UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN PROGRAMA J2ME PARA EL MONITOREO DE SISTEMAS EN MOVIMIENTO

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JESÚS ARTURO VERTIZ MÁRQUEZ

PROMOCIÓN
2001-II

LIMA – PERU
2011

DISEÑO E IMPLEMENTACIÓN DE UN PROGRAMA J2ME PARA EL MONITOREO DE SISTEMAS EN MOVIMIENTO

A mis padres, por su constante apoyo y porque siempre creyeron en mi

SUMARIO

Este proyecto consiste en el desarrollo e implementación de un programa J2ME, es decir, una aplicación en Java específica para dispositivos móviles, el cual informará la posición exacta de sistemas en movimiento (persona o alguna otra cosa) que posea un teléfono móvil con dicha aplicación.

Principalmente el proyecto lo componen tres elementos: una aplicación móvil (midlet), una aplicación web (servlet) y un teléfono móvil con receptor GPS interno (también podría ser uno externo). En este informe nos centraremos en la aplicación móvil y su conexión con el servidor y el receptor GPS (Global Positioning System).

La aplicación web nos permitirá guardar los datos de localización en un archivo de texto para su posterior proceso, por ejemplo la visualización en pantalla de la posición en la que se encuentra cada unidad móvil.

La aplicación instalada en el terminal móvil captura los datos generados por el receptor GPS, posteriormente los transmite al servidor web para que este finalmente los pueda almacenar en un fichero.

ÍNDICE

TABLA DE ILUSTRACIONES

PRÓLOGO

En este proyecto se pretende conseguir el monitoreo de la posición exacta de sistemas en movimiento que lleve consigo un dispositivo móvil con nuestra aplicación instalada, por medio del envío frecuente de datos de localización a un servidor web mediante enlaces inalámbricos. Es posible utilizar alguna aplicación cliente que se comunique con nuestro servidor, e interactúe con Google maps por ejemplo, para visualizar la ubicación del móvil dentro de algún mapa.

Nuestro programa J2ME ha sido diseñado e implementado mediante el entorno de desarrollo NetBeans, en el que se han ido desarrollando sus diferentes módulos, probándolos en el simulador y posteriormente su correcto y total funcionamiento con teléfonos Nokia que tengan receptor GPS interno o acepten conexiones de este tipo.

En el sector transportes, por ejemplo, la globalización de los mercados ha estructurado un cambio importante en la relación comercial entre el cliente y las empresas de transporte, ambos necesitan ejercer un mayor control sobre el servicio y los costos asociados, si quieren mantenerse realmente competitivos.

Dentro de este esquema el papel de la tecnología y en especial el de las comunicaciones móviles de datos, adquirirán un rol protagónico al interior de las empresas de transporte. Es justamente en este sentido que nace este proyecto, aprovechando la estrecha relación entre J2ME y las comunicaciones móviles para un monitoreo en tiempo real de sistemas en movimiento.

CAPÍTULO I
PLANTEAMIENTO DEL PROBLEMA

1.1 Objetivos

El presente proyecto tiene como objetivo la información de la posición exacta de sistemas en movimiento (persona o alguna otra cosa) que posean un teléfono móvil con nuestra aplicación móvil instalada.

El proyecto está compuesto básicamente de tres elementos: una aplicación móvil (midlet), una aplicación web (servlet), y un teléfono móvil. Centraremos el presente informe en la aplicación móvil hecho íntegramente en J2ME (la versión móvil de Java) [1], desde su diseño, desarrollo y funcionamiento final.

1.2 Alcances

La globalización de los mercados y la conectividad global que aporta Internet, no solamente han aumentado la competencia entre las empresas productoras de bienes, sino también entre las proveedoras de servicios, como es el caso del transporte. Esto, junto con la creciente informatización de las empresas, ha estructurado un cambio importante en la relación comercial entre el cliente (dueño de alguna carga o interesado en hacer el seguimiento) y el transportista (empresa proveedora de servicios, en este caso de transportes), ya que hoy en día, ambos necesitan ejercer un mayor control sobre el servicio y los costos asociados, si quieren realmente mantenerse competitivos.

Si bien cierto nuestro sistema en movimiento puede ser cualquier persona o algún objeto en general, el presente trabajo tiene mayor relevancia en el sector transporte, justamente debido al factor competitividad que sus protagonistas requieren indicado líneas arriba.

1.3 Monitoreo en tiempo real para sistemas en movimiento

Las comunicaciones móviles de datos aumentan la rentabilidad y la calidad de los servicios de transporte y como consecuencia directa, mejoran y facilitan el intercambio de información entre la empresa de transporte y sus clientes, creándose un círculo virtuoso, en el que ambos, cliente y proveedor, pueden optimizar sus procesos al reducir los tiempos

muertos por imprevistos. Esta optimización mutua, tiende a crear un lazo comercial más permanente entre el empresario y el transportista, que va más allá de una tarifa conveniente o un servicio de calidad.

Ante la necesidad de un servicio de localización y seguimiento de sistemas móviles en tiempo real, se plantea una solución sencilla y no muy cara, aprovechando la potencialidad que existe en J2ME y su estrecha relación con las comunicaciones móviles, que es la esencia del presente proyecto.

1.3.1 Sistemas en Tiempo Real

Un Sistema Tiempo Real es un sistema capaz de responder a un evento dentro de rangos de tiempo precisos, es decir, la respuesta del sistema no solo depende de los valores de salida sino también en el **tiempo** en que esta es aplicada. Esto significa que el sistema debe estar **sincronizado** con el medio exterior que pretende controlar. La siguiente figura 1.1 muestra este comportamiento.

