ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELECTRICA Y ELECTRÓNICA
CARRERA DE INGENIERIA EN ELECTRONICA Y TELECOMUNICACIONES

PROYECTO DE GRADO PARA LA OBTENCION DEL TITULO EN INGENIERIA

ANÁLISIS DE LA TECNOLOGÍA BLUETOOTH EN LA FORMACIÓN DE REDES WPAN EMPLEANDO DISPOSITIVOS MÓVILES

DANILO EDUARDO SALAZAR RAMOS

SANGOLQUI – ECUADOR

2011

Declaración de Responsabilidad

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

DANILO EDUARDO SALAZAR RAMOS

DECLARO QUE:

El proyecto de grado denominado "Análisis De La Tecnología Bluetooth En La Formación De Redes Wpan Empleando Dispositivos Móviles", ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie, de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 03 de Abril de 2011

Danilo Eduardo Salazar Ramos

Autorización de publicación

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

Yo, Danilo Eduardo Salazar Ramos

Autorizo a la Escuela Politécnica del Ejército la publicación, en la biblioteca virtual de la Institución del trabajo “Análisis De La Tecnología Bluetooth En La Formación De Redes Wpan Empleando Dispositivos Móviles”, cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y autoría

Sangolquí, 03 de Abril de 2011

Danilo Eduardo Salazar Ramos

Certificado de tutoría

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICADO

ING. GONZALO OLMEDO, Ph.D
ING. PAÚL BERNAL,

CERTIFICAN

Que el trabajo titulado "Análisis De La Tecnología Bluetooth En La Formación De Redes Wpan Empleando Dispositivos Móviles", realizado por Danilo Eduardo Salazar Ramos, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la ESPE, en el Reglamento de Estudiantes de la Escuela Politécnica del Ejército.

Debido a que se trata de un trabajo de investigación recomiendan su publicación.

El mencionado trabajo consta de un documento empastado y un disco compacto el cual contiene los archivos en formato portátil de Acrobat (pdf). Autorizan a Danilo Eduardo Salazar Ramos que lo entregue al Ingeniero Gonzalo Olmedo, en su calidad de Coordinador de la Carrera.

Sangolquí, 03 de Abril de 2011

Ing. Gonzalo Olmedo	Ing. Paúl Bernal
DIRECTOR	CODIRECTOR

AGRADECIMIENTO

A mis Directores de proyecto por su paciencia y colaboración durante todo el tiempo que estuve investigando sobre el tema. Grandes profesionales que son un ejemplo a seguir.

A mi familia, sobretodo a mi hermana quién retiteradamente de forma particular me recordaba que la investigación debe seguir adelante y a mi padre quién siempre confió en mí a pesar de las adversidades . Ellos son la razón por la que todos los días lucho.

Pero sobre todo a Dios por darme la oportunidad de prepapararme y concederme el don de la paciencia e inspiración para terminar esta etapa de mi vida. Señor a ti toda la Gloria por siempre

DEDICATORIA

A mi madre quién con su esfuerzo y dedicación me alentó durante todo el camino hasta llegar a la meta, desde que me llevó en su vientre. Quién fue mi primera maestra y aún lo sigue siendo cuando tropiezo en la vida.

Gracias madre querida por cuidarme y hacer de mi un hombre de bien, considera esta es la tesis que te faltaba para obtener tu título y que no la pudiste hacer por quedarte conmigo en casa. Quiero que sepas que no perdiste tu tiempo y que siempre hago todo lo posible para que estés orgullosa de mi :).

¡Por siempre Gracias!

Tu hijo.

PRÓLOGO

El no tener una infraestructura fija permite que las redes Ad Hoc se adapten a las necesidades de los usuarios, por lo que se presentan como una opción más flexible al momento de establecer comunicaciones para dispositivos móviles. La posibilidad de crear redes temporales en cualquier sitio, es decir que únicamente existen cuando la necesidad está presente, les permite adaptarse fácilmente al entorno en el que se encuentran; un claro ejemplo de ello es ante la recuperación por desastres que pudiese ocurrir en determinada zona.

Bluetooth, por su popularidad y sencillez, se ha convertido en el estándar “de facto” para formar redes Ad-Hoc, específicamente denominadas Piconets, que permiten que un maestro pueda tener hasta 7 esclavos al mismo tiempo. Se estimó que para el año 2009 más del 90% de los dispositivos móviles fabricados poseía está tecnología y que en el mercado alrededor del 60% de los mismos ya la incorporaba.

El lenguaje de programación Java ha estado presente en los dispositivos electrónicos desde su creación en los años 90. La recomendación JSR-82 ha permitido que múltiples aplicaciones sean desarrolladas para dispositivos móviles, desde juegos hasta aplicaciones de control de personal, siendo de vital importancia el comprender la implementación JABWT para cada dispositivo y con las características que maneja .

El costo de los módulos Bluetooth es bastante bajo en comparación con los de otras tecnologías inalámbricas, de igual manera su consumo de energía. Por tal motivo, se los están utilizando cada vez más para brindar conectividad a dispositivos electrónicos convencionales, como electrodomésticos, equipos médicos y en sensores para plantas de

producción. Por lo que se resulta inminente la presencia de redes Ad-Hoc o específicamente Piconets que deberán ser entendidas y manejadas por las personas, de ahí la importancia de su estudio. [

Es necesario mencionar que aún no existen dispositivos móviles en el mercado capaces de formar Scatternets, por lo que el estudio se centrará en validar el comportamiento de los celulares en las Piconets a través de una aplicación de mensajería utilizando el protocolo OBEX, donde se tendrá la mayor cantidad posible de esclavos y un maestro que centralice el intercambio de información.

Por utilizar el lenguaje JAVA se utilizará el entorno de desarrollo (IDE) de Sun Microsystems llamado NETBEANS, ya que es de libre distribución y compatible con la mayor cantidad de marcas de celulares y teléfonos inteligentes del mercado.

Debido a que JABWT solo marca un estándar en la programación y no los perfiles que cada dispositivo móvil debe tener, resulta indispensable escoger una marca de celular o teléfono inteligente que sea lo más compatible con el desarrollo de aplicaciones para Bluetooth. Es así que se propondrá una marca bajo la cual la aplicación corra de la mejor manera, teniendo como prioridad elegir un modelo existente en el mercado.

Por otro lado, se garantiza que la aplicación permitirá el enviar mensajes entre tres dispositivos móviles con el programa instalado sin inconvenientes, siempre y cuando cumplan con los perfiles de Bluetooth en su diseño; es decir, que se apeguen lo más posible al perfil de la marca y modelo de celular seleccionados.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

1. Antecedentes

La primera década de este siglo ha tenido sin lugar a dudas como centro a la información y la forma en cómo la obtenemos, es así que han aparecido nuevas formas de comunicarnos de formas cada vez más sencillas y rápidas. Bajo este entorno tenemos que los dispositivos móviles han aparecido como la solución ideal que conjuga de forma armónica la movilidad con la velocidad en las comunicaciones. Es así que estudios [1] por el año 2005 ya indicaban que alrededor del 80% de las personas de países del primer mundo tienen teléfonos móviles y en nuestro país el último informe del año 2009 [2] indicaba que existen 12.788.000 celulares entre las tres operadoras móviles. Sin lugar a duda los dispositivos móviles ya juegan un papel primordial en la forma en la que nos comunicamos pero aún no hemos visto la cúspide de su evolución.

1.1 Computación Ubicua y Dispositivos móviles.

En 1991, Mark Weiser en su trabajo "*The Computer for the Twenty-First Century*" habló por primera vez de la computación ubicua [3] y como esta permite al usuario tener un nivel de comunicación con la computadora tan natural que pueda enfocarse realmente en lo que quiere lograr mientras la computadora hace los procedimientos técnicos y matemáticos necesarios. Si bien Weiser concibió la aplicación de sus ideas dentro de la realidad virtual, es gracias al avance de los dispositivos móviles que hoy en día este concepto ha ganado espacio. Una total convergencia entre las computadoras personales y los dispositivos móviles se está

dando, atrás quedaron los días en los que se podía encontrar alguna diferencia entre una agenda personal (PDA) y un celular. De hecho, cada vez se encuentra menos diferencias entre una computadora personal y un celular, hoy en día esto se evidencia con la salida de teléfonos como el N900 de Nokia [4], que tiene un sistema operativo denominado MAEMO [5] y que no es más que una distribución de LINUX, el Android de Google [6], con un sistema operativo de código abierto desarrollado por Google, el IPHONE [7], con todo lo que conlleva tener el respaldo de Steve Jobs, toda la línea Blackberry y otros competidores con no mucho terreno como PALM o HTC. Pronósticos que indican que la tienda de aplicaciones de Apple tendrá 300000 aplicaciones y la de Android 75000 aplicaciones para el final de 2010[8] nos muestran que aún queda mucho camino por recorrer en el mundo de los dispositivos móviles.

Tabla 1.1 Sistemas Operativos para dispositivos móviles más populares [9]

	Symbian	**Java ME**	**Android**	**iPhone OS**	**Blackberry**
Fundamento	C++	Java	Java	Objective-C	Java
Dificultad en Aprendizaje	Complicado	Promedio	Promedio	Sencillo	Promedio
Desarrollo Multiplataforma	Sí	Sí	No	No	No
Costo Herramientas Desarrollo	Varía	Gratis	Gratis	99 usd al año	Gratis

En la Tabla 1. Se muestran los datos comparativos de los sistemas operativos más populares para dispositivos móviles en la actualidad, de esta información cabe destacar que java funciona como fundamento de tres de ellos. Para el año 2003 se dio la aparición de Symbian gracias al desarrollo en conjunto de varias empresas de telefonía móvil entre las que destacan Nokia, Sony Ericsson, Samsung, LG, Siemens entre otras. A lo largo de este tiempo Symbian se ha popularizado, en parte por la gran popularidad de los celulares Nokia pero por parte porque al utilizar el lenguaje

C++ como base es bastante confiable y permite manejar operaciones de bajo nivel. Actualmente Symbian aún lo utilizan varios modelos de celulares, sobre todo en los que no son considerados como inteligentes de las marcas Nokia, Sony Ericsson y Samsung. Por otro lado Java, gracias a su máquina virtual, siempre ha sido visto como el lenguaje perfecto para los dispositivos móviles desde los inicios del desarrollo de las aplicaciones para los dispositivos móviles y es así que actualmente existe una gran cantidad de aplicaciones desarrolladas en este lenguaje que funcionan inclusive sobre otros sistemas operativos como los de Android y Blackberry porque tienen implementada la máquina virtual java. Los otros sistemas operativos como Android, Blackberry y el iPhone OS fueron diseñados específicamente para teléfonos inteligentes con nuevas capacidades y cuyo enfoque va destinado a crear aplicaciones similares a las de una PC.

Sea con Java o con cualquier otro lenguaje de programación, lo que tenemos ahora es que la programación está enfocada en que las aplicaciones se adapten perfectamente a la rutina diaria de una persona con tal naturalidad que le sean imperceptibles para mejorar las condiciones de vida en las que se desenvuelve, esto es algo bastante similar al concepto de computación Ubicua de Weiser pero que actualmente es conocido como *Context Aware* [10].Para que una aplicación sea denominada como *Context Aware* debe cumplir con la premisa de que se debe adaptar al usuario tanto en función como en entorno, lo cual implica que la aplicación debe tener algo de “inteligencia” o por lo menos capacidades que van más allá de hacer llamadas o enviar mensajes para conocer al usuario, su comportamiento y la situación en la que se encuentra. Para alcanzar tal nivel de interactividad se necesita además de un hardware cada vez más robusto e inclusive de cambiar la forma en que los dispositivos se conectan, es ahí en donde entra en escena el concepto de redes Ad Hoc y de los beneficios que presenta.

1.2 Redes AD HOC

1.2.1 Definición

Por definición es aquella que se crea para un propósito específico y cuyos miembros están en igualdad de condiciones para comunicarse [11]. Estas redes son sinónimo de flexibilidad ya que pueden ser creadas en cualquier momento y bajo cualquier condición siempre y cuando sus miembros se encuentren listos para formarlas. Entre las múltiples aplicaciones que existen para este tipo de redes están las de permitir la comunicación en zonas de desastre o en zonas muy alejadas donde resulta muy difícil implementar una infraestructura de red típica. En esta sección se dará una pequeña introducción a esta tecnología pero se profundizará más sobre el tema en el capítulo II.

Las redes Ad Hoc casi siempre son relacionadas a tecnologías inalámbricas, puesto que resulta evidente que la forma más simple de construir una red de forma instantánea y sin infraestructura es sin utilizar cables. La IETF ha denominado a las redes Ad Hoc móviles como MANET por sus siglas en inglés y también definió que serían necesarios nuevos algoritmos y protocolos para el intercambio de información ya que TCP/IP no parece ser el más adecuado para este tipo de redes. El objetivo de los nuevos protocolos era el de ser adaptivos, es decir que puedan cambiar de ruta basándose en tráfico, congestión u caídas de enlaces en corto tiempo según los requerimientos del entorno; además, deben establecer redes independientes y descentralizadas que permitan la comunicación aún en las peores condiciones. Finalmente, para el caso específico de dispositivos móviles se debe considerar la baja carga que debe tener sobre el hardware, por las limitaciones de capacidad y energía que tienen.

1.2.2 Clasificación

A los protocolos desarrollados para redes Ad Hoc los podemos clasificar como proactivos y reactivos.

- Reactivos.- Son aquellos que únicamente establecen una ruta de comunicación al momento en que dos nodos necesitan comunicarse, es

decir que no hacen un intercambio previo de mensajes. Son conocidos como protocolos bajo demanda y tienen como desventaja el tiempo de retardo que se produce en la comunicación.

- Proactivos.- Son aquellos que se anticipan a los cambios en la estructura de la red mediante el intercambio de mensajes, lo cual permite que cada nodo elija de manera óptima una ruta a su destino. Tiene como desventaja el presentar una carga adicional de procesamiento para cada nodo.

Es así que después de varias propuestas de protocolos y algoritmos de encaminamiento, la IETF ha considerado a los siguientes protocolos como los más estables y prometedores para las redes Ad Hoc:

- Ad Hoc On Demand Distance Vector (AODV).- Es un protocolo de encaminamiento reactivo basado en la distancia y que utiliza una tabla de enrutamiento por nodo.
- Dynamic Source Routing for Protocol Mobile Ad Hoc Networks (DSR).- Es un protocolo de encaminamiento reactivo parecido a AODV pero que utiliza más información sobre el nodo origen.
- Optimized LinkState Routing Protocol (OLSR). - Encaminamiento proactivo que utiliza el intercambio de mensajes de control de topología para conocer la estructura de la red (similar a OSPF).

Se adentrará con más información sobre estos protocolos en el capítulo II que trata sobre las redes Ad Hoc específicamente. Lo que realmente resulta importante es el entender es el concepto de redes que no dependen de estructuras físicas y que pueden ser creadas en cualquier parte, se podría decir que cuando sea necesaria y basada en las necesidades del entorno.

Existen dispositivos específicamente diseñados para formar redes Ad Hoc que aplican algoritmos y protocolos de enrutamiento específicos para la formación de este tipo de redes. A pesar de que la mayoría se utilizan en laboratorios y centros de investigación, hay un caso muy particular y es el del proyecto *"One Laptop per Child" (OLPC)*[12] que busca crear laptops con fines educativos que tengan conexión a

Internet sin utilizar infraestructura para ello se utiliza la especificación, todavía en prueba, 802.11s que busca dar conectividad en forma de malla a una red inalámbrica, como se muestra en la Figura 1 en donde se observa que existen varios caminos para llegar a diferentes tipos de redes, incluyendo la misma Internet. El proyecto OLPC ha entregado aproximadamente 1124500 computadoras a países de África, América Latina y Norteamérica.

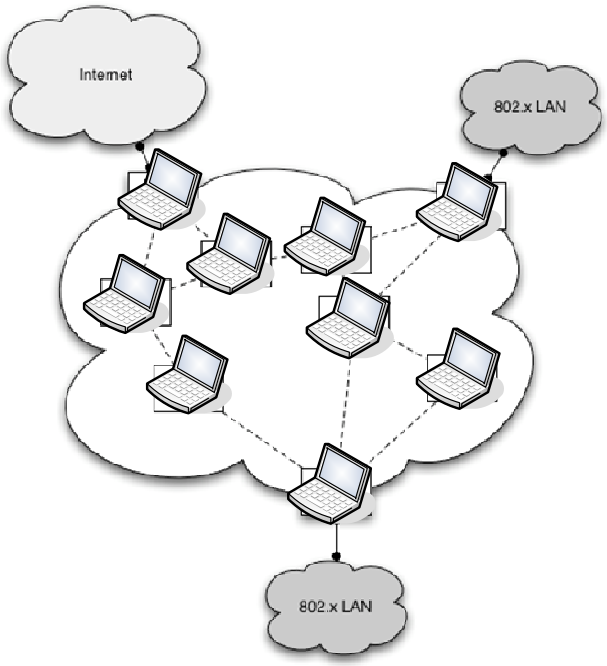


Figura 1.1 Topología en malla de una red Ad Hoc inalámbrica.

1.2.3 Aplicaciones

Si bien la creación de redes Ad Hoc parece complicada y destinada únicamente a investigación en laboratorios, la realidad es que hoy existen dos tipos de tecnología muy comunes que nos permiten crear tales redes. Wifi y Bluetooth se han convertido en dos estándares de comunicación bastante populares para comunicarnos inalámbricamente, siendo que actualmente estas dos tecnologías las tenemos en casi todas nuestras laptops o dispositivos móviles. Para el caso específico de Bluetooth tenemos que estudios [13][14][15] indican que para el 2008 existían 800 millones de dispositivos con este tipo de tecnología, que el 90% de los nuevos

dispositivos móviles fabricados ya la incluyen y que para el 2012 este mercado moverá 3100 millones de dólares.

1.3 Bluetooth

1.3.1 Definición

Es una especificación abierta de tecnología de radio de corto alcance, bajo costo y consumo de potencia para comunicación inalámbrica en redes ad hoc de voz y datos. Existen varias historias acerca del porqué de su nombre, pero una de las más difundidas indica que es debido a una analogía con el rey Harald Blatand quién unificó Dinamarca y Noruega, asimismo está tecnología pretende unificar al campo de las telecomunicaciones con la computación. Sea cual fuere la razón de su nombre, su especificación tiene varios puntos positivos que la convierten como un estándar prometedor para las redes ad hoc.

Tabla 1.2 Comparación entre las tecnologías inalámbricas de corto alcance

Característica	IrDA	WLAN	Bluetooth
Tipo de Conexión	Infrarroja, de banda angosta con línea de vista	Espectro Expandido	Espectro Expandido con QoS
Potencia de Tx	40-500 mW	100 mW	1-100 mW
Tasa de Transmisión	9600 bps	11-400 Mbps	1-54 Mbps
Alcance	1 m	100-300 m	10-100 m
Canales de Voz	No es posible	VoIP	3
Direccionamiento	Dirección física de 32-bit	MAC de 48-bit	MAC de 48-bit

Actualmente existen tres tecnologías de corto alcance en el mercado y una en desarrollo*. En la tabla 1.2 se muestra una comparativa entre las tres tecnologías existentes en el mercado de comunicación de corto alcance, de donde vemos algunas características relevantes de la tecnología Bluetooth sobre las demás. La primera es sin duda el consumo de energía, puesto que puede llegar a ser hasta el 1% del consumo total que tendríamos si utilizásemos el estándar 802.11a/b/g/n en

WLAN para una distancia menor a cien metros, esto representa una gran ventaja al momento de ser aplicado en dispositivos móviles con baterías más pequeñas puesto que les da mayor autonomía. Otro punto a favor es el que utiliza la comunicación por radio por espectro expandido pero con calidad de servicio (QoS) lo cual garantiza la confiabilidad en la transmisión de datos, esto fue definido desde un principio para la transmisión de voz esto a diferencia de WLAN, en donde no se ofrece calidad de servicio a nivel de enlace. Las ventajas de Bluetooth sobre la tecnología de comunicación infrarroja abundan, empezando por que no es necesario tener línea de vista para la comunicación, de igual manera que la distancia de comunicación es mucho más amplia pero la ventaja fundamental será la que Bluetooth permite tener funciones de formación de redes que la tecnología infrarroja no tiene.

Existen características especiales sobre la tecnología Bluetooth. Es así por ejemplo, que el concepto de red de área personal (PAN) fue introducido a la sociedad con la salida de esta tecnología al mercado y fue ahí cuando se marcó un hito para la computación ubicua y la formación de redes Ad Hoc. Por ello, actualmente se forman millones de redes Ad Hoc día a día con celulares que tienen Bluetooth habilitado para la transmisión de voz y datos con una increíble naturalidad y sencillez. Formar una red Ad Hoc con una laptop bajo Windows y utilizando la tecnología Wifi puede resultar complicado para un usuario novato pero es diferente el formar redes con celulares para compartir archivos usando Bluetooth, es por eso la razón de la popularidad de la tecnología.

Otro aspecto importante de la tecnología Bluetooth es el reconocimiento de usuarios de los servicios que ofrece. No existe tecnología que ofrezca algo similar, que por un simple escaneo podemos conocer a los potenciales nodos de la red, sus características y los servicios que puede ofrecer. Debemos recordar que Bluetooth no utiliza direcciones IP sino nombres para diferenciar a los dispositivos ya que la transmisión la hace en la capa de enlace.

Esta sección pretende dar una introducción sobre la tecnología. Detalles teóricos de la tecnología en la formación de redes y prestación de servicios se darán en el capítulo III que trata totalmente sobre la tecnología Bluetooth.

1.3.2 Historia

En un principio se buscó que mediante esta tecnología una persona pudiese utilizar todos los accesorios de su celular sin la necesidad de cables. Por ello, la empresa Ericsson para 1994 empezó a estudiar alternativas en tecnología y desde un principio se decidió que la tecnología debía tener un estándar abierto y debía usar radiofrecuencia para evitar que se tuvieran las limitaciones que hasta entonces tenía la tecnología infrarroja.

Para 1998, se crea el grupo de interés especial, SIG por sus siglas en inglés, para el desarrollo de la tecnología. A Ericsson se le sumaron Intel, IBM, Nokia y Toshiba para formar el SIG. Se mantiene la idea de tener un estándar abierto e invitan a otras compañías a unirse al grupo como miembros “promotoras”, lo cual les permitía obtener ventajas sobre los avances en la tecnología pero no eran miembros fundadores.

En el año de 1999 se suman al SIG 3Com, Agere, Microsoft y Motorola. Además se crea la categoría de miembro adoptivo, la cual es una asociación sin costo que le permite a una empresa certificar sus productos y promoverlos en el mercado; los miembros adoptivos tienen la posibilidad de certificar sus productos sobre la tecnología Bluetooth y comentar sobre la tecnología existente, sin embargo no reciben detalles sobre los últimos avances en la tecnología. Para este año la versión 1.1 de la tecnología permitía alcanzar velocidades de hasta 700 Kbps.

Con la salida al mercado de la versión 2.0+EDR de Bluetooth, para el año 2007, los dispositivos tienen la posibilidad de alcanzar hasta 3 Mbps a distancias de hasta 10m. Se define lo que es clase definida por la distancia y potencia de transmisión necesarias, como se indica en la tabla 1.3. De acuerdo a las clases

también se ve afectada la tasa de transmisión, siendo que a mayor distancia se tienen velocidades bastante a aproximadas a 700 Kbps.

Tabla 1.3 Clases de transmisión Bluetooth

Clase	Máxima potencia (W)	Distancia (m)
Clase 1	100 mW	100
Clase 2	2.5 mW	22
Clase 3	1 mW	6

En el año de 2008 se marca un hito en el número de miembros del SIG de Bluetooth, llegando a tener 12000 compañías como miembro entre adoptivos y promotores a nivel mundial. Así el SIG de Bluetooth se convirtió en el grupo de promotor más numeroso para una tecnología inalámbrica existente [16].

Es así que este año tuvo varias novedades en lo que respecta a la tecnología. Primero, se presentó el primer estándar para Bluetooth de bajo consumo de energía a utilizarse en sensores de automatización y para domótica basado en la tecnología Zigbee adquirida por Bluetooth en el año de 2006. Además, se presenta la versión 3.0 + HS que puede alcanzar velocidades de hasta 54 Mbps a una distancia de aproximadamente cinco metros y se espera que al menos el 1% del total del mercado de Bluetooth ya tenga incorporada esta tecnología hasta el primer trimestre del 2010 [17].

Actualmente el SIG está conformado por Ericsson, Intel, Lenovo (reemplazo de IBM), Microsoft, Motorola, Nokia y Toshiba como miembros asociados y más de 12000 empresas como miembros adoptivos y promotores a nivel mundial [16].

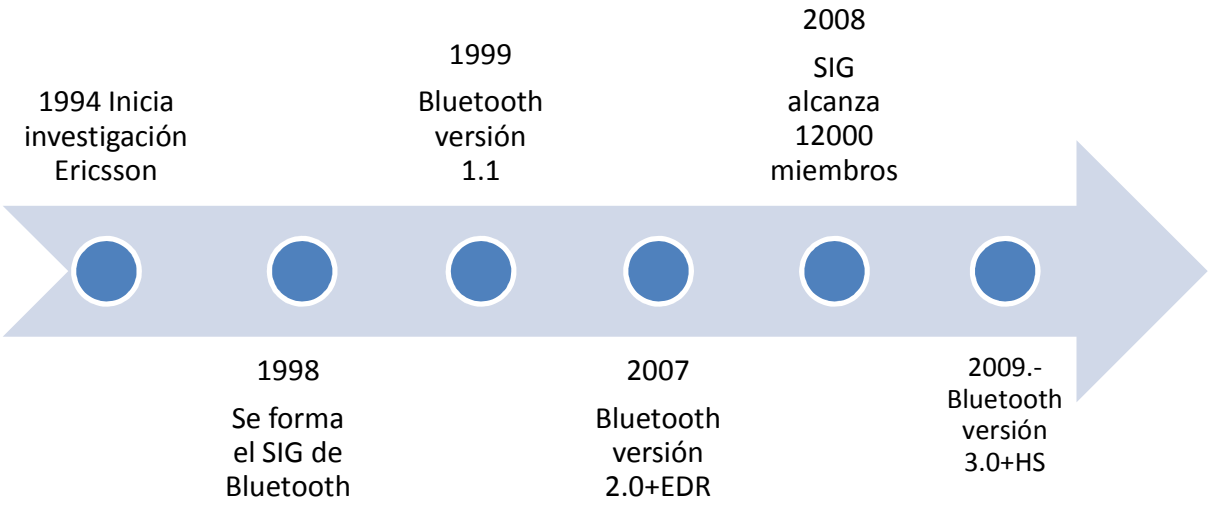


Figura 1.2 Línea de tiempo con aspectos relevantes sobre la tecnología Bluetooth.

Hasta el día de hoy Bluetooth es una tecnología de estándar abierto, pero existen especificaciones para los fabricantes de dispositivos que incluyan esta tecnología para poder asegurar la total compatibilidad en la comunicación de dispositivos de diferentes marcas.

1.3.3 Especificación Bluetooth

Se dan como resultado de un trabajo conjunto de las compañías miembro del SIG de Bluetooth y básicamente define el comportamiento completo del sistema (desde la radiofrecuencia hasta la capa de aplicación) para asegurar compatibilidad entre los dispositivos de diferentes fabricantes. Una especificación siempre es bastante amplia por lo que es pensada en ofrecer total compatibilidad.

Las especificaciones permiten conocer los perfiles que son aceptados y bajo los cuales se trabaja. Todo producto debe pasar por los perfiles para poder ser aprobado, a este proceso se le conoce como calificación. Los perfiles y las especificaciones se dan en forma de capas como el modelo OSI, como se muestra en la figura 1.3. En el capítulo 3 se da más información sobre la arquitectura de pila

Bluetooth y los perfiles que se tienen. Cabe destacar que todo lo que está por arriba de la interfaz HCI (Host Controller Interface) es perteneciente al host, es decir el software implementado para el dispositivo; mientras que lo que está por debajo de la interfaz HCI es básicamente hardware que permite la conectividad Bluetooth.

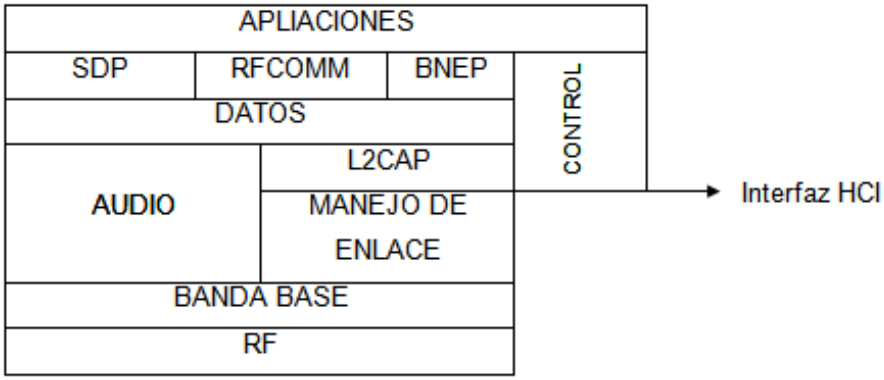


Figura 1.3 Pila de protocolos para Bluetooth

Como ya se mencionó anteriormente, actualmente tenemos las especificaciones para versiones 1.1, 2.1 +EDR y 3.0+HS siendo que cualquiera puede ser clase 1, 2 o 3 de acuerdo a las necesidades de distancia y potencia para la comunicación.

1.3.4 Aplicación

Una de las principales ventajas de tener un estándar abierto es que las personas pueden colaborar para mejorarlo y más aún trabajar sobre él para hallar nuevas aplicaciones. Atrás quedaron los días en los que la tecnología Bluetooth servía simplemente para conectar los audífonos a los celulares, a pesar de que aún es la aplicación más utilizada globalmente [17]. Hoy en día las aplicaciones de la tecnología Bluetooth se dan en las siguientes áreas:

- Salud
- Seguridad
- Marketing
- Automotriz
- Comunicación
- Automatización Industrial

- Domótica

De donde a continuación se muestra una lista con las aplicaciones más novedosas y sobresalientes, según la revistas SIGnature publicadas por el SIG de Bluetooth [17] para el primer y segundo cuartos de este año.

1. SYNC por Microsoft y Ford que permite dar comandos por voz en inglés a ciertos modelos de autos Ford para hacer llamadas por teléfono, obtener información de GPS y dar órdenes básicas de operación al auto.

Figura 1.4. Sistema SYNC de Ford y Microsoft.

2. El departamento de policía de Banglore en India utiliza dispositivos móviles Blackberry y conectividad Bluetooth para consultar antecedentes criminales de infractores mediante el acceso a una base de datos. Además, pueden agregar fotografías sobre nuevas infracciones a la base de datos.

Figura 1.5. Sistema de consulta móvil del departamento de policía de Banglore, India

3. La especificación Bluetooth de bajo consumo de energía será aplicada en sensores que determinar cuándo se otorgan puntos en los torneos de Taekwondo para las Olimpiadas del 2012

Figura 1.6. Sensores para determinar puntos en Taekwondo.

Como se mencionó anteriormente, Bluetooth consta básicamente de dos zonas en su pila de protocolo. La una está perfectamente definida por la parte de hardware para el equipo que permite obtener el medio de comunicación, mientras que la otra parte está conformada por el software que se monta sobre la estructura física que permite la comunicación. Sobre la parte de programación y software es que es necesario utilizar un lenguaje robusto y estable; es entonces ahí que entra el lenguaje Java, sumándole además la popularidad que tiene sobre los dispositivos móviles, que ya se discutió anteriormente.

1.4 Java

1.4.1 Definición

Es un lenguaje de programación orientado a objetos que elimina ciertas herramientas de bajo nivel para agilizar y simplificar la programación. Cualquier aplicación necesita de la máquina virtual de java (JVM) para ser ejecutada, esto le da cierta independencia del sistema operativo que use. Gracias a la JVM, este lenguaje ganó gran popularidad entre los dispositivos electrónicos ya que ocupaba escasos recursos en hardware al momento de implementar aplicaciones.

El lenguaje y su desarrollo ha está basado en cinco principios:

- Ser simple, orientado a objetos y familiar.
- Ser robusto y seguro.
- Ser de arquitectura neutral y portable.
- Tener un alto rendimiento al ejecutarse
- Ser interpretado, multifuncional y dinámico.

1.4.2 Historia

James Gosling inició con el desarrollo del lenguaje bajo el nombre de “Proyecto de Lenguaje Java” y durante los años noventa fue grandemente desarrollado bajo el concepto de “*Escribe una vez corre donde sea”* ó “*Write Once, Run Anywhere”* en inglés. La idea básicamente era darle independencia del sistema operativo que se use y del hardware a utilizar mediante la implementación de una máquina virtual que otorgaba todos los recursos necesarios para que el programa se ejecute sin problemas.

La mayor parte del desarrollo fue de Sun Microsystems hasta 1995, lo cual ya incluía el compilador, la máquina virtual y las librerías. La especificación fue controlada por Sun hasta el año de 2006 cuando la empresa liberó la mayor parte de las tecnologías java bajo la licencia GNU GPL, lo cual lo convierte en software libre.

El lenguaje ha tomado gran popularidad básicamente en dos segmentos, el uno relacionado con la electrónica y dispositivos de recursos limitados y el otro el segmento de manejo de bases de datos, donde existen herramientas de gestión como JBOSS [19] que agilitan los procesos de manejo de la información.

Actualmente existen varias plataformas de desarrollo para este lenguaje, algunas son totalmente gratuitas, pero predomina la herramienta de desarrollo desarrollada por Sun Microsystems llamada Netbeans [20] cuya interface se puede observar en la figura 1.7. Por otro lado también tenemos la suite de desarrollo de código abierto llamada Eclipse [21] que también es grandemente aceptada por los programadores a nivel mundial y cuya interface se puede observar en la figura 1.8. Ambas herramientas son poderosas y ofrecen funciones similares que permiten desarrollar aplicaciones en Java de manera rápida y segura.

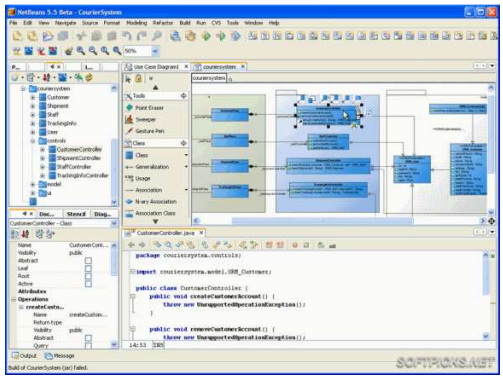


Figura 1.7 Interface gráfica de Netbeans

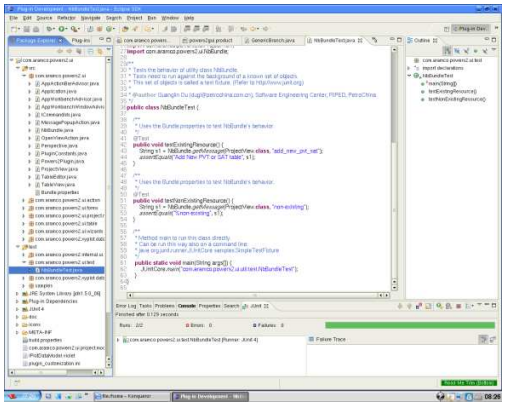


Figura 1.8 Interface gráfica de Eclipse

Actualmente se ha dividido al lenguaje Java en varias versiones de acuerdo a las aplicaciones que se le vaya a dar. Esto ha permitido un mayor desarrollo de las herramientas específicas en cada una de las distribuciones.

1.4.3 Entornos de Desarrollo

Cuando se lanzó el nuevo estándar denominado Java 2 en 1998 se crearon tres diferentes entornos para desarrollo y ejecución de aplicaciones:

- J2SE (Java 2 Estándar Edition).- Es un desarrollo de la versión inicial y permite el desarrollo de applets, pequeñas aplicaciones ejecutadas en un explorador web, y aplicaciones independientes.
- J2EE (Java 2 Enterprise Edition).- Está basado en la distribución estándar pero se añadieron características empresariales en campos

como redes, acceso de datos en bases de datos y entrada o salida de información que requieren de mayor almacenamiento, memoria y procesamiento para ser ejecutados.

- J2ME (Java 2 Micro Edition).- Es una versión de Java 2 para entornos más limitados como teléfonos celulares, agendas personales (PDA), etc. Los programas que se crean tienen similitud con los applets de J2SE.

Es así que desde la salida de J2ME empieza la revolución de aplicaciones java para celulares. Fue desde entonces que se empezaron a analizar las verdaderas características de los dispositivos móviles. Hasta el día de hoy tenemos cientos de miles de aplicaciones creadas en java para diferentes tipos de dispositivos móviles que van desde juegos en red hasta aplicaciones de redes sociales [22].

1.4.4 J2ME

La versión *portable* de Java 2 tiene diferencias básicas con respecto a la versión estándar que le permite tener la flexibilidad de operar en entornos limitados. Es así que las diferencias más importantes son que no existe soporte a las operaciones matemáticas en punto flotante, el método *finalize(),* utilizado para liberar memoria, tampoco se incluye en las librerías y se ven limitadas el número de excepciones que se pueden utilizar para el control de errores Además, el número de librerías en la distribución estándar de J2ME se limita a 30, aunque se pueden añadir complementos.

Los programas escritos en Java son generados en *bytecode y l*a máquina virtual (JVM) se encarga de interpretarlos al momento de la ejecutarlos. La máquina virtual asegura la funcionalidad de los programas a ejecutarse sin importar el sistema operativo para un grupo de dispositivos. Es entonces que cabe mencionar que debida a la diversidad de dispositivos que pueden ejecutar aplicaciones Java (desde Top Boxes para satélites hasta electrodomésticos), se han definido configuraciones y perfiles que aseguran la modularidad y funcionalidad de las aplicaciones que se pueden desarrollar.

1.4.5 Configuraciones

Permiten definir las mínimas capacidades de procesamiento para un conjunto de dispositivos. Están compuestos por la JVM y un conjunto mínimo de clases que le permiten ejecutar las aplicaciones según la clase de dispositivo. Actualmente tenemos dos tipos de configuraciones:

- Connected, Limited Device Configuration (CLDC).- Enfocada en el segmento de celulares, agendas personales y todo dispositivos con capacidades limitadas, que operan con baterías y que no están conectados todo el tiempo a una red local. En estos se implementa una versión reducida de la máquina virtual llamada KVM (Kilobyte Virtual Machine).
- Connected Device Configuration (CDC).- Está enfocado en dispositivos con mayores capacidades de conectividad y procesamiento como set-top boxes para televisores y dispositivos de comunicación de banda ancha. En esta categoría sí se implementa la JVM en su totalidad.

Siendo que actualmente todavía prevalecen los dispositivos móviles que incorporan la configuración CLDC en el mercado, esto en gran medida debido a que cuando se utiliza hardware de mayor capacidad con un sistema operativo incluido, se desarrollan lenguajes que aprovechan al máximo la funcionalidad del conjunto

1.4.6 Perfiles

Permiten añadir total funcionalidad a la interfaz de programación de aplicaciones (API) al añadir un entorno de ejecución más completo basado en características específicas. Entre los principales tenemos a *Mobile Information Device Profile (MIDP), Foundation Profile (FP) y Personal Profile (PP)*. A continuación se detalla un poco más acerca de cada uno de ellos:

- Mobile Information Device Profile.- Está diseñado para dispositivos móviles. Junto con CLDC ofrece un entorno completo para el desarrollo de aplicaciones, incluyendo capacidades de conectividad, interfaz de

usuario y almacenamiento. Las aplicaciones desarrolladas bajo este perfil son denominadas *MIDlet*.

- Foundation Profile.- Diseñado para dispositivos sin interfaz de usuario pero que tienen capacidades de conectividad. Es el perfil de más bajo nivel de CDC y se pueden añadir otros sobre él.
- Personal Profile.- Diseñado para dispositivos de mayor capacidad y permite migrar los programas directamente desde J2ME. Como ejemplos de dispositivos en esta categoría tenemos a módems de comunicación, consolas de juegos o top boxes de televisión.

<table>
<tr><td>Servidores y computadores empresariales</td><td>Computadores personales y de escritorio</td><td>Dispositivos de alto rendimiento terminales</td><td>Dispositivos con capacidades limitadas</td><td>Tarjetas Inteligentes</td></tr>
<tr><td>Paquetes opcionales</td><td rowspan="2">Paquetes opcionales</td><td>Paquetes opcionales</td><td rowspan="3">Paquetes opcionales</td><td rowspan="3"></td></tr>
<tr><td rowspan="3">Java 2 Platform, Enterprise Edition (J2EE)</td><td rowspan="3">Java 2 Platform, Standard Edition (J2SE)</td><td>Personal Profile</td></tr>
<tr><td>Foundation Profile</td><td>MIDP</td></tr>
<tr><td>CDC</td><td>CLDC</td><td>Smart card profile</td></tr>
<tr><td colspan="3">JVM</td><td>KVM</td><td>Card VM</td></tr>
</table>

Figura 1.9 Plataformas en Java. Zona azul indica J2ME

En la figura 1.9 se muestra una comparativa entre las diferentes plataformas de desarrollo de Java, de donde vemos que la zona azul representa los elementos que conforman a J2ME.

Para el año 2002 Java se convirtió en el aliado principal para el desarrollo de aplicaciones con conectividad inalámbrica. Especial interés para Bluetooth tiene la

recomendación JABWT que define las clases específicas para hacer desarrollo de aplicaciones en Bluetooth. Sobre el funcionamiento, clases y escritura de código sobre JABWT se hablará completamente en el capítulo IV.

CAPÍTULO 2

REDES AD HOC

1. Introducción

El objetivo de la computación ubicua se ha definido como el de proveer procesamiento de la información del entorno de una persona de manera que la misma no lo note ni tenga que cambiar su comportamiento para obtener resultados que le sean útiles en el momento [23]. Para esto, la computación ubicua utiliza tecnologías, algunas aún en desarrollo, que involucran aspectos que van más allá de calidad de servicio en la comunicación o eficiencia en la transmisión de datos e incluyen al consumo de energía, aprendizaje de ruta y algoritmos "inteligentes" como características relevantes a ser consideradas en su desarrollo [24]. Es así que el desarrollo de redes Ad Hoc se fundamenta en tecnologías inalámbricas como IEEE 802.11 a/b/g/n o Bluetooth y en algoritmos de enrutamiento bastante complejos pero que proporcionan la naturalidad de uso que el usuario requiere, todo esto aplicado a dispositivos móviles que se deben adaptar a la rutina de las personas.

Este capítulo trata completamente de las redes Ad Hoc, desde sus fundamentos hasta las tecnologías utilizadas actualmente para formarlas. Se revisará varios algoritmos de enrutamiento propuestos [25] y se comprenderá las diferencias entre los mismos para poder estudiar su aplicabilidad en dispositivos móviles en futuros capítulos. Finalmente, se dará una pequeña introducción sobre las tecnologías utilizadas en este tipo de redes y de las aplicaciones existentes sobre las mismas, sobre todo las que ya están en el mercado.

2. Fundamentos

Con la masificación del acceso a Internet en los años noventa se produjo un gran intercambio de información entre las personas que accedían mediante sus computadoras. Pronto las redes de área local (LAN) pasaron de estar únicamente en universidades a oficinas y a los hogares de las personas. El aparecimiento de Wifi a inicios de este siglo produjo que las redes de área local inalámbricas (WLAN) se volvieran cada vez más populares y con ello se facilitó el acceso a internet en sitios públicos así como en hoteles, aeropuertos, cafeterías y otros sitios que ofertaban el acceso a internet de manera gratuita e inalámbrica. Sin embargo, estas redes aún tienen una interacción con el usuario que no es natural y que hace que la misma tenga que aprender, retener y, solo a veces, entender los comandos que ejecuta en su computadora personal. Es así que los sistemas operativos diseñados en los ochenta no fueron pensados en adaptarse al comportamiento humano, sino más bien, por las limitaciones de la época, en facilitar los procesos matemáticos con el ingreso de información por periféricos que requerían de cierto entrenamiento.

Casi siempre se relaciona a la computadora personal con redes de área local de cable o inalámbricas y cualquier persona que desee acceder a ellas debe al menos comprender algo de sistemas operativos y de periféricos como ratón o teclado de lo contrario no podrá acceder a la computadora, a la red y por tanto a la información. Se dio una primera solución a este inconveniente con la salida al mercado de los celulares que incluían nuevos sistemas operativos que debían conservar la naturalidad de manejar un teléfono pero que además tenían características multimedia más avanzadas, tal es el caso se Symbian OS[26]. De igual manera con la inclusión de Bluetooth en los dispositivos móviles la gente empezó a formar redes de área personal (PAN) que le permitían intercambiar cualquier tipo de información a menor distancia pero que dejaban de lado dificultades relacionadas al sistema operativo y a direccionamiento, tal es el caso que se pasó de enviar un archivo a una dirección de protocolo internet (IP) a el nombre de un

dispositivo, que muchas veces era el nombre de la persona a la que se enviaba la información.

En la actualidad, para hacer realidad la computación ubicua se debe considerar el concepto de red de área corporal (BAN) que involucra la distribución de lo que hoy conocemos como periféricos de la computadora en el cuerpo de la persona [27]. Con esto, los periféricos intercambiarán constantemente información del entorno en el que se encuentra una persona, con fines específicos, sin modificar el comportamiento de la misma. Es así que aparece el concepto de A*mbient Intelligence* [28] que tiene como objetivo el de tener al ser humano en el centro de la información y que busca el integrar varios dispositivos digitales del entorno para obtener información que le sea útil a la persona El estudio de *Ambient Intelligence* es bastante amplio y no se profundizará pero es necesario aprender que su paradigma está orientado al tener información y comunicación que se ordena por sí misma mediante el intercambio de datos entre dispositivos inteligentes en nuestro entorno.

Es en la aplicación de toda la teoría sobre computación ubicua donde aparece el concepto de redes Ad Hoc. Sobre todo de las MANETs por su versatilidad y la poca necesidad de infraestructura que requiere para formarlas, además será necesario profundizar los estudios que tiene sobre la adaptabilidad de estas redes ante cambios y la rapidez con la que lo hacen, ya que resulta necesario mantener la fiabilidad del intercambio de información entre los diferentes dispositivos. En la Figura 2.1 se muestra una sencilla aproximación a la formación de una MANET.

Se cual fuese el caso, las redes de área local no encajan perfectamente en el campo de la computación ubicua, a pesar de que pueden ser usadas para sus fines. Por ello, el estudio debe centrarse en el enrutamiento de nodos para redes centradas en las persona y que recojan información acorde a sus necesidades de manera simple, como las PAN y BAN.

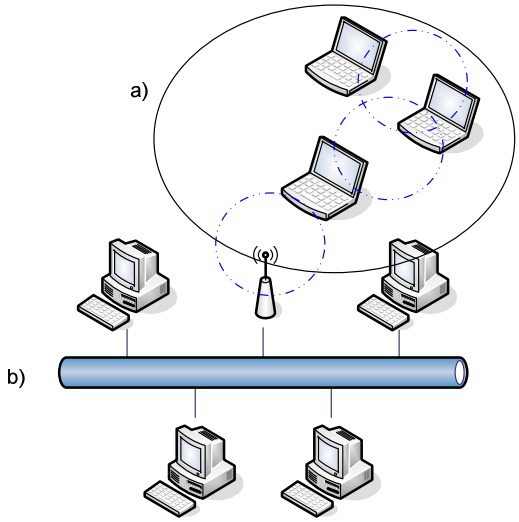


Figura 2.1 a) Ejemplo de red Ad Hoc. b) Red de Infraestructura

2.1. Redes de área personal (PAN)

El concepto de este tipo de redes nace de permitir que una persona pueda conectar la mayor cantidad de dispositivos electrónicos, conocidos como *gadgets,* a su celular de manera simple. La popularización de las PAN se dio gracias a Bluetooth y a la sencillez que se daba para conectar varios accesorios a los celulares.

Las PAN son también conocidas como WPAN (*Wireless Personal Area Network*) que hace más evidente la característica inalámbrica de la red pero que indican el mismo concepto de conectar dispositivos cercanos a la persona, establecido como el estándar IEEE 802.15. El alcance puede variar dependiendo de la tecnología que se utilice [29] pero generalmente permiten alcanzar desde diez a treinta metros. También de acuerdo a la tecnología se tiene la frecuencia a utilizar, aunque 2.4 Ghz parece ser la más prometedora y la que más avances ha tenido gracias a la tecnología Bluetooth de espectro ensanchado que permite evitar la interferencia con otras señales en la misma frecuencia y aprovechar el ancho de

banda [30]. En la tabla 2.1 se presenta una comparativa entre tres diferentes tipos de tecnología definidas en IEEE 802.15.

Tabla 2.1 Comparativa entre diferentes tipos de tecnología para WPAN

	Bluetooth 802.15.1	**UWB 802.15.3**	**ZigBee 802.15.4**
Frecuencia	2.4 - 2.48GHz	3.1-10.6GHz	868MHz 902-928MHz 2.4-2.48GHZ
Alcance	10 m	10 m	100 m
Tasa de Transferencia	3 Mbps	1 Gbps*	40 – 50 Kbps
Modulación	GFSK	BPSK	BPSK

Los avances que se han dado sobre esta tecnología radican en la flexibilidad y sencillez de formar redes pequeñas. Es así que se han desarrollado varias herramientas que van desde auriculares para PC hasta puntos de acceso para juegos en plazas y parques [31] [32]. Es así que existen cinco grupos de trabajo para este estándar:

- Grupo de Trabajo 1.- Que trata sobre WPAN bajo Bluetooth.
- Grupo de Trabajo 2.- Que trata sobre la coexistencia de los diferentes tipos de tecnología existentes en esta banda.
- Grupo de Trabajo 3.- Enfocado en el estudio de WPAN de alta velocidad para aplicaciones multimedia
- Grupo de Trabajo 4.- Enfocado en el estudio de WPAN de bajas velocidades que permiten el ahorro de energía.
- Grupo de Trabajo 5.- Funcionamiento de la tecnología en redes en malla.

De donde en conjunto establecen los parámetros de funcionamiento para WPAN sea cual fuese la tecnología procurando la interoperabilidad entre ellas y la compatibilidad con tecnologías de mayor alcance como WLAN o WMAN.

Este tipo de redes permitirá que una persona pueda estar comunicada totalmente con su entorno y obtener información sobre el mismo. Actualmente

existen varios productos que permiten interactuar con el celular de una persona; por ejemplo, una cerradura que interactúa con un algoritmo generado por un teléfono celular con bluetooth activado [33]. Su versatilidad permite que hoy sean considerados nuevos conceptos como redes de área de hogar (HAN) o que en industrias los sensores puedan formar WPAN como nodos que informan todo el tiempo sobre la producción.

2.2. Red de área corporal (BAN)

Este tipo de redes se ha definido para que no tenga un alcance más allá de uno a dos metros y algunos conceptos más ambiciosos incluyen utilizar al cuerpo humano como medio de transmisión de la información entre los diferentes nodos [34].

Este tipo de redes se apegan totalmente a la computación ubicua ya que busca que los nodos se encuentren distribuidos en el cuerpo de la persona intercambiando información constantemente. Por ello, podemos mencionar que los requerimientos más importantes para este tipo de redes son:

- Habilidad para interconectar dispositivos heterogéneos.- Desde dispositivos completos (miniPC) hasta partes de uno (micrófono).
- Autoconfiguración.- La conexión debe ser transparente al usuario
- Integración de servicios isócronos (transmisión de información a intervalos constantes) deben coexistir con transmisiones de internet (no en tiempo real) para lograr obtener y enviar información útil del y hacia el usuario.

Además, Philip R. Zimmerman, un estudioso sobre el tema, ha escrito sobre que también resulta necesario tomar en cuenta la energía a utilizarse. Así Zimmerman va aún más allá indicando que un billonésimo de amperio debe ser suficiente para proveer la transferencia de datos; en otras palabras, un simple apretón de manos debe ser capaz de iniciar una transmisión [35].

Analizando a este tipo de redes desde un punto de vista más integral, resulta evidente que al conectar una BAN con su entorno estaríamos formando una PAN y de igual manera, múltiples PAN formarían una LAN que podría tener una ruta de

acceso o *gateway* por la cual salir a una MAN o directamente a Internet. Esa es la versatilidad que se tiene en esta tecnología formada nodos inteligentes.

Actualmente, una de las tecnologías que ha tenido mayor respaldo para ser adoptada como estándar al formar redes BAN es Bluetooth de bajo consumo de energía [36], la cual busca tener las mismas características de comunicación de la tecnología Bluetooth original pero con mucho menor consumo de energía. En la figura 2.2 se muestra una ilustración presentada en la revista SIGnature del último cuarto de 2010 sobre las posibilidades de utilizar sensores en la ropa de la gente para asistirla a tener una vida saludable.

Figura 2.2 Potenciales sitios para ubicación de sensores con Bluetooth.

Surgirán muchas más aplicaciones en el futuro relacionadas con mejorar el estilo de la vida de la gente relacionadas con BAN. Hoy en día existe un dispositivo fabricado por la empresa *Kickbee* [37] que envía un mensaje de texto o publica un mensaje en una red social cuando un feto patea dentro del vientre de una madre. Este dispositivo aún no es totalmente portable ni tampoco sus consumos de energía son bajos, pero está basado en los principios de la computación ubicua y de las redes BAN. En la figura 2.3 se presenta una gráfica sobre este dispositivo y su uso.

Figura 2.3 Dispositivo de monitoreo a un bebe dentro del vientre de la empresa *Kikckbee*

3. Redes Ad Hoc

A diferencia de una red tradicional en donde la infraestructura fija, como enrutadores o conmutadores, juega un papel preponderante, en este tipo de redes no existe y los nodos o terminales dinámicamente se ordenan arbitrariamente y temporalmente. La formación de este tipo de redes busca un objetivo específico y por un tiempo limitado. Su importancia radica nuevamente en la flexibilidad que tienen para adaptarse a entornos difíciles como en sitios inaccesibles, donde ha ocurrido una catástrofe o donde se deben realizar comunicaciones por tiempo limitado. Sus debilidades están directamente relacionadas con la falta de una administración centralizada y una topología que se encuentra cambiando todo el tiempo, lo cual produce que la calidad en la comunicación no sea buena.

Las redes Ad Hoc están basadas en tres principios: se crean por sí mismas, se organizan por sí mismas y se administran por sí mismas. Esto hace que el ofrecer calidad de servicio en la comunicación (QoS) para la transmisión de multimedia se convierta en todo un reto. Es así que encuentro adecuada una definición dada en por Satyabrata Chakrabarti y Amitabh Mishra en su publicación *Quality of Service in Mobile Ad Hoc Networks: "*Se crean para ofrecer servicios exclusivamente a las

interacciones de sus nodos, y solo tales interacciones dan la administración y control a tales redes.” [38]

Por la adaptabilidad que tienen en sus protocolos de enrutamiento y formación ofrecen resistencia a las fallas o caídas de nodos en entornos de transmisión complicados. Por tal motivo han sido objeto de estudio para comunicaciones militares, policía y agencias de rescate bajo condiciones hostiles como desastres naturales o guerras. Sin embargo, hoy en día se busca la aplicación de estas redes en oficinas y hogares; una clara aproximación a eso se ha dado gracias a las *Piconet*, que no son más que redes Ad Hoc formadas por dispositivos Bluetooth.

Las redes Ad Hoc aún tienen varios retos que enfrentar para resolver pendientes como alto retardo, bajo ancho de banda, consumo de energía de los nodos y bajo consumo de energía de los mismos. Si bien varios de los problemas existentes para estas redes están relacionados íntimamente con la naturaleza de su formación también es evidente que resulta necesario un protocolo eficiente de administración y enrutamiento que mejore el desempeño de las mismas sin sacrificar sus principios.

3.1 Funcionamiento básico de operación de una red Ad Hoc

Si tenemos un conjunto de nodos desde A hasta E, como se muestra en la Figura 2.4 y los nodos A y B desean comunicarse, tenemos que existe un solo salto por lo que una comunicación de radiofrecuencia cualquiera lo podría hacer.

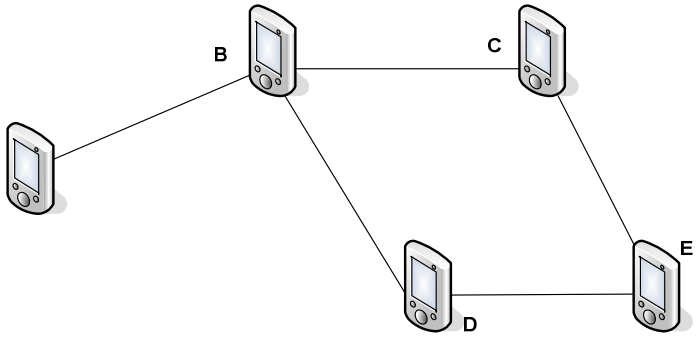


Figura 2.4 Ejemplo de topología de una red Ad Hoc

Si los nodos A y C desean iniciar una comunicación y están a una distancia tal que no sea posible establecer una comunicación directa, se deberá utilizar un camino intermedio, se da entonces el caso que el nodo B debe tener las funciones de enrutador.

Por tanto, resulta evidente que todos los nodos deben tener la capacidad de funcionar como enrutadores además de dispositivos finales. Además, resulta necesario evitar que los paquetes atraviesen infinitamente por varios caminos. Es así que un camino sin bucles entre un par de nodos se conoce como ruta.

Para un caso más general si tenemos k nodos denominados desde A hasta N y si se traza una ruta R(A,F) desde A hasta F existe una secuencia de nodos tal que:

$$R(A,F) \equiv |< N_0, N_1, ... N_K >$$

donde:

$$\begin{aligned} &N_0 = A \\ &N_K = F \\ &N_i \neq N_j \end{aligned} \qquad (N_i, N_{i+1}) \in (para\ i \neq j, 0 \leq i \leq k-1)$$

Que indican el camino R que se A debe tomar para llegar a su destino F. Donde la longitud de la ruta es:

$$\begin{aligned} &\left|R(A,F)\right| \\ &\left|< N_0, N_1, ..., N_K >\right| = K \end{aligned}$$

Y el valor de K indica el número de nodos que se recorren para llegar a un destino. Es así que para llegar al destino es necesario al menos cumplir con dos tareas:

1. El cálculo de la ruta R(A,F).
2. El envío del paquete por la ruta determinada.

Donde el cálculo de las rutas se da por algún algoritmo a nivel de capa de red. Ahora bien, si es que existen varias rutas entre un origen (A) y un destino (F) se debe calcular la mejor ruta basado en alguna métrica. Por ejemplo, si se usa la métrica de la distancia. Entonces:

$$R(A,F) = \min\left\{\left|R(A,F)\right|\right\}$$

Un nodo $N_i \neq F$ envía el paquete al siguiente salto mientras no se llegue al destino.

3.1.1 Formación de redes y establecimiento de rutas

La formación de una red Ad Hoc siempre comienza con al menos dos nodos haciendo conocer su respectiva información, a este proceso se lo conoce como *baconing*. Cuando un nodo puede establecer comunicación directa con otro los dos actualizan sus tablas de enrutamiento con la ubicación del otro nodo. Cuando un tercer elemento entra en la comunicación se puede dar uno de los siguientes casos:

- Los dos quieren establecer una conexión directa con el nodo al mismo tiempo.
- Solo uno busca establecer una comunicación con el nuevo elemento.

Pero cualquiera sea el caso, las tablas de enrutamiento se actualizarán inmediatamente en todos los nodos. En la figura 2.5 se muestra una ilustración del intercambio de información entre tres nodos, en un principio se produce una actualización únicamente entre los nodos A y B, mientras que posteriormente el nodo B manda una actualización al nodo A sobre la ubicación e información del nodo C. Al final todos los nodos conocen sus posiciones para poder intercambiar información.

