilidad, ha logrado incorporarse de manera satisfactoria en gran cantidad de marcas y modelos de teléfonos celulares, y junto con otras incorporaciones tecnológicas han provocado que se puedan crear aplicaciones más sofisticadas y de gran variedad con el objetivo principal de satisfacer distintas demandas en el mercado comercial.

Pero ¿Cómo se logra crear aplicaciones más sofisticadas? En la siguiente parte del capítulo se hablará de forma general sobre el desarrollo de aplicaciones móviles y las principales tecnologías para llevar a cabo esta labor.

PARTE 3
DESARROLLO DE APLICACIONES MÓVILES

Introducción

En la actualidad de igual forma que en una computadora para poder desarrollar aplicaciones en dispositivos móviles se utilizan lenguajes de programación de alto nivel, los cuales permiten escribir programas entendibles tanto para el hombre como para la máquina. Aunque los dispositivos móviles han tenido una gran evolución aún encuentran ciertas limitaciones en sus recursos de hardware ya que tienen pantallas más pequeñas, menor cantidad de memoria disponible, menores velocidades de procesamiento entre otras restricciones.

El desarrollo de aplicaciones móviles no es nada sencillo, ya que cada dispositivo móvil cuenta con una amplia gama de tecnologías de operación y programación que va de acuerdo con cada uno de los fabricantes. En un principio el desarrollo de aplicaciones era aún más complicado se realizaba en lenguaje máquina o ensamblador lo que generaba que se realizara un mayor esfuerzo y tiempo en su desarrollo. Conforme los dispositivos móviles fueron evolucionando los lenguajes de programación de alto nivel se fueron incorporando para el desarrollo de aplicaciones móviles. Uno de estos lenguajes de alto nivel fue Java el cual tiene la principal característica de escribir una vez el programa y ejecutarlo en cualquier plataforma a través de su máquina virtual, esta idea se transportó a su versión Java 2 Micro Edition (J2ME) que permite desarrollar aplicaciones para una amplia gama de dispositivos móviles.

Surgimiento de Java

Los inicios del lenguaje de programación Java se remontan al año de 1990, cuando James Gosling empleado de Sun Microsystems, se le encargó la labor de crear programas capaces de controlar aparatos domésticos. En un principio James Gosling junto con su equipo de trabajo comenzó su labor utilizando el lenguaje de programación C++, sin embargo este lenguaje no satisfacía totalmente las necesidades para realizar sus proyectos. De esta forma James Gosling tomó la decisión de escribir un lenguaje de programación simplificado que le permitiera evitar los problemas que le ocasionaba el trabajar con C++; para el nuevo

lenguaje de programación incorporó la sintaxis básica y el paradigma orientado a objetos del lenguaje C++. Al terminar el nuevo lenguaje lo llamaron Oak, el cual fue utilizado por primera vez en el Proyecto Green donde se intentó desarrollar un sistema de control de distintos tipos de dispositivos dentro del hogar, esto a través de una computadora de mano denominada StarSeven. El siguiente proyecto donde se utilizó el lenguaje Oak fue Video Demanda (VOD) el cual controlaría un sistema de televisión interactivo. Ambos proyectos no concluyeron en productos actuales pero aún así permitieron desarrollar el lenguaje Oak que con el paso del tiempo cambiaría su nombre por el de Java. Hasta esta etapa Java era un lenguaje poderoso, sencillo e independiente de plataforma. En 1993 con el gran auge de Internet el equipo Java descubrió que el lenguaje podía ser empleado para la programación Web por lo que surgen los applets y el soporte Java para los navegadores logrando popularizar más el lenguaje entre los programadores. Finalmente Sun Microsystems anuncia de manera oficial el lenguaje Java en 1995.

Funcionamiento de Java

El funcionamiento del lenguaje de programación Java esta basado en su máquina virtual que es el entorno de ejecución para las aplicaciones creadas bajo Java, lo que le permite ser un lenguaje multiplataforma teniendo alcance en cualquier plataforma que implemente la máquina virtual.

Java es tanto compilado como interpretado. El compilador Java se encarga de convertir el código fuente (archivos con extensión .java) a bytecodes (archivos con extensión .class) que son un conjunto de instrucciones parecidas al lenguaje máquina pero independientes de la plataforma los cuales son interpretados por la máquina virtual para ejecutar la aplicación.

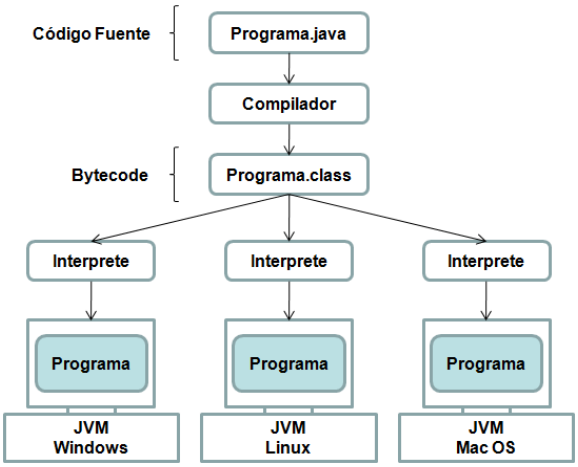


Figura 1.7

Esquema del funcionamiento de Java

Arquitectura de Java

El lenguaje Java proporciona un conjunto de APIs (Application Program Interface – Interfaz de Programación de Aplicaciones) con el objetivo de aportar a los programadores los recursos necesarios para el desarrollo de aplicaciones. Las APIs son conjunto de paquetes y clases con especificación de uso para efectuar las operaciones necesarias en una aplicación.

Sun Microsystems establece tres ediciones diferentes del lenguaje de programación: J2EE (Java 2 Enterprise Edition), J2SE (Java 2 Standard Edition) y J2ME (Java 2 Micro Edition). Cada una de las APIs empleadas en las tres ediciones están relacionadas entre sí, siendo ampliaciones o subconjuntos unas de otras.

J2EE (Java 2 Enterprise Edition): Esta edición agrupa un conjunto de APIs que son una ampliación de J2SE, esta orientada al entorno empresarial y se usa principalmente para el desarrollo de aplicaciones distribuidas.

J2SE (Java 2 Standard Edition): Esta edición agrupa un conjunto de APIs básica para el desarrollo de aplicaciones "stand alone". A partir de esta edición se da paso a las ediciones J2EE y J2ME.

J2ME (Java 2 Micro Edition): El conjunto de APIs que agrupa esta edición va orientada al desarrollo de aplicaciones para dispositivos con capacidades limitadas y restringidas, parte del API es un subconjunto de J2SE.

La arquitectura de la plataforma Java 2 se representa en la siguiente imagen, que también muestra la relación entre las tres ediciones.

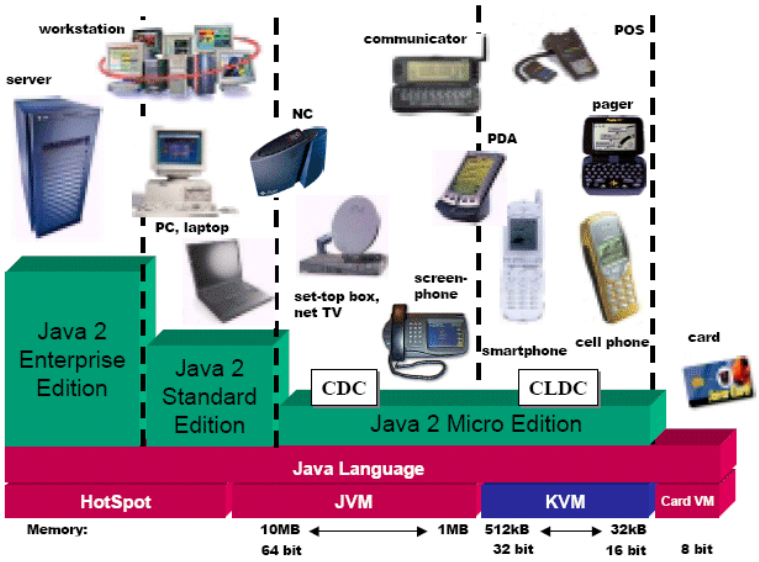


Figura 1.8

Arquitectura de la Plataforma Java

Tecnologías para el Desarrollo de Aplicaciones Móviles

En la sección anterior se mencionó que para el desarrollo de aplicaciones móviles se han podido incorporar lenguajes de programación de alto nivel como Java, se comentó la forma en que surgió, se explicó como funciona, y se describió de forma general la plataforma Java con sus tres ediciones para el desarrollo de diferentes tipos de aplicaciones, siendo J2ME (Java 2 Micro Edition) la tecnología destinada al desarrollo de aplicaciones móviles la cual hay que mencionar que no es la única tecnología disponible para realizar esta tarea. Antes de continuar hablando sobre la tecnología J2ME la cual es el principal objeto de estudio de este trabajo, hay que revisar algunas de las otras tecnologías que podemos encontrar para el desarrollo de aplicaciones móviles en las que destacan principalmente .Net Compact Framework, Flash Lite, y los Sistemas Operativos Symbian OS y Android.

.NET COMPACT FRAMEWORK: La tecnología .NET Compact Framework es sostenida por Microsoft, para desarrollar aplicaciones con esta tecnología es necesario utilizar el entorno de desarrollo visual studio.net mismo que no es distribuido de forma gratuita, por lo que hay que pagar por una licencia para su uso. Aplicaciones realizadas con esta tecnología son exclusivas para los dispositivos que implementan el sistema operativo Windows CE y Windows Mobile 2003 en su mayoría PDA como Pocket PC.

FLASH LITE: La tecnología Flash Lite es sostenida por Adobe, cada vez crece más el uso de esta tecnología en el mundo móvil cada vez son más los nuevos modelos de teléfonos celulares que incorporan soporte a Flash Lite. El realizar aplicaciones en esta tecnología es mucho más sencillo va enfocada principalmente al desarrollo de animaciones y juegos. Algunos de los inconvenientes de esta tecnología actualmente es el no tener soporte para acceso a hardware del dispositivo como Bluetooth o cámara, también no se incluye almacenamiento persistente de datos para las aplicaciones. De la misma manera que .NET Compact Framework el entorno de desarrollo para aplicaciones Flash Lite no es distribuido de forma gratuita.

SYMBIAN OS: A diferencia de las anteriores tecnologías Symbian OS es un Sistema Operativo desarrollado para dispositivos móviles. Pese a que en un principio fue creado por varias empresas de telefonía móvil actualmente la empresa Nokia es propietaria de Symbian OS. Para el desarrollo de aplicaciones basadas en Symbian OS se usa principalmente el lenguaje de programación C++, del cual se ofrece un conjunto de API's nativas del sistema permitiendo mayor rapidez en la ejecución de las aplicaciones siendo estas de gran calidad. Sin embargo el API es complicado, se necesitan programadores expertos, y el tiempo en el desarrollo de la aplicación es aún mayor que en otras tecnologías. Otra desventaja que se tiene al desarrollar aplicaciones basadas en Symbian es la poca portabilidad para ejecutar las aplicaciones en distintos dispositivo, no todos implementan Symbian OS y existen incompatibilidades de código entre diferentes series de Symbian. Cabe mencionar que en las más recientes versiones se da soporte para el desarrollo de aplicaciones a través de Python (PyS60) y Java (J2ME).

ANDROID: Al igual que Symbian OS Android es un Sistema Operativo basado en Linux y desarrollado por la "Open Handset Alliance" liderado por Google. Android incorpora su propia máquina virtual llamada "Delvik" que es el entorno de ejecución para las

aplicaciones que son desarrolladas a través de gran cantidad de librerías del lenguaje Java, y a diferencia de J2ME no esta sujeto a restricciones como los son las configuraciones y perfiles lo que indica una completa portabilidad para toda la gama de dispositivos que implementen el Sistema Operativo Android. Sin embargo esta tecnología apenas inicia apareciendo en septiembre del 2008 incorporándose hasta el momento en dos modelos de teléfono celular y ofreciendo como único entorno de desarrollo Eclipse que es distribuido de forma gratuita.

J2ME: La tecnología J2ME está sostenida por Sun Microsystems, apareció a finales de los noventa, lleva ya tiempo en el mercado del desarrollo de software lo que representa robustez y estabilidad al ser en este momento la tecnología más extendida, con mayor soporte y uso en una amplia gama de dispositivos móviles.

Con esto se finaliza el capítulo uno cumpliendo el objetivo de cubrir los antecedentes teóricos necesarios para abordar de manera adecuada el estudio que será realizado en capítulos posteriores, de los cuales en el siguiente se va a describir más detalladamente el funcionamiento y programación de aplicaciones utilizando la tecnología J2ME.

CAPÍTULO 2: LA TECNOLOGÍA JAVA 2 MICRO EDITION

PARTE 1

ARQUITECTURA DE LA TECNOLOGÍA JAVA 2 MICRO EDITION

PARTE 2

FUNCIONAMIENTO DE LA TECNOLOGÍA JAVA 2 MICRO EDITON

PARTE 3

PROGRAMACIÓN DE APLICACIONES CON JAVA 2 MICRO EDITION

PARTE 1
ARQUITECTURA DE LA TECNOLOGÍA JAVA 2 MICRO EDITION

Introducción

En un principio el lenguaje de programación Java (Oak) fue desarrollado con el fin de poder ser utilizado para crear aplicaciones que controlaran electrodomésticos como televisiones, refrigeradores, hornos de microondas entre otros. Por lo que la tecnología J2ME de nuevo regresa a la programación de dispositivos con capacidades reducidas y restringidas, brindando un mayor soporte a una amplia cantidad de dispositivos.

Entre los dispositivos móviles no se tienen las mismas arquitecturas hardware, estas van a variar dependiendo de las funciones para las que son creados y por las especificaciones de sus fabricantes. Por ello de la misma manera que en la edición estándar de Java (J2SE), J2ME brinda cierta portabilidad a las aplicaciones, las cuales podrán ser ejecutadas en los dispositivos a través de la máquina virtual que incorporen. Sin embargo dado a las limitaciones de hardware de los dispositivos móviles, no pueden implementar la misma máquina virtual que la edición J2SE, por lo que Sun Microsystems define varias máquinas virtuales para J2ME la KVM (Kilobyte Virtual Machine) y la CVM (Compact Virtual Machine), ambas suprimen ciertas características para poder ser implementadas.

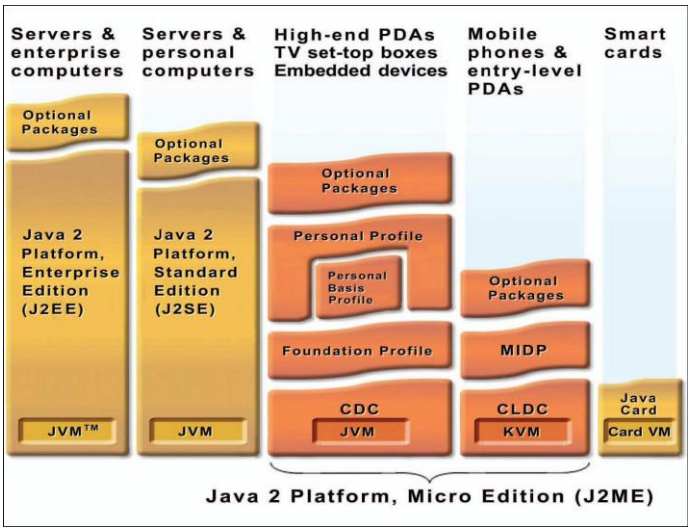


Figura 2.1

Arquitectura J2ME[5]

[5] http://java.sun.com/javame/technology/index.jsp/. 23 - enero - 2009

La imagen anterior muestra la arquitectura de J2ME, podemos distinguir la manera en que están definidas las máquinas virtuales (JVM) para cada edición de Java. La estructura de la arquitectura J2ME consta de cuatro partes principales:

- Paquetes Opcionales
- Perfiles
- Configuraciones
- Máquina Virtual

Cabe señalar que la máquina virtual para J2ME esta soportada por el Sistema Operativo del dispositivo móvil, por lo que cada fabricante es responsable de la implementación adecuada de la máquina virtual basándose en las especificaciones de Sun Microsystems para incorporar la KVM ó CVM en el Sistema Operativo.

Máquina Virtual KVM y CVM

La **KVM** es la implementación de la JVM de forma compacta y reducida, permite mantener las características principales que ofrece el lenguaje Java en dispositivos de recursos limitados, solo abarca unos pocos Kilobytes (de 40Kb a 80Kb) de memoria al ejecutarse. Sin embargo, el reducir y compactar la KVM hace que posea las siguientes limitaciones:

- No brinda soporte para punto flotante, por lo que los tipos de datos como double y float no existen. Esto se debe también a que parte de los dispositivos que implementan la KVM no disponen de los recursos necesarios para realizar este tipo de operaciones.
- No existe la finalización de instancias de clase por medio del método Object.finalize().
- La capacidad en el manejo de excepciones es más limitada.
- Únicamente pueden ser utilizados los cargadores de clases predefinidos en la KVM, por lo que no existen los cargadores de clases definidos por el usuario.
- No brinda soporte para los grupos de hilos (daemons).
- No implementa Interfaz Nativa Java (JNI).
- Para ninguna aplicación existen las referencias débiles.

- No brinda soporte para el API de reflexión (reflection), la cual permite obtener información de otros objetos en tiempo de ejecución.

Otra de las características que se ha modificado de la KVM es el verificador de clases que incluye JVM, que es el encargado de rechazar los archivos de las clases que no sean válidos. Para la KVM el algoritmo verificador de clases consta de dos partes un preverificador y un verificador de clases, el preverificador está ubicado fuera del dispositivo es decir el equipo (PC de escritorio) encargado de hacer la compilación de clases también es responsable de realizar la verificación previa de clases, mientras que el verificador se incorpora internamente en la KVM. La razón de separar en dos partes el verificador de clases se debe a que el verificador estándar que usa la JVM es demasiado grande para incorporarse en la KVM.

La **CVM** es la otra máquina virtual definida para J2ME, es de mayor tamaño que la KVM y va enfocada a dispositivos con procesadores de 32 bits y memoria disponible de hasta 2Mb. Es muy similar a la JVM, a excepción que cuenta con limitaciones gráficas IU (Interfaz de Usuario) y capacidad de memoria.

Para finalizar la descripción del nivel base de la arquitectura J2ME que son las máquinas virtuales KVM y CVM solo resta mencionar que cada una está diseñada para trabajar con diferentes configuraciones, KVM bajo la configuración CLDC (Connected Limited Device Configuration) y CVM bajo la configuración CDC (Connected Device Configuration).

Configuraciones

Las configuraciones son un conjunto básico de clases (APIs) disponibles para el desarrollo de aplicaciones para una categoría determinada de dispositivos que comparten características básicas y comunes en procesamiento y memoria. Una configuración va asociada a una máquina virtual (KVM - CVM), describe y detalla características soportadas por el lenguaje Java, características de la máquina virtual y las APIs soportadas. Actualmente para J2ME existen dos configuraciones definidas, la CLDC (Connected Limited Device Configuration) y la CDC (Connected Device Configuration) que contienen un subconjunto del API de J2SE añadiendo el paquete adicional javax.microedition.*

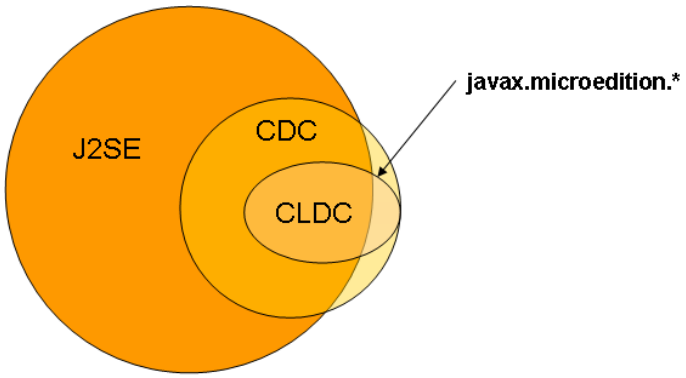


Figura 2.2

Comparativa entre las API de J2SE y Configuraciones J2ME

La imagen anterior muestra el paquete adicional javax.microedition.*, que contiene las clases nuevas incorporadas en J2ME, dichas clases serán estudiadas con mayor detalle en el momento de comenzar a programar aplicaciones (tercer parte del presente Capítulo).

La configuración **CLDC** (Connected Limited Device Configuration) utiliza la KVM (Kilobyte Virtual Machine) y va orientada a dispositivos con limitaciones en la capacidad de procesamiento, memoria y gráficos como lo son los teléfonos celulares y PDA. Para poder incorporar esta configuración los dispositivos deben ofrecer los siguientes requerimientos:

- Procesador de 16 – 32 bits.
- Memoria disponible de 160Kb a 512Kb para el entorno Java.
- Conexión a red inalámbrica.
- Brindar un bajo consumo de energía.

La configuración **CDC** (Connected Device Configuration) a diferencia de la CDLC (Connected Limited Device Configuration) utiliza la CVM (Compact Virtual Machine) y va orientada a dispositivos más avanzados que brindan mayor capacidad de procesamiento y de memoria, tales como sistemas de navegación en automóviles, decodificadores digitales de TV, reproductores Blue-ray, entre otros. Los dispositivos que incorporan la configuración CDC deben presentar los siguientes requerimientos:

- Procesador de 32 bits.
- Memoria disponible de 2Mb para el entorno Java.

- Conexión a red fija.
- Incorporar la funcionalidad completa de la JVM.

Las configuraciones son el segundo nivel en la arquitectura J2ME, se organizaron de esta forma para poder acaparar un rango mayor de dispositivos. Cada una de las configuraciones es la base para diferentes perfiles.

Perfiles

Los perfiles son un conjunto de APIs complementarias a las configuraciones para conformar un entorno completo para el desarrollo de aplicaciones. En una configuración se establecen un conjunto de APIs que definen las características de una categoría de dispositivos, mientras que en un perfil se establece un conjunto de APIs que definen las características de un dispositivo especifico. Son encargados de controlar el ciclo de vida de una aplicación, la interfaz de usuario, los mecanismos de seguridad y la persistencia de datos. También son construidos sobre las configuraciones, por lo que cada configuración dispone de sus perfiles complementarios.

Para la configuración CLDC se definen los perfiles:

Mobile Information Device Profile (MIDP).
PDA Profile.

Para la configuración CDC se definen los perfiles:

- Foundation Profile (FP).
- Personal Profile (PP).
- Personal Basis Profile (PBP).

Los perfiles componen el tercer nivel en la arquitectura J2ME, permiten la portabilidad entre diferentes dispositivos para las aplicaciones desarrolladas en J2ME y pueden extenderse a través de los paquetes opcionales.

Paquetes Opcionales

Los paquetes opcionales brindan una mayor funcionalidad específica a los perfiles aprovechando más ampliamente funciones adicionales en los dispositivos, aunque los paquetes no necesariamente pueden estar relacionados con algún perfil o configuración, por lo que son conjunto de APIs que no son necesariamente obligatorias de incorporar en los dispositivos que implementen alguna de las dos máquinas virtuales.

Para aprobar nuevos paquetes opcionales se recurre al JCP (Java Community Process) que es el proceso para elaborar y revisar las especificaciones (JSR) de la tecnología Java para impulsar la evolución de la plataforma. JSR es el documento presentado para proponer las nuevas condiciones y especificaciones, así como revisiones importantes que se deban realizar a estas mismas, actualmente existen más de 90 especificaciones de la tecnología Java.

En estas especificaciones (JSR) se definen algunos de los paquetes opcionales para J2ME, entre las cuales destacan:

- JSR 82 – Java APIs for Bluetooth.
- JSR 135 – Mobile Media API.
- JSR 66 – RMI API.
- JSR 211 – Content Handler API.
- JSR 239 – Java Binding for the OpenGL ES.
- JSR 172 – J2ME Web Services Specification.
- JSR 177 – Security and Trust Services APIs.
- JSR 219 – Security.
- JSR 209 – Advanced Graphics and User Interface.
- JSR 169 – JDBC.
- JSR 927 – Java TV API.

Aquí se termina por describir todos los niveles que componen la arquitectura J2ME, esta información va a permitir seleccionar y trabajar con el entorno adecuado para desarrollar aplicaciones en dispositivos móviles. El estudio principal de este trabajo es la tecnología J2ME para poder desarrollar aplicaciones que permitan interactuar una PC con un

dispositivo móvil vía Bluetooth, y siendo que gran parte de los dispositivos móviles que incorporan la tecnología Bluetooth y J2ME son los teléfonos celulares a partir de ahora se enfocará el estudio en el desarrollo de aplicaciones para este dispositivo, por lo que se estudiará y trabajará con el entorno que muestra la imagen:

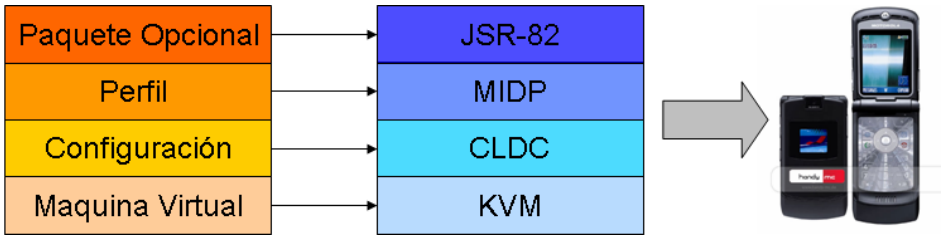


Figura 2.3

Entorno J2ME que se estudiará para desarrollar la aplicación de interacción vía Bluetooth entre una PC y un Teléfono Celular.

Para concluir esta primera parte del capítulo, vale mencionar que en la arquitectura que se acaba de revisar la máquina virtual KVM y CVM fueron especificadas y desarrolladas por Sun Microsystems, sin embargo los fabricantes de dispositivos móviles que brinden soporte para la tecnología J2ME pueden implementar sus propias máquinas virtuales siempre y cuando cumplan los requerimientos de la especificación (configuración y perfil) que sea incorporada en el dispositivo móvil.

En la siguiente parte del capítulo se estudiará el funcionamiento de las aplicaciones J2ME a través de la configuración CLDC y perfil MIDP actualmente los de mayor incorporación en los teléfonos celulares.

PARTE 2
FUNCIONAMIENTO DE LA TECNOLOGÍA JAVA 2 MICRO EDITION

Introducción

Ahora que se han descrito los componentes que conforman la arquitectura J2ME, se puede explicar con mayor claridad como funcionan las aplicaciones al ejecutarse en un dispositivo móvil en este caso el teléfono celular. Los teléfonos celulares son los dispositivos en los que más se ha expandido la tecnología J2ME, que bajo la configuración CLDC y perfil MIDP se ofrece un modo seguro de ejecución siguiendo un modelo de seguridad sandbox parecido al que se incorpora en los Applets (pequeñas aplicaciones Java que se ejecutan sobre un navegador web) de la edición J2SE. Con este modelo las aplicaciones solo pueden ejecutarse bajo el confinamiento de la máquina virtual establecida, por lo que ninguna aplicación desarrollada bajo la tecnología J2ME puede dañar el dispositivo ni alguna otra aplicación ajena a J2ME que se encuentre instalada.

MIDlets

Se denomina MIDlets a las aplicaciones desarrolladas en J2ME utilizando el perfil MIDP sobre la configuración CLDC. Cada dispositivo debe implementar un gestor de aplicaciones denominado AMS (Application Management System) que será responsable de gestionar el ciclo de vida del MIDlet y de controlar los estados en los que se encuentre mientras esta en ejecución.

El AMS (Application Management System) controla el ciclo de vida de un MIDlet que consta de las siguientes fases:

- Ubicar: es la fase previa a la instalación, se ubica y obtiene el MIDlet de alguna fuente, por lo que el AMS debe brindar los mecanismos necesarios para seleccionar y descargar el MIDlet deseado.
- Instalar: en esta fase se da el proceso de instalación del MIDlet descargado, el AMS debe controlar todo el proceso de la instalación notificando sobre el progreso, problemas, y termino de la instalación.

- Ejecutar: en esta fase se ejecuta el MIDlet, siendo el AMS responsable de brindar el mecanismo necesario para ejecutar el MIDlet y obtener los recursos necesarios, además tiene la función de controlar los estados del MIDlet.
- Actualizar: el AMS se encarga de manejar la lista de MIDlets que se encuentran instalados en el dispositivo, por lo que después de realizar una descarga de un nuevo MIDlet debe detectar y notificar si el MIDlet es una actualización para poder ser reemplazado.
- Borrar: en esta fase el AMS se encarga de borrar todos los registros del MIDlet seleccionado, por lo que debe de brindar un mecanismo necesario para borrar y notificar al usuario.

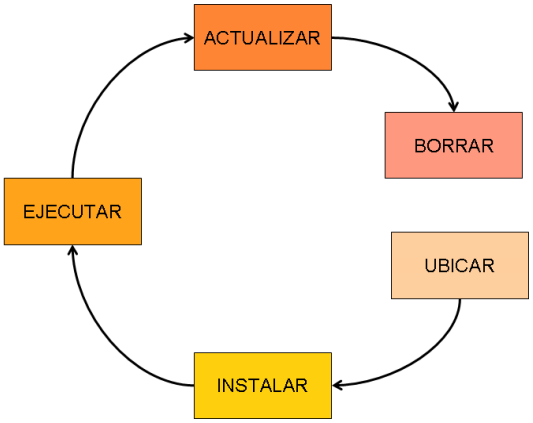


Figura 2.4

Ciclo de vida de un MIDlet a través de un dispositivo

Los estados que pueden presentarse en un MIDlet surgen mientras es ejecutado, estos son tres: activado, pausado y destruido, los cuales también son controlados por el AMS.

- Activado: en este estado el MIDlet se encuentra ejecutándose, haciendo uso de recursos del dispositivo.
- Pausado: en este estado el MIDlet no está en ejecución, está detenido temporalmente, por lo que no puede hacer uso de ningún recurso compartido.
- Destruido: en este estado el MIDlet no se encuentra en ejecución, libera todos los recursos y finaliza su actividad sobre el dispositivo.

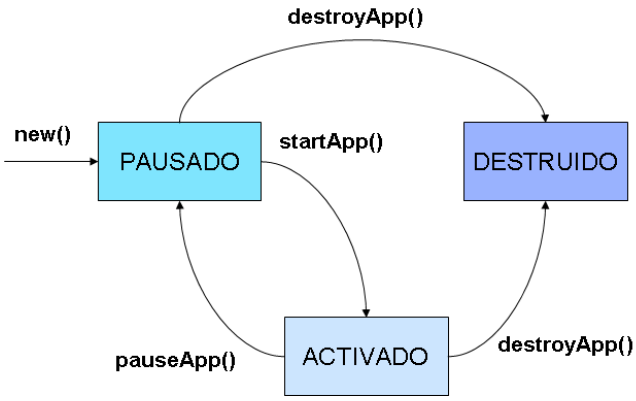


Figura 2.5

Estados de un MIDlet

La imagen muestra los estados en los que se puede encontrar un MIDlet mientras es ejecutado. Todo MIDlet debe extender la clase abstracta MIDlet y a través de la invocación a los métodos abstractos startApp() y pauseApp() se puede transitar entre los estados de Activado y Pausado, con el método abstracto destroyApp() se transita al estado Destruido este método puede ser llamado de cualquiera de los otros dos estados restantes, una vez realizado esto no se podrá transitar por ningún otro estado más y el MIDlet terminará su ejecución.

La manera de distribuir los MIDlets (aplicaciones) es a través de los archivos .JAR (Java ARchive) y .JAD (Java application descriptor). En un archivo .JAR se empaqueta un MIDlet o Suite MIDlet (varios MIDlets que comparten el mismo entorno de ejecución) junto con las clases y recursos necesarios (imágenes, sonidos, archivos adicionales) que componen la aplicación. Opcionalmente se puede distribuir el archivo .JAD que es un archivo de texto plano que proporciona la descripción del contenido del archivo .JAR, cada línea del texto brinda una propiedad (nombre:valor) a la cual se pude acceder desde la misma aplicación por medio de la clase Midlet con el método getAppProperty(name).

Para poder instalar el MIDlet (aplicación) es necesario de alguna forma poder transmitir al dispositivo móvil los archivos .JAR y .JAD, que serán ubicados por el AMS (Application Management System) que iniciará las fases del ciclo de vida del MIDlet descritas anteriormente.

Existen dos formas en las que el AMS (Application Management System) puede ubicar los MIDlets. Una es de manera local conectando el dispositivo móvil de forma directa con la

PC a través de cable serial USB, Bluetooth o Infrarrojos. Otra es de manera remota con el navegador WAP del dispositivo móvil conectarse por medio del protocolo HTTP/HTTPS con el servidor donde son publicadas las aplicaciones, comúnmente a este tipo de distribución se le conoce como OTA (Over-The-Air provisioning).

Configuración CLDC 1.0

"Es la especificación que define la plataforma estándar de J2ME para dispositivos móviles pequeños, con capacidad de conexión y recursos limitados."[6] Está definida bajo la JSR-30 y es la primera configuración incorporada en J2ME, actualmente se encuentra disponible la versión CLDC 1.1 que va enfocada a dispositivos de la nueva generación, la principal característica de esta versión más reciente es el soporte para operaciones de punto flotante y referencias débiles, cabe señalar que es compatible con versiones anteriores (CLDC 1.0). La configuración CLDC 1.0 es soportada por gran cantidad de modelos de teléfonos celulares, en la tabla 2.1[7] se muestran los paquetes incluidos.

PAQUETES	*DESCRIPCION*
java.io	Clases e Interfaces estándar de E/S de datos CLDC.
java.lang	Clases e Interfaces fundamentales de Java CLDC.
java.util	Clases Collection y manejo de fecha y hora CLDC.
javax.microedition.io	Clases e Interfaces de conexión genérica CLDC.

Tabla 2.1

Paquetes disponibles para la configuración CLDC 1.0

Perfil MIDP 2.0

"Extiende la JSR-37 mejorando el modelo de seguridad incorporando soporte para juegos 2D y sonido, mejorando también los controles de posicionamiento de elementos en la pantalla y la interfaz de usuario de estos elementos"[8]. La especificación JSR-37 hace referencia al perfil MIDP 1.0, el cual fue el primer perfil definido en la configuración CLDC, de igual forma que la configuración CLDC 1.0 el perfil MIDP 2.0 actualmente es

[6]FROUFE QUINTAS, Agustín et al. "J2ME Java 2 Micro Edition Manual de usuario y tutorial" p.37.
[7]http://java.sun.com/javame/reference/apis/jsr030/. 18-Febrero-2009.
[8]FROUFE QUINTAS, Agustín et al. "J2ME Java 2 Micro Edition Manual de usuario y tutorial" p.37.

el de mayor incorporación en los teléfonos celulares, en la tabla 2.2[9] se muestran los paquetes incluidos.

PAQUETES	*DESCRIPCION*
java.io	Clases e Interfaces estándar de E/S de datos MIDP
java.lang	Clases e Interfaces fundamentales de Java MIDP
java.util	Clases Collection y manejo de fecha y hora MIDP
javax.microedition.io	Clases e Interfaces de conexión genérica MIDP.
javax.microedition.Icdui	Clases e Interfaces para la interfaz de usuario MIDP.
javax.microedition.Icdui.game	Clases para desarrollo de juegos MIDP.
javax.microedition.media	Clases e Interfaces para audio MIDP.
javax.microedition.media.control	Interfaces para definir controles de audio MIDP.
javax.microedition.midlet	Clases para manejo de ciclo de vida MIDP.
javax.microedition.pki	Clases e Interfaces para seguridad MIDP.
javax.microedition.rms	Clases e Interfaces para la persistencia de datos MIDP.