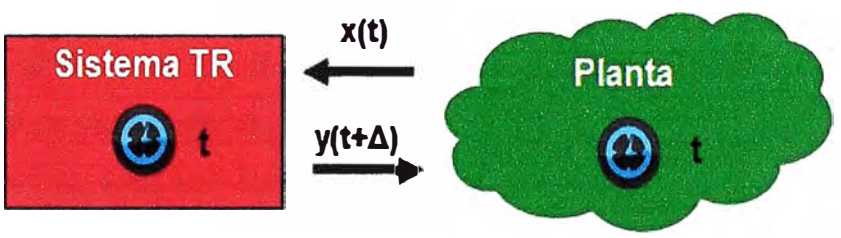


Fig. 1.1 Sistema Tiempo Real

Tiempo real, no es sinónimo de rapidez; esto significa que no es la latencia o la rapidez de la respuesta lo que nos enfoca en un sistema de tiempo real. No es la velocidad de la respuesta del sistema la que lo convierte en un sistema de tiempo real. El objetivo de los sistemas de tiempo real es asegurarse de que la latencia es la adecuada para resolver el problema al cual el sistema está dedicado, y esto pueden ser días, horas, segundos o microsegundos dependiendo de la aplicación.

Para garantizar el comportamiento correcto en el tiempo requerido se necesita que el sistema sea predecible.

1.4 Tecnologías actualmente en uso

Las alternativas tecnológicas para el seguimiento y control de sistemas en movimiento, son variadas y dependen de la naturaleza del negocio en el que se participa, del tamaño de la empresa y de cuán necesario sea hacer gestión en tiempo real de las unidades móviles. Pudiendo ir desde la más compleja, como es la transferencia de datos vía satélite, hasta la más simple, como es el traspaso de los datos registrados una vez que el

móvil regresa a la central. Sea cual sea la elección, el objetivo final es siempre monitorear la ruta que está siguiendo cada unidad móvil, de modo de ejercer un mayor control y gestión sobre ella.

La gran mayoría de las soluciones implementadas, utilizan un receptor GPS (Global Positioning System) o Sistema de Posicionamiento Geográfico, para determinar la posición exacta del móvil que se encuentra en movimiento, para ello utiliza un grupo de satélites, de ubicación siempre conocida, con los cuales se determinan las coordenadas. Las diferencias en estos casos, surgen solo en la forma y periodicidad con que el móvil transmite los datos a la central de monitoreo, en este caso un servidor web. Una alternativa al uso de GPS, es el rastreo radial de vehículos, utilizando ondas de radio de espectro expandido (*spread spectrum*), que puede ser una solución viable y a más bajo costo que el GPS, aunque requiere de una red de antenas dispuestas convenientemente, lo que lo hace espacialmente aplicable para el tracking o monitoreo del transporte público o de flotas con ruta fija.

1.5 Descripción de la interfaz

Nuestra aplicación móvil (midlet) inicialmente nos pide confirmar la dirección del servidor web al cual se va a conectar, así como el periodo T en segundos para hacer las capturas de localización. Una vez introducidos estos datos iniciales, el midlet empieza automáticamente la captura de datos del receptor GPS para mostrarlos en una segunda pantalla, luego los enviará al servidor web especificado. Repite esta operación según el periodo T en segundos.

1.6 Funcionamiento

Nuestro midlet tiene que ser ejecutado o iniciado manualmente, entonces este quedará funcionando en segundo plano, es decir, una vez iniciado, este no necesita la intervención del portador o usuario del móvil.

Automáticamente tratará de conectarse al sistema GPS para poder conocer su posicionamiento inicial. Una vez hecha la primera captura de datos y haberlos mostrado en pantalla, intentará de enviarlos automáticamente al servidor web. Aquí, nuestra aplicación web (servlet) es el encargado de la recepción y almacenaje de dichos datos. Si en caso no encontrase respuesta del servidor, el midlet obvia el envío de datos al mismo, hasta la próxima captura de localización que tendrá lugar después de un tiempo determinado T, introducido inicialmente.

También es posible enviar otros tipos de información en caso se requiera, como la distancia recorrida o su velocidad.

El proceso de comunicación entre el midlet y el servlet a grandes rasgos lo podemos apreciar en la siguiente fig. 1.2. La secuencia más detallada de dicho proceso se indicara en el siguiente subcapítulo.