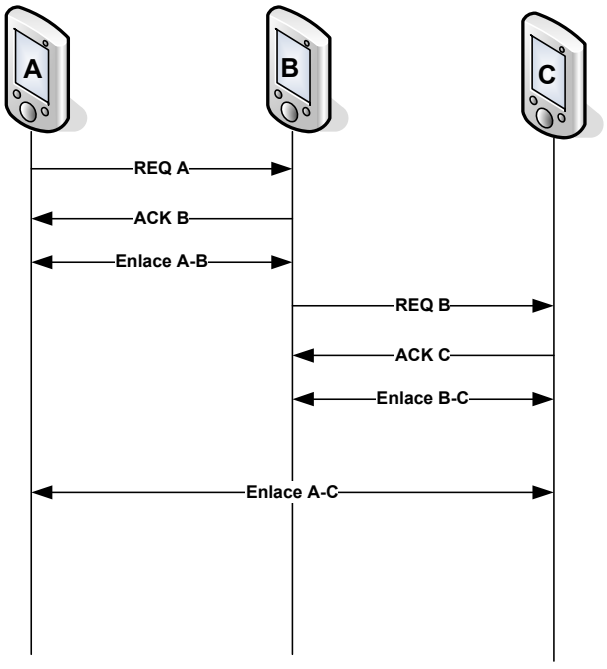


Figura 2.5. Establecimiento de enlace para un tercer nodo

En la figura 2.5 si el nodo C pudiese ser descubierto directamente por el nodo A, la información de B resulta innecesaria y debe desecharla; además, un protocolo de enrutamiento también resulta innecesario porque puede establecerse una comunicación directa entre los tres nodos.

La figura 2.4 nos muestra una estructura de red más compleja donde se puede analizar más casos de cambio de topología. Por ejemplo si el nodo C deja de ser visible por el nodo B y solo lo es por el nodo D y el nodo E entonces la información debe actualizarse en todos los nodos empezando primero por el nodo B y el nodo C, luego a el nodo A y al nodo E para finalmente actualizar al nodo D, con lo que se establece una nueva ruta.

Si aumentamos número de nodos la topología puede cambiar drásticamente y las actualizaciones se vuelven un problema. Es así que las actualizaciones pueden deteriorar drásticamente la comunicación cuando no funcionan correctamente. Por ejemplo, si un nodo envía un mensaje y aún no tiene actualizado en su tabla de enrutamiento un cambio en la topología o peor aún si la ruta alterna aún no ha sido hallada, el paquete se perderá inminentemente y la comunicación corre el riesgo de perderse inundando la red con mensajes erróneos. De la misma manera que para las redes con infraestructura es necesario que la red converja en el mínimo tiempo posible pero además es necesario que se de un comportamiento denominado como *combinatorially stable* (estable conjuntamente), que se da cuando los cambios en la topología se producen con suficiente tiempo para que las actualizaciones lleguen a todos los nodos. Por tanto, para redes Ad Hoc la estabilidad depende no solo de las condiciones de los enlaces sino de otros factores, entre los que se incluye la complejidad del protocolo de enrutamiento.

En la etapa de transmisión de datos se da por hecho de que la ruta existe y que fue la mejor elegida por el protocolo de enrutamiento. La pérdida de comunicación aquí se puede dar debido a la pérdida del enlace, desaparición de un nodo o a efectos externos inherentes a la distancia de comunicación.

Otro tema al que se le debe prestar atención es el de acceso al medio, puesto que no se puede manejar un protocolo de intercambio de información sobre el medio porque no usaría eficientemente los recursos limitados de ancho de banda de la red. Por ello, resulta necesario tener nuevos métodos de acceso al medio que permitan evitar las colisiones cuando los nodos necesiten intercambiar información.

3.1.2 Acceso al medio y Efectos HIDDEN/EXPOSED

Los casos de HIDDEN/EXPOSED ocurren en la etapa de transmisión de datos y se dan debido a errores en el control de envío de datos. Se puede tomar como ejemplo la figura 2.6, en donde existe una barrera entre el nodo B y el nodo D, tal barrera puede ser física o una distancia que impida que ambos nodos tengan comunicación directa. Es así que puede darse el caso de que el nodo B y el nodo D transmitan al mismo tiempo información al nodo C, dado que ninguno de los dos nodos está consciente de la existencia del otro sino únicamente de cómo llegar a ese destino si se desea, en este caso se produce un efecto de terminal oculto (*hidden terminal*). Por otro lado, el nodo C está transmitiendo al nodo D y el nodo B sabe de ello; entonces el nodo B no transmite al nodo A por evitar una colisión, este es el caso de terminal expuesto (exposed terminal).

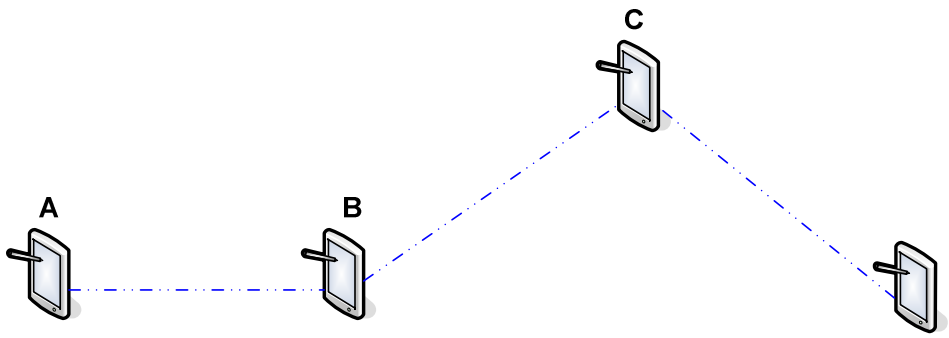


Figura 2.6 Ejemplo de caso de efecto de terminal oculto/expuesto.

Cualquiera de los dos efectos puede ser resuelto con un simple protocolo de intercambio de mensajes, que involucre dos estados:

- Petición de envío (RTS)
- Libre para envío (CTS)

Así por ejemplo, cuando un estado CTS se envía a los nodos B y D por el nodo C, pero el nodo no ha B enviado una petición de envío de información, entonces el nodo B puede saber que ese mensaje va para otro nodo del cual desconoce que no es el nodo A, entonces no transmite nada al nodo C pero puede enviar libremente información al nodo A.

Este tipo de señalización es un primer acercamiento al establecimiento de un método de control de acceso al medio (MAC) eficiente para las redes Ad Hoc. Satyabrata Chakrabarti y Amitabh Mishra tienen varios estudios sobre el control de acceso al medio para este tipo de redes [38], que describen la importancia de tener un método de acceso al medio que facilite el trabajo de enrutamiento de los nodos pero sin dejar de lado la importancia de un protocolo de enrutamiento robusto que permita mejorar la calidad en la comunicación en este tipo de redes

3.2 Enrutamiento para una red Ad Hoc

Cabe recalcar que, al igual en para una red con infraestructura, los protocolos de enrutamiento pueden ser tanto para un solo destinatario, conocidos como *unicast*, cuanto para varios destinatarios, en cuyo caso son conocidos como *multicast;* pero el estudio estará centrado únicamente en los protocolos *unicast* debido a que son los más apropiados al momento de desarrollar aplicaciones de mensajería y comunicaciones. Las comunicaciones *multicast* pueden ser analizadas para transmisión de multimedia, el cual no es el caso de esta tesis. Para obtener información de protocolos *multicast* recomiendo leer estudios realizados por Xiao Chen y Jie Wu, donde se trata el tema a profundidad [38].

3.2.1 Retos de los protocolos de enrutamiento para redes Ad Hoc

Además del compromiso entre complejidad y eficiencia que tienen los tradicionales protocolos de enrutamiento para redes de infraestructura, los protocolos para las redes Ad hoc deben considerar al menos los siguientes aspectos:

- Los cambios en la topología se dan constantemente y hacen que se disparan actualizaciones a todos los nodos en los protocolos que intercambian actualizaciones periódicas. Este intercambio de información produce que se saturar la red, haciendo que se caiga. Por ello, el protocolo de enrutamiento debe tener una forma de actualización que sea eficiente para evitar que los paquetes sean mandados por rutas incorrectas o más aún perdidos pero además con actualizaciones que eviten la saturación de la red.
- Existen limitaciones en aspectos como potencia de batería, ancho de banda de transmisión y tiempo de uso del procesador. Ya que las redes Ad Hoc que se forman con dispositivos móviles con dispositivos móviles deben conservar su batería por lo que se debe evitar actualizaciones que gasten la batería, como por ejemplo actualizar rutas que no se estén usando.
- Se debe considerar una métrica conjunta con varios aspectos no considerados antes como la batería de los equipos involucrados en la ruta, confiabilidad del enlace y equidad en el uso de las rutas. Por ejemplo, no conviene usar una ruta de pocos saltos si es que los enlaces no son confiables o los nodos involucrados tienen baja su batería.
- La Seguridad se vuelve un aspecto sumamente importante pues la información recorre indistintamente diferentes nodos, lo cual facilita la intromisión de nodos que perjudiquen la red, ya sea robando información o mandando información de enrutamiento incorrecta.

También cabe recalcar que los nodos se comunican de manera peer-to-peer (p2p) ya que funcionan como sistemas finales y como enrutadores a la vez por lo que es muy importante tomar en cuenta el nivel de carga de procesamiento de los

protocolos y la potencia en la transmisión para maximizar el tiempo y la calidad de comunicación en la red.

Tampoco se debe olvidar que el protocolo de enrutamiento debe considerar el control de acceso al medio. Por ello, se deben crear algoritmos para recuperación inmediata y eficiente de las inevitables colisiones de paquetes. Por ello este tipo de redes son denominadas como *Best Effort* service debido a sus limitaciones y retos, debido a que aún en las simulaciones más optimistas el nivel de cambio en la topología con un bajo nivel de nodos es un reto de enrutamiento [38].

3.2.2 Tipos de enrutamiento para redes Ad Hoc Unicast

Todos los protocolos de enrutamiento para formar redes Ad Hoc deben ejecutar un conjunto de instrucciones básicas para identificar una ruta. Todas las posibles rutas son conocidas únicamente como sugerencias, debido a que se dan reconfiguraciones de rutas por efectos como pérdidas de enlace o congestión de tráfico. Es así que los protocolos tienen un conjunto de funciones para garantizar en lo posible la prestación de servicios.

Se han propuesto varios protocolos de enrutamiento para las redes Ad Hoc, la mayoría aún en estudio y debate pero la IETF [17] permite hacer una identificación de los protocolos dividiéndolos en aquellos que están basados en los cambios de topología y en aquellos que están basados en la ubicación de los nodos. Así tenemos:

- Basados en topología.- Funcionan de igual manera que en los protocolos de las redes cableadas en donde se da especial énfasis a los estados de enlace. Generalmente se da la clasificación a los protocolos de enrutamiento como:
  - Periódicos (Proactivos).- Los cuales intercambian información sin importar si las rutas están transmitiendo datos. Cada nodo tiene la

información de enrutamiento necesaria y es responsable de enviar las actualizaciones debidas. Tiene como desventaja el que se produce un gran desperdicio de ancho de banda cuando se producen muchos cambios de topología.

- Bajo demanda (Reactivos).- Se crean rutas únicamente cuando se necesita transmitir información. Se necesita un proceso para descubrir una ruta y esta se mantiene mientras sea necesaria. En estos protocolos también se produce gasto de ancho de banda y pérdidas de paquetes cuando la topología cambia. Su desempeño mejora notablemente cuando no hay muchos cambios en topología.
- Híbridos.- Estos protocolos toman ciertas características de los dos anteriores.

- Basados en ubicación.- Estos protocolos además de la información requerida por topología se obtiene información sobre la ubicación física del nodo, para ello casi siempre se utiliza el sistema de posicionamiento global (GPS). El nodo a transmitir debe usar un servicio para hallar su ubicación luego la de su destino y envía tal dirección como parte de sus mensajes. Una tabla de enrutamiento no resulta necesaria y los nodos adyacentes son identificados por el alcance que tenga el nodo. El mensaje *beaconing* incluye el alcance, cualquier nodo más allá de los límites descarta el mensaje.

  Para este tipo de protocolos se necesita tener exactitud en tiempos y ubicaciones de los nodos, por ello resulta eficiente el establecer varios nodos de ubicación descentralizados para dar la información a los demás.

Al igual que para las redes con infraestructura, las redes Ad Hoc tienen principios de funcionamiento similares al momento de que un nodo envía actualizaciones a sus vecinos. Los principios en los que se basan pueden ser los siguientes:

- Vector Distancia (DV).- Al igual que en los protocolos de infraestructura, un nodo envía toda la tabla de enrutamiento a sus vecinos para actualizarlos.
- Estado de Enlace (LS).- En este principio un nodo envía únicamente el estado de sus enlaces de próximo salto a sus nodos vecinos.

3.3 Ejemplos de protocolos de Enrutamiento Unicast para redes Ad Hoc

Para entender mejor el comportamiento de los protocolos de enrutamiento en las redes Ad Hoc, a continuación se presentan los ejemplos más representativos de cada categoría, anteriormente mencionada, con una breve descripción de su funcionamiento y características más representativas.

3.3.1 Protocolos Proactivos

Estos protocolos son manejados generalmente por una tabla de enrutamiento cuya finalidad es que la información sobre las rutas sea intercambiada entre todos los nodos sin importar están en uso o no. Estos protocolos pueden usar el principio de vector distancia (DV), estado de enlace (LS) o ambos. El mantener una tabla de enrutamiento le permite reducir el retardo al momento de enviar los paquetes y también disminuir el tiempo de convergencia de la red. A continuación se muestran los ejemplos de protocolos proactivos más representativos.

- **Tailoring Distance vector Protocols - Distance Sequenced Distance Vector (DSDV)**

  Este protocolo está basado en el principio de vector distancia y utiliza números secuenciales para la decisión de rutas lo que permite:

  - Envejecimiento de ruta mediante el aumento constante de su valor y la ruta se vuelve inservible. Una ruta es borrada si ninguna actualización es recibida en algunos períodos de tiempo.

- Complementar la información de la tabla de enrutamiento. Así números impares dan a conocer rutas inalcanzables, mientras que números pares indican actualizaciones de ruta.
- Cuando se debe dar la actualización sobre una ruta se hace lo siguiente: si la ruta tiene un número de secuencia más reciente o el mismo pero con una mejor métrica se actualiza, de otra manera la actualización se desecha.

Este protocolo permite enviar toda la tabla de enrutamiento o solo actualizaciones cuando se produzcan cambios en la topología. Cualquiera sea el caso, las actualizaciones son enviadas cada Δt tiempo y este valor se cambia cuando lo hace la topología, adaptándolo a la estabilidad de la red.

Finalmente, las rutas inservibles son actualizadas sin ningún tipo de retardo mientras que las nuevas rutas tienen un retardo denominado *settling time* para evitar que una ruta poco confiable sea incluida inmediatamente. Para calcular el *setting time* se considera lo siguiente:

- Dirección destino
- El último valor de *settling time*
- El *settling time* promedio calculado hasta el momento

- **Link State Algorithm – Optimized Link State Algorithm (OLSR)**

  Este protocolo se basa en el fundamento de estado de enlace de sólo compartir información con los vecinos. Es así que permite que cada nodo conozca: información de sus vecinos, topología y tabla de enrutamiento.

  Su fundamento es la selección de los nodos denominados como *Multipoint Relay* (MPR) que permiten aliviar el tráfico cuando se intercambia información de actualización. Es así que cada nodo elije independientemente a un nodo MPR entre sus vecinos con quién tendrá comunicación bidireccional y a quién será el único a quién mande información sobre actualizaciones. En la figura

2.7 se muestra un ejemplo de selección de MPR para diferentes nodos en una topología de máximo dos saltos de distancia.

Otro concepto importante es el *Multipoint Relay Selector Set* que es definido como el conjunto de vecinos que consideran a un nodo como MPR. Cada nodo envía información periódicamente a sus vecinos junto con un mensaje de control denominado *Topology Control* (TC). Un TC permite anunciar al conjunto de *Multipoint Relay Selector Set* las rutas que están disponibles pero además usa un número secuencial que se incrementa cuando el *Multipoint Relay Selector Set* cambia. Cuando no se tiene un conjunto de MPR a quiénes informar no se intercambian mensajes TC pero cuando el conjunto de un nodo se vuelve cero (por las razones que sean) se deben seguir enviando mensajes TC para invalidar la información anterior. Es así como los mensajes TC permiten conocer la topología de la red.

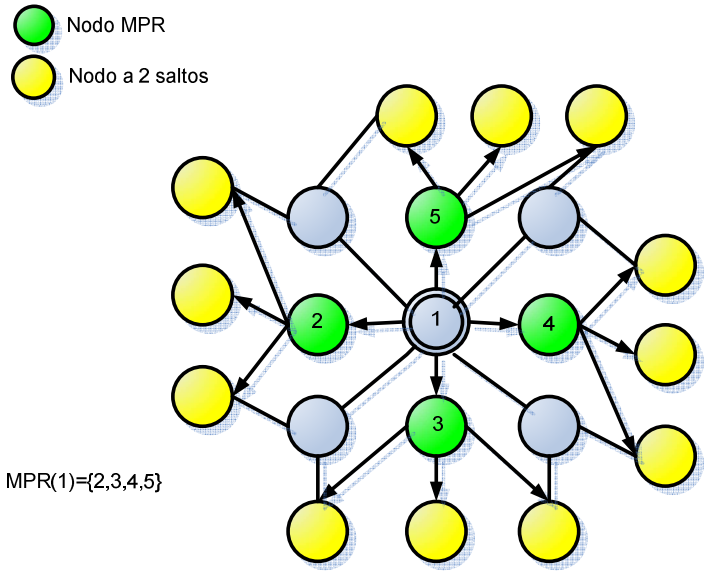


Figura 2.7. Ejemplo de selección de MPR en OLSR

- **Distance Vector and Link State - Fisheye State Protocol (FSR)**

  Este protocolo trata de aprovechar los beneficios de los principios de estado de enlace y de vector distancia. Tiene como particularidad la definición de ámbitos o alcances (*scopes*) como la zona hasta donde se transmite la información. Es así que tenemos que todos los nodos mantienen una tabla de

enrutamiento y una lista de vecinos (como los protocolos de estado de enlace). La información se intercambia únicamente entre vecinos y no se difunde en toda la red como lo hacen los de estado de enlace. La figura 2.8 muestra la división de una red por áreas acorde se va alejando del centro.

La información que se intercambia sobre las rutas viene acompañado por números secuenciales para indicar lo actualizada que está la información como en DSDV.

Los ámbitos o áreas están definidos como el conjunto de nodos que se pueden alcanzar en n saltos. Esto hace que los nodos tengan abundante información sobre las áreas cercanas pero cada vez menos información para las áreas más lejanas. Tal manejo de información puede llevar a una inadecuada selección de ruta en un principio pero a medida que el paquete llega al destino se tiene un mejor conocimiento sobre el área y un nodo puede redirigirlo de mejor manera.

Para este protocolo existen estudios que indican que tiene un buen comportamiento para redes con gran cantidad de nodos.

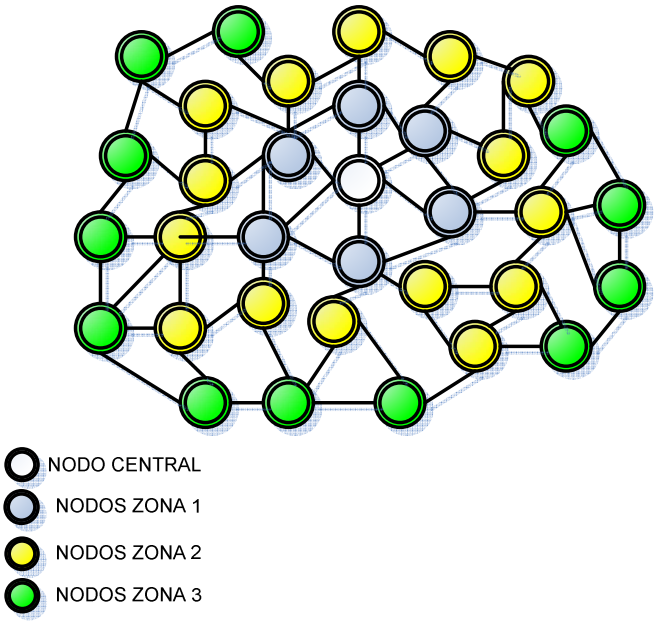


Figura 2.8 Definición de nodos en protocolo FSR

3.3.2 Protocolos Reactivos

Estos protocolos eliminan la necesidad de que los nodos mantengan tablas de enrutamiento y que intercambien actualizaciones periódicas sobre los estados de sus enlaces. Es así que los caminos se calculan sólo cuando resulta necesario, por lo que si por un camino no se ha enviado información simplemente no existe.

Las etapas para la construcción de rutas son las siguientes:

- Descubrimiento de rutas.- Esta etapa no se la realiza constantemente, pues cuando se traza una ruta se la mantiene por un período de tiempo. El proceso de descubrimiento se lo hace mediante múltiples peticiones que deben tener una respuesta cuando la petición encuentra el nodo. Este proceso se da de manera asíncrona.
- Mantenimiento de rutas.- Se refiere a aquellas aprendidas en el proceso de descubrimiento que resulten necesarias para intercambiar información. Cuando se ha descubierto una ruta se dice que la red ha adquirido un nuevo estado de enrutamiento.
- Eliminación de rutas (Opcional).- Esta etapa se da cuando una ruta resulta no ser más necesaria.

A continuación se presentan ejemplos sobre los protocolos reactivos más representativos.

- **Ad Hoc On Demand Distance Vector Routing (AODV)**

  Para este protocolo el proceso inicia cuando un nodo desea llegar a otro pero no existe una ruta predefinida para hacerlo. Es así que se lanza un paquete de petición denominado *RREQ* que al pasar por los nodos va incluyendo información sobre sí mismo y sobre los nodos anteriores por los que pasó, en forma de ubicación en reversa del nodo de origen. Cuando el destino nodo es encontrado, se envía un paquete de respuesta denominado *RREP* y el resto de direcciones recibidas no son consideradas en un período de tiempo. En la figura 2.9 se muestra cómo funciona el proceso cuando el nodo uno desea

alcanzar al quince, en este caso la ruta elegida fue [1, 2, 10, 9] por lo que las demás rutas son desechadas.

Una vez seleccionada la ruta, para mantenerla se intercambian mensajes *HELLO* con la ruta conocida. Pero cuando se produce la caída de un enlace se manda un paquete de *Unsolicited Route Reply* en el cual se da a conocer la invalidación del enlace. Cuando se pierde la ruta se vuelve a ejecutar la petición de ruta e inicia nuevamente el proceso de descubrimiento.

Además, este protocolo usa números secuenciales para invalidar rutas antiguas y así evitar bucles en el enrutamiento.

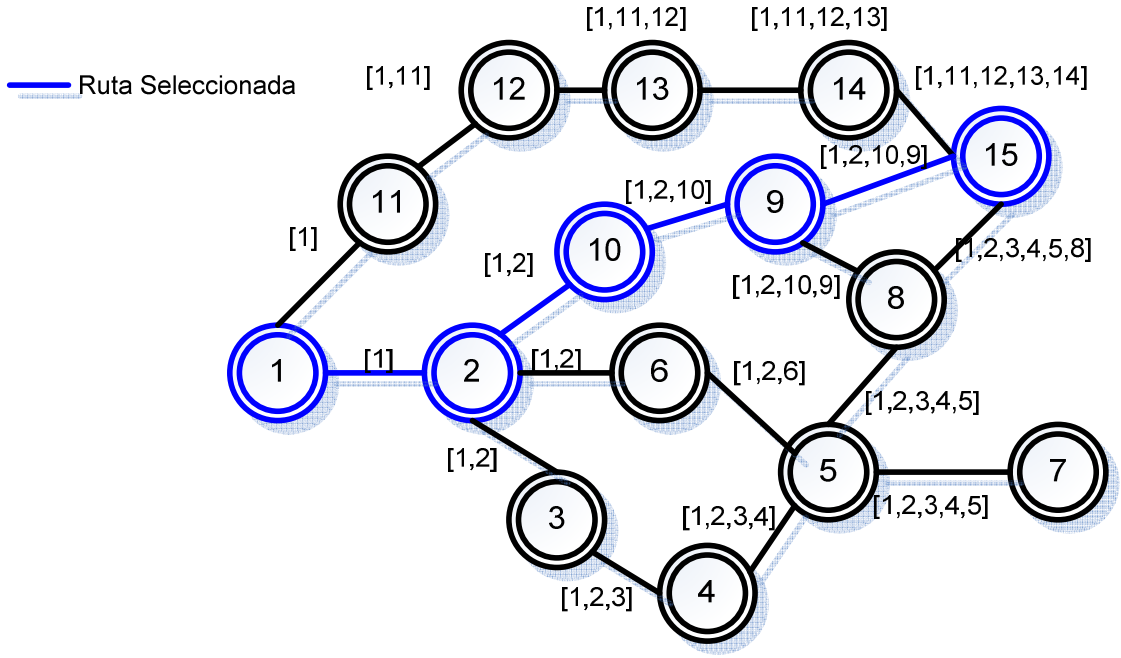


Figura 2.9. Ejemplo de selección de ruta en AODV

- **Adaptive Distance Vector (ADV)**

  El protocolo utiliza un principio bastante sencillo ya discutido anteriormente en la sección de calidad de servicio (QoS) de notificar sobre empezar la transmisión y cerrarla cuando ya no es necesaria entre las partes

involucradas. Es así que tiene como característica principal la de reducir la sobrecarga que al variar la frecuencia y tamaño de las actualizaciones de enrutamiento en respuesta al tráfico y a la movilidad de los nodos, al mantener las rutas para los "receptores activos". Se comporta como un algoritmo de vector distancia cualquiera al transmitir la información de los receptores activos.

Un nodo se lo etiqueta como receptor activo cuando tiene alguna conexión en trámite. Cuando un nodo quiere transmitir debe enviar un paquete de control denominado *init-connection* a todos los nodos de la red. Este paquete indica que necesita abrir una conexión con el nodo destino. Y cuando la conexión se cierra, el nodo origen manda un paquete de control denominado *end-connection*. Si el destino ya no tiene más conexiones activas también manda un mensaje de *non receive alert* a toda la red.

- **Temporally Ordered Routing Algorithm (TORA)**

  Este algoritmo es uno de los más completos y estudiados por ser altamente flexible ante cambios [38]. Es así que está diseñado para comportarse eficientemente a cambios topológicos y particiones en la red. Su nombre se deriva en que existe un reloj de sincronización para los eventos ocurridos dentro de la red.

  Cabe indicar que el objetivo no es buscar un enrutamiento óptimo sino rutas estables que puedan ser reparadas local y rápidamente. Para ello, se construye un gráfico acíclico directo (DAG) enrutado a un destino para tal propósito. El DAG se construye en base del peso o nivel de referencia asignado a los nodos y tiene la propiedad de que existe un único punto de llegada pero cada nodo debe tener al menos una ruta para llegar a tal punto. En la figura 2.10 se muestra una topología en la cual cada nodo tiene múltiples formas de llegar al nodo 8.

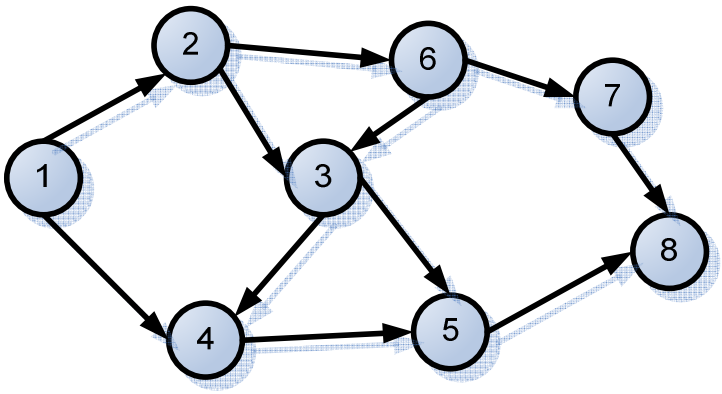
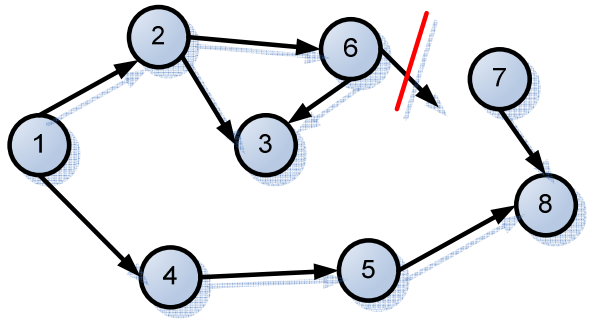


Figura 2.10 Ejemplo de establecimiento de DAG

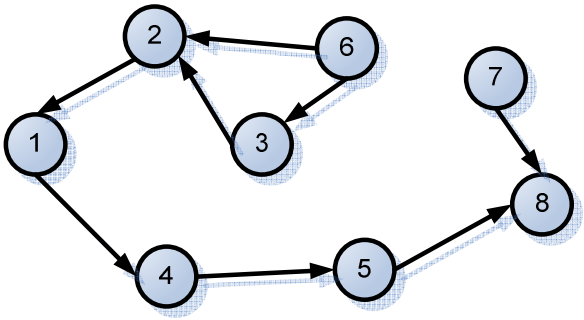
El funcionamiento básico del protocolo se puede dividir en tres partes:

- Descubrimiento de rutas.- Se intercambian pequeños paquetes de control. Los nodos reciben una dirección lógica basada en su peso para llegar al destino, que tiene el menor peso en la red. “Es como comparar un sistema de tuberías donde el destino es la parte más baja” [38].
- Mantenimiento de Rutas.- No se producen cambios ante cambios en la topología, por lo que si se cae una ruta a un nodo para llegar al destino, pero tiene otras rutas válidas disponibles, no se envía información. Sólo cuando un nodo no tiene rutas disponibles hace una petición para convertirse en el máximo global, la cual se propaga por la red, y entonces todos los nodos pierden las rutas de enlace hasta crear un nuevo DAG.
- Eliminación de rutas.- Cuando existe partición en la red, un nodo envía un paquete de borrado que borra todo el estado de enrutamiento.

En la figura 2.11 se muestra el proceso que se dispara cuando se pierde un enlace para llegar a un nodo destino. En este caso vemos como la conexión se pierde para el nodo seis para llegar al destino nodo ocho, al no existir rutas alternas, el nodo seis envía una petición para establecer un nuevo DAG teniéndolo como máximo. Al final la ruta se vuelve a establecer con una ruta alterna para el nodo seis.



a) Pérdida del enlace entre el nodo 6 y el nodo 8



b) Establecimiento de un nuevo DAG

Figura 2.11 Reacción de la topología ante cambios relevantes

3.3.3 Protocolos Híbridos

Este tipo de protocolos combinan las ventajas de los protocolos reactivos y proactivos enfocándose principalmente en reducir el retardo para conocer nuevas rutas y la carga en el intercambio de información.

El ejemplo más representativo de este tipo de protocolos es Zone Routing Protocol (ZRP). Este protocolo introduce el concepto de zona (Z) donde Z(k,n) significa un nodo n de radio k no tiene alcance más allá de k saltos, como se muestra en la siguiente ecuación:

$$Z(k,n) = \{i \mid H(n,i) \leq k\}$$

Donde H(n,i) es la distancia en número de saltos entre el nodo n y el nodo i. El nodo n es conocido como central de la zona de enrutamiento, e i es el nodo de borde para la zona. La idea es que el valor de K sea mucho menor en comparación con el diámetro total de la red, sino el protocolo pierde sentido. En la figura 2.12 se muestra

un ejemplo de delimitación de zona para un nodo, en este caso del nodo uno, con un alcance máximo de 2 saltos.

Para este protocolo se tienen cuatro grandes componentes para enrutamiento:

- IARP (IntrAzone Routing Protocol).- Las cuales son rutas creadas proactivamente para enrutamiento dentro de la zona.
- IERP (IntErzone Routing Protocol).- Se da cuando se tiene una distancia mayor a la de la zona de alcance del nodo. En este enrutamiento se envía peticiones exclusivas a los miembros denominados de “borde” de las zonas hasta llegar al destino.
- BRP (Bordercast Protocol).- Es el tipo de transmisión que se produce para los nodos de “borde”. Cuando un nodo n recibe un paquete y no está dentro de la zona de destino lo manda hacia otro nodo de borde hasta llegar a la zona de destino donde será enviado mediante IARP al nodo buscado.
- NDP (Layer-2 Neighbor Discovery Maintenance Protocol).- Ayuda a IARP para obtener información acerca de los vecinos en una determinada zona.

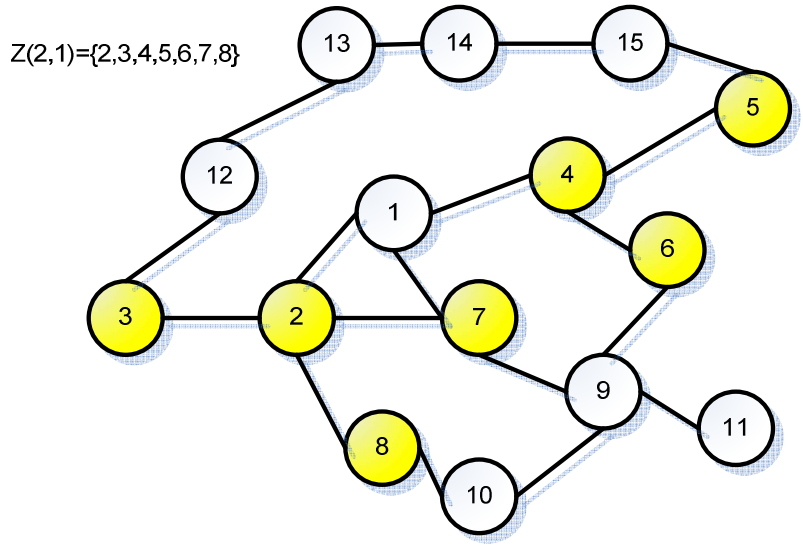


Figura 2.12. Ejemplo de vecindad producida con máximo 2 saltos

Su mayor desventaja es que al no existir intercambio de información entre los miembros de las regiones lo más común es que las zonas se sobrepongan y esto causa problemas al momento de hacer el enrutamiento entre zonas (BRP). Por otro lado el mantenimiento de rutas se lo hace únicamente para las que entre diferentes zonas, ya que cuando se producen errores en rutas dentro de zonas, es posible hacer un arreglo interno sobre los enlaces y de esta manera se impide que se tenga que hacer una actualización sobre toda la red.

3.3.4 Protocolos basados en la ubicación

Como ya se mencionó anteriormente, estos protocolos utilizan elementos externos para ubicar la posición del origen y del destino como soporte en el enrutamiento. El ejemplo más relevante es el protocolo *Location Aided Routing Protocol* (LAR) que busca tener una estimación de la ubicación del destino para hacer más eficiente el proceso de descubrimiento.

LAR introduce el concepto de *request zone*, haciendo referencia a aquellos nodos a los que se les hace la petición durante el proceso de descubrimiento. Tal zona es identificada gracias a la ayuda de un elemento externo como un GPS y se la conoce como el rectángulo más pequeño que incluye la ubicación de S y la zona denominada *expected zone*, que es aquella donde se espera que se encuentre el nodo destino B teniendo como origen a A para un tiempo T1 sabiendo que A conoció información de la posición y velocidad (de ser necesario) de B para un tiempo T0.

Por tanto un nodo intermedio j del proceso de enrutamiento envía la información al destino cuando lo recibe por primera vez y además cuando:

$$Dist + \delta > D_j$$

Donde el valor de Dist indica la distancia definida desde el origen hacia el destino y Dj es la distancia calculada al destino por el nodo. Además se añade un elemento de error denominado δ que puede representar un desfase en la distancia calculada por el GPS o un retardo en hallar la ruta correcta por primera vez.

Varios estudios indican que el tener la ubicación del nodo destino ayuda a disminuir la carga de los mensajes en la red para ubicar rutas y disminuye el tiempo de retardo en el envío de paquetes [39].

4. Tecnologías para redes Ad Hoc

Dejando de lado a estudios en centros especializados y en universidades existen dos tecnologías que actualmente predominan al momento de formar redes Ad Hoc en el mercado, estas son Wifi y Bluetooth. Esto sin duda avala la tendencia de que el éxito de una tecnología de red está íntimamente relacionado con el desarrollo de productos a precios competitivos. Es así que gracias a la alta difusión de estas tecnologías es posible hoy en día establecer pequeñas redes Ad Hoc en oficinas y hogares.

Hoy en día mayoría de dispositivos que implementan Wifi y Bluetooth no están diseñados específicamente para formar redes Ad Hoc por lo que resulta muy difícil que los protocolos de enrutamiento mencionados en la sección 3.3 puedan ser aplicados a nivel de capa de red. Sin embargo, emulaciones de los algoritmos presentados pueden ser escritos en la capa de aplicación del modelo OSI pero sacrificando estabilidad, velocidad y calidad de servicio pues tal capa no está diseñada para el enrutamiento de datos.

A pesar de las limitaciones, actualmente tenemos un auge en el aparecimiento de aplicaciones Ad Hoc para dispositivos electrónicos. Tal es el caso de la nueva consola de Sony la PlayStation Portable 3000 (PSP3000) que incluye una función denominada como *Ad Hoc Party* que permite que varios equipos puedan ser conectados sin una infraestructura previa, utilizando la conectividad Wifi de los mismos para compartir juegos en tiempo real. En la figura 2.13 se muestra el logo de la funcionalidad que está disponible para algunos juegos de la consola.

Figura 2.13. Logo publicitario de la funcionalidad Ad Hoc Party de PSP

También existen otras aplicaciones pero basadas en la tecnología Bluetooth que permiten formar redes Ad Hoc pero utilizando aplicaciones java en celulares. Tal es el caso de Xofto [40], la cual es una empresa Polaca dedicada a hacer juegos multiusuario para celulares.

Bluetooth es una tecnología muy popular a nivel global que permite formar redes Ad Hoc de manera rápida y sencilla. Tales redes son conocidas como *Piconets*, cuyo establecimiento y mantenimiento es más sencillo que los protocolos mencionados en la sección 3.3. Pero las *Piconet* tienen la limitante de tamaño y alcance, por ello resulta necesario el estudio de la formación de *Scatternets* que no son más que la conjunción de varias *Piconet.* En el siguiente capítulo se tratará a profundidad de la tecnología Bluetooth y de la formación de *Piconets*.

Resulta evidente que las redes Ad Hoc tienen un campo de aplicación definido en un futuro de dispositivos electrónicos con alta capacidad de procesamiento y mejores anchos de banda. Es de esperar que en el futuro salgan al mercado dispositivos que tengan su fundamento en la computación ubicua y en las redes sin infraestructura o Ad Hoc.

CAPÍTULO 3

BLUETOOTH

1. Introducción

Bluetooth se define como una tecnología de comunicación inalámbrica a corta distancia, rápida y confiable usada globalmente en dispositivos móviles como celulares, agendas y computadores personales, entre otros. Actualmente goza de alta popularidad en el mercado gracias a la sencillez de uso y al poco consumo de energía, lo cual ha hecho que sea incluida en la mayoría de dispositivos móviles [44].

Otra de las características positivas de esta tecnología es que usa la banda no licenciada ISM (*Industrial, Scientific and Medical*) de 2.400-2.483.5 GHz como se muestra en la tabla 3.1. Para evitar la interferencia con otros protocolos que utilizan la misma banda, el protocolo, en su versión estándar, divide la banda en 79 canales, de 1 MHz cada uno, y hace saltos pseudoaleatorios entre los mismos a una velocidad de hasta 1600 veces por segundo hasta encontrar el canal apropiado para la transmisión y de 3200 veces por segundo cuando se realiza la búsqueda de dispositivos por primera vez.

Tabla 3.1. Características en Frecuencia y Canalización de Bluetooth

Frecuencias en uso [GHz]	Canales
2.400-2.4835	f= 2402+k ; k=0,….,78 [MHz]

Cualquier tipo de enlace Bluetooth comienza con un proceso de búsqueda de dispositivos entre los cuales puede existir un enlace. Luego, viene la etapa de emparejamiento en la cual se establecen condiciones de seguridad y modos de transmisión entre los dos dispositivos para establecer el canal de comunicación. Es

así que con Bluetooth, a pesar de ser una tecnología inalámbrica, se establece un único enlace similar a uno físico entre dos dispositivos. Una vez establecido el canal se da el proceso de prestación de servicios ya sean estos de transmisión de datos o voz, de una o varias aplicaciones. En la figura 3.1 se muestra un esquema de funcionamiento de múltiples conexiones que utilizan un mismo enlace Bluetooth. Este funcionamiento es similar al de una red LAN en donde sobre un mismo medio se ofrecen varios servicios.

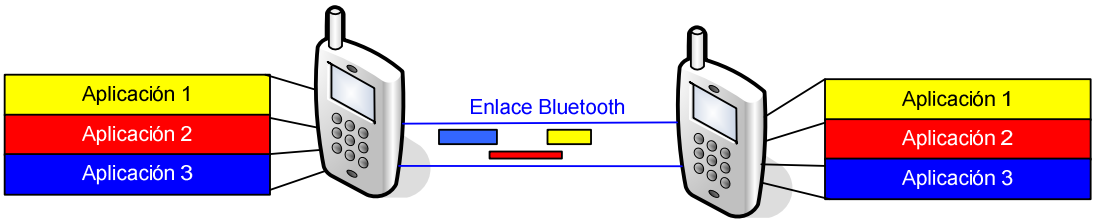


Figura 3.1 Esquema de utilización de un único canal físico Bluetooth.

A pesar de que en un inicio la tecnología fue ideada para proveer reemplazo a los cables que se conectaban desde los periféricos a las computadoras personales, actualmente la tecnología tiene desarrollos en áreas y otros ámbitos muy diferentes debido a su sencillez, popularidad y costo. Entre las áreas más ambiciosas del desarrollo de la tecnología se encuentran el área multimedia con la transmisión de voz y video en tiempo real gracias a la especificación Bluetooth 3.0 + HS [45] y en el área médica e industrial con Bluetooth de bajo consumo de energía [46]. En la figura 3.2 se muestra un estudio tomado de la revista SIGnature [47] sobre cómo las nuevas especificaciones Bluetooth de alta velocidad y bajo consumo de energía se están dando a conocer en el mercado.

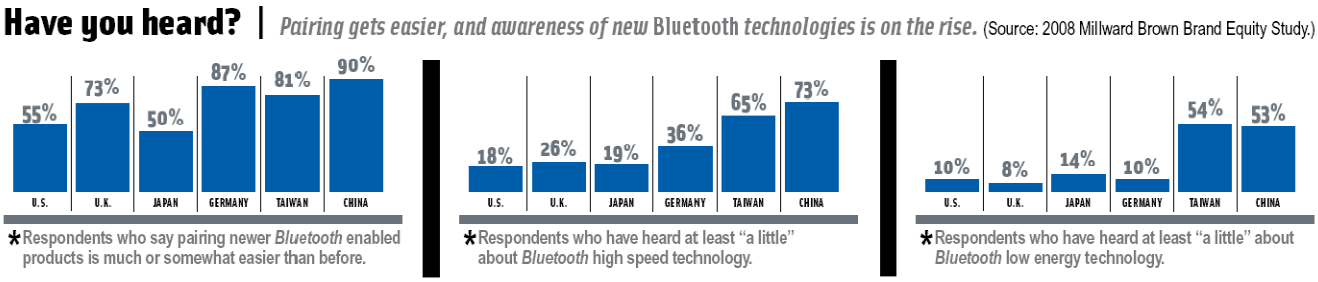


Figura 3.2. Estudio de Mercado sobre nuevas especificaciones Bluetooth.

En esta sección se ha dado una pequeña introducción sobre aspectos técnicos y comerciales de la tecnología Bluetooth. A lo largo de este capítulo se presentará un estudio más profundo acerca de las características más relevantes de la tecnología incluyendo el funcionamiento del canal físico y las nuevas tendencias para la tecnología, obtenidas de la última versión de la especificación emitida en diciembre de 2009 [48].

2. Fundamentos

El canal físico es la capa más baja de la arquitectura de la pila Bluetooth. Está caracterizado por ser la combinación pseudoaleatoria de saltos en frecuencia, asignación específica en slots de tiempo para las transmisiones y codificación de las cabeceras de los paquetes. Es así que para que dos dispositivos puedan comunicarse deben usar un canal físico compartido, en el cual sus transceptores funcionen a la misma frecuencia RF durante un determinado tiempo. Cuando un dispositivo está sincronizado en tiempo, frecuencia y en códigos de acceso a un determinado canal se dice que el dispositivo se encuentra conectado.

Para reducir la interferencia en frecuencia entre los dispositivos Bluetooth que puede causar colisiones en la transmisión de la información se utilizan códigos de acceso que funcionan como códigos correlatados para identificar a los dispositivos que están usando el canal, un sistema bastante similar al utilizado en la tecnología CDMA.

Un dispositivo Bluetooth puede hacer uso de un único canal físico a la vez. Por ello, para multiplexar múltiples conexiones el dispositivo utiliza un esquema TDM para dar servicio a los diferentes canales. Esta multiplexación le permite a un dispositivo tener comunicación con múltiples redes. Todos los canales físicos están subdivididos en slots de tiempo, pero su longitud varía según el canal.

Para un determinado dispositivo, un evento de transmisión o recepción tiene asociado un slot de tiempo específico y además una determinada frecuencia seleccionada. La tasa de saltos de frecuencia para la selección de la frecuencia es de 1600 saltos/s en el estado de conexión y de 3200 saltos/s en los estados de escaneo previo a la conexión.

Es así como la tecnología Bluetooth utiliza varios elementos para lograr la sencillez y rapidez en la comunicación. Es esta sección se presentan los aspectos más relevantes relacionados a características de uso del medio, modulación, potencia de transmisión y acceso al medio.

2.1 Clases

Se conoce como clase en Bluetooth a la especificación de potencia a utilizar al momento de realizar la transmisión. Los dispositivos clase 1 son los que mayor potencia utilizan para la transmisión y pueden alcanzar mayores distancias, mientras que los dispositivos de clase 3 son aquellos que utilizan la menor cantidad de potencia en las transmisiones y permiten el ahorro de batería para dispositivos con capacidades limitadas. En la tabla 3.2 se muestra un resumen con las características de cada una de las clases definidas para dispositivos Bluetooth.

Cualquier dispositivo Bluetooth debe encajar dentro de una de las tres clases. Además, existe un factor conocido como el control de potencia que se encarga de limitar la potencia que llega al receptor a no más de 4 dBm para evitar la saturación del mismo, en dispositivos clase 1. Para dispositivos clase 2 y clase 3 el control de

potencia es opcional pero puede ser utilizado para optimizar el consumo de energía y reducir la interferencia con otros dispositivos en la banda ISM.

Clase	Potencia de salida máxima (Pmax)	Potencia de salida mínima (Pmin)	Control de Potencia
1	100 mW (20 dBm)	1 mW (0 dBm)	Sí
2	2.5 mW (4 dBm)	0.25 mW (-6 dBm)	Opcional
3	1 mW (0 dBm)	N/A	Opcional

Tabla 3.2 Clases Bluetooth.

Se debe mencionar que los valores establecidos en la tabla 3.2 se dan considerando un conector sin pérdidas en la antena o en su defecto cuando se usa una antena de 0 dBi de ganancia y que en cualquiera de los casos, la sensibilidad del receptor Bluetooth debe ser menor o igual a -70 dBm debido a que ese valor está estimado que el nivel de entrada produce un 0.1% de BER.

Un dispositivo clase 1 puede ser capaz de adaptar su potencia de transmisión acorde a los requerimientos del receptor. Para ello, deben intercambiar mensajes en el enlace físico entre los dispositivos. Los pasos entre varios niveles de potencia mínimo y máximo para dispositivos clase 1 se pueden dar entre 2 dB como mínimo y 8 dB como máximo. Para los dispositivos clase 2 y 3, el transmisor no debe exceder el máximo establecido en la potencia de transmisión, de lo contrario el dispositivo receptor podría no tener la capacidad de intercambiar los mensajes para que se disminuya la potencia de transmisión y podría saturarse.

Un efecto indeseable se produce al momento de hacer el descubrimiento de los dispositivos por primera vez. Este se da cuando un dispositivo clase 1 está muy cercano a otro por lo que no lo descubre debido al alto nivel de potencia que recibe. Por ello, un dispositivo clase 1 podría utilizar niveles de potencia clase 2 y clase 3 al momento de hacer el escaneo de dispositivos y luego pasar nuevamente a los niveles de potencia clase 1 y 2 de ser necesario.

Finalmente, cabe mencionar que además de los niveles de potencia establecidos en la tabla 3.1 existen consideraciones de cada país a ser tomadas en cuenta. Por ejemplo en nuestro país, para la banda de 2.4 GHz existe una limitante de 1 [W] para la banda ISM se cual fuera la tecnología, modulación y antena que se esté utilizando [2].

2.2 Características de Modulación.

En esta sección cabe definir que dependiendo de la especificación varía la modulación que se utiliza. Esto es debido a que es posible utilizar la tecnología Bluetooth a una velocidad estándar (*Basic Rate)* o con la tecnología de velocidad mejorada EDR (*Enhanced Data Rate*).

2.2.1 Velocidad Estándar

Para Bluetooth según la especificación 4.0 se ha definido a GFSK (Gaussian Frequency Shift Keying) como el tipo de modulación a utilizarse bajo las siguientes características.

- Producto BT=0.5
- Índice de modulación entre 0.28 y 0.35
- Uno binario representado por una desviación positiva en frecuencia
- Cero binario representado por una desviación negativa en frecuencia.
- El período de muestreo debe ser menor a 20 ppm.
- El error de cruce por cero debe ser menor a 1/8 del período del símbolo.

En la figura 3.3 se muestra un diagrama de ojo de la modulación GFSK para Bluetooth de donde se puede observar la desviación en frecuencia mínima (Fmin) que se define como:

$$F\min = \min\left\{\left|F\min+\right|, F\min-\right\} \qquad \text{Eq. 3.1}$$

Tal valor de Fmin corresponde a secuencia 1010 que debe ser menor al 80 % de la desviación en frecuencia (Fd) con respecto a la frecuencia de transmisión (Ft),

que corresponde a una secuencia 00001111. Además, la desviación en frecuencia mínima nunca debe ser menor a 115 KHz

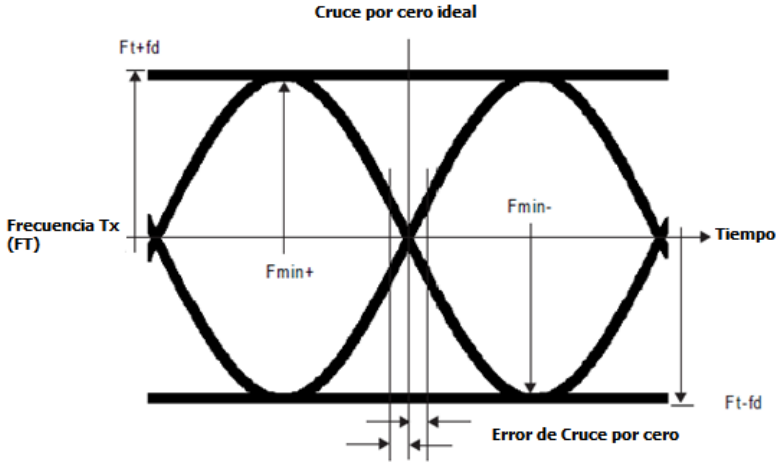


Figura 3.3 Diagrama de ojo para GFSK

En lo referente a la tolerancia de radiofrecuencia se tiene que la exactitud inicial en frecuencia se define como la capacidad de hallar el valor ideal en una transmisión antes de que algún paquete sea transmitido. Según la especificación de la versión 4 para Bluetooth se tiene un valor 75 KHz respecto al valor central. Cuando la comunicación ha sido iniciada se tienen diferentes valores de tolerancia en frecuencia según el tamaño o duración del paquete, como se muestra en la tabla 3.3.

Por otra parte, para diferentes tipos de paquetes se tienen diferentes tolerancias como se indica en la tabla 3.3. La longitud de un paquete está definida por su duración, es decir por la cantidad de slots que ocupa. Sobre el establecimiento de slots y su duración se tratará en la sección 3 de este capítulo que trata sobre Piconets.

Tabla 3.3 Desviaciones en frecuencia para diferentes longitudes de paquete

Duración del Paquete	Desviación en frecuencia
Paquete de un slot de duración	25 [KHz]
Paquete de tres slots de duración	40 [KHz]
Paquete de cinco slots de duración	40 [KHz]

2.2.2 Enhanced Rate

Su característica clave está en que el modo de modulación cambia con cada paquete. Es así que la primera parte de los paquetes son transmitidos a la velocidad básica de 1 Mbps con la modulación GFSK, mientras que las siguientes secciones de los paquetes utilizan la tasa de transmisión mejorada con modulación PSK, definida bajo los siguientes parámetros:

- Para 2 Mbps se debe utilizar la codificación diferencial rotada π/4 PSK (Phase Shift Keying) - π/4DQPSK.
- Para 3 Mbps se debe utilizar la codificación 8-aria diferencial PSK – 8DPSK.
- Se debe usar el muestreo de raíz cuadrática de coseno levantado.

Al utilizar la modulación PSK se debe obtener a la salida del transmisor una señal pasa banda que puede ser representada como:

$$S(t) = \mathrm{Re}\left| v(t)e^{j2\pi Fct} \right| \qquad \text{Eq 3.2}$$

Donde:

Fc es la frecuencia de la portadora

V(t) es la señal pasa bajo equivalente de la información formada acorde a la siguiente ecuación.

$$v(t) = \sum_{K} S_k \, p(t - kT) \qquad \text{Eq 3.3}$$

Donde p(t) es el muestreo raíz cuadrático de coseno levantado y el período T siempre debe ser de 1 [us] para todo valor de k.

Por otro lado si se tiene una modulación m-aria y una secuencia de datos bn tal que:

$\{bn\}$ existe para $n=1,2,3,\ldots N$ Eq 3.4

Debemos representarla con su correspondiente secuencia Sk de puntos con valores complejos tal que:

$\{Sk\}$ para $k=1,2,\ldots N/\mathrm{Log}_2(M)$ Eq 3.5

Donde:

$$S_K = S_{K-1} e^{j\varphi_k} \qquad k = 1,2,...N / Log_2(M) \qquad \text{Eq 3.6}$$

$$S_0 = e^{j\varphi} \qquad \varphi \in [0,2\pi) \qquad \text{Eq 3.7}$$

Y el valor de M es de cuatro para 2 Mbps y de 8 para 3 Mbps. Para el caso en el que el conjunto de valores N no sea un entero múltiplo de $Log_2(M)$ lo que se debe hacer es llenar la secuencia de ceros para completar la secuencia a un valor adecuado. Además la relación entre la entrada binaria b_k y la fase φ_k se da según la tabla 3.4 para la velocidad de 2 Mbps y para la velocidad de 3Mbps se da según la tabla 3.5.

Tabla 3.4 Relación entre la fase y la entrada binaria para π/4DQPSK.

b_{K-1}	b_k	φ_k
0	0	π/4
0	1	3π/4
1	0	-π/4
1	1	-3π/4

Tabla 3.5 Relación entre la fase y la entrada binaria para 8DPSK.

b_{K-2}	b_{K-1}	b_k	φ_k
0	0	0	0
0	0	1	π/4
0	1	0	3π/4
0	1	1	π/2
1	0	0	-π/4
1	0	1	-π/2
1	1	0	π
1	1	1	-3π/4

En este caso la tolerancia respecto a la frecuencia central es igual a aquella definida para la tasa de transferencia estándar de 75 Khz, para los paquetes al inicio de la transmisión. Cuando comienza la transmisión utilizando EDR la tolerancia cae a

10 KHz como se muestra en la figura 3.4, debido al cambio de modulación que se tiene, como se discutió previamente.

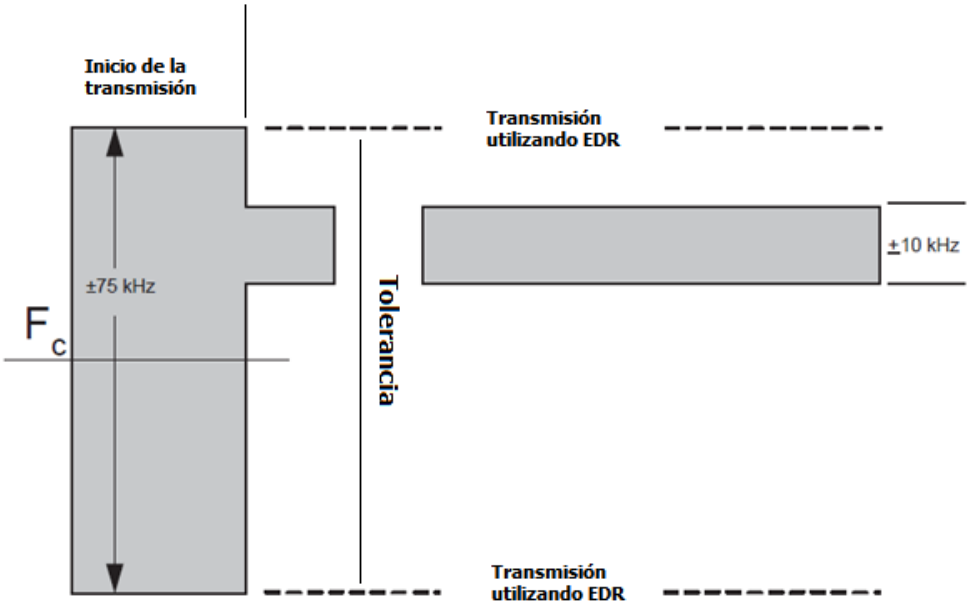


Figura 3.4 Tolerancia en frecuencia cuando se utiliza EDR.

2.3 Paquetes

El formato de los paquetes también varía dependiendo de la velocidad de transmisión que se tenga en la comunicación. En la figura 3.5 se muestra el formato del paquete general para la tasa de transmisión básica, donde se identifica los siguientes campos:

- Código de acceso
- Cabecera
- Payload o información.

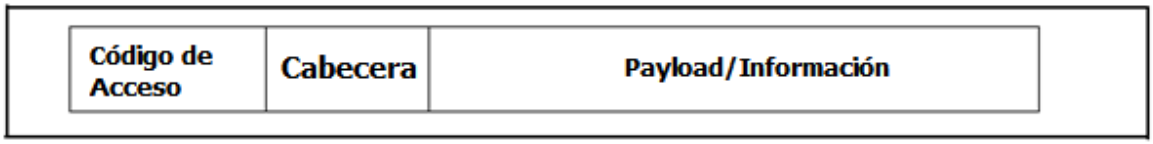


Figura 3.5 Formato del paquete para velocidad básica

Por otro lado, la figura 3.6 muestra el formato del paquete para una tasa de transmisión en EDR cada paquete consta de:

- Código de Acceso
- Cabecera
- Banda de Guarda
- Secuencia de Sincronización
- EDR Payload o Información
- Trailer

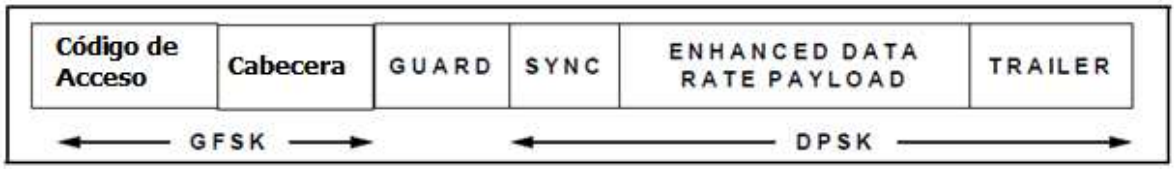


Figura 3.6 Formato del paquete para velocidad de transmisión con EDR

Como se observa en la figura, la banda de guarda permite hacer la transición entre los dos tipos de modulación que se hacen cuando se utiliza EDR.

2.4 Reloj

Cada dispositivo Bluetooth debe tener un reloj nativo derivado de su sistema pero se debe tomar en cuenta que tal reloj no tiene ningún tipo de relación con la hora del día y por ello puede inicializarse a conveniencia.

Por otro lado tenemos tres tipos de relojes utilizados en diferentes procesos de comunicación Bluetooth: el reloj nativo (CLKN), el reloj estimado (CLKE) y el reloj máster de una Piconet (CLK).

2.4.1 Reloj Nativo (CLKN)

Este reloj es derivado de aquel del sistema y es usado como la referencia para transmisiones y otras clases de reloj. Por otro lado cabe mencionar que el maestro nunca ajusta su reloj nativo durante la existencia de una Piconet.

2.4.2 Reloj máster de una Piconet (CLK)

CLK es la aproximación al reloj máster de una Piconet y debe ser usado para todas las actividades programadas que necesiten de temporización dentro de la misma. CLK se obtiene al añadir al reloj nativo (CLKN) un período de guarda u offset excepto para el dispositivo que funciona como maestro en la comunicación, como se muestra en la figura 3.7. El añadir el período de guarda busca que cada dispositivo tenga un reloj CLK coherente con el CLKN del maestro. Los períodos de guarda deben ser actualizados periódicamente mediante las cabeceras de código de acceso en cualquier paquete enviado por el maestro al esclavo, para evitar errores por mala sincronización.