Tabla 2.2

Paquetes disponibles para el perfil MIDP 2.0

Se puede observar que están definidos los paquetes utilizados en la configuración CLDC 1.0 ya que el perfil MIDP 2.0 complementa con más clases y paquetes brindando un entorno de desarrollo más amplio, por ejemplo en el paquete java.lang del perfil MIDP 2.0 se incluyen las clases Float y Double que solamente funcionarán sobre la configuración CLDC 1.1 que brinda soporte a operaciones de punto flotante.

En la siguiente parte del capítulo se van a describir y utilizar algunas de las principales clases e interfaces que componen los paquetes de la configuración CLDC 1.0 y perfil MIDP 2.0, esto va a permitir comprender de manera más amplia el funcionamiento de la tecnología J2ME, para posteriormente aplicar lo estudiado en el desarrollo del caso práctico a realizar.

[9]http://java.sun.com/javame/reference/apis/jsr118/. 18-Febrero-2009.

PARTE 3
PROGRAMACIÓN DE APLICACIONES CON JAVA 2 MICRO EDITION

Introducción

Para facilitar el trabajo a los programadores en el desarrollo de aplicaciones existen distintas herramientas de desarrollo denominadas IDE por sus siglas en ingles “Integrated Development Environment”, que son programas que brindan un entorno de programación para uno o varios lenguajes de programación, entre sus principales funciones destacan el contener un editor de código del lenguaje utilizado, compilador y depurador del mismo y en algunos casos una interfaz gráfica de ejecución. La tecnología J2ME no es la excepción y son varios los IDE disponibles entre los más utilizados están NetBeans Mobility Pack, Sun One Studio Mobile Edition, Java ME SDK 3.0 entre otros. En este caso para programar aplicaciones en J2ME se va a utilizar el IDE NetBeans 6.1 que incluye distintas herramientas complementarias para crear aplicaciones en J2ME y J2SE lo que va a ser de gran utilidad en capítulos posteriores, en especial para el desarrollo del caso práctico.

Niveles de abstracción en la programación para interfaz de usuario

Las aplicaciones creadas bajo CLDC y MIDP proporcionan diferentes elementos que conforman interfaces de usuario para interactividad y control de la aplicación para el usuario. Los niveles de abstracción en la Interfaz de Usuario se dividen en dos grupos de Alto nivel y Bajo nivel.

Una Interfaz de Usuario de Alto nivel de abstracción tiene componentes como botones, cajas de texto, formularios, listas entre otros. Cada dispositivo es encargado de implementar estos componentes, lo que deja el aspecto de la aplicación bajo control del dispositivo.

Una Interfaz de Usuario de Bajo nivel de abstracción permite tener el control sobre el aspecto de la aplicación, se puede establecer el manejo de eventos a través de diferentes teclas y establecer el contenido para mostrar en pantalla.

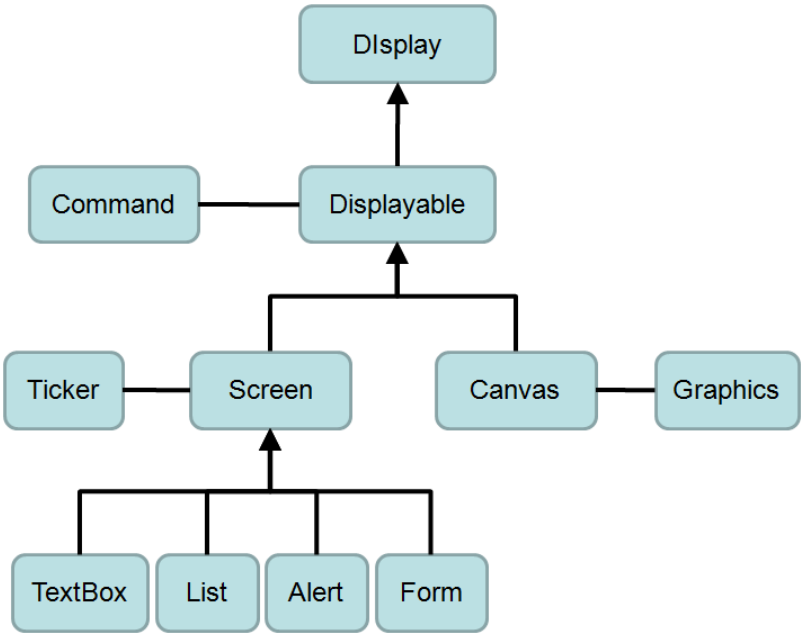


Figura 2.6

Jerarquía de Clases derivadas de la Clase Display

La figura anterior muestra la jerarquía de clases derivadas de la clase Display, que se encarga de manejar las operaciones de la pantalla. El objeto Display puede incluir varios objetos del tipo Displayable de los que derivan el objeto Screen y Canvas. Un objeto Screen pertenece a una interfaz de usuario de alto nivel de abstracción que contiene varios componentes (Objetos: TextBox, List, Alert, Form) que el dispositivo implementará de forma propia. Por lo que un objeto Canvas pertenece a una interfaz de usuario de bajo nivel de abstracción que va a controlar de forma específica los elementos que serán mostrados en pantalla.

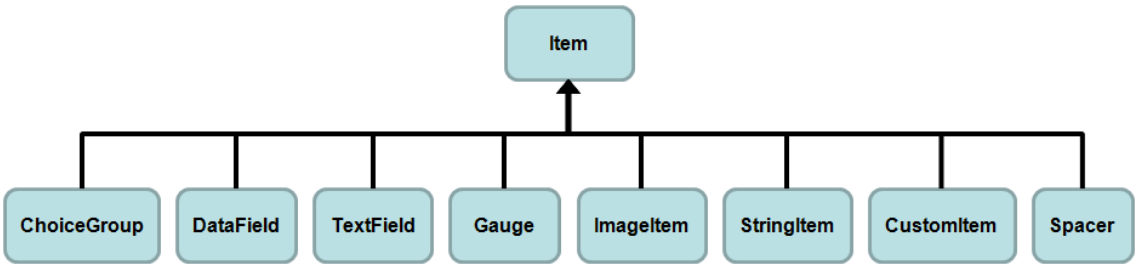


Figura 2.7

Clases derivadas de la Clase Item

La figura anterior muestra la jerarquía de clases derivadas de la clase Item, que es el tipo de objeto que puede ser incluido en un objeto Form, permitiendo integrar y mostrar distintos tipos de componentes.

Las clases que se utilizarán principalmente son las vistas en las dos figuras anteriores, que pertenecen al paquete javax.microedition.lcdui y junto con algunas de las clases definidas en otros paquetes se van a idear ejemplos para demostrar el funcionamiento y utilidad que nos pueden brindar para el desarrollo de nuestro caso práctico.

La siguiente figura muestra el emulador para J2ME incluido en el IDE NetBeans 6.1, que va a permitir ejecutar las aplicaciones y verificar su correcto funcionamiento.

Figura 2.8

Emulador para J2ME incluido en NetBeans6.1

Ejemplo 1: Programa "Hola Mundo J2ME" Comparativa al utilizar un API de bajo y alto nivel de abstracción

Se va a comenzar nuestro primer ejemplo con el ya clásico "Hola Mundo J2ME", aquí se va distinguir la principal diferencia en funcionamiento que se tiene al programar una interfaz de usuario de bajo y alto nivel de abstracción. Utilizando las clases MIDlet, Display, Command, Form, Canvas, Graphics, y la interfaz CommanListener se va a crear

un MIDlet que obedeciendo a un evento de comando mostrará el mensaje "Hola Mundo J2ME" a través de objetos desplegables Screen y Canvas.

De la misma forma que en otras ediciones de Java se define el paquete al que pertenece nuestra aplicación, seguido por la importación de los paquetes que se utilizarán.

La clase principal de nuestro MIDlet es **HolaMundo** y hereda de la clase abstracta MIDlet que contiene los métodos que manejarán los estados de nuestra aplicación.

Se declaran los objetos que se utilizarán para desplegar los mensajes a través de Screen y Canvas. Utilizando la clase Command se declara el objeto encargado de finalizar el MIDLET.

```
package ejemplo_1;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * @author JAVAXIAN
 */
public class HolaMundo extends MIDlet {

    Display pantalla; //Objeto Display
    HolaMundoForm form; //Objeto Form derivado de Screen
    HolaMundoCanvas canvas; //Objeto Canvas
    Command salir; //Objeto Command
```

Se empleará el método constructor de la clase para inicializar los objetos declarados. Los objetos **form** y **canvas** son creados a partir de las clases **HolaMundoForm** y **HolaMundoCanvas** que heredan de la clase Form y Canvas respectivamente.

Utilizando el objeto **pantalla** derivado de la clase Display se van a establecer los elementos que contendrá la pantalla del dispositivo.

```
//Metodo Constructor del MIDlet HolaMundo
public HolaMundo(){
        pantalla = Display.getDisplay(this);
        salir = new Command("SALIR",Command.EXIT,1);
        canvas = new HolaMundoCanvas(this);
        form = new HolaMundoForm("HOLA MUNDO J2ME (FORM)",this);
}
```

Se implementarán los métodos abstractos de la clase MIDlet, que representan los estados Activado, Pausado y Destruido ya descritos anteriormente. El método startApp() es el método por el cual nuestra aplicación iniciará su ejecución, que a diferencia de otras ediciones de java, en J2ME no se dispone de un método main. Este método primeramente establecerá en pantalla el saludo a través del objeto Form.

```
//Estado de Activado
public void startApp() {
        System.out.println("ACTIVADO");
        pantalla.setCurrent(form);
}
//Estado de Pausado
public void pauseApp() {
        System.out.println("PAUSADO");
}
//Estado de Destruido
public void destroyApp(boolean unconditional) {
        System.out.println("DESTRUIDO");
}
```

Por último se ha definido el método **salir()** que se encargará de finalizar la aplicación llamando al método destroyApp() y notifyDestroyed() que destruirá y notificará al dispositivo de la destrucción concluyendo así toda actividad del MIDlet.

```
//Método para cerrar la aplicacion
    public void salir(){
        destroyApp(true);
        notifyDestroyed();
    }
}
```

En la siguiente parte se declara la clase **HolaMundoForm** que hereda de Form, esta clase construirá el mensaje "Hola Mundo J2ME" que será mostrado en el objeto Form del MIDlet. Implementará la interfaz CommandListener que a través de su método commandAction() permitirá realizar las operaciones necesarias al producirse un evento a través de objetos Command.

En esta clase únicamente se definen dos variables, un objeto **HolaMundo** y un objeto Command que va a ser el encargado de provocar el evento para el cambio de mensaje "Hola Mundo J2ME" de un objeto tipo Form a un objeto tipo Canvas.

```
package ejemplo_1;
import javax.microedition.lcdui.*;

/**
 *
 * @author JAVAXIAN
 */
public class HolaMundoForm extends Form implements CommandListener{

   HolaMundo mid; //Objeto HolaMundo
   Command opcCanvas; //Objeto Command para la opcion Form
```

Por medio del método constructor se inicializan las variables. No existen métodos constructores de la clase Form que no reciban parámetros, uno de los constructores recibe como parámetro una cadena de texto, por lo que al heredar de la clase Form es necesario enviar como mínimo ese parámetro, por medio del método super() se manda la cadena de texto a la clase padre Form.

Para añadir los objetos Command se utiliza el método addCommand() y para poder escuchar los eventos generados se emplea el método setCommandListener().

```
//Método Constructor de la clase HolaMundoForm
   HolaMundoForm(String titulo, HolaMundo mid){
      //inicializacion de objetos
      super(titulo);
      this.mid = mid;
      opcCanvas = new Command("Canvas",Command.SCREEN,2);
      this.addCommand(mid.salir);
      this.addCommand(opcCanvas);
      this.setCommandListener(this);
   }
```

Se implementa el método commandAction() de la interfaz CommandListener, del cual obtenemos el evento para posteriormente realizar las operaciones apropiadas.

El evento producido por el objeto **salir** invoca al método salir del la clase **HolaMundo**, el cual finaliza la aplicación.

El evento producido por el objeto **opcCanvas** despliega en pantalla un objeto Canvas a través del método setCurrent() de la clase Display.

```
 //Método a implementar de la Interfaz CommandListener
   public void commandAction(Command cmd, Displayable dis) {
```

```
        if(cmd==mid.salir){
            mid.salir();
        }else if(cmd==opcCanvas){
            mid.pantalla.setCurrent(mid.canvas);
        }
    }
}
```

La última clase utilizada para esta aplicación es **HolaMundoCanvas** que hereda de Canvas y también implementa la interfaz CommandListener. Esta clase se encarga de construir el mensaje "Hola Mundo J2ME" que será mostrado en el objeto Canvas del MIDlet.

De igual forma que en la clase anterior emplearemos dos variables, un objeto **HolaMundo** y un objeto Command que en este caso provocará el evento para cambiar el mensaje "Hola Mundo J2ME" de un objeto tipo Canvas a un objeto tipo Form.

```
package ejemplo_1;
import javax.microedition.lcdui.*;

/**
 *
 * @author JAVAXIAN
 */
public class HolaMundoCanvas extends Canvas implements CommandListener{

    HolaMundo mid; //Objeto HolaMundo
    Command opcForm; //Objeto Command para la opcion Form
```

Con el método constructor se inicializan las variables y se agregan los comandos correspondientes.

```
//Método Constructor de HolaMundoCanvas que hereda de Canvas
    public HolaMundoCanvas(HolaMundo mid){
        //inicializacion de objetos
        opcForm = new Command("Form",Command.SCREEN,2);
        this.mid = mid;
        this.addCommand(mid.salir);
        this.addCommand(opcForm);
        this.setCommandListener(this);
    }
```

Se implementará el método abstracto paint() de la clase Canvas, el cual emplea un objeto de tipo Graphics para pintar el contenido que se va a desplegar en pantalla.

```
//Método a implementar de la clase abstracta Canvas
   protected void paint(Graphics g) {
      //Objeto Graphics g para manejo de graficos en pantalla
      g.setColor(0,0,0);
      g.fillRect(0,0,this.getWidth(),this.getHeight());
      g.setColor(255,255,255);
      g.drawString("HOLA MUNDO J2ME (CANVAS)",(this.getWidth()/2),(this.getHeight()/2),
               Graphics.BASELINE|Graphics.HCENTER);
   }
```

De la misma forma se implementa el método commandAction() para efectuar las acciones de cambio del contenido de la pantalla y salir de la aplicación.

```
//Método a implementar de la Interfaz CommandListener
   public void commandAction(Command cmd, Displayable dis) {
      if(cmd==mid.salir){
         mid.salir();
      }else if(cmd==opcForm){
         mid.pantalla.setCurrent(mid.form);
      }
   }
}
```

El emulador nos muestra nuestro MIDlet en ejecución, el cual muestra un mensaje de "Hola Mundo J2ME", y obedeciendo al evento del comando Canvas ó Form cambia el contenido de pantalla. Una tipo de Interfaz de Usuario de Alto nivel de abstracción se encuentra en la clase Form y Command, en la que se nos muestra el mensaje y comandos en zonas ya definidas por el dispositivo, a diferencia de una Interfaz de Usuario de Bajo nivel de abstracción que se encuentra en la clase Canvas, en la cual se tiene que especificar el contenido, así como posición y atributos de los elementos a ser mostrados.

Con esto se da por terminado el primer ejemplo "HOLA MUNDO J2ME" obteniendo las siguientes imágenes en la que se puede ver el resultado final de ejecutar el MIDlet sobre el emulador proporcionado en el IDE NetBeans 6.1.

Figura 2.9

MIDLET "HolaMundo" en ejecución

Ejemplo 2: Programa "Calculadora J2ME" Manejo de componentes derivados de las clases Screen e Item

Principalmente este ejemplo describirá la utilidad de un objeto derivado de la clase Form, el cual permite contener cualquier tipo de objeto derivado de la clase Item, lo que permite construir una interfaz de usuario lo más interactiva posible. Recuperando los valores establecidos por el usuario este MIDlet realizará operaciones básicas de una calculadora como suma, resta multiplicación ó división.

Se declaran objetos y variables que después son inicializados a través del método constructor de la clase. Gran parte de los objetos declarados derivan de la clase Item.

```
package ejemplo_2;

import java.io.IOException;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * @author JAVAXIAN
 */
public class Calculadora extends MIDlet implements CommandListener, ItemStateListener{

    Display pantalla; //Objeto Display
    Form formulario; //Objeto Form
    ChoiceGroup grupo; //Objeto ChoiceGroup
    TextField num_1,num_2; //Objetos TextField
    StringItem res; //Objeto StringItem
    Command salir,resultado; //Objetos Command
    Ticker ticker; //Objeto Ticker
    ImageItem imgItem; // Objeto ImageItem
    //Arreglo de objetos Spacer para espacios entre items
    Spacer espacio[] = new Spacer[3];
    //Arreglo que contiene nombre de las operaciones
    String opciones[] ={"Sumar","Restar","Multiplicar","Dividir"};
    //Arreglo para almacenar las imagenes a montar en el ChoiceGroup
    Image imagenes[] = new Image[4];
    //Imagen usada para la alerta
    Image imagenAlert,img;
    //Variable para guardar opcion seleccionada
    int op = 0;
    //Objeto de la clase Operaciones
    Operaciones operacion;
    //Objeto Alert
    Alert alerta;
```

Los objetos Form y Alert son derivados de la clase Screen, mientras que los objetos TextField, ChoiceGroup, ImageItem y Spacer son derivados de la clase Item, los cuales posteriormente incorporaremos sobre nuestro objeto Form. Para darle mejor vista a nuestra aplicación se ha utilizado la clase Ticker, Spacer e ImageItem, la primera clase se encarga de mostrar una cadena de texto desplazándose de derecha a izquierda en la parte superior de la pantalla, la segunda clase permite establecer espacios de separación entre objetos Item dentro del formulario, y la tercer clase permite incorporar una imagen de extensión .png como Item de un formulario.

Para esta ocasión se ha implementado una nueva interfaz ItemStateListener que permite utilizar eventos a partir del estado en que se encuentre un Item. En este ejemplo dichos eventos serán provocados por el objeto ChoiceGroup.

```
//Método Constructor
public Calculadora(){
      //inicializacion de objetos
      pantalla = Display.getDisplay(this);
      ticker = new Ticker("CALCULADORA J2ME");
      formulario = new Form("");
      num_1 = new TextField("Numero 1:","",9,TextField.NUMERIC);
      num_1.setLayout(1);
      num_2 = new TextField("Numero 2:","",9,TextField.NUMERIC);
      num_2.setLayout(1);
      res = new StringItem("Resultado: ","",StringItem.BUTTON);
      salir = new Command("SALIR",Command.EXIT,1);
      resultado = new Command("=",Command.OK,2);
      operacion = new Operaciones();
      espacio[0] = new Spacer(40,20);

      espacio[1] = new Spacer(20,20);
      espacio[2] = new Spacer(20,20);
      //Bloque try-catch para crear las imagenes del ChoiceGroup
      try{
         img = Image.createImage("/ejemplo_2/Calculadora2.png");
         imagenAlert = Image.createImage("/ejemplo_2/iconoPNG.PNG");
         imagenes[0] = Image.createImage("/ejemplo_2/IMGsuma2.PNG");
         imagenes[1] = Image.createImage("/ejemplo_2/IMGresta2.PNG");
         imagenes[2] = Image.createImage("/ejemplo_2/IMGmult2.PNG");
         imagenes[3] = Image.createImage("/ejemplo_2/IMGdiv2.PNG");
      }catch(IOException ioE){System.out.println("ERROR: "+ioE.getMessage());}
      imgItem = new ImageItem("", img, ImageItem.LAYOUT_CENTER, "Imagen Muy Grande");
      alerta = new Alert("ALERTA: Campos Vacios","Ingresar valor en campos de NUMERO !!!"
                  ,imagenAlert,AlertType.ERROR);
      alerta.setTimeout(Alert.FOREVER);

      grupo = new ChoiceGroup("Operacion: ",Choice.POPUP,opciones,imagenes);
      grupo.setSelectedIndex(0,true);
      //Añadiendo los elementos al formulario
      formulario.append(imgItem);
      formulario.append(espacio[0]);
      formulario.append(num_1);
      formulario.append(num_2);
      formulario.append(espacio[1]);
      formulario.append(grupo);
      formulario.append(res);
      formulario.setTicker(ticker);
      formulario.addCommand(salir);
      formulario.addCommand(resultado);
      formulario.setItemStateListener(this);
      formulario.setCommandListener(this);
   }
```

Nuevamente volvemos a implementar los métodos abstractos definidos en la clase abstracta MIDlet.

En el método startApp() se establece el objeto de tipo Displayable (Canvas o Screen) a ser mostrado, en este caso el objeto Form que a su vez contiene los Item ChoiceGroup, TextField y Spacer.

```
    //Estado Activado
    public void startApp() {
        System.out.println("ACTIVADO");
        pantalla.setCurrent(formulario);
    }
    //Estado Pausado
    public void pauseApp() {
        System.out.println("PAUSADO");
    }
    //Estado Destruido
    public void destroyApp(boolean unconditional) {
        System.out.println("DESTRUIDO");
    }
```

Al implementar el método commandAction() se establecen las operaciones que hay que realizar al provocarse un evento. El objeto **salir** permite cerrar la aplicación, mientras que el objeto **resultado** creará una instancia de la clase **Operaciones** que ejecutará la operación seleccionada con los valores contenidos en los Item TextField, que en caso de ser valores nulos desplegará un mensaje de alerta construido con un objeto Alert creado e inicializado con anterioridad.

```
//Método a implementar de la interfaz CommandListener
    public void commandAction(Command cmd, Displayable dis) {

        if(cmd==salir){
            destroyApp(true);
            notifyDestroyed();
        }else if(cmd==resultado){

            if(num_1.getString().equals("") || num_2.getString().equals("")){
              //SON NULOS"
                pantalla.setCurrent(alerta,dis);
            }else{
                //NO SON NULOS"
                long n1 = Long.parseLong(num_1.getString());
                long n2 = Long.parseLong(num_2.getString());
                String re = String.valueOf(operacion.ejecutar(op, n1, n2));
                res.setText(re);
            }
        }
    }
```

Por último se implementa el método itemStateChanged() de la interfaz ItemStateListener que nos permitirá conocer el estado de nuestro grupo de elementos en el ChoiceGroup para poder establecer la operación que se desea realizar en la calculadora.

```
//Método a implementar de la interfaz ItemStateListener
   public void itemStateChanged(Item I) {
         op = grupo.getSelectedIndex();
   }
}//Fin Clase
```

Posteriormente se define la clase **Operaciones** encargada de realizar las operaciones de suma, resta, multiplicación y división a través de método **ejecutar()** que recibe como parámetros la opción de la operación a realizar y los números para realizar la misma.

```
package ejemplo_2;

/**
 *
 * @author JAVAXIAN
 */
public class Operaciones {

   //Método encargado de ejecutar la operacion seleccionada
   public long ejecutar(int op,long num_1,long num_2){
      long resultado = 0;
      switch(op){
         case 0: resultado = num_1+num_2; break;
         case 1: resultado = num_1-num_2; break;
         case 2: resultado = num_1*num_2; break;
         case 3: resultado = num_1/num_2; break;
      }
      return resultado;
   }
}
```

Hay que señalar que el MIDLET “Calculadora J2ME” fue creado a partir de la configuración CLDC 1.0 y perfil MIDP 2.0 por lo que las operaciones son manejadas con valores enteros (int), en caso de que se quiera operar con valores decimales la configuración deberá ser cambiada a la versión CLDC 1.1 que brinda soporte para punto flotante y de igual forma en el código deberán ser sustituidas las variables int por variables float ó double.

El resultado del MIDLET en ejecución se muestra en las siguientes imágenes:

Figura 2.10

MIDLET "Calculadora J2ME" en ejecución

Ejemplo 3: Programa "BDme" Manejo en la persistencia de datos utilizando el paquete javax.microedition.rms

Utilizando el paquete javax.microedition.rms en este ejemplo se va a describir la forma de almacenar datos de manera persistente sobre una aplicación. Este MIDlet simulará operaciones de una pequeña base de datos como: agregar, actualizar ó eliminar un registro así como borrar por completo la base de datos. Para visualizar los datos almacenados a través de una sencilla tabla se utilizará la clase CustomItem que derivada de Item permite dibujar su propio contenido.

La clase principal que hereda de MIDlet es **BDme** que en esta ocasión establecerá en la pantalla de inicio un menú que será construido a partir de la clase List. Existen tres tipos de listas: exclusiva, múltiple e implícita, la lista exclusiva permite seleccionar tan solo un elemento de la lista, la lista múltiple permite seleccionar más de un elemento de la lista y la lista implícita genera un evento al seleccionar uno de sus elementos lo que en la aplicación permitirá mostrar el formulario adecuado a la opción seleccionada.

```
package ejemplo_3;

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

/**
 * @author JAVAXIAN
 */
public class BDme extends MIDlet implements CommandListener{

    Display pantalla; //Objeto Display
    List menu; //Objeto List
    Formularios form; //Objeto Form
    Command salir; //Objeto Command
    //Método Constructor
    public BDme(){
        //inicializacion de objetos
        pantalla = Display.getDisplay(this);
        menu = new List("***M E N U***",List.IMPLICIT);
        menu.append("Agregar Registro", null);
        menu.append("Actualizar Registro", null);
        menu.append("Eliminar Registro", null);
        menu.append("Borrar Registros", null);
        menu.append("Mostrar Tabla", null);
        salir = new Command("Salir",Command.EXIT,1);
        menu.addCommand(salir);
        menu.setCommandListener(this);
    }
```

Implementando los métodos abstractos del MIDlet en el método startApp() se va a establecer en pantalla la lista que contiene el menú de opciones.

```
    //Estado Activado
    public void startApp() {
        pantalla.setCurrent(menu);
        System.out.println("ACTIVADO");
    }
    //Estado Pausado
    public void pauseApp() {
        System.out.println("PAUSADO");
    }
    //Estado Destruido
    public void destroyApp(boolean unconditional) {
        System.out.println("DESTRUIDO");
    }
```

De la misma forma el evento de la lista implícita del menú es capturado en el método commandAction() y utilizando el campo SELECT_COMMAND del objeto **menu** se obtiene la opción seleccionada lo que va a permitir por medio de objetos Form y Alert visualizar el contenido adecuado en pantalla.

```
//Método a implementar de la interfaz CommandListener
   public void commandAction(Command cmd, Displayable dis) {

      if(cmd==salir){
         destroyApp(true);
         notifyDestroyed();
      }else if(cmd==menu.SELECT_COMMAND){
         switch(menu.getSelectedIndex()){
            case 0: form = new Formularios(this);
                  pantalla.setCurrent(form.obtenerForm(0));
            break;
            case 1: form = new Formularios(this);
                  pantalla.setCurrent(form.obtenerForm(1));
            break;
            case 2: form = new Formularios(this);
                  pantalla.setCurrent(form.obtenerForm(2));
            break;
            case 3: form = new Formularios(this);
                  pantalla.setCurrent(form.obtenerForm(3));
            break;
            case 4: form = new Formularios(this);
                  pantalla.setCurrent(form.obtenerForm(4));
            break;
         }
      }
   }
}//Fin Clase
```

Se define una clase **Formularios** que tendrá como función principal establecer en pantalla el contenido adecuado a la opción seleccionada en el menú. Según la opción seleccionada, la clase podrá crear tres diferentes formularios y una alerta por lo que también se encargará de realizar las operaciones establecidas en cada uno ellos.

Tendrá referencias a otras clases como: BDme, BaseRMS y Tabla. La clase BDme servirá para establecer en el MIDlet el formulario ó alerta sobre pantalla. La clase BaseRMS servirá para establecer las operaciones sobre la base de datos. La clase Tabla permitirá mostrar los datos sobre una tabla. Posteriormente cada una de estas clases será descrita con más detalle.

En el método constructor tan solo se inicializarán los objetos **base**, **mid** y **atras**.

```
package ejemplo_3;

import javax.microedition.lcdui.*;

/**
 *
 * @author JAVAXIAN
 */
```

```
public class Formularios implements CommandListener{

   Command atras,agr,act,eli; //Objetos Command
   Form agregar,actualizar,eliminar,borrar,tabla; //Objetos Form
   Tabla itemTabla; //Objeto Tabla derivado de CustomItem
   TextField nombre,id; //Objetos TextField
   BDme mid; //Objeto BDme derivado de MIDlet
   BaseRMS base; //Objeto BaseRMS
   Alert mensaje; //Objeto Alert
   //Variable para el mensaje de exito
   String exito = "Operacion realizada con EXITO!!!";
   //Variable para el mensaje de error
   String error = "Operacion no realizada ERROR!!!";
   //Método Constructor
   public Formularios(BDme mid){
      //inicializacion de objetos
      atras = new Command("ATRAS",Command.BACK,2);
      base = new BaseRMS();
      this.mid = mid;
   }
```

El método **generarMensaje()** se encarga de crear y mostrar en pantalla la alarma que indica si las operaciones sobre la base de datos han sido realizadas de manera exitosa ó errónea. Recibe como parámetro un valor boleano que de ser verdadero inicializa el objeto **mensaje** como una alerta con el texto de "EXITO" y de tipo de información, mientras que un valor falso inicializa la alerta con el "ERROR" y de tipo error.

```
//Método para generar el mensaje de exito o error en operacion
   public void generarMensaje(boolean e){
      mensaje = (e)?(new Alert("EXITO",exito,null,AlertType.INFO))
                  :(new Alert("ERROR",error,null,AlertType.ERROR));
      mensaje.setTimeout(Alert.FOREVER);
      mid.pantalla.setCurrent(mensaje, mid.menu);
   }
```

El método **obtenerForm()** nos permite obtener un objeto de tipo Form inicializando el objeto a partir del parámetro recibido que depende de la opción seleccionada en el menú de la pantalla inicial. Los tres diferentes formularios que son inicializados nos permiten realizar las operaciones de agregar, actualizar y eliminar. Mientras que la cuarta opción genera una alerta indicando si la base de datos puedo ser borrada por completo.

```
//Método para establecer Formulario
   public Form obtenerForm(int opcion){
      Form form = null;
      switch(opcion){
         //Formulario para AGREGAR
         case 0: agregar = new Form("Agregar Registro");
               agr = new Command("AGREGAR",Command.SCREEN,1);
```

```
            nombre = new TextField("Nombre: ","",30,TextField.ANY);
            agregar.append(nombre);
            agregar.addCommand(atras);
            agregar.addCommand(agr);
            agregar.setCommandListener(this);
            form = agregar;
        break;
        //Formulario para ACTUALIZAR
        case 1: actualizar = new Form("Actualizar Registro");
            act = new Command("ACTUALIZAR",Command.SCREEN,1);
            nombre = new TextField("Cambiar Nombre por: ","",30,TextField.ANY);
            id = new TextField("Actualizar ID:","",2,TextField.NUMERIC);
            actualizar.append(id);
            actualizar.append(nombre);
            actualizar.addCommand(atras);
            actualizar.addCommand(act);
            actualizar.setCommandListener(this);
            form = actualizar;
        break;
        //Formulario para ELIMINAR
        case 2: eliminar = new Form("Eliminar Registro");
            eli = new Command("ELIMINAR",Command.SCREEN,1);
            id = new TextField("Eliminar ID","",2,TextField.NUMERIC);
            eliminar.append(id);
            eliminar.addCommand(atras);
            eliminar.addCommand(eli);
            eliminar.setCommandListener(this);
            form = eliminar;
        break;
        //Alerta para avisar de datos borrados
        case 3: if(base.borrarTabla()){
                generarMensaje(true);
            }else{
                generarMensaje(false);
            }
        break;
        //Formulario para mostrar la Tabla de Datos
        case 4: tabla = new Form("Tabla de Registros");
            itemTabla = new Tabla("DATOS",base);
            id = new TextField("Eliminar ID","",2,TextField.NUMERIC);
            //tabla.append(id);
            tabla.append(itemTabla);
            tabla.addCommand(atras);
            tabla.setCommandListener(this);
            form = tabla;
        break;
    }

    return form;
}
```

Cada uno de los formularios al ser creados establecen sus propios objetos Command, de los cuales solo tienen en común el objeto **atrás** que se encarga de regresar la pantalla al menú principal. El método commandAction() se encarga de capturar los eventos y realizar las operaciones de cada uno de los formularios establecidos. Las operaciones las va realizar a través del objeto **base** de tipo BaseRMS.

```
//Método a implementar de la interfaz CommandListener
    public void commandAction(Command cmd, Displayable dis) {
        if(cmd==atras){
            mid.pantalla.setCurrent(mid.menu);
        }else if(cmd==agr){//Operaciones que ejecuta el comando AGREGAR
            base.abrirRegistros();
            if(base.agregar(nombre.getString())){
                base.cerrarRegistros();
                generarMensaje(true);
            }else{
                base.cerrarRegistros();
                generarMensaje(false);
            }
        }else if(cmd==act){//Operacions que ejecuta el comando ACTUALIZAR
            base.abrirRegistros();
            int idINT = Integer.parseInt(id.getString());
            if(base.actualizar(idINT, nombre.getString())){
                base.cerrarRegistros();
                generarMensaje(true);
            }else{
                base.cerrarRegistros();
                generarMensaje(false);
            }
        }else if(cmd==eli){//Operaciones que ejecuta el comando ELIMINAR
            base.abrirRegistros();
            int idINT = Integer.parseInt(id.getString());
            if(base.eliminar(idINT)){
                base.cerrarRegistros();
                generarMensaje(true);
            }else{
                base.cerrarRegistros();
                generarMensaje(false);
            }
        }
    }
}//Fin Clase
```

La clase BaseRMS se encarga de realizar las operaciones directamente con la base de datos, utilizando el paquete javax.microedition.rms puede crear ó borrar una base de datos que de aquí en adelante llamaremos RecordStore, dentro de este se pueden realizar operaciones como agregar, actualizar o eliminar registros.

La clase RecordStore no posee un método constructor por lo que para abrir o crear un RecordStore se utiliza el método openRecordStore() que recibe dos parámetros el primero de ellos una cadena de texto String que indica el nombre que identifica al RecordStore y el segundo parámetro un booleano que de tener valor verdadero permite crear un RecordStore en caso de no existir.

Para realizar alguna operación sobre los registros agregar, actualizar ó eliminar antes hay que abrir el RecordStore por lo que al finalizar las operaciones hay que cerrar el RecordStore a través del método closeRecordStore().