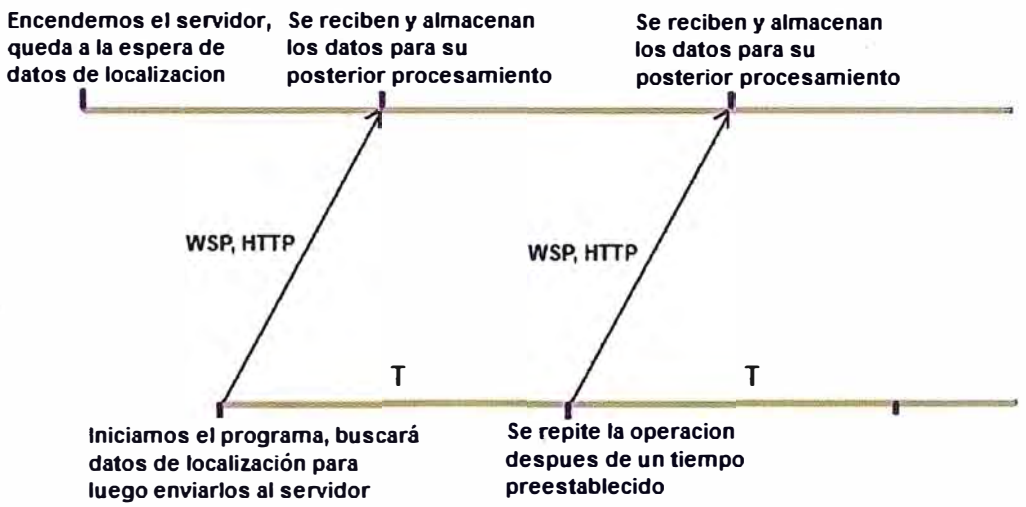


Fig. 1.2 Funcionamiento del midlet

1.7 Secuencia de envío y recibo de información

Un detalle de las aplicaciones inalámbricas es que la comunicación entre el teléfono móvil (nuestro midlet) y el servidor web (nuestro servlet) no se establece directamente, sino a través de lo que es denominado el WAP Gateway [2], cuya secuencia de operación se muestra en la fig. 1.3, y los pasos que a continuación se indican son mostrados en la misma figura.

1. El dispositivo móvil envía una petición URL al WAP Gateway mediante el protocolo WSP [2].
2. El WAP Gateway decodifica el mensaje (en formato binario) de petición y lo traduce a un formato http mediante una tabla de mapeo.
3. El WAP Gateway crea una conexión con el web server y envía una petición http hacia este.
4. La petición HTTP es procesada por el servidor Web. Puede tratarse de una petición estática o puede intervenir un CGI, ASP (Active Server Pages) o JSP.
5. El servidor web retorna una respuesta http, el cual contiene data o resultados de alguna aplicación.
6. El WAP Gateway codifica la respuesta y lo traduce en formato binario WSP usando la tabla de mapeo.

7. El WAP Gateway crea una conexión, y una respuesta WSP conteniendo WML es devuelto al móvil.

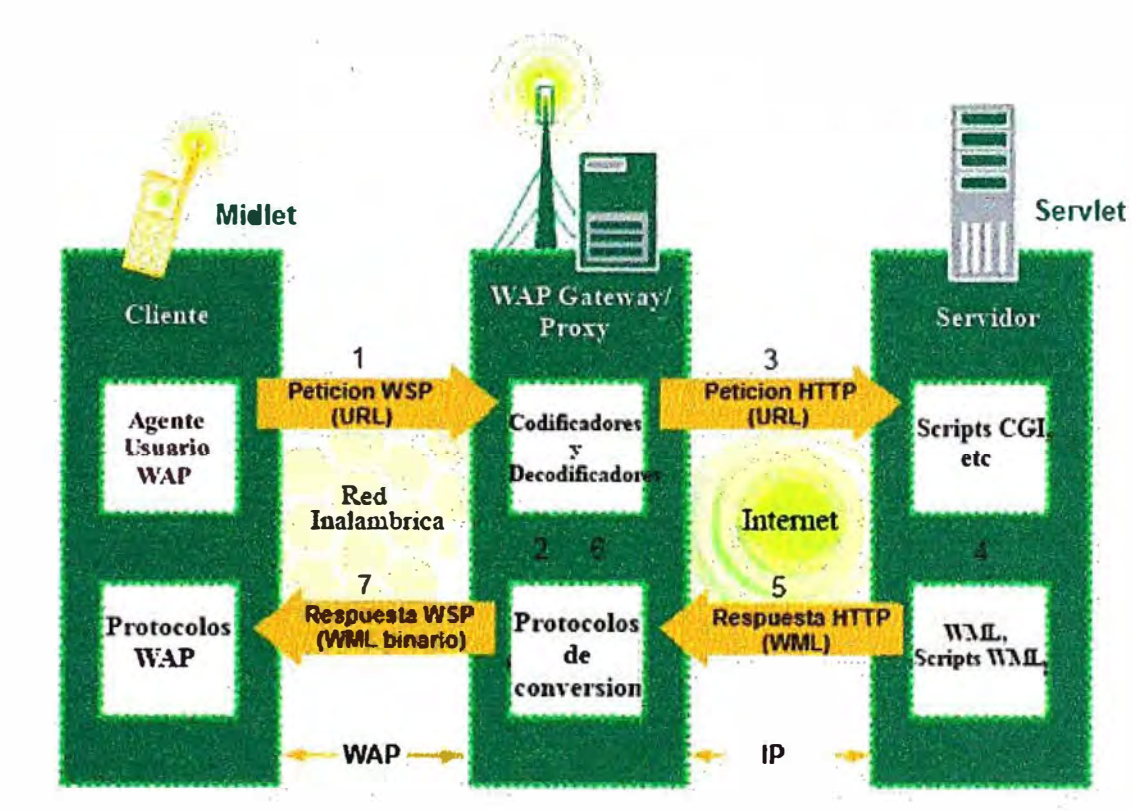


Fig. 1.3 Secuencia de envío y recibo de información

CAPÍTULO II
APLICACIONES J2ME (MIDLETS)

2.1 Introducción sobre Java

Java es un lenguaje de programación orientado a objetos desarrollado por la empresa Sun Microsystems en 1995 y que ha extendido ampliamente sus aplicaciones en el World Wide Web. Es un lenguaje de alto nivel y propósito general similar a C++, con marcadas características de seguridad y transportabilidad. Este lenguaje define una máquina virtual independiente de la plataforma donde se ejecuta, que procesa programas, llamados *Applets*. Además, debido al modo de ejecución de los *Applets,* este lenguaje es muy seguro frente a la presencia y ataque de virus informáticos.

La plataforma Java ha sido dividida en varias ediciones distintas según sus diferentes objetivos: *Java 2 Standard Edition*, orientada al desarrollo para computadores personales y aplicaciones en general, *Java 2 Micro Edition*, orientada al desarrollo para artículos pequeños y móviles como PDAs, *Java 2 Enterprise Edition*, orientada al desarrollo de aplicaciones corporativas, y *Java Card*, orientada a aplicaciones que se ejecutan en tarjetas de crédito con chip. Estas ediciones de Java las podemos ver en la figura 2.1.