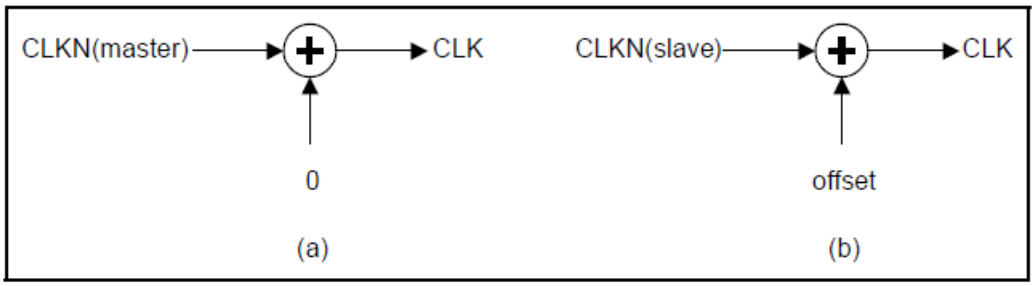


Figura 3.7 Esquema de establecimiento del reloj CLK

2.4.3 Reloj estimado (CLKE)

Es utilizado en la búsqueda de dispositivos por parte del maestro. Tiene una estructura similar a CLK, como se muestra en la figura 3.8, de añadir un período de guarda u offset al reloj de cada dispositivo pero con la diferencia de que no se da para mantener estabilidad de la comunicación sino para procurar que todos los relojes (maestros y esclavos) tengan tiempos similares y agilitar el proceso de descubrimiento.

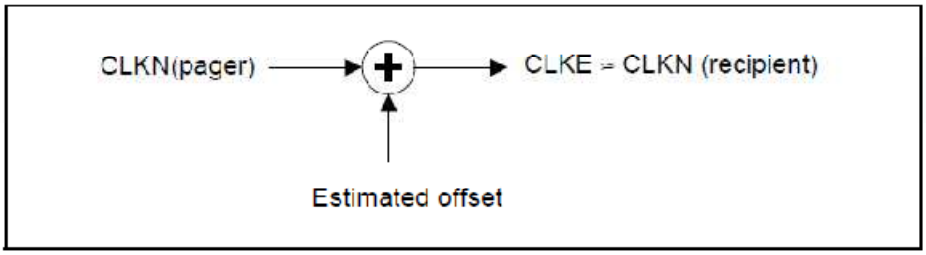


Figura 3.8 Esquema de establecimiento del reloj CLKE

2.5 Direccionamiento

Cada dispositivo Bluetooth tiene una dirección única de 48 bits denominada como BD_ADDR. Esta dirección debe ser otorgada por la autoridad de registro de la IEEE y seguir el estándar establecido en IEEE 802-20001[48]. El formato de la dirección para un dispositivo Bluetooth consta de tres partes, como se muestra en la figura 3.9:

- Lower Address Part (LAP).- Bits menos significativos en la dirección. Esta sección puede ser definida por la empresa fabricante y permite identificar ciertas características de los dispositivos.
- Upper Address Part (UAP).- Parte intermedia y está definida para la autoridad de registro de IEEE. Permite identificar al fabricante
- Non Significant Address Part (NAP).- Sección de bits más significativos pero que únicamente indican al fabricante y no características del dispositivo.

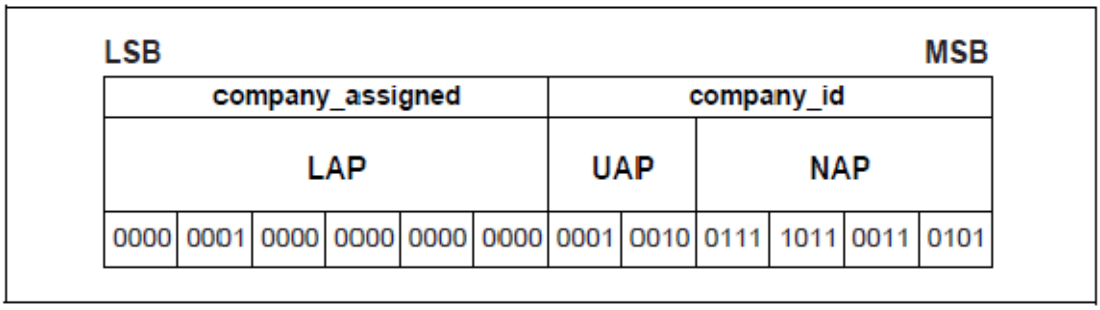


Figura 3.9.- Formato de las direcciones físicas Bluetooth.

Existen números que se encuentran reservados, en números hexadecimales comprenden direcciones LAP del 0x9E8B00 al 0x9E8B3F. Para conocer la lista de números asignados se puede verificar la página web oficial del SIG de Bluetooth.[2]

3. Piconet

Los sistemas Bluetooth proveen conectividad punto-punto o punto-multipunto. En una conexión punto-punto el canal físico es compartido entre los dispositivos Bluetooth mientras que en una conexión punto-multipunto el mismo canal debe ser

compartido entre varios dispositivos Bluetooth. Para el caso en el que varios dispositivos Bluetooth comparten el mismo medio se dice que una *piconet* se ha establecido.

En una Piconet, un dispositivo actúa como maestro mientras el resto actúan como esclavos, como se muestra en la figura 3.10 (b). Un maestro puede tener hasta siete esclavos activos pero muchos otros más como esclavos pasivos (*parked*). Un dispositivo Bluetooth se dice que está pasivo cuando no está enviando información por el canal pero está sincronizado con el maestro y en algún momento se puede convertir en un dispositivo activo. De cualquier manera, el acceso al canal está controlado por el maestro tanto para los dispositivos activos tanto para los dispositivos en estado pasivos.

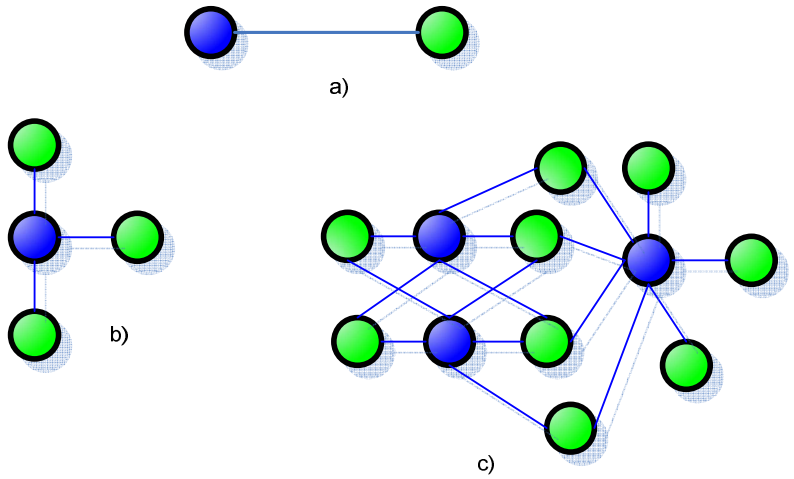


Figura 3.10. Topología de una Piconet de un solo esclavo a) de varios b) de una Scatternet c)

Cuando existen múltiples *Piconets*, solapamiento entre sus áreas y elementos comunes entre ellas a todo el conjunto de redes se le conoce como *Scatternet,* como se muestra en la figura 3.10(c). Si bien cada *Piconet* tiene un único maestro, sus dispositivos esclavos pueden participar en diferentes Piconets bajo un esquema TDM, que se estudiará las adelante. En una Scatternet, las Piconets que la forman no necesitan estar sincronizadas en frecuencia; de hecho cada una puede utilizar su

esquema de selección en frecuencia y de códigos de acceso de forma totalmente independiente.

Cabe mencionar que la pila de protocolos y de los perfiles que un dispositivo implementa queda a decisión del fabricante, por lo que la formación de S*catternets* aún no es una realidad si se utiliza únicamente dispositivos móviles existentes en el mercado, siendo que los dispositivos móviles únicamente pueden realizar enrutamientos de paquetes de hasta un salto a nivel de capa de red. Si se desea obtener mayor alcance se pueden utilizar otra clase de dispositivos que ofrecen características de funcionamiento similares a las de un enrutador [50]. Por ello es necesario destacar que esta tesis lo que pretende es utilizar una tabla de enrutamiento pero a nivel de capa de aplicación, la cual no es la solución más eficiente pero si aplicable a la mayoría de dispositivos móviles existentes en el mercado.

3.1 Códigos de Acceso

Todas las transmisiones Bluetooth sobre un canal físico comienzan con un código de acceso. Son de propiedad del canal físico y están siempre presentes en el inicio de cada paquete transmitido. Es así que se definen códigos de acceso de tres tipos:

- *Device Access Code (DAC).-* Este código es usado durante la etapa de descubrimiento y escaneo de los dispositivos. Está íntimamente relacionado con la dirección Bluetooth del dispositivo o BD_ADDR. Su longitud es de 68 bits.
- *Channel Access Code (CAC)*.- Es el único código de acceso que contiene un preámbulo y palabra de sincronización. Es utilizado para tener acceso al canal y su longitud es de 72 bits.
- *Inquiry Access Code (IAC).-* Permite enviar peticiones para un solo dispositivo, un conjunto de ellos o a todos los dispositivos que se encuentren dentro de la cobertura de un dispositivo. Cuando la petición es específica para un dispositivo o un grupo de características particulares se lo conoce como

Dedicated Inquiry Access Code (DIAC) mientras que cuando la petición es general para todos los dispositivos Bluetooth a la petición se la conoce como *General Inquire Access Code* (GIAC). Por ejemplo, si un dispositivo hace una búsqueda de todos los dispositivos con Bluetooth en el rango de cobertura enviará un código de acceso GIAC mientras que si busca únicamente a aquellos que tienen características de transmisión de voz enviará una petición DIAC.

3.2 Canales

Para el establecimiento de una Piconet existen cuatro tipos de canales físicos que pueden establecerse:

- Canal Piconet básico.
- Canal Piconet adaptado.
- Canal de escaneo y búsqueda.
- Canal de compaginación de escaneo

Los dos primeros canales son utilizados para comunicación entre dispositivos en una Piconet, mientras que el tercero y cuarto se utilizan para descubrir y conectarse a los dispositivos respectivamente. Solo se hablará de los tres primeros tipos de canales debido a que son fundamentales en la formación y comunicación dentro de una Piconet. Por otro lado, más adelante OBEX permitirá entender el funcionamiento del cuarto canal de una manera más simple.

3.2.1 Canal Piconet Básico

Es usado por defecto durante el estado de conexión. En este canal el maestro es quién controla el tráfico dentro de la Piconet mediante un esquema de poleo. En este el maestro está constantemente escuchando a los esclavos dándoles un tiempo específico de transmisión a cada uno. En este canal al dispositivo maestro, por definición, se conoce como aquel dispositivo que inicia la conexión mediante el escaneo o búsqueda de otros dispositivos.

El canal Piconet básico está caracterizado por la selección de frecuencia de manera pseudorandómica entre 79 canales RF. El tipo de salto en frecuencia está determinado por el reloj y la dirección Bluetooth (BD_ADDR) del maestro. Cuando la comunicación se ha establecido, cada esclavo debe añadir un tiempo de guarda a su reloj nativo para que pueda sincronizarse con el reloj del maestro, al nuevo reloj formado se lo conoce como CLK. El reloj CLK debe actualizarse regularmente durante la comunicación mediante el intercambio de mensajes.

El canal está dividido en slots de 625 [uS] que están enumerados desde 0 hasta 2^{27} y que se repiten en forma cíclica cuando se llega al límite. La forma de comunicación utiliza un esquema de división en tiempo TDD (Time-Division Duplex) para la transmisión alternada entre maestro y esclavos. En la figura 3.11 se muestra un ejemplo de comunicación entre maestro y esclavo para una comunicación Bluetooth en el canal Piconet básico, donde se observa que el inicio del paquete a transmitirse debe estar lo más alineado posible al inicio del slot en tiempo.

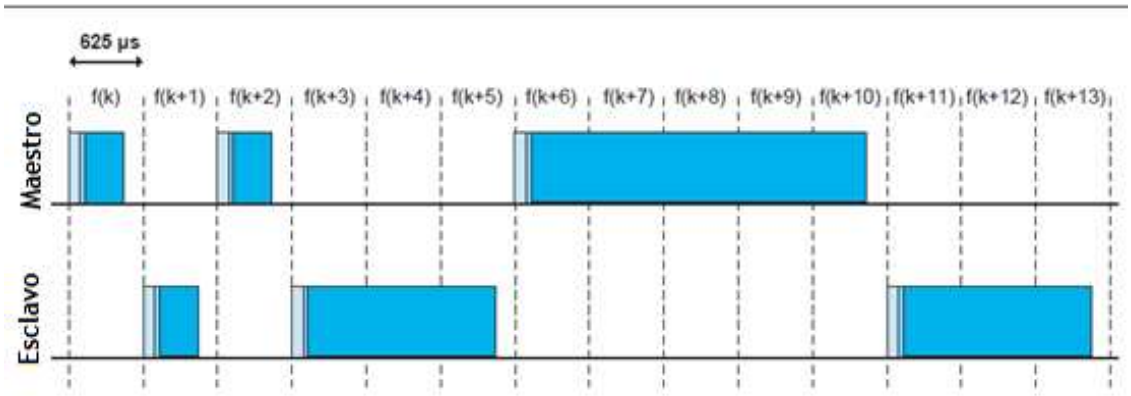


Figura 3.11 Transmisión en un canal Piconet básico

Además, de la Figura 3.11 puede observarse que la longitud de un paquete puede ocupar un solo slot de tiempo o extenderse hasta 5, dependiendo de las características de la transmisión y los permisos concedidos por el maestro para utilizar el canal.

Para este canal, la transmisión del maestro siempre debe empezar en slots de tiempo pares mientras que la comunicación iniciada por un esclavo lo debe hacer por slots de tiempo impares. Si un paquete ocupa más de un slot para ser transmitido, debe terminarse en un slot par si es enviado por el maestro y en uno impar si es transmitido por un esclavo, como se muestra en la figura.

- **Comunicación**

Cuando el maestro envía un paquete en una determinada frecuencia f(k), donde k indica el slot de tiempo, se espera una respuesta por parte del esclavo en Nx625 [uS] donde N es un número entero impar diferente de cero. Por otro lado, la transmisión por parte del mismo maestro está programada para un tiempo Mx1250 [uS] donde M es un número entero par diferente de cero, siempre y cuando no se envíen paquetes que ocupen más de un slot en tiempo.

De igual manera, la transmisión del esclavo comienza Nx625 [uS] después de la recepción de un paquete, donde N es un número impar diferente de cero. En la figura 3.12 se muestra como los slots de tiempo intercalan los paquetes que se envían y reciben entre el maestro y el esclavo.

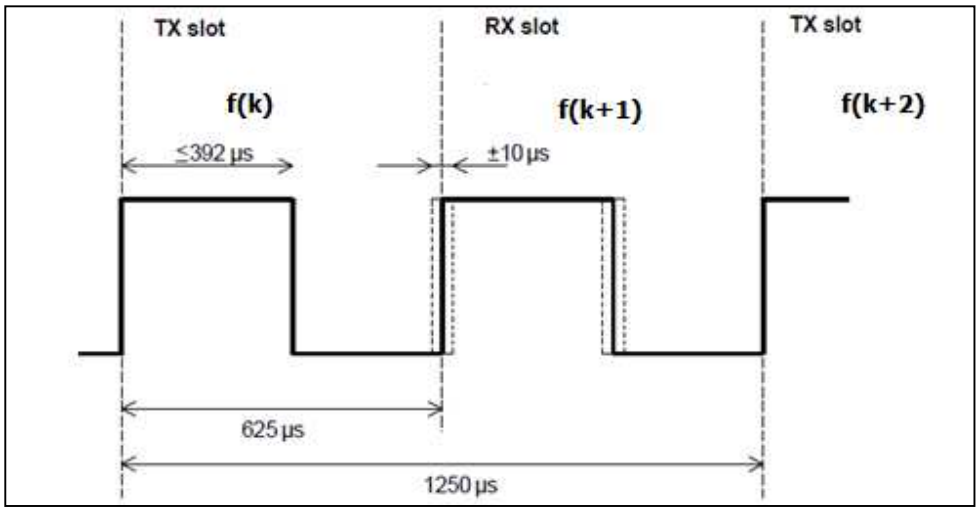


Figura 3.12. Ciclo de transmisión y recepción de paquetes en sentido maestro-esclavo.

Por conservar la sincronización y mantener al mínimo los errores de transmisión, se establecen períodos de guarda para el envío y recepción de paquetes. Es así que durante el modo de operación normal de la conexión, existe una ventana de 20 [uS] al inicio de cada slot de tiempo, que permite que un paquete pueda llegar hasta 10 [uS] antes o después del inicio exacto del slot calculado por el maestro. Se presenta una recomendación especial para los esclavos de establecer una ventana de hasta 250 [mS] para comunicarse con el maestro de la Piconet, para mantener la estabilidad de la misma.

Los esclavos durante un ciclo de recepción (RX slot) deben utilizar su código de acceso para hallar una correlación con el canal de comunicación correcto. Si no existe tal correlación entonces el dispositivo ira a modo de espera o *sleep* hasta el siguiente ciclo de recepción. Por otro lado, si existe un evento en el canal, el dispositivo permanecerá alerta para recibir el paquete a menos que sea para otro dispositivo o se detecte un error en el mismo.

- **Sincronización**

En el canal físico de Bluetooth, la sincronización de un esclavo puede verse grandemente afectada si es que no recibe un paquete en un máximo de 250 [ms]. Por ello, resulta importante hacer una re sincronización que consiste en que el dispositivo esclavo primero escucha al dispositivo maestro antes de enviar cualquier tipo de información. El término escucha implica que el esclavo únicamente debe recibir paquetes del maestro pero, por otro lado, el maestro debe enviar procurar que tales paquetes sean pequeños en los slots de tiempo definidos.

El tiempo que puede tomar la re-sincronización es de mínimo 20 [us], pero si el tiempo de búsqueda del maestro excede de 1250 [us] se debe evitar el solapamiento de las ventanas que se producen. Para ello se pueden utilizar dos técnicas:

1. Centrar a las ventanas de búsqueda en f(k), f(k+4)…F(k+4i) donde i es un entero y el producto da un valor máximo de 2500 [us]

2. Centrar a las ventanas de búsqueda en f(k), f(k+6)…F(k+6i) donde i es un entero y el producto da un valor máximo de 3750 [us]

En la figura 3.13 se muestra un esquema del funcionamiento de la búsqueda del maestro para la re sincronización del esclavo, en este caso el valor X [us] no excede el máximo de 1250 [us].

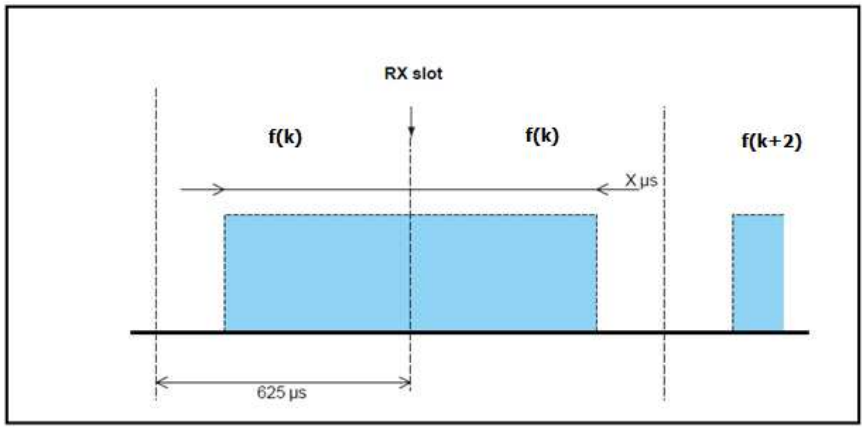


Figura 3.13. Establecimiento de tiempos recomendado para un esclavo durante la re sincronización

3.2.2 Canal Adaptativo Piconet

Es similar al canal básico con la diferencia de que para el canal adaptativo *Piconet* se debe usar al menos un número igual a 20 canales RF, y no los 79 usados por el otro tipo de canal. Además, utiliza la secuencia de salto entre canales adaptativos. Cabe mencionar que los dispositivos deben ser capaces de dar saltos adaptativos en frecuencia (AFH) para poder utilizar este canal.

Otra de las diferencias radica en la selección de frecuencia para el esclavo que es siempre adyacente a aquella utilizada por el maestro para la transmisión. Por otro lado, en lo referente a la comunicación, sincronización y temporización en la comunicación este canal se comporta de manera similar al básico.

3.2.3 Canal de escaneo y búsqueda

Este canal es definido al principio de la transmisión entre un maestro y un esclavo. La definición de maestro es similar a la que se tiene en un canal básico; es decir, que se conoce como maestro al dispositivo que comienza con la búsqueda y como esclavo al que es encontrado en mencionado proceso. El reloj utilizado durante esta etapa es el CLKE, sobre el cual se discutió anteriormente.

Su funcionamiento se fundamenta en que el maestro, durante la búsqueda, envía constantemente mensajes a los dispositivos. Tales mensajes son cortos y por ello la tasa de salto en frecuencia del maestro es de 3200 saltos/s. Debido a la corta longitud, es posible que el maestro envíe un sólo mensaje en dos frecuencias f(k), f(k+1) en un mismo slot de tiempo. Por ello, cuando un dispositivo es encontrado, utiliza el slot destinado para la recepción para enviar mensajes en respuesta al maestro en las mismas frecuencias en las que los recibió. En la figura 3.14 se muestra un esquema de transmisión y recepción entre el maestro y esclavo donde el maestro envía paquetes en las frecuencias f(k) y f(k+1) y las halla en el slot de recepción por parte del dispositivo esclavo.

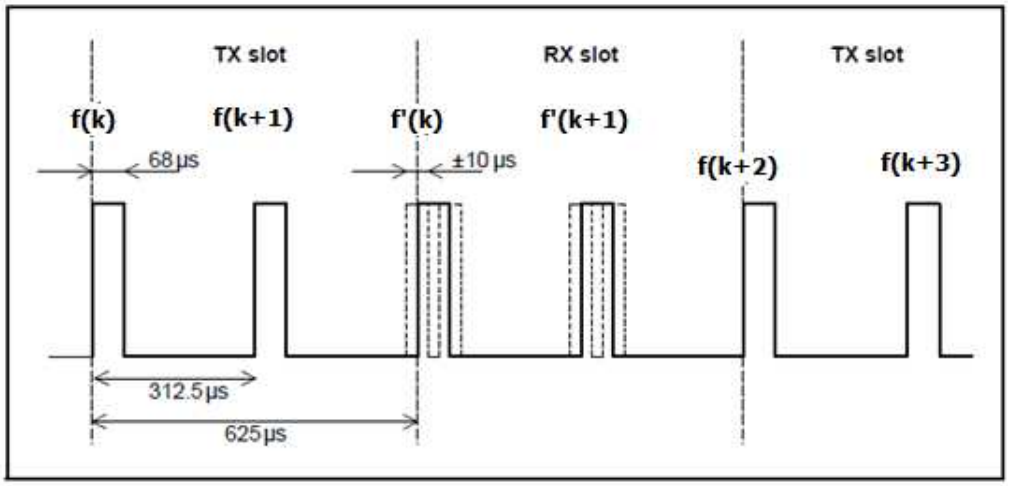


Figura 3.14. Utilización del medio en el canal de escaneo y búsqueda.

Durante la comunicación no todos los mensajes que envía el maestro deben tener respuesta por parte del esclavo. Es así, como en la figura 3.15 se muestra que son enviados dos mensajes en el primer slot desde el maestro al esclavo. El esclavo

únicamente recibe el primer paquete y envía una respuesta en la misma frecuencia pero en el siguiente slot. Así por ejemplo, si consideramos que el mensaje desde el maestro fue enviado en una frecuencia f(k) la respuesta por parte del esclavo será en la misma frecuencia y el siguiente mensaje enviado por el maestro se ubicará en la frecuencia f(k+1); además, si cada ventana ocupa 625 [us] la diferencia en tiempo entre ambos mensajes será de 1250 [us].

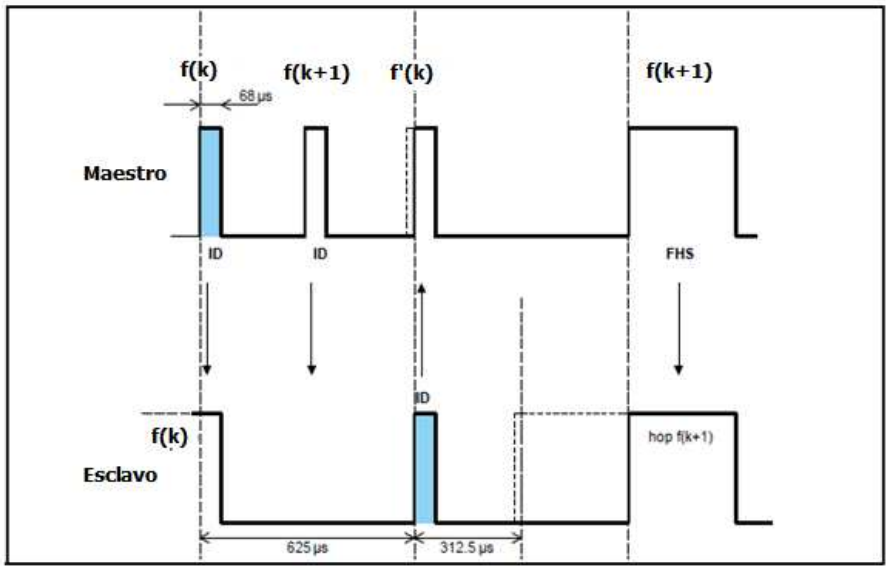


Figura 3.15 Comunicación maestro/esclavo en canal de escaneo y búsqueda

Para la recepción cabe recalcar que el ajuste en sincronización se da según el mensaje enviado por parte del maestro y no de la respuesta por parte del esclavo. Es así que si, por ejemplo, hubiese sido recibo el segundo mensaje en la frecuencia f(k+1) de la figura 3.15, el mensaje de respuesta debe ser en ese mismo tiempo a la misma frecuencia pero la comunicación en adelante debe ir según lo que indica el maestro y no el último mensaje enviado por el esclavo, como se observa en la figura.3.16 sería un error que el esclavo envíe la respuesta a mitad de cada slot, puesto que la selección de frecuencia se hizo al inicio y la respuesta se espera siempre al inicio del mismo.

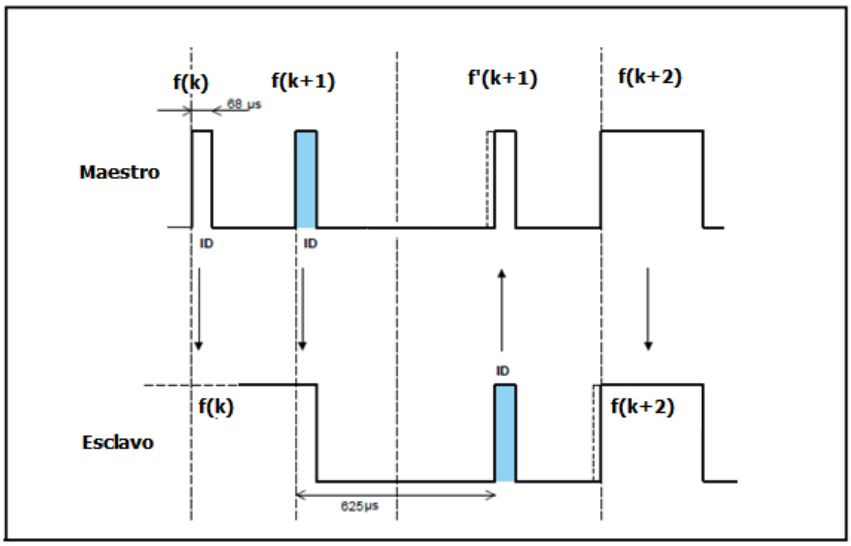


Figura 3.16 Comunicación maestro/esclavo en canal de escaneo y búsqueda

3.3 Enlaces Lógicos

De la misma manera en que se necesita establecer un canal físico entre uno o varios dispositivos Bluetooth, es necesario establecer un enlace lógico a nivel de capa de enlace. Básicamente se definen cinco tipos de enlaces lógicos para Bluetooth que permiten transmitir la información de diferente manera, como se muestra a continuación:

- *Link Control (LC)*
- *ACL Control (ACL-C)*
- *User Asynchronous/Isochronous (ACL-U)*
- *User Synchronous Data Logical Link(SCO-S)*
- *User Extended Synchronous Data Logical Link (eSCO-S)*.

3.3.1 Link Control (LC)

Capa de control de enlace. Permite llevar información de bajo nivel de enlace como control de errores o flujo. Este enlace está presente en todas las cabeceras de los paquetes.

3.3.2 ACL Control (ACL-C)

Administración del enlace. Permite llevar información de control intercambiada entre el maestro y el esclavo de manera confiable y limitada en el tiempo. El canal ofrece conectividad bidireccional y enlaces punto a punto. Este tipo de enlaces tienen prioridad sobre aquellos ACL-U, SCO-S y eSCO-s cuando se tiene saturación en el canal de comunicación.

3.3.3 User Asynchronous/Isochronous (ACL-U)

Este enlace permite llevar información asíncrona o isócrona de la capa L2CAP, cuyas características se verán más adelante. Este enlace lo que permite es transmitir los mensajes de la capa L2CAP en uno o más paquetes de banda base sin alterar el orden de los mismos y asegurando confiabilidad de la transmisión.

3.3.4 User Synchronous (SCO-S)

Este tipo de enlace transmite de manera transparente la información de manera síncrona. Debe ser utilizado para las transmisiones de voz para dispositivos Bluetooth. Dependiendo de la implementación en un dispositivo se puede tener un máximo de hasta tres conexiones síncronas simultáneamente. La comunicación puede ser bidireccional, simétrica, punto-punto, para canales de audio y video (AV) con una velocidad constante de 64 Kb/s.

3.3.5 User Extended Synchronous Data Logical Link (eSCO-S)

Al igual que el enlace anterior transporta información síncrona de manera transparente. La transmisión puede darse de forma bidireccional, simétrica o asimétrica y punto-punto. La sincronización se da con el reloj del dispositivo maestro Bluetooth (CLK).

3.4 Seguridad

Bluetooth provee diferentes niveles de seguridad sobre un enlace. Es así que se definen cuatro etapas de seguridad para un enlace:

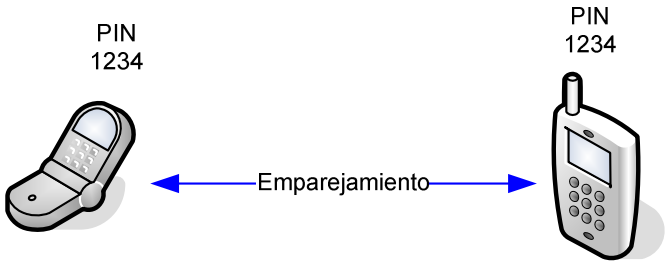
- Emparejamiento (Pairing).

- Autenticación (Authentication)
- Encriptación (Encryption)
- Autorización (Authorization)

3.4.1 Emparejamiento

Se da cuando dos dispositivos se encuentran por primera vez y necesitan establecer un enlace seguro. Para ello deben compartir una clave secreta (*secret shared key*) que permitirá autenticar a ambos dispositivos en el inicio de la comunicación. Tal clave es conocida como PIN y debe ingresarse en ambos dispositivos. La siguiente figura muestra el proceso de cómo dos dispositivos deben intercambiar la clave PIN para iniciar el proceso.

Este proceso es transparente para cualquier aplicación y es responsabilidad de cada dispositivo conocer cuál es el valor correcto del PIN. Después del proceso inicial, se guarda una clave secreta compartida dentro de cada dispositivo Bluetooth para permitir la validación de los dispositivos sin tener que repetir el proceso de emparejamiento.



3.17 Proceso de emparejamiento utilizando un valor de pin común

3.4.2 Autenticación

Verifica la identidad de un dispositivo mediante un esquema de intercambio de mensajes desafío y respuesta, similar a la autenticación CHAP de PPPoE. Cabe mencionar que la autenticación no valida usuarios sino dispositivos.

Como se muestra en la figura 3.18, el proceso inicia cuando un dispositivo A quiere autenticar a un dispositivo B enviándole un desafío. Cuando el dispositivo B lo recibe aplica un algoritmo de clave compartida y envía la respuesta al dispositivo A, quién debe realizar el mismo proceso internamente para saber si la respuesta enviada es la correcta,. A la final se autentica la identidad del dispositivo B al dispositivo A, pero no viceversa, por lo que es necesario realizar el proceso inverso para autenticar el dispositivo A con el dispositivo B.

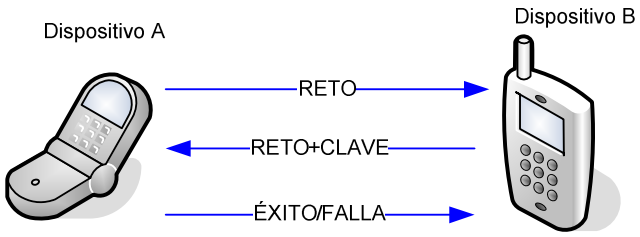


Figura 3.18. Esquema del proceso de autenticación.

Finalmente es necesario indicar que siempre y cuando la autenticación sea exitosa se podrá pasar al proceso de encriptación, de lo contrario es imposible

3.4.3 Encriptación

Su principal función es la de codificar los datos enviados entre los usuarios dentro de una conexión para evitar que elementos externos puedan obtener información acerca de la misma.

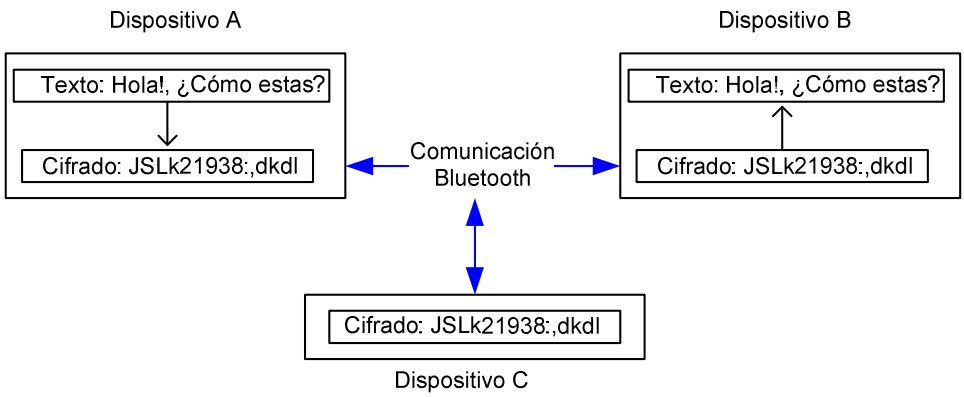


Figura 3.19 Esquema de funcionamiento de encriptación.

Para iniciar con la etapa de encriptación resulta necesario que un dispositivo, de los dos que han establecido una conexión, haga una petición al otro para codificar los paquetes de la comunicación. Si ambos dispositivos acuerdan el envío de la información de forma encriptada se procede a hacerlo, de lo contrario se cierra la conexión debido a que es imposible que envíe información encriptada únicamente en un sentido. En la figura 3.19 se muestra como el intercambio de información puede ser posible entre los dispositivos que han acordado utilizar encriptación y no entre terceros, ofreciendo un nivel de seguridad a nivel de capa de enlace.

3.4.4 Autorización

Esta etapa funciona de manera independiente para cada conexión que un dispositivo haga con otro en una comunicación. Es por ello que permite conocer cuándo una petición de conexión de un dispositivo Bluetooth debe ser aceptada.

En esta etapa aparece el concepto de dispositivo de confianza (*Trusted Device*) como aquel que tiene garantizada la autorización para cualquier tipo de petición que realice a un dispositivo Bluetooth local. El manejo y mantenimiento de los dispositivos de confianza depende exclusivamente de cada dispositivo.

Los niveles de seguridad se dan de manera secuencial, por lo que estar en un nivel indica implícitamente que otros están presentes. Por ejemplo, la encriptación requiere de autenticación y a su vez está necesita del emparejamiento. Resulta evidente que es imposible omitir niveles o que una aplicación no los ejecute de manera secuencial.

4. Implementación de la Arquitectura Bluetooth.

La implementación de la arquitectura Bluetooth en un dispositivo se divide en capas, algo similar a la estructura de la pila OSI. Básicamente podemos definir dos grandes componentes en la estructura de la arquitectura Bluetooth:

- Bluetooth host.- Involucra a todas las capas superiores y es usualmente implementada en software. Se encuentra generalmente integrada con el sistema operativo del dispositivo y permite el desarrollo de servicios mediante aplicaciones. Además, gracias a este componente es posible la construcción de los perfiles de servicio, que se discutirán más adelante.

- Bluetooth Controller (Bluetooth radio module). – Representa la parte física que brinda la conectividad Bluetooth. Puede ser interno o externo en el diseño del dispositivo. Si es externo, debe tener algún tipo de componentes de entrada y salida, por ello suelen tener conexión *Peripheral component microchannel interconnect architecture (PCMCIA)*, *universal asynchronous receiver-transmitter (UART)* o *universal serial bus (USB)*

Resulta evidente que debe existir una interfaz para conectar la parte física con el software implementado en un dispositivo Bluetooth. Tal interfaz se conoce como *Bluetooth Host Controller Interface* (HCI) y provee un conjunto de comandos al radio, controlador de banda base y administrador de enlace. HCI es una interfaz estándar única para acceder a las características de banda base, estado del hardware y control de registros. Actualmente, cuando se tiene en un solo chip la parte de radio y

software para Bluetooth, el uso de la interfaz HCI se ve disminuida por procesos internos del dispositivo.

La arquitectura Bluetooth está conformada por varios protocolos. En la figura 3.18 se muestran aquellos protocolos fundamentales dentro de la arquitectura y que tienen algún tipo de control directo por software, sobre todo por parte de JAVA. Es importante indicar que existen protocolos específicos creados para la arquitectura como SDP (Service Discovery Protocol) y adaptados como OBEX (Object Exchange Protocol), que originalmente era un protocolo para IrDA [52].

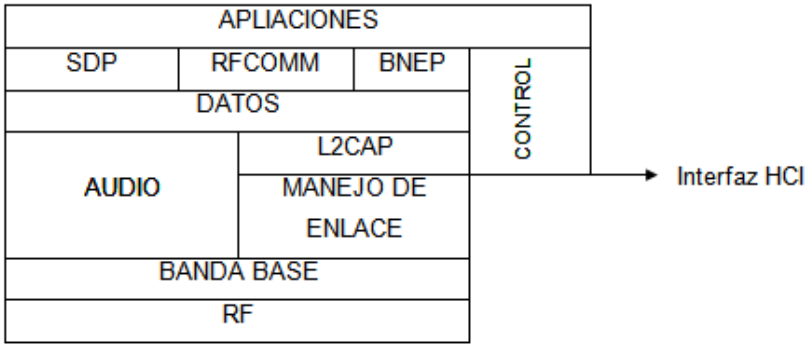


Figura 3.20 Pila de protocolos para Bluetooth

A continuación se presenta un detalle de los protocolos mostrados en la figura 3.18 y que forman parte de la pila que permite la comunicación Bluetooth:

- Bluetooth radio RF.- Define los requerimientos del transceptor del dispositivo que opera a 2.4 Ghz.
- Banda Base y control.- Permite establecer un conexión de radiofrecuencia entre los dispositivos Bluetooth cuando se desea una conexión. La parte de banda base maneja el procesamiento de canales y temporización; mientras que el control de enlace maneja el acceso al canal.
- Audio.- En la especificación Bluetooth se lo trata de manera única. El audio es enrutado directamente desde y hacia la capa de banda base (*Baseband Layer*) en un enlace SCO. Debe tomarse en cuenta que si existe una conexión

de voz sobre IP (VoIP) se utilizará ACL, ahí radica la diferencia del tratamiento que da Bluetooth a la voz y lo que le permite tener QoS.

- Manejo de enlace (LMP).- Se encarga de establecer el enlace mediante la configuración entre los dispositivos Bluetooth. También de administrar y negociar el tamaño de los paquetes. Además, se encarga de aspectos de seguridad como sincronización y encriptación.
- L2CAP.- Funciona como enlace entre las capas superiores e inferiores al multiplexar las diferentes conexiones lógicas hechas por las primeras.
- SDP.- Proporciona un medio para que las aplicaciones puedan requerir de servicios y características del mismo. Además, el conjunto de servicios disponible cambia dependiendo del ambiente y de las condiciones; por ejemplo, dispositivos que se encuentran en constante movimiento. Por tanto, SDP es diferente a una red tradicional y necesita estar constituida directamente por sobre L2CAP para brindar una comunicación confiable.
- RFCOMM.- Funciona como una especie de puerto serial virtual. Es una emulación de puerto serial sobre L2CAP. RFCOMM provee posibilidades de transporte para las capas superiores que necesitan utilizar una interface serial como mecanismo de comunicación. Provee múltiples conexiones concurrentes a un dispositivo o también diversas conexiones a múltiples dispositivos.
- BNEP (Bluetooth Network Encapsulation Protocol).- encapsula los paquetes de varios protocolos de networking y estos son transportados directamente sobre L2CAP. Se considera a BNEP como el paquete común de facto que permite el intercambio de información en redes Bluetooth; fue especificado en la versión 1.1.

Actualmente se han definido varios protocolos nuevos construidos por sobre los ya mencionados anteriormente, ubicados en la capa de aplicación. Por ejemplo:

- Audio/Video Control Transport Protocol.- Permite controlar el flujo de audio y video.

- Video Streaming Protocol.- Que permite definir la forma y sincronización cuando se desea dar el servicio de video en vivo para una conexión Bluetooth bajo la especificación 3.0
- Audio/Video Distribution Protocol.- Diseñado para la transmisión de video en multicast en redes Bluetooth.

Cabe mencionar que al contrario de las redes de área local (LAN) donde se encuentra el servicio y luego a los usuarios, en Bluetooth se buscan primero los usuarios y luego se accede a los servicios.

Existen tres protocolos a los cuales debemos prestar especial interés, RFCOMM, OBEX y L2CAP. La utilización de tales protocolos es básica al momento de realizar cualquier tipo de comunicación en Bluetooth.

4.1 Logical link control and adaption protocol (L2CAP)

Esencialmente, es una capa que permite multiplexar los accesos de las aplicaciones y capas superiores al medio para transmisiones Bluetooth, excepto para aquellas de establecimiento de circuitos de voz. Como se puede observar en la figura 3,18, L2CAP se encuentra por encima de HCI y existen varios protocolos que se encuentran sobre ella para obtener acceso a la transmisión por radiofrecuencia.

La importancia de L2CAP radica en que oculta los detalles de transmisión en banda base y únicamente presenta sus conexiones y paquetes a las capas superiores, facilitándoles los procesos de transmisión.

Al establecer una conexión sobre L2CAP estamos estableciendo una conexión asíncrona para la banda base entre dos dispositivos Bluetooth; cuando la comunicación es bidireccional se le conoce como canal orientado a conexión (*connection-oriented channels*) pero cuando es en una sola dirección se las conoce como canales no orientados a conexión (*connectionles channels*).

La figura 3.19 muestra claramente como L2CAP permite multiplexar varias comunicaciones que están produciendo simultáneamente entre dos dispositivos Bluetooth. En este caso se establecen dos comunicaciones independientes para una aplicación de capa superior y otra para RFCOOM, ambas full-dúplex.

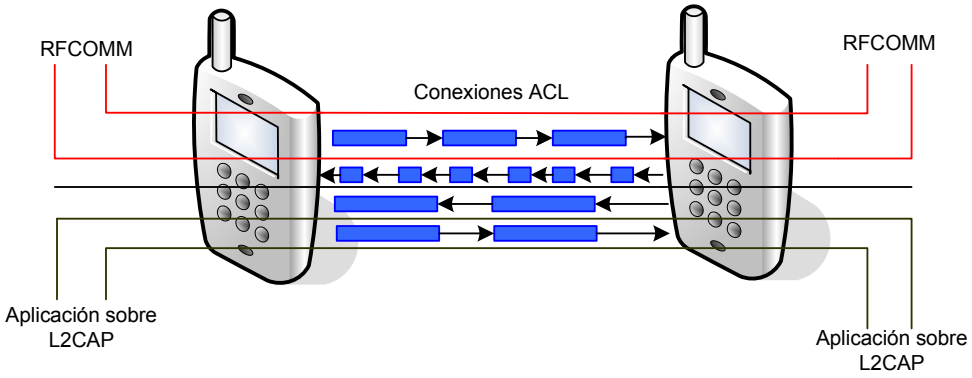


Figura 3.21 Funcionamiento de L2CAP

El tamaño de los paquetes que se envían por las conexiones L2CAP establecidas tiene un límite establecido previamente entre ambos dispositivos y conocido como *maximum transmission unit (MTU).* El valor de MTU puede ser diferente para una misma conexión pero en diferentes sentidos, todo va depender de la negociación inicial entre los dispositivos. Además, un paquete L2CAP debe ser dividido en una o varias partes para ser transmitidos en banda base, debido a que el máximo tamaño de un paquete en L2CAP es de 65535 bytes, un valor mucho más alto que los 339 bytes máximo para un paquete en banda base.

Otro de los beneficios de esta capa radica en su flexibilidad para adaptarse a diferentes perfiles y protocolos de capa superior. Inclusive siendo posible que en la interfaz de programación de aplicaciones (API) se pueda definir la forma en que se envían los bytes en paquetes y el control sobre el envío de los mismos en banda base de manera bastante sencilla. Sin embargo, si tal nivel de control sobre la transmisión en banda base no es necesario es mejor utilizar un protocolo de más alto nivel como OBEX y RFCOMM

4.2 RFCOMM

Este protocolo permite emular una conexión serial realizada por un puerto RS-232 entre dos dispositivos de forma inalámbrica. Por tanto, la información es enviada en ráfagas. Además, RFCOMM está basado en el estándar TS 07.10 que no hace distinción entre DCE y DTE en la comunicación serial [54]y al estar basado en capas inferiores permite establecer transmisiones confiables

Tabla 3.6 Especificaciones básicas de RFCOMM

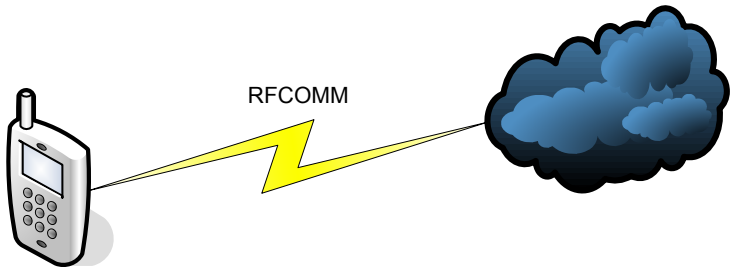
Parámetro	Valor
Tamaño máximo de la trama	23-32767(127 bits por defecto)
Tiempo de ACK	10-60 (20 s recomendado)

RFCOMM permite tener hasta 60 conexiones simultáneas entre dos dispositivos Bluetooth, según se indica específicamente en la implementación de la tecnología [55].

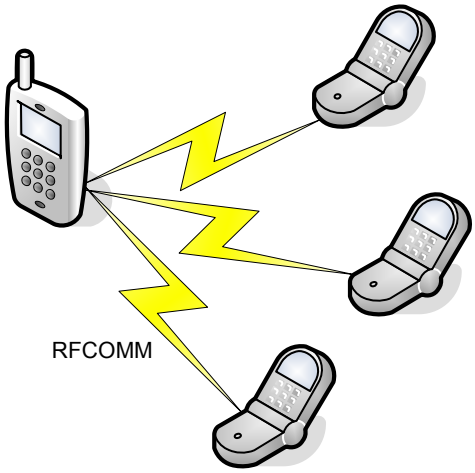
Las aplicaciones para RFCOMM están relacionadas con el reemplazo de cables para la comunicación serial. Se pueden presentar varios casos en la forma de transmisión que puede establecer RFCOMM, en la más simple la comunicación no es más que un enlace Bluetooth desde un dispositivo a otro. Por otro lado, si uno de las dos terminaciones es una red, RFCOMM utiliza la tecnología Bluetooth como un módem para acceder a la red. Finalmente, una configuración más compleja se da cuando se tienen varias interfaces, como en el caso en el que se utilicen módulos Bluetooth que permiten conectar a un dispositivo a varias redes diferentes, como se muestra en la figura 3.20

RFCOMM identifica entre dos tipos de dispositivos. Los primeros son dispositivos finales de comunicación; por ejemplo, computadores personales o impresoras. La otra clase de dispositivos son aquellos que son parte de un segmento de comunicación, por ejemplo módems. Sin embargo, el protocolo no hace distinción entre los dos tipos de dispositivo y les sirve de igual manera. La división se da por el tipo de información que necesitan manejar. Es así que los dispositivos de

comunicación necesitan únicamente cierto tipo de información y no la que se enviaría a los dispositivos de usuario final. Por ello, resulta importante que los fabricantes identifiquen la clase de dispositivos a fabricar y con quiénes puede intercambiar información ya que el protocolo no permite establecer diferencias entre dispositivos de comunicación y finales.



a) Conexión como módem a una red existente



b) Múltiples conexiones RFCOMM para un dispositivo

Figura 3.22 Diferentes formas de conexión utilizando RFCOMM.

4.2.1 Emulación de múltiples conexiones seriales

Como se mencionó anteriormente, dos dispositivos pueden tener múltiples conexiones seriales abiertas al mismo tiempo, teóricamente hasta 60 pero depende mucho de la implementación Bluetooth realizada en los dispositivos.

Para el manejo de múltiples conexiones se introduce el concepto de *Data Link Connection Identifier (DLCI)* que permite identificar la conexión de una aplicación en un entorno de cliente-servidor mediante un numero único entre los dos dispositivos. DLCI está compuesto por 6 bits pero en la práctica la numeración se da desde el 2 al 61, debido a que 0 se utiliza para control y 1,62 y 63 están reservados. El manejo de los DLCI depende directamente de cada dispositivo y son requeridas por aplicaciones que se ejecuten en el mismo. En la figura 3.21 se muestra un esquema de la emulación de múltiples conexiones seriales entre dos dispositivos.

Figura 3.23 Mútiples enlaces seriales emulados utilizando RFCOMM

4.3 OBEX

Es un protocolo independiente del medio que se utiliza para el envío de información mediante el establecimiento de sesiones. El protocolo se encuentra jerárquicamente sobre RFCOOMM y L2CAP, como se muestra en la figura 3.22. En un sistema Bluetooth, el propósito del protocolo OBEX es el permitir el intercambio de objetos de datos. El ejemplo típico es el de enviar una tarjeta de negocios.

Está basado en el protocolo definido para la tecnología Infrarroja, denominado como IrOBEX y se presenta como una alternativa al conocido protocolo de transporte de hipertexto (HTTP) para dispositivos con memoria y capacidades de procesamiento limitadas. Debido a que mientras HTTP da una respuesta por cada petición, IrOBEX permite dividir a las peticiones y respuestas en pequeñas partes, lo que facilita su procesamiento y la facilidad de cancelar una operación.

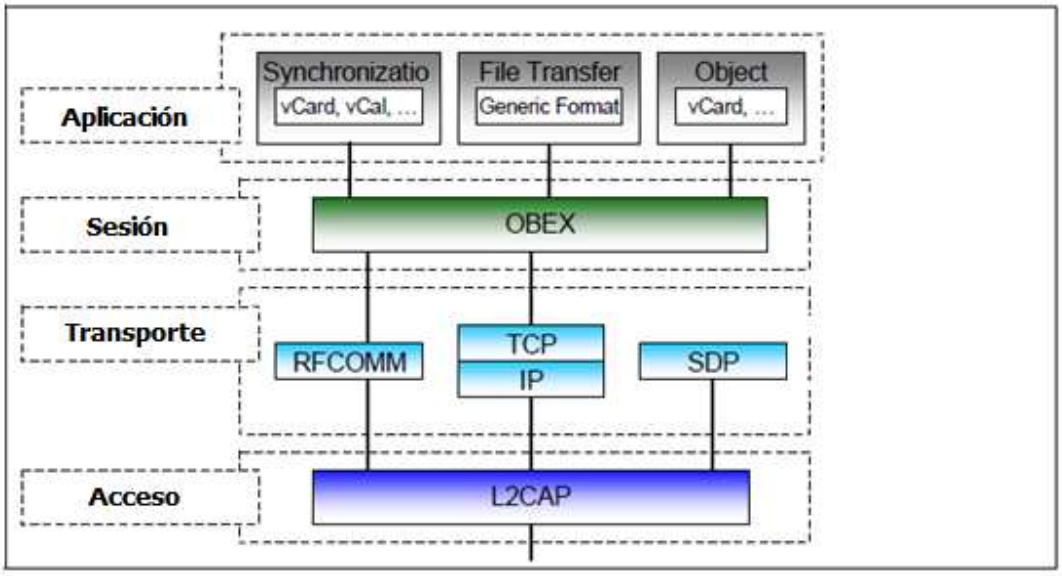


Figura 3.24 Jerarquía de Protocolos Bluetooth.

OBEX se encuentra íntimamente relacionado con el perfil GOEP para el intercambio de paquetes entre dispositivos Bluetooth. El campo de aplicación de esta tecnología es bastante amplio ya que involucra todos aquellos procesos de intercambio de información entre dispositivos móviles con Bluetooth como envío y recepción de imágenes, videos, sonidos, tarjetas de presentación, sincronización de dispositivos, conectividad con periféricos como impresoras, agendas personales electrónicas (PDAs), envío de mails, entre otras aplicaciones.

El entorno de funcionamiento del protocolo es mediante una sesión cliente-servidor. Y las operaciones en OBEX se manejan mediante peticiones, enviadas por el cliente, y respuestas, enviadas por el servidor. Después de enviar una petición el cliente espera hasta obtener una respuesta del servidor hasta enviar la siguiente petición.

Una frase tomada del libro de Kumar, Kline y Thompson [51] indica realmente cuál es el beneficio de usar OBEX como protocolos de sesión: “Al usar protocolos

como RFCOMM o TCP/IP se requiere que las aplicaciones conozcan cómo los datos son enviados y cuando enviar una respuesta, mientras que OBEX esconde este procedimiento dentro del protocolo. OBEX da sentido a los datos que se envían mientras que RFCOMM y TCP/IP simplemente envían bytes."

4.3.1 Sesión OBEX

Durante el intercambio de mensajes en una sesión OBEX se distinguen seis operaciones básicas sobre las cuales trabaja el protocolo:

- CONNECT

Toda conexión de OBEX bajo Bluetooth inicia con una petición CONNECT por parte del cliente al servidor. Al inicio de la conexión el cliente puede incluir cabeceras adicionales si desea. Cuando el servidor recibe la petición, el servidor la procesa y decide si la aceptará (OK, SUCCESS) o no. Cuando el servidor rechaza la petición manda un código de respuesta que especifica el porqué la petición no fue aceptada.

La figura 3.23 se muestra un esquema de inicio de sesión entre dos dispositivos. Se observa que se incluye un campo denominado *count* que permite que el cliente indique los objetos que serán transmitidos. En el caso de la conexión de la figura 3.23, el servidor acepta la petición sin inconvenientes.

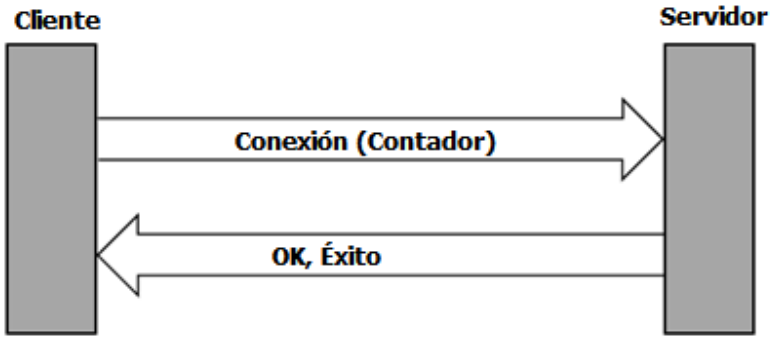


Figura 3.25 Inicio de sesión entre dos dispositivos bajo OBEX

En la figura 3.24a se muestra el formato del paquete que es enviado como petición desde el cliente al servidor, mientras la figura 3.24b muestra cuál es la

respuesta enviada desde el servidor. Cabe mencionar, que el bit menos significativo es el 7 y el más significativo es el 0 dentro del paquete.

Byte 0	Byte 1 y 2	Byte 3	Byte 4	Byte 5 y 6	Byte 7 hasta n
0x80	Petición de conexión	Versión de OBEX	Bandera	Longitud máxima de paquete	Cabeceras opcionales

a) Paquete de inicio de sesión enviado desde el cliente

Byte 0	Byte 1 y 2	Byte 3	Byte 4	Byte 5 y 6	Byte 7 hasta n
Código de respuesta	Respuesta a petición de conexión	Versión de OBEX	Bandera	Longitud máxima de paquete	Cabeceras opcionales

b) Respuesta de inicio de sesión enviada desde el servidor

Figura 3.26 Estructura de paquetes para inicio de sesión

Finalmente, cabe mencionar que una conexión no se termina cuando un archivo se terminó de transferir satisfactoriamente. La sesión se terminará posteriormente cuando se envíe el comando apropiado para hacerlo, de lo contrario seguirá abierta.

- SETPATH

Este comando permite que el usuario cambie la carpeta destino en el servidor. Por ejemplo, el cliente puede mandar una operación SETPATH con una cabecera NAME que especifica el nombre del directorio al cuál desea cambiarse. De igual manera, el servidor es el que decide si el cambio procede o no, pero si esta operación no es exitosa la conexión con el servidor no se pierde.

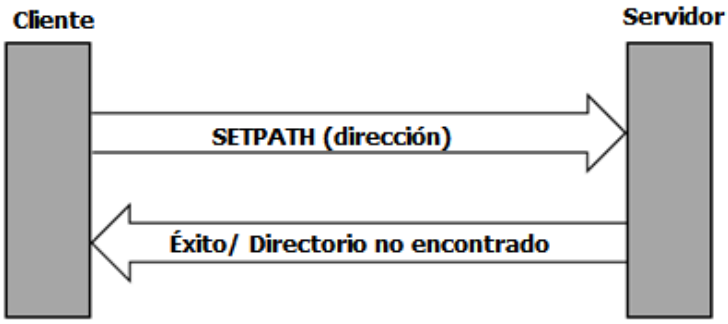


Figura 3.27 Intercambio de mensajes para el comando SETPATH

- PUT

Este comando permite que el cliente envíe un archivo al servidor. Si es el caso de que el archivo es demasiado grande y se lo debe dividir en partes más pequeñas es necesario que la primera petición PUT, además de contener la cabecera BODY con la primera parte del archivo, contenga una cabecera NAME que especifica el nombre del mismo. Cuando el servidor procesa la información responde con un mensaje CONTINUE y entonces el cliente envía el resto del archivo en cabeceras BODY hasta cuando envía una petición PUT con una cabecera END OF BODY para indicar que esa es la última parte del archivo que se estaba transmitiendo.

Cuando se ha completado la transferencia de un archivo, el servidor envía una respuesta confirmación (OK, SUCCESS) para informar al cliente que el proceso se ha realizado satisfactoriamente.

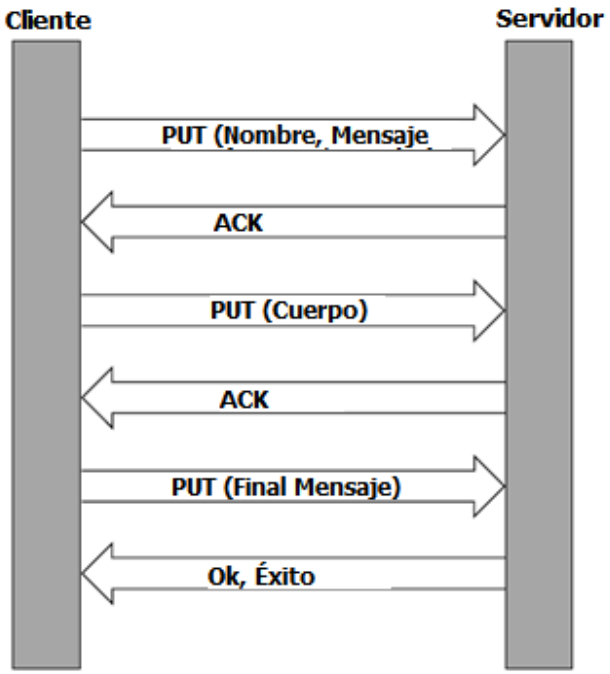


Figura 3.28 Intercambio de mensajes para el comando PUT

- GET

Cuando la conexión se ha establecido entre el cliente y el servidor, el cliente puede pedir archivos desde el servidor mediante una petición GET. Esta operación es similar a PUT, con la diferencia de que el sentido en el que se transmite la información es desde el maestro al esclavo.