En la clase BaseRMS tan solo definimos dos variables el objeto **rs** que permitirá trabajar sobre el RecordStore y la variable **ID_BD** que es el identificador ó nombre del RecordStore.

```
import java.util.*;
import javax.microedition.rms.*;
/**
 *
 * @author JAVAXIAN
 */
public class BaseRMS {
    RecordStore rs = null; //Objeto RecordStore
    private final String ID_BD = "Nombres"; //Variable para nombre de la BASE
```

El método abrirRegistros() creará ó abrirá un RecordStore existente.

```
//Método para abrir el RecordStore
    public void abrirRegistros(){
        try{
            rs = RecordStore.openRecordStore(ID_BD, true);
        }catch(Exception e){//El registro no pudo ser abierto
                System.out.println(e.toString());
        }
    }
```

El método cerrarRegistros() cerrará el RecordStore que se esté utilizando. No hay que olvidar que por cada llamada para abrir el RecordStore debe existir una llamada para cerrar el RecordStore para poder finalizar y guardar los datos de forma correcta.

```
//Método para cerrar el RecordStore
    public void cerrarRegistros(){
        try{
            rs.closeRecordStore();
        }catch(Exception e){//El registro no pudo ser cerrado
            System.out.println(e.toString());
        }
    }
```

Para añadir un registro por medio del objeto **rs** se hace una llamada al método addRecord() el cual solamente puede almacenar arreglos de bytes. Estas operaciones se encuentran en el método **agregar()** que recibe como parámetro una cadena String de la cual se obtienen sus bytes en un arreglo para ser almacenado en el RecordStore.

```
//Método para agregar un registro en el RecordStore
   public boolean agregar(String nombre){
      byte[] registro;
      registro = nombre.getBytes();
      try{
         rs.addRecord(registro, 0, registro.length);
      }catch(Exception e){//No se pudo agregar el registro
         return false;
      }
      return true;
   }
```

Para eliminar un registro se hace una llamada al método **deleteRecord()** enviando como único parámetro un número entero único que sirve de identificador para el registro. Esta operación se encuentra en el método eliminar() que recibe como parámetro el valor del identificador para eliminar el registro.

```
//Método para eliminar un registro en el RecordStore
   public boolean eliminar(int id){
      try{
         rs.deleteRecord(id);
      }catch(Exception e){//No se pudo eliminar el registro
         return false;
      }
      return true;
   }
```

Para actualizar un registro se hace una llamada al método **setRecord()** el cual actualiza algún registro existente. Esta operación se encuentra en el método **actualizar()** que recibe el id del registro a modificar con el valor nuevo del registro.

```
//Método para actualizar un registro en el RecordStore
   public boolean actualizar(int id, String nvo_nombre){
      byte[] registro;
      registro = nvo_nombre.getBytes();
      try{
         rs.setRecord(id, registro, 0, registro.length);
      }catch(Exception e){//No se pudo actualizar registro
         return false;
      }
      return true;
   }
```

Hay más de una forma de recuperar todos los registros de un RecordStore, para este ejemplo se almacenan primero en un RecordEnumeration haciendo una llamada al método enumerateRecords() de esta manera con un ciclo while permite ir recuperando cada uno de los registros almacenados para posteriormente ser añadidos en un objeto Vector como objetos de tipo **Persona**. El método **obtenerRegistros()** es el encargado de realizar estas operaciones.

```
//Método para obtener los registros del RecordStore
   public Vector obtenerRegistros(){
      Vector vec_base = new Vector();
      Persona persona;
      try{
         if(rs.getNumRecords()>0){
            RecordEnumeration re = rs.enumerateRecords(null,null,false);
            RecordEnumeration reID = rs.enumerateRecords(null, null, false);
            while(re.hasNextElement()){
               String str = new String(re.nextRecord());
               String strID = new String(String.valueOf(reID.nextRecordId()));
               persona = new Persona(strID,str);
               vec_base.addElement(persona);
            }
            re.destroy();
         }
      }catch(Exception e){//No se pudo obtener registros
         System.out.println(e.toString());
      }
      return vec_base;
   }
```

Para borrar por completo un RecordStore existente se hace una llamada al método deleteRecordStore(). Esta operación la realiza el método **borrarTabla()** que recibe como parámetro una cadena String que contiene el nombre del RecordStore a eliminar.

```
//Método para borrar los registros del RecordStore
   public boolean borrarTabla(){
      try{
         RecordStore.deleteRecordStore(ID_BD);
      }catch(Exception e){//No se pudo borrar los registros
         return false;
      }
      return true;
   }
}
```

Hay que señalar que en esta clase los métodos **agregar()**, **actualizar()**, **eliminar()** y **borrarTabla()** retornan un valor booleano, si es verdadero indica que la operación fue

realizada de manera correcta sin entrar en ninguna excepción, de ser falso indica una excepción y la operación no se concluye.

Para mostrar los datos almacenados se dibuja una tabla en pantalla y la clase **Tabla** es encargada de realizarlo. Esta clase es la que hereda de CustomItem lo que va a permitir dibujar una tabla sobre un Item personalizado.

```
package ejemplo_3;
import java.util.*;
import javax.microedition.lcdui.*;
/**
 *
 * @author JAVAXIAN
 */
public class Tabla extends CustomItem{
    int filas = 0; //variable para No. de Filas de la Tabla
    int columnas = 3; // variable para No. de Columnas de la Tabla
    int id_tamX = 20; // variable para tamaño en "x" del campo ID
    int datos_tamX = 150; // variable para tamaño en "x" del campo NOMBRE
    int tamY = 20; // variable para tamaño en "y" de ambos campos
    BaseRMS b; //Objeto BaseRMS
    //Objeto Vector que guarada los valores obtenidos del RecordStore
    Vector vec_tabla;
    //Método Constructor
    public Tabla(String titulo,BaseRMS base){
        //inicializacion de objetos
        super(titulo);
        b = base;
        b.abrirRegistros();
        vec_tabla = b.obtenerRegistros();
        filas = vec_tabla.size();
    }
```

Al heredar de la clase CustomItem hay que implementar sus métodos abstractos.

```
// ***Métodos Abstractos de la clase CustomItem***
    //Método que devuelve la anchura minima del area contenida
    protected int getMinContentWidth() {
        return (datos_tamX+id_tamX) + 1;
    }
    //Método que devuelve la altura minima del area contenida
    protected int getMinContentHeight() {
       return (filas * tamY) + 1;
    }
    //Método que devuelve la anchura preferida del area contenida
    protected int getPrefContentWidth(int width) {
        return (datos_tamX+id_tamX) + 1;
    }
    //Método que devuelve la altura preferida del area contenida
    protected int getPrefContentHeight(int height) {
        return (filas * tamY) + 1;
    }
```

El método paint() es otro de los métodos abstractos de CustomItem y nos va a permitir dibujar el contenido sobre el Item de forma similar a Canvas. Es en este método donde se va a dibujar la tabla con los valores obtenidos del RecordStore.

```
//Método paint para dibujar el contenido del CustomItem
   protected void paint(Graphics g, int w, int h) {

      for (int i = 0; i <= filas; i++) {
         g.drawLine(0, i * tamY, (datos_tamX+id_tamX), i * tamY);
      }
      for (int i = 0; i <= columnas; i++) {
         if(i==0){
            g.drawLine(i, 0, i, filas * tamY);
         }else
         if(i==1){
            g.drawLine(i * id_tamX, 0, i * id_tamX, filas * tamY);
         }else
         if(i==2){
            g.drawLine(datos_tamX+id_tamX, 0, datos_tamX+id_tamX, filas * tamY);
         }
      }
      for(int i = 1;i<=vec_tabla.size();i++){
         String str = ((Persona)(vec_tabla.elementAt(i-1))).getId();
         String str2 = ((Persona)(vec_tabla.elementAt(i-1))).getNombre();
         g.drawString(str, 2, (i * tamY)+1, Graphics.BOTTOM | Graphics.LEFT);
         g.drawString(str2, 22, (i * tamY)+1, Graphics.BOTTOM | Graphics.LEFT);
      }
      b.cerrarRegistros();
   }
}//Fin Clase
```

Para finalizar el ejemplo se va a describir la clase **Persona**. Esta clase encapsula los valores del id y nombre que se han almacenado en el RecordStore, por cada valor que es recuperado es creada una instancia de esta clase con el valor del id y nombre que posteriormente es añadido en un Vector de objetos tipo **Persona**. Este Vector es utilizado por la clase Tabla para mostrar los valores sobre la tabla dibujada.

La clase tan solo define las variables id y nombre con sus métodos setters y getters.

```
package ejemplo_3;
/**
 *
 * @author JAVAXIAN
 */
public class Persona {
   private String id;//variable para valor id
   private String nombre;// variable para valor nombre
   //Método Constructor
   public Persona(String id, String nombre){
```

```
        //inicializacion de variables
        this.id = id;
        this.nombre = nombre;
    }

    //**Metodos getter y setter
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}//Fin Clase
```

Parte del MIDlet "BDme "en ejecución se muestra en las siguientes capturas de pantalla del emulador.

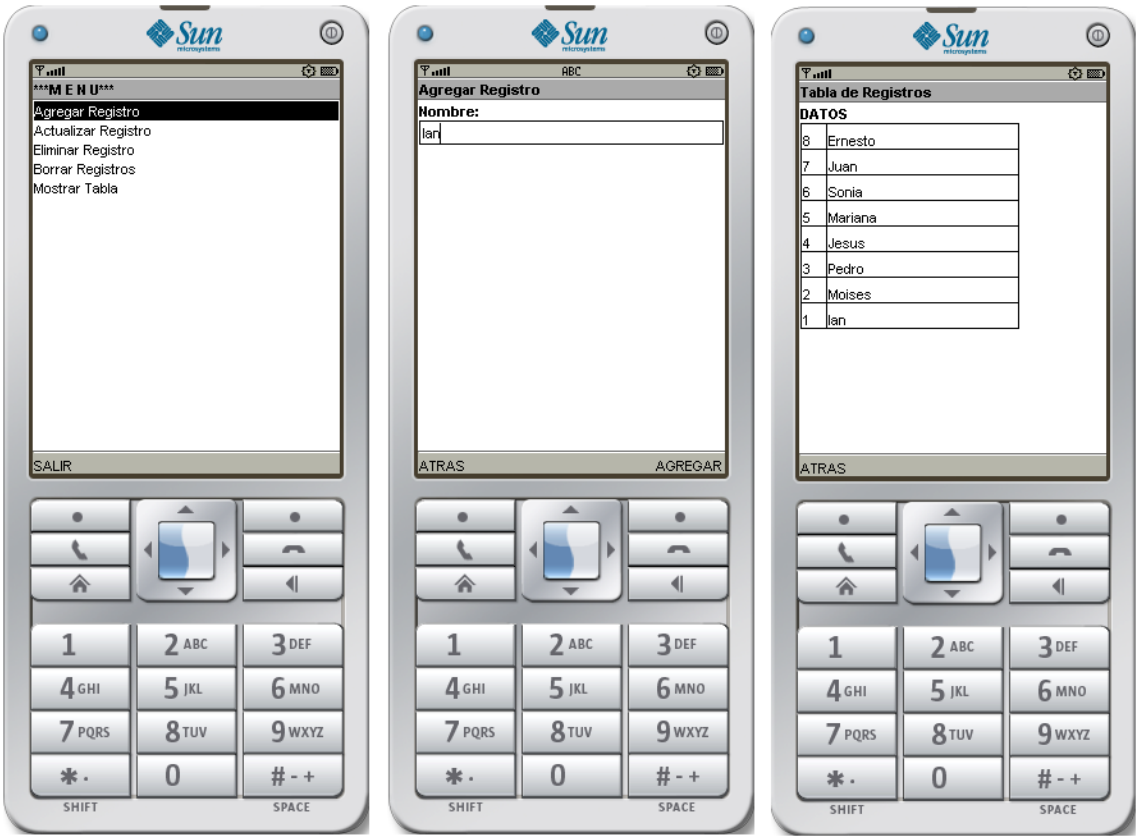


Figura 2.11

MIDLET "Bdme" en ejecución

Imágenes que muestran el menú principal, el formulario para agregar un registro y los registros almacenados en el RecordStore. En el primer ejemplo se distinguió la diferencia entre una interfaz de alto y bajo nivel de abstracción, cabe aclarar que en este último ejemplo la clase CustomItem nos brinda una combinación de ambos niveles ya que el dispositivo lo reconoce como algún otro Item pero el programador es el encargado de pintar su contenido.

Con este tercer ejemplo se da por terminado el capítulo dos cumpliendo el objetivo de introducirnos al funcionamiento y programación con J2ME. En el siguiente capítulo se estudiará a fondo la programación de aplicaciones utilizando la especificación JSR - 82 para J2ME y su adaptación BlueCove para J2SE. Esto con el fin de brindar los elementos necesarios para el desarrollo del caso práctico a desarrollar en el cuarto capítulo.

CAPÍTULO 3: API'S DE J2ME PARA BLUETOOTH

PARTE 1

DESCRIPCIÓN DE LA ESPECIFICACIÓN JSR - 82

PARTE 2

PROGRAMACIÓN DE CLIENTES Y SERVICIOS BLUETOOTH EN J2ME UTILIZANDO LA ESPECIFICACIÓN JSR - 82

PARTE 3

PROGRAMACIÓN DE SERVICIOS BLUETOOTH EN J2SE UTILIZANDO IMPLEMENTACIÓN BLUECOVE

PARTE 1
DESCRIPCIÓN DE LA ESPECIFICACIÓN JSR – 82

Introducción

En el primer capítulo se describió como surge, como funciona y lo ampliamente extendida que se encuentra la Tecnología Bluetooth en diferentes dispositivos especialmente los teléfonos celulares que en su mayoría también incorporan soporte para la tecnología J2ME. Para conjuntar y enriquecer más el potencial de las aplicaciones a través de estas dos tecnologías surge como paquete opcional la especificación JSR – 82 que "Pretende estandarizar un conjunto de APIs Java que permitan a los dispositivos móviles CLDC, ó incluso a cualquier dispositivo con soporte Java, integrarse en un entorno Bluetooth"[10].

Paquetes y Clases de la especificación JSR -82

La especificación JSR – 82 cuenta con dos paquetes que pueden trabajar de forma independiente javax.bluetooth y javax.obex. El paquete javax.bluetooth define las clases e interfaces necesarias para descubrir servicios y dispositivos así como también establecer una conexión y comunicación la cual es a un bajo nivel de abstracción a través de flujos de datos ó transmisión de arrays de bytes. El paquete javax.obex permite utilizar un protocolo denominado OBEX (OBject EXchange) brindando un alto nivel de abstracción similar a HTTP y enfocado principalmente al intercambio de archivos.

Anteriormente cuando se expuso la tecnología Bluetooth, se reviso la pila de protocolos Bluetooth la cual permite mantener un diálogo fluido entre diferentes dispositivos a través de diferentes aplicaciones. Con los paquetes javax.bluetooth y javax.obex la especificación JSR – 82 no implementa en su totalidad la pila de protocolos Bluetooth señalando como principal ausencia el protocolo para transmisión de audio.

[10] FROUFE QUINTAS, Agustin et al. "J2ME Java 2 Micro Edition Manual de usuario y tutorial" p.38.

El paquete javax.bluetooth contiene las siguientes clases, interfaces y excepciones:

DiscoveryListener	Interfaz que permite recibir eventos de dispositivos y servicios descubiertos.
L2CAPConnection	Interfaz que representa una conexión orientado al canal L2CAP.
L2CAPConnectionNotifier	Interfaz que provee un notificador de conexión L2CAP.
ServiceRecord	Interfaz que describe las características de un servicio Bluetooth.
DataElement	Clase que define varios tipos de datos para los valores de atributos de un servicio Bluetooth.
DeviceClass	Clase que representa el tipo de dispositivo "Class of Device" (COD) registrado, definido en la especificación Bluetooth.
DiscoveryAgent	Clase que proporciona métodos para realizar el descubrimiento de dispositivos y servicios.
LocalDevice	Clase que representa el dispositivo Bluetooth local.
RemoteDevice	Clase que representa el dispositivo Bluetooth remoto.
UUID	Clase que define el identificador único universal.
BluetoothConnectionException	Excepción lanzada cuando una conexión Bluetooth (L2CAP, RFCOMM ó OBEX) no puede establecerse con éxito.
BluetoothStateException	Excepción lanzada cuando una solicitud hecha al sistema Bluetooth no puede ser atendida en el momento.
ServiceRegistrationException	Excepción lanzada cuando se falla al añadir o modificar un registro de servicio al Service Discovery Database (SDDB).

Tabla 3.1

Paquete javax.bluetooth

El paquete javax.obex contiene las siguientes clases, interfaces y excepciones:

Authenticator	Interfaz que proporciona una manera de llevar a cabo la autenticación.
ClientSession	Interfaz que proporciona métodos para las solicitudes OBEX.
HeaderSet	Interfaz que define los métodos set y get de los valores de cabeceras OBEX.
Operation	Interfaz que proporciona los medios para manipular una operación OBEX PUT o GET.
SessionNotifier	Interfaz que define un notificador de conexión del lado del

	servidor para conexiones OBEX.
PasswordAuthentication	Clase que mantiene combinaciones de nombre y contraseña de usuario.
ResponseCodes	Clase que contiene lista de respuestas validas de código de un servidor, que puede enviar a un cliente.
ServerRequestHandler	Clase que define un escuchador de eventos que respondan a solicitudes del servidor OBEX.

Tabla 3.2

Paquete javax.obex

Implementación de la especificación JSR – 82 para J2SE BlueCove

Aunque la especificación JSR – 82 va enfocada para el desarrollo de aplicaciones en J2ME, actualmente existen diferentes implementaciones desarrolladas por terceros que permiten utilizar la especificación JSR – 82 en la edición J2SE. BlueCove es una implementación no oficial que proporciona el API Bluetooth para J2SE. Originalmente fue desarrollado por Intel Research, pero actualmente es mantenido por voluntarios. Es compatible con las versiones 1.1 en adelante de la máquina virtual de Java y se encuentra disponible para los Sistemas Operativos: Windows, Linux y Mac.

Bluecove brinda soporte para los siguientes perfiles Bluetooth:

- SDPA (Service Discovery Application Profile).
- RFCOMM (Serial Cable Emulation Protocol).
- L2CAP (Logical Link Control and Adaptation Protocol).
- OBEX (Generic Object Exchange Profile).

En la siguiente figura se muestra la forma en que se estructura la implementación BlueCove.

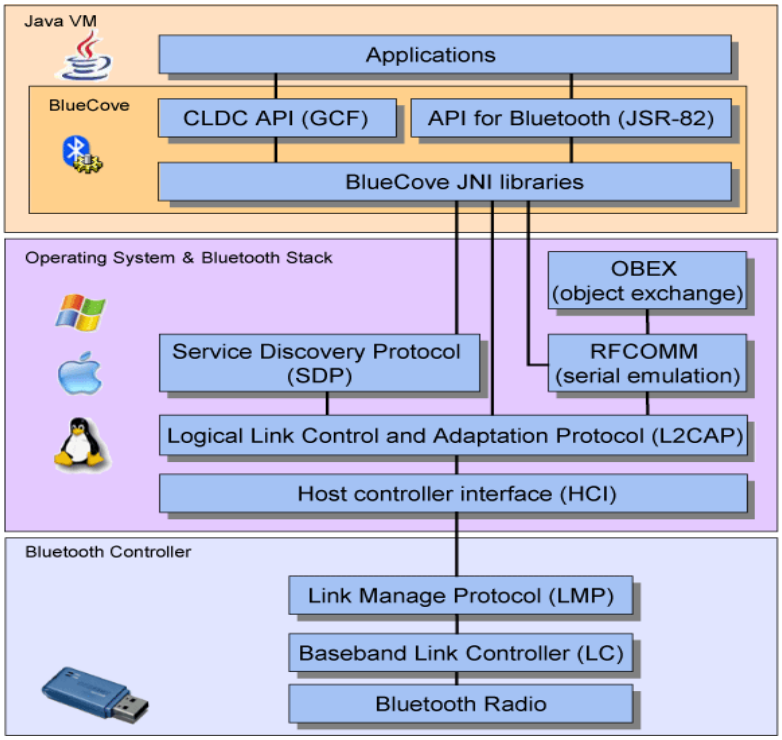


Figura 3.1

Estructura de la implementación BlueCove[11]

Paquetes y Clases del API BlueCove

BlueCove contiene la misma API definida en la especificación JSR – 82 de J2ME.

El paquete javax.bluetooth contiene las siguientes clases, interfaces y excepciones:

DiscoveryListener	Interfaz que permite recibir eventos de dispositivos y servicios descubiertos.
L2CAPConnection	Interfaz que representa una conexión orientado al canal L2CAP.
L2CAPConnectionNotifier	Interfaz que provee un notificador de conexión L2CAP.
ServiceRecord	Interfaz que describe las características de un servicio Bluetooth.
DataElement	Clase que define varios tipos de datos para los valores de atributos de un servicio Bluetooth.
DeviceClass	Clase que representa el tipo de dispositivo "Class of Device" (COD) registrado, definido en la especificación Bluetooth.

[11] http://www.bluecove.org/. 14 - Abril - 2009

DiscoveryAgent	Clase que proporciona métodos para realizar el descubrimiento de dispositivos y servicios.
LocalDevice	Clase que representa el dispositivo Bluetooth local.
RemoteDevice	Clase que representa el dispositivo Bluetooth remoto.
UUID	Clase que define el identificador único universal.
BluetoothConnectionException	Excepción lanzada cuando una conexión Bluetooth (L2CAP, RFCOMM ó OBEX) no puede establecerse con éxito.
BluetoothStateException	Excepción lanzada cuando una solicitud hecha al sistema Bluetooth no puede ser atendida.
ServiceRegistrationException	Excepción lanzada cuando se falla al añadir o modificar un registro de servicio al Service Discovery Database (SDDB).

Tabla 3.3

Paquete javax.bluetooth de BlueCove

El paquete javax.obex contiene las siguientes clases, interfaces y excepciones:

Authenticator	Interfaz que proporciona una manera de llevar a cabo la autenticación.
ClientSession	Interfaz que proporciona métodos para las solicitudes OBEX.
HeaderSet	Interfaz que define los métodos set y get de los valores de cabeceras OBEX.
Operation	Interfaz que proporciona los medios para manipular una operación OBEX PUT o GET.
SessionNotifier	Interfaz que define un notificador de conexión del lado del servidor para conexiones OBEX.
PasswordAuthentication	Clase que mantiene combinaciones de nombre y contraseña de usuario.
ResponseCodes	Clase que contiene lista de respuestas validas de código de un servidor, que puede enviar a un cliente.
ServerRequestHandler	Clase que define un escuchador de eventos que respondan a solicitudes del servidor OBEX.

Tabla 3.4

Paquete javax.obex de BlueCove

Protocolos SPP, L2CAP y OBEX

Un protocolo es un conjunto de procedimientos que controlan la secuencia de mensajes que se puede realizar durante la comunicación entre dos dispositivos. Para el API JSR – 82 se definen tres protocolos SPP, L2CAP y OBEX.

El protocolo SPP (Serial Port Profile).- Este protocolo se comunica a través de flujos de datos "Streams", es utilizado en apliaciones donde se pretende remplazar el cable serial para realizar la comunicación y en aplicaciones donde habrá una constante comunicación y procesamiento de datos entre cliente y servidor.

El protocolo L2CAP (Logical Link Control and Adaptation Protocol).- Este protocolo se comunica a través de paquetes por medio de arreglos de bytes, principalmente para el envio de objetos.

El protocolo OBEX (Object Exchange).- Similar al protocolo L2CAP pero manejado a un alto nivel de abstraccion, que es similar a HTTP, se utiliza principalmente para el envio de archivos u objetos.

Para el estudio de este trabajo será utilizado el protocolo SPP (Serial Port Profile), el cual nos brinda diferentes ventajas como permitir flujo constante de datos, fácil de utilizar y brindar soporte gran cantidad de dispositivos tanto móviles (teléfonos celulares) como computadoras.

Clientes y Servidores Bluetooth

La comunicación entre dispositivos Bluetooth se da de la forma cliente – servidor, por lo que algún dispositivo se encargará de brindar un servicio (servidor) y algunos otros se encargarán de acceder a dicho servicio (cliente). Para la programación de una aplicacion Bluetooth deberán realizarse diferentes acciones que van a depender si se trata de una aplicación cliente o aplicación de servidor.

Un cliente Bluetooth deberá realizar las siguientes acciones:

- Buscar dispositivos: En esta parte la aplicación se encargará de buscar y descubrir los dispositivos que se encuentren a su alcance y en modo conectable.

- Buscar servicios: Una vez descubiertos dispositivos, la aplicación se encargará de buscar los servicios que estos puedan ofrecer.

- Establecer conexión: Encontrados los servicios requeridos, la aplicación se encargará de conectarse al servidor.

- Realizar comunicación: Ya establecida una conexión la aplicación se encargará de la comunicación enviando y recibiendo datos.

Un servidor Bluetooth deberá realizar las siguientes acciones:

- Establecer un tipo de conexión servidora: La apliación definirá una conexion servidora estableciendo el tipo de protocolo que será utilizando, en el caso de JSR - 82 SPP, L2CAP y OBEX.

- Establecer atributos del servicio: Una vez establecida la conexión servidora, la aplicación podrá definir atributos del servicio que van a describir el tipo de servicio e información acerca del mismo.

- Atender conexiones: Ya iniciado el servicio, la aplicación se encargará de atender las conexiones solicitadas por clientes para poder ser abiertas.

- Realizar comunicación: Establecida ya una conexión con el cliente la aplicación podrá realizar una comunicación enviando y recibiendo datos.

Ya descritas las acciones que debe realizar una aplicación como cliente ó servidor, se procederá a la programación de aplicaciones utilizando el paquete JSR – 82 de J2ME. En la siguiente parte del capítulo se van a idear ejemplos que puedan vislumbrar la funcionalidad y utilidad de la programación de aplicaciones Bluetooth.

PARTE 2

PROGRAMACIÓN DE CLIENTES Y SERVICIOS BLUETOOTH EN J2ME UTILIZANDO LA ESPECIFICACIÓN JSR - 82

Introducción

Antes de comenzar a programar, hay que hablar claro acerca del significado de BCC (Bluetooth Control Center) y sobre la inicialización de la pila.

El BCC es responsable de controlar nuestro dispositivo Bluetooth, puede ser una aplicación nativa del dispositivo o un API definida por separado del paquete J2R – 82. El BCC es una implementación libre para el fabricante, sin embargo la especificación JSR – 82, establece para el BCC los siguientes requerimientos:

- Incluir configuración de seguridad en el dispositivo Bluetooth.
- Proporcionar una lista de dispositivos Bluetooth ya conocidos, aunque se encuentren fuera del rango de alcance.
- Proporcionar una lista de dispositivos Bluetooth que son de confianza, aunque se encuentren fuera del rango de alcance.
- Proporcionar un mecanismo para vincular dos dispositivos que intentan conectarse por primera vez.
- Proporcionar un mecanismo para establecer la autorización de las solicitudes de conexión.
- La información contenida en el BCC no puede ser modificada o alterada por otro que no sea el mismo BCC.

La inicialzación de la pila Bluetooth tiene acceso directo al dispositivo Bluetooth, consiste en realizar determinados procedimientos en los que el principial objetivo es conseguir que

el dispositivo Bluetooth se encuentre listo para iniciar una comunicación inalámbrica, el encargado de realizar este procedimiento es el BCC.

Descrito lo anterior que forma parte importante de la seguridad en las aplicaciones Bluetooth, se dará inicio al primer ejemplo el cual será una aplicación que obtenga las propiedades de nuestro dispositivo Bluetooth con el objetivo de poder conocer sus capacidades al momento de programar una aplicación cliente o servidor.

Ejemplo 1: Programa "PropiedadesBT" Obtención de propiedades del dispositivo Bluetooth

La siguiente aplicación obtendrá y mostrará las características con la que cuenta nuestro dispositivo Bluetooth. Esta aplicación no es cliente ni servidor, ya que no va a realizar ningún tipo de comunicación. Durante el ejemplo se explicará lo referente a la programación con la especificación JSR – 82 no entrando a detalle a los elementos de la programación CLCD Y MIDP que ya se describieron en el capitulo anterior.

La primera parte del código está compuesta por la declaración e inicialización de las variables a emplear, para este ejemplo será mostrado un menú con la opción de "Mi dispositivo" y "Historial", la primera va a devolver una lista con valores que definen las principales características y capacidades del dispositivo Bluetooth, mientras que la segunda va a devolver una lista de los dispositivos Bluetooth ya conocidos que hayan sido establecidos por el propio usuario.

```
package ejemplo_1;
import java.io.IOException;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
/**
 * @author JAVAXIAN
 */
public class PropiedadesBT extends MIDlet implements CommandListener{
    Display pantalla;
    List menu;
    Command salir,atras;
    TextBox texto;
    public PropiedadesBT(){
        pantalla = Display.getDisplay(this);
```

```
        menu = new List("***MENU***",List.IMPLICIT);
        menu.append("Mi dispositivo", null);
        menu.append("Historial", null);
        salir = new Command("SALIR",Command.EXIT,1);
        menu.addCommand(salir);
        menu.setCommandListener(this);
    }
```

Se implementan los métodos de la clase abstracta MIDlet y el método de la interfaz CommandListener, en el cuál se definen las llamadas a los métodos para realizar las operaciones adecuadas dependiendo del evento producido.

```
public void startApp() {
        pantalla.setCurrent(menu);
    }
    //Estados del MIDlet
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command cmd, Displayable dis) {
        if(cmd==salir){
            destroyApp(true);
            notifyDestroyed();
        }else if(cmd==menu.SELECT_COMMAND){
            switch(menu.getSelectedIndex()){
                case 0:
                    StringBuffer propiedades = obtenerPropiedades();
                    texto=new TextBox("***Propiedades***","",500,TextField.ANY);
                    texto.setString(propiedades.toString());
                    atras=new Command("ATRAS",Command.BACK,1);
                    texto.addCommand(atras);
                    texto.setCommandListener(this);
                    pantalla.setCurrent(texto);
                break;
                case 1:
                    StringBuffer historial = obtenerHistorial();
                    texto=new TextBox("***Historial***","",500,TextField.ANY);
                    texto.setString(historial.toString());
                    atras=new Command("ATRAS",Command.BACK,1);
                    texto.addCommand(atras);
                    texto.setCommandListener(this);
                    pantalla.setCurrent(texto);
                break;
            }
        }else if(cmd==atras){
            pantalla.setCurrent(menu);
        }
    }
```

El método **obtenerPropiedades()** regresará el objeto de tipo StringBuffer que contiene el valor de la lista de propiedades del dispositivo Bluetooth. Dentro del método se definen un objeto de tipo LocalDevice y uno de tipo DeviceClass.

```
public StringBuffer obtenerPropiedades(){
      StringBuffer p = new StringBuffer();
      LocalDevice ld = null;
     DeviceClass tipo = null;
```

LocalDevice es la clase que representa al dispositivo local Bluetooth, mientras que la clase DeviceClass representa un tipo particular de dispositivo (teléfono celular, computadora, impresora, PDA entre otros).

Haciendo una llamada al método getLocalDevice() de la clase LocalDevice se puede obtener un objeto LocalDevice que primeramente se utilizará para recuperar un objeto DeviceClass llamando al método getDeviceClass(). La Excepción que se puede generar al realizarse esta operación es BluetoothStateException que notificará que el objeto LocalDevice no pudo ser inicializado.

```
try{
   ld = LocalDevice.getLocalDevice();
   tipo = ld.getDeviceClass();
}catch(BluetoothStateException eb){
   System.out.println("ERROR "+eb.getMessage());
}
```

Una vez inicializado el objeto LocalDevice, se da comienzo a la obtención de propiedades del dispositivo, utilizando el metódo getBluetoothAddress() se va a conseguir la dirección física del dispositivo mientras que invocando el método getFriendlyName() se consigue el alias o apodo establecido por el propio usuario. Al obtener una propiedad esta se añade al objeto StringBuffer que se retornará como valor final del método.

```
 p.append("Dirección: "+ld.getBluetoothAddress()+"\n");
p.append("Alias: "+ld.getFriendlyName()+"\n");
```

El método getDiscoverable() retornará un valor entero que identifica la conectividad en que se encuentra el dispositivo. Se puede obtener uno de tres valores posibles, el valor retornado es evaluado en un bloque switch – case en el que se utilizan variables estáticas de la clase DiscoveryAgent (descrita posteriormente) para poder determinar el valor correcto que será añadido al objeto StringBuffer.

```
int modo = ld.getDiscoverable();
switch(modo){
   case DiscoveryAgent.NOT_DISCOVERABLE:
      p.append("Modo: NO DETECTABLE"+"\n");
```

```
        break;
        case DiscoveryAgent.GIAC:
           p.append("Modo: GENERAL"+"\n");
        break;
        case DiscoveryAgent.LIAC:
           p.append("Modo: LIMITADO"+"\n");
        break;
    }
```

La siguiente tabla muestra los valores de conectividad en el dispositivo Bluetooth:

MODO DE ACCESO	NOMBRE	DESCRIPCIÓN	VALOR
NOT_DISCOVERABLE	Not discoverable	No permite que el dispositivo sea descubierto.	0
GIAC	General/Unlimited Inquiry Access Code	Conectividad Ilimitada, permite que el dispositivo sea descubierto.	10390323
LIAC	Limited Inquiry Access Code	Conectividad Limitada, permite acceso temporal.	10390272

Tabla 3.5

Tipos de conectividad[12]

El método getProperty() es capaz de devolver distintas propiedades, el valor devuelto va a depender del parámetro enviado (String con valor de parámetro a recuperar).

```
p.append("Version: "+
      ld.getProperty("bluetooth.api.version")+"\n");
p.append("Cambio maestro/esclavo: "+
      ld.getProperty("bluetooth.master.switch")+"\n");
p.append("No. max de atributos: "+
      ld.getProperty("bluetooth.sd.attr.retrievable.max")+"\n");
p.append("No. max de conexiones: "+
      ld.getProperty("bluetooth.connected.devices.max")+"\n");
p.append("Tamaño de paquetes en bytes(L2CAP): "+
      ld.getProperty("bluetooth.l2cap.receiveMTU.max")+"\n");
p.append("No. max de servicios encontrados: "+
      ld.getProperty("bluetooth.sd.trans.max")+"\n");
p.append("Responder inquery scan: "+
      ld.getProperty("bluetooth.connected.inquiry.scan")+"\n");
p.append("Responder page scan: "+
      ld.getProperty("bluetooth.connected.page.scan")+"\n");
p.append("Responder inquery: "+
      ld.getProperty("bluetooth.connected.inquiry")+"\n");
p.append("Responder page: "+
      ld.getProperty("bluetooth.connected.page")+"\n");
```

[12] HOPKINS, Bruce et al. "Bluetooth For Java" Capítulo 4 p.5.

La siguiente tabla muestra los valores que puede recuperar el método getProperty():

PROPIEDAD	DESCRIPCIÓN	VALOR VÁLIDO
bluetooth.api.version	La versión del API que es compatible, esta propiedad no se refiere al número de especificación Bluetooth.	Depende de la versión del API
bluetooth.master.switch	¿Está permitido el cambio maestro/esclavo?.	Verdadero ó Falso
bluetooth.sd.attr.retrievable.max	Número máximo de atributos de servicio por cada service record.	Entero base 10
bluetooth.connected.devices.max	Número máximo de dispositivos conectados que puede soportar.	Entero base 10
bluetooth.l2cap.receiveMTU.max	Tamaño máximo en bytes del paquete receiveMTU en el protocolo L2CAP.	Entero base 10
bluetooth.sd.trans.max	Número máximo de servicios que se pueden descubrir por transacción.	Entero base 10
bluetooth.connected.inquiry.scan	¿Puéde el dispositivo local responder a una petición "inquery" mientras el dispositivo tiene establecido un vinculo a otro dispositivo?.	Verdadero ó Falso
bluetooth.connected.page.scan	¿Puéde el dispositivo local aceptar una conexión de un dispositivo remoto si este ya esta conectado a otro dispositivo remoto?.	Verdadero ó Falso
bluetooth.connected.inquiry	¿Puéde el dispositivo local iniciar un "inquery" mientras esta conectado a otro dispositivo?.	Verdadero ó Falso
bluetooth.connected.page	¿Puéde el dispositivo local establecer una conexión con un dispositivo remoto si el dispositivo local ya se encuentra conectado con otro dispositivo?.	Verdadero ó Falso

Tabla 3.6

Propiedades disponibles para getProperty()[13]

Finalmente las últimas tres propiedades indican el tipo de dispositivo del que se trata y si presta algún servicio, estas propiedades se obtienen a través del objeto DeviceClass invocando a los métodos: getMajorDeviceClass(), getMinorDeviceClass(), getServiceClasses().

```
p.append("Tipo de dispositivo (Major Class): "+
      tipo.getMajorDeviceClass()+"\n");
p.append("Tipo de dispositivo (Minor Class): "+
```

[13]THOMPSON J, Timothy et al. "BLUETOOTH APPLICATION PROGRAMMING WITH THE JAVA APIs ESSENTIALS EDITION" p.127.

```
            tipo.getMinorDeviceClass()+"\n");
        p.append("Tipo de dispositivo (Service Class): "+
            tipo.getServiceClasses()+"\n");
        return p;
    }//Fin método obtener propiedades
```

La siguiente tabla muestra algunos de los valores más comunes que se pueden obtener de los métodos getMajorDeviceClass(), getMinorDeviceClass():

MAJOR CLASS (valor entero)	MINOR CLASS (valor entero)	DESCRIPCION MAJOR CLASS	DESCRIPCION MINOR CLASS
0	----------	Varios	----------
256	0	Computadora	Sin asignar, varios
256	4	Computadora	Escritorio
256	8	Computadora	Servidor
256	12	Computadora	Laptop
256	16	Computadora	Sub-laptop
256	20	Computadora	PDA
256	24	Computadora	Watch size
512	0	Teléfono	Sin asignar, varios
512	4	Teléfono	Celular
512	8	Teléfono	Inalámbrico
512	12	Teléfono	Teléfono inteligente (Smart Phone)
512	16	Teléfono modem	----------
768	0	Punto de acceso LAN/network	Totalmente disponible

Tabla 3.7

Descripción Major Class y Minor Class[14]

El próximo método **obtenerHistorial()** se encarga de mostrar los dispositivos Bluetooth con los que se a tenido contacto y el usuario ha establecido como dispositivos ya

[14]HOPKINS, Bruce et al. "Bluetooth For Java" Capítulo 4 p.6

conocidos. De la misma manera que el método anterior retornará un objeto StringBuffer con el resultado de la operación realizada. Se utiliza un objeto LoacalDevice para posteriormente obtener un objeto DiscoveryAgent.