2.2 Definición de J2ME

La plataforma J2ME (**Java 2 Micro Edition**) es una familia de especificaciones que definen varias versiones minimizadas de la plataforma Java 2; estas versiones minimizadas pueden ser usadas para programar en dispositivos electrónicos; desde teléfonos móviles, en PDAs, hasta en tarjetas inteligentes, etc. Estos dispositivos presentan en común que no disponen de abundante memoria ni mucha potencia en el procesamiento, ni tampoco necesitan de todo el soporte que brinda el J2SE, (la plataforma estándar de Java usada en sistemas de escritorio y servidor – Java 2 Standard Edition). En conclusión, J2ME es la versión de Java orientada a los dispositivos móviles.

Debido a que los dispositivos móviles tienen una potencia de cálculo baja e interfaces de usuario pobres, es necesaria una versión específica de Java destinada a estos

dispositivos, ya que el resto de versiones de Java, no encajan dentro de este esquema. J2ME es por tanto, una versión "reducida" de J2SE.

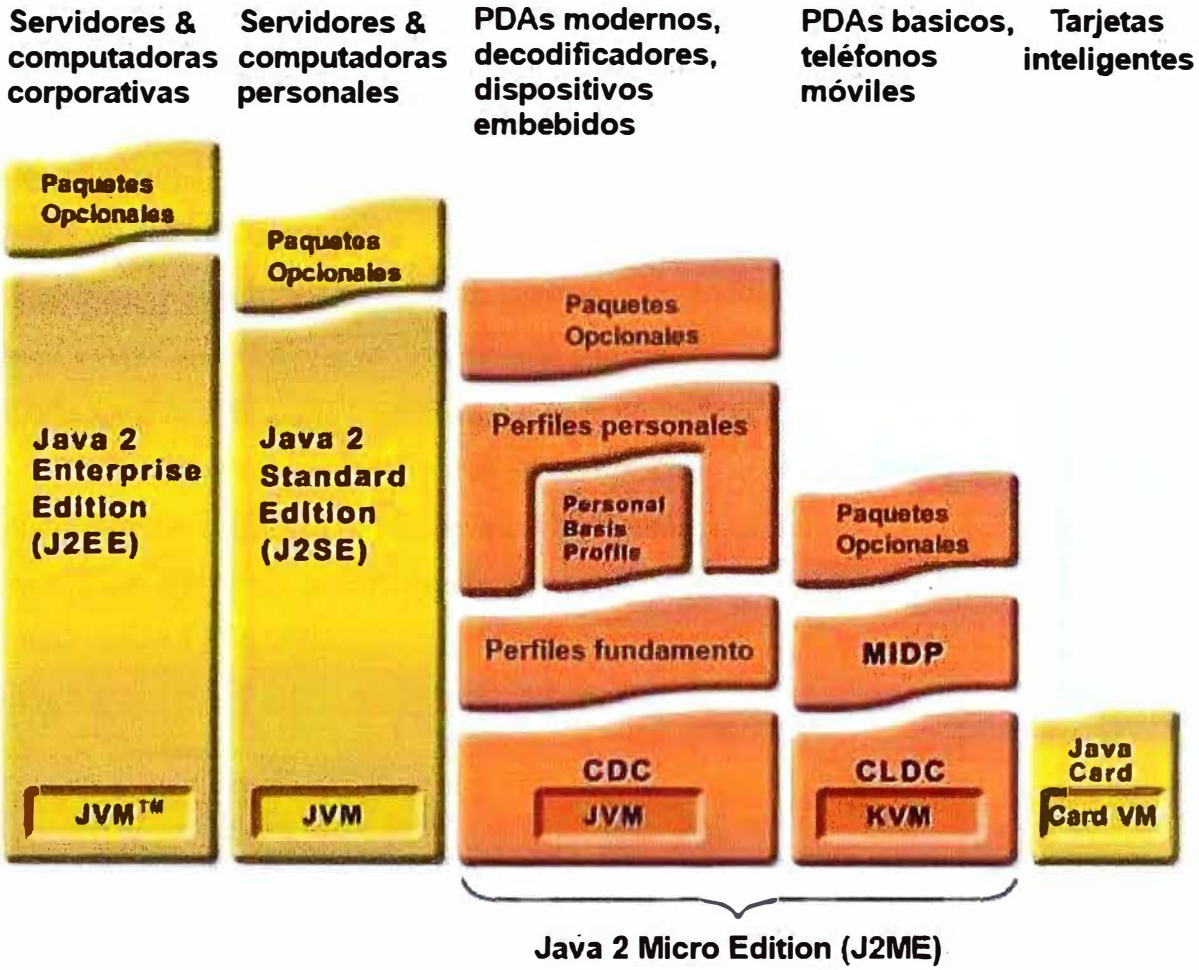


Fig. 2.1 Las diferentes ediciones de Java

J2ME contiene una mínima parte de las APIs (Application Programming Interface – Interfaz de Programación de Aplicaciones) de Java. Esto es debido a que la edición estándar de APIs de Java ocupa 20 MB, y los dispositivos pequeños disponen de una cantidad de memoria mucho más reducida. En concreto, J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util. Esta parte de la API que se mantiene fija forma parte de lo que se denomina **configuración**, la cual se detalla más ampliamente en la siguiente subsección (2.3). Otras diferencias con J2SE vienen dadas por el uso de una maquina virtual distinta de la clásica JVM (Java Virtual Machine) denominada KVM (Kylo Virtual Machine). Esta KVM tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la JVM [3]. Las diferentes ediciones de Java con sus respectivos objetivos en el mercado lo podemos apreciar en la figura 2.2.