El objeto retorna al cliente como una secuencia de cabeceras y el cliente debe enviar un paquete de petición por cada paquete de respuesta.

- ABORT

Es un tipo especial de operación que interrumpe una operación PUT/GET o también múltiples peticiones y respuestas PUT/GET que existan entre el cliente y servidor.

- DISCONNECT

Al finalizar una sesión OBEX el cliente debe enviar una petición DISCONNECT, que usualmente no tiene cabeceras adicionales aunque pueden ser incluidas. En la figura 3.27a se muestra el formato del paquete que es enviado desde el cliente para cerrar una sesión y en la figura 3.27b se muestra la respuesta y el código que es enviado por el servidor.

Al producirse una operación de desconexión el servidor libera los recursos que estaban al servicio del cliente. El servidor envía un mensaje OK, SUCCESS para indicar que la sesión finalizó correctamente.

Es importante mencionar que el cierre de una conexión OBEX no indica que el enlace físico entre el cliente y el servidor también lo haya hecho.

Byte 0	Byte 1 y 2	Byte 3
0x81	Longitud del Paquete	Cabeceras opcionales

a) Paquete enviado desde el cliente al servidor

Byte 0	Byte 1 y 2	Byte 3
0xA0	Respuesta de longitud del Paquete	Cabeceras opcionales

b) Paquete en respuesta enviado desde el servidor

Figura 3.29 Estructura de los paquetes del comando DISCONNECT

- PUT-DELETE

  Esta operación es similar a PUT pero con la diferencia que únicamente se incluye la cabecera NAME y no con BODY en los paquetes. Esta operación permite indicar al servidor que borre al objeto con el nombre que se indica.

- CREATE-EMPTY

  Es una operación PUT que permite crear un objeto sin información o vacío. Esta operación contiene la cabecera NAME y END OF BODY.

La especificación OBEX ha definido un grupo de cabeceras mínimo para el intercambio de paquetes y su codificación, los cuales se mencionan a continuación:

- NAME.- Nombre del objeto. Debe ser una cadena unicode
- LENGTH.- Tamaño del objeto.
- TYPE.- Especifica el *Multipurpose Internet Mail Extensions (MIME)* del objeto
- COUNT.- El cual especifica el número de objetos a enviar y recibirse al inicio de la conexión.
- DESCRIPTION.- Una breve descripción del objeto.
- HTTP.- Especifica una cabecera HTTP.
- BODY.- Información o parte en la que fue dividida la información a intercambiarse. Permite intercambiar mensajes con el servidor OBEX mediante peticiones PUT o REQUEST.
- END OF BODY.- La última parte del objeto enviado. Al igual que BODY permite intercambiar mensajes con el servidor mediante peticiones PUT o REQUEST.

Cabe mencionar que la especificación también permite añadir hasta 64 cabeceras definidas por el usuario, divididas en cuatro grupos con diferentes tipos de datos (Cadenas Unicode, enteros sin signo de 4 bytes, bytes, secuencias de bytes).

Se ha mostrado una introducción sobre los protocolos más importantes para la comunicación en Bluetooth, pero es la capa de aplicación la que permite que un usuario tenga una experiencia real de cómo se aplica la tecnología. Es así que existen aplicaciones comunes en Bluetooth que han sido definidas bajo el nombre de perfiles y que actualmente permiten identificar las características de los dispositivos y de cómo interactúan con los usuarios.

5. Perfil Bluetooth

Un perfil Bluetooth permite definir formas comunes de usar los protocolos y sus características como modelos que son usados para aplicaciones específicas de la tecnología. Al igual que los protocolos fueron definidos por Bluetooth SIG [53].

Los perfiles no están por sobre la pila de protocolos sino más bien están presentes en cada una de las capas. Un perfil puede indicar las características más importantes que necesita de diferentes capas. Por ello, un perfil puede usar diferentes características de protocolos de diferente nivel o de un mismo nivel según los requerimientos que se tenga.

Un dispositivo Bluetooth puede ser compatible con varios perfiles dependiendo de su aplicación y fabricante. Actualmente, se han definido varios perfiles Bluetooth, en áreas tan diversas que van desde la telefonía hasta la transmisión de video [53]; sin embargo, existen cuatro perfiles conocidos cómo básicos:

- *Generic Access Profile (GAP).-* Puede definirse como la base de otros perfiles y, de una u otra manera, todos están basados en este. Este perfil indica los procedimientos fundamentales para al establecimiento de conexiones entre dispositivos, lo cual incluye el descubrimiento de los dispositivos, manejo y

administración del enlace y procedimientos relacionados al uso de diferentes niveles de seguridad.

- *Serial Port Profile (SPP).-* Permite emular una conexión por cable serial. Este protocolo actúa directamente sobre el protocolo RFCOMM para establecer una comunicación serial que puede ser usada para el reemplazo de cables de comunicación.
- *Service Discovery Application Profile (SDAP).-* Permite el descubrimiento de servicios en dispositivos con conectividad Bluetooth. Este perfil debe ser usado por cualquier aplicación que desee obtener algún tipo de servicio en un dispositivo Bluetooth.
- *Generic Object Exchange Profile (GOEP).-* Es uno de los pocos perfiles que se pueden definir como abstractos porque no se implementa directamente sino define maneras en cómo los paquetes deben intercambiarse entre dos usuarios. Tiene como objetivo definir parámetros para el intercambio de información para casos concretos; tal es el caso de los perfiles constituidos sobre OBEX.

Los perfiles se comportan de manera jerárquica. Por ejemplo, el protocolo de transferencia de archivos está constituido sobre GOEP, que depende de SPP y este a su vez está basado en GAP. La figura 3.28 muestra las relaciones entre varios protocolos Bluetooth definidas por el SIG para la especificación 4.0 [53].

Otra forma de clasificar a los perfiles también es por su funcionalidad. Así, los protocolos básicos son conocidos como *Transport profiles* sobre los que se pueden construir otros aplicativos conocidos como *application profiles*.

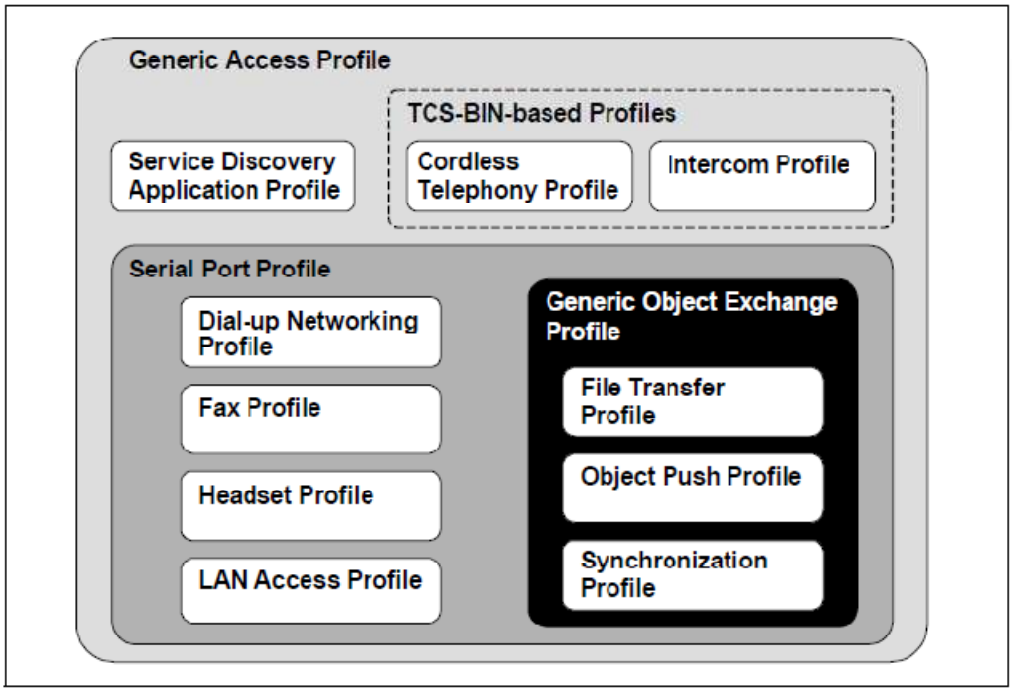


Figura 3.30 Estructura de constitución de varios perfiles Bluetooth

Una vez definidos los perfiles es posible entender las aplicaciones de la tecnología. Prácticamente existen perfiles definidos para la mayoría de casos en los cuales Bluetooth puede ofrecer soluciones. Pero sin duda, lo que ha permitido la diversificación de los campos de aplicación de la tecnología ha sido su continuo avance mediante la investigación que se da fuera de laboratorio gracias a varias empresas medianas y pequeñas que conforman el 70% de los miembros del Bluetooth SIG[56].

6. Tendencias actuales para Bluetooth

Atrás quedaron los días en los cuales Bluetooth era utilizado únicamente para conectar periféricos a una PC. Actualmente, tenemos que el desarrollo de Bluetooth está enfocado en dos ejes, el primero es el de proveer conexiones con suficiente ancho de banda para transmitir video y voz de alta calidad de forma inalámbrica y el segundo el de proveer dispositivos que puedan estar presentes en casi todos los dispositivos que usa una persona diariamente para hacer realidad la computación

ubicua. Es así que tenemos a Bluetooth de alta velocidad y a Bluetooth de bajo consumo de energía; el primero enfocado en dar solución a las necesidades multimedia actuales y el segundo que permite incluir la tecnología Bluetooth en prácticamente cualquier dispositivo.

6.1 Bluetooth de bajo consumo de energía

Básicamente es una extensión de la versión estándar de la tecnología. Su importancia radica, como su nombre lo indica, en su bajo consumo de energía y en que es compatible con los dispositivos Bluetooth estándar cuando está trabajando en *dual mode*.

El diseño de esta tecnología tuvo la finalidad de simplificar el protocolo al máximo para disminuir el consumo de energía y acelerar el proceso de comunicación. Es así que existen dos versiones de este tipo de tecnología:

- *Stand Alone*.- Esta implementación permite incluir la tecnología Bluetooth en pequeños dispositivos como relojes, llaveros o todo tipo de sensores que tienen limitaciones de consumo de energía y que tienen tamaño limitado.
- *Dual Mode*.- Permite la comunicación con dispositivos Bluetooth estándar al estar implementado en un hardware compartido de mayor tamaño. Tal hardware permite obtener las ventajas del bajo consumo de energía cuando sea necesario pero también permite la comunicación con dispositivos Bluetooth estándar.

Existen únicamente tres canales a disposición para establecer la comunicación entre dos dispositivos, esto reduce los saltos en frecuencia y los tiempos para que dos dispositivos puedan establecer una conexión de segundos a milisegundos. Esta facilidad para establecer una conexión hace innecesaria la sincronización periódica, ahorrando aún más energía. Además, los algoritmos pseudoaleatorios para la selección de frecuencia resultan innecesarios, pues al ser únicamente tres canales los dispositivos saltan redundantemente entre esas frecuencias hasta hallar la más adecuada para la transmisión.

La tabla 3.7 muestra una comparativa entre las dos tecnologías Bluetooth donde se puede observar que con la tecnología de bajo consumo de energía no está contemplada la formación de Scatternets y la velocidad de transmisión es considerablemente más baja, esto es debido a que no se busca transmitir la información a altas velocidades entre varios dispositivos sino establecer enlaces punto a punto entre dispositivos de características limitadas.

Tabla 3.7 Comparativa de la tecnología Bluetooth

Característica	Tecnología Bluetooth clásica	Tecnología Bluetooth de bajo consumo
Frecuencia	2.4 [GHz]	2.4 [GHz]
Distancia	10 [m]	10 [m]
Velocidad de transferencia	1-3 [Mbps]	1 [Mbps]
Velocidad de transferencia para una aplicación	0.7-2.1 [Mbps]	0.2 [Mbps]
Nodos/ Esclavos activos	7-16777184	Ilimitados
Tiempo en enviar la información	100 [ms]	<6 [ms]
Transmisión de Voz	Sí	No
Topología de red	Scatternet	Bus en estrella
Consumo de potencia	1 (referencia)	0.01 – 0.5 (de la referencia)
Permite perfiles	Sí	Sí
Descubrimiento de servicios	Sí	Sí

Finalmente, cabe mencionar que esta tecnología está enfocada principalmente a dar solución al mercado industrial y médico, debido a que son dispositivos que necesitan tener alta resistencia a la interferencia, optimización en el uso de energía, sencillez en la comunicación y bajo costo. Esta tecnología ha despertado gran interés en los fabricantes de chips, pues su simplicidad hace que sea prácticamente aplicable a cualquier dispositivo electrónico existente [57].

6.2 Bluetooth Versión 3.0 +HS

Esta versión también deriva de la especificación estándar Bluetooth y lo que permite es utilizar el radio definido en IEEE 802.11 para obtener tasas de

transferencia más altas pero manteniendo la sencillez de establecer una conexión sin la necesidad de una infraestructura de red previa.

La búsqueda de aumentar el ancho de banda tiene la finalidad de permitir la transmisión de audio y video de alta calidad entre dos dispositivos Bluetooth. También busca reducir los tiempos de sincronización para archivos y el desarrollo de más aplicaciones en tiempo real basadas en Bluetooth.

La versión 3.0 + HS de Bluetooth se apega a las especificaciones de *Ulta Wide Band* (UWB). UWB se describe como una clase de radios que usan señales de banda ancha para alcanzar los objetivos de sus aplicaciones. El principal aspecto de UWB es el compromiso que se tiene entre el ancho de banda y los bajos niveles de transmisión que se debe tener para ser incluidos en dispositivos móviles y ser económicamente eficientes. Si bien los bajos niveles de potencia parecen ser una limitante estos se ven compensados por el excelente ancho de banda disponible que permite obtener transmisiones a altas velocidades, algo similar a lo que sucede con la tecnología de acceso múltiple por división de código (CDMA). La figura 3.29 muestra el aprovechamiento del ancho de banda frente a tecnologías de banda angosta que utilizan mayores potencias.

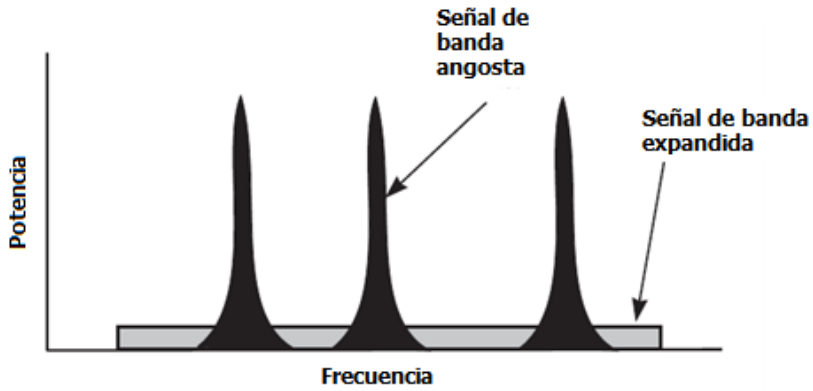


Figura 3.31 Comparativa entre señales de banda expandida y angosta

La tabla 3.8 muestra las características más importantes de la versión 3.0 +HS comparándolas con la versión predecesora 2.0 +EDR, donde se observa claramente

el aumento de velocidad que está íntimamente relacionado con la salida al mercado de los perfiles de transmisión de video de alta calidad y de multimedia en tiempo real.

Tabla 3.8 Comparativa entre la versión 2.1 y 3 de Bluetooth.

Característica	Bluetooth V2.1 + EDR	Bluetooth V3.0 + HS
Frecuencias	2.4 [GHz]	5 [GHz]
Alcance	10 [m]	10 [m]
Velocidad de transferencia	1-3 [Mbps]	54 [Mbps]
Velocidad de transferencia para una aplicación	0.7-2.1 [Mbps]	24 [Mbps]
Seguridad	Definida en capa de aplicación.	128b AES
Formación de Scatternet	Sí	Sí
Creación de perfiles	Sí	Sí

Bluetooth de alta velocidad espera ser implementado en la mayor parte de dispositivos multimedia existentes con la finalidad de brindar mayor ancho de banda a las conexiones.

CAPÍTULO 4

EL LENGUAJE JAVA

1. Introducción

Si bien la tecnología Bluetooth se ha tenido una alta difusión a lo largo de esta década, no es posible dejar de lado el tremendo impacto que ha tenido la programación en dispositivos móviles utilizando el lenguaje Java. En el año 2000 se marco un hito en la programación con la salida de J2ME y la propuesta de crear aplicaciones para dispositivos con características limitadas. Es así que hoy en día si conjugamos el número de dispositivos móviles que incluyen conectividad Bluetooth y aquellos que soportan aplicaciones Java tendremos más del 80% de celulares y PDAs existentes en el mercado global.

El futuro para este lenguaje de programación en dispositivos móviles es bastante prometedor. Por un lado está el hecho de que actualmente existen millones de aplicaciones escritas en este lenguaje, ya sean comerciales o de aficionados pero por otro que los nuevos sistemas operativos para teléfonos inteligentes como ANDROID de Google están basados en el mismo y poseen una alta compatibilidad con las aplicaciones escritas en J2ME.

Las características únicas de abstracción, herencia y portabilidad de Java son las que le convierten en un lenguaje perfecto para ser utilizado en dispositivos de características limitadas. Además, J2ME es una especificación bastante sencilla y ligera que puede ser implementada en casi cualquier dispositivo. Para Bluetooth, el lenguaje Java ha definido específicamente la recomendación JSR-82, que establece

métodos directos y claros para sacar provecho a la implementación Bluetooth de un dispositivo móvil, esta especificación también es conocida como JABWT.

En este capítulo se presenta una pequeña introducción al lenguaje Java y a las implementaciones J2ME y JABWT. Además, se mostrarán y analizarán las clases y métodos fundamentales tanto para establecer cuanto para gestionar las conexiones; pero sobretodo, se presentará la estructura básica de comunicación en un ambiente de cliente servidor mediante la utilización del protocolo OBEX para más adelante realizar la implementación de la aplicación de mensajería.

2. Lenguaje Java

En esta sección se dará una pequeña introducción a las palabras clave, elementos y sintaxis de la programación bajo este lenguaje. Sin embargo, cabe mencionar que Java tiene una sintaxis muy similar a la de lenguaje C ó C++ por lo que si se tiene experiencia programando bajo ese lenguaje será mucho más simple asimilar el contenido expuesto a continuación.

Como ya se mencionó en el capítulo uno, Java es un lenguaje totalmente orientado a objetos que elimina las herramientas de bajo nivel para minimizar los potenciales errores relacionados con el manejo de punteros y memoria. Básicamente un objeto está definido como una estructura que integra los datos, presentados mediante variables, y el código, mediante funciones o métodos. Es así que, un objeto se considera como una unidad en la cual su comportamiento puede ser visualizado por su código y su estado actual por sus variables o datos. Este manejo del software orientado a objetos se conoce como su paradigma y tiene ventajas como la reutilización de código, generalización de funciones o la simplicidad en la programación. Tales características han permitido que la programación orientada a objetos gane popularidad y que actualmente sea la más utilizada [58].

La segunda característica fundamental de Java es la independencia sobre la plataforma en la que corre gracias a la máquina virtual (JVM) sobre la que se ejecuta. La JVM se puede entender como un programa en código nativo del sistema operativo que interpreta el hardware y las funciones de bajo nivel del dispositivo. Es así que, la portabilidad está relacionada con que una aplicación en Java no se compile una sola vez, como aquellos programas en C++, sino que sean compilados justo en el momento en que se los ejecuta. Esta ejecución, que se la puede considerar como en tiempo real, puede reducir la eficiencia y rapidez de una aplicación si es que la máquina virtual no está perfectamente integrada con el hardware o si el mismo no es el adecuado para ejecutar la aplicación. Es por eso que un programa en Java genera un código conocido como *Bytecode* (*Java Bytecode*), que se lo puede considerar como un intermedio entre el código fuente de la aplicación y el código de máquina que se obtiene bajo una compilación.

Es necesario indicar que actualmente existen tres distribuciones de Java. J2SE es conocida como la versión básica del lenguaje y es aquella que incluye las interfaces de programación (APIs) necesarios para desarrollar programas en computadores personales. La distribución J2SE incluye mejoras en el manejo de variables y conexiones a bases de datos, con un enfoque empresarial. Y finalmente, se ha definido a J2ME como una especificación reducida pero totalmente funcional para dispositivos móviles con capacidades de procesamiento y memoria limitadas.

2.1 Variables y tipos de datos

Una variable debe ser definida por un tipo y un nombre, al igual que en el lenguaje C++. Al momento de definir las variables se deben tomar en cuenta que no se debe incluir espacios en blanco, tampoco que nunca dos variables pueden tener el mismo nombre y finalmente que una variable no puede utilizar palabras reservadas (como por ejemplo *if*). Además, existe una convención al momento de definir los nombres para variables y clases. Así, el nombre de una variable siempre comienza con minúscula mientras que el de una clase siempre lo hace con una letra mayúscula.

La similitud con el lenguaje C++ también se evidencia en los tipos de datos. En la tabla 4.1 se muestran los tipos de datos válidos para Java, el tipo de información que pueden representar y las especificaciones en las que se encuentra.

Tabla 4.1 Variables en J2ME

Tipo de dato	Información que representa	Distribución
Byte	Ocho Bits	Todas
Short	Número entero de 16 bits	Todas
Int	Número entero de 32 bits	Todas
Long	Número entero de 64 bits	Todas
Char	Carácter ASCII	Todas
Boolean	Valor verdadero o falso (excluyente)	Todas
Float	Datos con coma flotante (32 bits)	J2SE y J2EE
Double	Datos con coma flotante (64 bits)	J2SE y J2EE

De la tabla 4.1 también podemos observar que no es posible incluir variables con punto decimal para J2ME, la cual es una de las características que aligera la implementación. Por otro lado, las operaciones que se pueden realizar sobre las variables están definidas por cinco tipos de operadores básicos: de asignación, aritméticos, relacionales, lógicos, a nivel de bit.

2.2 Operaciones básicas en Java

2.2.1 Operadores de asignación

La operación de asignación está íntimamente relacionada con el símbolo =, lo cual le permite asignarle un valor a una variable, que si inicialmente no se lo hace en su declaración tiene un valor *null*.

2.2.2 Operadores aritméticos

Los operadores aritméticos básicamente permiten realizar operaciones matemáticas sobre los valores de las variables. Cabe diferenciar que existen operadores aritméticos unarios y binarios. Los operadores unarios son los que involucran una sola variable como por ejemplo ++, que permite incrementar su valor

actual en una unidad. Para los operadores unarios cabe recordar que si están tras la variable hacen el incremento antes de su uso mientras que si están después se hace uso del valor incrementado inmediatamente. Por otro lado, los operadores binarios permiten realizar operaciones aritméticas básicas entre al menos dos variables. En la tabla 4.2 se muestran los operadores binarios básicos utilizados en el lenguaje Java.

Tabla 4.2 Operadores aritméticos básicos en Java

Operación	Significado
a + b	Suma de a y b
a - b	Diferencia de a y b
a * b	Producto de a y b
a / b	Diferencia entre a y b
a % b	Resto de la división entre a y b

2.2.3 Operadores relacionales

Realizan operaciones que dan como resultado únicamente valores de verdadero o falso en respuesta a la comparación de dos variables. La tabla 4.3 muestra las operaciones relacionales permitidas en Java.

Tabla 4.3 Operadores relacionales en Java

Operación	Significado
a > b	Verdadero si a es mayor que b
a < b	Verdadero si a es menor que b
a >= b	Verdadero si a es mayor o igual que b
a <= b	Verdadero si a es menor o igual que b
a == b	Verdadero si a es igual que b
a ¡= b	Verdadero si a es distinto que b

2.2.4 Operadores Lógicos

De forma similar a las operaciones relacionales devuelven únicamente valores de verdadero o falso pero en respuesta a operaciones lógicas como Y, O y NO. La tabla 4.4 muestra las operaciones lógicas permitidas en Java.

Tabla 4.4 Operadores lógicos en Java

Operación	Significado
a && b	Y lógico
a || b	O Lógico
!a	Negación (No)

2.2.5 Operadores de Bits

Estas operaciones permiten manipular directamente los bits de una variable. Las operaciones permitidas se indican en la tabla 4.5.

Tabla 4.5. Operadores de bits en Java

Operación	Significado
a >> b	Desplaza los bits de a hacia la derecha b veces
a << b	Desplaza los bits de a hacia la izquierda b veces
a <<< b	Igual que la operación anterior pero sin el bit de signo
a & b	Suma lógica entre a y b
a | b	O lógico entre a y b
a ^ b	O exclusivo (xor) entre a y b
~a	Negación lógica de a (not)

Al momento de realizar operaciones combinadas para ejecutar múltiples acciones, la jerarquía de operadores debe ser tomada en cuenta para saber cuáles se realizarán primero. En la tabla 4.6 se muestra una lista de las operaciones con su respectiva ponderación en la jerarquía, donde 1 indica el nivel más alto.

Tabla 4.6 Jerarquía de las operaciones en Java

Operación	Jerarquía
Operadores unarios sufijos.	1
Operadores unarios.	2
Declaración de variable.	3
Multiplicadores.	4
Suma o resta.	5
Desplazamiento de bits.	6
Operadores relacionales.	7
Igualdad.	8
Operación de bits AND.	9
Operador de bits XOR.	10
Operador de bits OR.	11

Operador AND lógico.	12
Operador OR lógico.	13
Operadores condicionales.	14
Asignación.	15

2.3 Clases y objetos en Java

A una clase se la puede considerar como una unidad compuesta por atributos y métodos. Los atributos son las características propias de la clase y están representadas por las variables de la misma. Por otro lado, los métodos son las funciones que puede cumplir, representadas evidentemente por el código dentro de la clase. Cabe mencionar que existe un tipo especial de método conocido como constructor y que se crea automáticamente cuando se crea un objeto y que además lleva su mismo nombre.

La unicidad del objeto determina una de las tres principales características de los objetos, conocida como encapsulamiento. La segunda característica es conocida como herencia y está dada por la capacidad de que una clase pueda transferir sus funciones a otra. La sintaxis de la herencia indica que se debe poner el nombre de la clase hija seguida por la palabra *extends* y el nombre de la clase de la cual hereda las funciones. Finalmente, la tercera característica es el polimorfismo que le permite a una clase cumplir con múltiples funciones según los parámetros que se le envíen y necesidades de la ejecución. Tanto el encapsulamiento, cuanto la herencia y el polimorfismo no son características exclusivas de Java sino que deben estar presentes en cualquier lenguaje orientado a objetos.

Es muy común que varias clases cumplan funciones similares entre sí o que se complementen mutuamente para realizar una función en específica. Por ello, es posible agrupar a las clases en paquetes donde se los puede administrar bajo un nombre común. Dependiendo del entorno de desarrollo y de las librerías se tiene una gran diversidad de paquetes dentro de Java. Para incluir a un determinado paquete dentro de nuestro código de programación simplemente debemos utilizar la palabra *package* seguida su nombre.

2.3.1 Atributos de las clases en Java

Al momento de definir una clase se pueden utilizar cuatro modificadores que permiten especificar su tipo: *Abstract, Final, Public* y *Synchronizable*, cuyas características se detallan en la tabla 4.7.

Tabla 4.7 Modificadores de clases en Java

Tipo	Descripción
Abstract	Indica que la clase tiene al menos un método abstracto. No se puede instanciar un objeto de este tipo sino únicamente implementar los métodos que tiene.
Final	Esta clase no puede ser heredada por ninguna otra.
Public	Esta clase puede ser accedida por otras pertenecientes al mismo paquete o importadas.
Synchronizable	Indica que esta clase puede ser accedida por un solo hilo de ejecución a la vez.

De igual manera, existen modificadores para las variables miembro dentro de las clases que se muestran en la tabla 4.8. Tales modificadores básicamente definen el nivel de acceso a la información que tienen otras clases.

Tabla 4.8 Modificadores de variables miembro de clases en Java

Tipo	Descripción
Public	Su valor puede ser accedido desde fuera de la clase a la que pertenece
Protected	Únicamente las subclases pueden acceder a su valor.
Private	Su valor puede ser accedido únicamente por la clase a la que pertenece.
Friendly	Su valor puede ser accedido solo por las clases pertenecientes al mismo paquete. Este es el modificador usado por defecto.

2.4 Estructuras de control y datos

No se adentrará mucho en este aspecto debido a que las estructuras de control y datos son bastante similares a las que podemos encontrar en lenguaje C o C++. Para el caso de las estructuras de control, todas las mencionadas en la tabla 4.9 son equivalentes a las existentes en el lenguaje C++.

Tabla 4.9 Estructuras de control existentes en Java y C++

Nombre	Descripción
If-Else	Estructura de si-caso contrario
Switch	Estructura de comparativa
For	Estructura de bucle
While	Estructura de bucle condicional
Do-While	Estructura de bucle condicional

La única estructura que difiere de las que podemos encontrar en el lenguaje C++ se conoce como *Try-Catch* y nos permite tomar acciones específicas en caso de que se produzca un error en la ejecución del código. La sintaxis de la estructura *Try-Catch* se muestra en la figura 4.1, donde podemos analizar que se intentará ejecutar las acciones que están en la sección *Try,* pero si ocurre un error las sentencias debajo de *catch* son las que serán ejecutadas. No todo tipo de errores pueden ser manejados por Java y generalmente esta estructura es utilizada al momento de interactuar con el usuario.

```
try {
   // Código que pudiese generar Errores
    }catch(“Excepción de Java”) {
      // Código de manejo de la "excepción"
    }
```

Figura 4.1. Sintaxis de la estructura Try-Catch en Java

En lo referente a las estructuras de datos se puede mencionar que en Java se manejan *Strings* como cadenas de caracteres y arreglos o *arrays* que cumplen funciones similares a matrices y que generalmente son usadas para operaciones matemáticas complejas. Además, es indispensable aclarar que el puntero *Thi*s es una estructura de datos que siempre apunta a la clase actual donde se realiza la

ejecución. El puntero *This* es utilizado comúnmente para el manejo de la información cuando existen variables públicas en diferentes clases.

3. J2ME

En el primer capítulo se dio una introducción sobre J2ME como una versión portable de Java, totalmente enfocada al desarrollo de aplicaciones para aplicaciones móviles. Por tanto, esta sección estará totalmente enfocada al análisis de la configuración CLDC, debido a que es la más utilizada para los dispositivos móviles existentes en el mercado.

J2ME se sustenta en dos entornos de desarrollo, bajo la configuración CLDC, por un lado se encuentra la base que hereda clases de J2SE y por otro MIDP que añade clases para manejar la interface con el usuario. En la figura 1.9 se puede observar como MIDP está constituida sobre CLDC en la pila que constituye la configuración para dispositivos con capacidades limitadas.

3.1 CDLC

Establece el conjunto de clases esenciales, con funciones específicas, que permiten desarrollar aplicaciones para dispositivos con capacidades limitadas. Las características mínimas que debe tener un dispositivo móvil para ser compatible con CLDC son:

- 160 [KB] de memoria disponible. Donde 128 [KB] son para la KVM y 32 [KB] de memoria volátil para la ejecución de aplicaciones.
- Procesador de 16 bits.
- Conexión a cualquier tipo de red.

Por otro lado, las limitaciones que se tienen en esta configuración están íntimamente relacionadas con la baja memoria y capacidad de procesamiento de los dispositivos. Así, las restricciones más importantes son las siguientes:

- No es posible realizar operaciones matemáticas con punto flotante.
- No existe el método *Object.finalize* que permite liberar la memoria.

- El manejo de excepciones y errores se ve limitado según:
    - Sólo las capacidades nativas proporcionadas por CLDC son accesibles.
    - Sólo las APIs definidas dentro de CLDC están disponibles.

Sin embargo, un fabricante puede definir el manejo excepciones y errores adicionales si el dispositivo puede hacerlo.

3.1.1 El API de CLDC

El API no es más que la base sobre la que se construye la configuración CLDC. En esta se encuentran las clases básicas que permiten el desarrollo de las aplicaciones. Como caso particular cabe mencionar que el API de CLDC contiene una serie de interfaces propias dedicadas al desarrollo para servicios de red. A continuación se presentan los paquetes más relevantes dentro del API, con sus respectivas clases e interfaces.

- **Java.lang**

Este paquete está íntimamente relacionado con los fundamentos del lenguaje Java. Por ello define el soporte para las capacidades del lenguaje como la definición de los tipos primitivos de variables, cadenas, excepciones, entre otros. Las clases e interfaces que se encuentran dentro del paquete se muestran en la tabla 4.10.

Tabla 4.10. Clases e interfaces definidas dentro de java.lang

Nombre	Definición
Boolean	Encapsulamiento del tipo de variable tipo *boolean*.
Byte	Encapsulamiento del tipo de variable tipo *byte*.
Character	Encapsulamiento del tipo de variable tipo *char*.
Integer	Encapsulamiento del tipo de variable tipo *int*.
Short	Encapsulamiento del tipo de variable tipo *short*.
Long	Encapsulamiento del tipo de variable tipo *long*.
Class	Permite obtener información sobre la ejecución de una clase.
Math	Incluye el acceso a varias operaciones y constantes matemáticas.
Runtime	Otorga acceso al entorno de ejecución.
String	Representa una cadena de texto constante.
StringBuffer	Representa una cadena de texto, de longitud y valor variable.

System	Proporciona acceso a los recursos del sistema.
Thread	Crea un hilo de ejecución paralelo al principal dentro del mismo programa.
Throwable	Proporciona soporte para el control de excepciones.
Object	Es la superclase del resto de clases en Java.

- **Java.util**

Este paquete incluye interfaces y clases que permiten manejar diversos tipos de información que van desde fechas hasta estructuras de datos. En la tabla 4.11 se muestran las interfaces que se incluyen dentro del paquete

Tabla 4.11. Clases e interfaces definidas dentro de java.util

Nombre	Descripción
Calendar	Incluye funciones de manejo de fechas y funciones que permiten convertir valores numéricos en fechas.
Date	Representa un instante de tiempo.
Enumeration	Es una interface que describe cómo manejar la iteración entre un grupo de valores.
Hashtable	Permite asociar valores con claves bajo una equivalencia predefinida.
Random	Genera números pseudoaleatorios
Stack	Una colección con gestión LIFO (*Last Intro first output)*
TimeZone	Permite representar la zona horaria
Vector	Una colección en forma de matriz dinámica

- **Java.io**

Este paquete proporciona clases e interfaces para la escritura y lectura de datos. Existen varios tipos de clases e interfaces definidos en este paquete, por ello sólo las más utilizadas se incluyen en la tabla 4.12.

Tabla 4.12 Clases e interfaces definidas dentro de java.io

Nombre	Descripción
DataInput	Interfaz que define los métodos para leer datos desde un flujo binario a tipos primitivos
DataInputStream	Es un flujo desde el cual se leen datos como tipos primitivos
DataOutput	Es una interfaz que define métodos para escribir datos en forma de tipos primitivos en un flujo binario

DataOutputStream	Es un flujo que escribe datos en tipos primitivos en un flujo en formato binario
InputStream	Es la clase base para todos los flujos de entrada de información
InputStreamReader	Es un flujo desde el que se pueden leer caracteres de texto
OutputStream	Es la clase base para todos los flujos de salida de información
OutputStreamWriter	Es un flujo en el que se pueden escribir caracteres de texto
PrintStream	Es un flujo de escritura que facilita el “envío” de datos en forma de tipos primitivos

Debido a la gran cantidad de dispositivos CLDC existentes en el mercado, se dificulta el proveer soporte para funciones de red a nivel de configuración. Por ello existe *Generic Connection Framework* (GCF) encargado de manejar las características de conectividad que CLDC no.

3.2 El API de GCF

Es una plataforma de trabajo compuesta por una serie de interfaces de conexión junto con una clase denominada como *Conector* que permiten establecer diferentes tipos de conexiones.

El único paquete dentro del API es *javax.microedition.io* que incluye todas las funciones de conectividad necesarias. Las interfaces que se manejan son las que se muestran en la tabla 4.13.

Tabla 4.13 Interfaces de conectividad definidas en GFC

Interface	Descripción
Connection	Establece una conexión básica que únicamente puede ser abierta o cerrada.
ContentConnection	Es un flujo de conexión que proporciona acceso a datos web.
DatagramConnection	Es una conexión para manejar comunicaciones orientadas a paquetes.
InputConnection	Es una conexión de entrada para las comunicaciones del dispositivo.
OutputConnection	Es una conexión de salida para las comunicaciones del dispositivo
StreamConnection	Es una conexión bidireccional para las comunicaciones

	del dispositivo
StreamConnectionNotifier	Es una notificación que espera a que se establezca una conexión.

3.3 API de MIDP

Como se mencionó en el primer capítulo, los perfiles permiten definir estructuras de comportamiento similares para dispositivos móviles. Por tanto, permiten el desarrollo de tareas para un determinado dispositivo. Para el entorno de desarrollo de MIDP tenemos que se tienen clases heredadas directamente de J2SE y otras que son únicas dentro de J2ME. En la tabla 4.14 se muestran las clases que se heredan directamente de J2SE.

Tabla 4.14 Clases en MIDP heredadas de J2SE.

Clase	Descripción	Paquete en que se incluyen
Timer	Proporciona funcionalidad para crear tareas programadas temporalmente.	Java.util
TimerTask	Representa una tarea que es temporizada a través de la clase Timer	Java.util

Por otro lado tenemos las interfaces propias que son diseñadas específicamente para la programación de MIDlets o aplicaciones para dispositivos móviles. Es así que existen cuatro paquetes exclusivos que contienen todas las interfaces y clases.

3.3.1 Paquete javax.microedition.midlet

Es el paquete fundamental dentro de J2ME y contiene únicamente a la clase MIDlet de la cual heredarán todas las aplicaciones creadas. MIDlet proporciona la funcionalidad básica para que una aplicación se pueda ejecutar dentro de un dispositivo móvil.

3.3.2 Paquete javax.microedition.lcdui

Este paquete contiene clases e interfaces que incluyen componentes de interfaz de usuario específicos para las pantallas de los dispositivos móviles. La importancia de esta clase radica en que es la encargada de crear un nexo con el usuario para el manejo del comportamiento de la aplicación mediante la pantalla. En la tabla 4.15 se muestran las interfaces incluidas dentro del paquete.

Tabla 4.15 Interfaces incluidas dentro del paquete javax.microedition.lcdui

Interface	Descripción
Choice	Describe una serie de elementos sobre los que el usuario debe escoger.
CommandListener	Provee una interface de monitorización de eventos (*listener)* para gestionar la ejecución de un comando.
ItemStateListener	Provee una interface de monitorización de eventos (*listener)* para gestionar los eventos sobre los elementos.

Por otro lado, el paquete incluye varias clases para mostrar información en pantalla. Las más importantes se muestran en la tabla 4.16 con su respectiva descripción.

Tabla 4.16 Clases incluidas dentro del paquete javax.microedition.lcdui

Clase	Descripción
Canvas	Permite realizar gráficas en pantalla a bajo nivel.
Command	Representa un comando a alto nivel que se ejecuta en un MIDlet.
Display	Representa la pantalla del dispositivo y maneja las interacciones con el usuario.
Font	Permite definir las características de texto en pantalla.
Graphics	Permite realizar operaciones gráficas bidimensionales como dibujo de líneas, elipses, texto e imágenes.
Item	Es un componente que asocia un elemento con una etiqueta.
Screen	Presenta una pantalla a alto nivel que puede ser usada para indexar elementos adicionales que serán mostrados al usuario.

Cabe mencionar que sobre el control de pantalla a bajo y alto nivel se profundizará más adelante en la sección dedicada totalmente a las estructuras de los MIDlet.

3.3.3 Paquete javax.microedition.io

Maneja las operaciones de entrada y salida de datos que no son controladas por el API de CDLC. Además añade la interface HTTP mediante *HttpConnection*.

3.3.4 Paquete javax.microedition.rms

Agrega un sistema de persistencia basado en registros para almacenar información conocido como *Record Management System* (RMS), el cual se analizará más adelante. Las interfaces más importantes dentro de este paquete se muestran en la tabla 4.17 mientras que las clases son mostradas en la tabla 4.18.

Tabla 4.17 Interfaces incluidas dentro del paquete javax.microedition.rms

Interface	Descripción
RecordComparator	Permite comparar dos registros
RecordEnumeration	Permite realizar iteraciones sobre los registros.
RecordFilter	Permite realizar el filtraje de un registro
RecordListener	Permite monitorear cambios dentro de los registros

Tabla 4.18 Clases incluidas dentro del paquete javax.microedition.rms

Clase	Descripción
InvalidRecordException	Se ejecuta cuando una operación falla porque el indicador del registro no es válido
RecordStore	Equivale a un almacenador de registros
RecordStoreException	Se ejecuta cuando se produce un error general.
RecordStoreFullEsception	Se ejecuta cuando una operación de guardado en el registro falla porque el mismo está lleno.
RecordStoreNotFoundException	Se ejecuta cuando existe un error debido a que no se ha podido encontrar el registro indicado.

4. MIDlet

Un MIDlet es básicamente una aplicación Java autónoma que se ejecuta en un entorno limitado y que posee un conjunto definido de librerías heredadas de una clase fundamental que son utilizadas para su compilación. Las características más relevantes de un MIDlet son las siguientes:

- Debe ejecutarse sobre un dispositivo específico.

- No tiene método main() sino mas bien tres estados: ejecución, pausa y finalizado.
- El nombre de la clase principal tiene que ser idéntico al nombre del archivo que lo contiene.
- Debe ser heredera de la clase *javax.microedition.midlet.MIDlet.*.* e implementar los siguientes métodos específicos:
    - o startApp() relacionado con la ejecución
    - o pauseApp() relacionado con la pausa de la aplicación
    - o destroyApp() relacionado con la finalización de la ejecución.

Aún cuando alguno de ellos no tenga código o no vaya a ser utilizado.

4.1 Estructura de un MIDlet

Es importante recalcar que la estructura básica de un MIDlet siempre es la misma y está compuesta por al menos los métodos *startApp(), pauseApp()* y *destroyApp()*. Estos métodos no necesitan estar en un orden específico puesto que la programación orientada a objetos hace que la ejecución se de cuando son llamadas y no de manera secuencial. La figura 4.2 muestra la sintaxis más comúnmente utilizada para estos tres métodos, donde cabe recalcar que el método *startApp()* se ejecutará inmediatamente inicie el MIDlet mientras que *destroyApp()* lo hará cuando termine la ejecución.

```
a) public void startApp() throws MIDletStateChangeException {}
b) public void pauseApp() {}
c) public void destroyApp (boolean unconditional) {}
```

Figura 4.2 Sintaxis de los métodos fundamentales de un MIDlet

Además, un MIDlet debe incluir en el comienzo de su código los paquetes necesarios para ser ejecutado, esto es algo similar a las cabeceras que se incluyen en un programa en el lenguaje C++. La palabra clave utilizada para incluir los paquetes es *import* y la sintaxis indica que se debe poner a continuación el nombre del paquete. Por ejemplo, para incluir el paquete MIDlet la sintaxis sería *import*

javax.microedition.midlet.;* donde el asterisco indica que se deben importar todas las librerías dentro del paquete. La figura 4.3 muestra un esquema de la estructura básica de un MIDlet.

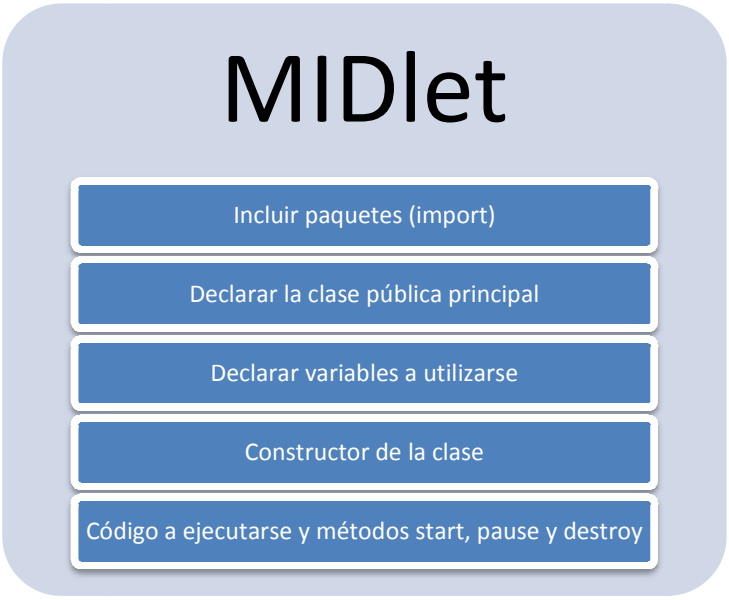


Figura 4.3 Estructura básica de un MIDlet

Es necesario mencionar que el constructor es el que permite que la clase sea creada por primera vez y tome una parte de memoria para su ejecución. El constructor tiene como características que lleva el mismo nombre de la clase y no tiene tipo de retorno, ni siquiera *void*.

Cuando un MIDlet es compilado genera dos archivos con extensiones .JAR y .JAD, respectivamente. Los archivos .JAR son paquetes comprimidos que incluyen las clases (.class), imágenes, sonidos y todo lo necesario para que se ejecute la aplicación. Además incluyen un archivo conocido como *manifest* (.mf) que contiene toda la información sobre las clases contenidas en el archivo .JAR. Por otro lado, los archivos .JAD contienen la información que permite instalar correctamente la aplicación en el dispositivo y también aquella relacionada con la versión, autor y recursos utilizados por la misma.

El mayor desafío al momento de crear un MIDlet se encuentra en encontrar un dispositivo con los recursos adecuados para ejecutarlo. Esto debido a que las librerías y características de los dispositivos móviles varían según el fabricante y en muchos casos la KVM tiene funcionalidades limitadas por las propias características propias del dispositivo. La compilación de un programa en una PC no garantiza que este se vaya a ejecutar correctamente en el dispositivo móvil; de hecho, la ejecución en un emulador tampoco lo hace.

4.2 Entorno gráfico

Como se mencionó anteriormente, el paquete *javax.microedition.lcdui* nos ofrece herramientas para crear interfaces que nos permitirán interactuar con el usuario. Es así que podemos tener dos tipos de entorno gráfico denominados como *screen* (pantalla) y *canvas* (lienzo) a través de la clase *Displayable*, acorde a lo que se muestra en la figura 4.4.

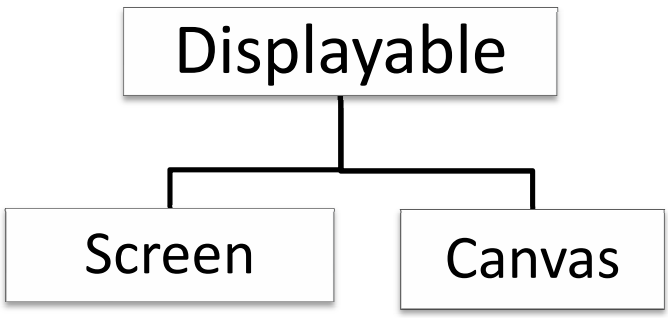


Figura 4.4. Entornos gráficos derivados de la clase *Displayable*

El entorno *screen* nos permite incluir elementos de control en la interfaz, como menús, botones, textos, entre otros para complementar la ejecución del programa con la interacción del usuario. Por otro lado, *canvas* nos permite manipular la pantalla a un nivel más bajo, pudiendo inclusive definir las propiedades de los pixeles que se van a mostrar. Es por ello que el entorno *canvas* es el preferido al momento de crear juegos para dispositivos móviles, mientras que la sencillez de la interfaz *screen* la convierte en predilecta al momento de crear aplicaciones de gestión de datos.

Sea cual sea el entorno gráfico, al ejecutarse un MIDlet se crea un objeto *display* encargado de mostrar y manejar la información en pantalla. Por tanto, es necesario hacer referencia a este objeto para tener control sobre la información que se desea publicar, mediante el método *getDisplay()*. La figura 4.5 muestra un ejemplo de una variable denominada *display* que apunta al objeto *display* actual mediante el puntero *this*, que como mencionamos anteriormente permite conocer a ubicación actual de la ejecución.

```
display = Display.getDisplay(this);
```

Figura 4.5. Ejemplo de aplicación del objeto *display*

Para la aplicación de mensajería no se necesita tener un control de bajo nivel sobre la pantalla a utilizarse en el dispositivo móvil. Por tanto, resulta más útil y simple el estudiar las características de formularios que se pueden crear para tener interactividad con el usuario. La figura 4.6 muestra los elementos gráficos más comunes, existentes para el entorno gráfico *screen.*

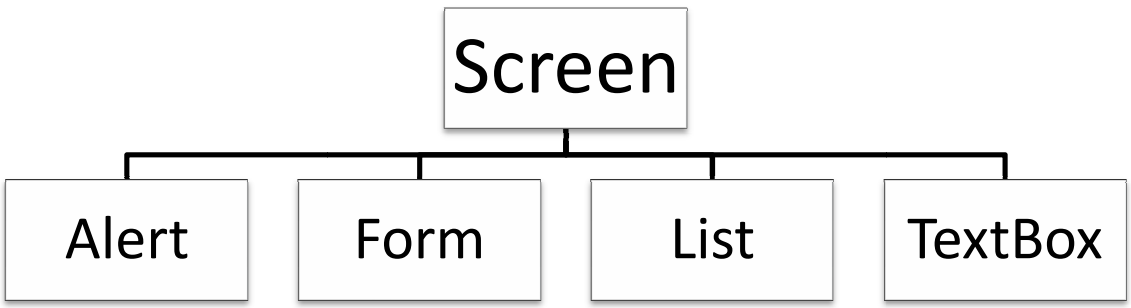


Figura 4.6. Elementos para el diseño de formularios incluidos en *Screen*

Es necesario mencionar que por las características de J2ME, y de los dispositivos móviles, no es posible mostrar más de un elemento gráfico a la vez; por tanto, el elemento a utilizarse ocupará toda la pantalla. Es entonces que nace la necesidad de incluir botones de navegación que permitirán que el usuario navegue

entre diferentes formularios o pantallas. El método de la clase *display* que permite definir el elemento en pantalla es *setCurrent()*, donde se incluye el nombre del elemento dentro de los paréntesis.

4.2.1 Clase *Alert*

Como su nombre lo indica, permite enviar mensajes de alerta o notificaciones al usuario. Este elemento permanece en pantalla hasta que se produzca la ejecución de un comando del tipo OK. La figura 4.7 muestra un ejemplo de mensaje de alerta

Figura 4.7. Ejemplo en pantalla de *Alert*

La figura 4.8 muestra la sintaxis de este elemento, de donde únicamente los campos de título, texto de alerta y tipo de alerta son necesarios. La imagen es un campo opcional que puede ser incluido para guiar al usuario. Los diferentes tipos de alerta que se pueden tener son *Alarm, Confirmation, Error, Info* y *Warning* y se los puede usar dependiendo del mensaje que queramos mostrar al usuario. En ciertos dispositivos los tipos de error suelen estar relacionados con diferentes tonos de alerta, pero esto no es una regla para todos los dispositivos.

```
Alert (String título, String texto_alerta, Image imagen_alerta, AlertType tipo_alerta)
```

Figura 4.8. Sintaxis *Alert*

La figura 4.9 muestra un ejemplo de código en el cual se declara una alerta denominada error que será mostrada en pantalla mediante el método *setCurrent* hasta que el usuario responda con una acción de comando del tipo OK.

```
Alert error = new Alert ("Error","El dato no es válido", null, AlertType.ERROR);
display.setCurrent(error);
```

Figura 4.9 Ejemplo de aplicación de *Alert*

4.2.2 Clase *List*

Como su nombre lo indica, esta clase permite manejar listas de elementos, cuyos valores son predefinidos desde la programación. La figura 4.10 muestra un ejemplo de este tipo de elemento mientras que en la tabla 4.19 se muestran los diferentes tipos de listas que se pueden definir.

Tabla 4.19. Tipos de listas que se pueden definir en *List*

Tipo	Descripción
Exclusive	Listas en las cuales sólo puede ser seleccionado un elemento a la vez.
Implicit	Listas en las cuales se selecciona el elemento que tiene el foco por defecto
Multiple	Listas que permite hacer una múltiple selección de elementos

La figura 4.11 muestra la sintaxis utilizada para este tipo de elementos, donde se puede observar que se necesita de un arreglo de cadenas de caracteres para ser incluidos en la lista. Según indica la sintaxis es posible incluir gráficos dentro de la lista de ser necesario, pero de no hacerlo se debe incluir el valor *null* dentro de la sintaxis. Además. cabe mencionar que las listas del tipo *Exclusive* e *Implicit* tienen el método *getSelectedIndex()* que retorna el índice del elemento que ha sido seleccionado.

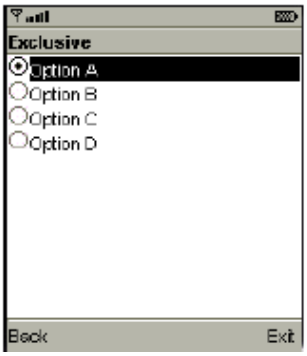


Figura 4.10 Ejemplo en pantalla de *List*

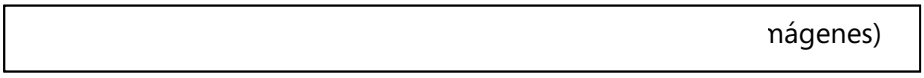


Figura 4.11 Sintaxis de *List*

4.2.3 Clase Form

Esta es tal vez la clase más utilizada en la programación de MIDlets utilizando java, en gran parte debido a la versatilidad de sus aplicaciones. La clase *Form* permite crear pantallas que pueden tener diversos tipos de objetos, como cuadros de texto, imágenes, entre otros. Si hiciésemos una analogía, *Form* nos permite crear el equivalente a ventanas en Windows.

La figura 4.12 muestra un diagrama de la estratificación de los elementos que podemos tener dentro de un formulario o *Form*, donde se observa que todos los elementos son derivados de la clase *Item*. Por su lado, la clase *Item* es un elemento visual que no ocupa toda la pantalla, sino que formará parte de la interfaz de usuario junto con otros elementos, añadidos dentro del formulario.

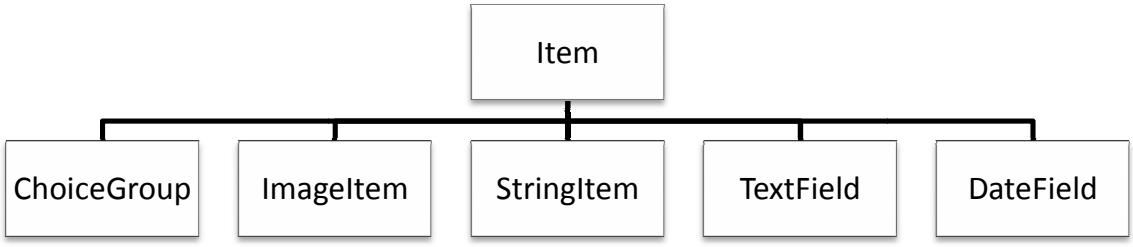


Figura 4.12 Elementos que son controlados por la clase *Item*

Además, *Item* cumple la función de manejar el orden de los elementos que son incluidos dentro del formulario. Dicho control lo lleva mediante la asignación de índices a cada elemento que es insertado mediante el método *append().* La figura 4.13 muestra la sintaxis del método, donde no se indica el índice que se requiere para el elemento puesto que el método lo asigna automáticamente desde el cero.

```
append ( item elemento)
```

Figura 4.13. Sintaxis de la clase *Item*

Otros métodos de la clase *Item* que pueden ser utilizados para el manejo de los elementos de un formulario se muestran en la tabla 4.20.

Tabla 4.20. Métodos incluidos en la clase *Item*

Método	Función
delete ()	Elimina un elemento del formulario
insert (índice,elemento)	Inserta un elemento en el índice indicado
set (índice, elemento)	Sustituimos un elemento por otro del índice indicado
size()	Permite conocer el número de elementos dentro del formulario
itemStateChanged(elemento)	Indica si se ha producido un cambio en el valor del elemento

En la figura 4.12 se muestran los diferentes elementos que pueden ser incluidos dentro del formulario. Cabe mencionar que la mayoría de ellos son similares a las clases revisadas anteriormente con la diferencia de que no ocuparán toda la pantalla sino que estarán indexados en diferentes partes de un formulario.

4.2.4 Clase *StringItem*

Esta clase cumple con la función de añadir texto a un formulario, de manera que puedan ser leídos pero no modificados por el usuario. La figura 4.13 muestra un ejemplo del uso de esta clase en pantalla.

Figura 4.13 Ejemplo en pantalla de *StringItem*

La sintaxis de esta clase es bastante simple, como se muestra en la figura 4.14, puesto que únicamente es necesario definir una etiqueta y el texto a mostrar. Si no se desea incluir la etiqueta simplemente se debe colocar “”.

```
StringItem(“Etiqueta”, “Texto a mostrar”
```

Figura 4.14. Sintaxis de *StringItem*

4.2.5 Clase *ImageItem*

Esta clase permite añadir imágenes o gráficos al formulario, que son incluidos dentro del paquete del MIDlet. Es importante mencionar que si no es posible mostrar la imagen durante la ejecución es posible mostrar un texto alternativo.

La figura 4.15 muestra la sintaxis utilizada para esta clase donde cabe recalcar al parámetro *layout* que define la posición donde se ubicará la imagen. Los cuatro valores más comunes que puede tomar se muestran en la tabla 4.21.

Tabla 4.21. Valores que puede tomar la clase *ImageItem*

Valor	Descripción
LAYOUT_DEFAULT	La ubicará a continuación del último elemento ingresado.
LAYOUT_LEFT	La ubicará justificada la izquierda.
LAYOUT_RIGHT	La ubicará justificada a la derecha.
LAYOUT_CENTER	La ubicará centrada en la pantalla.

```
ImageItem ("Etiqueta", Image img, layout, "texto_alternativo")
```

Figura 4.15. Sintaxis de *ImageItem*

Además, la sintaxis indica que se debe colocar la ruta donde se encuentra la imagen. Para ello se recomienda que la imagen se incluya en la carpeta *scr* o raíz donde se está trabajando en el código fuente. Dependiendo del compilador este proceso puede ser automático [59]

4.2.6 Clase *TextField*

Esta clase permite añadir cuadros de texto dentro de un formulario. Su funcionamiento es similar a la de un cuadro *TextBox* que es utilizado en otros lenguajes de programación. La figura 4.16 muestra un ejemplo de uso en pantalla de esta clase.

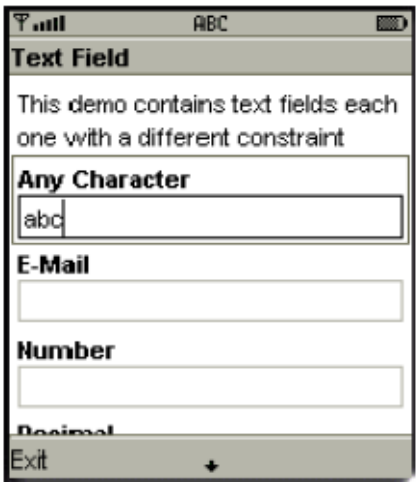


Figura 4.16. Ejemplo en pantalla de *TextField*

La figura 4.17 muestra la sintaxis utilizada para este elemento. El parámetro limitante puede tomar cualquiera de los valores mostrados en la tabla 4.22

Tabla 4.22 Valor del campo limitante en *TextField*

Valor	Descripción
ANY	No existe restricción
EMAILADDR	Permite el ingreso de una dirección de e-mail.
NUMERIC	Sólo caracteres numéricos
PASSWORD	No mostrará los caracteres que sean ingresados
URL	Sólo direcciones URL.

```
TextField (“Etiqueta”, “Texto”, int tamaño_max, limitante)
```

Figura 4.17 Sintaxis de *TextField*

4.2.7 Clase *ChoiceGroup*

Esta clase permite mostrar una lista de la cual se puede seleccionar uno o varios elementos. Su comportamiento es similar a la de la clase *List*, por lo que su sintaxis es bastante similar como se muestra en la figura 4.18.

```
ChoiceGroup (“Etiqueta”, int tipo_lista, String[] elementos, image[] imágenes)
```

Figura 4.18. Sintaxis de *ChoiceGroup*

La lista puede incluir sólo opciones en texto o acompañadas de imágenes, al igual que para la clase *List*. Además, los tipos de lista son los mismos que para la clase *List*. La figura 4.19 muestra un ejemplo de uso de *Choice Group* acompañado de otros elementos dentro de un formulario.