La clase DiscoveryAgent es una de las más importantes, esta clase va a permitir realizar la búsqueda de dispositivos y servicios. Para esta ocasión tan solo se utilizará para obtener un historial de dispositivos Bluetooth ya conocidos. Esto a través del método retrieveDevices() que puede recibir alguno de estos dos parámetros: DiscoveryAgent.PREKNOWN ó DiscoveryAgent.CACHED ambos variables estáticas de la clase DiscoveryAgent.

DiscoveryAgent.PREKNOWN devolverá un arreglo de objetos RemoteDevices que representan los dispositivos remotos especificados por el usuario como ya conocidos, mientras que DiscoveryAgent.CACHED devolverá el arreglo de objetos RemoteDevice de dispositivos que hayan sido descubiertos en búsquedas anteriores.

```
public StringBuffer obtenerHistorial(){
      StringBuffer historial = new StringBuffer();
      LocalDevice ld = null;
      try{
         ld = LocalDevice.getLocalDevice();
      }catch(BluetoothStateException eb){
         System.out.println("ERROR "+eb.getMessage());
      }
      DiscoveryAgent da = ld.getDiscoveryAgent();
      RemoteDevice[] conocidos = da.retrieveDevices(DiscoveryAgent.PREKNOWN);
```

Posteriormente se evalúa con una condición if-else si el arreglo de objetos RemoteDevice a sido llenado, de ser así a través de un ciclo se recorre cada localidad obteniendo la dirección y alias de cada dispositivo remoto almacenado, esto será invocando a los métodos getBluetoothAddress() y getFriendlyName() pero esta vez de la clase RemoteDevice. A diferencia del método getFriendlyName() de la clase LocalDevice, en la clase RemoteDevice este método recibe un argumento de tipo boleano para indicar si debe de forzar (true) o no (false) a contactar al dispositivo remoto para obtener un alias.

```
if(conocidos==null){
         historial.append("no se encontraron registros");
     }else{
         for(int i=0;i<conocidos.length;i++){
            try {
```

```
                String alias = conocidos[i].getFriendlyName(true);
                String direccion = conocidos[i].getBluetoothAddress();
                String resultado = alias+"("+direccion+")";
                historial.append("\n");
                historial.append(resultado);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
    return historial;
  }//Fin Metodo obtenerHistorial()
}//Fin Clase
```

Las siguientes imágenes muestran el MIDlet "PropiedadesBT" en ejecución:

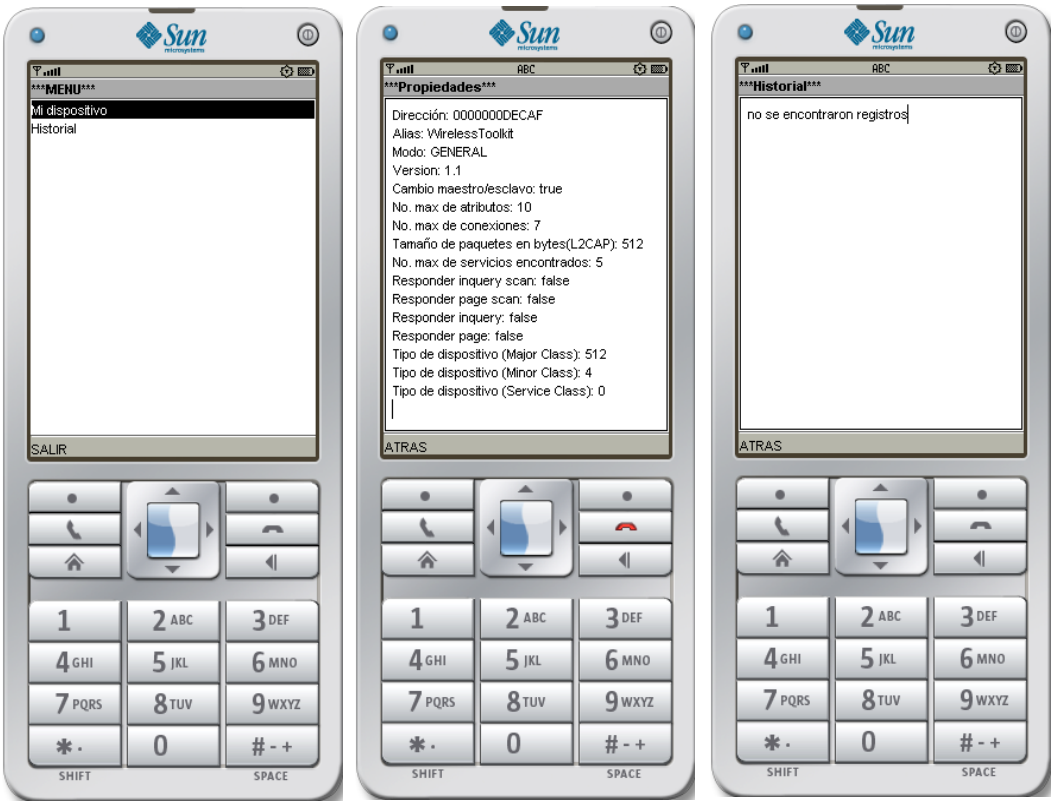


Figura 3.2

MIDLET PropiedadesBT en ejecución

Para el próximo ejemplo se realizarán operaciones más detalladas sobre la programación de un cliente y un servidor para establecer una comunicación entre dos dispositivos Bluetooth.

Ejemplo 2: Programa "BTCHATme" Creación de clientes y servicios Bluetooth utilizando el protocolo SSP

Esta aplicación permitirá enviar mensajes de texto por Bluetooth de un dispositivo a otro, simulando tener un chat. El MIDlet permite establecer una conexión como cliente ó servidor por lo que en este ejemplo se va a describir la manera de programar un cliente y un servidor bajo el protocolo SPP (Serial Port Profile).

La aplicación consta de tres clases principales **BTCHATme**, **BTClienteCHAT** y **BTServidorCHAT**.

BTCHATme es la clase principal que hereda de MIDlet y se encarga de mostrar y realizar las operaciones adecuadas en pantalla. Primeramente se declaran las variables que se van a utilizar.

```
package ejemplo_2;
import java.io.IOException;
import java.util.Vector;
import javax.bluetooth.RemoteDevice;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
/**
 * @author JAVAXIAN
 */
public class BTCHATme extends MIDlet implements CommandListener{
    Display pantalla;
    List menu,dispositivos;
    Command salir,conectar,select,atras,cmd_ok,cmd_nick,cancelar;
    TextBox chat;
    Form form;
    Image img;
    BTClienteCHAT cliente;
    BTServidorCHAT servidor;
    TextField campo;
    Vector vec_dispositivos;
    String nick = ": ";
```

Los objetos **cliente** y **servidor** son los que van a permitir iniciar un cliente o un servidor Bluetooth en el MIDlet. En el objeto de tipo Vector se van añadir los dispositivos remotos que sean descubiertos durante el proceso de búsqueda.

El método constructor inicializará las variables necesarias para mostrar el menú principal que permitirá iniciar un cliente ("Buscar CHAT") ó servidor ("Crear CHAT") Bluetooth, la

última de las opciones permite establecer un nick ó apodo que identificará a los usuarios durante la conversación.

```
public BTCHATme(){
        pantalla = Display.getDisplay(this);
        salir = new Command("Salir",Command.EXIT,1);
        atras = new Command("Atras",Command.BACK,1);
        cancelar = new Command("Cancelar",Command.CANCEL,1);
        select = new Command("Seleccionar",Command.SCREEN,1);
        menu = new List("***M E N U***",List.IMPLICIT);
        menu.setSelectCommand(select);
        menu.addCommand(salir);
        menu.append("Buscar CHAT",null);
        menu.append("Crear CHAT",null);
        menu.append("Crear Nick",null);
        menu.setCommandListener(this);
    }
```

El método **obtenerDispositivos()** se encarga de desplegar en pantalla la lista de todos los dispositivos remotos encontrados durante una búsqueda y que han sido añadidos en un objeto tipo Vector. Esta lista desplegada permite seleccionar el dispositivo con el que se desea establecer una conexión.

```
 //Método que muestra en pantalla la lista de dispositivos encontrados
    public void obtenerDispositivos(){
        try{//Imagen creada para cada elemento de la lista
            img = Image.createImage("/ejemplo_2/celular_2.png");
            System.out.println("ENTRA IMAGEN");
        }catch(IOException ioE){
            System.out.println("ERROR: "+ioE.getMessage());
        }
        dispositivos = new List("Dispositivos Encontrados",List.IMPLICIT);
        //Se recuperan valores del vector objetos RemoteDevice
        for(int i=0;i<vec_dispositivos.size();i++){
            RemoteDevice remoto = (RemoteDevice) vec_dispositivos.elementAt(i);
            String alias="";
            String direccion="";
            try {
                direccion = remoto.getBluetoothAddress();
                alias = remoto.getFriendlyName(true);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
            //Se añade cada dispositivo por dirección y alias
            dispositivos.append(alias+"("+direccion+")",img);
            System.out.println(alias+"("+direccion+")");
        }
        conectar = new Command("Conectar",Command.SCREEN,0);
        dispositivos.setSelectCommand(conectar);
        dispositivos.addCommand(atras);
        dispositivos.setCommandListener(this);
        pantalla.setCurrent(dispositivos);
    }
```

El siguiente método **mostrarChat()** va a desplegar en pantalla los mensajes que se han enviado durante la conversación, recibe como parámetro la cadena String que contiene los mensajes de cada usuario.

```
//Método encargado de mostrar la conversacion en un TextBox
   public void mostrarChat(String mensaje){
      if(mensaje!=null){
         chat = new TextBox("***CHAT***",mensaje,500,TextField.UNEDITABLE);
      }else{
         chat = new TextBox("***CHAT***","",500,TextField.UNEDITABLE);
      }
      cmd_ok = new Command("Escribir",Command.OK,1);
      chat.addCommand(cmd_ok);
      chat.addCommand(atras);
      chat.setCommandListener(this);
      pantalla.setCurrent(chat);
   }
```

La última parte del código de la clase BTCHATme son los métodos implementados de la clase abstracta MIDlet y la interfaz CommandListener que se encargará de llevar el manejo de eventos producidos al operar como un cliente ó servidor Bluetooth.

```
 //Estados del MIDlet
   public void startApp() {
      pantalla.setCurrent(menu);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {

      if(servidor!=null){
            servidor.terminarServidor();
            servidor.cerrarComunicacion();
         }else if(cliente!=null){
            cliente.terminarCliente();
            cliente.cerrarComunicacion();
         }else{}
   }
   //Método de la interfaz CommandListener
   public void commandAction(Command cmd, Displayable dis) {
      if(cmd==salir){//Comando Salir
         destroyApp(true);
         notifyDestroyed();

      }else if(cmd==atras){//Comando atras
         if(dis==dispositivos){
            pantalla.setCurrent(menu);
         }else if(dis==chat){
            if(servidor==null){
               System.out.println("ENTRA CERRAR CLIENTE");
               cliente.terminarCliente();
```

```
                cliente.cerrarComunicacion();
                cliente = null;
                pantalla.setCurrent(menu);
            }else{
                System.out.println("ENTRA CERRAR SERVIDOR");
                servidor.terminarServidor();
                servidor.cerrarComunicacion();
                servidor = null;
                pantalla.setCurrent(menu);
            }
        }else if(dis==form){
            pantalla.setCurrent(menu);
        }
    }else if(cmd==select){//Comando seleccionar
        switch(menu.getSelectedIndex()){
            //Iniciar cliente Bluetooth
            case 0: cliente = new BTClienteCHAT(this,nick);
                    cliente.buscarDispositivos();
                    form = new Form("Iniciando...");
                    form.addCommand(cancelar);
                    form.setCommandListener(this);
                    form.append(new Gauge("Buscando Dispositivos...",false,
                            Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING));
                    pantalla.setCurrent(form);
            break;
            //Iniciar servidor Bluetooth
            case 1: servidor = new BTServidorCHAT(this,nick);
            break;
            //Establecer nick
            case 2: form = new Form("Ingresar NICK");
                    campo = new TextField("nick: ","",10,TextField.ANY);
                    form.append(campo);
                    cmd_nick = new Command("Guardar",Command.OK,1);
                    form.addCommand(cmd_nick);
                    form.addCommand(salir);
                    form.setCommandListener(this);
                    pantalla.setCurrent(form);
            break;
        }
    }else if(cmd==conectar){//Comando conectar
        RemoteDevice rd =(RemoteDevice)vec_dispositivos.elementAt(
                dispositivos.getSelectedIndex());
        cliente.buscarServicios(rd);
        form = new Form("Conectando...");
        form.addCommand(cancelar);
        form.setCommandListener(this);
        form.append(new Gauge("Buscando Servicio...",false,
                Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING));
        pantalla.setCurrent(form);
    }else if(cmd==cmd_ok){//Comando aceptar
        if(dis==chat){
            form = new Form("ESCRIBIR MENSAJE");
            campo = new TextField("msj: ","",20,TextField.ANY);
            form.append(campo);
            cmd_ok = new Command("Enviar",Command.OK,1);
            form.addCommand(cmd_ok);
            form.addCommand(salir);
            form.setCommandListener(this);
            pantalla.setCurrent(form);
        }else if(dis==form){
```

```
            if(servidor==null){
               cliente.enviar(campo.getString());
            }else{
               servidor.enviar(campo.getString());
            }
         }
      }else if(cmd==cmd_nick){//Comando nick
         nick = campo.getString()+": ";
         pantalla.setCurrent(menu);
      }else if(cmd==cancelar){//Comando cancelar
         if(dis==form){
            if(form.getTitle().equals("Iniciando...")){
               cliente.cancelarBusquedaDispositivos();
               pantalla.setCurrent(menu);
            }else{
               cliente.cancelarBusquedaServicios();
               pantalla.setCurrent(menu);
            }
         }
      }
   }
}//Fin Clase
```

La siguiente clase a describir es **BTClienteCHAT** que es la clase va a realizar las operaciones relacionadas a un cliente Bluetooth como búsqueda de dispositivos, búsqueda de servicios, conexión, comunicación entre otros. Esta clase implementa dos interfaces DiscoveryListener y Runnable. La interfaz DiscoveryListener define los siguientes cuatro métodos: deviceDiscovered(), inquiryCompleted(), servicesDiscovered(), y serviceSearchCompleted(), los dos primeros son invocados al descubrir un dispositivo en una búsqueda y al completar la operación de búsqueda de dispositivos remotos, mientras que los dos restantes se invocan al descubrir un servicio en un dispositivo remoto y al finalizar las operaciones de búsqueda de servicios en dispositivos remotos. La interfaz Runnable define el método run() que iniciará su actividad mediante un objeto Thread.

```
package ejemplo_2;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
```

```
/**
 *
 * @author JAVAXIAN
 */
public class BTClienteCHAT implements DiscoveryListener,Runnable{
```

Se declaran las variables globales que se van a utilizar de las cuales se van a entender más su función al explicar las operaciones que realiza cada método definido.

```
    LocalDevice ld = null;
    DiscoveryAgent da = null;
    ServiceRecord servicio = null;
    StreamConnection conexion = null;
    DataInputStream entrada = null;
    DataOutputStream salida = null;
    BTCHATme midlet;
    String url,nick;
    StringBuffer cadena = new StringBuffer();
    Thread hilo;
    int n_busqueda;
    boolean ciclo;
    public static final UUID ID_SERVICIO = new UUID(0x1380);
    public static final UUID[] SERVICIOS = new UUID[]{ID_SERVICIO};
    public static final int[] ATRIBUTOS = null;
```

El método constructor inicializa los primeros parámetros que se van a utilizar, principalmente se arrancará el hilo de ejecución que permitirá recibir y procesar los datos.

```
    public BTClienteCHAT(BTCHATme midlet_super,String nick_super){
        this.midlet = midlet_super;
        this.nick = nick_super;
        ciclo = true;
        hilo = new Thread(this);
        hilo.start();
    }
```

Utilizando el método **buscarDispositivos()** el MIDlet va a realizar la búsqueda de dispositivos Bluetooth que se encuentren dentro del rango de alcance. Anteriormente se explicó que la clase LocalDevice representa nuestro propio dispositivo Bluetooth, por lo que primeramente se obtiene un objeto LocalDevice, con el cual se establecerá la conectividad en que se encontrará el dispositivo. Para realizar una búsqueda y después una conexión se definirá el tipo de conectividad GIAC a través del método setDiscoverable(), este método recibe como parámetro un número entero que representa el tipo de conectividad, para esto se utiliza la variable estática GIAC de la clase DiscoveryAgent. La clase DiscoveryAgent es la clase que permite realizar las operaciones de búsqueda de

dispositivos y búsqueda de servicios, para el objetivo del método se utiliza un objeto DiscoveryAgent que invoque el método startInquiry(), que recibe dos parámetros primero el número entero que representa el tipo de conectividad y segundo uno objeto DiscoveryListener el cual está implementado en la misma clase y notifíca de los eventos de búsqueda de dispositivos.

```
public void buscarDispositivos(){
    midlet.vec_dispositivos = new Vector();
    try{
        ld = LocalDevice.getLocalDevice();
        ld.setDiscoverable(DiscoveryAgent.GIAC);
        da = ld.getDiscoveryAgent();
        da.startInquiry(DiscoveryAgent.GIAC, this);
    }catch(BluetoothStateException eb){
        Alert alerta = new Alert("Error","No se puede iniciar busqueda de dispositivos. "
              +eb.getMessage(),null,AlertType.ERROR);
        midlet.pantalla.setCurrent(alerta);
    }
}
```

Una vez localizados los dispositivos cercanos, hay que buscar los servicios en los que se esta interesado, para esta aplicación se va a buscar un servicio de chat que esta identificado y definido en la variable global ID_SERVICIO de tipo UUID(Universally Unique Identifier) que es la clase que representa identificadores para los protocolos y servicios, y por medio del constructor se envía la variable de tipo long con el valor (5000) del identificador, en este ejemplo representado en hexadecimal (0x1380). Con el método buscarServicios() el MIDlet realiza la búsqueda de servicios en el dispositivo remoto seleccionado por el usuario, invocando el método searchServices() de la clase DiscoveryAgent se inicia el proceso de búsqueda de servicios, el cual recibe cuatro parámetros, primero un arreglo de enteros que identifica atributos del servicio(ningun atributo definido para este ejemplo), segundo un arreglo de UUID para todos los servicios que se buscarán(uno solo para este ejemplo), tercero un objeto RemoteDevice que representa el dispositivo remoto donde se buscarán servicios, y por último un objeto DiscoveryListener que notificará los eventos de la búsqueda de servicios.

```
public void buscarServicios(RemoteDevice rd){
    try {
        n_busqueda = da.searchServices(ATRIBUTOS, SERVICIOS, rd, this);
    } catch (BluetoothStateException ex) {
        Alert alerta = new Alert("Error","No se puede iniciar busqueda de servicio. "
              +ex.getMessage(),null,AlertType.ERROR);
    }
}
```

Para ambas búsquedas de dispositivos y servicios la clase AgentDiscovery ofrece métodos para cancelar estas operaciones, cancelInquiry() para cancelar búsqueda de dispositivos y cancelServiceSearch() para cancelar búsqueda de servicios. El método cancelInquiry() recibe como parametro un objeto DiscoveryListener encargado de los eventos de búsqueda, mientras que el método cancelServiceSearch() recibe un entero que identifica el número de búsqueda realizado, este valor es retornado por el mismo método que inicia la búsqueda searchServices(). Ambas operaciones para cancelar las realiza el MIDlet a través de los métodos cancelarBusquedaDispositivos() y cancelarBusquedaServicios().

```
public void cancelarBusquedaDispositivos(){
    da.cancelInquiry(this);
}

public void cancelarBusquedaServicios(){
    da.cancelServiceSearch(n_busqueda);
}
```

El método deviceDiscovered() de la interfaz DiscoveryListener es invoncado el producirse un evento en el que se encuentra un dispositivo remoto, recibe como parámetros un objeto RemoteDevice y un objeto DeviceClass. En el MIDlet es añadido un objeto RemoteDevice en un Vector por cada llamada a este método.

```
public void deviceDiscovered(RemoteDevice rd, DeviceClass dc) {
    System.out.println("Dispositivo descubierto");
    midlet.vec_dispositivos.addElement(rd);
}
```

Al finalizarse la búsqueda es invocado el método inquiryCompleted(), que recibe un valor entero que representa el motivo por el cual a finalizado la búsqueda de dispositivos. Por medio de las siguientes variables estáticas de la clase DiscoveryListener se obtiene el motivo de finalización.

- DiscoveryListener.INQUIRY_COMPLETED.- Indica que la búsqueda fue finalizada con exito.

- DiscoveryListener.INQUIRY_TERMINATED.- Indica que la búsqueda ha sido cancelada.

- DiscoveryListener.INQUIRY_ERROR.- Indica que se ha producido un error durante la búsqueda.

```
public void inquiryCompleted(int tipo) {
    Alert alerta;
     switch(tipo){
        case DiscoveryListener.INQUIRY_COMPLETED:
            System.out.println("BUSQUEDA TERMINADA");
            alerta = new Alert("ATENCION","BUSQUEDA TERMINADA",
                    null,AlertType.INFO);
            midlet.pantalla.setCurrent(alerta);
            midlet.obtenerDispositivos();
        break;
        case DiscoveryListener.INQUIRY_TERMINATED:
            System.out.println("BUSQUEDA CANCELADA");
             alerta = new Alert("ATENCION","BUSQUEDA CANCELADA",
                    null,AlertType.INFO);
               midlet.pantalla.setCurrent(alerta);
        break;
        case DiscoveryListener.INQUIRY_ERROR:
             System.out.println("BUSQUEDA ERROR");
             alerta = new Alert("ERROR","BUSQUEDA ERROR",
                    null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
        break;
    }
}
```

El método servicesDiscovered() es similar el método deviceDiscovered() con la diferencia que se invoca al producirse un evento en el que se descubren los servicios en un dispositivo remoto, el método recibe como parámetros un entero que identifica el proceso de búsqueda y un arreglo ServiceRecord que representa los registros de servicios encontrados. El MIDlet tan solo buscará un servicio (servicio de chat) por lo que directamente se almacena el servicio que pudiera encontrarse en una variable global ServiceRecord que posteriormente se utiliza para obtener la url con la que se establecerá la conexión.

```
public void servicesDiscovered(int n_busqueda, ServiceRecord[] sr) {
    for(int i=0;i<sr.length;i++){
        servicio = sr[i];
    }
}
```

También al finalizar la búsqueda de servicios es invocado el método serviceSearchCompleted() que recibe dos parámetros enteros, el primero indica el identificador del proceso de búsqueda y el segundo el motivo de la finalización de la búsqueda de servicios que también son representados por las siguientes variables estáticas de la clase DiscoveryListener.

- DiscoveryListener.SERVICE_SEARCH_COMPLETED.- Indica que la búsqueda de servicios finalizó con normalidad.

- DiscoveryListener.SERVICE_SEARCH_TERMINATED.- Indica que la búsqueda de servicios ha sido cancelada.

- DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE.- Indica que el dispositivo ya no se encuentra accesible.

- DiscoveryListener.SERVICE_SEARCH_NO_RECORDS.- Indica que no se han encontrado registros de servicios.

- DiscoveryListener.SERVICE_SEARCH_ERROR.- Indica que ha ocurrido un error durante la búsqueda de servicios.

```
public void serviceSearchCompleted(int n_busqueda, int resp) {
   Alert alerta;
   switch (resp) {
      case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
         iniciarComunicacion();
         midlet.mostrarChat(nick);
         break;
      case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
         alerta = new Alert("ATENCION","SERVICIO CANCELADO",
                  null,AlertType.INFO);
            midlet.pantalla.setCurrent(alerta);
         break;
      case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
         alerta = new Alert("ERROR","DISPOSITIVO NO ALCANZABLE",
                  null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
         break;
      case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
         alerta = new Alert("ERROR","REGISTROS DE SERVICIO NO ENCONTRADOS",
                  null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
         break;
      case DiscoveryListener.SERVICE_SEARCH_ERROR:
         alerta = new Alert("ERROR","ERROR EN BUSQUEDA DE SERVICIO",
                  null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
         break;
   }
}
```

Después de ser invocado el método serviceSearchCompleted() si el resultado es completado satisfactoriamente (DiscoveryListener.SERVICE_SEARCH_COMPLETED) se realiza la llamada al método iniciarComunicacion() que permitirá establecer una conexión e iniciar una comunicación enviando y recibiendo mensajes en el chat. Primeramente se obtiene la URL necesaria que contiene los datos necesarios para poder realizar la conexión esto se realiza por medio del objeto ServiceRecord que representa el servicio encontrado invocando el método getConnectionURL() que recibe dos parámetros el primero de ellos sirve para especificar si hay que autenticar y encriptar la conexión teniendo las tres siguiente variables estáticas para establecer.

- ServiceRecord.AUTHENTICATE_ENCRYP.- Indica que se requiere autenticación y encriptación.

- ServiceRecord.AUTHENTICATE_NOENCRYPT.- Indica que se requiere autenticación y no encriptación.

- ServiceRecord.NOAUTHENTICATE_NOENCRYPT.- Indica que no se requiere autenticación ni encriptación.

El segundo parámetro es un valor booleano, true para indicar si deseamos que nuestro dispositivo sea maestro ó false si no importa el rol maestro/esclavo que tome nuestro dispositivo.

Una vez obtenida la URL será enviada como parámetro al método estático open() de la clase Connector que devolverá el tipo de objeto que representa la conexión. En este caso para el protocolo SPP que se esta utilizando el valor retornado por el método open() es de tipo StreamConnection con el cual se podrá abrir un flujo de datos de entrada (DataInputStream) y salida (DataOutputStream) esto invocando a los métodos openDataInputStream() y openDataOutputStream() respectivamente.

```
public void iniciarComunicacion(){
      try {
                url = servicio.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
                conexion = (StreamConnection) Connector.open(url);
                entrada = conexion.openDataInputStream();
```

```
                salida = conexion.openDataOutputStream();

            }catch (IOException ex) {
                ex.printStackTrace();
                Alert alerta = new Alert("ERROR","NO SE PUEDE ESTABLECER CONEXION",
                        null,AlertType.ERROR);
                midlet.pantalla.setCurrent(alerta);
            }
    }
```

Por medio del método **enviar()** el MIDlet realiza el envío de mensajes de texto en el chat, estos mensajes son escritos en el flujo de datos a través del objeto DataOutputStream que ofrece varios métodos, para este caso se utiliza el método writeUTF() que recibe como único parámetro una cadena String. Posteriormente se llama al método flush() que hace el volcado de del flujo de datos. Todos los mensajes que se van enviando en el chat se van agregando a un objeto StringBuffer que es utilizado para mostrar la conversación completa en pantalla.

```
public void enviar(String mensaje){
    cadena.append(nick+": "+mensaje+"\n");
    try {
        salida.writeUTF(cadena.toString());
        salida.flush();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    midlet.mostrarChat(cadena.toString());
}
```

Para poder leer los mensajes de textos que son escritos en el flujo de datos se utiliza el objeto DataInputStream que ofrece varios métodos que sirven para leer la forma en que han sido escrito los datos, en este caso se utiliza el método readUTF(). El MIDlet recibe mensajes por medio del método **recibir()** que realiza las operaciones anteriormente descritas, para posteriormente mostrar los mensajes recibidos en pantalla.

```
public void recibir(){
    String mensaje = null;
    try {
        mensaje = entrada.readUTF();
        cadena.delete(0,cadena.length());
        cadena.append(mensaje);
        midlet.mostrarChat(cadena.toString());
    } catch (IOException ex) {
        cerrarComunicacion();
    }
}
```

El método **cerrarComunicacion()** finaliza la conexión y los flujos de datos de entrada y salida. Utilizando cada uno de los tres objetos es invocado el método close() para después asignar valores nulos a los objetos y liberar recursos.

```
public void cerrarComunicacion(){
    try {
        if(entrada!=null){
            entrada.close();
            entrada = null;
        }
        if(salida!=null){
            salida.close();
            salida = null;
        }
        if(conexion!=null){
            conexion.close();
            conexion = null;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Para poder recibir permanentemente los mensajes que son enviados se utiliza el método run() que es invocado en el momento de iniciar el hilo de ejecución (objeto Thread), en el método run es definido de inicio un ciclo while dentro del cual se evalúa si existe una conexión iniciada, de cumplirse esta condición es invocado el método recibir() para leer los mensajes, esta operación se repetirá hasta que se de por terminada la conexión.

```
public void run() {
    while(ciclo){
        if(conexion!=null){
            recibir();
        }
    }
}
```

A través del método **terminarCliente()** se pondrá fin al ciclo while definido dentro del método run(), esta operación cambia el valor booleano que evalúa el ciclo while, lo que permitirá concluir el método run() y finalizar la actividad como cliente del chat.

```
public void terminarCliente(){
    ciclo = false;
}
}//Fin Clase
```

La siguiente clase **BTServidorCHAT** permite al MIDlet crear el servicio de chat en el que un cliente se conectará e iniciará una conversación. Esta clase tan solo implementa la interfaz Runnable.

```
package ejemplo_2;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
/**
 *
 * @author JAVAXIAN
 */
public class BTServidorCHAT implements Runnable{
```

Se definen las variables globales que serán empleadas en los métodos de la clase que van a permitir establecer el servicio de chat y la comunicación con el cliente.

```
   LocalDevice ld = null;
   StreamConnectionNotifier notifier = null;
   StreamConnection conexion = null;
   DataInputStream entrada = null;
   DataOutputStream salida = null;
   public static final UUID ID_SERVICIO = new UUID(0x1388);
   public static final UUID[] SERVICIOS = new UUID[]{ID_SERVICIO};
   Thread hilo = null;
   BTCHATme midlet;
   String nick;
   StringBuffer cadena = new StringBuffer();
   boolean ciclo;
```

El método constructor de igual forma que en la clase **BTClienteCHAT** arranca el hilo de ejecución que primeramente iniciará el servicio de chat, y posteriormente establecerá la conexión con el cliente y procesará los datos recibidos.

```
public BTServidorCHAT(BTCHATme midlet_super,String nick_super){
      this.midlet = midlet_super;
      this.nick = nick_super;
      ciclo = true;
      hilo = new Thread(this);
      hilo.start();
   }
```

La parte principal de esta clase se encuentra dentro del método run(). De inicio se obtiene el objeto LocalDevice y se establece la conectividad en modo GIAC.

```
public void run() {//Inicio método run()
        try {
        ld = LocalDevice.getLocalDevice();
        if (!ld.setDiscoverable(DiscoveryAgent.GIAC)) {
            Alert alerta = new Alert("ERROR","NO SE PUEDE ESTABLECER EL SERVIDOR",
                    null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
        }
```

Para poder crear el servicio es necesario enviar una URL (cadena String) al método open() de la clase Connector. Primero se establece el protocolo con el que se realizarán las operaciones, para este caso ya se ha definido el protocolo SPP ("btspp"), seguido de quien brindará el servicio que es el propio MIDlet ("localhost") y el identificador del servicio (valor del objeto UUID), posteriormente pueden definirse alguno de los parámetros de la siguiente tabla:

NOMBRE	DESCRIPCIÓN	VALOR VÁLIDO
Name	Especifíca nombre del servicio	Cadena String
Authorize	Indica si la conexión debe ser autorizada por el servidor	True ó False
Autentícate	Indica si la conexión debe ser auntenticada por el servidor.	True ó False
Encrypt	Indica si la conexión debe ser encriptada por el servidor.	True ó False
Master	Indica si el servidor toma el rol de maestro	True ó False

Tabla 3.8

"Descripcion de parametros para la URL"

```
StringBuffer url = new StringBuffer("btspp://localhost:");
url.append(ID_SERVICIO.toString());
url.append(";name=BTCHATme;authorize=false");
```

Con estos datos en la URL el método open() podrá devolver un objeto StreamConnectionNotifier que permitirá atender las conexiones entrantes de los clientes a través del método acceptAndOpen(). Todo el proceso descrito anteriormente se denomina registro de servicio que al finalizar es agregado al SDDB (Service Discovery Database) del sistema Bluetooth que será descrito con más detalle en la siguiente parte del capítulo.

```
            midlet.mostrarChat(nick);

             while(ciclo){
                if(conexion!=null){
                   recibir();
                }else{
                   conexion = notifier.acceptAndOpen();
                   entrada = conexion.openDataInputStream();
                   salida = conexion.openDataOutputStream();
                }
             }
          } catch (BluetoothStateException ex) {
             ex.printStackTrace();
          } catch(IOException ex){
             ex.printStackTrace();
          }
    }
```

De concretarse satisfactoriamente el registro del servicio, el MIDlet mostrará la pantalla con la cual esperará iniciar una conversación con un cliente. Inicialmente se pude ver que el ciclo while es continuo, al llamar el método acceptAndOpen() el ciclo queda bloqueado, por lo que hasta recibir un cliente continua la ejecución del resto del código. Una vez recibida la conexión del cliente pueden crearce los objetos StreamConnection, DataInputStream y DataOutputStream. Posteriormente el ciclo while continuará ejecutándose para recibir permanentemente los mensajes del cliente.

Una vez teniendo los valores de los objetos StreamConnection, DataInputStream y DataOutputStream, la comunicación se va a realizar de la misma forma que se describió anteriormente para un cliente. Los métodos encargados de la comunicación son **recibir()** y **enviar()**.

```
public void recibir(){
      String mensaje = null;
      try {
         mensaje = entrada.readUTF();
         cadena.delete(0,cadena.length());
         cadena.append(mensaje);
         midlet.mostrarChat(cadena.toString());
      } catch (IOException ex) {
         cerrarComunicacion();
         Alert alerta = new Alert("ERROR","SE HA CERRADO CONEXION AL CHAT S",
                  null,AlertType.ERROR);
               ex.printStackTrace();
      }
   }
   public void enviar(String mensaje){
      cadena.append(nick+": "+mensaje+"\n");
      try {
         salida.writeUTF(cadena.toString());
         salida.flush();
```

```
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        midlet.mostrarChat(cadena.toString());
    }
```

El método **cerrarComunicacion()** permitirá finalizar los objetos StreamConnection, DataInputStream y DataOutputStreamque que representan la conexión y los flujos de datos de entrada y salida.