2.3 Máquinas virtuales de J2ME, Configuraciones y Perfiles

La edición micro de Java se compone, además del lenguaje, de una máquina virtual, configuraciones, perfiles y paquetes adicionales. La máquina virtual es la base de la plataforma. La maquina virtual es personalizada para un sistema operativo de un dispositivo en particular, es el intérprete del lenguaje y sobre la cual se han de ejecutar las aplicaciones, también sobre esta máquina virtual corren las configuraciones (CDC y CLDC, ver 2.3.1 y 2.3.2), las cuales incorporan Apis básicas para la creación de aplicaciones y sirven de soporte a los perfiles. Los perfiles incluyen la mayor parte de las clases y Apis que se van a utilizar en la programación, como pueden ser instrucciones de entrada y salida o de inicio y terminación de la aplicación.

Los paquetes adicionales son aquellos que la especificación de la tecnología inalámbrica Java (JSR185) [4] no establece como obligatorios para incorporar en los dispositivos. Ejemplos de esto pueden ser las Apis de mensajes inalámbricos (WMAPI) y de multimedia (WMAPI).

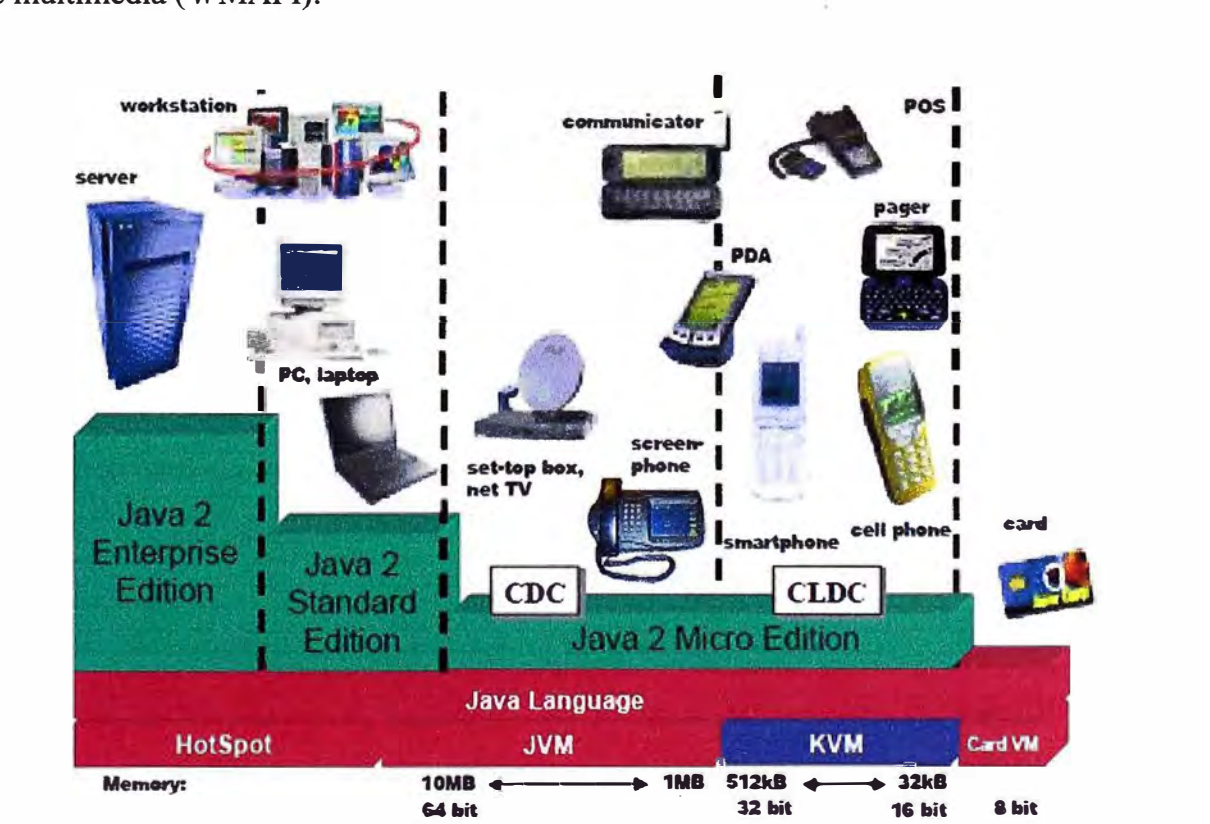


Fig. 2.2 Ediciones Java y sus objetivos en el mercado

2.3.1 CDC, Configuración de dispositivos con conexión (Conected Device Configuration)

Está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, electrodomésticos, sistemas de navegación de automóviles y decodificadores de

televisión. CDC como ya se mencionó antes utiliza una Máquina Virtual de Java similar a la de J2SE, pero con algunas limitaciones en el apartado gráfico y de memoria del dispositivo. Esta máquina virtual es la que se ha visto como CVM (CDC Virtual Machine). La CDC está enfocada a dispositivos con las siguientes capacidades:

• Procesador de 32 bits.

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

• Poseer la funcionalidad completa de la Máquina Virtual Java2.

• Conectividad a algún tipo de red.

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

La CLDC está orientada a dispositivos dotados de conexión pero con limitaciones en cuanto a capacidad gráfica, computacional y memoria. Unos ejemplos de estos tipos de dispositivos son los teléfonos móviles, las PDAs y los organizadores personales. CLDC es la base para que los perfiles (como MIDP o PDAP) funcionen, proveyendo las Apis básicas y la máquina virtual (KVM – Kylo Virtual Machine). CLDC está diseñada para equipos microprocesadores RISC o CISC de 16 a 32 bits y con una memoria mínima de 160 KB para la pila de la tecnología Java.