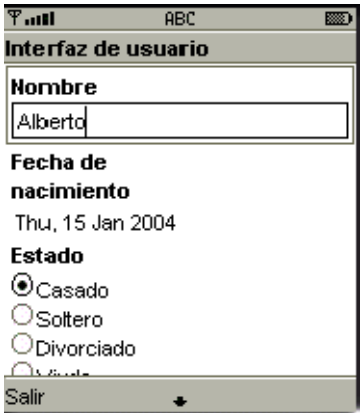


Figura 4.19 Ejemplo en pantalla de *ChoiceGroup*

Finalmente, es importante mencionar que existen otros dos tipos de elementos que también pueden ser utilizados dentro de un formulario:

- *DateField.-* Permite añadir un calendario mediante el cual se facilita la selección de fecha. La figura 4.20 muestra un ejemplo del uso de esta clase.

Figura 4.20a Ejemplo en pantalla de *DateField*

- *Gauge.-* Permite añadir una barra de progreso en la pantalla que puede ser utilizada para mostrar el avance que se da durante la carga de una aplicación, procesamientos, entre otros. La figura muestra la aplicación de esta clase.

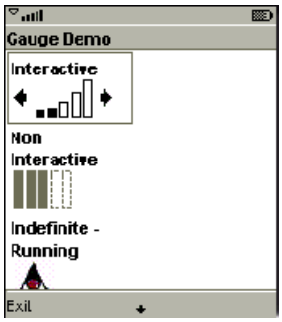


Figura 4.20b Ejemplo en pantalla de *Gauge*

Además del diseño de pantallas para establecer una interface con el usuario, es necesario permitir la navegación entre los diferentes formularios mediante controles de navegación. Tales controles son conocidos como comandos.

4.2.8 Comandos

Es un elemento que permite interactuar con el usuario de manera que pueda simular acciones. Están representados por botones de acción y usualmente aparecen en la parte inferior de la pantalla del dispositivo móvil. Todo comando proviene de la clase *Command()*. La figura 4.21 muestra la sintaxis de este tipo de elemento.

```
Command("Nombre",Tipo,Prioridad)
```

Figura 4.21 Sintaxis de *Command*

El parámetro de prioridad dentro de la sintaxis indica la jerarquía de un comando sobre otro. Si no se especifica el comando EXIT tiene la más alta prioridad.

Tabla 4.23 Tipos de comandos existentes en J2ME

Comando (Tipo)	Acción
Command.OK	Confirma una selección
Command.CANCEL	Cancela la acción actual
Command.BACK	Regresa al formulario anterior
Command.STOP	Detiene una operación
Command.HELP	Muestra el archivo de ayuda definido en el MIDlet
Command.EXIT	Salir de la aplicación.

Para añadir un comando en pantalla es necesario utilizar la palabra clave *addCommand(“Comando”)*, especificando al comando definido previamente. Además, es necesario establecer que se esté constantemente verificando el cambio del valor del comando mediante el comando *setCommandListener()*. Ahora bien, cuando un usuario ha hecho uso de un comando se ejecuta el método *commandAction()* que devuelve el comando ejecutado y la pantalla de donde se ejecutó el comando. La figura 4.22 muestra un ejemplo de aplicación de *commandAction().*

```
Public void commandAction(Command c, Displayable s) {
if (c == salir) {
destroyApp();
notifyDestroyed();
}
}
```

Figura 4.22. Ejemplo de aplicación de *commandAction()*

De la figura 4.22 se puede observar que el comando definido se llama salir y que lo que permite es cerrar la aplicación. Además, los métodos *destroyApp()* y *notifyDestroyed*() permiten detener la ejecución del MIDlet y notificar al dispositivo respectivamente.

5. JABWT

JABWT nace bajo la necesidad de que el estándar Bluetooth no define herramientas para el desarrollo de aplicaciones que utilicen la pila de protocolos. Por ello, el objetivo principal es definir un API estándar que permita crear aplicaciones en entornos abiertos sobre las tecnologías Java y Bluetooth. Cabe mencionar que esta tecnología está basada en la especificación de la versión 1.1 de Bluetooth lo que le permite ser compatible con las especificaciones v2 y v3 pero no lo será con aquella de bajo consumo de energía a menos que se la utilice en modo extendido.

Desde un principio JAWBT fue concebido para brindar los beneficios de ambas tecnologías, Bluetooth y Java, de manera simple para que la mayor cantidad de programadores la adoptara al momento de desarrollar aplicaciones. Es así, que únicamente se manejan un total de 21 clases que permiten manejar procesos complicados a bajo nivel de manera transparente. Por otro lado, en un principio se pensó en que se debían crear perfiles acorde a los definidos por el Bluetooth SIG según las aplicaciones; sin embargo, se consideró que sería imposible ir a la par con la aparición de nuevos perfiles en Bluetooth, por lo que se concluyó que simplemente los perfiles de JAWBT estarían constituidos sobre OBEX, RFCOMM y L2CAP. De esta manera, es posible acceder a tales protocolos directamente mediante el lenguaje Java.

Las aplicaciones que se le puede dar a esta tecnología van más allá de la programación en dispositivos móviles gracias a la recomendación JSR-197 [60], que permite la migración de aplicaciones en J2ME hacia J2SE. Por tanto, es posible crear aplicaciones que corran tanto celulares, agendas personales, televisores hasta dispositivos con mayor capacidad como computadores personales. Simplemente es necesario que incluyan ciertas características básicas para soportar la tecnología.

5.1 Características en Hardware de JABWT

Como se mencionó anteriormente, definir un API que pudiese ser un compendio de todas las características de los dispositivos compatibles con J2ME y Bluetooth era todo un reto debido a la diversidad de equipos que incluyen ambas tecnologías. Por ello, la solución más eficiente fue dar soporte a la base de la tecnología J2ME mediante los protocolos fundamentales.

Es importante considerar que para que un dispositivo pueda incluir JABWT debe al menos ser compatible con J2ME y tener conectividad Bluetooth. Por ello, resulta evidente que los requerimientos en hardware serán similares a aquellos para los dispositivos CDLC; sin embargo, existen características adicionales que deben

ser implementadas en los dispositivos para el manejo y gestión de las conexiones. Por ello, la especificación de JABWT define las siguientes características:

- Únicamente las librerías CLDC son necesarias.
- Definición del API de OBEX debe ser independiente de la de Bluetooth, debido a que OBEX puede utilizar otros medios de transporte (IrDA, USB, TCP).
- Debe existir un centro de gestión de conexiones para evitar que conexiones concurrentes se afecten una a otra pero a que la vez no se interrumpa la comunicación. Tal centro será conocido como *Bluetooth Control Center* (BCC).
- Se debe permitir la comunicación *peer to peer*. Por tanto, debe ser posible que cualquiera de los dos miembros en una conexión Bluetooth sea maestro o esclavo.
- Debe existir la posibilidad de crear perfiles Bluetooth sobre los APIs definidos para los protocolos RFCOMM, L2CAP y OBEX.

La implementación de JABWT no se encuentra sobre toda la pila de protocolos definidos para Bluetooth, sino que únicamente toma los elementos más importantes que le permitan extender la aplicación del mismo. Por ello, tal implementación resulta menos flexible debido a que esperaría que la mayoría de partes de la pila se encuentre presente en el dispositivo, algo que no ocurre necesariamente para todos los dispositivos, como se mencionó anteriormente en el capítulo tres. Además, resulta necesario indicar que debido a que varias de las aplicaciones dedicadas a voz que utilizan canales Bluetooth son definidas en código nativo del dispositivo, se decidió no incluir soporte para ellos y para todos aquellos perfiles de telefonía o manejo de conexiones *dial up* (TCS Binary, BNEP).

La figura 4.23 muestra que la implementación de JABWT accede únicamente a secciones de la pila del protocolo Bluetooth. Los cuadros en azul representan los protocolos a los cuales una aplicación puede acceder directamente (L2CAP, RFCOMM, SDP, OBEX, LMP). Sin embargo, puede ser posible que JABWT no

tenga acceso a toda la funcionalidad de dicha capa; por ejemplo, se tendrá soporte para los enlaces orientados a conexión dentro de la capa L2CAP pero no existirá soporte para aquellos enlaces no orientados a conexión.

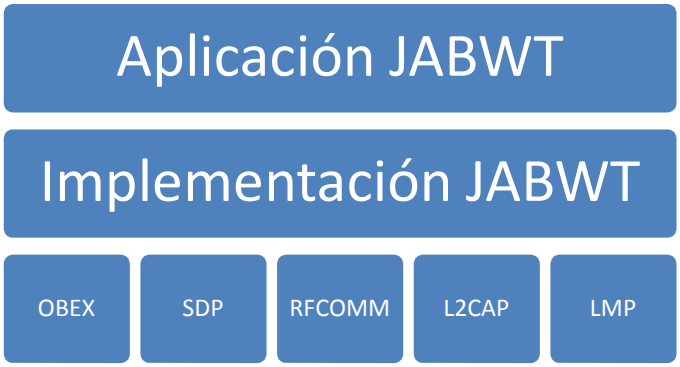


Figura 4.23 Implementación de JABWT sobre un dispositivo.

Por tanto, existen cuatro servicios que pueden ser implementados utilizando JABWT directamente. Están el descubrimiento de dispositivos y servicios, el registro de servicios, el establecimiento de conexiones RFCOMM, L2CAP y OBEX y finalmente el establecimiento de conexiones de manera segura utilizando cualquiera de los protocolos anteriormente mencionados.

Sin embargo, existen servicios que no pueden ser manejados directamente por el API definido para JABWT debido a que no fueron especificados dentro de sus objetivos como conexiones asíncronas, manejo de aplicaciones, entre otras que dependen de las implementaciones nativas para cada dispositivo.

5.2 Arquitectura y funcionalidad

JABWT fue definido en la recomendación JSR-82 y se determinó que únicamente depende de CLDC para operar. Sin embargo, casi siempre se lo define junto al perfil de MIDP en J2ME, debido a que los dispositivos móviles son los que utilizan más extensivamente sus servicios.

Su funcionalidad está fundamentada dentro de tres ejes. El primero es el de descubrimiento, donde se deben tomar a consideración los procesos de exploración de un dispositivo, sus servicios y el registro de los mismos. El segundo es el de comunicación, que incluye el establecimiento de conexiones entre diferentes aplicaciones y bajo diferentes protocolos (RFCOMM, L2CAP, OBEX). Finalmente existe el manejo de dispositivos, que permite el manejo local y remoto de los estados y propiedades de los dispositivos conocidos, así como manejar el aspecto de la seguridad que se mantiene con las conexiones establecidas.

Si bien no existe una arquitectura estándar definida para incluir a JABWT en un dispositivo móvil, existen varios parámetros que los fabricantes deben tomar en cuenta al momento de la implementación. El primero de ellos es que siempre el nivel más bajo estará conformado por el sistema operativo del dispositivo y la forma en que este implementa la pila de protocolos Bluetooth. Como ya se mencionó anteriormente, cada fabricante decide de qué manera, cómo y cuáles elementos de pila Bluetooth incluye en sus dispositivos por lo que también se pueden incluir aplicaciones, conocidas como nativas, que interactúan directamente con el sistema operativo, que además están escritas en su mismo lenguaje y que manejan operaciones de bajo nivel. Ubicado justamente sobre el sistema operativo debe encontrarse la KVM y CLDC para proveer todas las clases y elementos necesarios para la ejecución de aplicaciones escritas en java y así permitir que MIDP y JABWT puedan ser implementados en el sistema. Finalmente, una la aplicación corre utilizando los recursos tanto de MIDP como de JABWT y de CDLC directamente. La figura 4.24 muestra un ejemplo de cómo JABWT es incluido dentro de la arquitectura de dispositivo móvil Motorola [61], donde se observa que se ha tomado a consideración los lineamientos anteriormente mencionados.

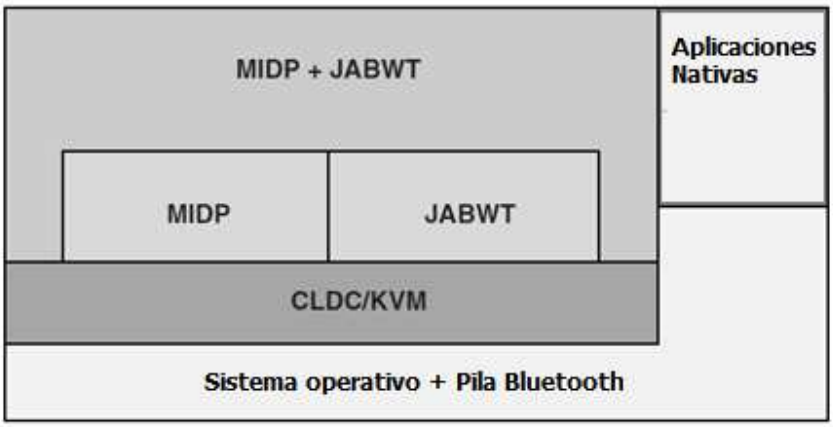


Figura 4.24. Implementación de JABWT en un dispositivo Motorola

JABWT se adapta totalmente al entorno de cliente-servidor típico de una conexión Bluetooth. Bajo este modelo, el servidor es aquel que presta asistencia u ofrece algún tipo de servicio que el cliente no posee localmente. Para que tal servicio pueda ser ofrecido a un cliente remotamente es necesario que sea definido y descrito en un registro de servicios y este a su vez sea incluido a la base de datos conocida como *service discovery database* (SDDB). En la figura 4.25 se muestra un diagrama del procedimiento de registro y búsqueda de servicios en un entorno de cliente-servidor.

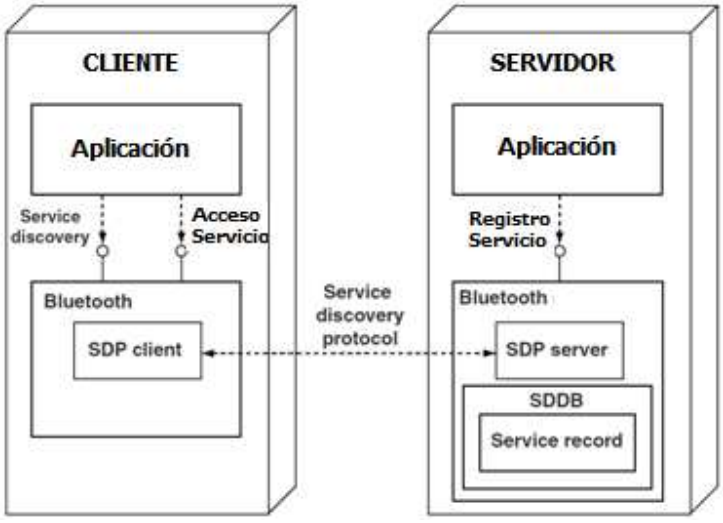


Figura 4.25 Diagrama de conexión cliente servidor en JABWT

El protocolo de descubrimiento de servicios (SDP) es aquel que permite a un dispositivo Bluetooth reconocer todas las prestaciones que puede obtener de otro cuando lo necesite. Como ya se mencionó anteriormente, el dispositivo servidor almacena un registro del servicio en la SDDB y al mismo tiempo permite que los dispositivos remotos sepan de los servicios que puede ofrecer. La información obtenida a través del protocolo de descubrimiento es suficiente para que un cliente decida si es necesario o no establecer una conexión con el servidor, que siempre espera una petición antes de establecer un enlace.

Dentro del entorno de cliente servidor dentro de JABWT se han definido responsabilidades específicas tanto para los dispositivos clientes cuanto para los servidores, las cuales son mostradas en la tabla 4.24. Esta división de responsabilidades facilita el proceso de comunicación y a la vez simplifica el proceso de descubrimiento de fallas en la misma.

Tabla 4.24 Responsabilidades definidas para cliente y servidor en una conexión Bluetooth

Responsabilidades Cliente	Responsabilidades Servidor
Utilizar el protocolo SDP para localizar servicios en dispositivos remotos.	Crear un registro del servicio que describa claramente el servicio que ofrece una determinada aplicación.
Alinearse con las características de seguridad solicitadas para poder acceder a los servicios.	Añadir el registro de servicio a la SDDB para que los potenciales clientes conozcan sobre ellos. Así como actualizar la información del mismo si existiesen cambios o eliminar el registro cuando el servicio ya no se encuentre disponible.
Iniciar una comunicación con el servidor a fin de obtener el recurso deseado.	Definir las medidas de seguridad asociadas con un servicio y que deben ser dadas a conocer a los clientes.

La implementación Bluetooth debe permitir establecer períodos de búsqueda continuos de servicios a fin de encontrar el deseado.	Aceptar las conexiones de los clientes que requieren del servicio.
	La implementación Bluetooth debe permitir el acceso a la SDDB y a los registros de la misma. Además debe mantener y permitir las conexiones con los dispositivos remotos.

Si bien el entorno de cliente-servidor es básico en la comunicación Bluetooth, es imposible pensar que este puede establecerse estrictamente en una aplicación de JABWT. La idea de ubicuidad se hace presente a través de aplicaciones que trabajen bajo entornos *peer-to-peer*, en los cuales los papeles de cliente-servidor puedan ser definidos dinámicamente durante la comunicación.

5.2.1 Bluetooth Control Center (BCC)

El centro de control Bluetooth es un requisito necesario que se encuentra definido en la especificación para la implementación de JABWT. BCC permite el manejo de varias conexiones simultáneas que pudiese tener un dispositivo. Sin embargo, no existe ningún API que permita tener acceso a BCC y más aún es posible que el ni siquiera BCC esté escrito en lenguaje java. Los detalles de implementación de este centro de control y su manejo han sido dejados a criterio de los fabricantes, por lo que comúnmente se encuentran escritos en código nativo del dispositivo y muchas veces pueden inclusive no tener interfaz para interactuar con el usuario. La figura 4.26 muestra la ubicación del BCC dentro de la arquitectura del dispositivo, de donde es importante indicar que BCC puede resolver conflictos entre aplicaciones de JABWT y otras nativas o escritas bajo otro lenguaje dentro del dispositivo.

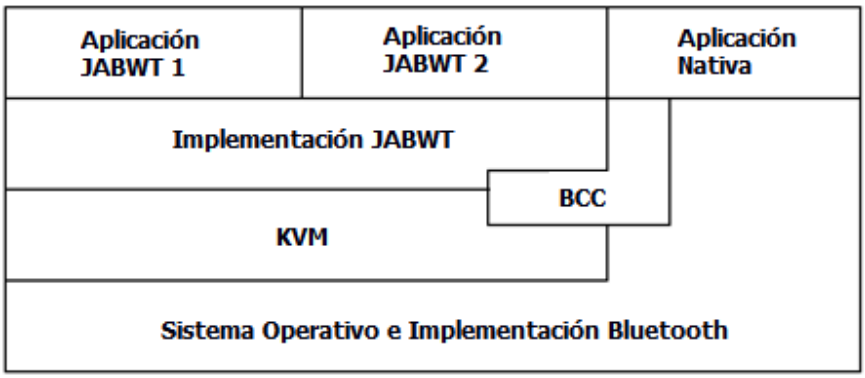


Figura 4.26. Diagrama de implementación de BCC en un dispositivo

El centro de control nace de la necesidad de permitir que varias aplicaciones utilicen el medio Bluetooth dentro de un mismo dispositivo sin afectarse mutuamente. Para esto, BCC cumple con las siguientes tareas:

- Resolver conflictos de peticiones entre aplicaciones.- Debido a que diversas aplicaciones pueden requerir de diferentes niveles de seguridad. Por ello el BCC es el encargado de manejar independientemente las conexiones sin permitir que una afecte a la otra.
- Permitir la modificación de permisos y propiedades de la pila Bluetooth del dispositivo.- Debido a que existen propiedades del sistema que no pueden ser modificados directamente por JABWT. Por ejemplo, parámetros como medidas de seguridad o registro de dispositivos conocidos.
- Manejar las operaciones de seguridad que requieran de interacción con el usuario.- Relacionado directamente con el establecimiento de una conexión Bluetooth mediante del ingreso de un valor de PIN, como se mencionó en el capítulo 3.

5.3. Interfaz de programación de aplicaciones (API)

Todas las operaciones involucradas con el descubrimiento, establecimiento y transferencia de información pueden ser manejadas a través de JABWT. En esta sección se presentarán las instrucciones relacionadas directamente con todos los procesos básicos para el establecimiento y gestión de conexiones utilizando Bluetooth.

Una de las operaciones fundamentales independiente de todo proceso de establecimiento de conexión en Bluetooth es el descubrimiento de las propiedades del dispositivo local, manejada por la clase exclusiva *LocalDevice* en JABWT. Para obtener el objeto que permite el manejo local se utiliza el método *LocalDevice.getLocalDevice()* que puede ejecutar una excepción del tipo *BluetoothStateException* si existe algún problema con la parte física de la implementación Bluetooth.

Entre los varios métodos existentes resalta *LocalDevice.getProperty()*, puesto que permite conocer las características del sistema e información adicional sobre la implementación de la pila con las posibles restricciones que el fabricante del dispositivo pudo haber incluido. Este método resulta útil para conocer mejor acerca de cómo trabaja la pila Bluetooth en los dispositivos. En la tabla 4.25 se muestra una lista con la información más relevante que se puede obtener.

Tabla 4.25. Información obtenida a partir del método *getProperty()*

Propiedad	Descripción
bluetooth.api.version	Indica la versión de JABWT más no la de la implementación Bluetooth.
bluetooth.master.switch	Indica si es posible que el dispositivo sea maestro o esclavo.
bluetooth.connected.devices.max	Número máximo de dispositivos conectados que se puede manejar
bluetooth.sd.trans.max	Número máximo de procesos de descubrimiento de dispositivos que se puede tener al mismo tiempo.
bluetooth.connected.inquiry.scan	Indica si es posible que el dispositivo local responda ante una petición de descubrimiento cuando ya ha establecido un enlace con otro dispositivo.
bluetooth.connected.page.scan	Indica si es posible que el dispositivo local acepte la conexión de establecimiento de enlace cuando ya ha establecido uno con otro dispositivo.
Bluetooth.connected.inquiry	Indica si posible que el dispositivo

	local inicie un proceso de búsqueda cuando ya ha establecido un enlace.
Bluetooth.connected.page	Indica si posible que el dispositivo local solicite el establecimiento de un enlace cuando ya tiene uno con otro dispositivo.

5.3.1 Operaciones de descubrimiento de dispositivos y servicios.

Como se mencionó en el capítulo 3, antes de establecer una conexión un dispositivo Bluetooth primero realiza una búsqueda sobre los dispositivos cercanos con los que puede comunicarse. Posteriormente, realiza una búsqueda de servicios que puede obtener de cada uno de aquellos dispositivos. Es así como toda conexión Bluetooth comienza, por ello es importante presentar las clases, métodos y definiciones que permiten realizar la búsqueda de dispositivos y servicios en JABWT.

El proceso de búsqueda de dispositivos es conocido como *inquiry* y permite identificar a los dispositivos que se encuentren en determinada área. Para ello, a nivel físico, todo dispositivo Bluetooth tiene un identificador único conformado por 6 bytes, el cual es incluido por el fabricante y que además proporciona información básica acerca del mismo y de los servicios que tiene disponibles. Un dispositivo puede iniciar una búsqueda de tipo general o una de tipo limitada; la diferencia entre ambas es que la primera permite identificar a todos los terminales que se encuentren en determinada zona mientras que la otra filtra la información identificando a aquellos dispositivos que cumplen con determinadas características. Para Bluetooth v2.0 + EDR se definió un nuevo tipo de *inquiry* que permite conocer cuáles dispositivos cumplen con esa especificación; sin embargo, esta búsqueda en naturaleza no difiere mucho de una búsqueda de carácter limitado.

En general, un dispositivo puede estar en estado de descubrimiento general o limitado cuando desea ser contactado por otros dispositivos o en estado invisible

cuando no desea serlo. Un dispositivo en estado de descubrimiento general responde únicamente a peticiones de descubrimiento del mismo tipo, mientras que un terminal en estado de descubrimiento limitado puede responder a peticiones de descubrimiento del mismo y a peticiones generales. Obviamente, un dispositivo invisible no responderá a ningún tipo de petición de descubrimiento. La información enviada como respuesta es la dirección física e información de registro al dispositivo que inició con la búsqueda. La información de registro de cada dispositivo se encuentra compuesta por la clase principal de servicio, la clase principal de dispositivo y la clase inferior de dispositivo. En la tabla 4.26 se muestran los diferentes tipos de clases principales de servicio, mientras en la tabla 4.27 se muestran las principales clases de dispositivos definidos hasta hoy por el Bluetooth SIG, las clases inferiores de dispositivos derivan de las principales[62].

Tabla 4.26. Clases de servicio definidas en Bluetooth

Servicio	Tipo de servicio
Limited Discoverable Mode	El dispositivo se encuentra en modo de descubrimiento limitado
Positioning	El dispositivo permite identificar la posición actual.
Networking	El dispositivo tiene la capacidad de establecer redes (LAN, WAN, AD-Hoc, etc)
Rendering	Es un dispositivo de reproducción (impresora, parlantes,etc)
Capturing	El dispositivo permite la captura de medios (escáner, micrófono, etc)
Object Transfer	El dispositivo permite la transferencia de archivos utilizando GOEP
Audio	El dispositivo permite la utilización de servicios de audio
Telephony	El dispositivo permite establecer conexiones de audio dedicadas a voz.
Information	El dispositivo puede actuar como servidor de información (Web, Wap, etc)

Tabla 4.27 Clases de dispositivos definidos en Bluetooth

Clase de Dispositivo	Ejemplos
Computer	Laptop, Desktop, PDA.
Phone	Celular, Inalámbrico, módem.
LAN/Access Point	Puntos de acceso de todo tipo.
Audio/Video	Micrófonos, parlantes, VCR, monitor.
Peripheral	Teclado, ratón, joystick
Imaging	Impresora, escáner
Wearable	Aplicaciones de Bluetooth de bajo consumo de energía
Toy	Juguetes en general
Health	Aplicaciones de Bluetooth de bajo consumo de energía
Miscellaneous	Cualquier dispositivo que no se incluya dentro de las categorías anteriores.

Es necesario tomar en cuenta que un proceso de búsqueda de dispositivos puede tomar de ocho a diez segundos, en los cuales se tiene la probabilidad de encontrar al 95% del total de dispositivos. Pero además de tiempo también la búsqueda utiliza recursos de la batería del dispositivo, por lo que JABWT trata de reducir al mínimo la cantidad de veces que resulta necesario iniciar un proceso de búsqueda. Para ello se define el concepto de dispositivos conocidos como aquellos con los cuales se interactúa frecuentemente y el de los dispositivos en caché como aquellos que fueron encontrados en un proceso previo de búsqueda. Los dispositivos conocidos permanecen en un registro para ser accedidos directamente sin la necesidad de iniciar un proceso de búsqueda mientras que los dispositivos en caché permanecen durante un tiempo registrados en el dispositivo, que considera que existe la probabilidad de que aún estén disponibles y de que pueden ser accesibles. El tiempo y forma en que se manejen los registros de estos tipos de dispositivos dependerá del BCC y de lo definido por el fabricante.

A través de *LocalDevice* podemos definir el estado en el que el dispositivo se encuentra al momento de ser encontrado. La clase *DiscoveryAgent* permite conocer si el dispositivo se encuentra en estado de descubrimiento general, limitado o si está

invisible mediante el método *getDiscoverable()*. Además, mediante el método *setDiscoverable()* es posible cambiar el estado actual de descubrimiento del dispositivo según los argumentos que se muestran en la tabla 4.28.

Tabla 4.28. Argumentos para definir estado de descubrimiento en *setDiscoverable()*

Argumento	Descripción
DiscoveryAgent.GIAC	Permite establecer el dispositivo en modo de descubrimiento general
DiscoveryAgent.LIAC	Permite establecer el dispositivo en modo de descubrimiento limitado
NOT_DISCOVERABLE.-	Permite establecer el dispositivo en modo invisible.

Debido a que el BCC es el encargado de manejar el estado de descubrimiento en el que se encuentra el dispositivo, es posible obtener una excepción del tipo *BluetoothStateException* si la BCC no admite los cambios realizados. Más aún JABWT tampoco indica qué debe hacer el BCC cuando se están ejecutando varias aplicaciones que trabajan bajo diferentes tipos de modos de descubrimiento, simplemente se dan tres sugerencias que pueden o no ser adoptadas por los fabricantes:

- Se mantiene el modo de descubrimiento inicializado por la primera aplicación que fue ejecutada hasta que esta se cierre.
- Se mantiene el modo de descubrimiento inicializado por la aplicación ejecutada más recientemente hasta que esta se cierre.
- Establecer una jerarquía de modos de descubrimiento que sea utilizada por las aplicaciones. Por ejemplo, se puede establecer al modo de descubrimiento como el más importante y al modo invisible como el menos importante por lo que una aplicación que utilice un modo de descubrimiento general obligará a todas a usarlo por ser la de más alta jerarquía.

Una vez que se han encontrado los dispositivos es posible conocer los servicios que cada uno de ellos ofrece. Tal búsqueda se realiza únicamente entre dos terminales, la una que funciona como servidor y muestra sus servicios y la otra como

cliente que puede o no hacer uso de ellos. Para que un servicio esté presente en un dispositivo que funciona como servidor es necesario que sea registrado en su SDDB especificando claramente cuál es su función y la manera en la cual un cliente puede acceder a él. La especificación Bluetooth no indica de qué manera se debe incluir un servicio en el registro por lo que es posible que los diferentes fabricantes tengan diversas maneras de hacerlo. Sin embargo, esto no debe afectar la interoperabilidad de dispositivos, más aún una aplicación que cumpla con las especificaciones de JABWT debe tener la capacidad de ser portable entre diversas plataformas de sistemas operativos y terminales de diversos fabricantes.

Básicamente se pueden identificar dos tipos de servicios que pueden ser ofrecidos por un dispositivo. Aquellos definidos por los perfiles Bluetooth son aplicaciones se han convertido en un estándar y que han sido incluidos como operaciones comunes dentro de la especificación Bluetooth [62]. Para que un servicio sea incluido dentro de un perfil es necesario que complete un proceso de calificación que es certificado por el Bluetooth SIG. Estos servicios pueden ser incluidos como aplicaciones nativas dentro del sistema operativo sin la necesidad de que utilicen a JABWT. Por otro lado, existen los servicios personalizados los cuales no son incluidos dentro de los perfiles Bluetooth y que no tienen ningún proceso de calificación. Tales servicios necesitan que una aplicación específica se encuentre instalada tanto en el dispositivo cliente cuanto en el servidor para que funcione correctamente. Por tanto, se puede concluir que la aplicación de mensajería a desarrollarse en este proyecto de tesis será personalizada e incluirá una aplicación que funcione para un terminal cliente y para otro servidor.

Cuando una aplicación busca por un servicio lo hace mediante la identificación de un valor de UUID (*Universal Unique Identifier*), que no es más que una secuencia única de bits que identifica las características de un servicio. Los valores UUID pueden ser encontrados en la asignación de números del Bluetooth SIG, más aún los que son utilizados bajo perfiles, pero otros pueden ser definidos por las aplicaciones, sobre todo las que son personalizadas. Es así que cuando un servicio ha encontrado

un servicio lo que realmente identificó fue un valor UUID y lo reporta mediante el evento *servicesDiscovered()*.

Los tipos de servicios también pueden ser categorizados dependiendo de la forma en la que el cliente accede a ellos. El más utilizado por la sencillez de registro de servicios personalizados es el conocido como *run-before-connect*, en el cual se tiene como premisa que la aplicación del servidor se encuentra ejecutándose y lista para recibir las peticiones del cliente. La figura 4.27 muestra un esquema de funcionamiento acerca de cómo la aplicación en el servidor hace el registro del servicio al utilizar JABWT. Los cuadros indican los objetos que generan eventos mientras que las flechas son los mensajes que son enviados entre ellos. Se puede identificar claramente a un objeto conector que se encarga de realizar la creación del registro de servicio, a otro objeto notificador que permite la inclusión del registro del servicio en la SDDB del dispositivo y finalmente a los objetos de registro de servicio y de registro en la SDDB que son los encargados de manejar la información del servicio y la ubicación en la SDDB del dispositivo respectivamente.

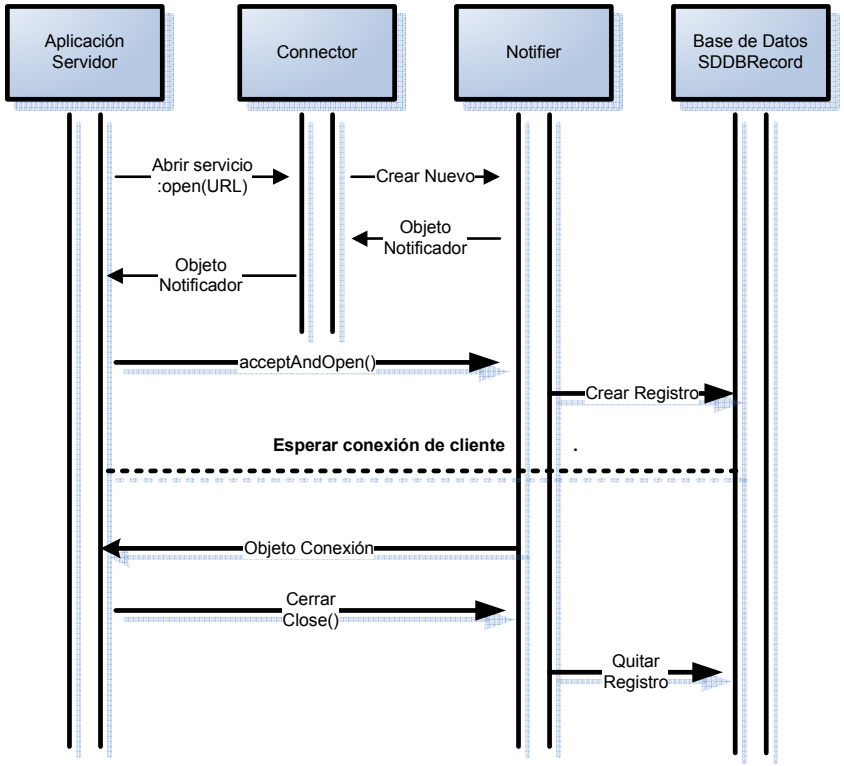


Figura 4.27 Ciclo de vida de una aplicación *run-before-connect*

De la figura 4.27 también podemos identificar tres métodos que son enviados en forma de mensaje desde la aplicación del servidor al conector:

- Open (URL).- Que permite crear un registro de servicio Bluetooth si en el parámetro URL se incluye *btspp://localhost:* para una conexión serial, *btgoep://localhost* para una conexión OBEX o *btl2cap://localhost* para una conexión basada en L2CAP.
- acceptAndOpen().- Especifica cuando se crea una copia del registro del servicio en la SDDB del dispositivo. Este comando comúnmente es ejecutado cuando existe la petición de conexión del primer cliente.
- Close().- Permite que el objeto notificador borre el registro del servicio en la SDDB mas el objeto conector no pierde la información del servicio durante un tiempo determinado por la BCC.

Exclusivamente estos tres métodos son utilizados para la creación, registro y eliminación de servicios en JABWT. Es así que resulta evidente que el proceso de registro no puede ser manipulado directamente por la aplicación JABWT debido a que únicamente tiene acceso al objeto conector, el cual no interactúa directamente con la SDDB. Es por ello que se dice que JABWT tiene un proceso de registro de servicios casi automático que facilita el proceso de establecimiento de los mismos.

El método open del objeto conector es el único que permite hacer el registro de la aplicación. Su sintaxis se muestra en la figura 4.28 y dependiendo del protocolo que se vaya a utilizar la URL puede empezar con *btspp* para conexiones de emulación del puerto serial, *btgoep* para conexiones que utilicen OBEX y *btl2cap* para conexiones L2CAP. El parámetro *name* indica el nombre con el cual el servicio será añadido al registro.

```
Connector.open("URL"; name="nombre")
```

Figura 4.28. Sintaxis de *Connector*

La figura 4.29 muestra más específicamente un ejemplo de establecimiento de registro para una sesión OBEX, donde se observa claramente que el servicio se

establece localmente mediante *localhost* y que existe un número UUID único asociado al mismo. Como ya se mencionó anteriormente, el valor UUID es único dependiendo del servicio que se vaya a ofrecer, para este caso específico el valor ingresado no hace referencia a ninguno de los valores UUID definidos para los perfiles Bluetooth. Finalmente, se puede destacar que la abreviación *btgoep* hace referencia al perfil genérico de intercambio de objetos (GOEP) al que OBEX se encuentra bastante ligado.

```
Connector.open(btgoep://localhost:0E18AE04148A11D7929B00B0D03D76EC;
name=”Servidor_Obex")
```

Figura 4.29 Ejemplo de sintaxis de *Connector*

Si bien el registro de servicios es único para cualquier tipo de protocolo que se vaya a utilizar en la comunicación, el registro en la SDDB y el cierre de conexiones no funcionan de la misma manera. En la tabla 4.29 se muestran las diferentes interfaces utilizadas para el manejo de los registros en la SDDB y el cierre de conexiones.

Tabla 4.29. Interfaces utilizadas para el manejo de registros en SDDB

Protocolo	Interface para añadir/eliminar servicios al SDDB	Método abrir conexión	Método cerrar conexión
Emulación Serial	StreamConnectionNotifier	acceptAndOpen()	Close()
L2CAP	L2CAPConnectionNotifier	acceptAndOpen()	Close()
OBEX	SessionNotifier	acceptAndOpen()	Close()

El proceso de búsqueda de servicios puede ser algo bastante simple o complejo, todo dependerá de cuál sea el objetivo de la aplicación que estamos desarrollando y de cómo utilizamos los diferentes métodos de búsqueda. El método fundamental para realizar la búsqueda de servicios es *searchServices()*, que es parte de la clase *DiscoveryAgent* y que está compuesto por cuatro argumentos. El primero indica la lista de atributos del registro de servicio buscado; por defecto se buscarán

los valores incluidos en el registro que fueron incluidos en el servidor mediante el método *open()*. El segundo es la lista de valores UUID que deben ser localizados en el servidor o dispositivo remoto, mientras más valores UUID más específica será la búsqueda. El tercer argumento especifica mediante el objeto *RemoteDevice* cuál es el dispositivo del cual se desea obtener servicio. El cuarto argumento es el objeto *DiscoveryListener* que notifica cuando los servicios fueron encontrados. El método que utiliza para notificar sobre un servicio es *servicesDiscovered()*, que además envía un identificador acompañado al mismo, puesto que puede ser llamado varias veces. Si varios servicios son encontrados, estos se ordenan dentro de un arreglo de objetos *ServiceRecord* junto con los atributos de los mismos.

Cuando no es posible ejecutar el método *searchServices*, debido a errores en el manejo de la pila bluetooth, se obtendrá una excepción del tipo *BluetoothStateException.* Una vez completada la búsqueda de servicios, se ejecutará el método *serviceSearchCompleted()*, cuyos posibles valores se muestran en la tabla 4.30. Cuando se desea cancelar la búsqueda de servicios se puede evocar el método *cancelServiceSearch()*.

Tabla 4.30. Respuestas obtenidas a partir de *serviceSearchCompleted()*,

Respuesta de *serviceSearchCompleted()*	**Causa**
SERVICE_SEARCH_COMPLETED	La búsqueda de servicios en al menos un dispositivo fue exitosa
SERVICE_SEARCH_TERMINATED	La búsqueda fue cancelada utilizando el método *cancelServiceSearch()*
SERVICE_SEARCH_ERROR	Ocurrió un error durante la búsqueda de servicios
SERVICE_SEARCH_NO_RECORDS	No existe el servicio buscado
SERVICE_SEARCH_DEVICE_NOT_REACHABLE	El dispositivo remoto no puede ser localizado

Si bien los métodos anteriormente mencionados permiten realizar el descubrimiento de dispositivos y servicios de manera bastante específica, no siempre es necesario realizar un proceso tan detallado. De hecho, en la

especificación para JABWT se define el método *selectService()* que permite realizar la búsqueda de dispositivos y sus servicios de manera simple y rápida.

El método *SelectService()* devuelve una cadena de texto que puede ser utilizada por *Connector.open()* para realizar la conexión a determinado servicio en un dispositivo. Aún si el servicio no puede ser encontrado el método devuelve la cadena de tipo *null.* Los parámetros que necesita son el valor UUID a buscar en el dispositivo servidor, las especificaciones de seguridad mínimas para establecer la conexión y finalmente el especificar si el dispositivo debe funcionar como maestro en la conexión.

Es recomendable ejecutar cualquier método de búsqueda de servicios o dispositivos en un proceso o *thread* paralelo, debido a que JABWT no especifica de que manera *selectService()* realiza el descubrimiento de dispositivos y es posible que se inicie con una *inquiry* que deje la pantalla de la aplicación sin respuesta, mientras se ejecuta.

Otro aspecto a considerar es el valor de UUID. Si bien se puede especificar un número arbitrario hexadecimal como UUID, esto no garantiza que sea único y no genere algún conflicto con otra aplicación. Por ello, existen aplicaciones para generar UUIDs que tienen métodos que ofrecen una alta probabilidad de generar un número UUID único. Tal es el caso de la aplicación *GUIDGen* [63] que conjuga la dirección MAC de la máquina en la que se genera el código y la hora en la cual lo hace. La figura 4.30 muestra la interfaz de la aplicación que trabaja bajo Windows.

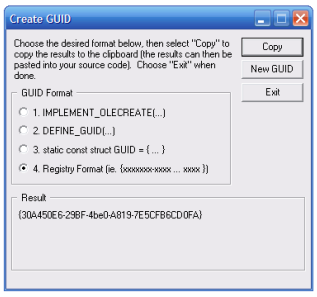


Figura 4.30 Entorno gráfico de *GUIDGen*

Es necesario mencionar que el rango de valores UUID desde 0000000-0000-1000-8000-00805F9B34FB hasta FFFFFFFF-0000-1000-8000-00805F9B34FB están reservados por el SIG de Bluetooth para especificaciones definidas en perfiles.

5.4 API de OBEX

EL API de OBEX es independiente del de Bluetooth en JABWT. Por tanto, un dispositivo que incluye Bluetooth no necesariamente tiene soporte para OBEX. Esto permite que su desarrollo sea independiente de los APIs de Bluetooth y que sea utilizado bajo diferentes medios de transmisión. Por otro lado, si bien este API permite tener control de bajo nivel sobre la comunicación establecida también esconde algunos detalles del protocolo, sobre todo aquellos detalles relacionados a la conversión a bits de las cabeceras que se intercambian y de las peticiones de conexión al inicio. Es por ello que, debido a que un desarrollador no puede definir el tamaño de los paquetes que se intercambiarán en la comunicación, es trabajo del API dividir en pequeñas partes las peticiones *PUT* y GET que se intercambiarán durante la comunicación.

Para el manejo de las conexiones existen funciones derivadas de la interfaz *javax.microedtion.io.Connection* que dependen del método *connector.open()* que devuelve tres interfaces según el estado de la conexión:

- *Javax.obex.ClientSession.*- Cuando se ha recibido una cadena de conexión del cliente.
- *Javax.obex.SessionNotifier.*- Cuando existen conexiones con el servidor.
- *Javax.obex.operation.*- Cuando se intercambian los mensajes PUT y GET en la comunicación.

Además, como métodos adicionales independientes dentro del API se tiene a:

- *Javax.obex.Authenticator*.- Que se puede utilizar cuando se desea manejar encriptación para la comunicación, como se mencionó en el capítulo anterior.
- *Javax.obex.HeaderSet.*- Que permite definir las cabeceras de OBEX que usualmente no tienen métodos específicos para hacerlo.

Finalmente, las clases exclusivas dentro del API son:

- *Javax.obex.PasswordAuthentication.-* Almacena el nombre de usuario y contraseña cuando se utiliza encriptación en la comunicación.
- *Javax.ResponseCodes*.- Define todos los códigos válidos que un servidor puede enviar a un cliente.
- *Javax.obex.ServerRequestHandler.-* Define los métodos que se evocan según las peticiones que recibe el servidor.

5.4.1 Establecimiento de una conexión

Para establecer una conexión, es necesario que primero se defina una cadena que debe ser enviada como parámetro a *connector.open()* tanto por el cliente como por el servidor. La figura 4.31 muestra la sintaxis estándar que debe usar tal cadena para cualquier medio de transmisión excepto para RFCOMM. Por ejemplo, si el protocolo de transporte para una conexión es TCP/IP el campo {transport} tendría el valor de tcpobex, mientras que en {target} se debe especificar la dirección IP y el puerto del servidor; en este caso no existen valores que puedan ser utilizados en {params}.

```
{transport}obex://{target}{params}
```

Figura 4.31 Sintaxis de una cadena estándar para conexión

Cuando se utiliza como medio de transporte a RFCOMM la sintaxis es diferente porque está basado en el perfil GOEP de Bluetooth. La figura 4.32 muestra la sintaxis de la cadena que debe ser utilizada, donde el campo {target} debe incluir la dirección Bluetooth y el canal RFCOMM a ser utilizado mientras que los parámetros válidos que pueden ser enviados en {params} se incluyen en la tabla 4.31.

```
btgoep://{target}{params}
```

Figura 4.32 Sintaxis específica de conexión para RFCOMM

Tabla 4.31 Valores de parámetros que pueden ser utilizados en una cadena de conexión en RFCOM

Nombre	Descripción	Valor
master	Especifica si el dispositivo debe actuar como maestro en la conexión	True, False
authenticate	Especifica si es necesario autenticar al usuario remoto antes de iniciar una conexión	True, False
encrypt	Especifica si se debe utilizar un medio encriptado para la conexión	True, False
authorize	Especifica si cada una de las peticiones al servicio necesita ser autorizada	True, False
name	Especifica el nombre del servicio en *Service Record*	Cualquier cadena

Cuando se ha enviado la cadena de conexión, el método *Connector.open()* devuelve el objeto *ClientSession*, el cual indica que se ha establecido un enlace a nivel de la capa de transporte pero no que existe uno a nivel de OBEX. Para ello se debe utilizar el método *ClientSession.connect()* para iniciar la comunicación y *ClientSession.disconnect()*para cerrarla. Es necesario que la comunicación bajo OBEX siempre se cierre antes de cerrar la establecida en la capa de trasporte. Por otro lado, exclusivamente al servidor se le envía la interfaz *SessionNotifier* que le permite aceptar las conexiones del cliente mediante el método *acceptAndOpen()* que retorna un objeto *Connection* que representa la capa de transporte de la conexión establecida con el cliente. El servidor debe implementar los métodos a los cuales debe responder; por ejemplo, si desea responder a peticiones GET debe implementar *onGet().*

- **Comunicación con el servidor**

Como se explicó en el capítulo anterior, es necesario que el cliente realice una petición CONNECT para iniciar la comunicación. Para ello, el servidor debe implementar el método *onConnect()* y el cliente debe usar el método *getResponseCode()* para conocer la respuesta del mismo. En la tabla 4.32 se muestra una lista de los posibles valores que el servidor puede enviar [64]. Cuando

el cliente recibe una respuesta *OBEX_HTTP_OK* se indica que la conexión a nivel de la capa de OBEX se ha establecido correctamente.

Tabla 4.32. Posibles respuestas enviadas por el servidor al establecer una conexión

Valor	Descripción
OBEX_HTTP_OK	Indica que la petición fue aceptada.
OBEX_HTTP_BAD_REQUEST	La petición no es la adecuada.
OBEX_HTTP_CONFLICT	Existe un conflicto a nivel de capa de transporte
OBEX_HTTP_FORBIDDEN	Indica que el enlace está prohibido
OBEX_HTTP_NOT_ACCEPTABLE	Indica que la petición no puede ser aceptada
OBEX_HTTP_NOT_IMPLEMENTED	Indica que el servidor no ha implementado el método necesario para dar una respuesta
OBEX_HTTP_NOT_AUTHORITATIVE	Indica que no existe una autorización para acceder al servicio

La mayor parte de transferencia de información se produce mediante operaciones GET y PUT, que permiten enviar el contenido del cuerpo del mensaje. Ambas operaciones generan un objeto conocido como *Operation* que permite controlar el flujo de la información que se transmite. Cuando se desea obtener la información proveniente de un emisor es necesario utilizar los métodos *InputStream()* o *DataInputStream()* mientras que para enviarla se utilizan *OutputStream()* y *DataOutputStream,* respectivamente. En la figura 4.33 se muestra un esquema de cómo se realiza el intercambio de datos entre el cliente y el servidor en una sesión OBEX basada en JABWT, cabe mencionar que la implementación en el dispositivo es la encargada de la traducción en cabeceras de la información que se envía.

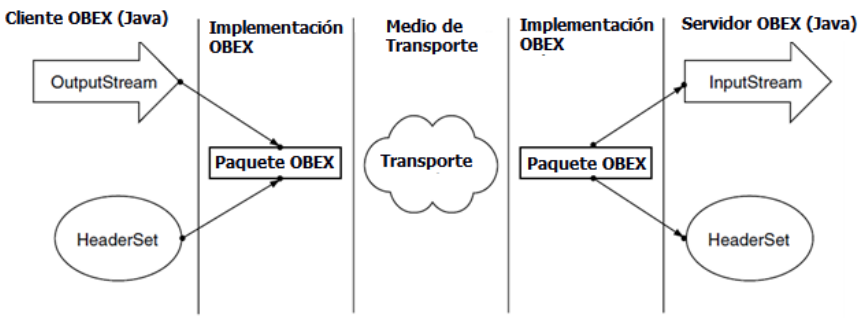


Figura 4.33 Intercambio de paquetes OBEX en un entorno Cliente-Servidor

Las operaciones PUT y GET tienen diferencias esenciales que deben ser tomadas en cuenta al momento de utilizarlas. Las figuras 4.34 y 4.35 muestran que los tres estados básicos de funcionamiento tanto para las operaciones *PUT()* cuando *GET()* son petición, respuesta y terminada pero los cambios entre ellos se producen de diferente manera. Mientras que en una operación *PUT()* para pasar del estado de petición al de respuesta basta con cerrar el método *OutputStream* o invocar a *getresponsecode()* del objeto *Operation*, para la operación *GET()* también es posible utilizar métodos como *openDataInputStream()* o *openInputStream()* para entrar al estado de respuesta. El cerrar el objeto operación en cualquier momento producirá que tanto la operación GET() cuanto PUT() terminen; pero por otro lado, el utilizar el método *getResponseCode()* únicamente provocará que la terminación de la operación GET().

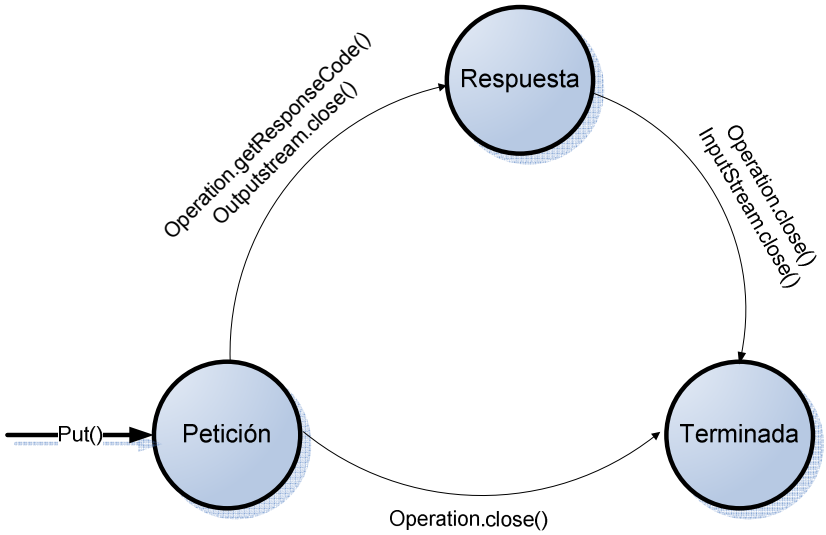


Figura 4.34. Diagrama de estados para la operación *PUT()*

Es necesario tener cautela al momento de evocar los métodos y objetos cuando se utilizan las operaciones PUT() y GET() debido se encuentran divididas en pequeñas transacciones de peticiones y respuestas. Así por ejemplo, el llamar el método *read()* dentro de *InputStream()* antes de cerrar a OutputStream, cualquiera sea el caso, producirá que la aplicación no responda debido a que no se ha recibido la totalidad de la información a ser transmitida. Por otro lado, el cliente en cualquier instante puede terminar cualquier sesión mediante el método *abort()*, el cual provoca

que tanto *InputStream, OutputStream* y *Operation* sean cerrados sin importar el estado de las operaciones PUT() o GET().

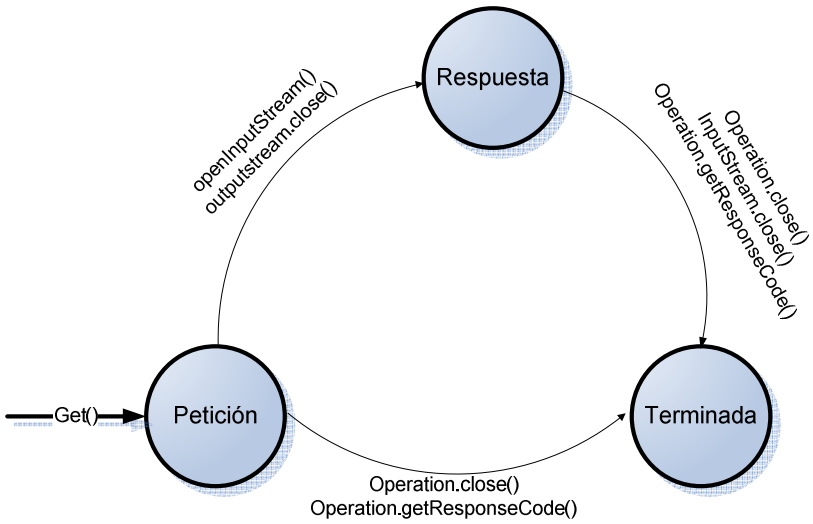


Figura 4.35. Diagrama de estados para la operación *PUT()*

- **Implementación en servidores**

La implementación de una aplicación para el servidor OBEX es ligeramente más sencilla que para un cliente, esto en gran parte a que siempre este último es el encargado de iniciar la comunicación y de solicitar obtener los servicios del primero. La clase fundamental que debe utilizarse es *ServerRequestHandler* en la cual se deben cargar únicamente los métodos a los cuales el servidor estará dispuesto a responder. Por ejemplo, si el servidor responderá a operaciones *PUT* y *CONNECT* los métodos que deben ser implementados son *onConnect()* y *onPut().* En la tabla 4.33 se muestran los métodos que pueden ser implementados acorde a las operaciones que el servidor desee responder.

Tabla 4.33. Servicios que pueden ser implementados en un servidor OBEX

Operación	Método
CONNECT	onConnect()
SETPATH	onSetPath()
GET	onGet()
PUT	onPut()
DELETE	onDelete()
DISCONNECT	onDisconnect()

Si existe una petición por parte de un cliente a una operación que el servidor no ha implementado se le enviará un mensaje que incluya el código *OBEX_HTTP_NOT_IMPLEMENTED* indicándole que es imposible responder a su solicitud. Como ya se mencionó anteriormente, toda la información se transmite mediante cabeceras, que indican cuál es la operación que se desea establecer y parámetros adicionales, si es que son necesarios. El servidor puede utilizar el método *getReceivedHeaders()* para obtener información acerca de las últimas cabeceras recibidas mediante la creación de un objeto *HeaderSet.*

No es posible crear cabeceras adicionales a las incluidas en la implementación de JABWT, pero estas pueden ser personalizadas mediante la utilización del método *createHeaderSet()* con la creación del objeto *HeaderSet* por parte del cliente para la transferencia de información. Una vez creado *HeaderSet,* el método *setHeader()* permite agregar cabeceras simplemente indicando su identificación y valor. En la tabla 4.34 se indica una lista de cabeceras con su significado acorde a la especificación OBEX.

Tabla 4.34 Tipos de cabeceras existentes en OBEX

Valor	Significado	Tipo de variable
COUNT	Se utiliza en la operación CONNECT para especificar el número de objetos que se transmitirán en la comunicación	java.lang.Long
NAME	Nombre del objeto	java.lang.String
TYPE	Tipo de objeto[65]	java.lang.String
LENGTH	Tamaño del objeto	java.lang.Long
TIME_ISO_8601	Incluir el tiempo según ISO 8601	java.util.Calendar
TIME_4_BYTE	Incluir el tiempo en formato de 4 bytes	java.util.Calendar
DESCRIPTION	Breve descripción del objeto	java.lang.String
TARGET	Servicio OBEX objetivo	byte[]
HTTP	Indica que se incluirá una cabecera HTTP	byte[]
APPLICATION_PARAMETER	Especifica un parámetro específico de la aplicación	byte[]

0x30 a 0x3F	Cabecera definida por usuario para enviar una cadena de caracteres	java.lang.String
0x70 a 0x7F	Cabecera definida por usuario para enviar un arreglo de bytes	byte[]
0xB0 0xBF	Cabecera definida por usuario para enviar un byte.	java.lang.Byte
0xF0 a 0xFF	Cabecera definida por usuario para enviar un entero sin signo.	java.lang.Long

De la tabla 4.34, la columna variable tipo indica el tipo de dato que debe ser enviado al momento de darle un valor al objeto, si no se utiliza el tipo de dato indicado se ejecuta una excepción del tipo *IllegalArgumentException.* Además, se observa que las cabeceras identificadas desde 0x30 hasta 0xFF pueden ser utilizadas por usuarios para enviar cualquier tipo de información, comúnmente estas cabeceras sirven para el intercambio de datos al utilizar operaciones *PUT* y *GET*. Es necesario indicar que la única cabecera que no puede ser modificada por el usuario es *CONNECTION-ID* ya que permite diferenciar los múltiples servicios que se están ofreciendo y únicamente la implementación JABWT puede acceder a ella. La mayoría de cabeceras son enviadas desde el lado del cliente, sin embargo el servidor puede crearlas mediante el método *createHeaderSet() en ServerRequestHandler.*

Las operaciones GET y PUT trabajan de un modo levemente diferente, debido a que utilizan el objeto *Operation* que permite tener acceso a la cabecera *BODY*, la cual maneja el cuerpo del mensaje que se transmite. Es necesario indica que el objeto *Operation* es diferente a aquel definido para el cliente, por lo que no posee las restricciones establecidas de cambio de estado tanto para *InputStream* cuanto para *OutputStream*. Por otro lado, no es posible invocar a los métodos *getResponseCode()* y *abort()* desde el lado del servidor; sin embargo, cuando se recibe una petición *abort()* se cierra el objeto *Operation*.

Finalmente, es necesario mencionar que la implementación JABWT siempre verifica el código de respuesta que es enviado al cliente. Cuando se produce un error y la implementación produce un error no esperado el código que se envía al cliente es OBEX_HTTP_INTERNAL_ERROR.

CAPÍTULO 5

APLICACIÓN DE MENSAJERÍA

1. Introducción.

Después de haber presentado en capítulos anteriores el fundamento teórico necesario para la comprensión de la tecnología Bluetooth y su manejo mediante Java, en este se mostrará la lógica, el código y la implementación de la aplicación de Mensajería. Asimismo, los resultados obtenidos al ejecutarlo sobre diferentes dispositivos móviles existentes en el mercado

El desarrollo de la aplicación partirá de la construcción de una más simple y se mostrará por sobre todo su lógica. Además se procurará mantener su simplicidad para que pueda ejecutarse en la mayor cantidad de dispositivos móviles.

Será necesario evaluar la situación actual del mercado global y local con la finalidad de relacionar la aplicabilidad del programa de mensajería en casos reales. Para ello, se presenta información de mercados actualizada y relevante de las empresas líderes en el sector de estudios y análisis de tendencias tecnológicas.