```
public void cerrarComunicacion(){
        try {
            if(entrada!=null){
                entrada.close();
                entrada = null;
            }
            if(salida!=null){
               salida.close();
               salida = null;
            }
            if(conexion!=null){
               conexion.close();
               conexion = null;
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
```

Para finalizar el servicio el MIDlet invoca el método **terminarServicio()** que finaliza el objeto StreamConnectioNotifier que remueve el servicio de registro del SDDB (Service Discovery Database) esto a través del método close() y posteriormente asignando un valor nulo para liberar recursos y de igual manera como se hizo en el cliente se cambia el valor booleano que evalúa el ciclo while lo que permite concluir el método run() y dar por terminada la actividad como servidor chat.

En las siguientes imágenes se muestra parte de la ejecución del ejemplo descrito:

Figura 3.3

MIDlet BTCHATme en ejecución

Aquí termina la descripción sobre la programación de aplicaciones Bluetooth con la especificación JSR – 82 para J2ME. En la siguiente parte del capítulo se tratará la programación de aplicaciones de servicios Bluetooth con la adaptación Bluecove para J2SE.

PARTE 3

PROGRAMACIÓN DE SERVICIOS BLUETOOH EN J2SE UTILIZANDO IMPLEMENTACIÓN BLUECOVE.

Introducción

El principal objetivo de crear servicios Bluetooth a través de una PC, es aprovechar la capacidad y potencialidad que brinda en software y hardware, como se ha visto pese al gran desarrollo que han tenido los dispositivos móviles aún cuentan con restricciones y limitaciones en algunos elementos software y hardware. Para desarrollar aplicaciones Bluetooth en una PC son amplias las herramientas que se pueden encontrar, por fines prácticos se eligió trabajar con la adaptación BlueCove que consta únicamente de un archivo .JAR que nos permite utilizar las clases y paquetes de la especificación JSR – 82 (J2ME) en la versión estándar de Java (J2SE), de esta forma podrá ser aprovechado todo lo estudiado para el desarrollo del caso práctico que permitirá trabajar de forma interactiva una PC con un dispositivo móvil (Teléfono Celular) vía Bluetooth.

El archivo .JAR puede ser descargado desde el sitio oficial http://www.bluecove.org. El archivo .JAR únicamente tiene que ser agregado al classpath del proyecto J2SE. Sin embargo hay una serie de limitaciones especificas que van a depender del fabricante del dispositivo Bluetooth, estas especificaciones de igual forma las podemos encontrar en el sitio oficial de Bluecove. El equipo en el que se realizan los ejemplos cuenta con un dispositivo Bluetooth de fabricante Broadcom que cuenta con el soporte necesario a las principales caracteristicas que se utilizarán que son el protocolo SPP y múltiples conexiones simultáneas (hasta 7) permitiendo de esta manera programar servicios Bluetooth empleando J2SE.

Ejemplo 3: Programación de atributos en un registro de servicio con Bluecove.

Anteriormente se describió la manera de realizar el registro de un servicio Bluetooth que es la forma de dar de alta el servicio que se brindará. Cada vez que se realiza el registro de servicio este queda almacenado en el SDDB (Service Discovery Database) que es el repositorio de todos los registros de servicios en el sistema Bluetooth.

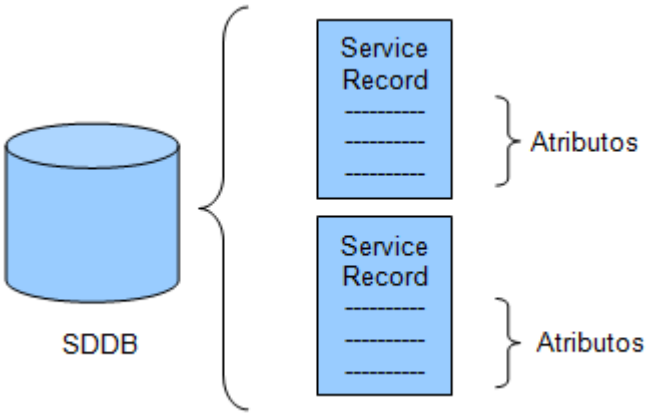


Figura 3.4

Diagrama del SDDB

Cada elemento Service Record define atributos propios que contienen información que permiten describir las características y tipo de servicio que se brinda. El siguiente ejemplo describe como agregar atributos a un Service Record que describirán un servicio hipotético de una librería y los libros de programación que puede contener.

La aplicación consta de dos clases la primera que se va a describir es la clase **PrincipalBT** que extiende de la clase JFrame e implementa la interfaz ActionListener lo que permitirá utilizar una interfaz gráfica y manejo de eventos de la misma.

```
package ejemplo_3;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Enumeration;
import javax.bluetooth.DataElement;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/**
 *
 * @author JAVAXIAN
 */
public class PrincipalBT extends JFrame implements ActionListener {
```

Las variables utilizadas van a permitir crear la ventana, el menú y el área de texto donde se mostrará el resultado final. A través del método constructor se inicializan estas variables.

```
    JMenuBar barraMenu;
    JMenu menu;
```

```
    JMenuItem itemServidor,itemServidor2,itemSalir;
    JTextArea area;
    JScrollPane scroll;
    AtributosServicioBT asbt;

    public PrincipalBT(){
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        this.setSize(400,400);
        area = new JTextArea();
        area.setEditable(false);
        scroll = new
JScrollPane(area,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTA
L_SCROLLBAR_ALWAYS);
        barraMenu = new JMenuBar();
        menu = new JMenu("MENU");
        itemServidor = new JMenuItem("Iniciar Servidor...");
        itemServidor.addActionListener(this);
        itemServidor2 = new JMenuItem("Detener Servidor...");
        itemServidor2.addActionListener(this);
        itemServidor2.setEnabled(false);
        itemSalir = new JMenuItem("Salir");
        itemSalir.addActionListener(this);
        menu.add(itemServidor);
        menu.add(itemServidor2);
        menu.add(itemSalir);
        barraMenu.add(menu);
        this.setJMenuBar(barraMenu);
        this.add(scroll);
    }
```

El método actionPerformed() se define en la interfaz ActionListener y se encarga de procesar los eventos producidos por el menú de la aplicación.

```
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==itemSalir){
            area.setText("SALIR.....");
            System.exit(0);
        }else if(e.getSource()==itemServidor){
                iniciarServidor();
        }else if(e.getSource()==itemServidor2){
                detenerServidor();
        }
    }
```

El método **iniciarServidor()** permite iniciar el servicio junto con sus atributos por medio de la clase **AtributosServicioBT** esperando para aceptar conexiones cliente.

```
    public void iniciarServidor(){
        asbt = new AtributosServicioBT(this);
        asbt.establecerServidor();
        itemServidor2.setEnabled(true);
    }
```

Mientras que el método **detenerServidor()** permite detener la actividad y liberar los recursos ocupados por el servicio Bluetooth iniciado por la clase **AtributosServicoBT**.

```
public void detenerServidor(){
    asbt.terminarServidor();
    itemServidor2.setEnabled(false);
}
```

Con el método **mostrarAtributos()** se visualizan en el cuadro de texto los atributos establecidos al servicio Bluetooth creado. Recibe como parámetro un objeto DataElement que es el tipo de dato que representa los atributos de un servicio.

```
public void mostrarAtributos(DataElement de){
    StringBuffer cadena = new StringBuffer();
    Enumeration enumeration = (Enumeration) de.getValue();
    while(enumeration.hasMoreElements()){
        de = (DataElement) enumeration.nextElement();
        cadena.append(de.getValue().toString()+"\n");
    }
    area.setText(cadena.toString());
}
```

Por último se declara el método main() por el cual la aplicación va a iniciar su ejecución.

```
public static void main(String[] args) {
    PrincipalBT principal = new PrincipalBT();
    principal.setTitle("EJEMPLO 3 ATRIBUTOS DE SERVICIOS");
    principal.setVisible(true);
}
}//Fin Clase
```

La clase encargada de crear y asignar los atributos al servicio es **AtributosServicioBT**. Esta operación se realiza de la misma forma como se ha descrito en ejemplos anteriores utilizando los mismo métodos de inicialización y finalización del servicio.

```
package ejemplo_3;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
```

```
import javax.microedition.io.StreamConnectionNotifier;

/**
 *
 * @author JAVAXIAN
 */
public class AtributosServicioBT implements Runnable{
    LocalDevice ld = null;
    DiscoveryAgent da = null;
    Thread hilo;
    ServiceRecord servicio = null;
    StreamConnectionNotifier notifier = null;
    StreamConnection conexion = null;
    PrincipalBT padre;
    public static final UUID ID_SERVICIO = new UUID(0x1388);
    public static final int LISTA_LIBROS = 0x8831;
    public static final UUID[] SERVICIOS = new UUID[]{ID_SERVICIO};
    public static final int[] ATRIBUTOS = new int[]{LISTA_LIBROS};
    boolean ciclo;

    public AtributosServicioBT(PrincipalBT padre){
        this.padre = padre;
    }

     public void establecerServidor(){
        ciclo = true;
        hilo = new Thread(this);
        hilo.start();
    }

    public void terminarServidor(){
        ciclo = false;
        if(notifier!=null){
            try {
                notifier.close();
                notifier = null;
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
    public void run() {
        try {
            ld = LocalDevice.getLocalDevice();
            if(!ld.setDiscoverable(DiscoveryAgent.GIAC)){
                System.out.println("No se puede iniciar servidor");
            }
            StringBuffer url = new StringBuffer("btspp://localhost:");
            url.append(ID_SERVICIO.toString());
            url.append(";name=BTL2CAPme;authorize=false;");
            notifier = (StreamConnectionNotifier) Connector.open(url.toString());
            servicio = ld.getRecord(notifier);
            /*Codigo empleado para agregar y mostrar los atributos (descrito posteriormente)*/
            while(ciclo){
                conexion = notifier.acceptAndOpen();
            }
            }catch (BluetoothStateException ex) {
                ex.printStackTrace();
            }catch (IOException ex) {
                ex.printStackTrace();
```

```
        }
    }
}//Fin Clase
```

Cada ServiceRecord definido puede tener varios atributos (según lo permita el sistema Bluetooth) estos atributos son representados por objetos DataElement que pueden encapsular datos como: Booleano, Entero, String, UUID y Enumeration (Colleccion de DataElement).

La siguiente tabla muestra las propiedades de los objetos DataElement que pueden encapsularse:

TIPO DE DATO DATAELEMENT	DESCRIPCIÓN	TIPO DE DATO EN JAVA	MÉTODO
DataElement.NULL	Representa un valor nulo.	Null	
U_INT_1, U_INT_2, U_INT_4, INT_1, INT_2, INT_4 e INT_8	Almacena valores enteros de 1,2,4 u 8 bytes con ó sin signo.	Long	getLong()
DataElement.BOOL	Representa un valor booleano.	Bolean	getBoolean()
DataElement.STRING	Almacena una cadena de texto.	java.lang.String	getValue()
DataElement.URL	Almacena una URL.	java.lang.String	getValue()
DataElement.UUID	Almacena un UUID.	javax.bluetooth.UUID	getValue()
DataElement.DATSEQ	Almacena una secuencia de objetos DataElement.	java.util.Enumeration	getValue()
DataElement.DATALT	Almacena una secuencia de objetos alternativos DataElement.	java.util.Enumeration	gatValue

Tabla 3.9

Tipos de datos DataElement[15]

Para saber que tipo de dato se recupera, se puede invocar el método getDataType() que retorna un entero que representa alguno de los tipos de datos DataElement mostrados en la tabla anterior. El siguiente fragmento de código es el que completa la clase **AtributosServicioBT** en el que establece atributos al servicio creado para posteriormente ser mostrados en un cuadro de texto.

```
/*Codigo empleado para agregar y mostrar los atributos*/
        servicio = ld.getRecord(notifier);
```

[15]GIMENO BRIEBA, Alberto. "JSR-82: Bluetooth desde Java" p. 11.

```
DataElement lista = new DataElement(DataElement.DATSEQ);
lista.insertElementAt((new DataElement(DataElement.STRING,"J2ME")),0);
lista.insertElementAt((new DataElement(DataElement.STRING,"J2SE")),1);
lista.insertElementAt((new DataElement(DataElement.STRING,"J2EE")),2);
lista.insertElementAt((new DataElement(DataElement.STRING,"C++")),3);
lista.insertElementAt((new DataElement(DataElement.STRING,"C#")),4);
servicio.setAttributeValue(LISTA_LIBROS, lista);
padre.mostrarAtributos(lista);
```

Enviando como parámetro el objeto StreamConnectionNotifier al método getRecord() de la clase LocalDevice se obtiene el objeto ServiceRecord con el que podremos establecer los atributos de servicio creados, esto invocando el método setAtributeValue() que recibe dos parámetros, primero un valor entero que identifica el atributo y segundo el valor del atributo de tipo DataElement. Existen diferentes métodos constructores para crear objetos DataElement aqui se utilizó un constructor que recibe dos parámetros el primero es el tipo de dato DataElement y el segundo el valor que encapsulará (new DataElement(int valueType, java.lang.Object value)).

Posteriormente para que un dispositivo cliente pueda obtener los atributos de servicio, debe invocar el método getAttributeValue() de la clase ServiceRecord, objeto que se obtiene en la búsqueda de servicios al invocarse el método servicesDiscovered() que notifíca descubrimiento de un servicio. El método getAttributeValue() recibe como parámetro un valor entero que representa el valor que identifica el atributo del servicio (para este ejemplo el valor es 0x8831). Las siguientes imágenes muestran la aplicación en ejecución:

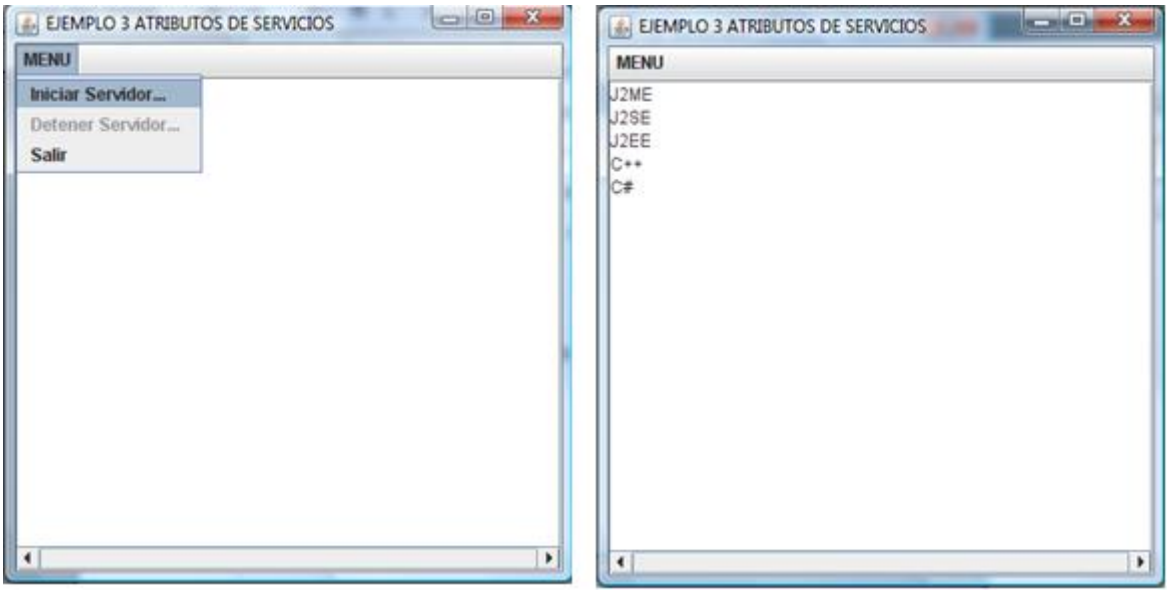


Figura 3.5

Ejecución de programa de Registro de Servicios con Atributos

De esta manera sabiendo los atributos que describen un servicio, el cliente obtendrá información más detallada que indicará si es el servicio solicitado para establecer una conexión.

En el próximo ejemplo se explicará la forma en la que un servicio Bluetooth puede atender múltiples conexiones de clientes.

Ejemplo 4: Programación de un Servidor Bluetooth para múltiples clientes del MIDlet "BTCHATme" utilizando Bluecove.

El objetivo de este ejemplo es programar un servidor Bluetooth que sea capaz de escuchar y atender múltiples conexiones de clientes Bluetooth, para esto se aprovechará el MIDlet "BTCHATme" (ejemplo 2) .Se utilizará el mismo código para obtener propiedades Bluetooth y se va a modificar parte del codigó para establecer un servicio que atienda múltiples conexiones de clientes. Como resultado final se tendrá una aplicación chat en J2SE que permitirá tener conversaciones simultaneas con varios clientes del MIDlet "BTCHATme", por lo que no se entrará mucho en detalle en cuanto a la programación de la interfaz grafica.

La clase principal es **PrincipalBT** y tiene un funcionamiento similar al del ejemplo anterior, brindando un menú con opciones de iniciar servidor, detener servidor y obtener las propiedades Bluetooth del servidor, mismas que nos indicarán el número máximo de conexiones clientes que puede atender.

```
package ejemplo_4;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
/**
 *
 * @author JAVAXIAN
 */
public class PrincipalBT extends JFrame implements ActionListener{

    JMenuBar barraMenu;
    JMenu menu;
```

```
    JMenuItem itemCliente,itemServidor,itemServidor2,itemPropiedades,itemSalir;
    JTextArea area;
    JScrollPane scroll;
    ServidorBT servidor=null;

    public PrincipalBT(){
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        this.setSize(400,400);
        area = new JTextArea();
        area.setEditable(false);
        scroll = new
JScrollPane(area,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTA
L_SCROLLBAR_ALWAYS);
        barraMenu = new JMenuBar();
        menu = new JMenu("MENU");
        itemCliente = new JMenuItem("Iniciar Cliente...");
        itemCliente.addActionListener(this);
        itemServidor = new JMenuItem("Iniciar Servidor...");
        itemServidor.addActionListener(this);
        itemServidor2 = new JMenuItem("Detener Servidor...");
        itemServidor2.addActionListener(this);
        itemServidor2.setEnabled(false);
        itemPropiedades = new JMenuItem("Propiedades");
        itemPropiedades.addActionListener(this);
        itemSalir = new JMenuItem("Salir");
        itemSalir.addActionListener(this);
        //menu.add(itemCliente);
        menu.add(itemServidor);
        menu.add(itemServidor2);
        menu.add(itemPropiedades);
        menu.add(itemSalir);
        barraMenu.add(menu);
        this.setJMenuBar(barraMenu);
        this.add(scroll);
    }

    public void actionPerformed(ActionEvent e) {

        if(e.getSource()==itemSalir){
            area.setText("SALIR.....");
            System.exit(0);
        }else if(e.getSource()==itemServidor){
                iniciarServidor();
        }else if(e.getSource()==itemPropiedades){
                verificarPropiedades();
        }else if(e.getSource()==itemServidor2){
                detenerServidor();
        }
    }

    public void iniciarServidor(){
        servidor = new ServidorBT(this);
        servidor.establecerServidor();
        itemServidor2.setEnabled(true);
    }

    public void detenerServidor(){
        servidor.terminarServidor();
    }
```

```
    public void verificarPropiedades(){
        PropiedadesBT pbt = new PropiedadesBT();
        StringBuffer propiedades = pbt.obtenerPropiedades();
        area.setText(propiedades.toString());
    }

    public static void main(String[] args) {
        PrincipalBT principal = new PrincipalBT();
        principal.setTitle("EJEMPLO JSR-82 CON BLUECOVE");
        principal.setVisible(true);
    }
}//Fin Clase
```

La clase encargada de iniciar el servicio es **ServidorBT** que ocupa la mayor parte del código visto en el ejemplo 2 de la clase **BTServidorCHAT**, con las principales diferencias de solo usar los métodos **terminarServidor()**, run() y utilizar el método **establecerServidor()** para arrancar el hilo de ejecución.

```
package ejemplo_4;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
/**
 *
 * @author JAVAXIAN
 */
public class ServidorBT implements Runnable{
    LocalDevice ld = null;
    DiscoveryAgent da = null;
    Thread hilo;
    ServiceRecord servicio = null;
    StreamConnectionNotifier notifier = null;
    StreamConnection conexion = null;
    DataInputStream entrada = null;
    DataOutputStream salida = null;
    PrincipalBT padre;
    StringBuffer cadena;
    Vector conexiones = null;
    private int rol=0;
    public static final UUID ID_SERVICIO = new UUID(0x1388);
    public static final UUID[] SERVICIOS = new UUID[]{ID_SERVICIO};
    public static final int[] ATRIBUTOS = null;
    boolean ciclo;

    public ServidorBT(PrincipalBT padre){
        conexiones = new Vector();
```

```
        cadena = new StringBuffer();
        this.padre = padre;
    }
    public void establecerServidor(){
        ciclo = true;
        hilo = new Thread(this);
        hilo.start();
    }
  public void run(){
    try {
                LocalDevice localDevice = LocalDevice.getLocalDevice();
                if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC)) {
                    cadena.append("NO SE PUEDE ESTABLECER EL SERVIDOR");
                    cadena.append("\n");
                    padre.area.setText(cadena.toString());
                }
                StringBuffer url = new StringBuffer("btspp://localhost:");
                url.append(ID_SERVICIO.toString());
                url.append(";name=Servicio chat;authorize=false");
                notifier = (StreamConnectionNotifier) Connector.open(url.toString());
                cadena.append("Servidor iniciado...");
                cadena.append("\n");
                padre.area.setText("Servidor iniciado...");
                cadena.delete(0,cadena.length());
                cadena.append("Recibiendo mensaje.");
                cadena.append("\n");
                padre.area.setText(cadena.toString());
                while(true){
                    conexion = notifier.acceptAndOpen();
                    NvoCliente nvocliente = new NvoCliente(conexion,this);
                }

            } catch (BluetoothStateException ex) {
                ex.printStackTrace();
            } catch(IOException ex){
                ex.printStackTrace();
            }
  }

  public void terminarServidor(){
        ciclo = false;
        if(notifier!=null){
            try {
                notifier.close();
                notifier = null;
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

En el método run() se define un ciclo while que primeramente es continuo para estar atendiendo las solicitudes de conexión por parte de los clientes. Simplemente se obtiene una conexión cliente y junto con la referencia de la misma clase (**ServidorBT**) se inicializa un objeto de tipo **NvoCliente** que representa un cliente nuevo que ha establecido una conexión.

La clase **NvoCliente** se encarga de manejar la conexión y los flujos de datos de entrada y salida que nos permitirán comunicarse en la conversación de chat. Las conexiones de los clientes son independientes, por lo que cada cliente tendrá su propia conversación con el servidor.

```
package ejemplo_4;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import javax.microedition.io.StreamConnection;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
/**
 *
 * @author JAVAXIAN
 */
public class NvoCliente extends JFrame implements Runnable,ActionListener{
    private StreamConnection conexion = null;
    private DataInputStream entrada = null;
    private DataOutputStream salida = null;
    ServidorBT serv;
    Thread hilo_c;
    JTextArea area;
    JScrollPane scroll;
    JTextField texto;
    JButton boton;
    StringBuffer cadena = new StringBuffer();
    String nick = "Servidor";

    public NvoCliente(StreamConnection conexion, ServidorBT serv) {
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        this.setLayout(null);
        this.setSize(400,400);
        boton = new JButton("ENVIAR");
        boton.setBounds(280,320,100,20);
        boton.addActionListener(this);
        area = new JTextArea();
        area.setEditable(false);
        scroll = new
JScrollPane(area,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTA
L_SCROLLBAR_ALWAYS);
        scroll.setBounds(10,10,380,300);
        texto = new JTextField();
        texto.setBounds(10,320, 250,20);
        this.add(scroll);
        this.add(texto);
        this.add(boton);
        this.setTitle("Conversacion");
        this.setVisible(true);

        hilo_c = new Thread(this);
        hilo_c.start();
```

```
        try {
            this.conexion = conexion;
            this.serv = serv;
            entrada = conexion.openDataInputStream();
            salida = conexion.openDataOutputStream();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    public void run() {
        while(true){
            recibir();
        }
    }
    public StreamConnection getConexion() {
        return conexion;
    }
    public void setConexion(StreamConnection conexion) {
        this.conexion = conexion;
    }
    public DataInputStream getEntrada() {
        return entrada;
    }
    public void setEntrada(DataInputStream entrada) {
        this.entrada = entrada;
    }
    public DataOutputStream getSalida() {
        return salida;
    }
    public void setSalida(DataOutputStream salida) {
        this.salida = salida;
    }
    public void cerrarComunicacion(){
         try {
            if(entrada!=null){
                entrada.close();
                entrada = null;
            }
            if(salida!=null){
               salida.close();
               salida = null;
            }
            if(conexion!=null){
               conexion.close();
               conexion = null;
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    public void recibir(){
        String mensaje = null;
        if(conexion!=null){
                try {
                    mensaje = entrada.readUTF();
                    cadena.delete(0,cadena.length());
                    cadena.append(mensaje);
                    area.setText(mensaje);
                } catch (IOException ex) {
```

```
                ex.printStackTrace();
                cerrarComunicacion();
            }
        }

    }
    public void enviar(String mensaje){
        try {
            cadena.append(nick+": "+mensaje+"\n");
            salida.writeUTF(cadena.toString());
            salida.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        area.setText(cadena.toString());
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==boton){
            String cadena = texto.getText();
            enviar(cadena);
        }
    }
}
```

Como puede mostrar el código la comunicación es manejada de la misma forma en que se describió en el ejemplo 2, por lo que no se ve necesario explicar el resto de los métodos de la clase **NvoCliente**.

En las siguientes imágenes se muestra el servidor en múltiples conversaciones con diferentes clientes.

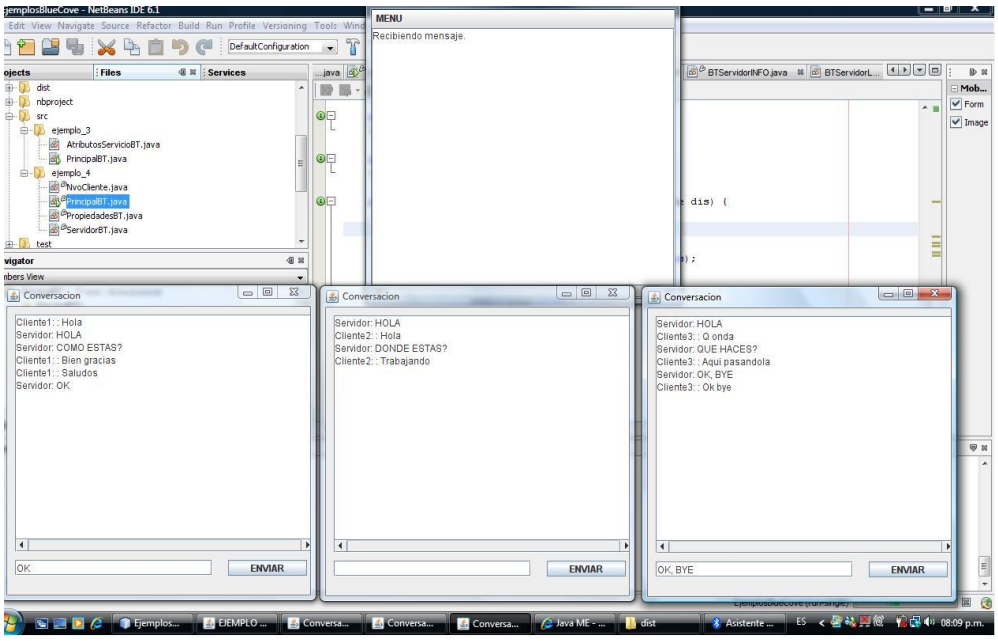


Figura 3.6

Servidor Bluetooth en conversación con múltiples clientes del Midlet "BTCHATme"

Con este ejemplo se da por concluido el capítulo 3 que cumple el objetivo de la programación de clientes y servicios Bluetooth en J2ME y J2SE. En el siguiente capítulo se aplicará de forma práctica todo lo estudiado, por lo que se desarrollará un sistema de conexión interactiva entre una PC y un Móvil para la difusión de actividades escolares vía Bluetooth.

CAPÍTULO 4: DESARROLLO CASO PRÁCTICO, SISTEMA DE DIFUSIÓN DE ACTIVIDADES ESCOLARES VÍA BLUETOOTH

PARTE 1

ANÁLISIS CASO PRÁCTICO

PARTE 2

DISEÑO CASO PRÁCTICO

PARTE 3

IMPLEMENTACIÓN CASO PRÁCTICO

PARTE 1
ANÁLISIS CASO PRÁCTICO.

Planteamiento del problema

La difusión de actividades escolares se puede considerar un sistema informativo dirigido principalmente a la comunidad escolar (alumnos y docentes) que brinda información de interés personal sobre diferentes tipos de actividades realizadas como eventos culturales, deportivos, escolares entre otros.

Actualmente hay varias formas de difundir esta información de interés, típicamente se utiliza la distribución de volantes, boletines, gacetas, además de la colocación de anuncios, todo esto sobre lugares concurridos. Otra forma de difundir la información es por medio de portales Web, que brinda un gran alcance gracias al creciente incremento de usuarios y accesibilidad del Internet.

En conjunto estas dos maneras de difundir la información pueden funcionar de manera adecuada cumpliendo el objetivo principal de difundir la información a la mayor cantidad posible de personas pertenecientes a la comunidad escolar. Sin embargo aún se pueden encontrar algunas limitaciones.

En el primer caso la difusión de información de volantes, boletines y gacetas puede no contar con la suficiente cantidad para distribuir en todo momento a la comunidad escolar y los anuncios no permanecen el tiempo suficiente, ya que estos pueden ser destruidos por distintos factores como el mal clima ó por la misma comunidad escolar, limitando de esta forma una mayor difusión de información, además de generar más desechos ambientales que generan otros problemas.

El segundo caso puede considerarse más estable en la difusión de la información, siempre que el servidor del portal Web este encendido y en actividad la información permanecerá el tiempo suficiente para ser consultada. Sin embargo en un momento determinado puede contar con limitaciones al no tener los recursos necesarios (computadora con acceso a Internet) para consultar dicha información.

Durante los capítulos anteriores se realizó el estudio de la tecnología J2ME en el desarrollo de aplicaciones móviles para conexiones interactivas PC – Dispositivo Móvil a través de la tecnología de conexión inalámbrica Bluetooth. Para este capítulo se va a evaluar en práctica lo estudiado, por lo que se ideó un caso práctico que permite complementar y ampliar la difusión de información escolar, cubriendo en mayor medida las limitaciones mencionadas anteriormente.

El Sistema BTINFOme que se divide en dos aplicaciones cliente y servidor permite principalmente realizar por medio de un teléfono celular una conexión a una PC vía Bluetooth para solicitar y consultar en tiempo real información de interés que haya sido publicada en el sistema. Por parte de la aplicación cliente se podrá buscar los servidores disponibles, consultar la información publicada, almacenarla y borrarla. Mientras que por la aplicación servidor podrá editarse la información, publicarla y atender a los clientes que soliciten la conexión.

El desarrollo del Sistema BTINFOme es muy viable, los teléfonos celulares forman parte de la vida cotidiana de la comunidad escolar y gran cantidad de estos dispositivos móviles brindan soporte para las tecnologías J2ME y Bluetooth junto con especificaciones de ambas, lo que permitirá que la información pueda llegar a mayor número de clientes del sistema de difusión.

Por otro lado son bajos los recursos requeridos por el servidor, emplea la máquina virtual de java que es uno de los lenguajes de programación de mayor uso en la actualidad, utiliza el archivo bluecove.jar distribuido de manera gratuita que corresponde a la especificación JSR -82 implementada en J2SE y por último el dispositivo Bluetooth interno ó externo en la PC que se puede conseguir sin ninguna dificultad.

Para la elaboración del Sistema BTINFOme se utilizarán elementos que componen la Ingeniería de Software: fases, herramientas y procesos de desarrollo con UML, describiendo las principales características de funcionamiento que conforman el Sistema BTINFOme de parte del cliente y de parte del servidor.

Diagrama general de casos de uso del cliente

Este diagrama va a permitir tener una visión general sobre el funcionamiento del Sistema BTINFOme que se va a desarrollar.

El siguiente diagrama representa el funcionamiento por parte del cliente:

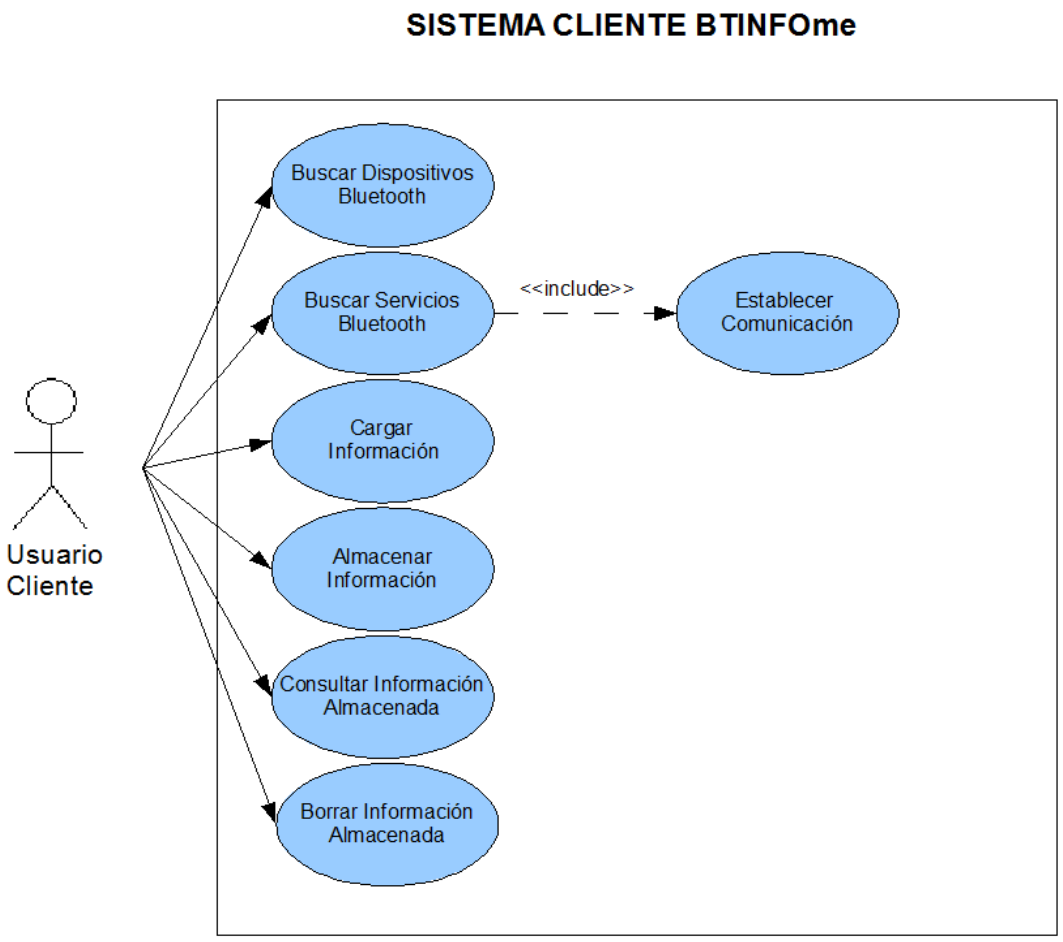


Figura 4.1

Diagrama General de Casos de Uso del cliente

Cada uno de los casos de uso son las diferentes actividades en las que el usuario cliente estará involucrado directamente a través del dispositivo móvil.