2.3.3 Perfiles en J2ME

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

El perfil establece unas APIs que definen características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración.

Hay que tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad especifica. Anteriormente se vio que para una configuración determinada se usaba una Maquina Virtual Java específica. Con los perfiles ocurre algo similar, existen unos perfiles que se usarán sobre la configuración CDC y otros sobre la CLDC.

2.4 MIDP (Mobile Information Device Profile)

En este apartado veremos en detalle el **MIDP (Mobile Information Device Profile)** para la plataforma **J2ME**. Nuestro objetivo es ver el API para poder escribir programas en Java para ser usados en dispositivos móviles. Estos programas para dispositivos móviles se denominan midlets [5].

MIDP está diseñado para trabajar sobre CLDC (Connected Limited Device Configuration) v1.0, v1.1 y probablemente en nuevas revisiones. Aunque se tendrá en cuenta que la mayoría de las características de MIDP v2.0 se basaran en CLDC v1.1

Ejemplos de dispositivos móviles que soportan las características básicas para los midlets son los nuevos modelos de teléfonos móviles o PDAs. Aunque un PC puede cumplirlos también.

MIDP nos proporciona un perfil que se apoya en CLDC y que nos va a proporcionar los paquetes y clases necesarios para el desarrollo de nuestras aplicaciones. MIDP está orientado principalmente a teléfonos móviles, aunque existe una implementación para PalmOS (versión 3.5 y superiores) y PocketPC, por lo que es también utilizable en casi cualquier PDA.

Se denomina **midlet** a las aplicaciones Java realizadas usando la especificación de MIDP, es decir, un midlet será una aplicación que puede usar la funcionalidad aportadas por MIDP y por CLDC. Un midlet siempre estará compuesto, al menos, por una clase principal que hereda directamente de la clase *javax.microedition.midlet.MIDlet*.

2.4.1 El API de MIDP 2.0

Para realizar nuestros midlets además de CLDC tendremos que conocer el API que nos proporciona MIDP. El API se compone de las clases y paquete heredados de CLDC y de otra serie de paquetes situados en la jerarquía *javax.microedition.*

El API de MIDP 2.0, se compone de los siguientes paquetes:

Paquete de Ciclo de vida de las Aplicaciones (*javax.microedition.midlet*): Este paquete permite a las aplicaciones MIDP (midlets) interactuar con el entorno, sobre el cual la aplicación se está ejecutando.

Paquete de Interfaz (*javax.microedition.lcdui*): Este es la parte del API dedicada al interfaz de usuario (UI –User Interfaz). Proporciona un conjunto de características para la implementación de interfaces en MIDP.

Paquete de Red (*javax.microedition.io*): MIDP proporciona soporte de red basándose en CLDC.

Paquete de Juegos (*javax.microedition.lcdui.game*): Este es la parte del API dedicada a juegos. Proporciona una serie de clases que permiten construir juegos rico en contenidos para dispositivos móviles. No presente en MIDP 1.0 .

Paquete de Clave Pública (*javax.microedition.pki*): Certificados usados para autentificar información proveniente de conexiones seguras. No presente en MIDP 1.0 .

Sonido:

javax.microedition.media: El API Media de MIDP 2.0 es un bloque directamente compatible con la especificación Mobile Media API (MMAPI). MMAPI extiende la funcionalidad de J2ME proporcionando audio, video y otras características multimedia. Es un paquete opcional, simple y ligero, que también permite acceder a los servicios multimedia nativos de nuestro dispositivo móvil (como las cámara de fotos de los móviles).No presente en MIDP 1.0.

javax.microedition.media.control: Este paquete define los tipos de control específicos que pueden ser usados en el reproductor (Player) de la API Media. No presente en MIDP 1.0.

Paquete de Persistencia (*javax.microedition.rms*): MIDP proporciona este mecanismo para que los midlets guarden persistentemente datos y posteriormente puedan recuperarlos.

Paquetes principales:

java.lang (CLDC): Las clases del lenguaje incluidas en el perfil provenientes de J2SE.

java.util (CLDC): Las clases de utilidades incluidas en el perfil provenientes de J2SE.

2.4.2 Midlet: Ciclo de vida

En la sección 2.4 dijimos que un midlet siempre heredará de la clase ***javax.microedition.midlet.MIDlet***. Los métodos de esta clase permiten a nuestra aplicación crear, empezar, parar y destruir un midlet, estos métodos son la interfaz del Midlet, que va a permitir a nuestro dispositivo poder manejar múltiples midlets, sin tener que estar todos ejecutándose en el mismo entorno. El sistema de nuestro dispositivo puede seleccionar que Midlet está activo usando los métodos correspondientes para empezar o pausar.

El ciclo de vida de un midlet se compone de lo siguientes estados: **Pausado**, **Activo** o **Destruido**. Sólo puede estar en un estado a la vez. La figura 2.3 muestra como se pasa de un estado a otro.

Cuando un midlet se carga en memoria, inicialmente pasa al estado Pausado, entonces se realiza la inicialización de la clase (método *startApp()*). Si el midlet lanza una excepción durante la ejecución de su constructor, se destruye. El midlet puede pasar de

Activo a Pausado, cuando, por ejemplo, recibimos una llamada en nuestro móvil; es el sistema quien pasa nuestro midlet de Activo a Pausado y viceversa. Un midlet puede ser lanzado y parado todas las veces que queramos, pero sólo puede ser destruido una vez.