Finalmente, será posible evaluar el desempeño de la aplicación bajo diferentes aspectos que permitirán determinar la factibilidad de implementar redes Ad-Hoc en la situación tecnológica actual.

2. Aplicación JABWT simple

Si bien en el capítulo anterior se trató a profundidad la sintaxis y estructura del lenguaje Java, es necesario también ejemplificar su aplicación real en un entorno de programación para una comunicación Bluetooth. Por ello, a continuación se mostrará la estructura de un programa básico que establecerá un enlace Bluetooth entre dos dispositivos.

El programa estará basado en un entorno de cliente-servidor, en donde el primero enviará un mensaje que será mostrado en la pantalla del servidor siempre y cuando se establezca el enlace Bluetooth entre ambos. Para ello, se establecerán los permisos necesarios de descubrimiento de dispositivos y servicios tanto en JABWT cuanto en el BCC. Por otro lado, se establecerá un enlace RFCOMM debido a su sencillez y flexibilidad, como se mostró en el capítulo tres.

Para esta aplicación se creará un MIDlet base que se conocerá como *BluetoothMIDlet* y que implementará una estructura básica que podrá ser utilizada tanto para la aplicación cliente cuanto para la del servidor al momento de ejecutarse. En la figura 5.1 se muestra un esquema del funcionamiento de la aplicación.

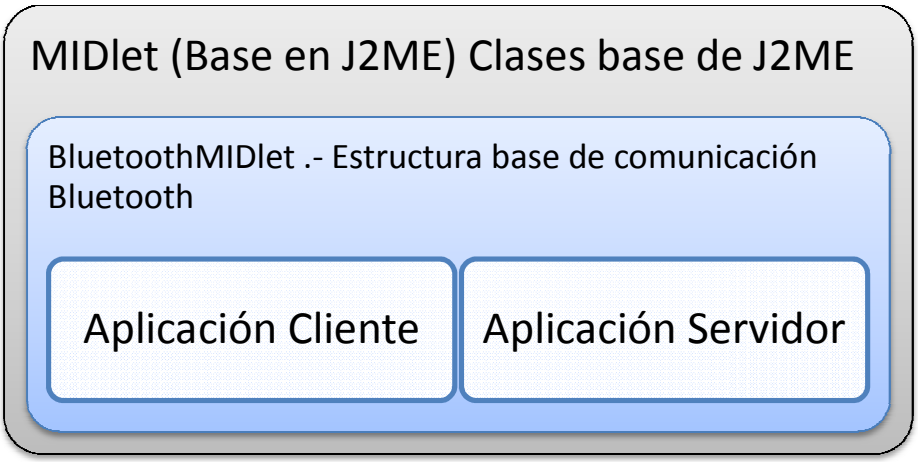


Figura 5.1 Abstracción de la aplicación de comunicación básica en BluetooothMIDlet

El código de *BluetoothMIDlet* es bastante simple pero muestra la aplicación de la teoría anteriormente expuesta sobre Java y más específicamente J2ME. En primer lugar se importan los paquetes necesarios para la ejecución, para luego crear el objeto que hereda las características básicas de un MIDlet y donde resaltan los estados de ejecución, pausa y finalización. En la figura 5.2 se muestra el código básico de *BluetoothMIDlet,* dónde se muestra esquemáticamente su composición.

```
package hello;                                          PAQUETE AL QUE PERTENECE

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;                      PAQUETES QUE SE INCLUYEN
import javax.microedition.io.*;
import javax.bluetooth.*;

public class BluetoothMIDlet extends MIDlet implements Runnable, CommandListener {
private boolean midletPaused = false;
public void startApp() throws MIDletStateChangeException {
     new Thread(this).start();
   }
public void pauseApp() {                                CÓDIGO DE LA APLICACIÓN
       midletPaused = true;
   }
   public void destroyApp(boolean unconditional) {}
}
```

Figura 5.2 Código base de BluetoothMIDlet

Todo MIDlet pertenece a un paquete, ya sea que haya sido incluido previamente en el API de desarrollo o que sea creado como parte de una aplicación específica. Para el caso de *BluetoothMIDlet* el paquete que lo contiene se denomina *hello* y se ha creado al mismo tiempo dentro de la aplicación de prueba. Un paquete lo podemos comparar con una carpeta que incluye varias hojas que tienen información similar o que cumplen un propósito; bajo ese esquema, las hojas son las clases. Por ello, en la figura 5.2 se observa que se incluyen varios paquetes que nos permitirán invocar a métodos existentes en J2ME. El símbolo asterisco al final de un paquete indica que se deben importar todas las clases contenidas dentro del

paquete. En la tabla 5.1 se especifica la función de los paquetes incluidos dentro de este MIDlet.

Tabla 5.1 Paquetes utilizados en *BluetoothMIDlet*

Paquete	Aplicación
javax.microedition.midlet.*;	Paquete fundamental de cualquier MIDlet.
javax.microedition.lcdui.*;	Permite tener control sobre la pantalla del dispositivo
import javax.microedition.io.*;	Permite manejar las operaciones de entrada y salida de datos
import javax.bluetooth.*;	Paquete fundamental para el manejo de operaciones Bluetooth

Como se ha indicado previamente, toda aplicación creada en J2ME es heredera de la clase MIDlet, por ello resulta necesario expresarlo en la declaración de la clase mediante la expresión *extends.* Por otro lado, en la misma declaración se pueden implementar otros métodos específicos de J2ME mediante la palabra reservada *implements.* Para el caso de *BluetoothMIDlet* se han implementado dos métodos específicos: *Runnable* y *CommandListener*, el primero permite manejar la ejecución del programa y el segundo los elementos de los botones de comando existentes en pantalla.

En la figura 5.2 se puede observar que el código de la aplicación refleja claramente lo expuesto previamente acerca de la estructura de un MIDlet. Cabe recalcar que este MIDlet no implementa código puesto que es una base sobre las que otros se ejecutarán. Por ello, únicamente se declaran variables y se toma una parte de memoria para la ejecución de la aplicación mediante *new Thread* que toma como parámetro al puntero *this* que siempre apunta a la posición de memoria disponible.

Como ya se mencionó anteriormente, se creará la aplicación de cliente sobre la estructura de BluetoothMIDlet. Para esto únicamente es necesario recordar que el dispositivo que inicia una comunicación Bluetooth se convierte en el esclavo, mientras el otro dispositivo ya debió previamente implementar los servicios que son

requeridos. Siendo así, se creará un entorno cliente-servidor bastante simple pero que servirá de apoyo para la aplicación de mensajería. En la figura 5.3 se puede observar un esquema de funcionamiento de la aplicación a implementarse, donde se identifican claramente tres etapas, la primera en la cual se identifica al servidor y su servicio, posteriormente se da el establecimiento de la conexión con todos los elementos que requiera (uso de PIN, clave, etc) y finalmente tenemos el envío de un único mensaje del cliente al servidor.

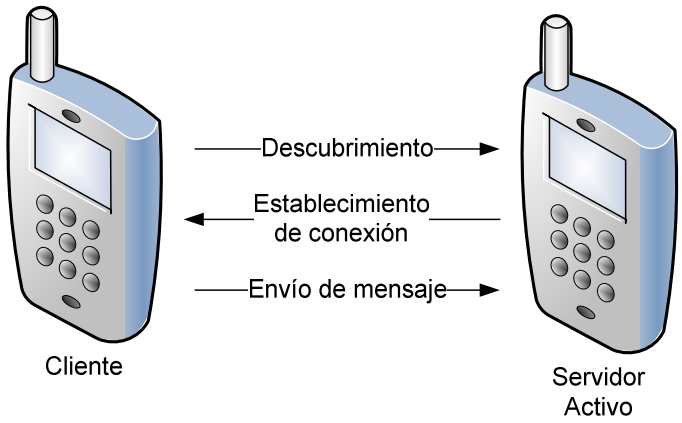


Figura 5.2 Diagrama de aplicación de comunicación básica en Bluetooth

Para la aplicación de cliente y servidor debemos considerar que únicamente será necesario implementar el método de ejecución *run()* ya que todo el manejo de memoria y control de pantalla se hace mediante la abstracción creada por el BluetoothMIDlet. Por ello, para crear la pantalla que se mostrará simplemente declaramos una variable del tipo *form,* en la cual también podemos añadir botones de comando, como se expuso en el capítulo anterior. En la figura 5.3 se puede observar como se ha creado una variable del tipo *form* para ser presentada en pantalla con el título de cliente, en la cual se incluye el botón de salida *Exit* con la más alta prioridad.

```
Form f= new Form ("Cliente");
    f.addCommand(new Command("Exit",Command.EXIT,1));
    f.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
```

Figura 5.3a. Establecimiento de la pantalla del MIDlet cliente.

Una vez establecido el entorno de usuario que manejará la aplicación es necesario definir las variables que permitirán manejar el enlace Bluetooth. Para ello, resulta necesario invocar al dispositivo sobre el que se ejecuta la aplicación mediante el método *getLocalDevice()* de la clase *LocalDevice* y asignarla a una variable del mismo tipo. De igual manera, se debe establecer una variable que permita realizar la búsqueda de servicios, para ello se utiliza el método *getDiscoveryAgent* de la misma clase *LocalDevice* y se le asigna a una variable de tipo *DiscoveryAgent*. En la figura 5.4 se muestra un ejemplo de declaración de variables para invocar al dispositivo local mediante la variable denominada local y para realizar la búsqueda de servicios mediante aquella denominada agente.

```
LocalDevice local=LocalDevice.getLocalDevice();
DiscoveryAgent agente= local.getDiscoveryAgent();
```

Figura 5.3b. Variables de manejo de conexión para el cliente de la aplicación

Una vez establecidas las variables de búsqueda y de manejo del dispositivo local se debe establecer una cadena de texto que permita realizar la conexión entre el cliente y el servidor. Esta cadena debe ser obtenida mediante la variable *agent* utilizando el método *selectService.* Como se mencionó anteriormente, el método de conexión será bastante sencillo por lo que no se autenticará el acceso al dispositivo servidor ni se encriptará la comunicación.

En la figura 5.4 se puede observar la sintaxis de la operación que nos permite declarar la variable conexión de tal manera que contenga la información para establecer la conexión entre el cliente y el servidor; además, se puede observar que se define las características de la conexión para que no se autentique ni tampoco se encripte la comunicación. Cabe mencionar que el valor UUID se asignó según lo indicado en el capítulo anterior para que su valor sea único y no interfiera con otras aplicaciones que se pudiesen estar ejecutando en el entorno.

```
UUID valorUUID = new UUID("86b4d249fb8844d6a75ec265dd1f6a3", false);
String conexion=agente.selectService(valorUUID, ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false);
```

Figura 5.4. Declaración de la cadena de conexión.

Cuando la conexión se ha establecido con éxito el valor que tomará la cadena será diferente a cero, tal condicionante debe ser utilizada para detectar y mostrar el mensaje de error en pantalla. En la figura 5.5 se muestra un ejemplo de condicionamiento donde se muestra un mensaje de error al momento de obtener una cadena de tipo *null* para realizar la conexión.

```
if(conexion!=null){
        }
    } else {
      //Mensaje de Error por no encontrar al servidor
      f.append("Imposible acceder al servicio");
    }
```

Figura 5.5. Validación de la cadena de conexión Bluetooth en J2ME

Una vez validada la cadena para la conexión, se debe establecer una variable de tipo *StreamConnection* que permita realizar la transferencia de información. Esta clase de datos se la puede obtener gracias a la clase *javax.microedition.io,* la cual además es la encargada de manejar las operaciones de entrada y salida de datos en J2ME. La variable a ser creada requiere como dato la cadena de conexión creada previamente y permite manejar un conjunto de operaciones de entrada y salida de información. Para el caso específico de esta aplicación, el cliente únicamente requiere enviar un mensaje al servidor y no en viceversa, por tanto solo se requiere tener el control de la salida de información mediante el método *openOutputStream()*. En la figura 5.6 se muestra un ejemplo en código de lo anteriormente expuesto, dónde con es la variable de manejo de las operaciones de entrada y salida, mientras que out permite manejar la salida de información hacia el servidor.

```
StreamConnection conn=(StreamConnection) Connector.open(conexion);
 OutputStream out=conn.openOutputStream();
```

Figura 5.6. Declaración de las variables de establecimiento de la conexión Bluetooth en la aplicación.

Una vez establecida la variable de salida de datos solo es necesario utilizar el método *write* para enviar la información. Para ello únicamente es necesario indicar el texto que se desea enviar seguido de la función *getBytes()* que permite traducir el texto plano a información que se pueda enviar por las capa de transmisión, siendo totalmente transparente para el programador. Una vez transmitida la información se debe cerrar la conexión con el método *close* tanto para la variable de manejo de entrada y salida de datos cuanto para la variable de control de salida de datos, en ese orden. Es necesario mencionar que cerrar la conexión de la variable de manejo de datos antes que la de control puede producir un error en el centro de control de conexiones Bluetooth (BCC) del dispositivo. En la figura 5.7 se puede observar el envío del mensaje Hola Mundo a través de la variable out utilizando el método write y la secuencia de cierre del flujo de información entre el cliente y el servidor.

```
out.write("Hola, mundo".getBytes());
out.close();
conn.close();
f.append("Mensaje Enviado Correctamente");
```

Figura 5.7. Ejemplo de transmisión de información utilizando el método *write* y cierre de una conexión Bluetooth establecida.

Al igual que una operación de ingreso de datos, una transmisión Bluetooth también se debe incluir dentro de una operación *try-catch*, debido a que pueden ocurrir imprevistos tanto en el manejo de datos por el manejo de *StreamConnection* cuanto en la transmisión de información por Bluetooth. En la tabla 5.2 se muestran los tipos de errores que se pueden dar durante la transmisión; sin embargo, en cualquiera de los dos casos el mensaje de error se lo puede conocer mediante el método *getMessage.*

Tabla 5.1 Tipos de excepciones que pueden ser generadas en una transmisión Bluetooth

Excepción	Razón
IOException	Errores producidos durante la transmisión de datos independiente del medio a utilizarse
BluetoothStateException	Errores producidos a nivel del centro de control Bluetooth (BCC)

El código de la aplicación del servidor mantiene una estructura similar a la presentada con la del cliente, según lo indicado en la figura 5.2, pero con un código de programación totalmente diferente que permite implementar los servicios solicitados. De igual manera, elementos del código como la creación de un formulario para que sea presentado al usuario, la inclusión de cabeceras y la detección de excepciones son similares a lo mostrado anteriormente. Por tanto, la parte esencial de la programación se encuentra en la operación *try-catch* que debe ser capaz de reconocer la información enviada desde el cliente.

El servidor después de implementar los servicios debe ser descubierto por el cliente. Por ello, es necesario que el mismo se encuentre en estado de descubrimiento general o *GIAC* como se describió en el capítulo tres, que trató sobre la tecnología Bluetooth. Para cambiar el estado de visibilidad de un dispositivo en J2ME se utiliza el método *setDiscoverable,* perteneciente a la clase del dispositivo local mediante la declaración de una variable del tipo *LocalDevice*, como se lo hizo para obtener acceso al dispositivo local en el cliente. En la figura 5.8 se muestra un ejemplo de código a implementar para cambiar la visibilidad del dispositivo a general y mostrar un error en pantalla en el caso en el que se produzca.

```
LocalDevice local=LocalDevice.getLocalDevice();
if (!local.setDiscoverable(DiscoveryAgent.GIAC)){
   pantalla.append("Fallo en cambiar a Modo descubierto");
  return; }
```

Figura 5.8. Ejemplo de código para cambiar modo de descubrimiento Bluetooth en J2ME

En la pantalla del servidor no se mostrará nada hasta que llegue el mensaje del cliente, por tanto es necesario establecer una variable que advierta acerca de que una conexión desea ser establecida. Para ello se debe declarar una variable del tipo *StreamConnectionNotifier* que establezca un canal para la entrada de información mediante una dirección específica. La sintaxis de la dirección puede ser observada en la figura 4.28, donde se utiliza el método *open* de la clase *Connector.* En este caso la conexión será del tipo serial y se utilizará el valor de UUID indicando en el cliente. En síntesis, la apertura del canal es la implementación del servicio puesto que la aplicación servidor lo único que hará es mostrar en pantalla el mensaje del cliente.

Una vez establecido el notificador hay que ligarlo con una cadena para el ingreso de datos, para ello es necesario declarar una variable del tipo *StreamConnection* que permita establecer una conexión con la información del notificador y otra del tipo *InputStream* para el ingreso de datos. Las operaciones de entrada y salida de datos mediante input y output funcionan de manera similar a lo expuesto en el capítulo cuatro sobre las operaciones put y get. En la figura 5.9 se muestra un ejemplo de código, donde se puede observar claramente la relación que tiene el notificador con la variable de establecimiento de la conexión y la de recepción de datos desde el cliente.

```
//Crear un objeto de conexion al servidor para aceptar la conexion del cliente
 StreamConnectionNotifier aviso_entrada= (StreamConnectionNotifier)
Connector.open("btspp://localhost:86b4d249fb8844d6a75ec265dd1f6a3");
 //Aceptar una conexion del cliente
StreamConnection conn=aviso_entrada.acceptAndOpen();
 //Abrir la entrada
InputStream entrada=conn.openDataInputStream();
```

Figura 5.9. Establecimiento y manejo de las variables de entrada de información en el lado del servidor en una aplicación J2ME básica.

Es propicio recordar que en la aplicación del cliente los datos fueron enviados en bytes de acuerdo a lo indicado en el método *getBytes,* por lo que será necesario

que la información sea convertida a texto una vez que ingrese. Para esto se conoce que el byte al final de la transmisión siempre será -1 y que se puede utilizar el método *toString* para el intercambio de bytes a texto para un vector.

Una vez finalizada la comunicación, se procede a cerrar el canal de comunicación de igual manera que se lo hizo para el cliente, de forma inversa a la manera en la que fueron declaradas. Es decir, primero la variable de ingreso de datos, luego la de manejo de comunicación y finalmente el notificador, todas mediante el método *close.*

En la figura 5.10 se muestra el comportamiento de la aplicación en dos teléfonos convencionales compatibles con J2ME mediante el simulador estándar incluido dentro de la interfaz de desarrollo Sun Java (TM) Wireless Toolkit 2.5.2_01 for CLDC. El comportamiento en un dispositivo móvil convencional dependerá mucho de los perfiles que el mismo implemente y de la BCC, sobre estos aspectos se verá más adelante en este capítulo.

Figura 5.10.Comportamiento de una aplicación de comunicación Bluetooth básica en el simulador.

3. Descripción de la aplicación de Mensajería

Una vez descrita y ejemplificada una comunicación básica en Bluetooth, se puede detallar más a profundidad sobre la aplicación de mensajería que se implementará para estudiar el comportamiento de los dispositivos móviles en la formación de Piconets.

Una vez establecido que el funcionamiento de una Piconet está basado en una relación entre un servidor con varios clientes y que las clases en J2ME permiten fácilmente crear aplicaciones bajo un entorno cliente-servidor es posible definir que la aplicación de mensajería tendrá al menos dos clases, una para el cliente y otra para el servidor.

Para este caso se debe considerar que el mensaje que un usuario envíe podrá tener un tamaño variable, por lo que no será recomendable que una sola variable sea la que lo maneje. Por otro lado, se debe incluir un campo para indicar el destinario del mensaje, para que el servidor pueda realizar el envío del mismo. Por tanto, será necesario crear una clase que contenga tanto el cuerpo del mensaje a enviarse cuanto uno para indicar el nombre del destinatario. En la figura 5.11 se muestra el esquema de funcionamiento de la aplicación con las tres clases anteriormente mencionadas.

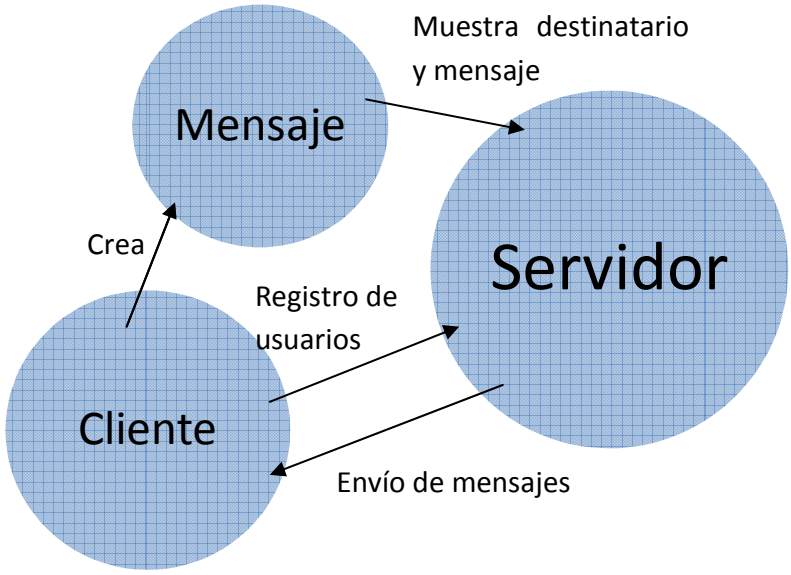


Figura 5.11.Esquema de funcionamiento de la aplicación de mensajería utilizando Bluetooth.

La programación de esta aplicación está totalmente orientada al manejo de objetos, según el pragmatismo de Java, lo cual permite que el procesamiento no esté enfocado en un solo código y a la vez facilita la detección de errores y posibles mejoras a la aplicación. Por ello, en adelante se mostrará los esquemas de programación y código de cada una de las tres partes anteriormente mencionadas.

3.1 Aplicación para el servidor en Java

Es necesario identificar claramente cuáles son las tareas que debe cumplir el servidor para poder escribir el código del mismo. Este trabajo permitirá entender mejor el funcionamiento de la aplicación y facilitará su escritura en cualquier lenguaje de programación.

Si se toma como premisa el entorno cliente-servidor en la que está basada la comunicación Bluetooth, resulta evidente que una de las principales tareas que tendrá el servidor es la de implementar los servicios antes de que el cliente los solicite. Por otro lado, debemos considerar las tareas básicas de conmutación y almacenamiento de mensajes para distribuirlos adecuadamente a los clientes. Y finalmente, el servidor debe ser responsable de la gestión de sus usuarios para evitar inconvenientes en la comunicación. Es así que en la tabla 5.2 se muestran seis tareas básicas que debe cumplir el servidor en el entorno de la aplicación de mensajería.

Tabla 5.2 Tareas y funciones del servidor de mensajería.

Tareas del Servidor	1. Debe estar presente para aceptar las solicitudes de los clientes. 2. Debe realizar el registro de los usuarios. 3. Debe guardar el contenido de los mensajes y el destinatario. 4. Debe conmutar correctamente la información a los usuarios cuando estos lo soliciten. 5. Debe validar las peticiones que vienen de los clientes. 6. Debe manejar las sesiones de los usuarios.

De las tareas indicadas en la tabla 5.2 pueden derivar otras más específicas relacionadas con el tipo o protocolo de comunicación y el lenguaje de programación que se utilice. Sin embargo, tales tareas no se muestran en la tabla porque pueden ser incluidas dentro de alguna de las categorías de la tabla 5.2; y por otro lado son

más parecidas a condicionamientos de la tecnología a utilizarse que responsabilidades del servidor.

La aplicación básicamente cumple con dos tareas, la de guardar los mensajes provenientes de todos los usuarios y de posteriormente conmutarlos cuando sea requerido. La tarea de guardar los mensajes se produce de manera lineal y únicamente requiere de la creación de una clase mensaje que sea guardada en la tabla de mensajes que mantiene el servidor. Por otro lado, resulta más complejo el proceso de conmutación y envío de mensajes, puesto que requiere de al menos tres etapas, la de recepción de la petición, búsqueda de mensaje y finalmente la del envío del mensaje al destinatario correcto. La figura 5.12 muestra el esquema de cambio de estado del servidor cuando existe una petición de envío de mensaje por parte de un cliente.

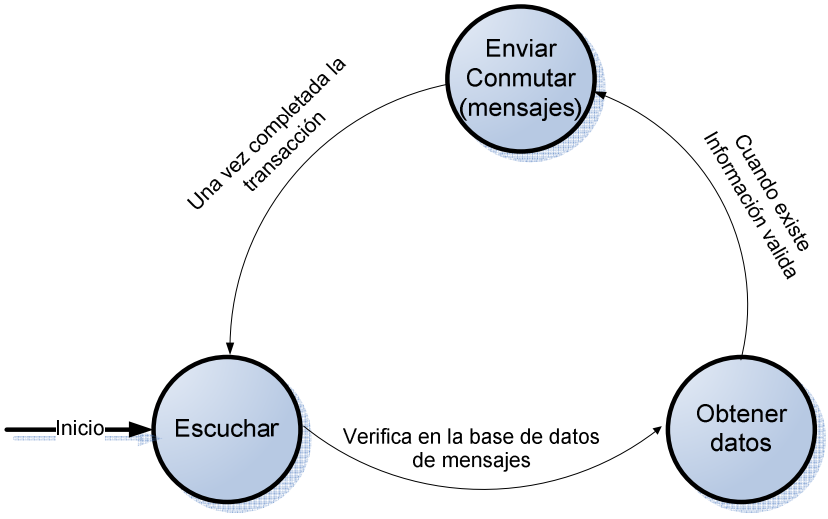


Figura 5.12.Esquema de estados del servidor en la aplicación de mensajería.

De la figura 5.12 se puede observar que siempre el estado de inicio para la operación de envío de mensajes es la de escuchar. De hecho, el servidor siempre deberá estar listo a recibir las peticiones que vienen por parte de los clientes. Este comportamiento es análogo al observado para el protocolo OBEX en las figuras 4.34 y 4.35 del capítulo anterior. Por tanto, siempre será el cliente quién comience la comunicación con el servidor para enviar u obtener los mensajes que requiera, algo que es totalmente compatible con una comunicación Bluetooth convencional. Más

adelante se expondrá en el código del programa que las operaciones de envío y obtención de mensajes se encuentran íntimamente relacionados con las funciones PUT y GET del protocolo OBEX.

Es necesario considerar para este caso que la estructura del servidor además de cumplir con las tareas de envío de mensajes debe tener una interfaz que sea capaz de interactuar con el usuario y de mostrar los mensajes de estado, que inclusive facilitarían la detección de errores. Por esto, el servidor debe constar de una clase específicamente diseñada para realizar las operaciones de conmutación de mensajes y otra para la interacción con el usuario; sin embargo, al usuario final únicamente se le debe mostrar una aplicación única e íntegramente eficiente. En la figura 5.13 se muestra el esquema de trabajo del servidor cuando consta de dos partes.

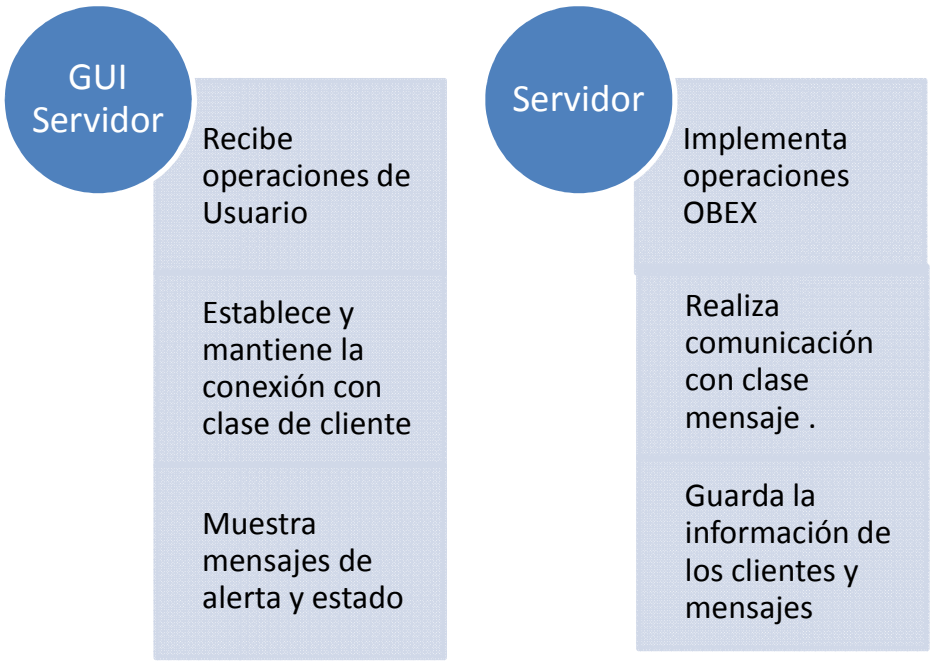


Figura 5.13.Esquema de las clases de trabajo del servidor en la aplicación de mensajería.

Por tanto, si bien la clase del servidor en esta aplicación es la más compleja porque debe implementar los servicios basados en OBEX y gestionar las operaciones de conmutación y envío de mensajes; su estructura bidimensional permite que cada clase se enfoque en una tarea específica. Tal división facilita las

mejoras en el código que se pudieran presentar, como manejo de bases de datos, medios de comunicación u procesamientos de memoria paralelos [66]

3.1.1 Código del Programa

Es importante realizar una revisión de las partes más relevantes del código para comprender de qué manera se puede implementar lo anteriormente expuesto como la dualidad del servidor. En ambas clases será necesario incluir las cabeceras que permiten manejar el protocolo OBEX. Sin embargo, únicamente en la clase de manejo de las peticiones de los clientes y conexión con la clase Mensaje se incluirán los procesos PUT y GET; en adelante a esta clase se la llamará Servidor, mientras que la de presentación y manejo de interacciones con el usuario será llamada appServidor.

Para la clase servidor se incluirán adicionalmente los paquetes *java.io* y *java.util*, el primero que permite la gestión de datos mediante flujos y el otro contiene varios utilitarios de trabajo entre los que se incluyen un generador de números aleatorios y el manejo de arreglos de bits. Análogamente, cada paquete es utilizado en tareas que permiten que el servidor trabaje tanto con la clase cliente cuanto con la de mensajería; es así, mientras *java.io* tiene más herramientas orientadas a la comunicación con el cliente, *java.util* posee estructuras que facilitan la gestión de la información.

Como ya se mencionó anteriormente, las operaciones PUT y GET del protocolo OBEX son fundamentales en la aplicación de mensajería. En la figura 5.14 se observa un esquema donde se muestra la relación existente entre PUT y el envío de mensajes por parte de cualquier cliente y GET para obtenerlos del servidor. La implementación de tales operaciones se dará mediante la creación de funciones o métodos dentro de la clase servidor; onPut() y onGet(), respectivamente.

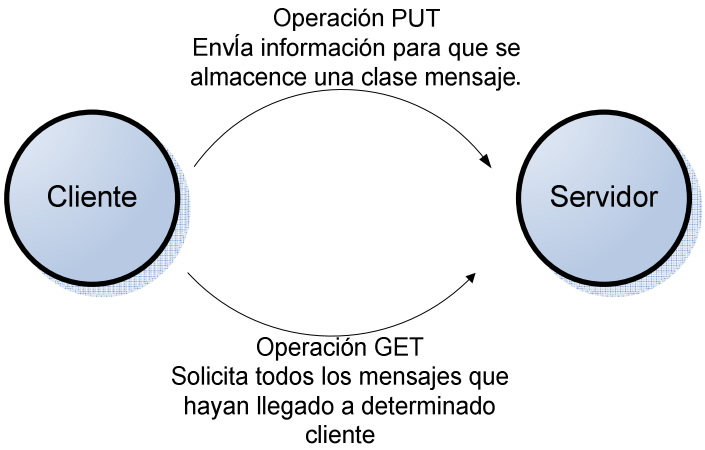


Figura 5.14.Esquema de trabajo de operaciones PUT y GET en la aplicación de mensajería.

La principal característica del método onPut() es la de ser pública debido a que debe ser accedida por la clase del cliente y también interactúa con la clase appServidor. Por otro lado, requiere de un parámetro enviado desde el cliente que contenga las cabeceras con la información del mensaje y destinatario. Esta información en conjunto se conoce como operación y se la debe declarar como tal en la clase del cliente, lo cual se mostrará más adelante. Una vez recibida la operación, el procesamiento que se debe realizar es bastante sencillo debido al uso de las cabeceras OBEX, por lo cual únicamente se requiere almacenar la información en una lista, tabla o matriz para que pueda ser enviada por la operación onGet(). En este caso se utilizará un vector de manejo de listas incluido dentro del paquete *java.util*, cuya declaración se puede observar en la tabla 5.3.

El flujo de manejo de la información se puede observar en la figura 5.15, donde se observa que una vez recibidas las cabeceras por parte del cliente únicamente se requiere realizar la validación de la información y el ingreso de la misma en el vector previamente creado. Los métodos utilizados son nativos del paquete javax.obex.

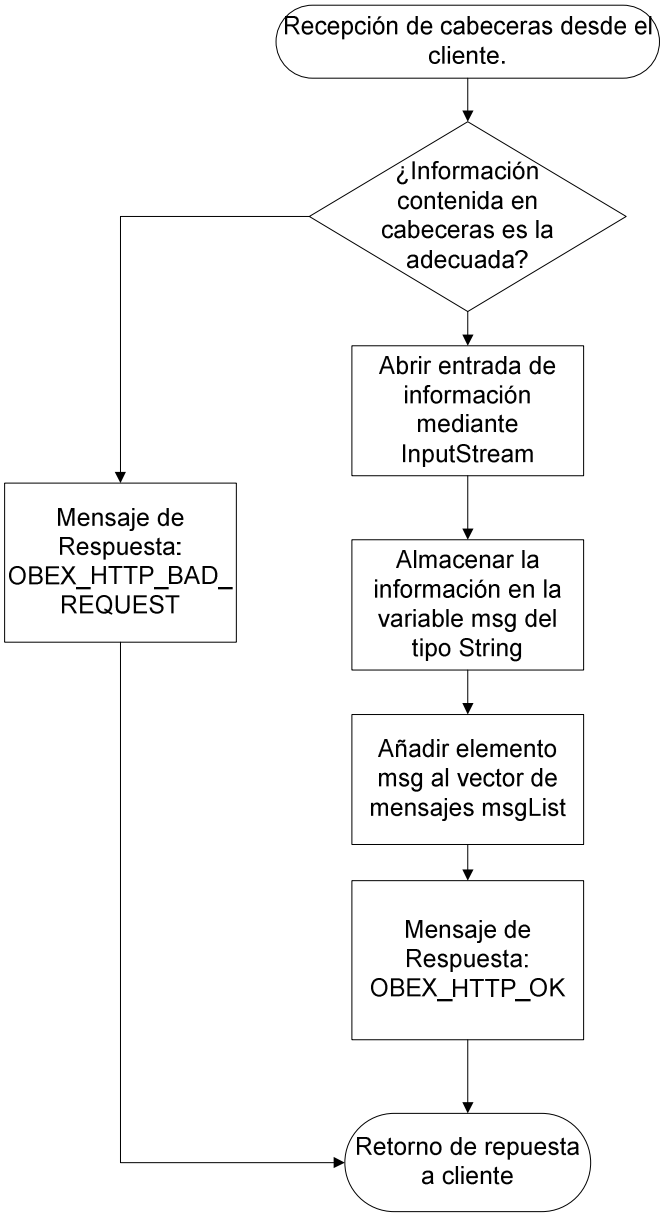


Figura 5.15.Flujo de manejo de información en la operación onPut de la clase servidor.

Como se observa en la figura 5.15, la variable de manejo del mensaje se denomina msg y es del tipo *StringBuffer,* cuya declaración se puede observar en la tabla 5.3. Por otro lado, el vector creado para almacenar cada uno de los mensajes se denomina msgList y para indexar información se utiliza el método *addElement*. En la figura 5.16 se muestra un ejemplo de código de declaración de la variable de manejo de los mensajes y del vector que los contiene.

```
//Recibir las cabeceras enviadas por el Cliente.
HeaderSet headers =op.getReceivedHeaders();
//Entrada de datos provinientes del cliente.
length=input.read(data);
//Crear un flujo para la entrada de datos del cliente
StringBuffer msg=new StringBuffer();
//Ingreso de información del cliente
msg.append(new String(data,0,length));
//Creación de objeto mensaje en el vector.
msgList.addElement(new Mensaje(name,msg.toString()));
//Respuesta de proceso exitoso al cliente.
return ResponseCodes.OBEX_HTTP_OK;
```

Figura 5.16. Código base para la creación del mensaje en la aplicación de mensajería.

Tabla 5.3. Descripción de objetos referenciados en la operación PUT del servidor de mensajería.

Declaración de Objeto	Descripción
Public Vector “nombre_variable”	Permite el manejo de un vector que maneja objetos de cualquier clase de forma de lista indexada.
StringBuffer “nombre_variable” =new StringBuffer();	Elemento de almacenamiento de múltiples cadenas de texto mediante ingreso de datos por flujos (streams).
HeaderSet “nombre_variable”	Permite crear un objeto para el manejo de cabeceras de protocolo OBEX
InputStream “nombre_variable”	Permite la entrada de flujo de información hacia la clase.

La interacción entre el cliente el servidor se dará por mensajes enviados como cabeceras debido a la utilización del protocolo OBEX [67], como se puede observar en la figura 5.9 mediante el envío del código de respuesta OBEX_HTTP_OK. Este método de comunicación permite conocer específicamente para nuestra aplicación con certeza cuáles son las acciones que se están llevando a cabo durante la sesión facilitando su comprensión y reduciendo la posibilidad de errores al crear un entorno limitado de respuestas válidas. Por ello, cuando un mensaje llega a la clase servidor o cliente y no está totalmente claro, se debe pedir la retransmisión del mismo. De

igual manera, la información es transportada en forma de paquetes lo cual reduce el tiempo de transmisión de cada uno y permite que la retransmisión sea más sencilla en caso de existir errores. No todas las respuestas OBEX se enviadas como cabeceras en la aplicación de mensajería sino únicamente las que se muestran en la tabla 5.4, debido a que muchas no tienen relación con el medio que se está utilizando o con los errores que se pudiesen producir.

Tabla 5.4 Posibles respuestas OBEX del servidor en la aplicación de mensajería

Respuesta	Descripción
OBEX_HTTP_OK	Indica que la petición se ha realizado correctamente o que una tarea ha finalizado
OBEX_HTTP_INTERNAL_ERROR	Indica que existe un error del protocolo relacionado con el dispositivo o con la BCC.
OBEX_HTTP_NOT_FOUND	Indica que la petición realizada no encontró respuesta o que el usuario no posee mensajes
OBEX_HTTP_BAD_REQUEST	Indica que la petición no se realizó correctamente.

La implementación de la operación GET en el método onGet() tiene las mismas restricciones de información que fueron consideradas para onPut(), por lo que es necesario validar la información proveniente desde el cliente. Sin embargo, en este método será necesario realizar una búsqueda del mensaje que desea el cliente en el vector creado anteriormente para posteriormente enviarlo mediante cabeceras. En la figura 5.17 se muestra el flujo de manejo de la información proveniente del cliente para este caso.

Para realizar la búsqueda del mensaje solicitado por el cliente, es necesario que el mismo envíe la información de su identificación en forma cabeceras para que una vez recibido por el servidor se pueda obtenerlo mediante el método *NAME* de la clase *HeaderSet.* Finalmente, la exploración del mensaje dentro del vector se lo puede hacer con cualquier algoritmo básico, en este caso se incluyó un condicional dentro de un bucle que recorre el contenido de todo el vector, que cumple con la sintaxis establecida en *JAVA* y *J2ME*. Los abstractos más importantes del código anteriormente mencionado se muestran en la figura 5.18.

Cuando se he encontrado que el destinario posee mensajes que le deben ser enviados, es necesario crear una clase de tipo *Mensaje* temporalmente para almacenar toda la información que deba ser enviada hasta que el cliente la reciba. Esto se lo hace mediante el método *setHeader()*, que permite definir las cabeceras a enviarse y cuya sintaxis se encuentra en la figura 5.19. Finalmente, para enviar el mensaje es necesario abrir el flujo con una variable de salida mediante el método *openOutputStream()* y enviar la información mediante el método *Write()* que contendrá como argumentos el mensaje solicitado por el cliente, como se observa en la figura 5.19.

Cabe destacar, que la operación GET permitirá que el cliente obtenga todos los mensajes que le hayan sido enviados durante el tiempo en el cual el servidor estuvo presente. Lo cual hace que los usuarios puedan recibir mensajes aún cuando no estén registrados en el servidor. Por tanto, resulta evidente la responsabilidad del servidor de crear, ordenar y guardar de la mejor manera los mensajes que haya recibido, para poder enviarlos posteriormente a los clientes.

```
 //Obtener el nombre de usuario de quien solocita el mensaje
HeaderSet header =op.getReceivedHeaders();
String name=(String)header.getHeader(HeaderSet.NAME);

 //Encotrar el mensaje para el usuario especificado
int length =msgList.size();
Mensaje temp=null;
for(int i=0;i<length;i++){
        temp=(Mensaje)msgList.elementAt(i);
        if(temp.oUserName().equals(name)){
        break;
        }}
```

Figura 5.18. Código base para la búsqueda de mensajes en el servidor a partir del nombre del cliente.

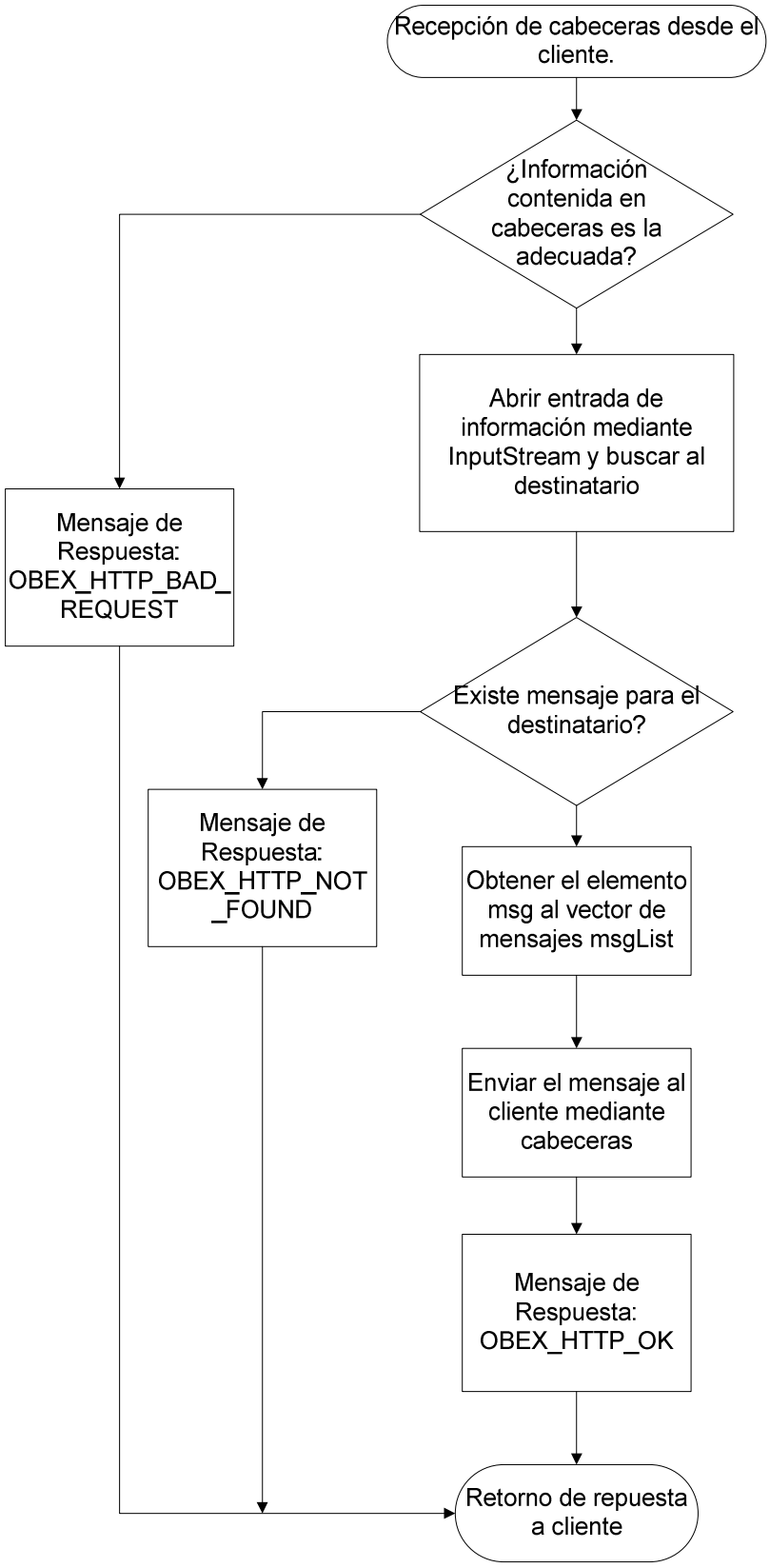


Figura 5.17.Flujo de manejo de información en la operación onGet de la clase servidor.

```
//enviar el mensaje al cliente (temp contiene mensaje para cliente)
    OutputStream out=op.openOutputStream();
    out.write(temp.oMensaje().getBytes());

//Quitar el element enviado del vector de mensajes.
    msgList.removeElement(temp);

//Cerrar flujos de entrada (out) y salida (op)
    out.close();
    op.close();
```

Figura 5.19. Código base para el envío de información al usuario una vez encontrado, mediante la utilización de flujos de salida (*Output Streams*)

3.1.2 Entorno gráfico del Servidor

Una vez establecidos los métodos onPut y On Get dentro de la clase Servidor, resulta necesario resaltar el código más relevante de la clase appServidor que permite interactuar con el usuario y proporciona la interfaz gráfica de la aplicación. A diferencia de la clase Servidor, appServidor se enfocará en crear la interfaz usuario del aplicativo Servidor y de controlar las conexiones realizadas con los diferentes clientes. Por ello, esta clase no incluye algoritmos de bucles o validaciones sino una estructura lineal que se apega totalmente a la presentada en la figura 5.2 sobre la composición básica de un MIDlet.

La clase appServidor deberá implementar el método *Runnable* que le permitirá manejar las peticiones de los clientes y *CommandListener* para poder controlar los eventos producidos por los botones de comando que aparecen en pantalla. Por otro lado, se deberá controlar el acceso al medio para todas las conexiones Bluetooth a realizarse mediante el paquete javax.bluetooth, cuya funcionalidad se muestra en la tabla 5.4

Tabla 5.4 En la tabla se muestra las cabeceras incluidas dentro de la clase servidor.

Paquete	Descripción
Java.util	Contiene utilitarios para el trabajo sobre datos (clases de versiones anteriores, internacionalización, fecha y hora, manejo de arreglos de bits).
Java.io	Permite la entrada y salida de datos mediante flujos (*streams)*
javax.microedition.io	Provee un mecanismo más simplificado de ingreso y salida de información mediante conectores (connector)
javax.microedition.midlet	Fundamental para la creación de una clase MIDlet.
javax.microedition.lcdui	Permite crear el entorno visual para el MIDlet.
javax.bluetooth.	Permite manejar las operaciones de acceso y control a Bluetooth dentro del dispositivo.

El entorno gráfico del servidor se mantendrá bastante simple debido a que no se requiere un alto nivel de interacción con el usuario. Por otro lado, de esta manera se aligera el procesamiento del dispositivo móvil y se reduce la posibilidad de errores por incompatibilidades de las pantallas en los dispositivos móviles. De esta manera, se creará un formulario sencillo donde únicamente se mostrarán los mensajes sobre el estado del servidor y un botón de comando para la salida de la aplicación; todo bajo un solo hilo o *thread* de ejecución como se muestra en la figura 5.20

```
principal=new Form("Servidor de mensajes en OBEX");
principal.addCommand(new Command("SALIR",Command.EXIT,1));
principal.setCommandListener(this);
new Thread(this).start();
```

Figura 5.20. Creación de la interfaz de usuario de la clase appServidor

Una vez creada la interfaz es necesario que sea mostrada al usuario como la principal. Para ello se deben utilizar los métodos getDisplay y startApp, para la abstracción de la pantalla del dispositivo dentro de una variable y la presentación del formulario respectivamente. En ambos casos el puntero reservado *this* es de gran ayuda ya que estará apuntando a la aplicación en ejecución, según lo mostrado en la

figura 5.20 por la creación de un *thread* específico. En la figura 5.21 se muestra el código utilizado para la presentación del formulario en pantalla.

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

 public void startApp() throws MIDletStateChangeException {
    Display currentDisplay=Display.getDisplay(this);
    currentDisplay.setCurrent(principal);

 }
```

Figura 5.21. Presentación del formulario principal de appServidor en pantalla

La clase appServidor además de mostrar el entorno gráfico debe comunicarse con la clase cliente. Para tal fin, ambas clases deben establecer un método de intercambio de mensajes coordinados para evitar errores dentro de la comunicación y minimizar el gasto innecesario de ancho de banda. Para ello OBEX ofrece un método de señalización basado en notificaciones de sesión (*SessionNotifier*) las cuales pueden ser usadas para el control de los flujos de entrada y salida de información (*streams)* manejados por la clase Servidor. En la figura 5.22 se muestra un esquema de funcionamiento de los notificadores dentro de la aplicación de mensajería, donde se puede observar que mientras el notificador no sea enviado, los canales para el envío o recepción de mensajes no serán abiertos.

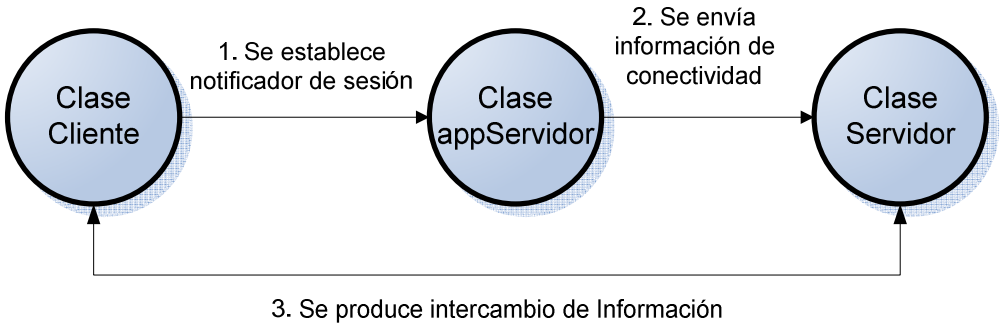


Figura 5.22. Esquema de funcionamiento de notificadores en aplicación de Mensajería.

Para iniciar la comunicación es necesario establecer la cadena de conexión, para ello se utilizará un UUID válido generado por el método mostrado en el capítulo anterior para realizar la conexión con el cliente. De manera indirecta la utilización del UUID permite la identificación del servicio y garantiza que ningún otro aplicativo podrá usarlo, proveyendo un primer nivel de seguridad. El UUID deberá implementarse como una dirección para acceder a los servicios precedida por la palabra *btgoep* relacionada con la tecnología Bluetooth dentro de una variable tipo cadena de caracteres o *string,* como se muestra en la figura 5.23.

```
CONNECTION_STRING= "btgoep://localhost:750ef04247a54693b6384708eb87ec5e"
```

Figura 5.23. Cadena utilizada para la comunicación de OBEX sobre Bluetooth.

Únicamente resta el manejo de de la tecnología Bluetooth dentro del dispositivo móvil para completar el código de appServidor. Básicamente, la clase deberá encargarse de que el servidor sea visible para todos los clientes y de aceptar las conexiones de los mismos. Para ello simplemente se utilizan los métodos *getLocalDevice* y *setDiscoverable* la clase Local para tomar control sobre el dispositivo Bluetooth e indicar el estado de descubrimiento general (GIAC), respectivamente. Todos estos procesos deberán manejarse dentro de operaciones *try-catch* debido a que implican ingreso o salida de datos y acceso a la BCC que podrían producir excepciones. En la figura 5.24 se muestra un abstracto del código más importante de manejo de acceso al medio Bluetooth dentro del método principal *run.*

```
public void run(){
  //Crea un objeto de conexion al servidor y hace que el dispositivo local pueda ser descubierto.
  try{
  LocalDevice local=LocalDevice.getLocalDevice();
  local.setDiscoverable(DiscoveryAgent.GIAC);
  notificadorSesion = (SessionNotifier)Connector.open(CONNECTION_STRING);
  }catch(IOException e){
    principal.append("Imposible de iniciar el servidor (IOException:"+ e.getMessage()+ ")");
    return;
  }
  principal.append("Servidor Iniciado");
  while(true){
    //Aceptar la conexion desde un cliente y notificar a la clase Servidor
    Servidor server=new Servidor(msgList);
    try{
      notificadorSesion.acceptAndOpen(server);
.....
```

Figura 5.24. Abstracto de acceso a medio Bluetooth dentro de la clase appServidor.

3.2 Aplicación para los clientes en Java

De igual manera que se hizo para el servidor, se debe identificar cuáles son las tareas que debe cumplir el cliente dentro del entorno de la aplicación de mensajería. Como premisa se tendrá la de iniciar la sesión con el servidor, ya sea para el envío cuanto para la recepción de mensajes; sin embargo, se tendrá otras tareas derivadas de su funcionamiento las cuales se indican en la tabla 5.5

Tabla 5.5 Tareas y funciones del cliente de mensajería.

Tareas del Cliente	1. Debe iniciar la comunicación con el servidor. 2. Debe enviar la información de registro necesaria para ser identificado por el servidor. 3. Debe crear el mensaje a ser enviado indicando el destinatario. 4. Debe solicitar los mensajes una vez conectado al servidor y enviar comandos coherentes al servidor para obtener servicios. 5. Debe terminar la sesión cuando sea requerido.

Existen varias diferencias entre la clase del cliente y la del servidor. La primera es su unicidad debido a que no se la dividirá puesto que no requiere interactuar con más de una clase y su carga de trabajo está totalmente relacionada con la interacción con el usuario. De ahí que la segunda diferencia es una interfaz de usuario más compleja que facilite y valide el ingreso de datos para su posterior envío. Finalmente, esta clase requerirá un uso más estricto del medio Bluetooth debido a que debe empaquetar y enviar la información, así como iniciar las sesiones OBEX necesarias. Es así que si para la clase servidor se requería tener un estricto control sobre los procedimientos internos de la aplicación para el cliente el objetivo será reducir al mínimo la posibilidad de errores que se pudiesen dar en la interacción con el usuario y su correcto envío por el medio Bluetooth, como se observa en la figura 5.25.

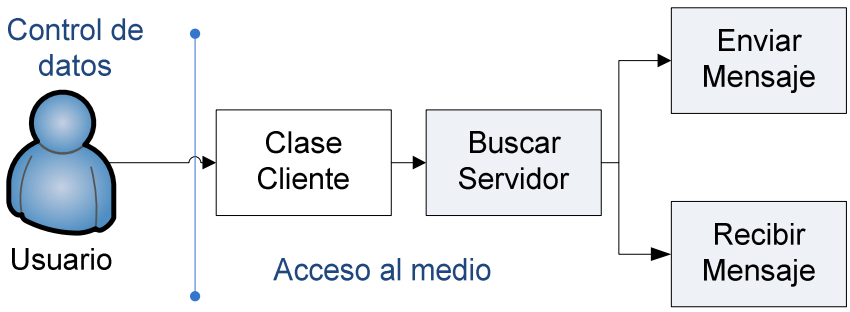


Figura 5.25. Esquema de principales objetivos de control de la clase cliente

En la implementación, la clase cliente llevará el nombre de appCliente. En la misma se incluyen los mismos paquetes que para appServidor que se muestran en la tabla 5.4 debido a que se realizarán tareas de acceso al medio similares o complementarias a las realizadas previamente. Cabe mencionar que las tareas de diseño y presentación en pantalla son similares a las realizadas para appServidor con el evidentemente aumento de elementos que se ha mencionado previamente.

Cuando un usuario ingrese a la aplicación de mensajería como cliente, lo primero que deberá hacer es registrarse mediante un nombre de usuario, pues será la única manera de identificarlo. Para ello, se deberá crear una interfaz que permita el ingreso y validación de la información ingresada. Una vez dentro, el usuario tendrá la opción de enviar un mensaje o recibirlo; si elige el enviarlo, la pantalla deberá mostrar dos campos de texto para ingresar el destinatario y el mensaje. Por otro lado, si desea recibirlo, el texto del mensaje será mostrado en pantalla. Cualquiera sea el caso, se deberá incluir un botón de comando para regresar al menú de selección de opciones.

El diseño de formulario de ingreso será bastante simple, únicamente se debe considerar la validación del campo de nombre de usuario para evitar que se lo deje en blanco. Además, se deberán incluir dos botones de comando, uno para registrarse en la aplicación y otro para salir de la misma. Cabe mencionar que inmediatamente después de ingresar el nombre de usuario y registrarse, la clase empezará a realizar la búsqueda del servidor de mensajes por lo que tomará aproximadamente unos diez segundos, en los cuales se deberá notificar al usuario sobre tal proceso para evitar que la aplicación tenga la apariencia de que se ha congelado y que no responde a los comandos. En la implementación este formulario se llamará ingreso y el diseño anteriormente discutido se puede observar en la figura 5.26. La validación se puede observar en la figura 5.27, donde únicamente se requiere de un condicional para verificar que el objeto no esté vacío y se desplegará un objeto gráfico para alertar al usuario. Los lugares de implementación varían, el

formulario debe hacerlo dentro del método *startApp* que se invoca cuando se inicia el MIDlet y la validación dentro de *commandAction* que es lanzado una vez que se ejecuta un botón de comando.

```
Form ingreso = new Form("Login");
      ingreso.append(new TextField("User Name", "", 10, TextField.ANY));
      ingreso.addCommand(new Command("Login", Command.OK, 1));
      ingreso.addCommand(new Command("Exit", Command.EXIT, 2));
      ingreso.setCommandListener(this);
      pantalla.setCurrent(ingreso);
…..
```

Figura 5.26. Diseño del formulario de entrada para la clase cliente.

```
 //Obtener el nombre de usuario del formulario
TextField userTextField = (TextField) ((Form) d).get(0);
if (userTextField.getString().trim().equals("")) {
Alert error = new Alert("Error", "Por favor introduzca un nombre antes de seguir", null, AlertType.ERROR);
     error.setTimeout(Alert.FOREVER);
      pantalla.setCurrent(error);
       return;
}
 userName = userTextField.getString();
 new Thread(this).start();
……..
```

Figura 5.27. Validación del campo de texto de nombre de usuario para la clase cliente.

El proceso más complejo y crítico dentro de la clase cliente es la de la búsqueda del servidor, puesto que la tecnología Bluetooth no garantiza totalmente que un elemento pueda ser encontrado durante una búsqueda [68]. De hecho, este es uno de los elementos más importantes a analizar en la aplicación, ya que de acuerdo al fabricante del dispositivo y a la implementación del BCC se podría inclusive negar el acceso a este tipo de conexión, ya sea en el cliente cuanto en el servidor. Este proceso será indiferente para el usuario quién únicamente verá un mensaje de que se ha encontrado el servidor o de que la operación fue fallida. Como inicio, se debe establecer la cadena de caracteres con la dirección UUID establecida

en el servidor para poder encontrar el servicio y tomar el control del dispositivo local mediante el método *getLocalDevice* de la clase *LocalDevice*, como se utilizó en el servidor. Posteriormente, se debe definir un agente para el descubrimiento de servicios del tipo *DiscoveryAgent*, que permitirá seleccionar el servicio con la UUID definida mediante el método *selectService*. Si la cadena de conexión es nula o se produce un error durante el proceso se debe considerar como fallido. En la figura 5.28 se muestra la implementación de la búsqueda del servidor de mensajes, donde el nombre del formulario es búsqueda y permitirá observar la ejecución de la selección de servicio y establecimiento de sesión, que se observará más adelante.

```
//Crea una ventana para mantener al usuario informado en el progreso
      Form busqueda = new Form("Conectando...");
      busqueda.append("Buscando por el Servidor de mensajes...");
      busqueda.addCommand(new Command("Exit", Command.EXIT, 1));
      busqueda.setCommandListener(this);
      pantalla.setCurrent(busqueda);
 //Busca por un servidor
         LocalDevice local = LocalDevice.getLocalDevice();
         DiscoveryAgent agent = local.getDiscoveryAgent();
         String connString = agent.selectService(new UUID("750ef04247a54693b6384708eb87ec5e", false),
ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
         if (connString == null) {
            busqueda.append("Imposible de encontrar el servidor!!");
            displayError("Error", "Imposible de encontrar un servidor");
            return;
         }
         busqueda.append("Done\n");
.....
```

Figura 5.28. Búsqueda del servidor de mensajes en la aplicación de mensajería

De la figura 5.28 se puede observar que el objeto *connString* permite la conexión entre la clase cliente y servidor, pero también requiere de varios parámetros que son enviados mediante el método *selectService*. Entre los más relevantes están la de no autenticar la conexión ni tampoco usar encriptación para la comunicación; esto debido a que el objetivo de la aplicación de mensajería es

ejecutarse sobre la mayor cantidad de dispositivos móviles, y aquellos con sistema operativo más simple podrían no ser compatibles con tales características. La utilización de *connString* se hará una vez encontrado el servidor para dar el inicio de la sesión enviándolo como parámetro a un objeto *ClientSession* mediante el método *connect.* Este procedimiento marca el inicio de la sesión OBEX entre el cliente y el servidor, por lo que en adelante será posible intercambiar información mediante el envío y recepción de cabeceras.