Diagrama general de casos de uso del servidor

El próximo diagrama representa el funcionamiento por parte del servidor:

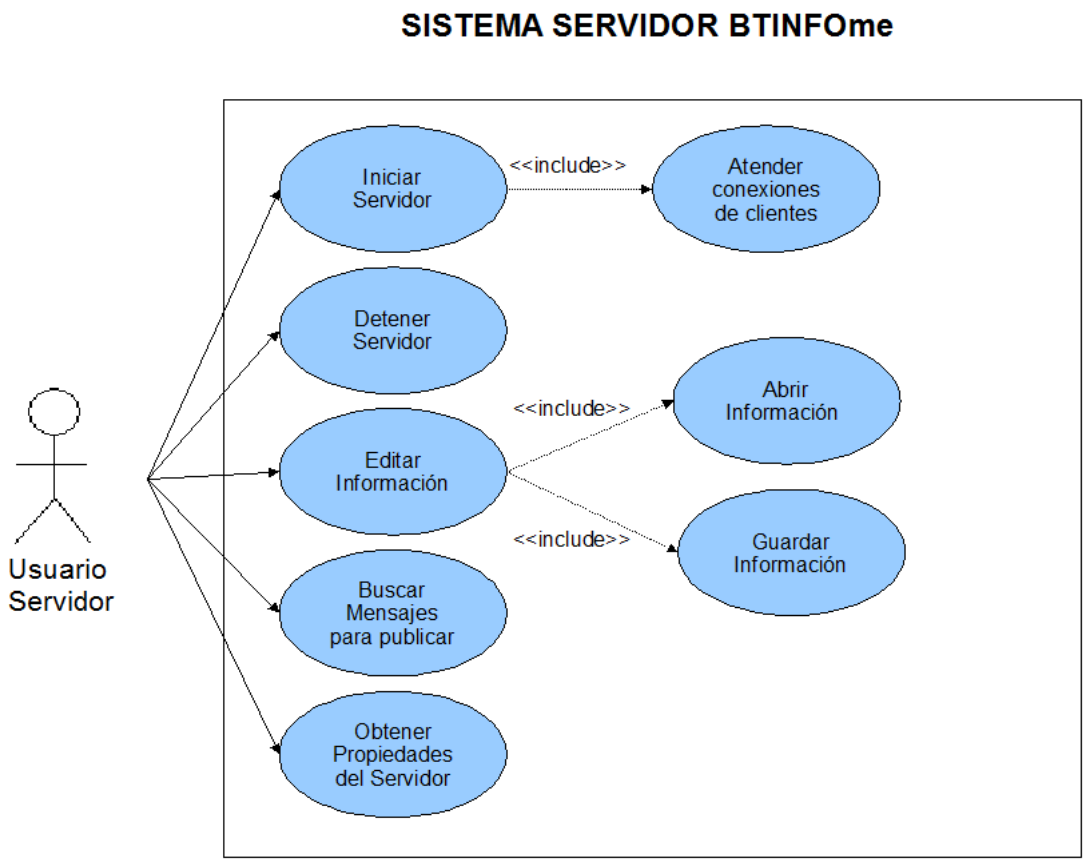


Figura 4.2

Diagrama General de Casos de Uso del servidor

De la misma forma que pasa con el cliente, cada uno de los casos de uso son las actividades en las que el usuario servidor estará involucrado.

A partir de estos diagramas se puede comenzar el diseño en la siguiente parte del capítulo, que va a permitir ir realizando cada una de las actividades presentadas en los casos de uso para la codificación del Sistema BTINFOme cliente y servidor.

PARTE 2

DISEÑO CASO PRÁCTICO.

Diagrama de clases del cliente

Para el diseño del Sistema BTINFOme se eligió trabajar con los diagramas de clases y secuencia de UML, que son de los más importantes y de mayor utilidad en el diseño orientado a objetos. El diagrama de clases va a permitir tener una visión de los componentes que integran el sistema y la manera en que interactúan en un modelo estático.

El siguiente diagrama corresponde al Sistema BTINFOme cliente:

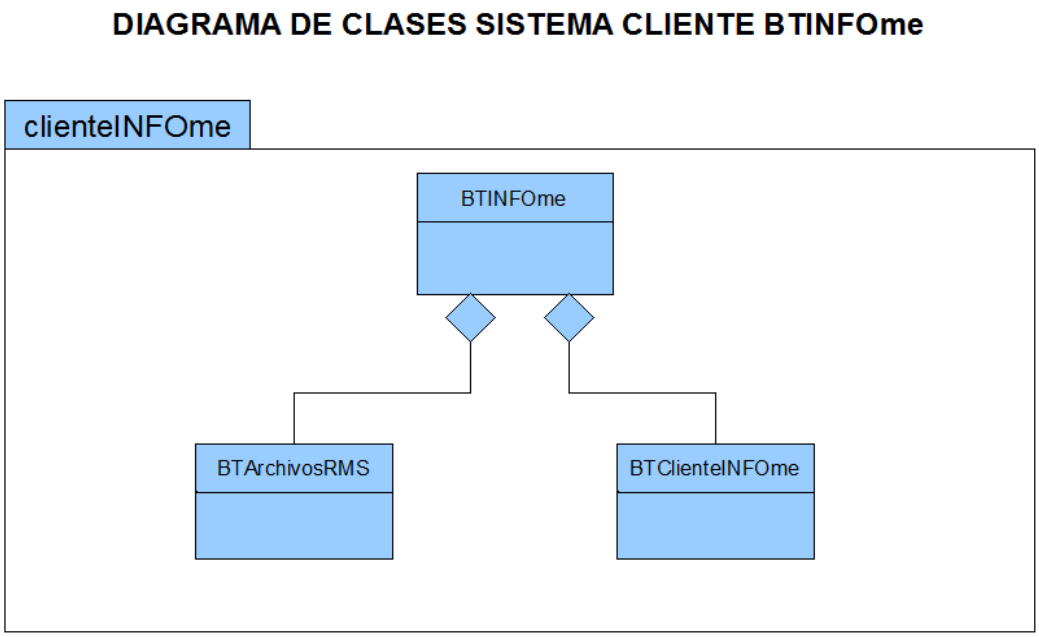


Figura 4.3

Diagrama de Clases del cliente

Tan solo son tres las clases que corresponden a esta parte del sistema, la principal es **BTINFOm**e que es la clase que hereda de MIDlet y se encarga del ciclo de vida del MIDlet, así como del despliegue de pantallas y control de eventos de las mismas.

La clase **BTArchivosRMS** se encarga directamente de las operaciones de registros sobre el ReccordStore creado por el MIDlet. Ofrece operaciones como agregar, eliminar y obtener registros (información publicada), así como crear ó borrar el RecordStore.

La clase **BTCLienteINFOme** se encarga directamente de las operaciones con el dispositivo Bluetooth, a través de esta se pueden buscar dispositivos, buscar servicios, establecer una conexión con el servidor para enviar y recibir información vía Bluetooth.

Diagrama de clases del servidor

El siguiente diagrama corresponde al Sistema BTINFOme servidor:

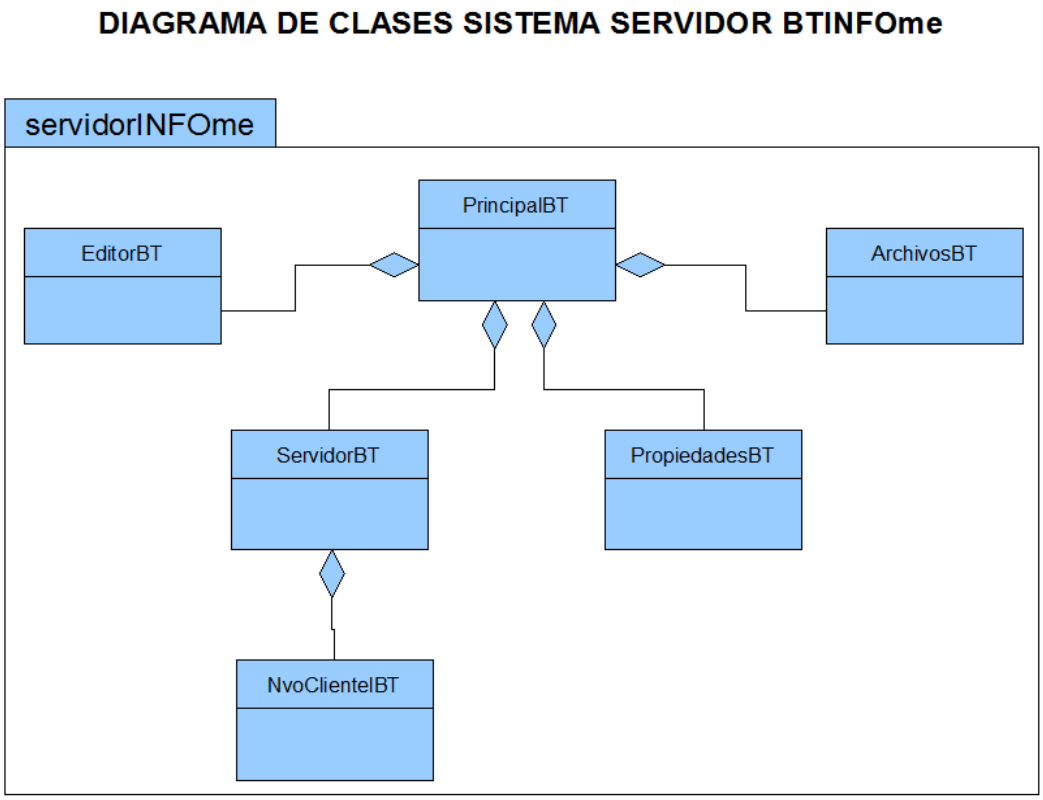


Figura 4.4

Diagrama de Clases del servidor

Para esta parte del sistema son seis las clases que lo componen, siendo la principal de ellas **PrincipalBT** encargada de la interfaz gráfica y el control sobre las otras clases que la componen.

La clase **ServidorBT** se encarga de realizar las operaciones del dispositivo Bluetooth, desde esta clase se establece el servicio Bluetooth y se atienden las conexiones solicitadas por clientes. Esta clase se compone por la clase **NvoClienteBT** que permite interactuar al enviar y recibir información de cada cliente de manera individual.

La clase **PropiedadesBT** simplemente se encarga de obtener las características de trabajo que ofrece el dispositivo Bluetooth instalado en la PC.

La clase **EditorBT** se encarga de brindar un editor de mensajes, el cual puede abrir, editar y guardar archivos que contengan la información que desea ser publicada en el servidor Bluetooth.

Por último la clase **ArchivosBT** va a realizar una búsqueda en el directorio seleccionado por el usuario de archivos (con extensión .bti) que pueden ser publicados, brinda la interfaz para seleccionar alguno de los archivos encontrados.

Diagramas de secuencia y codificación del cliente

Estos diagramas van a permitir visualizar la manera en que interactúan los objetos de clases definidas a través del tiempo en un modelo dinámico realizando las actividades vistas en cada uno de los casos de uso definidos.

La parte cliente del Sistema BTINFOme tiene los siguientes diagramas de secuencia:

Buscar dispositivos
Buscar servicios
Cargar información
Almacenar información
Consultar información almacenada
Borrar información almacenada

Para esta parte solo se van a describir los tres que describen las principales actividades del cliente. Sin embargo el resto de los diagramas de secuencia se podrán consultar en el apéndice A.

Cada uno de los diagramas de secuencia descritos parte de un escenario ideal que se ha establecido, por lo que en la codificación se pueden encontrar segmentos de código que no estén mencionados ó descritos en los diagramas de secuencia.

Iniciando la parte del cliente, y estableciendo un escenario ideal en el que el cliente busca y encuentra dispositivos, se va a comenzar por el primero de los casos de uso "Buscar Dispositivos" del Sistema BTClienteINFOme, del que se obtiene el siguiente diagrama de secuencia.

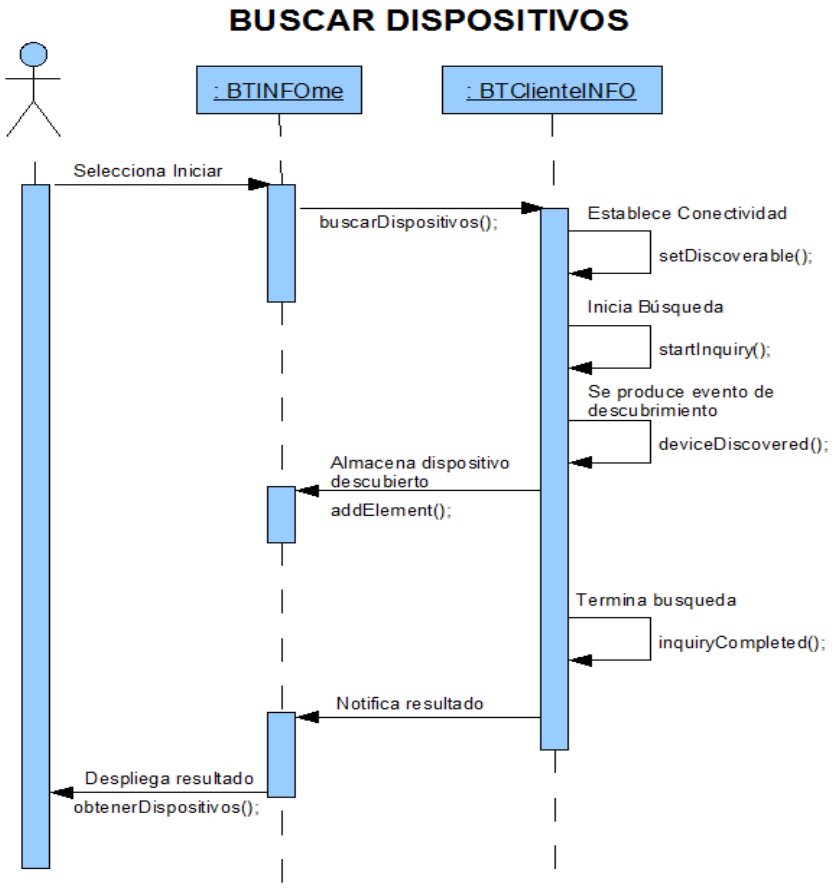


Figura 4.5

Diagrama de secuencia "Buscar Dispositivos"

Partiendo del diagrama se comienza la codificación para cubrir la actividad descrita por el caso de uso.

En la clase **BTINFOme**:

```
//Selección de iniciar búsqueda
  cliente = new BTClienteINFO(this);
  cliente.buscarDispositivos();
  cancelar = new Command("Cancelar",Command.CANCEL,1);
  barra = new Form("Iniciando...");
  barra.addCommand(cancelar);
  barra.setCommandListener(this);
  barra.append(new Gauge("Buscando Dispositivos...",false,
          Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING));
  pantalla.setCurrent(barra);
```

En la clase **BTClienteINFO**:

```
//Método Constructor
public BTClienteINFO(BTINFOme super_midlet){
	this.midlet = super_midlet;
}

public void buscarDispositivos(){
	midlet.vecRemotos = new Vector();
	try {
	   ld = LocalDevice.getLocalDevice();
	   ld.setDiscoverable(DiscoveryAgent.GIAC);
	   da = ld.getDiscoveryAgent();
	   da.startInquiry(DiscoveryAgent.GIAC, this);
	} catch (BluetoothStateException bse) {
	   System.out.println("ERROR: "+bse.getMessage());
	   Alert alerta = new Alert("Error","No se puede iniciar busqueda de dispositivos. "
	         +bse.getMessage(),null,AlertType.ERROR);
	   midlet.pantalla.setCurrent(alerta);
	}
}

//Método de la interfaz DiscoveryListener, notifica descubrimiento de dispositivo
public void deviceDiscovered(RemoteDevice rd, DeviceClass dc) {
	midlet.vecRemotos.addElement(rd);
}

//Metodo de la interfaz DiscoveryListener, notifica el termino de la busqueda de dispositivos
public void inquiryCompleted(int tipo) {
	Alert alerta;
	 switch(tipo){

	   case DiscoveryListener.INQUIRY_COMPLETED:
	      System.out.println("BUSQUEDA TERMINADA");
	      alerta = new Alert("ATENCION","BUSQUEDA TERMINADA",
	            null,AlertType.INFO);
	      midlet.obtenerDispositivos();
	   break;
	   case DiscoveryListener.INQUIRY_TERMINATED:
	      System.out.println("BUSQUEDA CANCELADA");
	       alerta = new Alert("ATENCION","BUSQUEDA CANCELADA",
	            null,AlertType.INFO);
	         midlet.pantalla.setCurrent(alerta,midlet.menu);
	   break;
	   case DiscoveryListener.INQUIRY_ERROR:
	       System.out.println("BUSQUEDA ERROR");
	       alerta = new Alert("ERROR","BUSQUEDA ERROR",
	            null,AlertType.ERROR);
	       midlet.pantalla.setCurrent(alerta,midlet.menu);
	   break;
	}
}
```

En la clase **BTINFOme**:

```
//Método para desplegar lista de dispositivos encontrados
public void obtenerDispositivos(){

	listRemotos = new List("Dispositivos Encontrados",List.IMPLICIT);
	obtenerImagen("dispBTPNG.png");
	//Se recuperan valores del vector objetos RemoteDevice
```

```
        for(int i=0;i<vecRemotos.size();i++){
            RemoteDevice remoto = (RemoteDevice) vecRemotos.elementAt(i);
            String alias="";
            String direccion="";
            try {
                direccion = remoto.getBluetoothAddress();
                alias = remoto.getFriendlyName(true);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
            //Se añade cada dispositivo por dirección y alias
            listRemotos.append(alias+"("+direccion+")",img);
        }
        explorar = new Command("Explorar",Command.SCREEN,1);
        //atras = new Command("Atras",Command.BACK,1);
        listRemotos.addCommand(atras);
        listRemotos.setSelectCommand(explorar);
        listRemotos.setCommandListener(this);
        pantalla.setCurrent(listRemotos);
}
```

Es de esta forma que el cliente desde un dispositivo móvil puede iniciar la búsqueda de dispositivos Bluetooth y visualizar en pantalla aquellos que se encuentren en un rango de alcance. Esto posteriormente le permitirá seleccionar aquel dispositivo que le pueda brindar un servicio de difusión de información, lo que da paso al siguiente diagrama de secuencia.

Para el diagrama de secuencia “Buscar Servicios” se ha establecido el escenario ideal en el que el cliente busca y encuentra servicios para después establecer una conexión con el servidor.

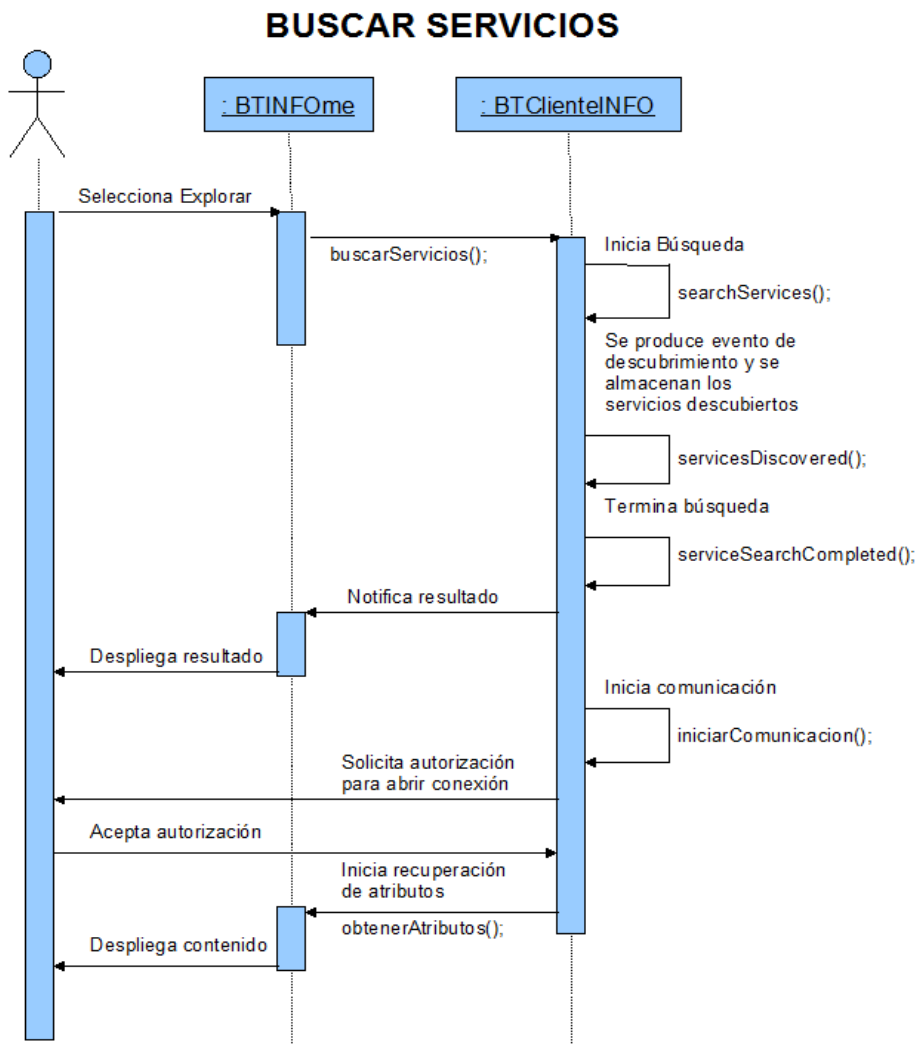


Figura 4.6

Diagrama de Secuencia "Buscar Servicios"

Utilizando el diagrama anterior se inicia la codificación que describe el caso de uso "Buscar Servicios".

En la clase **BTINFOme**:

```
//Selección para iniciar búsqueda de servicios
  RemoteDevice rd =(RemoteDevice)vecRemotos.elementAt(
        listRemotos.getSelectedIndex());
  cliente.buscarServicios(rd);
  form = new Form("Conectando...");
  form.addCommand(cancelar);
  form.setCommandListener(this);
  form.append(new Gauge("Buscando Servicio...",false,
        Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING));
  pantalla.setCurrent(form);
```

En la clase **BTClienteINFO**:

```
//Método que inicia el proceso de búsqueda de servicios
public void buscarServicios(RemoteDevice rd){
	try {
	    n_busqueda = da.searchServices(ATRIBUTOS, SERVICIOS, rd, this);
	    System.out.println("Inicia Busqueda de servicio en..."+rd.getBluetoothAddress());
	} catch (BluetoothStateException bse) {
	     Alert alerta = new Alert("Error","No se puede iniciar busqueda de servicio. "
	          +bse.getMessage(),null,AlertType.ERROR);
	}

}
//Método de la interfaz DiscoveryListener, notifica descubrimiento de servicio
public void servicesDiscovered(int n_busqueda, ServiceRecord[] sr) {
	for(int i=0;i<sr.length;i++){
	    servicio = sr[i];
	}
}

//Método de la interfaz DiscoveryListener, notifica el termino de la busqueda de servicios
public void serviceSearchCompleted(int n_busqueda, int resp) {
	Alert alerta;
	switch (resp) {
	    case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
	        System.out.println("Busqueda completada " + n_busqueda +
	              "con normalidad CONECTANDO....");

	        iniciarComunicacion();
	        midlet.obtenerAtributos();
	        break;
	    case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
	        System.out.println("Busqueda cancelada");
	        alerta = new Alert("ATENCION","SERVICIO CANCELADO",
	                null,AlertType.INFO);
	           midlet.pantalla.setCurrent(alerta,midlet.listRemotos);
	        break;
	    case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
	        System.out.println("Dispositivo no alcanzable");
	        alerta = new Alert("ERROR","DISPOSITIVO NO ALCANZABLE",
	                null,AlertType.ERROR);
	           midlet.pantalla.setCurrent(alerta,midlet.listRemotos);
	        break;
	    case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
	        System.out.println("No se encontraron registros" +
	              " de servicio");
	        alerta = new Alert("ERROR","REGISTROS DE SERVICIO NO ENCONTRADOS",
	                null,AlertType.ERROR);
	           midlet.pantalla.setCurrent(alerta,midlet.listRemotos);
	        break;
	    case DiscoveryListener.SERVICE_SEARCH_ERROR:
	        System.out.println("Error en la busqueda");
	        alerta = new Alert("ERROR","ERROR EN BUSQUEDA DE SERVICIO",
	                null,AlertType.ERROR);
	           midlet.pantalla.setCurrent(alerta,midlet.listRemotos);
	        break;
	}
}

//Método para iniciar la comunicacion con el servidor
public void iniciarComunicacion(){
	try {
```

```
        //String que contiene la url para abrir la conexión
            url =
        servicio.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
            conexion = (StreamConnection) Connector.open(url);
            entrada = conexion.openDataInputStream();
            salida = conexion.openDataOutputStream();

          }catch (IOException ex) {
            ex.printStackTrace();
            Alert alerta = new Alert("ERROR","NO SE PUEDE ESTABLECER CONEXION",
                  null,AlertType.ERROR);
            midlet.pantalla.setCurrent(alerta);
          }
}
```

En la clase **BTINFOme**:

```
//Método que obtiene los atributos del servidor que corresponden a los elementos publicados
public void obtenerAtributos(){
    listElementos = new List("Elementos",List.IMPLICIT);
    obtenerImagen("infPNG.png");//Método que establece imagen para mostrar en lista
    DataElement elemento = cliente.servicio.getAttributeValue(cliente.LISTA_MENSAJES);
    Enumeration enumeration = (Enumeration)elemento.getValue();
    while(enumeration.hasMoreElements()){
      elemento = (DataElement) enumeration.nextElement();
      listElementos.append(elemento.getValue().toString(),img);
    }
    cargar = new Command("Cargar",Command.OK,1);
    listElementos.setSelectCommand(cargar);
    listElementos.addCommand(atras);
    listElementos.setCommandListener(this);
    pantalla.setCurrent(listElementos);

}
```

Una vez establecida la conexión se puede iniciar la comunicación con el servidor, que a través de los atributos de servicio permite al cliente obtener y visualizar la información publicada. Este proceso está establecido en el siguiente diagrama de secuencia "Cargar Información" en el que se ha establecido un escenario ideal en el que el cliente selecciona la opción de su interés y obtiene la información publicada en el servidor.

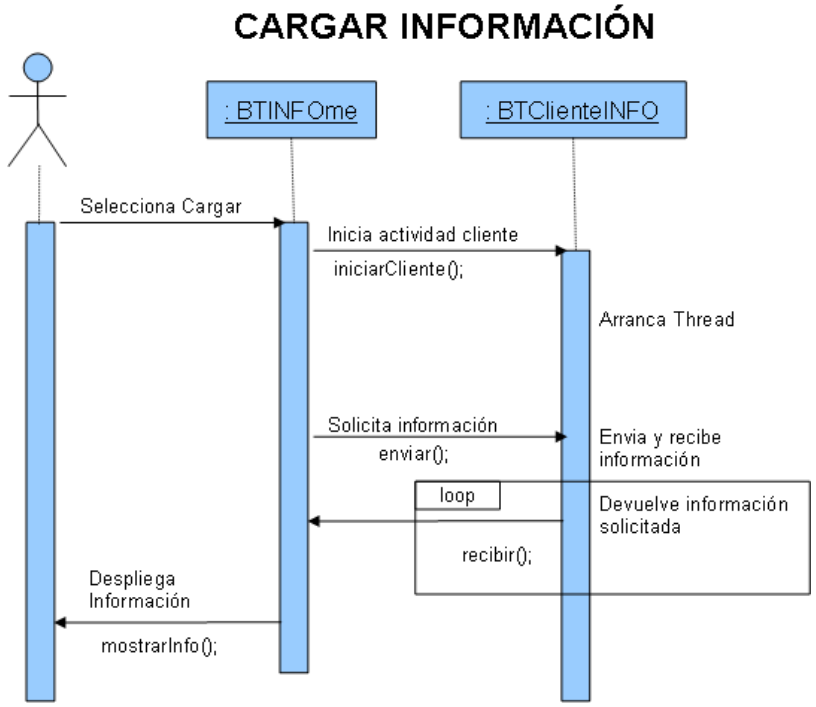


Figura 4.7

Diagrama de Secuencia "Cargar Información"

Una vez establecido el diagrama se implementa el código que va a describir el caso de uso "Cargar Información"

En la clase **BTINFOme**:

```
//Selección para cargar información
   cliente.iniciarCliente();
   barra = new Form("Cargando...");
   barra.addCommand(cancelar);
   barra.setCommandListener(this);
   barra.append(new Gauge("Recuperando Informacion...",false,
        Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING));
   pantalla.setCurrent(barra);
   cliente.enviar(listElementos.getString(listElementos.getSelectedIndex()));
```

En la clase **BTClienteINFO**:

```
public void iniciarCliente(){
        if(!ciclo){
            ciclo = true;
            hilo = new Thread(this);
            hilo.start();
        }
}
//Método que arranca el Thread de ejecucion para recibir datos
public void iniciarCliente(){
        if(!ciclo){
            ciclo = true;
            hilo = new Thread(this);
```

```
            hilo.start();
        }
}
//Método run de la interfaz Runnable
public void run() {
        while(ciclo){
            if(conexion!=null && entrada!=null && salida!=null){
                recibir();
            }
        }
}
//Método que envia datos al servidor
    public void enviar(String opcion){
        try {
            salida.writeUTF(opcion);
            salida.flush();
            enviado = true;

        } catch (IOException ex) {
            ex.printStackTrace();
        }
}
//Método que recibe datos del servidor
    public void recibir(){
        String mensaje = null;
        if(enviado){
            enviado = false;
            byte[] imagen = null;
            try {
                 int tamaño = entrada.readInt();
                 mensaje = entrada.readUTF();
                 System.out.println(tamaño);
                 if(tamaño>0){
                     imagen = new byte[tamaño];
                     entrada.readFully(imagen,0,tamaño);
                 }
                midlet.mostrarInfo(mensaje,imagen);
            } catch (IOException ex) {
                cerrarComunicacion();
                ex.printStackTrace();
            }
        }
}
```

En la clase **BTINFOme**:

```
//Método que despliega información recibida del servidor
public void mostrarInfoArchivos(String mensaje,byte[] imagen){
        Image itemImg = obtenerLogo(imagen);
        String titulo = listArchivos.getString(listArchivos.getSelectedIndex());
        System.out.println("MENSAJE ENTRA");
        System.out.println(mensaje);
        infoArchivos = new Form("INFORMACION ALMACENADA");
        contenido = new StringItem(titulo,mensaje);
        imgLogo = new ImageItem("FES
Aragon",itemImg,Item.LAYOUT_CENTER,"Error",Item.BUTTON);
        infoArchivos.append(imgLogo);
        infoArchivos.append(contenido);
```

```
        infoArchivos.addCommand(atras);
        infoArchivos.setCommandListener(this);
        pantalla.setCurrent(infoArchivos);
}
//Método que inicializa la imagen adjunta a la información
public Image obtenerLogo(byte[] imagen){
        imagenRecibida = imagen;
        System.out.println(imagen);
        Image logo = null;
        if(imagen!=null){
            ByteArrayInputStream bais = new ByteArrayInputStream(imagen);
            try {//imagen recibida del servidor
                logo = Image.createImage(bais);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
        else{
            try{//Imagen valor por default para logo
                logo = Image.createImage("/imagenes/Fes.png");
            }catch(IOException ioE){
                ioE.printStackTrace();
            }
        }
        return logo;
}
```

Todos los diagramas anteriores con su respectiva codificación forman parte del diseño por la parte del cliente del Sistema BTINFOme, principalmente se intenta describir los procesos en los que el cliente interactúa vía Bluetooth con el servidor. A continuación se describen los principales diagramas de secuencia del servidor del Sistema BTINFOme.

Diagramas de secuencia y codificación del servidor

La parte servidor del Sistema BTINFOme tiene los siguientes diagramas de secuencia

- Iniciar servidor
- Detener servidor
- Obtener propiedades del servidor
- Buscar información a publicar
- Editar información

De la misma manera en la que se trabajó con el diseño del cliente, solamente serán descritas las actividades de los tres principales diagramas de secuencia y el resto podrán ser consultados en el apéndice A.

Estos diagramas de secuencia también parten de un escenario ideal, por lo que pueden existir partes de código que no sean descritas en el diagrama de secuencia.

El primero de los diagramas de secuencia es el que describe el caso de uso "Iniciar Servidor", en el cual se maneja un escenario ideal en el que el usuario realiza la publicación de la información que respectivamente ya ha seleccionado.

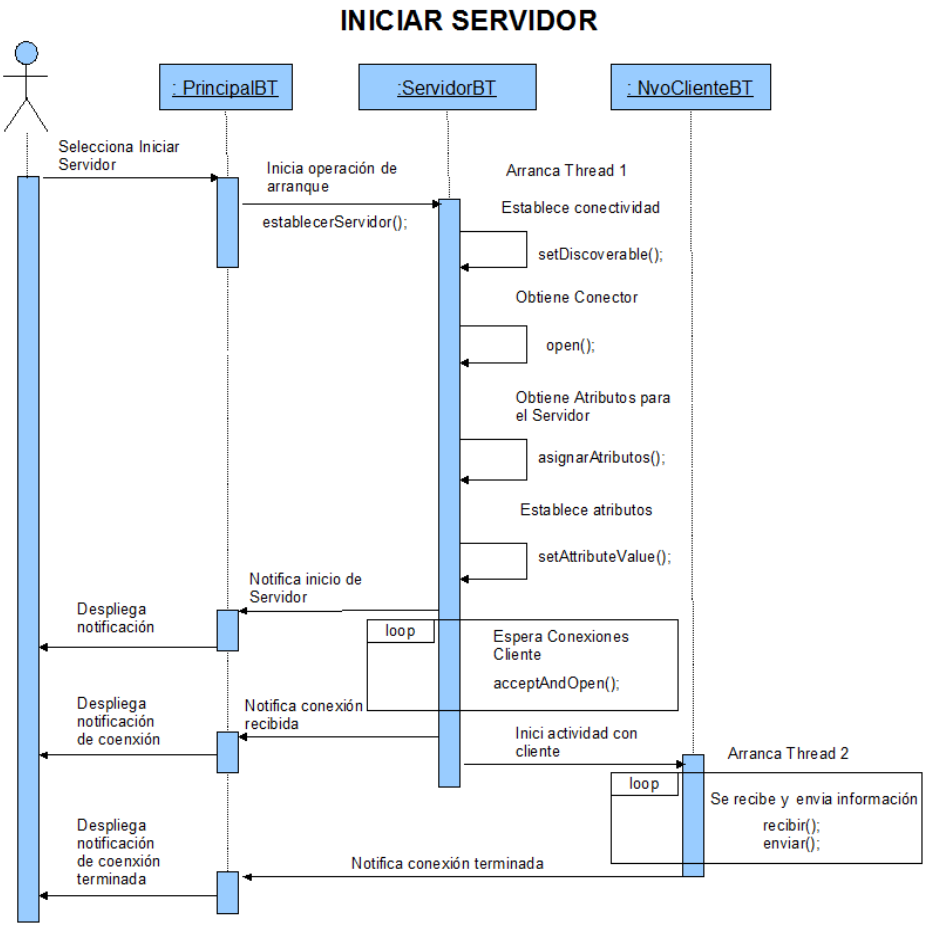


Figura 4.8

Diagrama de Secuencia "Iniciar Servidor"

Ya establecido el diagrama de secuencia "Iniciar Servidor", se comienza la codificación que llevará acabo las actividades establecidas.