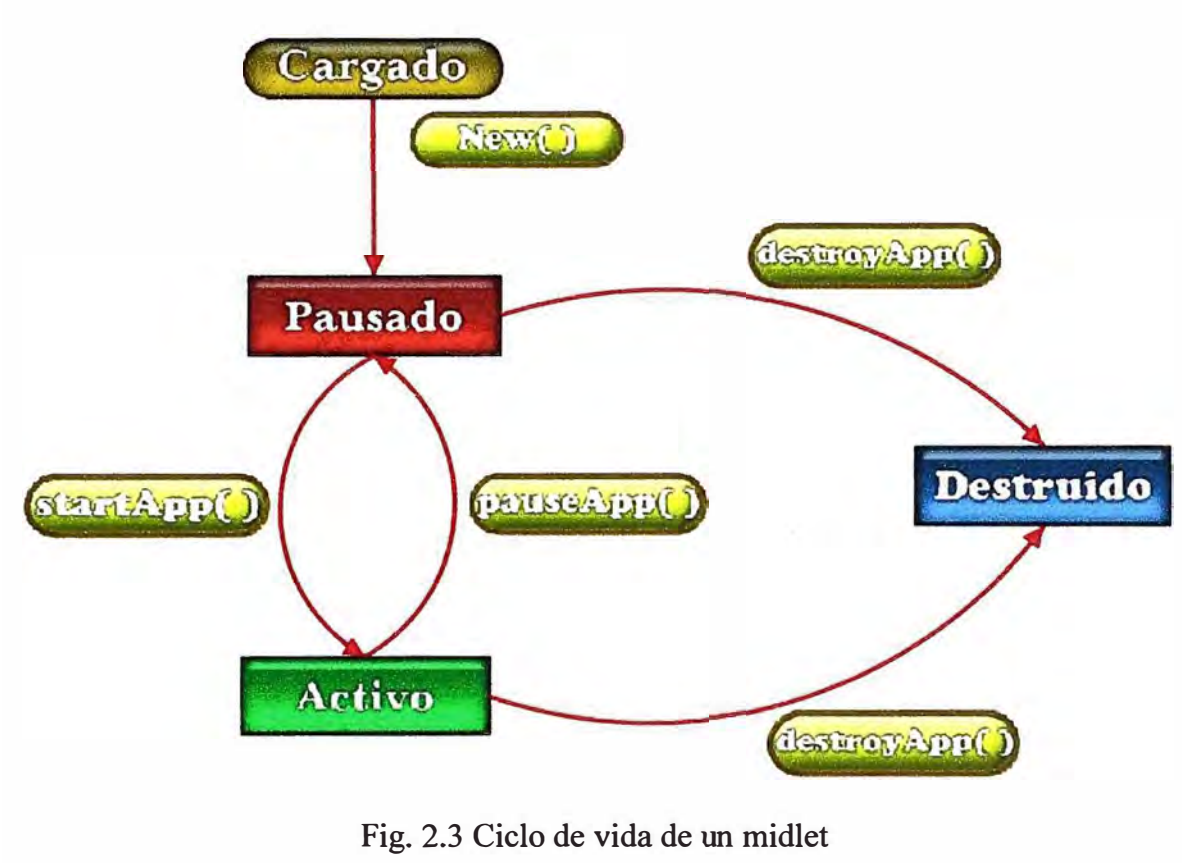


Fig. 2.3 Ciclo de vida de un midlet

2.4.3 Almacenamiento Persistente

El perfil MIDP cuenta con un sistema de administración de registros (RMS) para almacenar datos de forma persistente; es decir, permite guardar información que se conservará en la memoria física del dispositivo después que la aplicación finalice su ejecución, y estarán disponibles cuando el MIDlet sea ejecutado nuevamente.

El RMS se estructura en base a RecordStores equivalentes a una tabla de un RDBMS. Cada RecordStore es una colección ordenada de registros de dos campos, un identificador asignado por el RMS y un campo binario de longitud variable.

2.4.4 Conectividad

La conectividad es un aspecto importante dentro de J2ME, por ello el perfil MIDP cuenta con acceso a variados tipos de conexiones que incluyen sockets, HTTP, SMS, Bluetooth, entre otros. El conjunto de clases que permite realizar conexiones se denomina Generic Connection Framework (GCF).

Para abrir cualquier tipo de conexión se usa la instrucción *Connector.open* (url), y el GCF retornará un objeto del tipo adecuado para manejar la conexión. La conexión que se

usó en el proyecto es el http (área dominada por protocolos de internet), el cual es creado usando una url de conexión de tipo *http://servidor:puerto*. El objeto devuelto por la llamada a *Connector.open* tiene métodos para efectuar peticiones y obtener información de la conexión.

CAPÍTULO III
ENTORNO DE DESARROLLO

3.1 Definición

Un entorno de desarrollo integrado o, en inglés, Integrated Development Environment ('IDE'), es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

3.2 Netbeans

Hemos usado exclusivamente el entorno de desarrollo NetBeans para la implementación de nuestra aplicación, programada completamente en java, aunque se podría haber usado cualquier otro IDE que soporte la programación de aplicaciones para móviles (versión J2ME de Java).

El IDE NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

Estas han sido elegidas por su facilidad de uso y su integración ya que contienen todos los elementos necesarios directamente integrados, consiguiendo una comodidad hasta ahora inexistente en el resto de entornos probados. Netbeans es un IDE creado por Sun para programar en Java. Da la posibilidad de crear aplicaciones para todas sus plataformas entre ellas J2ME, ya que desde la misma página de Sun se puede descargar la herramienta con diversos módulos que se integran con la misma en un solo click.