```
        //Estebleciendo una capa de transporte al servidor
         sesionCliente = (ClientSession) Connector.open(connString);
         busqueda.append("Done\n");
         busqueda.append("Estableciendo una conexion OBEX..");
         HeaderSet header = sesionCliente.connect(null);
         if (header.getResponseCode() != ResponseCodes.OBEX_HTTP_OK) {
            busqueda.append("La conexion fue rechazada");
            displayError("Rechazada", "La conexion ha sido rechazada por el (0)" + "servidor.(0x" +
Integer.toHexString(header.getResponseCode()) + ")");
            sesionCliente.close();
            sesionCliente = null;
            return;
         }
……
```

Figura 5.29. Establecimiento de la sesión OBEX entre el cliente y el servidor de mensajería

En la figura 5.29 se observa el inicio de la sesión OBEX y la creación de un conjunto de cabeceras para el intercambio de información mediante el objeto *HeaderSet.* Para finalizar con establecimiento de la conexión se utiliza el método *connect* sin cabeceras y la única respuesta válida es OBEX_HTTP_OK; cualquier otra respuesta diferente indicará que la conexión no puede establecerse correctamente y debe cancelarse. Por defecto, el método *connect* nunca devolverá un elemento vacío o null.

Todo el procedimiento de descubrimiento del servidor y establecimiento de una sesión OBEX debe incluirse dentro de un procedimiento *try-catch* por la cantidad de

excepciones que pueden producirse. Las causas de las excepciones son diversas y van desde elementos incompatibles con la BCC hasta métodos de ingreso de información por la implementación de J2ME en el dispositivo. Sea cual sea la excepción debe ser considerada y si su causa está relacionada con el acceso al medio Bluetooth o al protocolo OBEX debe terminarse con la sesión.

Una vez establecida la conexión con el servidor, el usuario podrá empezar a interactuar con el mismo para enviar o recibir mensajes. Para esto, se deberá crear una interfaz que funcione como pantalla principal que nos permita seleccionar las acciones a tomar y a la que se vuelva después de haberlas terminado. Para esto, se creará una función o método específico para facilitar el proceso de invocarlo en cualquier parte cuando sea necesario. Su nombre será opcionesUsuario y no necesitará que se le envíe parámetros para su ejecución. Dentro del método se definirá el diseño del formulario que incluirá una lista exclusiva para enviar o recibir el mensaje. Los botones de comando también se incluyen para afirmar la selección o finalizar la aplicación. En la figura 5.30 se muestra el código más relevante en el diseño del formulario de selección.

```
    private void opcionesUsuario(Alert dialog) {
        List requestForm = new List("Mensaje", List.EXCLUSIVE);
        requestForm.append("Enviar un mensaje", null);
        requestForm.append("Recibir un mensaje", null);
        requestForm.addCommand(new Command("Ok", Command.SCREEN, 1));
        requestForm.addCommand(new Command("Exit", Command.EXIT, 1));
..... }
```

Figura 5.30. Diseño del formulario de selección de envío o recepción de mensajes para la clase cliente.

Hasta el momento al usuario se le presentarán tres pantallas, dentro de dos de ellas tendrá que interactuar mientras que la tercera mostrará referencias sobre el proceso de establecimiento de conexión. En la figura 5.31 se muestran las tres ventanas en la secuencia en que aparecen, según el simulador, justo antes del

momento de seleccionar si se envía o recibe un mensaje y cuando el proceso de conexión ha sido exitoso.

Figura 5.31.Pantallas de registro y selección de acción en la aplicación del cliente de mensajería.

Los procesos de envío y recepción de mensajes para el cliente están basados en la lógica presentada para el servidor ligada con las operaciones PUT y GET de OBEX. Durante la implementación, se descubrió que denominar a un método de la misma manera producía errores de ambigüedad a pesar de que estuviesen en clases distintas. Por ello, el método análogo a PUT se denominará enviarMensaje mientras que el de GET se denomina obtenerMensajes.

El formulario para el envío de mensajes únicamente requiere de dos cuadros de texto donde introducir la información sobre el destinatario y el mensaje en sí. El diseño es similar al realizado anteriormente y en la implementación el formulario se denomina *sendForm* como se muestra en la figura 5.32. El cambio de pantalla se dará inmediatamente el usuario haya presionado el botón de comando OK del menú de selección principal por lo que el código debe estar dentro del método commandAction.

```
Form sendForm = new Form("Envio");
sendForm.append(new TextField("Usuario", null, 10, TextField.ANY));
sendForm.append(new TextField("Mensaje", null, 250, TextField.ANY));
sendForm.addCommand(new Command("Exit", Command.EXIT, 1));
sendForm.addCommand(new Command("Send", Command.ITEM, 1));
sendForm.setCommandListener(this);
pantalla.setCurrent(sendForm);
…… }
```

Figura 5.32. Diseño del formulario para el envío de mensajes para la clase cliente.

Al método enviarMensaje se lo debe invocar enviándole como parámetros tanto la cadena de texto que contiene el mensaje cuanto la del mensaje para que únicamente se realice el envío de información, de manera similar al servidor. Para ello, se deberá crear un conjunto de cabeceras que contengan el nombre y el mensaje de manera diferenciada para que el servidor pueda identificarlos al recibirlos. Las cabeceras y la información son enviadas mediante flujos de salida (*OutputStreams)*. La simplicidad en la comunicación se da gracias a la utilización del protocolo OBEX; sin embargo, será necesario que se verifique que la respuesta del servidor una vez finalizada la transmisión de la información sea únicamente OBEX_HTTP_OK. El envío de mensaje debe ser incluido dentro de un proceso *try-catch* para alertar en cualquier momento al usuario que se ha producido una excepción y conocer el problema que se tiene. Finalmente, en la figura 5.33 se puede observar el código más relevante del método.

```
private void enviarMensaje(String nombre, String mensaje) {
Operation op = null;
byte[] data = mensaje.getBytes();
Alert error = null;
        //Establecer las cabeceras para el envío de la información
        HeaderSet header = sesionCliente.createHeaderSet();
        header.setHeader(HeaderSet.NAME, nombre);
        header.setHeader(HeaderSet.LENGTH, new Long(data.length));
        op = sesionCliente.put(header);
        OutputStream out = null;
        out=op.openOutputStream();
        //Enviar el mensaje al servidor via OutputStream
         out.write(data);

        //Verificar el codigo de respuesta
        int code = op.getResponseCode();
        if (code != ResponseCodes.OBEX_HTTP_OK) {
          //Presentar error en Pantalla
        }
      out.close();
….
```

Figura 5.33. Código más relevante de la implementación del método enviarMensaje en la clase cliente.

La obtención de mensajes mediante el método obtenerMensajes no requiere que se le envíe ningún parámetro sino que utiliza el nombre de usuario ingresado para identificarse con el servidor y pedir los mensajes que tenga como destinatario. El envío de cabeceras se hace mediante el método *get* de un conjunto de cabeceras previamente creado. Una vez recibida la información por parte del servidor es necesario validarla para identificar si la respuesta contiene un mensaje, nos indica que no existen mensajes que tengan al usuario como destinatario o simplemente un error; para ello se utiliza el método *getResponseCode* de la operación establecida. En la figura 5.34 se muestra la implementación de las diferentes etapas del método de obtención de mensajes.

```
//Especificar el nombre de usuario para el que recibe el mensaje
HeaderSet header = sesionCliente.createHeaderSet();
header.setHeader(HeaderSet.NAME, userName);
Operation op = sesionCliente.get(header);