En la clase **PrincipalBT**:

```
/Método que da inicio al servidor
public void iniciarServidor(){

    if(mensajesBTI!=null){
       servidor = new ServidorBT(this);
       servidor.establecerServidor();
```

```
            itemServidorD.setEnabled(true);
            itemServidorI.setEnabled(false);
        }else{
            area.append("No hay elementos que publicar"+"\n");
        }
}
```

En la clase **ServidorBT**:

```
//Método Constructor
public ServidorBT(PrincipalBT padre){
        conexiones = new Vector();
        this.padre = padre;
}
//Método que arranca Thread para iniciar servidor y atender conexiones cliente
    public void establecerServidor(){
        ciclo = true;
        hilo = new Thread(this);
        hilo.start();
}
//Método run de la interfaz Runnable
  public void run(){
    try {
                ld = LocalDevice.getLocalDevice();
                if (!ld.setDiscoverable(DiscoveryAgent.GIAC)) {
                    padre.area.append("NO SE PUEDE ESTABLECER EL SERVIDOR"+"\n");
                }
                StringBuffer url = new StringBuffer("btspp://localhost:");
                url.append(ID_SERVICIO.toString());
                url.append(";name=BTINFOme;authorize=false");
                notifier = (StreamConnectionNotifier) Connector.open(url.toString());
                servicio = ld.getRecord(notifier);
                asignarAtributos();
                servicio.setAttributeValue(LISTA_MENSAJES, lista);
                padre.area.append("Servidor iniciado..."+"\n");
                while(ciclo){
                    conexion = notifier.acceptAndOpen();
                    PrincipalBT.clientes_totales++;
                    padre.totales.setText("Clientes Totales: "+
                            String.valueOf(PrincipalBT.clientes_totales));
                    new NvoCliente(conexion,this);
                }
            } catch (BluetoothStateException ex) {
                padre.area.append("Error en el dispositivo Bluetooth");
                ex.printStackTrace();
            } catch(IOException ex){
                ex.printStackTrace();
            }
}
//Método que estable atributos a partir de los archivos seleccionados
public void asignarAtributos(){
    lista = new DataElement(DataElement.DATSEQ);
    int aux = 0;
    for(int i=0; i<padre.mensajesBTI.length;i++){
        if(padre.mensajesBTI[i].isSelected()){
            lista.insertElementAt(new DataElement(
                    DataElement.STRING,padre.mensajesBTI[i].getText()),aux);
            padre.area.append("Publicando: "+padre.mensajesBTI[i].getText()+" "+aux+"\n");
```

```
            aux++;
        }
    }
}
```

En la clase **NvoClienteBT**:

```
//Método Constructor
public NvoCliente(StreamConnection conexion, ServidorBT serv) {
        ciclo = true;
        hilo_c = new Thread(this);
        hilo_c.start();
        try {
            this.conexion = conexion;
            this.serv = serv;
            entrada = conexion.openDataInputStream();
            salida = conexion.openDataOutputStream();
            PrincipalBT.clientes_actuales++;
            serv.padre.actuales.setText("Clientes Actuales: "+
                        String.valueOf(PrincipalBT.clientes_actuales)+" / ");
        } catch (IOException ex) {
            System.out.println("ERROR AQUI");
            ex.printStackTrace();
        }
}
//Método run de la interfaz Runnable
public void run() {
        while (ciclo) {
            if(conexion!=null && entrada!=null && salida!=null){
                recibir();
            }
        }
}
//Método que se encarga de leer los datos recibidos
public void recibir() {
        String opcion = null;
        String info = "";
        byte imagen[] = null;
            try {
                opcion = entrada.readUTF();
                info = cargarInfo(opcion);
                imagen = cargarImagen(opcion);
                enviar(info,imagen);
            } catch (IOException ex) {
                terminarCliente();
                cerrarComunicacion();
                PrincipalBT.clientes_actuales--;
                serv.padre.actuales.setText("Clientes Actuales: "+
                        String.valueOf(PrincipalBT.clientes_actuales)+" / ");
            }
}
//Método que recupera la información solicitada
public String cargarInfo(String opcion) {
        String txt = "";
            try {
                StringWriter w = new StringWriter();
                FileReader archivo = new FileReader(serv.padre.directorio + "/"+
                    opcion);
                int cod=0;
                while ((cod = archivo.read())!= -1) {
```

```
                w.write(cod);
            }
            StringBuffer c = w.getBuffer();
            txt = c.toString();
            w.close();
            archivo.close();
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    return txt;
}
//Método que recupera la imagen adjunta al mensaje
public byte[] cargarImagen(String opcion){
    byte[] imagen = null;
    try {
        String nombreIMG = opcion.substring(0,opcion.lastIndexOf("."));
        BufferedImage biImagen = ImageIO.read(new File(
                serv.padre.directorio + "/"+nombreIMG+".png"));
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ImageIO.write(biImagen, "png", baos);
        imagen = baos.toByteArray();
    } catch (Exception e) {
        e.printStackTrace();
        return imagen;
    }
    return imagen;
}
//Método que envia información al cliente
   public void enviar(String mensaje,byte[] imagen) {
    int tamaño = 0;
    try {
        if(imagen!=null){
            tamaño = imagen.length;
        }
        salida.writeInt(tamaño);
        salida.writeUTF(mensaje);
        if(tamaño>0){
            salida.write(imagen,0,imagen.length);
        }
        salida.flush();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

A través de todo este proceso el usuario puede iniciar el servidor dejándolo disponible para responder cada una de las peticiones que puedan realizar uno ó varios clientes. En cambio en el siguiente diagrama de secuencia “Detener Servidor” se describe la forma en que el servidor detiene toda actividad con el cliente. En el diagrama se estableció el escenario ideal en el que el servidor detiene su actividad sobre los servicios Bluetooth que se hayan iniciado y se cierran las conexiones cliente abiertas.

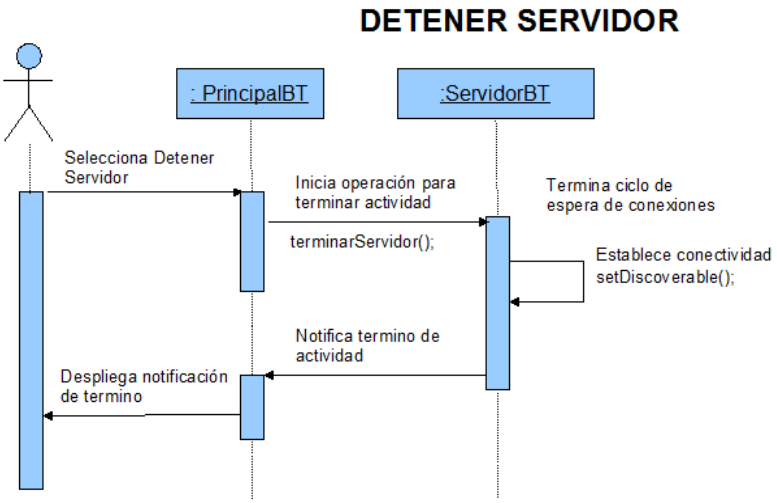


Figura 4.9

Diagrama de Secuencia "Detener Servidor"

A diferencia de la codificación en el diagrama de secuencia anterior, para este diagrama de secuencia es más simple de realizar ya que involucra menos actividad a realizar.

En la clase **PrincipalBT**:

```
//Método que detiene al servidor
public void detenerServidor(){
	servidor.terminarServidor();
	itemServidorD.setEnabled(false);
	itemServidorI.setEnabled(true);
}
```

En la clase **ServidorBT**:

```
public void terminarServidor(){
	ciclo = false;
	for(int i=0;i<conexiones.size();i++){
	    ((NvoCliente) conexiones.elementAt(i)).terminarCliente();
	   ((NvoCliente) conexiones.elementAt(i)).cerrarComunicacion();
	}
	if(notifier!=null){
	   try {
	      notifier.close();
	      notifier = null;
	      ld.setDiscoverable(DiscoveryAgent.NOT_DISCOVERABLE);
	      padre.area.append("Servidor Detenido..."+"\n");
	   } catch (IOException ex) {
	      ex.printStackTrace();
	   }
	}
	conexiones.removeAllElements();
	conexiones = null;
}
```

En la clase **NvoClienteBT**:

```
//Método que finaliza ciclo while del método run
public void terminarCliente(){
	ciclo = false;
}

//Método que termina la comunicación y conexión con el cliente
public void cerrarComunicacion() {
	try {
	    if (entrada != null) {
	        entrada.close();
	        entrada = null;
	    }
	    if (salida != null) {
	        salida.close();
	        salida = null;
	    }
	    if (conexion != null) {
	        conexion.close();
	        conexion = null;
	    }
	} catch (IOException ex) {
	    ex.printStackTrace();
	}
}
```

Hasta ésta parte el usuario del servidor puede detener la actividad del servicio Bluetooth que ofrece y cerrar los vínculos con cada cliente que se encuentre conectado.

Los dos anteriores diagramas de secuencia y su respectiva codificación forman parte del diseño del Sistema BTINFOme del lado del servidor que describen las principales actividades en las que el servidor interactúa vía Bluetooth con el cliente.

El siguiente diagrama de secuencia forma parte de la edición de la información que se publica en el servidor, describe el caso de uso “Editar Información” en el que se establece un escenario ideal en el cual un cliente abre un archivo (de extensión .bti) edita y almacena el contenido que desea difundir para posteriormente adjuntar una imagen ilustrativa sobre el contenido.

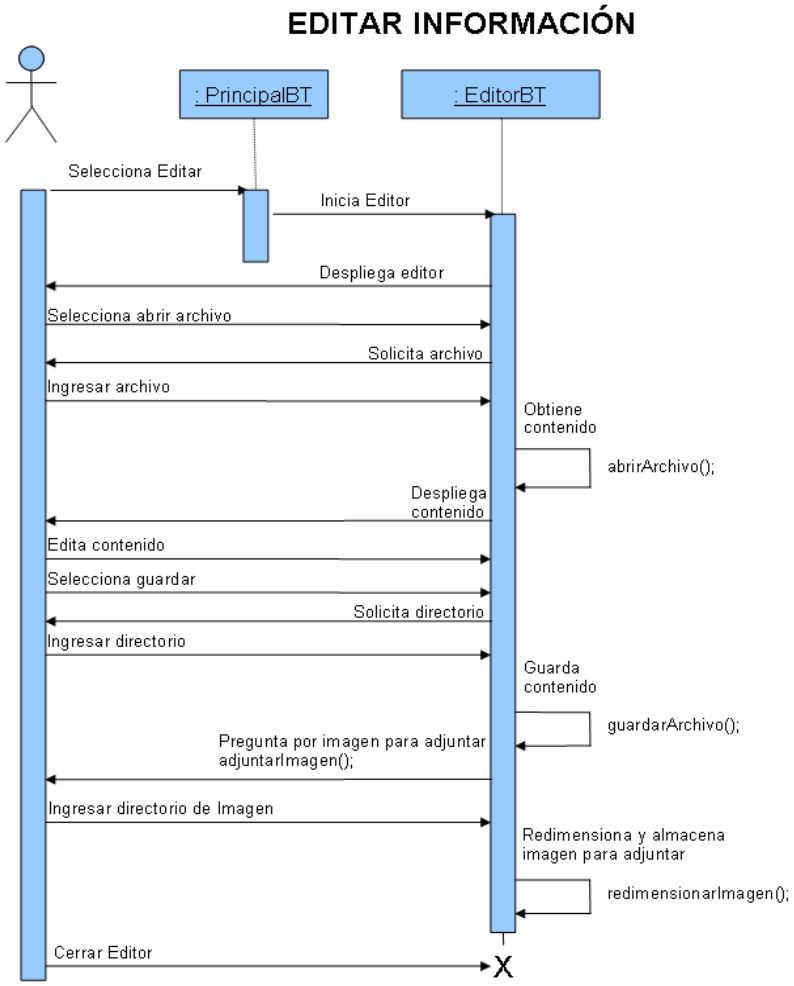


Figura 4.10

Diagrama de Secuencia "Editar Información"

En la clase **PrincipalBT**:

```
//Método que llama al editor
public void abrirEditor(){
	new EditorBT();
}
```

En la clase **EditorBT**:

```
//Método Constructor
public EditorBT(){
	this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
	this.setSize(400,400);
	this.setLocation(160,120);
	this.setTitle("EDITOR DE MENSAJES");
	this.setVisible(true);
	areaMensaje = new JTextArea();
	scroll = new
JScrollPane(areaMensaje,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORI
ZONTAL_SCROLLBAR_ALWAYS);
	barraMenu = new JMenuBar();
	menuMensaje = new JMenu("MENSAJE");
	itemAbrir = new JMenuItem("Abrir...");
```

```
        itemAbrir.addActionListener(this);
        itemGuardar= new JMenuItem("Guardar...");
        itemGuardar.addActionListener(this);
        itemSalir = new JMenuItem("Salir");
        itemSalir.addActionListener(this);
        menuMensaje.add(itemAbrir);
        menuMensaje.add(itemGuardar);
        menuMensaje.add(itemSalir);
        barraMenu.add(menuMensaje);
        this.setJMenuBar(barraMenu);
        this.add(scroll);
}

//Método para abrir archivo (se selecciona abrir en el menu)
public void abrirArchivo(){
        String tipos[] = {"bti"};
        jfc = new JFileChooser();
        FileNameExtensionFilter filtro = new FileNameExtensionFilter("Mensajes (*.bti)",tipos);
        jfc.setSelectedFile(jfc.getSelectedFile());
        jfc.setAcceptAllFileFilterUsed(false);
        jfc.setFileFilter(filtro);
        int returnVal=jfc.showOpenDialog(this);
        if(returnVal==JFileChooser.APPROVE_OPTION){
            try {
                  String ruta = jfc.getSelectedFile().toString();
                  StringWriter w = new StringWriter();
                  FileReader archivo =  new FileReader(ruta);
                  int cod=0;
                  while ((cod = archivo.read())!= -1) {
                     w.write(cod);
                  }
                  StringBuffer c = w.getBuffer();
                  areaMensaje.setText(c.toString());
                  w.close();
                  archivo.close();
               }catch (FileNotFoundException ex) {
                     ex.printStackTrace();
               }catch (IOException ex) {
                     ex.printStackTrace();
                }
          }else if(returnVal==JFileChooser.CANCEL_OPTION){
                 return;
          }
}

//Método para guardar archivo (selecciona guardar en el menu)
public void guardarArchivo(){
        String tipos[] = {"bti"};
        jfc = new JFileChooser();
        FileNameExtensionFilter filtro = new FileNameExtensionFilter("Mensajes (*.bti)",tipos);
        jfc.setAcceptAllFileFilterUsed(false);
        jfc.setFileFilter(filtro);
        int returnVal=jfc.showSaveDialog(this);
        if(returnVal==JFileChooser.APPROVE_OPTION){
           String ruta = jfc.getSelectedFile().getPath();
           try {
                FileWriter resultado = new FileWriter(ruta+".bti");
                resultado.write(areaMensaje.getText());
                resultado.close();
                adjuntarImagen(ruta);
           }catch (IOException ex) {
```