int code = op.getResponseCode();
switch (code) {

//Si todo el mensaje ha sido leido imprimir el mensaje
case ResponseCodes.OBEX_HTTP_OK:
msg = new Alert("Mensaje", buf.toString(), null, AlertType.INFO);
break;
//No existen mensajes para este usuario en el servidor
case ResponseCodes.OBEX_HTTP_NOT_FOUND:
msg = new Alert("Sin Mensajes", "Sin mensajes en el servidor", null, AlertType.INFO);
break;
//Un error ocurrio asi que imprimirlo
default:
msg = new Alert("Error", "Error durante la comunicacion: (" + Integer.toHexString(code) +
")", null, AlertType.INFO);
break;
        }
        op.close();
....
```

Figura 5.34. Código más relevante de la implementación del método obtenerMensajes en la clase cliente.

De la figura 5.34 resalta la utilización del método *bufftoString()*, utilizada para que el flujo de información recibida por *InputStream* sea convertida directamente a texto sin la utilización de ningún algoritmo adicional, como sería necesario en otros lenguajes de programación de más bajo nivel como C++. En este caso la conversión se simplifica y se reduce la posibilidad de error. Resulta evidente que el procedimiento de comunicación con el servidor debe incluirse dentro de un proceso *try-catch*, por la cantidad de excepciones que podrían producir.

Una vez realizada la revisión del código más relevante de la aplicación, podremos hacer el análisis de su comportamiento y desempeño sobre varios

dispositivos móviles. En la figura 5.35 se muestra un ejemplo de ejecución de la aplicación de mensajería utilizando el simulador, lo cual establece un marco referencial óptimo bajo el cual funcionará nuestra aplicación.

a. Envío de mensaje por parte de cliente 5550002 a 5550001

b. Recepción de mensaje por parte de cliente 5550001

Figura 5.35. Comportamiento de la aplicación de mensajería sobre el simulador.

4. Implementación del código en Netbeans

Existen varios entornos de programación para Java, de los cuales resaltan Netbeans y Eclipse como los más utilizados a nivel profesional. Este último es totalmente desarrollado por una comunidad de *software* libre y tiene distribuciones para varias plataformas, mientras que el primero es un aplicativo propietario de ORACLE-SUN aunque mantiene un grupo de desarrolladores independientes. Por ello, resulta que Netbeans es la opción predilecta para quienes se inician en Java y en este caso en específico se la ha utilizado como el entorno de desarrollo. Además

en la tabla 5.6 se muestra una ponderación de las principales razones por las cuales también destacó sobre Eclipse.

Tabla 5.5 Comparativa de entornos de programación Java (Puntuación sobre 5)

Característica	**NETBEANS** **(Sobre 5, más alto es mejor)**	**ECLIPSE** **(Sobre 5, más alto es mejor)**
Multiplataforma	5	5
Cantidad de paquetes específicos	5	5
Entorno de desarrollo para MIDlets	5	4
Soporte (en línea)	5	5
Entornos de simulación	4	4
Facilidad de uso	4	3
Total	28	26
Costo	Gratuito	Gratuito

La apertura gradual del código Java durante la primera década del siglo XXI produjo que el lenguaje se enriqueciese mediante una gran comunidad en línea de desarrolladores atraídos por las características del lenguaje y la apertura del mismo. Sin embargo, la adquisición por parte de ORACLE a SUN MICROSYSTEMS produjo un retraso de la liberación del código y el aumento de restricciones de arquitectura al momento de utilizar el mismo. Tal es así que hasta el primer trimestre de 2011 se mantuvo un litigio con GOOGLE debido a que la arquitectura de ANDROID viola las patentes adquiridas por la gigante estadounidense [69]. El tema no afecta de manera directa a los desarrolladores de aplicaciones pero si establece que el lenguaje de programación no es totalmente abierto y que en el futuro podrían esperarse cambios radicales que establezcan una cisma entre la programación de Java entre Netbeans y otras plataformas, sobre todo en el entorno de J2EE.

Netbeans ofrece un entorno dedicado para el desarrollo de aplicaciones móviles, el cual facilita su creación y la conexión con paquetes específicos. Su entorno gráfico es bastante simple y ofrece un sistema de asistencia sobre la utilización de clases y variables mientras se escribe el código, en la figura 5.36 se muestra el entorno de programación en Netbeans.

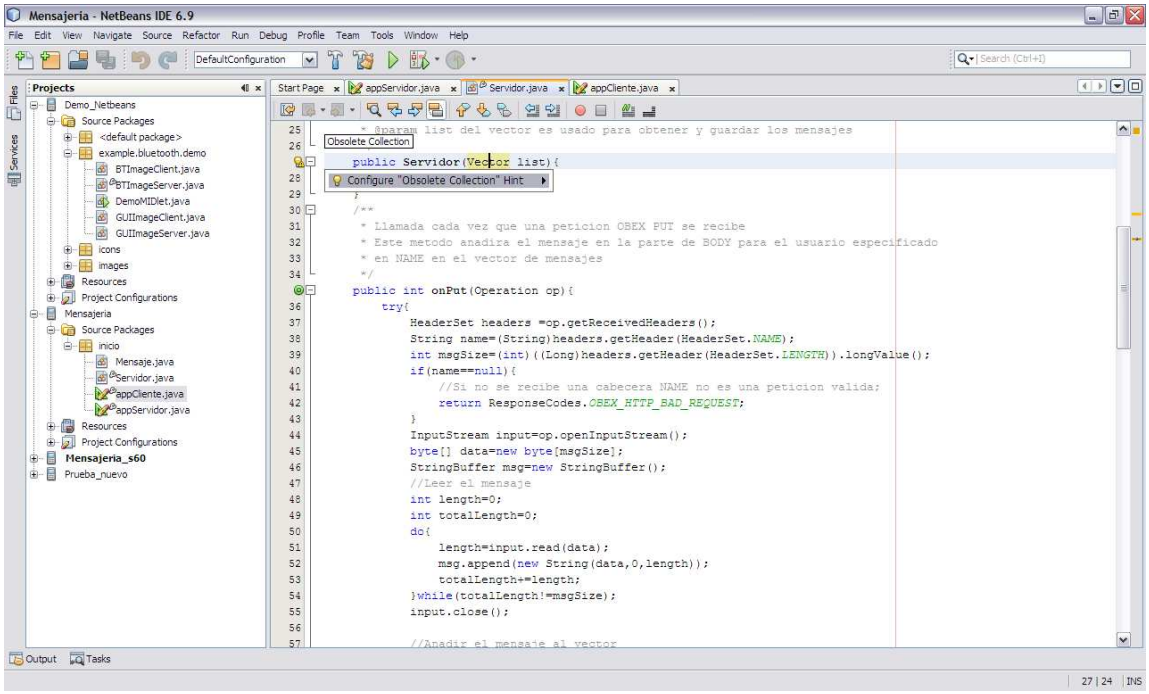


Figura 5.36. Entorno gráfico para la programación en Netbeans.

La creación de una aplicación representa la de un nuevo proyecto en Netbeans y este a su vez la del paquete que contendrá las clases. Un proyecto puede contener varios paquetes y estos a su vez diversos archivos como imágenes, clases, textos. Además, también incluye una sección donde almacena los recursos que solicita y otro para las configuraciones que las incluidas por defecto por Netbeans o específicas. Para crear el proyecto se recurre al menú *File* y posteriormente al submenú *New Project* mientras que para agregar archivos se utiliza el submenú *New File.* En la figura 5.37 se muestra la estructura de un paquete en Netbeans de nombre *Demo_Netbeans* donde se incluyen varios archivos.

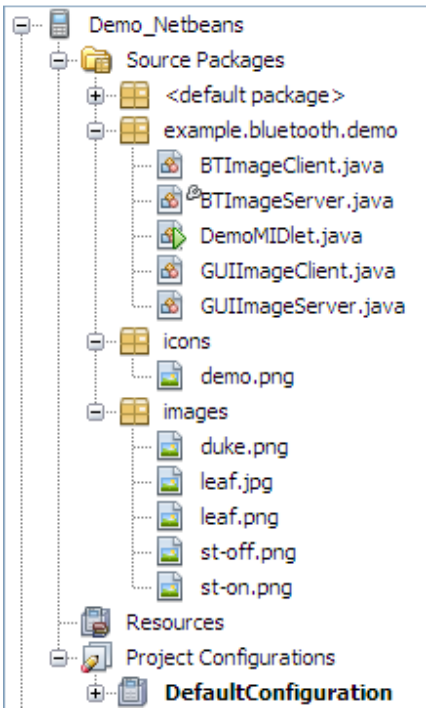


Figura 5.37. Estructura del paquete *Demo_Netbeans*

Al crear un proyecto se mostrará una ventana con varias categorías para seleccionar la tecnología Java que se desea utilizar, ya sea estándar como J2ME o propietaria como Groovy. Cada una incluye tipos predeterminados de proyectos, dependiendo de la aplicación que se desee crear. En este caso se seleccionará la categoría *Java ME* y como proyecto base *Mobile Application*, como se muestra en la figura 5.38.

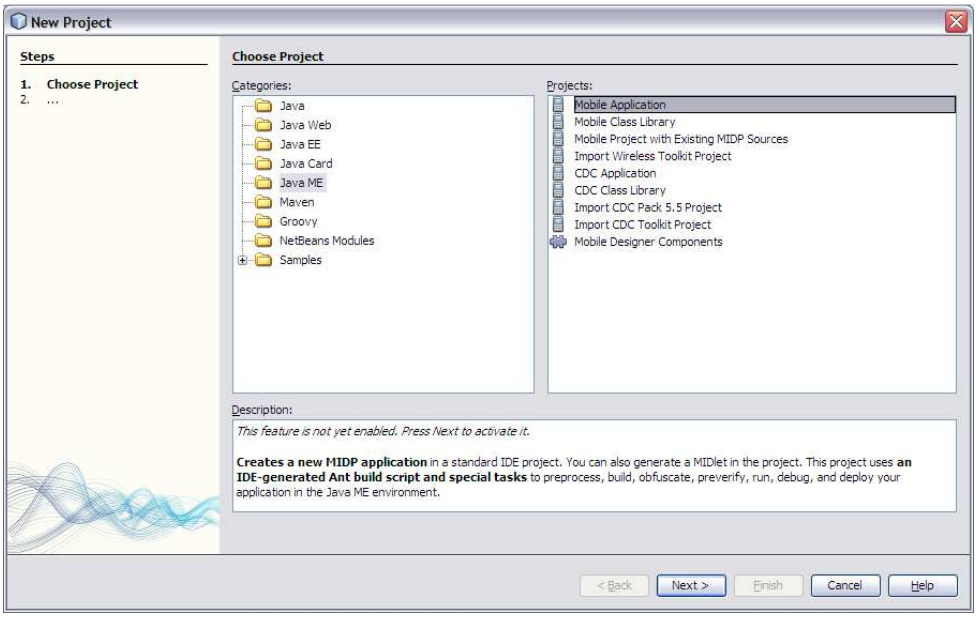


Figura 5.38. Selección de tecnología Java y proyecto base en Netbeans

Posteriormente, la creación requiere que se le dé un nombre, el cual también será el de la clase principal automáticamente y se establezca una configuración por defecto. En este caso, el proyecto se llamará Mensajería y utilizará la configuración incluida por Netbeans para dispositivos J2ME con CDLC, con el perfil MIDP 2.0 debido a que la versión 2.1 no ha sido ampliamente adoptada por dispositivos móviles todavía. En la figura 5.39 se muestra la selección de características del proyecto.

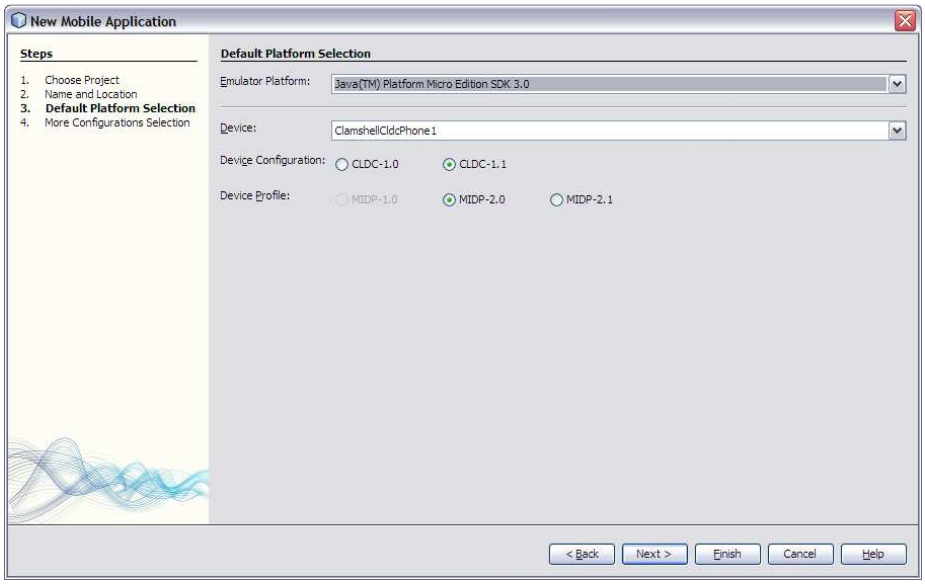


Figura 5.39. Selección de la plataforma y perfil del proyecto J2ME en Netbeans

De ser necesario, se podrá establecer configuraciones adicionales para la creación del proyecto, tal es el caso del desarrollo de aplicaciones para dispositivos Blackberry [70]. A partir de ahí, la creación de las clases para el cliente y servidor simplemente será mediante la inclusión de MIDlets, como se muestra en la figura 5.40 donde el archivo deberá llevar el mismo nombre de la clase para evitar problemas durante la compilación.

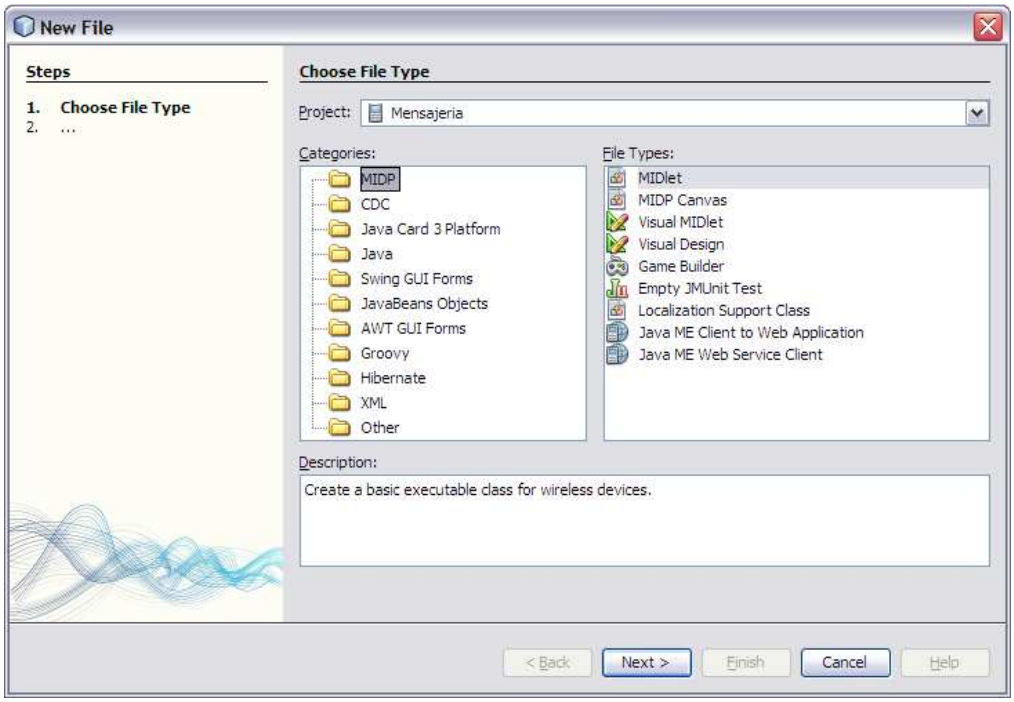


Figura 5.40. Adición de archivos MIDlet para la creación de clases en Netbeans.

De la figura 5.40 se puede observar que existen MIDlets visuales, que son aquellos que se muestran en pantalla. Para el caso de la aplicación de mensajería, únicamente las clases appCliente y appServidor deberán ser de este tipo puesto que las otras únicamente ejecutan tareas que no requieren interacción con el usuario. En la figura 5.41 se muestra la estructura del proyecto Mensajería.

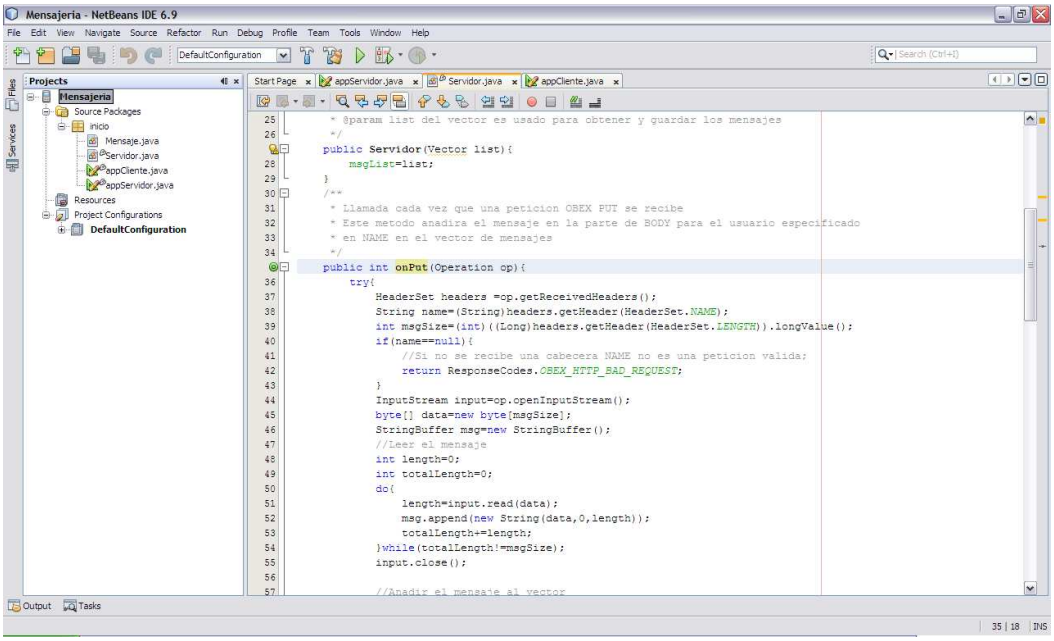


Figura 5.41. Estructura de proyecto de Mensajería en Netbeans.

5. Implementación en dispositivos móviles

Uno de los aspectos más importantes a considerar en el desempeño de la aplicación es su desenvolvimiento sobre dispositivos móviles existentes en el mercado. Para ello, fue necesario realizar un modesto estudio de la situación actual de los teléfonos móviles, a nivel global y más específicamente para Ecuador. Debido a que existen diferentes fabricantes, el análisis no está relacionado únicamente en la cantidad de celulares sino en sus características de *hardware* y *software* así como la apertura de sus plataformas a nuevas aplicaciones.

Los aspectos a evaluarse son la compatibilidad del lenguaje Java, entornos de desarrollo, situación sistema operativo, posición en el mercado y perspectivas de crecimiento. De igual manera, se ha establecido a Nokia, Motorola y Samsung como referentes para cada uno de los elementos a evaluarse debido a su posición preponderante y tamaño corporativo. Caso aparte, conforman Apple y RIM (Blackberry) ya que presentan dispositivos totalmente propietarios que marcan claramente una tendencia hacia la consolidación de sus marcas y no a tecnologías, por ello no fueron tomados en cuenta para este análisis.

5.1 Situación actual de Mercado

Indudablemente Nokia conserva su condición de mayor fabricante de dispositivos móviles a nivel mundial. Tal posición se ha visto amenazada por dos frentes, el primero por parte de empresas asiáticas como ZTE, Huawei o la misma Samsung en el segmento de gama baja mientras que por el otro frente tiene que enfrentar a los gigantes Apple, RIM (*Research In Motion)* y toda la nueva línea de celulares inteligentes que incluyen Android.

En la tabla 5.6 se muestra la participación en el mercado de varios fabricantes de dispositivos móviles. La información es resultado de un estudio realizado por Gartner, empresa especializada en análisis de mercado y tendencias [71]. En este se observa como Nokia se mantiene como líder teniendo como escolta a Samsung mientras que Motorola no ocupa un sitial entre los cinco primeros.

Tabla 5.6 Posición en el mercado de principales fabricantes de dispositivos móviles (Gartner a febrero de 2011)

Fabricante	**Ventas de 2010 en millones**	**Participación de mercado en 2010**
Nokia	461.3	28.9%
Samsung	281	17.6%
LG	114.1	7.1%
RIM	47.4	3%
Apple	46.5	2.9%
Sony Ericsson	41.8	2.6%
Motorola	38.5	2.4%
ZTE	28.7	1.8%
HTC	24.6	1.5%
Huawei	23.8	1.5%
Otros	488.5	30.6%

A nivel regional, según un informe de la empresa de estudios de mercado ADMOB [72], Nokia también es líder dentro de las cinco empresas consideradas

para el presente análisis como se muestra en la figura 5.42. En este caso el estudio muestra que existe una gran cantidad de dispositivos Iphone, sin embargo es necesario indicar que el mismo considera a dispositivos de gama media o alta y no de baja, donde Nokia se lleva una gran tajada del mercado. Samsung y Motorola no ocupan posiciones relevantes en la lista y su porción de mercado está por debajo del 10%.

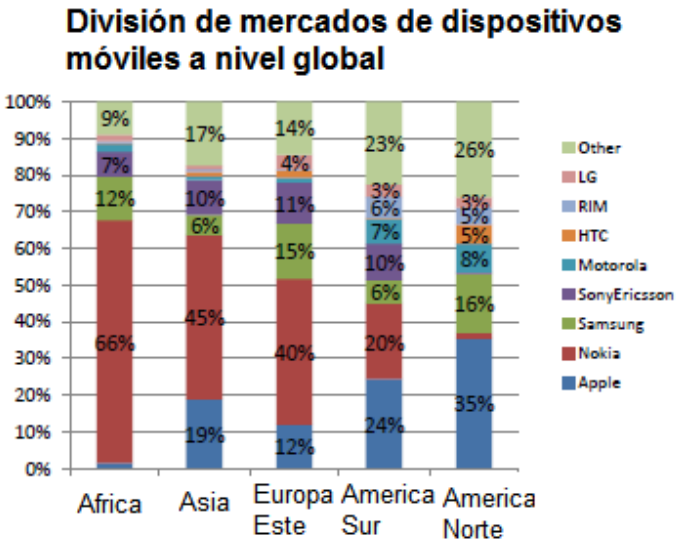


Figura 5.42 Situación actual del mercado global de principales fabricantes de dispositivos móviles (ADMOB a Octubre de 2010)

Es así que bajo este aspecto, el gran ganador es Nokia sobre sus otros dos contendores. Sin embargo, la empresa ha estado en un proceso de transición ante la imposibilidad de presentar un dispositivo móvil de gama alta que cumpla con las expectativas de un mercado cada vez más orientado hacia la transmisión de datos. Aun presentándose el caso, Nokia mantiene una gran porción de mercado y canales de distribución que lo mantendrán como líder por algún tiempo.

5.2 Sistema operativo.

Actualmente, existe gran cantidad de sistemas operativos para dispositivos móviles [73] pero únicamente entre cuatro se reparten más del 95% del mercado. Symbian, Android, RIM y iOS ocupan los cuatro primeros lugares en lo que respecta

a cantidad de dispositivos existentes con sus aplicaciones en el mercado, como se observa en la tabla 5.7 del estudio realizado por Gartner [71].

Tabla 5.7 Posición en el mercado de principales sistemas operativos de dispositivos móviles (Gartner a febrero de 2011)

Sistema Operativo	Ventas de 2010 en millones	Participación de mercado en 2010
Symbian	111.5	37.6%
Android	67.2	22.7%
RIM	47.4	16%
iOS	46.5	15.7%
Otros	11.4	3.8%

Symbian se mantiene como líder en el sector, pero cabe recordar que hace cuatro años su cuota de mercado era de cerca del 70% y tanto Nokia como Samsung y Motorola fabricaban todos sus dispositivos con este sistema operativo [71], actualmente solo la finlandesa lo hace. Android apenas tiene dos años y ha tenido un crecimiento exponencial que lo ubica inclusive por encima del iOs de Apple y RIM que tienen mucho más tiempo compitiendo en el sector.

Hasta hace menos de un semestre existía *The Symbian Foundation* como un organismo enfocado en el desarrollo del sistema operativo líder en el mercado, Symbian. Sus miembros eran todos los mayores fabricantes de celulares en el mercado entre los que se encontraban Motorola, Samsung, Sony Ericsson y Nokia como líder. Tanto el código como el sistema operativo eran de dominio público y podían ser descargados de la página web. Sin embargo, el 17 de Diciembre de 2010, Nokia ejecuta el cese de operaciones de la fundación y la privatización de todos los desarrollos futuros de la tecnología, dejando de lado una gran comunidad en línea que respaldaba la tecnología [74].

Actualmente, la gigante Nokia ha realizado un acuerdo con Microsoft para fabricar sus dispositivos inteligentes con el sistema operativo *Windows Phone 7*, que fue lanzado en el año 2010 y cuya cuota de mercado es marginal; sin embargo,

según indicó su CEO era necesario debido a la imposibilidad de que Symbian compitiese con iOS y Android en ese segmento. Por otro lado, Samsung y Motorola han preferido utilizar Android para sus *smartphones* en gran medida. En la tabla 5.8 se puede observar la relación de los tres fabricantes de dispositivos seleccionados con los sistemas operativos previamente mencionados.

Tabla 5.8 Relación fabricantes de dispositivos y sistemas operativos

Fabricante	Symbian	Windows	BADA	Android
Nokia (Teléfonos)	Sí	No	No	No
Nokia (Teléfonos Inteligentes)	Sí	Sí	No	No
Samsung (Teléfonos)	Sí	No	No	No
Samsung (Teléfonos Inteligentes)	No	Sí	Sí	Sí
Motorola (Teléfonos)	No	No	No	Sí
Motorola (Teléfonos Inteligentes)	No	No	No	Sí

5.3 Compatibilidad con Java

De alguna u otra manera, el lenguaje Java está presente en tres de los cuatro sistemas operativos para dispositivos móviles más grandes existentes en el mercado. Para Symbian la compatibilidad es total con la máquina virtual de Java para la ejecución de J2ME. Por otro lado, Android y RIM tienen variaciones del lenguaje que hacen que las extensiones y clases que se utilizan para la ejecución sean diferentes pero se mantiene el paradigma de Java. Únicamente el iOS de Apple es incompatible para la ejecución de Java, debido a que su entorno de desarrollo es propietario.

En la tabla 5.9 se muestra un resumen sobre los lenguajes de programación que se pueden utilizar para crear aplicaciones en los diferentes sistemas operativos con la única diferenciación de que para aplicaciones en Android y RIM se necesitan utilizar clases exclusivas que no se encuentran como estándar en J2ME.

Tabla 5.9 Compatibilidad de Java en diferentes sistemas operativos móviles

	Symbian	**Android**	**RIM**
Lenguajes de programación disponibles	Phyton J2ME Web Runtime (WRT) Flash	Java/C++	Java (clases específicas)
Lenguaje del núcleo	C++	C++, Java	C++, Java
Compatibilidad con J2ME	Sí, implementa JVM	No, no implementa JVM sino Dalvik Virtual Machine (exclusiva)	No a pesar de que implementa JVM

Por tanto, Symbian es el único sistema operativo que permite la ejecución del estándar J2ME a pesar de los otros sistemas operativos utilizan Java en su núcleo. Caso especial es el de Android, que posee una versión modificada de *Java Virtual Machine* en sus dispositivos como se muestra en la figura 5.43. Tal variación es conocida como *Dalvik Virtual Machine* y es exclusiva para Android; sin embargo, no ha sido del agrado de ORACLE quien declara que viola sus derechos de propiedad intelectual [69].

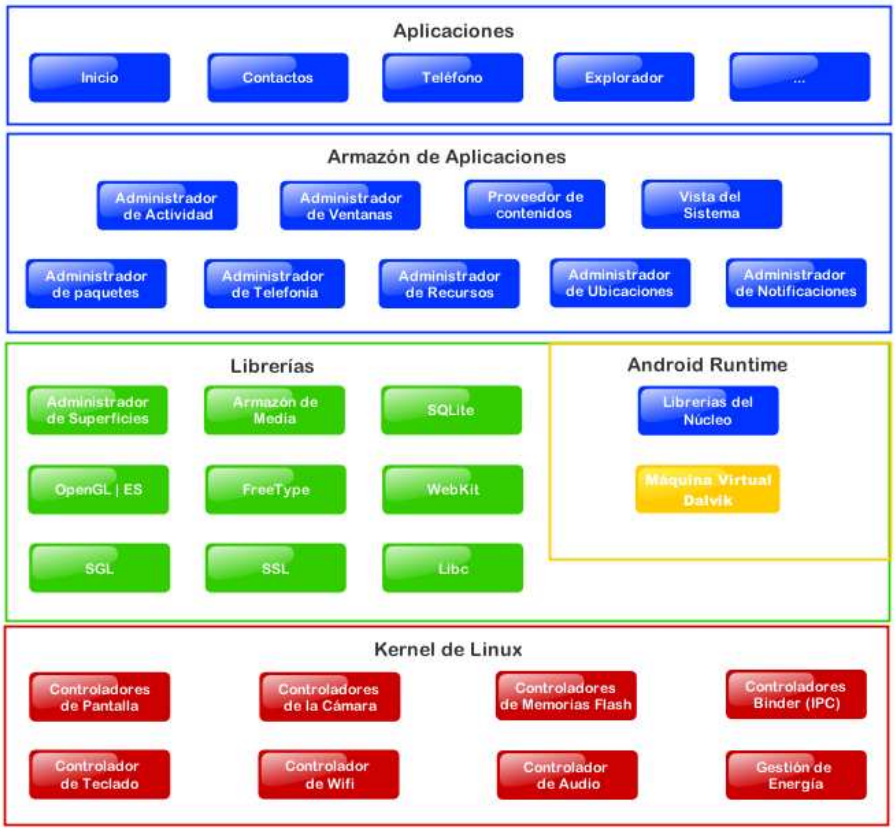


Figura 5.43 Estructura del sistema operativo Android de Google.

5.4 Perspectivas de crecimiento

Hoy en día, el sistema operativo Android concentra la mayor expectativa debido a su rápido crecimiento. Sin embargo, es innegable la cantidad de dispositivos con Symbian existentes en el mercado y la posición de Nokia como el mayor fabricante de dispositivos móviles en el mundo. Por ello, según Gartner, Android para el 2014 aún no será el sistema operativo más utilizado en el mundo pero sí estará muy cerca de arrebatarle al primer lugar a Symbian, como se muestra en la tabla 5.10 [76].

Tabla 5.10 Estimación de división de mercado de sistemas operativos para 2014

	2011	2014
Symbian	34.2%	30.2%
Android	22.2%	29.6%
RIM	15.0%	11.7%
iOS	17.1%	14.9%
Windows	5.2%	3.9%

Otro de los aspectos a considerarse para el afianzamiento de un sistema operativo será la estructura formal que tenga por detrás, es decir una empresa tecnológicamente avanzada y eficiente que la respalde, pero también aquella informal conformada por el apoyo de pequeñas empresas o comunidades de desarrolladores a la tecnología. En este aspecto, Android también tiene una estructura formal e informal bastante fuerte que permite ser optimistas con su futuro. Otra historia tienen todos los otros sistemas operativos que aún mantienen una estructura rígida de comunidad en línea a pesar de estar respaldadas por empresas sólidas. En la tabla 5.11 se muestran las referencias más relevantes para desarrolladores de los diferentes sistemas operativos.

Tabla 5.11 Referencias para desarrolladores de aplicaciones de los principales sistemas operativos

	Página Oficial	**Comunidades en línea**
Symbian	www.forum.nokia.com	No existe
Android	developer.android.com	android-developers.blogspot.com code.google.com/android
RIM	us.blackberry.com/developers/	No Existe
iOS	developer.apple.com	No existe
Windows	create.msdn.com	No Existe

Desde otra perspectiva, la cantidad de aplicaciones creadas hasta la actualidad también representa un factor que debe ser analizado. Nuevamente, Android está a la cabeza con un crecimiento exponencial de sus aplicaciones disponibles en su mercado *Android Market* a pesar de que la *AppStore* de Apple aún encabeza la lista con la mayor cantidad de aplicaciones. En lugares rezagados tenemos las tiendas de aplicaciones de Symbian y Android, como se muestra en la tabla 5.12 [77]

Tabla 5.12 Número de aplicaciones disponibles para cada sistema operativo móvil.

Tienda	**Número de Aplicaciones**	**Plataforma**
Android Market	290,000 (Mar 2011)	Android
App Store	350,000 (Ene 2011)	iOS

App World	16,121 (Dic 2010)	BlackBerry OS
Ovi Store	43,535 (Oct 2010)	Symbian, Java, Maemo
Windows Phone Marketplace	11,367 (Mar 22 2011)	Windows Phone 7

El crecimiento es el último aspecto a analizarse en lo que respecta a sistemas operativos. La conclusión resulta evidente, Android será a futuro el sistema operativo más eficiente para dispositivos inteligentes e inclusive puede adaptarse a teléfonos de gama baja; sin embargo, Symbian aún cuenta con un mercado bastante amplio gracias al respaldo de Nokia por lo que aún será necesario tomarlo en cuenta por algún tiempo. Finalmente, cabe mencionar que Oracle a principios de 2011 a anunciado que elaborará un entorno de programación unificado para el desarrollo de aplicaciones en Java sin importar el sistema operativo que maneje [78]

5.5 Entorno de Desarrollo y simuladores.

En este aspecto se valorará por fabricante la información entregada para el desarrollo de aplicaciones. Esto involucra las clases, entornos de programación, simuladores y herramientas que brinde para programadores. Foros y soporte en línea de terceros también brindarán puntaje adicional a los fabricantes.

- **Nokia**

La figura 5.44 muestra la página principal para desarrolladores de Nokia, mostrada en la tabla 5.11. En la misma se muestran los enlaces para acceder a la documentación sobre las librerías que se pueden utilizar en los diferentes lenguajes de programación así como ayudas o ejemplos para novatos en programación.

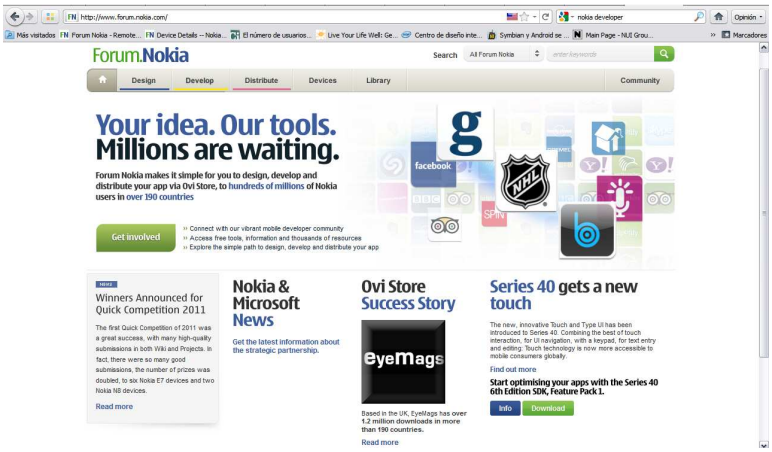


Figura 5.44 Página Principal de desarrolladores Nokia

La comunidad oficial en línea también puede ser accedida desde la página principal y cuenta con gran cantidad de información. Según se indica en la página web existen 208436 temas en discusión y 8840 artículos de una enciclopedia colaborativa en línea similar a *Wikipedia.* Nokia está totalmente relacionado con Symbian y aún no presenta herramientas para el desarrollo de aplicaciones bajo Windows Phone 7.

La sección de dispositivos móviles es bastante completa y muestra información de todos los modelos desarrollados por la compañía durante los cinco años anteriores. Muestra sus características en *software* más importantes y perfiles de manejo de *hardware* implementados, entre los que se pueden encontrar los de Bluetooth. En la figura 5.45 se muestra la página de descripción del dispositivo Nokia E72.

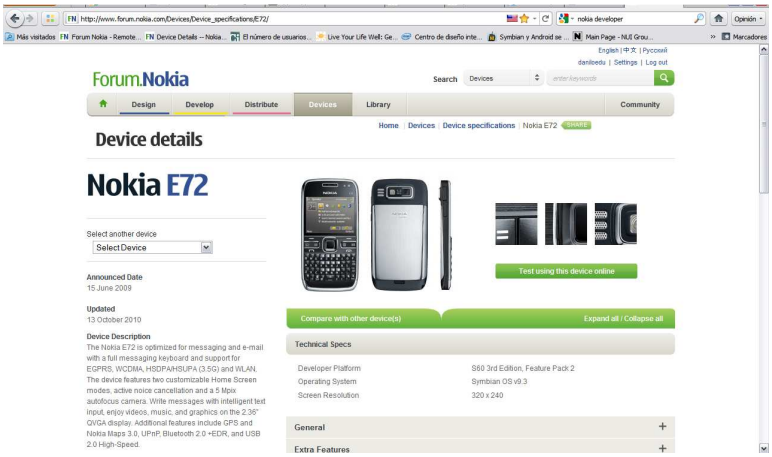


Figura 5.45 Descripción de dispositivos Nokia en la web de desarrolladores Nokia

En lo que respecta a simuladores, posee una herramienta muy poderosa en línea que permite emular el comportamiento de cualquier dispositivo Nokia con todas sus características. Esta funcionalidad se ofrece de manera gratuita y mantiene disponible gran cantidad de celulares existentes en el mercado. El comportamiento es bastante real y muy similar al del dispositivo real. En la figura 5.46 se muestra un ejemplo de simulación del *Smartphone* N8 de Nokia

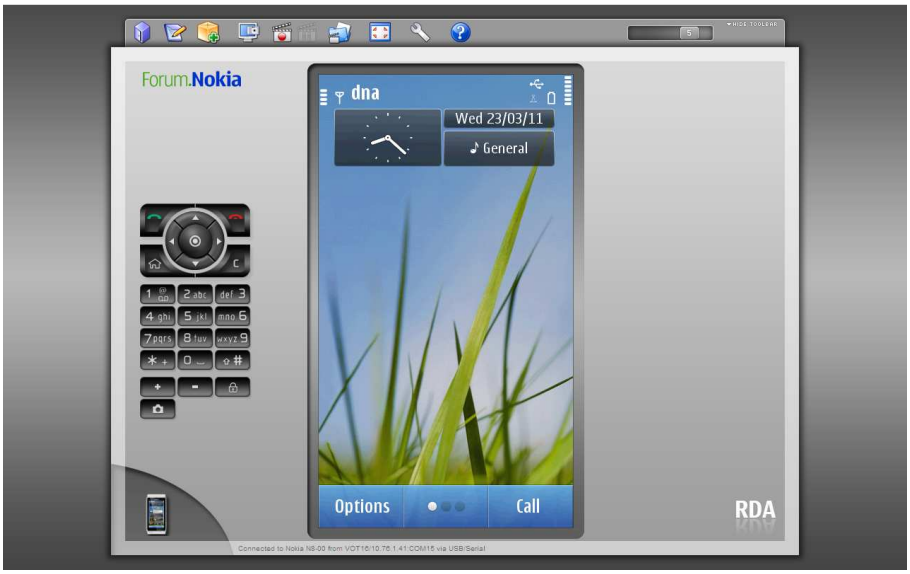


Figura 5.45 Simulación del *Smartphone* N8 en la web de desarrolladores de Nokia

De igual manera, ofrece herramientas para el desarrollo de aplicaciones específicas para dispositivos Nokia que son totalmente compatible con el entorno de

programación Netbeans. Toda la información se encuentra en la página de desarrolladores de Nokia.

- **Motorola**

La empresa estadounidense Motorola están enfocada totalmente en el desarrollo de aplicaciones para Android, inclusive para sus dispositivos de gama baja. En la figura 5.46 se muestra la página principal de programadores de Motorola (http://developer.motorola.com/).

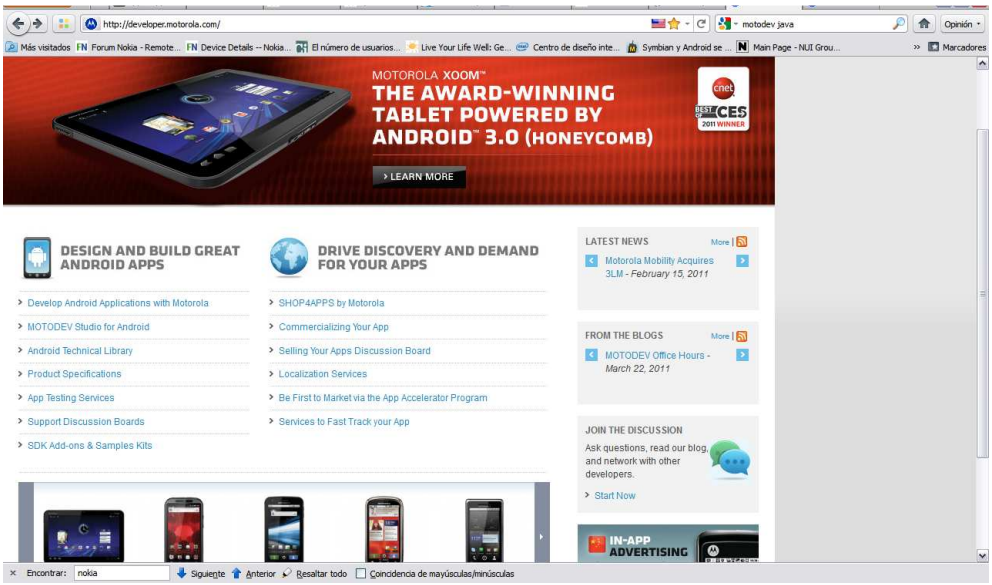


Figura 5.46 Página oficial de desarrolladores de Motorola.

Al igual que Nokia, ofrece una documentación bastante completa sobre las especificaciones de sus dispositivos móviles, como se muestra en la figura 5.47. Sin embargo, el foro de discusión es bastante pequeño lo que se deba probablemente a que la mayor cantidad de información ya se encuentra disponible en el sitio web oficial de programadores de Android que es totalmente independiente de Motorola

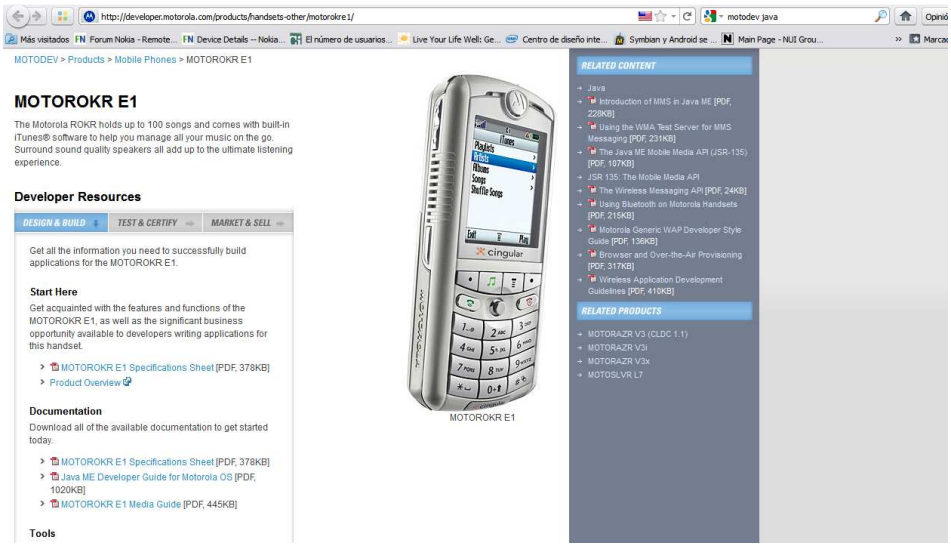


Figura 5.47 Descripción de dispositivos Motorola en la web de desarrolladores Motorola

El compromiso con Android de la empresa es evidente, tanto que actualmente el 30% del total de dispositivos móviles con ese sistema operativo son fabricados por Motorola [72]. Por lo que su entorno de programación es el mismo y no posee herramientas adicionales. Sin embargo, tiene un tutorial que muestra como migrar aplicaciones J2ME a Android para programadores novatos [79]

En lo referente a simuladores, no posee una herramienta como Nokia y únicamente hace referencia a aplicaciones de terceros encargadas de la emulación de dispositivos móviles, como *Deviceanywhere*, que ofrece un amplio portafolio de celulares entre los que se incluyen dispositivos de otras marcas como el Iphone de Apple [80].

- **Samsung**

Samsung ha enfocado sus productos en varios frentes, implementa dispositivos basados en Java, Bada, Windows y Android. Los dispositivos basados en Java son similares a los de Nokia basados en Symbian. En la figura 5.48 se muestra la página principal de programadores de Samsung (http://innovator.samsungmobile.com/)

Figura 5.48 Página oficial de desarrolladores de Samsung.

De igual manera, presenta las características principales de dispositivos de acuerdo a la tecnología que implementan. Para el caso de Java, no presenta la totalidad de dispositivos existentes en el mercado sino únicamente unos referenciales, en la figura 5.49 se muestra un ejemplo de un dispositivo de la empresa. Esta limitante resulta frustrante al tener que recurrir a sitios web de terceros para conocer sobre las características de los dispositivos.

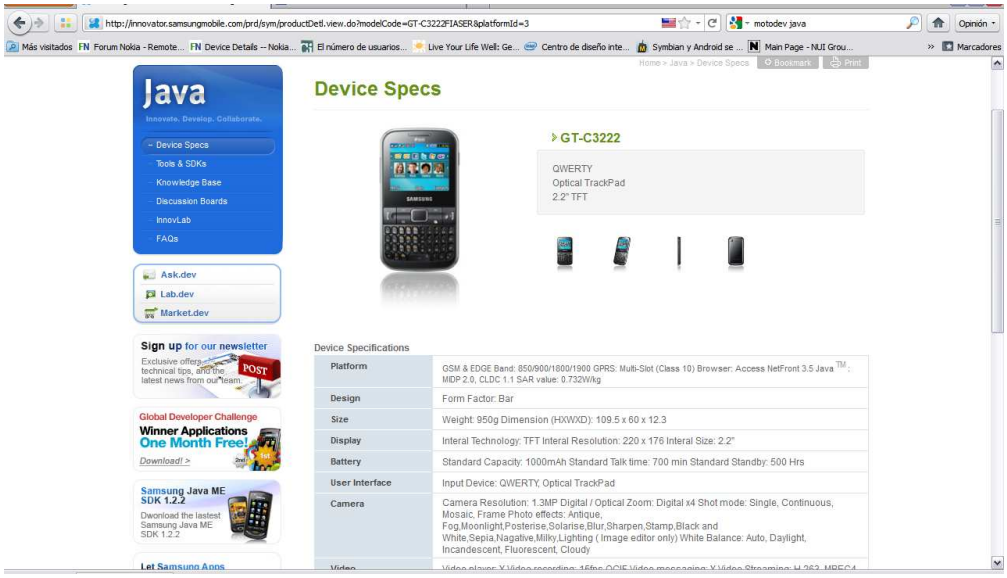


Figura 5.49 Descripción de dispositivos Samsung en la web de desarrolladores Samsung

Los foros de discusión son bastante limitados y no ofrece información sobre simuladores disponibles, ni siquiera referenciales de terceros. Tampoco sobre complementos al entorno de programación.

Una vez analizados los tres casos, se puede evaluar cada uno de los aspectos como se observa en la tabla 5.13, donde resulta como claro ganador el fabricante Nokia por el respaldo ofrecido a sus programadores con todas las herramientas en línea.

Tabla 5.13 Valoración de herramientas de programación para cada fabricante de dispositivos móviles. Cada elemento tiene como calificación máxima 5.

	Nokia	**Motorola**	**Samsung**
Documentación de dispositivos	5	4	2
Complementos para el entorno de programación	5	1	1
Herramientas de simulación	5	2	0
Soporte en línea de comunidades	5	1	1
Total	20	8	4
Gratuidad de herramientas	Sí	Sí	Sí

Por tanto, para la aplicación de mensajería se utilizarán dispositivos móviles de marca Nokia cuya compatibilidad con Bluetooth haya sido previamente verificada y que ejecuten el sistema operativo Symbian por ser el más ampliamente difundido en el país, como se muestra en la tabla 5.14, obtenida de la empresa *GlobalStats*, encargada de estadísticas tecnológicas [81].

Tabla 5.14 División del mercado ecuatoriano en sistemas operativos móviles

	Ecuador (Marzo 2011)
Symbian	57.14%
Android	5.03%
RIM	11.77%
iOS	10.88%
Samsung	7.9%

6. Implementación en dispositivo móvil

Para la aplicación de mensajería, el dispositivo móvil debe cumplir con un mínimo de requerimientos para ejecutar la aplicación de manera correcta. Evidentemente será necesario que el dispositivo sea compatible con J2ME y más específicamente con JSR-82 y que permita el intercambio de paquetes mediante objetos (GOEP) para la utilización del protocolo OBEX. Sin embargo, para la estabilidad de la aplicación y ejecución se determinó que el conjunto de características son más variadas y se muestran en la tabla 5.15.

Tabla 5.15 Requerimientos para la ejecución de la aplicación de Mensajería

	Características
Java (Recomendaciones)	JSR 139 Connected, Limited Device Configuration (CLDC) 1.1 JSR 118 MIDP 2.1 JSR 248 Mobile Service Architecture Subset for CLDC JSR 82 Java™ APIs for Bluetooth 1.1
Bluetooth (Perfiles)	A2DP, AVRCP, BIP, DUN, FTP, GAP, GAVDP, GOEP, HFP, HID, HSP, OPP, PBAP, SAP, SDP, SPP

Debido a que la implementación de la pila Bluetooth varía dependiendo del fabricante, es muy probable que la aplicación se ejecute sin cumplir con todas las características mencionadas en la tabla 5.15, pero no se asegura su estabilidad.

La instalación de la aplicación se la hará mediante la utilización de Bluetooth; es decir, el archivo ejecutable .jar será enviado directamente al dispositivo. Dependiendo de la versión

de Symbian implementada en el dispositivo se permitirá la selección de la carpeta o no. De igual manera, dependiendo de la BCC los permisos se otorgarán automáticamente o se deberá gestionarlos durante la ejecución de la aplicación. En la figura 5.50 se muestra la secuencia de pantallas de instalación para un dispositivo Nokia N93 con sistema operativo Symbian S60, mientras que en la figura 5.51 el mismo proceso para un Nokia N8 con sistema operativo Symbian^3.

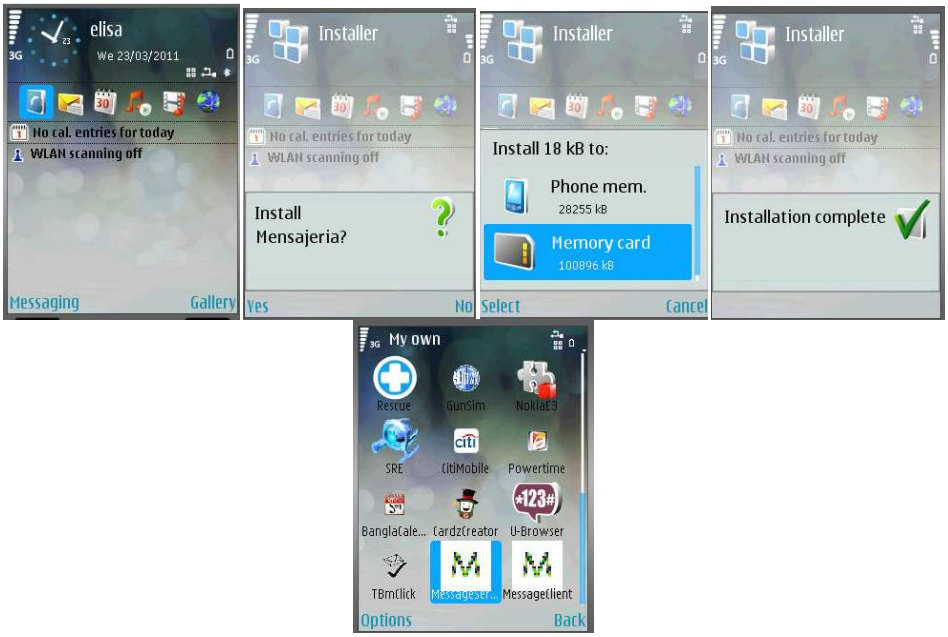


Figura 5.50 Secuencia de instalación de aplicación de mensajería en un dispositivo Nokia N93

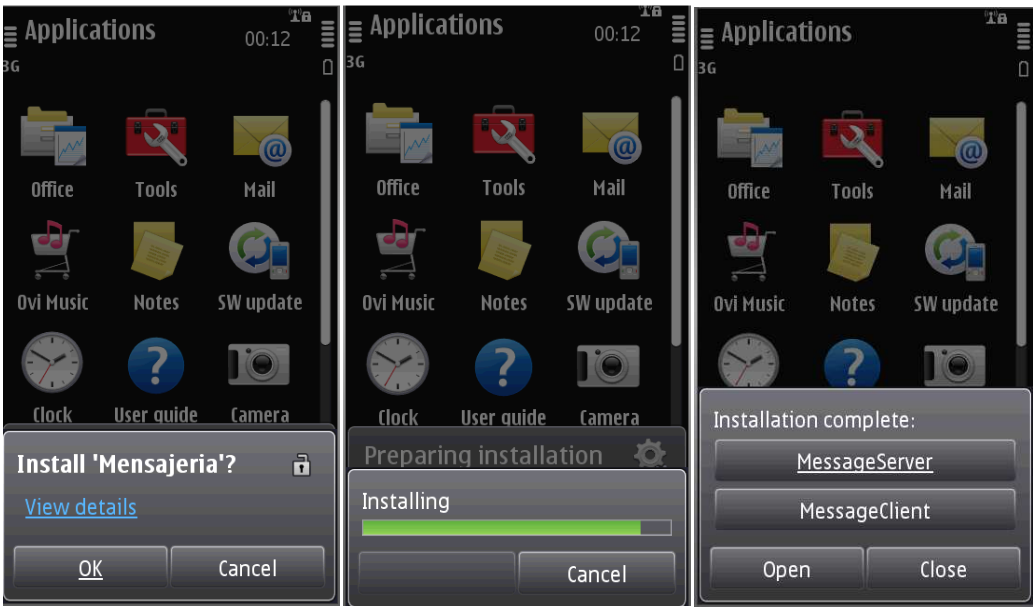


Figura 5.51 Secuencia de instalación de aplicación de mensajería en un dispositivo Nokia N8

6.1 Herramientas a utilizarse

Para evaluar el comportamiento de la aplicación en el dispositivo móvil no se utilizarán herramientas adicionales. El desempeño se probará mediante la ejecución del programa en diversos dispositivos bajo diferentes condiciones para conocer el alcance y cantidad de nodos que se puede manejar.

En primera instancia se realizará una prueba sobre la estabilidad de ejecución en varios dispositivos móviles de diferentes marcas y modelos a pesar de que previamente se indicó que no era lo más óptimo pero es necesario conocer los resultados. Posteriormente, se evaluará la distancia que se puede alcanzar utilizando únicamente dispositivos móviles, el tiempo de establecimiento de conexión, cantidad de nodos que un dispositivo puede manejar, capacidad para identificar el servicio y la eficiencia en la entrega de mensajes. Los resultados de las pruebas realizadas se muestran a continuación.

6.2 Ejecución de la aplicación en varios dispositivos móviles

Para este caso, se instalará la aplicación en varios modelos de dispositivos móviles de diferentes marcas y únicamente se establecerá si la aplicación puede ejecutarse o no. No se hará análisis más profundos sobre su estabilidad y conflictos con el centro de control BCC.

En la tabla 5.16 se muestran la lista de dispositivos que fueron utilizados para realizar las pruebas de ejecución de mensajería, en donde también se indica si la aplicación funcionó correctamente o no. La gran mayoría de dispositivos son de la empresa Nokia principalmente porque existen en gran cantidad en el mercado ecuatoriano, como se indicó anteriormente y también por el simulador en línea incluido en el sitio web de desarrolladores de Nokia.

Tabla 5.16 Requerimientos para la ejecución de la aplicación de Mensajería

Fabricante	Modelo	Sistema Operativo	Resultado de Ejecución
Nokia	2680s	Symbian S40	Exitoso
Nokia	2690	Symbian S40	Exitoso
Nokia	2700	Symbian S40	Exitoso
Nokia	2710	Symbian S40	Exitoso
Nokia	5130	Symbian S40	Exitoso
Nokia	5310	Symbian S40	Exitoso
Nokia	N95 8GB	Symbian S60	Exitoso
Nokia	N95	Symbian S60	Fallido
Nokia	N93	Symbian S60	Exitoso
Nokia	N86 8MP	Symbian S60	Exitoso
Nokia	N97	Symbian S60	Exitoso
Nokia	X6-00	Symbian S60	Exitoso
Nokia	N97 Mini	Symbian S60	Exitoso
Nokia	N81	Symbian S60	Exitoso
Nokia	N76	Symbian S60	Exitoso
Nokia	E51	Symbian S60	Exitoso
Nokia	E52	Symbian S60	Exitoso
Nokia	E55	Symbian S60	Exitoso
Nokia	E66	Symbian S60	Exitoso
Nokia	E62	Symbian S60	Exitoso
Nokia	E71x	Symbian S60	Exitoso
Nokia	E72	Symbian S60	Exitoso
Nokia	E73	Symbian S60	Fallido
Nokia	E75	Symbian S60	Exitoso
Nokia	5500	Symbian S60	Fallido
Nokia	5230	Symbian S60	Exitoso
Nokia	5250	Symbian S60	Exitoso
Nokia	5350	Symbian S60	Exitoso
Nokia	5730	Symbian S60	Exitoso
Nokia	5800	Symbian S60	Exitoso
Nokia	6110	Symbian S60	Fallido
Nokia	6760	Symbian S60	Exitoso
Nokia	C5-03	Symbian S60	Exitoso
Nokia	N900	Maemo	Fallido
Nokia	N8	Symbian^3	Exitoso
Nokia	E7-00	Symbian^3	Exitoso
Nokia	C7-00	Symbian^3	Exitoso
Nokia	C6-01	Symbian^3	Exitoso
Sony Ericsson	W302	Symbian	Fallido
Sony Ericsson	W705	Symbian	Fallido
Sony Ericsson	W205	Symbian	Fallido
Sony Ericsson	K850	Symbian	Fallido
Sony Ericsson	W902	Symbian	Fallido
Sony Ericsson	W960	Symbian	Fallido
Sony Ericsson	W580	Symbian	Fallido
Samsung	S5230 Star	Symbian	Exitoso

Samsung	B5310	Symbian	Exitoso
Samsung	S3650 Corby	Symbian	Exitoso
Samsung	B3410W	Symbian	Exitoso
LG	MG295d	Symbian	Fallido
Motorola	V3i	Propietario Motorola	Exitoso
Motorola	MOTOPEBL U6	Propietario Motorola	Exitoso
Motorola	MOTORAZR V3 (CLDC 1.1)	Propietario Motorola	Exitoso
Motorola	MOTOSLVR L7	Propietario Motorola	Exitoso
Motorola	E1070	Propietario Motorola	Exitoso
Motorola	MOTORIZR Z3	Propietario Motorola	Exitoso
Motorola	MOTOROKR E6	Propietario Motorola	Exitoso
Motorola	MOTOSLVR L7i	Propietario Motorola	Exitoso
Motorola	W510	Propietario Motorola	Exitoso
HUAWEI	U7510	Symbian	Fallido
HUAWEI	U1270	Symbian	Fallido
Motorola	DROID	Android	Fallido
Apple	Iphone 3Gs	iOS 4.1	Fallido
Blackberry	Storm 9500	RIM 4.2	Fallido
Blackberry	Pearl	RIM 3.0	Fallido
HTC	ST6356	Android	Fallido
HTC	INCREDIBLE	Android	Fallido
Samsung	Galaxy S	Android	Fallido
GOOGLE	NEXUS ONE	Android	Fallido

Se considero una notable diversidad de dispositivos móviles para hacer las pruebas de funcionamiento de la aplicación de mensajería. Sin embargo, los dispositivos Nokia representan un 56.52% del total de los celulares probados, como se muestra en la tabla 5.17. En lo referente a sistemas operativos predomina Symbian con 73.9%, como se muestra en la tabla 5.18. Finalmente, aún dentro de las distribuciones de Symbian existen variantes debidas a personalizaciones de los fabricantes, en tal caso Symbian S60 de Nokia predomina con un 53% del total de distribuciones existentes como se muestra en la tabla 5.19.

Tabla 5.17 Presencia de sistemas operativos donde se realizaron las pruebas de ejecución de la aplicación de mensajería.

Sistema Operativo	Número de Dispositivos
Android	5
iOS	1
RIM	2
Symbian	51
Otros	10

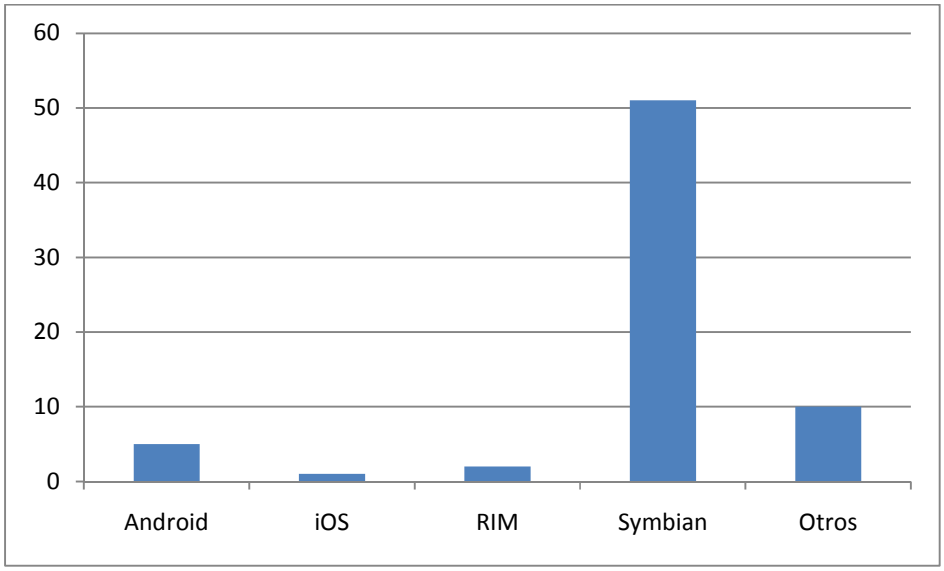


Figura 5.52 Presencia de sistemas operativos donde se realizaron las pruebas de ejecución de la aplicación de mensajería.

Tabla 5.18 División de fabricantes de dispositivos móviles donde se realizaron pruebas de ejecución de la aplicación de mensajería.

Fabricante	Cantidad de dispositivos
Nokia	39
Samsung	5
LG	1
Sony Ericsson	7
Motorola	10
Blackberry	2
HTC	2
HUAWEI	2
GOOGLE	1

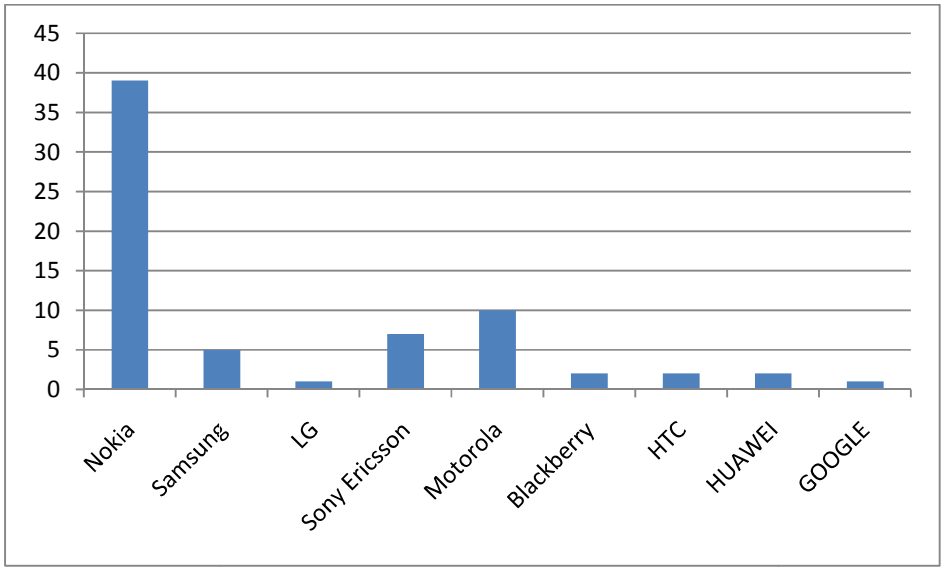


Figura 5.53 División de fabricantes de dispositivos móviles donde se realizaron pruebas de ejecución de la aplicación de mensajería.

Tabla 5.19 Presencia de distribuciones Symbian en dispositivos móviles donde se realizaron pruebas de ejecución de la aplicación de mensajería.

Symbian S60	27
Symbian S40	6
Symbian 3	4
Symbian	14

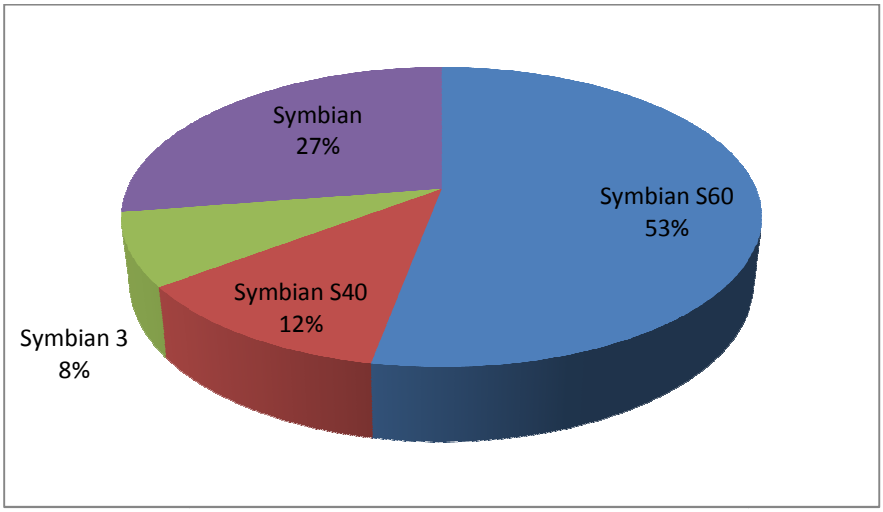


Figura 5.54 Presencia de distribuciones Symbian en dispositivos móviles donde se realizaron pruebas de ejecución de la aplicación de mensajería.

De la tabla 5.16 se puede observar que se han incluido teléfonos con sistema operativo Android, RIM y iOS para validar lo que se expuso anteriormente en la teoría. Evidentemente al ampliar el espectro de dispositivos móviles con sistema operativo no compatible el porcentaje de ejecución exitosa es del 65.22%, como se observa en la en la tabla 5.20. Por otro lado cuando se refiere únicamente a dispositivos de sistema operativo Symbian la efectividad en la ejecución sube más de cinco puntos porcentuales al ubicarse en 72.55%, como se muestra en la tabla 5.21.

Tabla 5.20 Porcentaje de ejecuciones exitosas de la aplicación de mensajería en dispositivos móviles de varios sistemas operativos

	Número de dispositivos	Porcentaje del total
Ejecución exitosa	45	65,22
Ejecución Fallida	24	34,78

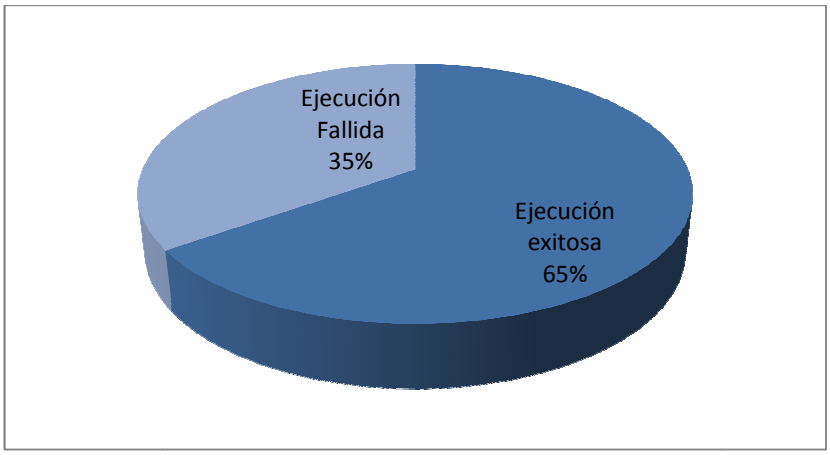


Figura 5.55 Porcentaje de ejecuciones exitosas de la aplicación de mensajería en dispositivos móviles de varios sistemas operativos

Tabla 5.21 Porcentaje de ejecuciones exitosas de la aplicación de mensajería en dispositivos móviles con sistema operativo Symbian.

	Cantidad de dispositivos	**Porcentaje del total**
Ejecución exitosa sobre Symbian	37	72,55
Ejecución Fallida sobre Symbian	14	27,45

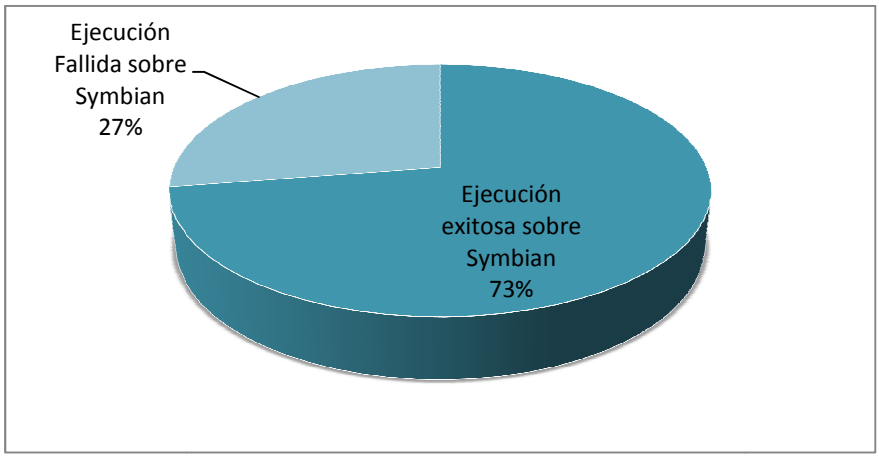


Figura 5.56 Porcentaje de ejecuciones exitosas de la aplicación de mensajería en dispositivos móviles con sistema operativo Symbian.

6.3 Evaluación de la distancia que se puede alcanzar.

La distancia siempre ha presentado una limitante para la tecnología Bluetooth, debido a los bajos niveles de potencia en la transmisión que se utilizan. Según el estándar para un radio clase 2 el máximo poder permitido es de alrededor de 4 dBm con lo que se podría alcanzar una distancia aproximada para la comunicación de unos 10m. Es evidente que los fabricantes no instalan radios que utilicen la máxima potencia posible en sus dispositivos para prolongar la duración de la batería, por lo que se puede esperar una comunicación efectiva a menos de 5m de separación entre dispositivos móviles.

En la tabla 5.22 se muestra los resultados obtenidos en pruebas de campo para dos dispositivos móviles existentes en el mercado. Tales valores corresponden al promedio de diez mediciones realizadas bajo un entorno abierto, sin presencia de lluvia ni de señales que pudieran alterar la comunicación. Además identifican la distancia máxima para una pérdida significativa de la señal y la pérdida total de la conexión. En las figuras 5.57 y 5.58 se muestran las mediciones realizadas cuando

el dispositivo E72 se comportó como servidor mientras que en las figuras 5.59 y 5.60 lo obtenido cuando se comportó como cliente.

Tabla 5.22 Distancia máxima alcanzada por los dispositivos móviles en la aplicación de mensajería.

	Distancia máxima alcanzada para comunicación [m]	Distancia máxima de desconexión [m]
Nokia E72 servidor – Nokia 2680s cliente	5.25	7.32
Nokia E72 cliente – Nokia 2680s servidor	4.60	6.39

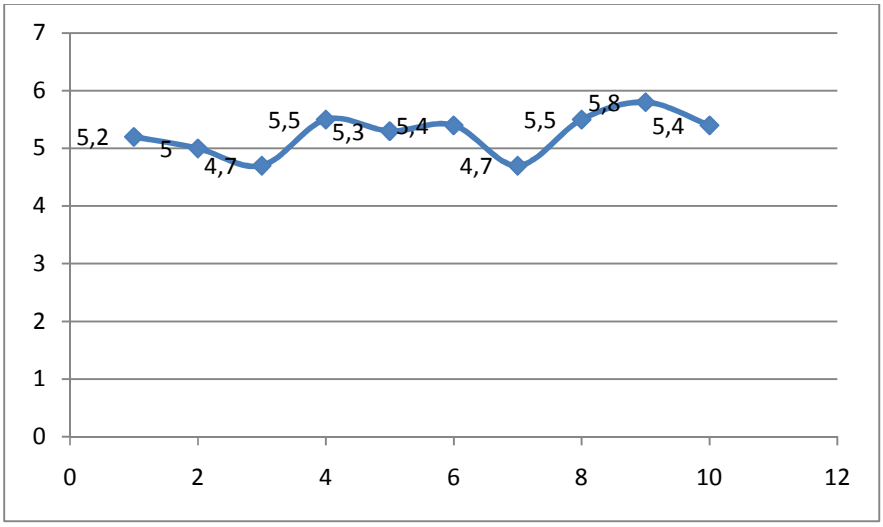


Figura 5.57 Mediciones de la distancia máxima alcanzada para la comunicación entre los dispositivos móviles en la aplicación de mensajería cuando el dispositivo E72 se comportó como servidor

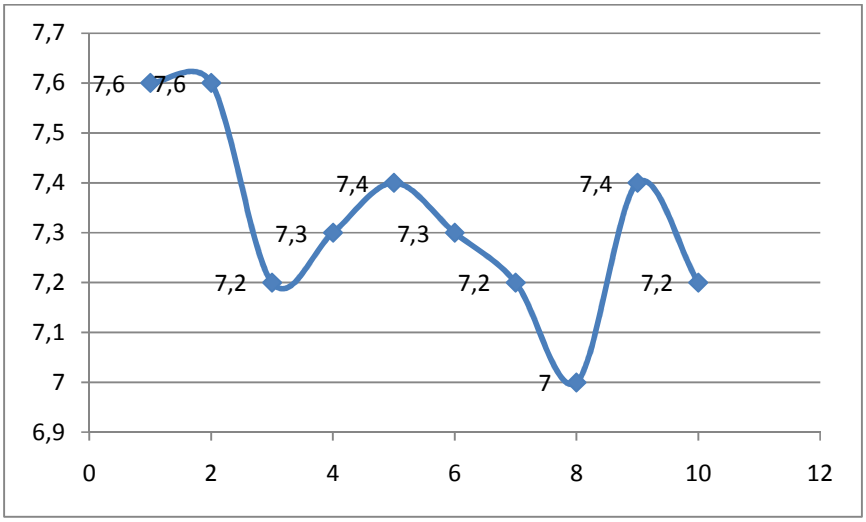


Figura 5.58 Mediciones de la distancia máxima alcanzada para la conexión entre los dispositivos móviles en la aplicación de mensajería cuando el dispositivo E72 se comportó como servidor

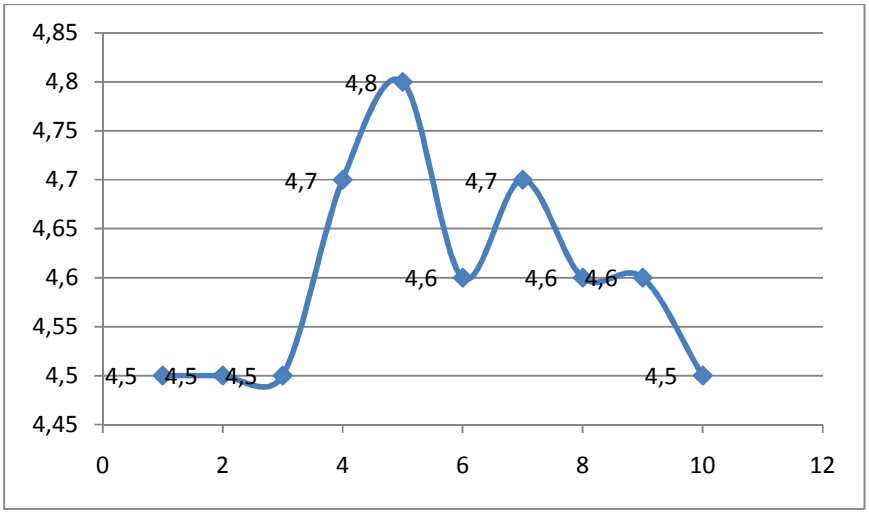


Figura 5.59 Mediciones de la distancia máxima alcanzada para la comunicación entre los dispositivos móviles en la aplicación de mensajería cuando el dispositivo E72 se comportó como cliente

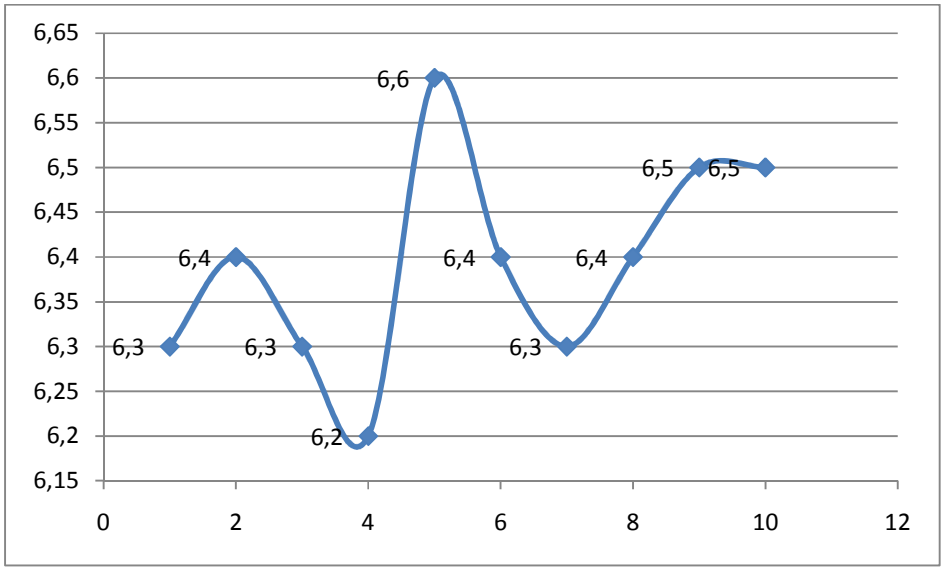


Figura 5.60 Mediciones de la distancia máxima alcanzada para la conexión entre los dispositivos móviles en la aplicación de mensajería cuando el dispositivo E72 se comportó como cliente

Otro de los factores que afecta la comunicación es la banda de 2.4 Ghz que es utilizada por la constante interferencia por parte de otras tecnologías. Sin embargo, esta interferencia es menor a la producida por la presencia de cuerpos en un entorno cerrado, más aún si son seres humanos, ya que están compuestos en un 80% de agua. En la tabla 5.23 se observa las distancias promedio alcanzadas para la comunicación en un entorno bajo estas consideraciones.

Tabla 5.23 Distancia máxima alcanzada por los dispositivos móviles en la aplicación de mensajería cuando existen factores de interferencia externos

	Presencia de interferencia en banda de 2.4 GHz	Presencia de más de cinco personas en la habitación de comunicación
Nokia E72 servidor – Nokia 2680s cliente	4.63	3.34
Nokia E72 cliente – Nokia 2680s servidor	4.15	3.54

Lo que demuestra que el entorno juega un papel fundamental en el desempeño de la comunicación por Bluetooth al utilizar la aplicación de mensajería. Cabe destacar que en ningún caso la aplicación o el dispositivo móvil colapsaron

totalmente, mostrando la independencia de ejecución de la aplicación en Java del sistema operativo del celular.

- **Evaluación del tiempo de establecimiento de conexión y certeza de encontrar el servicio.**

Para esta prueba se utilizaron varios dispositivos móviles compatibles con la aplicación de mensajería. Sin embargo todos fueron del fabricante Nokia debido a que presenta las mejores condiciones para la ejecución del servicio.

Se realizaron veinte mediciones de donde el tiempo promedio para encontrar el servicio fue de 11.29 segundos como se muestra en la tabla 5.24. Este tiempo va acorde a lo establecido por el estándar Bluetooth que indica que se requiere mínimo 10 segundos para realizar la búsqueda de un dispositivo móvil.

Tabla 5.24 Medición de tiempos de establecimiento de conexión entre cliente y servidor en la aplicación de mensajería.

Mediciones	Tiempo [s]
1	11,2
2	11,5
3	11,3
4	11,2
5	11,4
6	11,3
7	11,2
8	11,5
9	11,4
10	11,4
11	11,3
12	11,1
13	11,2
14	11,4
15	11,3
16	11,1
17	11,2
18	11,2
19	11,2
20	11,4
Promedio	11,29

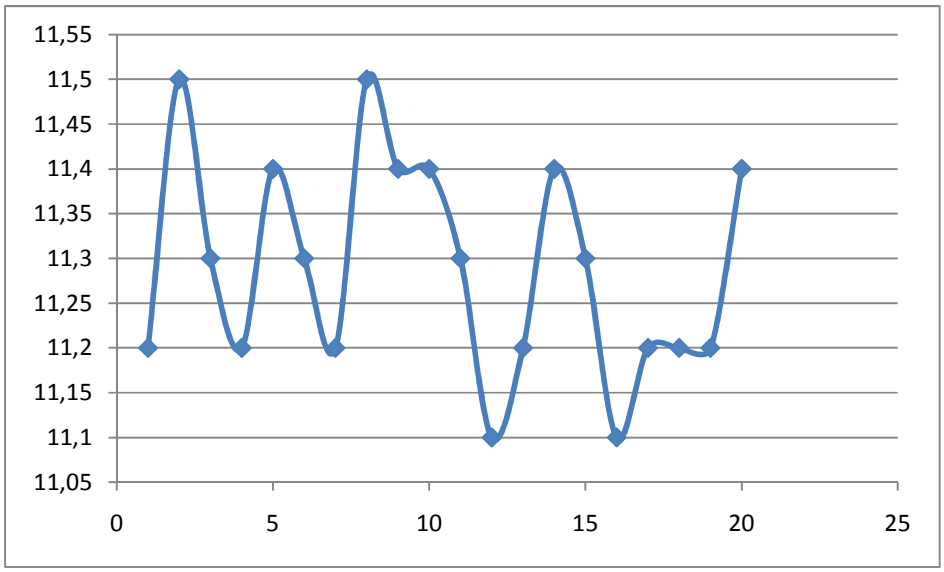


Figura 5.61 Mediciones de los tiempos para la conexión entre los dispositivos móviles en la aplicación de mensajería.

El otro factor a evaluar es la efectividad de encontrar el servicio. En este punto cabe destacar que una vez encontrado el dispositivo, en el 100% de las pruebas se pudo acceder al servicio. Sin embargo, el proceso de hallar al servidor no tiene la misma eficiencia, por ello al momento de realizar las pruebas por dos ocasiones se tuvo que repetir la búsqueda, esto indica que en un 90% de los casos el cliente pudo encontrar al servidor, como se muestra en la figura 5.62

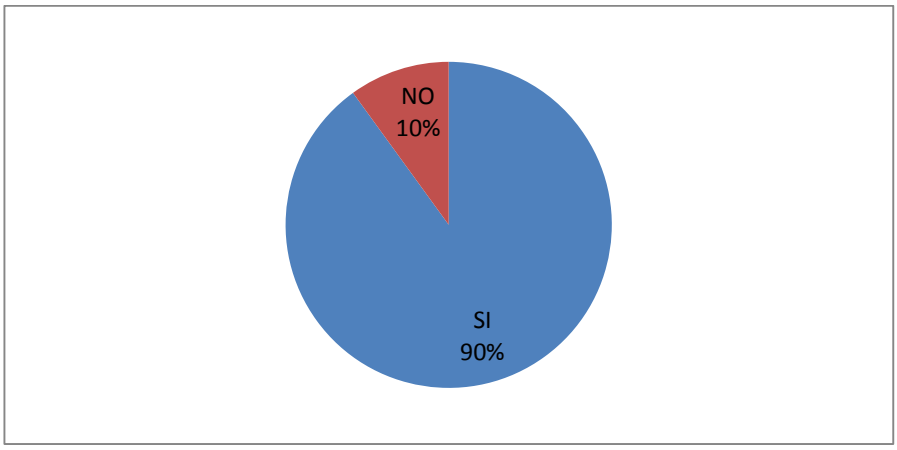


Figura 5.62 Porcentaje de efectividad en la búsqueda del servidor en la aplicación de mensajería..

6.4 Evaluación de la cantidad de nodos que un dispositivo móvil puede manejar

Este aspecto presentó la mayor limitante de la aplicación. Las pruebas indicaron que los dispositivos móviles que incluyan todas las características indicadas en la tabla 5.15, como es el caso del celular Nokia E72, pueden enlazarse al mismo tiempo con otros dos equipos a la vez. Cualquier otro equipo de menor capacidad puede enlazarse únicamente con un dispositivo. Esto debido a la capa de transporte OBEX que se utiliza.

Por tal razón, se elaboró un aplicativo que utilice únicamente la capa RFCOMM para la comunicación. Sin embargo, los resultados tampoco fueron satisfactorios debido a que al eliminar el protocolo OBEX se suprime también la transmisión de mensajes por medio de cabeceras y el manejo de los flujos de información se vuelve más complejo, teniendo que establecerse un método de transmisión asincrónico para el funcionamiento de la aplicación, lo cual desemboca en la súbita terminación de la aplicación cuando Symbian y la BCC no pueden manejar la información.

En cualquier caso, cabe recordar que el funcionamiento de la aplicación de mensajería se da aún cuando el cliente no está conectado. Por ello, la necesidad de que el servidor esté conectado con la mayor cantidad de clientes no es imperiosa. En la tabla 5.25 se muestran los resultados sobre los que se comentó previamente.

Tabla 5.25 Cantidad de dispositivos con los que puede conectarse el servidor en la aplicación de mensajería bajo diferentes condiciones de comunicación.

	Dispositivos con los que puede conectarse como clientes (utilizando OBEX)	Dispositivos con los que puede conectarse como clientes (utilizando RFCOMM)
Nokia E72 servidor	2	3
Nokia 2680s servidor	1	3

6.5 Evaluación de la eficiencia en la entrega de mensajes.

En este aspecto, los resultados fueron bastante satisfactorios debido a que en la totalidad de los casos los mensajes fueron entregados a sus destinatarios. Esto incluye eventos en los cuales el mensaje fue enviado previamente al registro del cliente y de usuarios que tengan nombres similares, en la tabla 5.26 se muestran los resultados obtenidos para las pruebas de envío de mensajes.

Tabla 5.26 Resultados de entrega de mensajes a destinatarios en la aplicación de mensajería.

Operación	Resultado
Porcentaje de entrega de mensajes a destinatarios	100%
Entrega de mensajes enviados previos al registro del cliente	Sí
Recepción de totalidad de menajes enviados previo al registro del ciente	Sí
Diferenciación por nombre de usuario	Sí
Diferenciación de nombre de usuario por letras mayúsculas y minúsculas	Sí
Envío de múltiples mensajes a destinatarios (previo su registro)	Sí

La aplicación se comporta de manera adecuada cuando es ejecutada en dispositivos móviles compatibles. Sin embargo, las características limitadas de los celulares existentes en el mercado no permiten que se pueda lograr el funcionamiento deseado para las Piconets, peor aún para la formación de una Scatternet. Por ello, resulta necesario buscar métodos alternativos para mejorar la calidad y alcance de la comunicación.

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

1. Conclusiones

1.1 Cualquier protocolo de enrutamiento mostrado puede ser utilizado en la aplicación para entornos más amplios con el respectivo aumento en el procesamiento de los nodos que conforman la red lo cual compromete la vida de la batería del dispositivo móvil

1.2 Teóricamente se han definido protocolos híbridos que reducen considerablemente la posibilidad de errores de enrutamiento y tiempo de convergencia de la red pero cuya complejidad implica un nivel de procesamiento bastante alto que no puede ser aplicado para los dispositivos móviles actuales.

1.3 La aplicación de mensajería escrita en J2ME implementa una tabla de enrutamiento en la capa de aplicación y no en la red lo que implica un aumento del retraso en el procesamiento, conmutación de la información y acceso al nivel físico de transmisión por Bluetooth. La solución no es óptima pero va acorde a lo definido por el estándar Java.

1.4 La distancia alcanzada entre dispositivos móviles al utilizar Bluetooth no es extensa debido a que la tecnología no lo permite ni está diseñada para tales propósitos. Por ello, un aumento de cobertura se debe dar por un aumento en el número de nodos que componen una red

1.5 El número de dispositivos móviles que pueden estar enlazados al mismo tiempo varía de acuerdo al tipo de canal Bluetooth que se esté utilizando. Por ello, un dispositivo puede comunicarse con un único dispositivo mediante el canal de voz y hasta con tres con la utilización de los canales de paquetes de datos.

1.6 La creación de Piconets bajo una estructura de cliente servidor facilita el proceso de formación de redes ad hoc a pesar de centralizar los servicios. Más aún la formación de Scatternets alivia la carga de

procesamiento para los servidores de las Piconets y provee métodos de respaldo ante caídas de nodos o clientes.

1.7 El establecimiento de un valor UUID facilita la identificación de servicios en una aplicación J2ME con Bluetooth, pero además provee un primer nivel de seguridad para la comunicación de los dispositivos móviles.

1.8 El sistema operativo Symbian se mantiene como líder de mercado. Sin embargo, la imposibilidad de competir con otros a nivel de teléfonos inteligentes ha hecho que sus futuros desarrollos se vean comprometidos.

1.9 Android, es el sistema operativo con mayor acogida de finales de la década tanto en crecimiento cuanto en perspectivas de desarrollo. Tiene la fortaleza de estar basado en una sólida estructura brindada por Linux y Java. Además, la apertura que brinda a sus desarrolladores para crear aplicaciones.

1.10 Las tendencias indican que a futuro los sistemas operativos que predominarán en el mercado tendrán código abierto y serán totalmente personalizables. Esta libertad, es una herramienta muy poderosa que debe ser utilizada para la mejora continua de las aplicaciones y de los mismos sistemas operativos.

1.11 La apertura de Android puede ser utilizada para la creación de aplicaciones nocivas que sirvan para robar información de usuarios. Sin embargo, la sociedad actual se encuentra más preparada al momento de decidir los aplicativos que usa para no correr experiencias anteriores de Windows. La información de soporte en línea y redes sociales presentarán una gran ayuda para el crecimiento de Android.

1.12 El entorno en el cual se ejecuta la aplicación de mensajería juega un papel muy importante en el alcance que tengan los dispositivos entre sí para poder comunicarse. Sin embargo, esto no afecta la entrega de la información debido a la calidad de servicio que implementa Bluetooth.

1.13 El tiempo promedio de descubrimiento de dispositivos en la aplicación de mensajería es de 11,29 segundos, el cual está cercano a los 10 segundos definidos por el estándar para la tecnología Bluetooth.

1.14 En un aplicación práctica la distancia de comunicación entre dispositivos es menor a los diez metros indicados por el estándar Bluetooth debido a que los fabricantes incluyen módulos con menor potencia de transmisión para prolongar la duración de la batería.

1.15 Existe una probabilidad del 90% de que se encuentre el servicio durante la búsqueda del servidor por parte del cliente en la aplicación de mensajería.

1.16 Existe total certeza de que los mensajes serán entregados a los destinatarios aún cuando el mensaje haya sido enviado previo el registro del cliente.

1.17 La situación actual y futura del mercado de los dispositivos móviles marca un nuevo comienzo para el desarrollo de aplicaciones en la formación de redes Ad Hoc y comunicaciones alternativas a las redes fijas.

2. Recomendaciones

Existe la factibilidad de que los dispositivos móviles conmuten mensajes entre sí sin la utilización de una red fija, lo cual trae los beneficios que se han discutido previamente. Sin embargo, la tecnología no es todavía la adecuada para alcanzar óptimos niveles de comunicación. Por ello, a continuación se presentan mejoras tanto a nivel de *Hardware* cuanto de *Software* para el mejor desempeño de la aplicación de mensajería.

2.1 Software

La aplicación de mensajería es bastante sencilla y tiene como objetivo identificar claramente los elementos de una comunicación Bluetooth mediante la utilización de J2ME. Por ello, los elementos que se pueden añadir para la mejora del

código son varios, desde la encriptación del canal para la comunicación hasta la asignación de contraseñas para el ingreso de usuarios. Sin embargo, ese no es el objeto a discutir sino la plataforma implementada en los dispositivos móviles en Java y las potenciales mejoras que se podrían dar con la utilización de un sistema operativo más robusto.

Android tiene las mayores perspectivas de crecimiento según todas las empresas de análisis de mercado y tiene como principal ventaja el estar basado en un código robusto de Java y Linux. Entre las ventajas más notables del sistema operativo están la de ser multitarea, poseer total control multimedia, soporte de procesamiento más veloz, reconocimiento de voz y lo más importante es la posibilidad de manejo de hardware de forma nativa mediante Java lo cual permitirá tener control de acceso al medio sin tener que lidiar con la BCC al momento de realizar un programa en Android.

Al utilizar Android se podrían establecer varias mejoras que son nombradas en la tabla 5.27 para mejorar el desempeño de la aplicación. Entre las más importantes están la de poder gestionar de manera directa las conexiones con los clientes de manera directa, respaldar la base de datos de mensajes en la memoria del dispositivo móvil, utilizar el procesamiento de voz para la creación de mensajes mediante el habla del usuario y reducción del tiempo en la búsqueda de dispositivos. El número de conexiones simultáneas aún dependerá del controlador Bluetooth que el fabricante desee instalar.

Tabla 6.1 Potenciales mejoras de la aplicación de mensajería al utilizar Android

Mejora	Causa
Gestión de conexiones de clientes	Acceso a directo al hardware del dispositivo
Respaldar la información en la memoria del dispositivo	Acceso a directo al hardware del dispositivo
Crear mensajes de texto a partir de	Manejo mejorado de procesamiento de voz

comandos de voz	
Reducción del tiempo de búsqueda de dispositivos	Acceso directo a las interrupciones del dispositivo
Aumento de capacidad de procesamiento de mensajes	El sistema operativo puede manejar procesadores de mayor velocidad
Aumento de conexiones simultáneas	Debido a que posee características de punto de servicio básicas *(dependerá del fabricante)*

La principal desventaja de Android es que no implementa la máquina virtual de Java sino una específica denominada Dalvik, la cual presenta varias diferencias con respecto a la primera en las clases que contiene y en los métodos que se utilizan. Si bien no forma parte del estándar, es una muy buena herramienta que puede ser utilizada.

Otro sistema operativo que está en auge es *Meego*, el cual es una opción conjunta aún en fase de desarrollo entre Intel y Nokia. Está basado en Linux pero no tiene soporte Java por lo que no se podría migrar la aplicación de mensajería de manera directa. Su principal ventaja al momento de salir al mercado será la capacidad de manejar procesadores x86 como el ATOM de Intel y no solo ARM, como lo hace actualmente Android. Cabe mencionar que hasta el primer trimestre de 2011 aún no existe un dispositivo en el mercado que tenga este sistema operativo y más aún su desarrollo se ha visto comprometido por el anuncio de Nokia de utilizar el sistema operativo Windows Phone 7.

2.2 Hardware

Existen dos aspectos básicos a considerar, el procesador que se utiliza y el módulo Bluetooth incluido dentro de los dispositivos móviles. Ambos representan un esfuerzo de los fabricantes de incluir elementos que prolonguen la vida de la batería y no reduzcan la movilidad de los celulares.

Acerca del procesamiento, la empresa norteamericana Intel ha anunciado que para fines del 2011 lanzará un dispositivo móvil con su procesador Intel ATOM el cual no comprometerá la duración de la batería [18]. De llegar a darse, los dispositivos móviles tendrían la posibilidad de ejecutar mayor cantidad de tareas en paralelo y soportar sistemas operativos más robustos.

Por otro lado, existen módulos Bluetooth de gran alcance, como los provistos por la empresa AIRCABLE que pueden alcanzar distancias de hasta 100 metros con la utilización de un máximo de 70 mA como pico en transmisión. De igual manera, se garantiza la conexión por medio de siete canales. De esta manera, se podría resolver la limitante de alcance que tienen los dispositivos móviles.

Los elementos adecuados para la configuración de un dispositivo móvil que sea óptimo para la transmisión de información pueden no ser incluidos nunca en un teléfono celular. Por ello, se implementó un dispositivo que cumpla con la mayor cantidad de parámetros expuestos previamente sobre procesamiento y sistema operativo. La base en *hardware* del equipo está compuesto por un procesador Intel Atom mientras que el sistema operativo es una distribución específica de Android para equipos X86 que aún no es definitiva pero es bastante estable. Las características del equipo se muestran en la tabla 5.28, mientras que un costo estimado hasta el primer trimestre de 2011 se muestra en la tabla 5.29.

Tabla 6.2 Características de dispositivo implementado que pueda establecer redes Ad-Hoc

Elemento	Descripción
Sistema Operativo	android-x86 versión1.6 (Donut)
Procesador	Intel Atom D510MO
Radio Bluetooth	Aircable Host XR2
Antena utilizada	Omnidireccional de 12 dBi
Tarjeta Madre	Intel Atom D510MO

Almacenamiento	Samsung 250 Gb

Tabla 6.3 Costo del dispositivo implementado para la formación de redes AdHoc

Elemento	Costo (usd)
Procesador y Tarjeta Madre	**110.00**
Radio Bluetooth	**155.60**
Antena utilizada	**20.00**
Almacenamiento	**59.00**
Case específico	**30.00**
Total	**374,60**

Sobre estos componentes se ha podido trabajar de igual manera que sobre un dispositivo móvil con Android instalado por defecto. La imposibilidad de ejecutar aplicaciones J2ME no ha permitido verificar el funcionamiento de la aplicación de mensajería pero deja abierto el camino para continuar con el estudio de una solución Ad-Hoc rentable y tecnológicamente eficiente.

REFERENCIAS BIBLIOGRÁFICAS.

[1].http://www.google.com.ec/url?sa=t&source=web&ct=res&cd=2&ved=0CA8QFjAB&url=http%3A%2F%2Fsiteresources.worldbank.org%2FEXTEDEVELOPMENT%2FResources%2F20071129_Interview_Hannes_v0.4.doc%3Fresourceurlname%3D20071129_Interview_Hannes_v0.4.doc&ei=vGQiS9HsB8mztgfUu-XJBw&usg=AFQjCNF3nTSoGC7_Az4GiW1E2KDGZLBFXA&sig2=E7GHyOilQW4wVfoXqhZpHQ, Informe Banco Mundial sobre tecnologías móviles.

[2]. http://www.supertel.gov.ec/pdf/estadisticas/sma.pdf, Superintendencia de Telecomunicaciones.

[3]. González Chaparro Diego, Computación Ubicua, 2003, 4 de diciembre de 2009

[4].http://www.amazon.com/Nokia-N900-Unlocked-Computer Touchscreen/dp/B002OB49SW/ref=sr_1_3?ie=UTF8&s=wireless&qid=1260032240&sr=1-3, 4 de diciembre de 2009, Dispositivo Nokia en Amazon

[5]. http://maemo.org/, Sistema operativo MAEMO.

[6]. http://www.android.com/, Sistema operativo Android

[7]. http://es.wikipedia.org/wiki/IPhone_OS, Referencia a sistema operativo IOS de Apple

[8]., http://theappleblog.com/2009/12/03/app-store-world-domination-in-2010-300000-apps-strong/, Comparación entre tiendas electrónicas en línea (Apple Store – Android Store)

[9] http://en.wikipedia.org/wiki/Mobile_development, Desarrollo en plataformas móviles.

[10] Manzoni Pietro y Cano Juan Carlos, Redes Inalámbricas Ad Hoc, 2008. 4 de diciembre de 2009

[11] Mohammad Llyas, “The Handbook of Ad Hoc Wireless Networks”, First Edition, CRC Press, 2003.

[12]. http://laptop.org/en/, One Laptop per Child Project.

[13]. https://www.bluetooth.org/apps/content/Default.aspx?doc_id=224505, Bluetooth SIG, IMS Research,

[14] Bluetooth SIG, Brand Study Wave 6, 2009, 7 de diciembre de 2009

[15]https://www.bluetooth.org/apps/content/default.aspx?doc_id=165348, 7 de diciembre de 2009, Bluetooth SIG, IDC Research.

[16].Bluetooth SIG, http://www.bluetooth.org, 7 de diciembre de 2009.

[17] Bluetooth SIG, SIGnature Magazine SIGnature Magazine, Revista Oficial del SIG de Bluetooth, Q2 2008, Q3 2008 y 1Q 2009, 7 de diciembre de 2009

[18], http://spanish.alibaba.com/product-cgs/bluetooth-module-btm0604c2p-226112146.html, Precios de módulos Bluetooth de proveedores asiáticos.

[19] http://www.jboss.org/, 8 de diciembre de 2009, Aplicación JBOSS.

[20]., http://netbeans.org/ Página oficial de entorno de desarrollo Netbeans.

[21]., http://www.eclipse.org/, Página oficial de entorno de desarrollo Eclipse.

[22]., http://www.getjar.com/, Página con aplicativos en java GETJAR

[23] Gonzáles Chaparro Diego, Computación Ubicua, Universidad Rey Juan Carlos, España, 30 de Junio de 2003

[24] Li, Jinyang, “Capacity of Ad Hoc Wireless Networks”, M.I.T Laboratory for Computer Science.

[25] http://www.ietf.org/dyn/wg/charter/manet-charter.html, Internet Engineering Task Force (IETF).

[25] http://www.symbian.org/, Symbian Community.

[26], www.jneuroengrehab.com, Journal of Neouroengineering and rehabilitation

[27], http://en.wikipedia.org/wiki/Ambient_intelligence, Inteligencia de Ambiente referencia en Wikipedia.

[28], http://www.networkdictionary.com/wireless/WPAN.php, Diccionario en Redes.

[29] http://www.ieee802.org/15/pub/TG4a.html, IEEE 802.15 WPAN Low Rate Alternative PHY Task Group 4a (TG4a).

[30] http://reviews.cnet.com/bluetooth-headset-buying-guide/, Guía de compras CNET.

[31] http://www.bluegiga.com/Bluetooth_access_server_products, fabricante de dispositivos Bluetooth BLUEGIGA

[32], http://www.smarthome.com/, Empresa de domótica Smarthome

[33] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC552302/, 7 de Enero de 2010, National Center for Biotechnology Information

[34] Abderrezak Rached, A secure Architecture for Mobile Ad Hoc Networks, Springer Berlin, 2006.

[35] http://www.bluetooth.com/Bluetooth/Products/low_energy.htm, 8 de Enero de 2010, Bluetooth SIG

[36] Kickbee Inc, http://www.kickbee.net/, 14 de Enero de 2010

[37] Mohammad Llyas, the Handbook of Ad Hoc Wireless Networks, CRC Press, 2003.

[38] http://www.ietf.org/dyn/wg/charter/manet-charter.html, Especificación MANET de IETF

[39] http://tools.ietf.org/html/draft-ietf-manet-fsr-03#section-7.1.3, IETF Tools

[40]http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F4150629%2F4150630%2F04151636.pdf%3Farnumber%3D4151636&authDecision=-203, IEEE Explore.

[41] Maher Hamdi, “GPS-free positioning in mobile ad hoc networks”, Institute for computer Communication and Applications, 2006

[42] Bluegiga, CaseStudy for Xofto, Bluegiga, 2007

[43] http://www.qj.net/psp/homebrew-applications/a-simple-guide-to-help-you-party-in-the-ad-hoc-party.html, PSP Magazine

[44] http://blogs.zdnet.com/ITFacts/?p=6435

[45]http://www.bluetooth.com/Spanish/Technology/Works/Pages/Core_Specification_v30.aspx, Especificación de Core Bluetooth.

[46]http://www.bluetooth.com/Spanish/Technology/Works/Pages/Bluetooth_low_energy_technology.aspx

[47] SIGnature Magazine, Revista Oficial del SIG de Bluetooth, Q2 2008, Q3 2008 y 1Q 2009.

[48] http:// standards.ieee.org/getieee802/download/802-2001.pdf, Estándares IEEE 802.x

[49] FORUM NOKIA, “Games Over Bluetooth Recommendations to Game Developers”, Version 1.0, Nokia, 2003.

[50]http://www.ericsson.com/ericsson/corpinfo/publications/review/1999_04/files/es19990404.pdf, Publicación Ericsson sobre Bluetooth

[51] Kumar Bala, Kline Paul, Thompson Timothy, “Bluetooth Application Programming with the JAVA APIs”, First Edition, Morgan Kaufmann, 2004.

[52]http://developer.symbian.org/wiki/index.php/Symbian_OS_Communications_Programming/10._OBEX, Programación sobre Symbian.

[53]https://www.bluetooth.org/login/default.aspx?ReturnUrl=/technical/assignednumbers/home.htm, Números asignados en Bluetooth.

[54] http://www.fer.hr/_download/repository/ts_101369v060300p.pdf, Digital cellular telecommunications system (Phase 2+) Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol.

[55] https://www.bluetooth.org/Technical/Specifications/enhancements.htm, Bluetooth SIG, Bluetooth Specification V4.0 released on Dec 2009.

[56] https://www.bluetooth.org/apps/directory/, Miembros del directorio del SIG Bluetooth.

[57] http://www.ti.com/ww/en/analog/bluetooth/index.htm, Instrumentos de medición y control en Bluetooth.

[58] http://java.sun.com/javame/index.jsp, Especificación de Java sobre JAVA ME.

[59] http://netbeans.org/, Entorno de Desarrollo Netbeans.

[60] http://jcp.org/en/jsr/detail?id=197, recomendación JSR-197

[61] Coad Peter, Nicola Jill, Pearson Education, First Edition, 2009

[62] https://www.bluetooth.org/technical/assignednumbers/baseband.htm, Números Bluetooth asignados en banda base.

[63] http://msdn.microsoft.com/en-us/library/ms241442%28VS.80%29.aspx, Librerías de desarrollo Microsoft.

[64]http://www.bluecove.org/bluecove/apidocs/javax/obex/ResponseCodes.html, Página de aplicación de desarrollo Bluecove.

[65] http://www.utoronto.ca/webdocs/HTMLdocs/Book/Book-3ed/appb/mimetype.html

[66] Kumar Bala, Kline Paul, Thompson Timothy, “Bluetooth Application Programming with the JAVA APIs”, First Edition, Morgan Kaufmann, 2004.

[67] Hopkins Bruce, Ranjith Antony, “Bluetooth for Java”, First Edition, Apress, 2003.

[68] Mohammad Llyas, “The Handbook of Ad Hoc Wireless Networks”, First Edition, CRC Press, 2003.

[69] http://www.elmundo.es/elmundo/2010/08/13/navegante/1281691500.html, Demanda de Oracle a Google,

[70] us.blackberry.com/developers/, API específico de Blackberry.

[71] http://www.cnnexpansion.com/tecnologia/2011/02/10/nokia-celulares-ventas-gartner-cnn, situación de mercado de dispositivos móviles a 2011.

[72]http://metrics.admob.com/wp-content/uploads/2010/06/May-2010-AdMob-Mobile-Metrics-Highlights.pdf, Mediciones de mercado de ADMOB.

[73] http://en.wikipedia.org/wiki/File:Mobile_os.png, Estructura del sistema operativo Android.

[74]http://blog.symbian.org/2010/12/17/symbian-foundation-is-completing-its-transition-to-a-licensing-body/, Fin de Symbian Foundation

[75]http://www.bbc.co.uk/news/business-12427680, Alianza Nokia-Microsoft

[76]http://www.gartner.com/it/page.jsp?id=1434613, Expectativas de mercado para 2014 según Gartner.

[77]http://en.wikipedia.org/wiki/List_of_digital_distribution_platforms_for_mobile_devices, Situación de mercados de aplicaciones en línea,

[78] http://www.cioperu.pe/articulo/6653/oracle-lanza-framework-de-desarrollo-movil-de-java.aspx, Creación de entorno unificado por ORACLE.

[79]http://developer.motorola.com/platforms/java/, Migración de aplicaciones en Java ME a Android.

[80]http://www.deviceanywhere.com/, Aplicación que incluye una gran base de datos de dispositivos móviles para simulación.

[81]http://gs.statcounter.com/#mobile_os-EC-monthly-200912-201103, Posición de Symbian en Ecuador según Global Stats

ANEXOS

[1] Código fuente de la aplicación de conexiones RFCOMM.

[2] Código fuente de la aplicación de Mensajería

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

PC.- Personal Computer

IETF.- Internet Engineering Task Force

MANET.- Mobile Ad Hoc Network

AODV.- Ad Hoc On Demand Distance Vector

DSR- Dynamic Source Routing for Protocol Mobile Ad Hoc Networks

OLSR.- Optimized LinkState Routing Protocol

TC.- Topology control.

QoS.- Calidad de Servicio Quality of Service

SIG.- Special Interest Group

EDR.- Enhanced data rate

HS.- High Speed

PAN.- Personal Area Network

HCI.- Host Controller Interface

JVM.- Java Virtual Machine

J2SE (Java 2 Estándar Edition)

Wifi.- Wireless Fidelity

LAN.-Local Area Network

PAN.- Personal Area Network

WLAN.- Wireless Local Area Network

BAN.- Body Area Network

WMAN.- Wireless Metropolitan Area Network.

HAN.- Home Area Network

MAC.- Media Access Control

P2P.- Peer To Peer

GPS.- Global Positioning System

DV.- Vector Distancia *(Vector Distance)*

LS.- Estado de Enlace (*Link State)*

DSDV.- Distance Sequenced Distance Vector

OLSR.- Optimized Link State Algorithm

MPR.- Multipoint Relay

TC.- Topology Control

FSR.- Fisheye State Protocol

AODV.- Ad Hoc On Demand Distance Vector Routing

ADV.- Adaptive Distance Vector

TORA.- Temporally Ordered Routing Algorithm

DAG.- Direct Acyclic Graphic

ABR.- Associativity Based Routing Protocol

ZRP.- Zone Routing Protocol

IARP.- IntrAzone Routing Protocol

IERP.- IntErzone Routing Protocol

BRP.- Bordercast Protocol

NDP.- Layer-2 Neighbor Discovery Maintenance Protocol

LAR.- Location Aided Routing Protocol

PSP.- PlayStation Portable

ISM.- Industrial, Scientific and Medical

EDR.-Enhanced Data Rate

GFSK.- Gaussian Frequency Shift Keying

PSK.- Phase Shift Keying

CLKN.- Reloj Nativo

CLK.- Reloj máster Bluetooth

CLKE.- Reloj Estimado Bluetooth

TDD.- Time-Division Duplex

AFH.- Adaptive Frequency hopping

LC.- Link Control

ACL-C.- ACL Control

ACL-U.- ACL User Asynchronous/Isochronous

SCO-S .- User Synchronous Data Logical Link

eSCO-S.- User Extended Synchronous Data Logical Link.

OSI.- Open System Interconnection

PCMCIA.- Peripheral component microchannel interconnect architecture

UART.- universal asynchronous receiver-transmitter

USB.- universal serial bus

HCI.- Host Controller Interface

VoIP.- Voice Over IP

LMP.- Link Manager Protocol

GAP.- Generic Access Profile

SPP.- Serial Port Profile

SDAP.- Service Discovery Application Profile

GOEP.- Generic Object Exchange Profile

DLCI.- Data Link Connection Identifier

MTU.- maximum transmission unit

CDMA.- Code Division multiple access

UWB.- Ultra Wide Band

DAC.- Device Access Code

CAC.- Channel Access Code

IAC.- Inquiry Access Code

DIAC.- Dedicated Inquiry Access Code

GIAC.- General Inquire Access Code

LIAC.- Limited Inquiry Access Code

GCF.- Generic Connection Framework

RMS.- Record Management System

BCC.- Bluetooth Control Center.

SDDB.- Service Discovery Database

SDP.- Service Discovery Protocol

UUID.- Universal Unique Identifier

[1] Código fuente de la aplicación de conexiones RFCOMM.

[2] Código fuente de la aplicación de Mensajería