```
                ex.printStackTrace();
            }
        }else if(returnVal==JFileChooser.CANCEL_OPTION){
                return;
        }
}
//Método que pregunta por la imagen para adjuntar
public void adjuntarImagen(String rutaSalida){
        int n = JOptionPane.showConfirmDialog(
            this, "¿Deseas adjuntar una imagen al mensaje?","Adjuntar Imagen",
            JOptionPane.YES_NO_OPTION);
        if (n == JOptionPane.YES_OPTION) {
            String tipos[] = {"jpg"};
            jfc = new JFileChooser();
            FileNameExtensionFilter filtro = new FileNameExtensionFilter("Imagenes (*.jpg)",tipos);
            jfc.setSelectedFile(jfc.getSelectedFile());
            jfc.setAcceptAllFileFilterUsed(false);
            jfc.setFileFilter(filtro);
            int returnVal=jfc.showOpenDialog(this);
            if(returnVal==JFileChooser.APPROVE_OPTION){
                String rutaEntrada = jfc.getSelectedFile().getPath();
                redimensionarImagen(rutaEntrada,rutaSalida);
            }else if(returnVal==JFileChooser.CANCEL_OPTION){
                    return;
            }
        }else if (n == JOptionPane.NO_OPTION) {
            return;
        }else {
            return;
        }
}
//Método para redimensionar imagen JPG para visualizar en el dispositivo móvil
public void redimensionarImagen(String rutaEntrada, String rutaSalida){
        try {
                BufferedImage original = ImageIO.read(new File(rutaEntrada));
                int w = original.getWidth(), h = original.getHeight();
                BufferedImage nueva = new BufferedImage(70*w/w,70*h/w,
BufferedImage.TYPE_3BYTE_BGR);
                Graphics2D g = nueva.createGraphics();
                g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                        RenderingHints.VALUE_INTERPOLATION_BILINEAR);
                g.drawImage(original,0,0,50*w/w,50*h/w, null);
                ImageIO.write(nueva, "png", new File(rutaSalida+".png"));
            } catch (Exception ex) {
                ex.printStackTrace();
            }
}
```

Lo revisado en el último diagrama y su codificación, permiten al usuario del servidor preparar la información para su publicación en el servidor del Sistema BTINFOme.

Con esto se concluye el diseño de las principales funciones que componen el Sistema BTINFOme cliente y servidor. Permitiendo continuar en la siguiente parte con la implementación del mismo.

PARTE 3

IMPLEMENTACIÓN CASO PRÁCTICO.

Diagrama de despliegue (UML)

Este diagrama nos va a permitir modelar de forma general la arquitectura hardware en el momento en que es ejecutado el Sistema BTINFOme, mostrando los diferentes componentes que lo conforman.

El siguiente diagrama de despliegue representa el Sistema BTINFOme en ejecución:

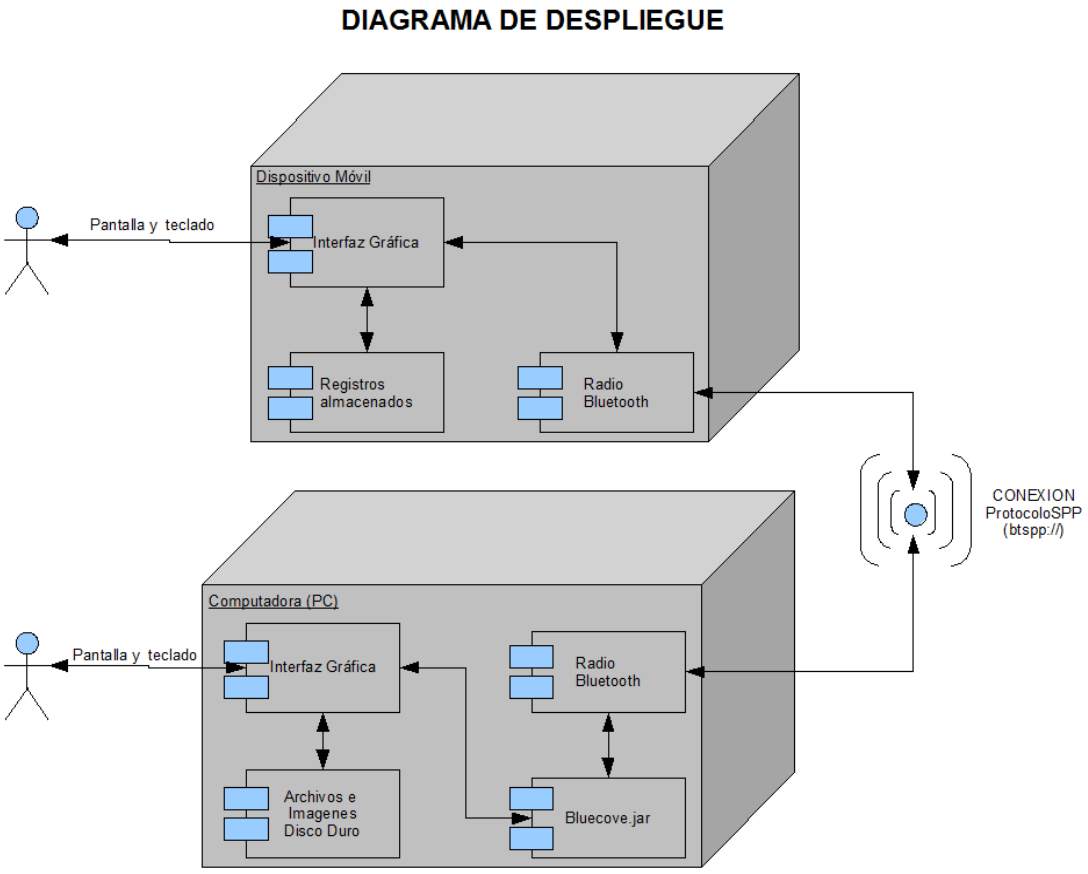


Figura 4.11

Diagrama de Despliegue del Sistema BTINFOme

El diagrama anterior muestra los principales componentes de los que se requiere para que el sistema funcione del lado cliente y servidor. En el siguiente tema se hará referencia a los requerimientos no funcionales del cliente y servidor del Sistema BTINFOme.

Requerimientos no funcionales del cliente

Los requerimientos no funcionales indican los componentes hardware y software necesarios para el Sistema BTINFOme.

Los componentes hardware necesarios por parte del cliente son:

- Dispositivo Móvil
- Radio Bluetooth incluido en el dispositivo móvil

Se reitera que con dispositivo móvil se refiere a teléfono celular el cual se mencionó que es uno de los dispositivos móviles en los que más se ha extendido la tecnología J2ME, dejando claro que en algún futuro se puede encontrar algún otro dispositivo móvil que cumpla con los mismos requerimientos.

Los componentes software necesarios por parte del cliente son:

- Soporte del dispositivo móvil para la tecnología J2ME con configuración CLDC 1.0 ó superior, perfil MIDP 2.0 ó superior y API JSR – 82 (paquete opcional).

Requerimientos no funcionales del servidor

Los componentes hardware necesarios por parte del servidor son:

- Computadora (PC) con microprocesador de 32 ó 64 bits a 1.6MHz ó superior, memoria RAM de 1Gb, disco duro de 20Gb
- Radio Bluetooth incluido ó agregado por algún puerto USB, el radio debe ser de alguno de los siguientes fabricantes: Broadcom (Widcomm), Winsock (Microsoft), BlueSoleil (IVTCorporation), Mac OSX, Linux BlueZ.

Estos son los principales fabricantes en los que se brinda soporte por Bluecove. Sin embargo antes de utilizar alguno se debe de conocer sus limitaciones que vienen marcadas en el siguiente enlace http://code.google.com/p/bluecove/wiki/stacks.

En el caso del Sistema BTINFOme se trabajó con el radio Bluetooth Broadcom (Widcomm) integrado en la Computadora, el cual ofrece como limitación notoria para el sistema, el no soportar el cambio de conectividad mediante software (LocalDevice.setDiscoverable()) si el dispositivo Bluetooth se encuentra apagado, por lo que el cambio de conectividad únicamente podrá realizarse si el dispositivo Bluetooth se enciende manualmente.

Los componentes software necesarios por parte del servidor son:

- Máquina Virtual de Java 1.6 ó superior.
- Implementación Bluecove (paquete bluecove-2.1.0.jar) que puede descargarse desde http://bluecove.org/.
- Sistema Operativo Windows XP ó Vista de 32 bits.
- Controladores instalados y actualizados del Radio Bluetooth.

Todos los requerimientos mencionados son los que se utilizaron para el desarrollo del caso práctico y diferentes ejemplos a lo largo del trabajo. Por último será mostrado el funcionamiento del Sistema BTINFOme en ejecución utilizando capturas de pantalla del emulador.

Pruebas funcionales del Sistema BTINFOme

Las pruebas funcionales aquí mostradas se hicieron con el emulador de Bluecove que se brindad en su pagina principal http://bluecove.org para emular conexiones Bluetooth.

Las primera pruebas son realizadas a la parte del servidor del Sistema BTINFOme, se muestra la venta principal con su menú inicial que contiene todas las opciones que puede realizar el usuario del servidor.

- Buscar Mensajes.- Permite la búsqueda de archivos para publicar
- Iniciar.- Inicia el servicio Bluetooth con la publicación de los archivos seleccionados
- Detener.- Detiene la actividad del servicio Bluetooth
- Editar.- Permite editar la información que será publicada

- Propiedades.- Despliega las características principales del dispositivo Bluetooth del servidor.
- Salir.- Termina la aplicación.

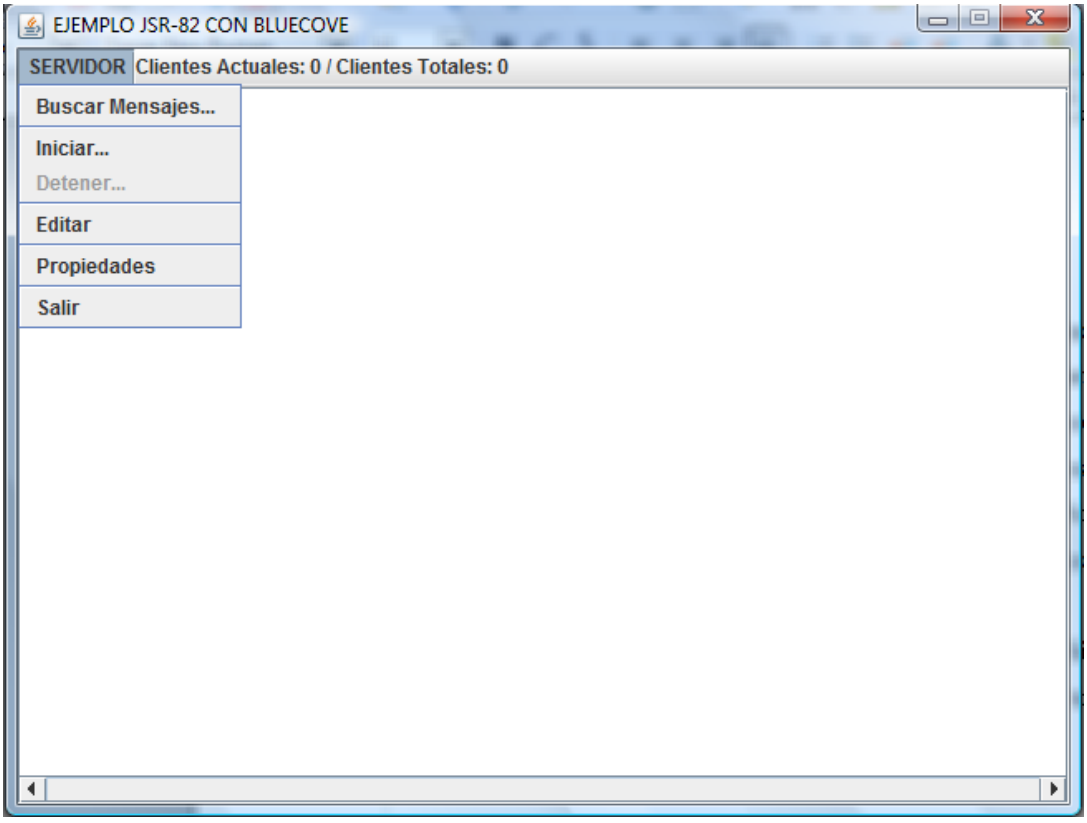


Figura 4.12

Pantalla principal del servidor

La primera opción va a permitir buscar en los directorios los archivos (con extensión .bti) que estemos interesados en publicar. Una vez seleccionado el directorio se cargan los archivos (con extensión .bti) localizados para seleccionar y aceptar aquellos que se desean publicar.

Habiendo ya seleccionado los archivos para publicar se puede iniciar el servidor para difundir la información.

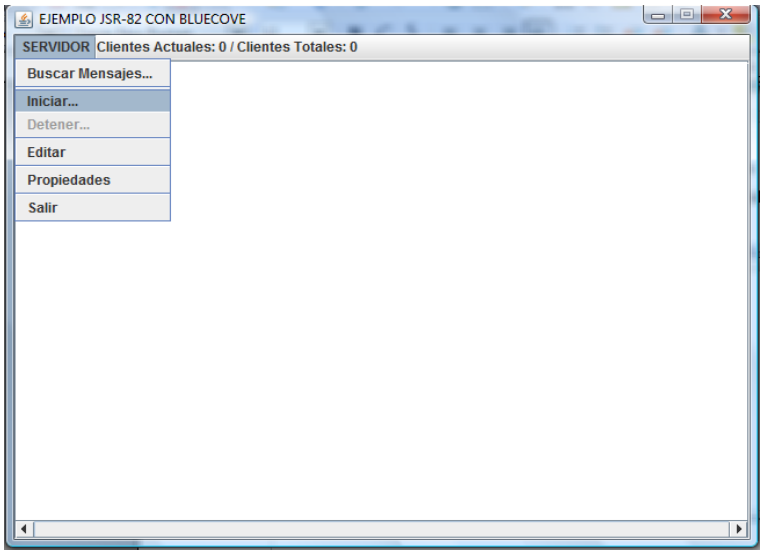


Figura 4.13

Selección de opción "Iniciar"

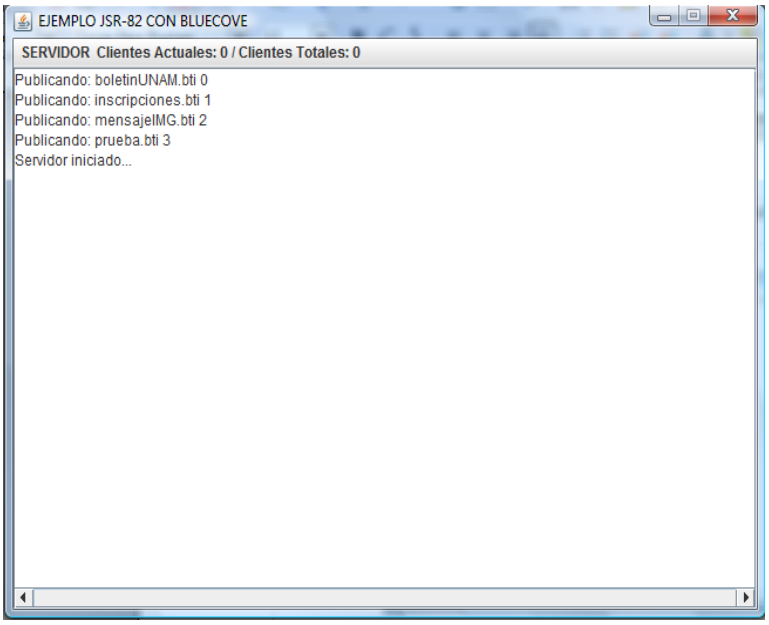


Figura 4.14

Despliegue de resultados al iniciar el servidor

Posteriormente puede detenerse la actividad del servidor con la opción "Detener...", que termina la atención a clientes cerrando las conexiones abiertas y cambiando el modo de conectividad del dispositivo Bluetooth.

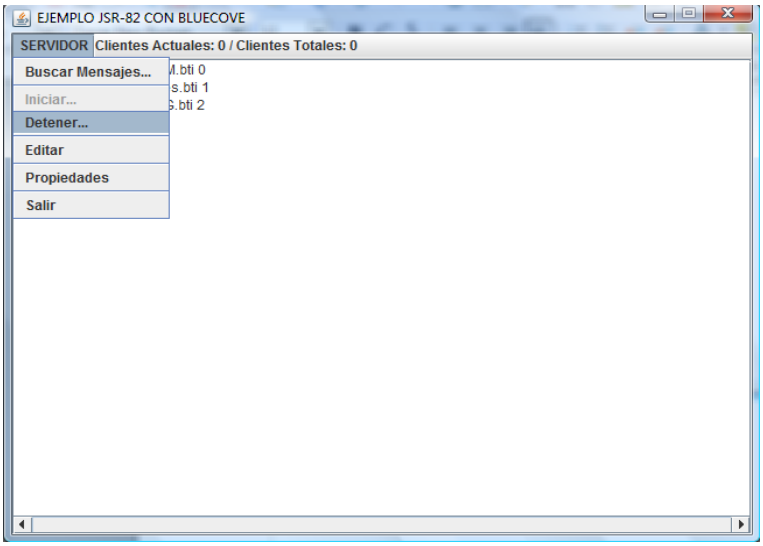


Figura 4.15

Selección de opción "Detener"

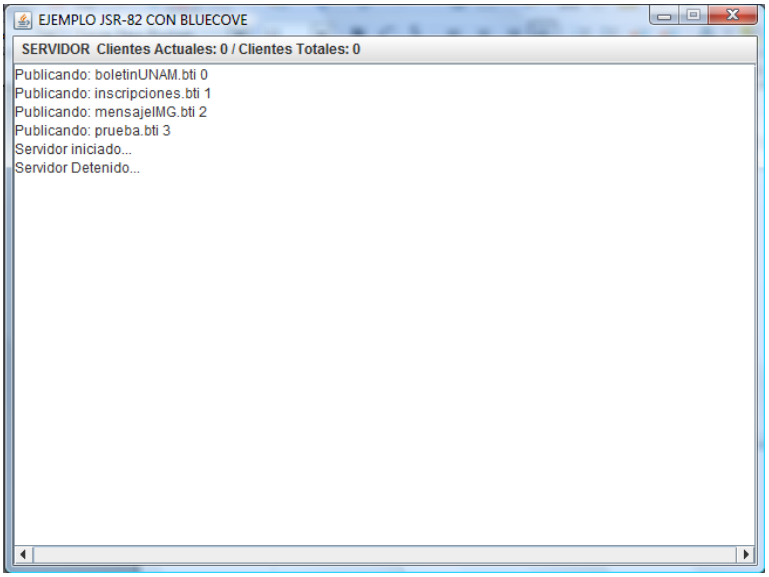


Figura 4.16

Despliegue de resultado al detener el servidor

Para la edición de mensajes se utiliza la opción editar, que desplegará el editor que nos permitirá crear ó modificar archivos (con extensión .bti) para publicar

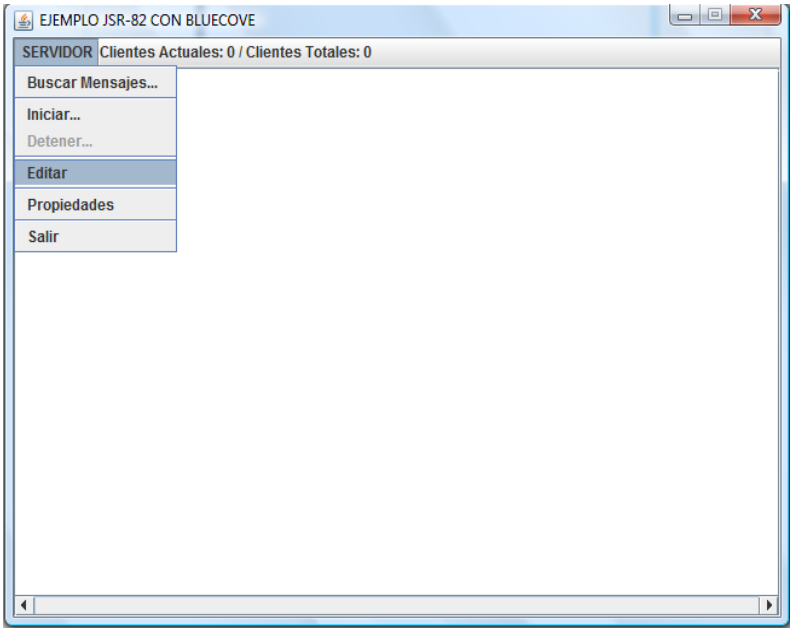


Figura 4.17

Selección de opción "Editar"

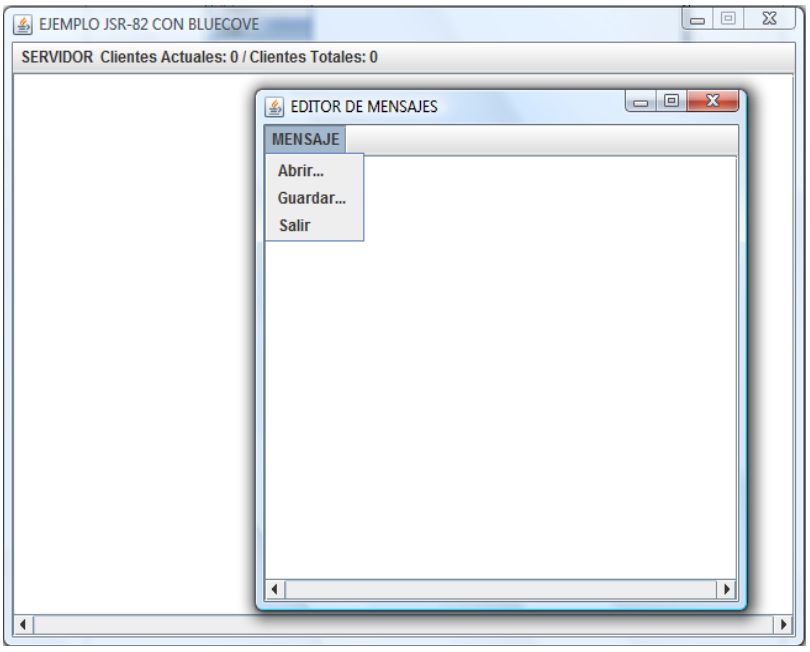


Figura 4.18

Despliegue en pantalla del editor

Opcionalmente podrá seleccionarse una imagen en formato JPG para adjuntarse junto al mensaje publicado, el editor se encarga de redimensionar la imagen a un tamaño adecuado para su envío y visualización en el dispositivo móvil.

Para las pruebas en el dispositivo móvil se utilizó el emulador "MicroEmulator" que puede ser descargado desde su sitio Web http://www.microemu.org/ y brinda soporte para trabajar junto al emulador de Bluecove.

El menú inicial en el dispositivo móvil presenta las siguientes opciones:

- Iniciar.- Comienza la búsqueda de dispositivos Bluetooth dentro del rango de alcance.
- Mis Documentos.- Muestra la información que ha sido almacenada.
- Borrar Documentos.- Elimina toda la información almacenada.
- Salir.- Termina la ejecución del MIDlet.

Se muestra la pantalla principal del MIDlet BTINFOme que contiene el menú inicial.

Figura 4.19

Menú principal de MIDlet

La primera opción comienza la búsqueda de dispositivos Bluetooth, que al encontrarlos serán desplegados en lista.

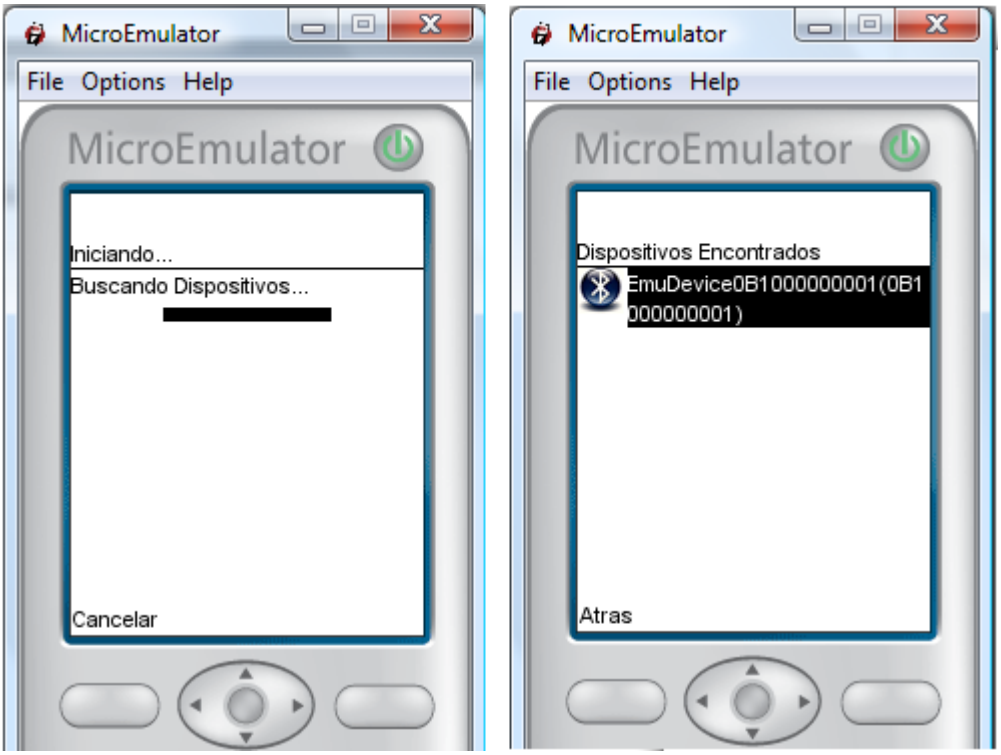


Figura 4.20

Búsqueda exitosa de dispositivos Bluetooth

El dispositivo localizado corresponde al emulador del servidor Bluecove que se esta utilizando. Una vez localizados los dispositivos se inicia la búsqueda de los servicios, la recuperación de atributos y se establece la conexión.

Figura 4.21

Despliegue de elementos publicados en el servidor

Una vez establecido lo anterior se despliega en pantalla los atributos del servicio que corresponden a los elementos que fueron publicados en el servidor. Seleccionado alguno de ellos se cargará en pantalla la información publicada.

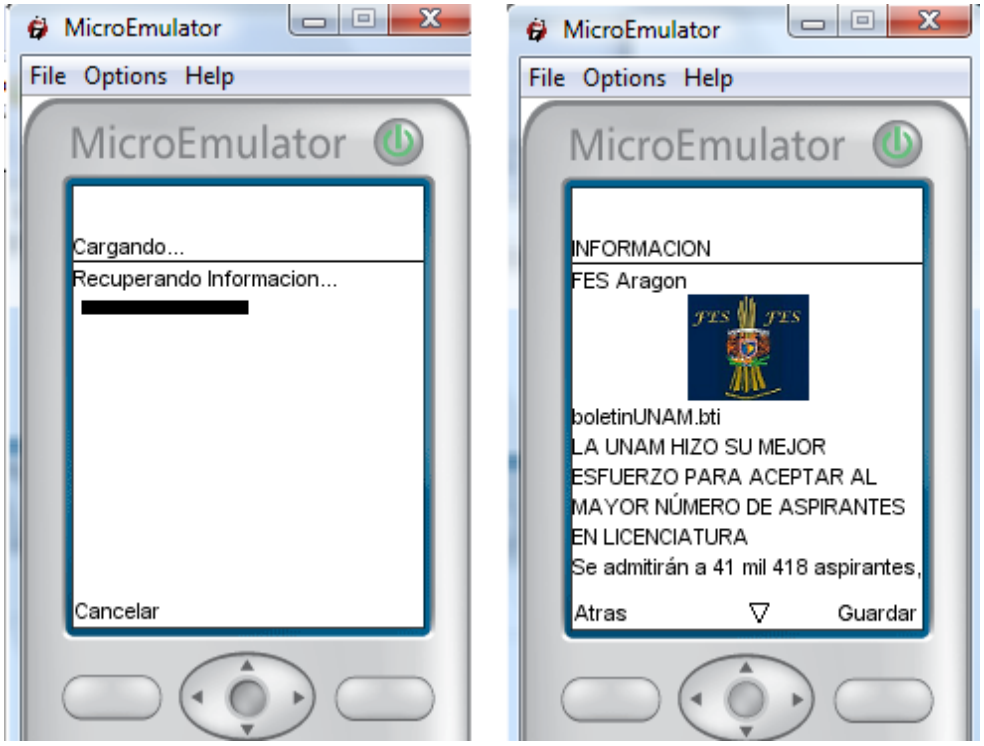


Figura 4.22

Despliegue de información publicada

Posteriormente la información consultada puede ser almacenada en el dispositivo móvil, obteniéndola de nuevo desde la opción de Mis Documentos del menú principal.

Únicamente se realizaron las pruebas funcionales sobre las principales actividades del Sistema BTINFOme que se trabajaron durante el diseño, por lo que el resto de la funcionalidad podrá apreciarse ejecutando directamente el sistema utilizando los recursos adjuntos en CD-ROM, mismos que se describen en el Apéndice B del presente trabajo.

Para finalizar se muestra una captura de pantalla del Sistema BTINFOme en funcionamiento, el cual consta de un servidor que brinda atención a dos clientes.

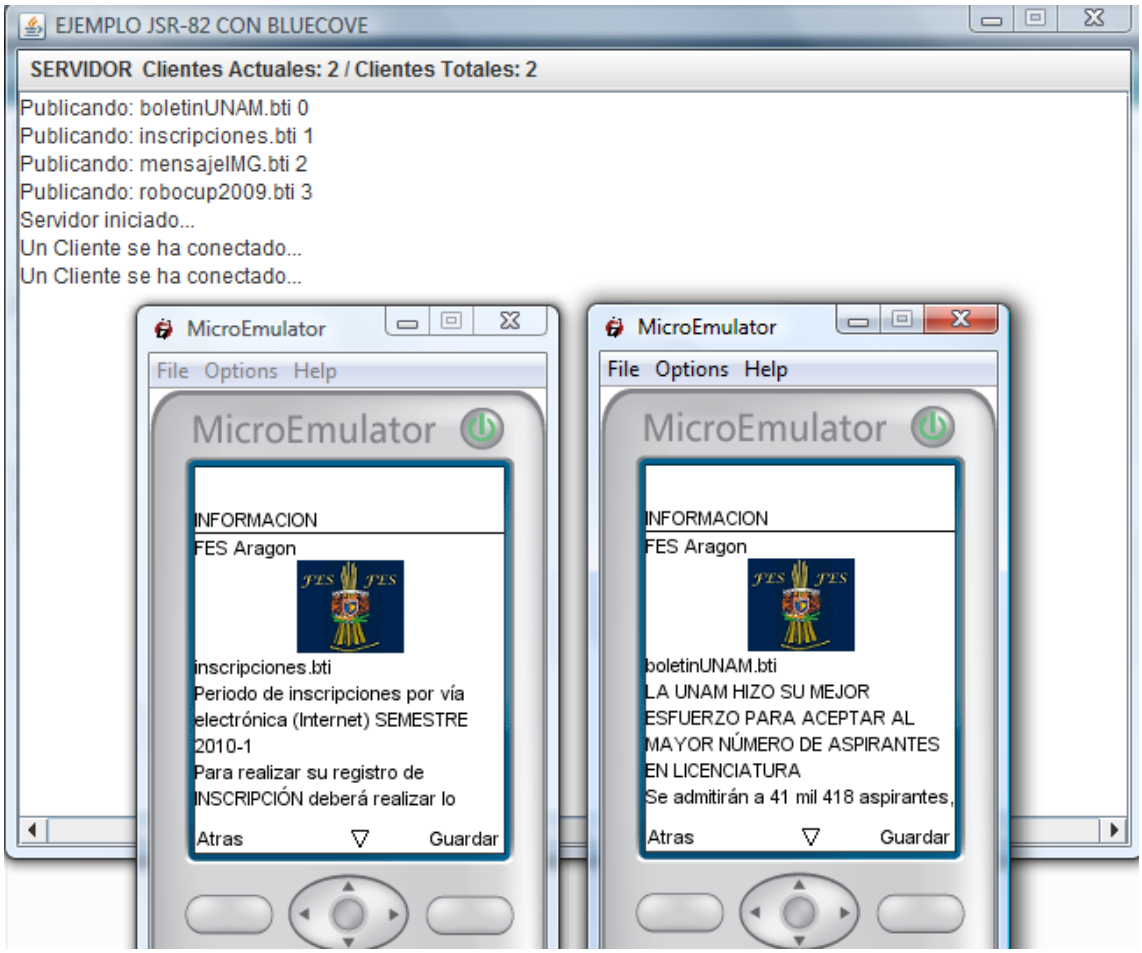


Figura 4.23

Sistema en funcionamiento, servidor atendiendo dos clientes

Sin embargo estas mismas pruebas se hicierón con tres dispositivos móviles reales y la PC, siendo los dispositivos móviles representantivos de tres principales compañias fabricantes de télefonos celulares Nokia, SonyEricsson y Motorola, demostrando la alta compatibilidad con la que se realizó el Sistema BTINFOme.

Las pruebas realizadas en el teléfono celular Nokia se completaron de manera satisfactoria, mostrando el dispositivo gran rapidez al trabajar en la búsqueda de dispositivos, búsqueda de servicios, establecimiento de la conexión y al recibir la información del servidor. De igual forma el despliegue de componentes en pantalla fue el óptimo de acuerdo a su resolución.

Las pruebas realizadas al teléfono celular SonyEricsson se completaron de manera satisfactoria, sin embargo a diferencia del modelo Nokia este presentó un poco más de rezago al trabajar en la búsqueda de dispositivos y al atender los eventos generados por

algún comando en pantalla. El despliegue de componentes en pantalla lo realizó de manera óptima acorde a su resolución.

Las pruebas realizadas en el teléfono celular Motorota se completaron de manera satisfactoria, comparado a los dos anteriores presentó una velocidad media al trabajar en la búsqueda de dispositivos, búsqueda de servicios, establecimiento de la conexión y al recibir la información del servidor. El despliegue de componentes en pantalla lo realizó de manera óptima de acuerdo a su resolución.

El servidor que fue una laptop, funcionó correctamente con cada una de las pruebas en los teléfonos celulares, permitiendo atender a los tres de forma simultánea.

Las pruebas descritas corresponden a la parte principal que se describió en el desarrollo de este caso práctico. Las partes correspondientes para almacenar información, consultar información almacenada y eliminar información almacenada funcionaron de manera satisfactoria en los tres teléfonos celulares, teniendo como única dificultad habilitar en los permisos Java la escritura y lectura de datos para el modelo de SonyEricsson.

El resto de las actividades no cubiertas durante el desarrollo del sistema por parte del servidor fueron probadas de manera satisfactoria.

Así se da por finalizado el capitulo 4 cumpliendo el objetivo de evaluar y mostrar de manera práctica todo lo estudiado en los capítulos anteriores permitiendo el desarrollo de una aplicación móvil con conexión interactiva PC – Dispositivo Móvil vía Bluetooth utilizando J2ME.

Para finalizar adecuadamente el presente trabajo solo resta presentar en la siguiente parte las conclusiones que se obtuvieron a lo largo de todo el estudio realizado en el presente trabajo.

CONCLUSIONES

CONCLUSIONES

En la actualidad ya es una realidad el gran crecimiento y expansión que han tenido diferentes tipos de dispositivos móviles y de igual forma cada vez es mayor la demanda para el desarrollo de aplicaciones móviles, el cual se complica por la diversidad de arquitecturas, capacidades y funciones de la diferente gama de dispositivos móviles.

Se ha presentado la tecnología J2ME como una buena solución para el desarrollo de aplicaciones móviles, brinda gran soporte para el desarrollo en diversos tipos de dispositivos móviles ofreciendo portabilidad y aprovechando de manera efectiva alguna otra tecnología incorporada como lo es el Bluetooth que siendo otra tecnología de comunicación inalámbrica de gran expansión complementa la diversidad de aplicaciones que pueden ser desarrolladas con la tecnología J2ME.

La programación con la tecnología J2ME y la especificación JSR – 82 no presenta demasiadas dificultades si ya se esta familiarizado con la programación orientada a objetos. Sin embargo al momento de implementar alguna aplicación es importante conocer la gama de dispositivos sobre los que se vaya a programar dicha aplicación, porque si bien es dicho que la tecnología J2ME brinda portabilidad, la diversidad de dispositivos móviles existentes ofrece diferentes restricciones y características en software y hardware lo que puede marcar diferencias de funcionamiento de un dispositivo móvil a otro.

El Sistema BTINFOme permitió de manera práctica aplicar lo estudiado y vislumbrar el potencial de la tecnología J2ME y Bluetooth teniendo como resultado un sistema interactivo entre el dispositivo móvil y la PC beneficiando a la comunidad escolar con un sistema gratuito de difusión de información.

Aunque la tecnología J2ME este enfocada en el desarrollo de aplicaciones para dispositivos con recursos limitados en software y hardware, no significa que el desarrollo sea más corto ó más simple ya que puede presentar dificultades igual que en un sistema grande, por lo que la Ingeniería de Software se aplica bajo las mismas condiciones en el desarrollo de aplicaciones móviles, ejemplo claro el Sistema BTINFOme una aplicación móvil (cliente) y una aplicación de escritorio (servidor) desarrollados bajo los mismos criterios de la Ingeniería de Software.

No hay duda del brillante presente y futuro que tendrán los diversos dispositivos móviles, lo que abrirá diferentes oportunidades en el desarrollo de aplicaciones móviles, ampliando el área para el desarrollo profesional en la Ingeniería en Computación. Y aunque en un futuro la tecnología J2ME sea la más ó menos utilizada, el mercado para el desarrollo de aplicaciones móviles seguirá creciendo y ampliando el campo laboral y de estudio para diversos tipos de profesionistas.

El presente trabajo se enfocó en la tecnología J2ME que actualmente es la más expandida para el desarrollo de aplicaciones móviles, y aunque en México es poco el auge que se tiene en esta área, se puede aprovechar de manera óptima el estudio realizado para aquellas personas que deseen involucrarse más en hacer crecer y aprovechar esta área del mercado poco explotada que puede brindar grandes beneficios a la sociedad.

Personalmente el presente trabajo deja una gran satisfacción por los logros alcanzados al cumplir cada uno de los objetivos propuestos, resolviendo y afrontando de manera adecuada diferentes problemas encontrados al transcurrir cada uno de los capítulos realizados. El desarrollo de todo este trabajo permitió aprender y crecer más como persona y como profesionista.

APÉNDICES

APÉNDICE A

APÉNDICE B

APÉNDICE A

Diagramas de secuencia capítulo 4 Sistema BTINFOme cliente

Diagrama de secuencia que describe la forma en que el MIDlet BTINFOme almacena la información consultada por el cliente.

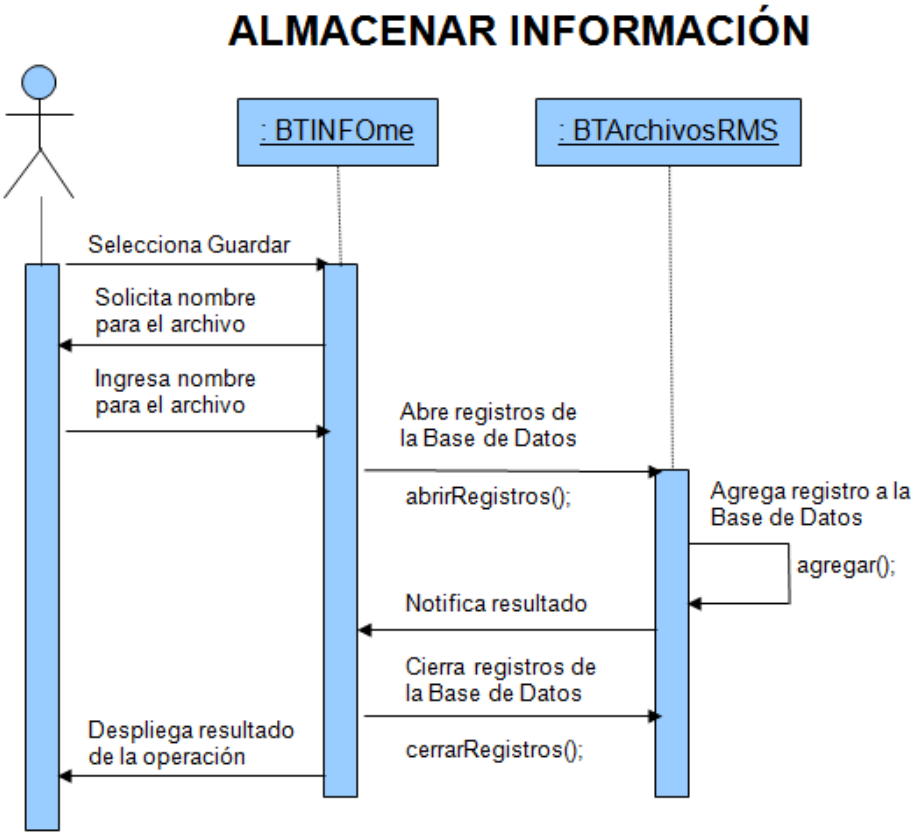


Diagrama de secuencia que describe la manera en que el MIDlet BTINFOme recupera la información almacenada para ser consultada por el cliente.

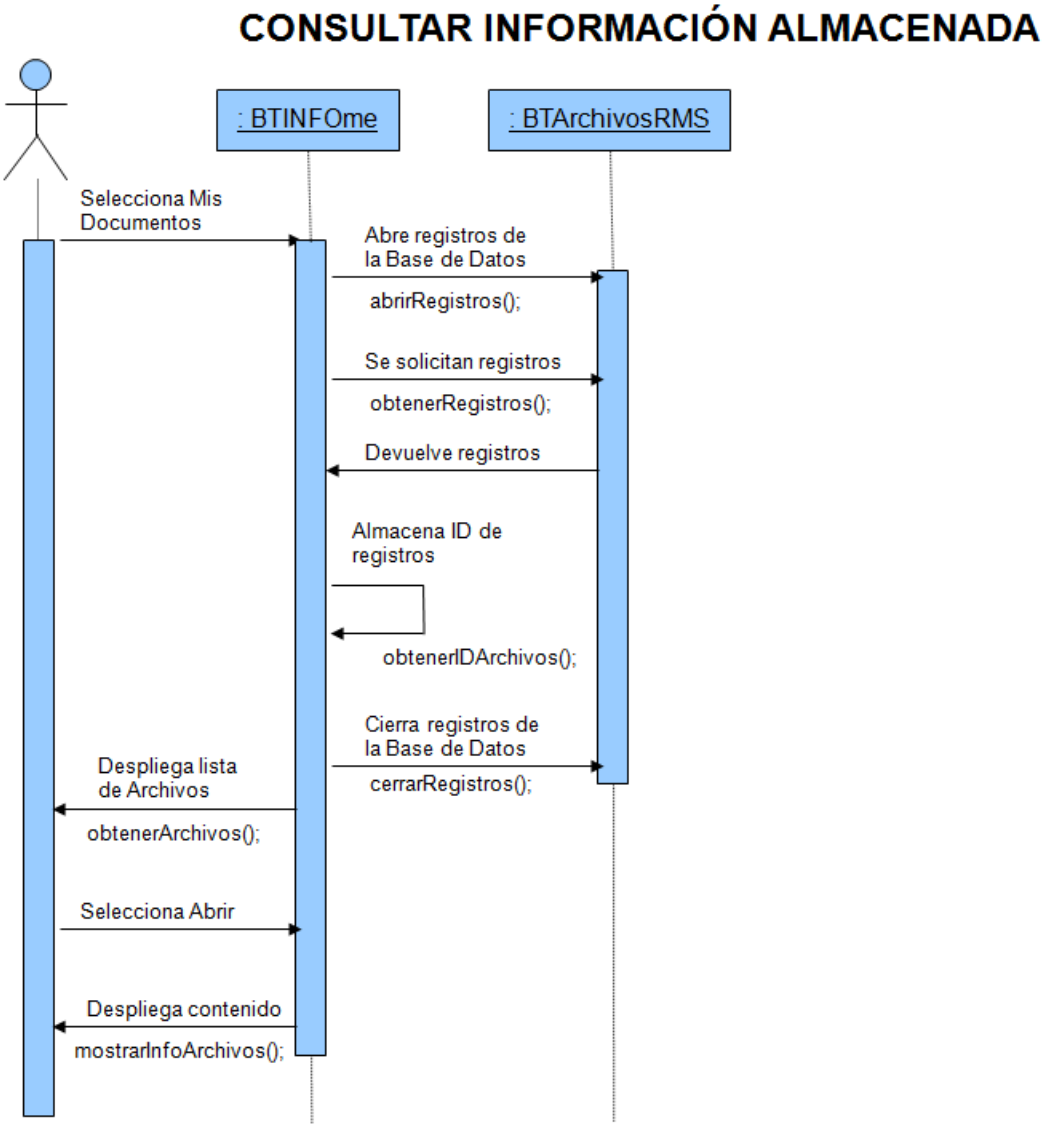
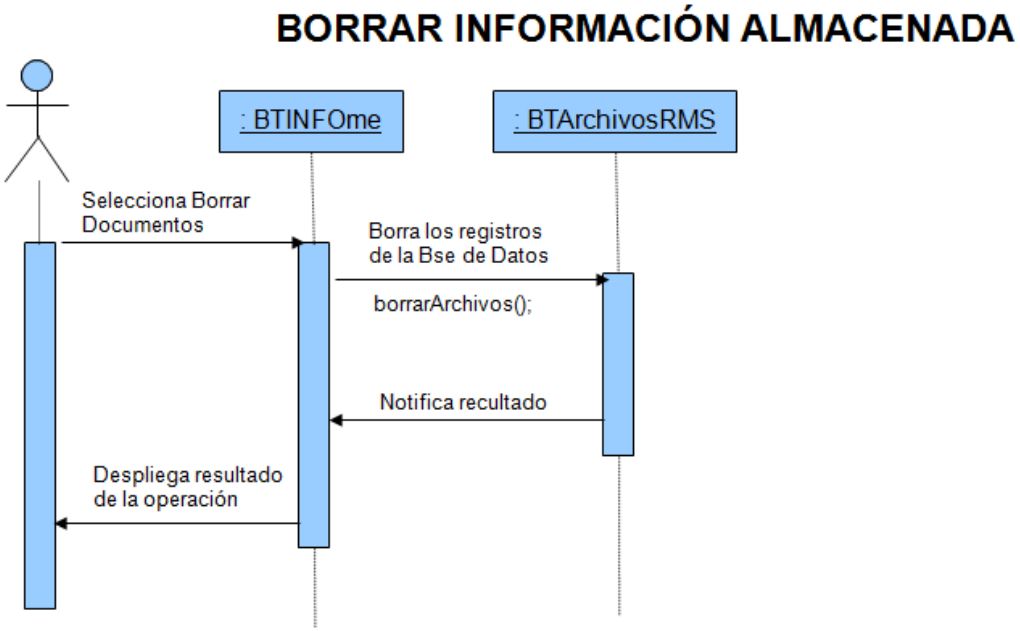


Diagrama de secuencia que elimina todos los registros creados por el almacenamiento de información.



Diagramas de secuencia capítulo 4 Sistema BTINFOme servidor

Diagrama de secuencia que muestra las propiedades funcionales del dispositivo Bluetooth en el servidor BTINFOme.

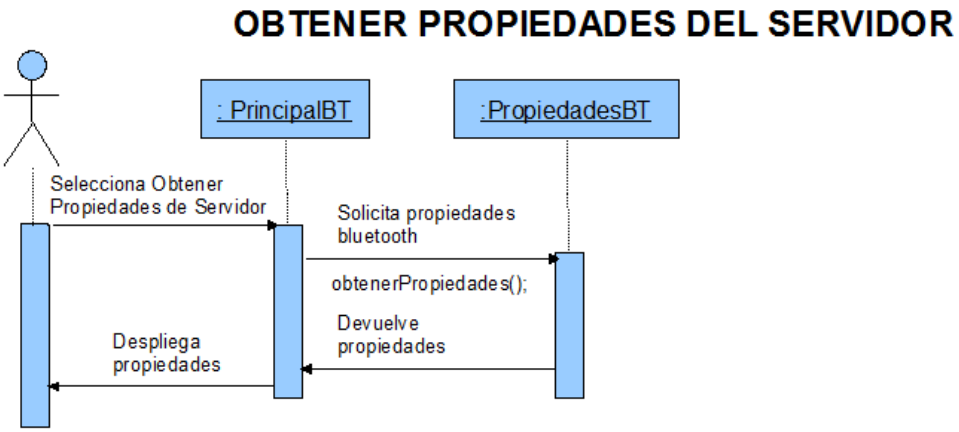
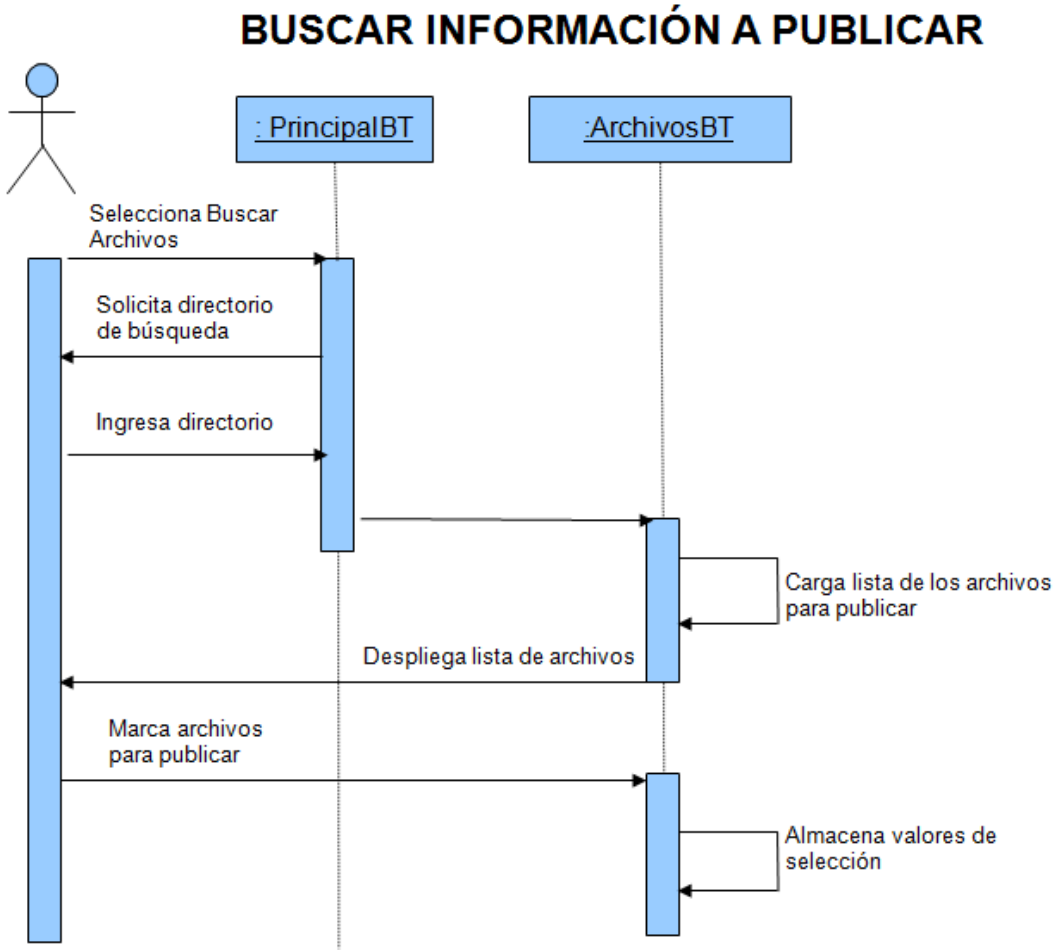


Diagrama de secuencia que permite buscar información que pueda ser publicada dentro del servidor del Sistema BTINFOme.



APÉNDICE B

Descripción del contenido adicional en el CD-ROM

El contenido presenta diversas carpetas en las que se agrupan los ejemplos que están descritos en cada uno de los capítulos.

La carpeta "**/Capitulo_2**" contiene lo siguiente:

- Carpeta "/ejemplos_1_2_3" en la que se encuentran los archivos Ejemplos_CAPITULO2.jad y Ejemplos_CAPITULO2.jar que incluyen la Suite MIDlet de los tres ejemplos vistos en el capítulo 2. Los archivos .jad y .jar son los que deberán transferirse al dispositivo móvil (teléfono celular) para probar su funcionamiento de forma real.

La carpeta "**/Capitulo_3**" contiene lo siguiente:

- Carpeta "/ejemplos_1_2" que contiene los archivos Ejemplos_CAPITULO3.jad y Ejemplos_CAPITULO3.jar que incluyen la Suite MIDlet de los dos primeros ejemplos vistos en el capítulo 3 para J2ME.

- Carpeta "ejemplos_3_4" que contiene los archivos Ejemplo3.jar y Ejemplo4.jar que para su correcto funcionamiento emplean la implementación Bluecove (bluecove-2.1.0.jar adjunto en la carpeta "/lib") para J2SE, ambos archivos representan los últimos dos ejemplos vistos en el capítulo 3.

La carpeta "**/Capitulo_4**" contiene lo siguiente:

- Carpeta "**/ClienteINFOme**" que contiene los archivos ClienteINFO.jad y ClienteINFO.jar que es la aplicación cliente del caso práctico desarrollado, ambos archivos deberán transferirse al dispositivo móvil para probar su funcionamiento de forma real.

- Carpeta "**/ServidorINFOme**" que contiene el archivo Servidor_CAPITULO4.jar que utiliza la implementación Bluecove (bluecove-2.1.0.jar adjunto en la carpeta "/lib") y es la aplicación servidor del caso práctico desarrollado.

- Carpeta "**/ArchivosJAR**" que contiene los archivos bluecove-2.1.0.jar, bluecove-emu-2.1.0, y microemulator.jar los cuales son utilizados para ejecutar los emuladores y visualizar el Sistema BTINFOme en funcionamiento, tal y como se presentó en el capítulo 4. Adicionalmente se agrega el archivo bluecove-gpl-2.1.0.jar el cual debe ser utilizado junto con bluecove-2.1.0.jar en Sistemas Operativos Linux.

- Archivo "**EmuladorBT.bat**" ejecuta el emulador de hardware Bluetooth. Este emulador representará el hardware Bluetooth para el cliente y servidor del Sistema BTINFOme, ejecutados por los archivos "ClienteBT.bat" y "ServidorBT.bat".

- Archivo "**ClienteBT.bat**" ejecuta la aplicación cliente del caso práctico desarrollado en el emulador del dispositivo móvil (teléfono celular) y utiliza el emulador de hardware Bluetooth establecido por el archivo "EmuladorBT.bat".

- Archivo "**ServidorBT.bat**" ejecuta la aplicación servidor del caso práctico desarrollado y utiliza el emulador de hardware Bluetooth establecido por el archivo "EmuladorBT.bat".

La carpeta "**/Herramientas**" contiene lo siguiente:

- Archivos "jdk-6u6-windows-i586-p.exe" y "netbeans-6.1-windows.exe". El primero archivo utilizado para la instalación de la Máquina Virtual Java 1.6 y el segundo para la instalación del IDE NetBeans 6.1. Ambas herramientas utilizadas para el desarrollo de ejemplos y caso práctico del presente trabajo.

Para finalizar se rectifica que el código de los ejemplos y caso práctico ya han sido explicados a lo largo de los capítulos, por lo que no se ve necesario agregarlo en el material adicional.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

LIBROS

Comunicaciones Inalámbricas.
David Roldán.
Alfa Omega RA-MA.

Java 2 Micro Edition (Manual de usuario y tutorial).
Agustín Froufe Quintas. Patricia Jorge Cárdenes.
Alfa Omega RA-MA.

Performance Modeling and Análisis of Bluetooth Networks.
Jelena Misic. Vojislav B. Misic.
AUERBACH.

Bluetooth Application Programming with the Java APIs Essentials Edition
Timothy J. Thompson, Paul J. Kline, and C Bala Kumar.
Morgan Kaufmann.

Bluetooth for Java
Bruce Hopkins and Ranjith Antony
Apress

TUTORIALES

Tutorial Tecnología Bluetooth.
Ayala Danny. Zamora Miguel.

Tutorial Java a Tope: J2ME (JAVA 2 MICRO EDITION)
Sergio Gálvez Rojas**,** Lucas Orteda Díaz
Universidad de Malaga.

Tutorial Java 2 Micro Edition: Soporte Bluetooth
Pedro Daniel Borches Juzgado
Universidad Carlos III de Madrid

Tutorial JSR-82: Bluetooth desde Java
Alberto Gimeno Brieba

Tutorial Programación de videojuegos para móviles con J2ME
Alberto García Serrano

Tutorial Object-Oriented Application Analysis and Design for Java Technology (UML) Student Guide
Sun Microsystems

Tesis J2ME Bluetooth Programming
André N. Klingsheim
Universidad de Bergen Noruega

INTERNET

http://www.bluetooth.com/bluetooth/
http://java.sun.com/
http://java.sun.com/javame/index.jsp
http://java.sun.com/developer/technicalArticles/javame/mobilemarket/
http://bluecove.org/
http://code.google.com/p/bluecove/wiki/stacks
http://www.microemu.org/
http://msdn.microsoft.com/es-es/library/f44bbwa1.aspx
http://www.adobe.com/es/products/flashlite/features
http://www.symbian.org/
http://www.android.com/
http://articles.techrepublic.com.com/5100-10878_11-6171367.html
http://www.portalplanetasedna.com.ar/telefono1.htm
http://www.yucatan.com.mx/especiales/celular/3g.asp