Wireless Toolkit también pertenece al conjunto de aplicaciones de Sun y es un emulador muy potente y coherente con el entorno de programación CLDC/MIDP. Wireless Toolkit es una herramienta eficiente y necesaria para ejecutar y probar nuestras aplicaciones J2ME.

3.2.1 Instalación de Netbeans IDE

Directamente desde la página de la herramienta es posible descargarla, este IDE es totalmente gratuito y su página oficial es: *http://www.netbeans.org.* Al momento del desarrollo del presente proyecto, la versión disponible era la 6.9.1, vea la fig. 3.1.

Es una creación de Sun y hoy por hoy una de sus puntas de lanza ya que pretenden que este IDE acabe sirviendo de base para cualquier programador que pretenda crear una aplicación con Java, PHP, C/C++ y otros. En la misma página se encuentran todos los módulos (ad-ons) necesarios. En este caso la dirección web de lo que se necesita y sobre lo que versa la explicación es la siguiente: *http://java.sun.com/javase/downloads/netbeans.html.*

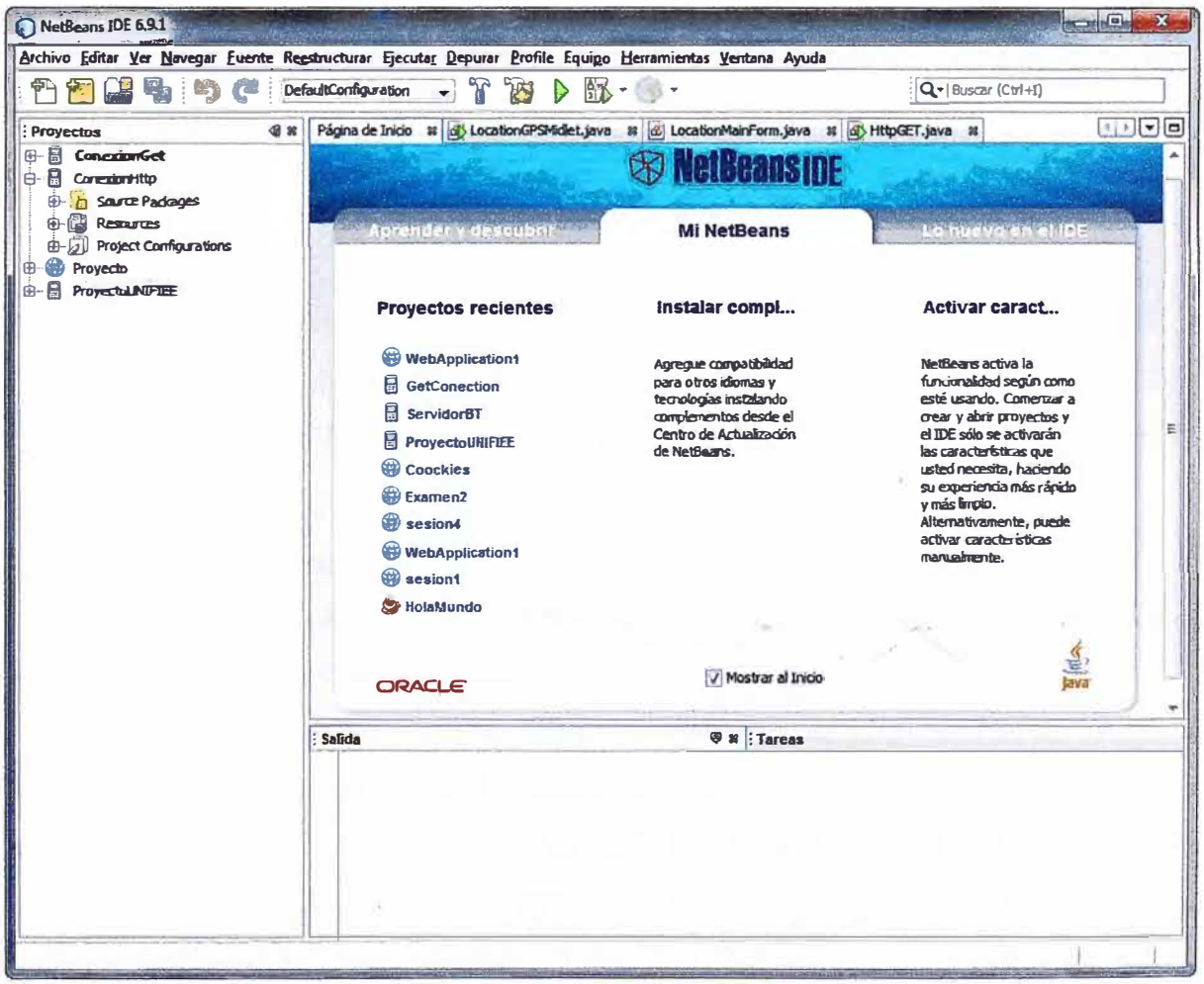


Fig. 3.1 Netbeans IDE

3.3 Emulador Wireless Toolkit

J2ME Wireless Toolkit es un sencillo IDE para el desarrollo de midlets, que nos permite compilar, preverificar y ejecutar el código en emuladores de teléfonos móviles y pagers. Visualizaremos un pequeño midlet ejemplo, el HelloWorld, a través de este emulador. También veremos una breve introducción a los componentes de interfaz de usuario que proporciona MIDP.

Una vez instalado el entorno de desarrollo Netbeans, se instala este módulo que hace posible el fácil desarrollo y demás aplicaciones móviles: *http://www.oracle.com/technetwork/java/download-135801.html.* Se muestra la interfaz grafica del emulador en la siguiente figura 3.2.

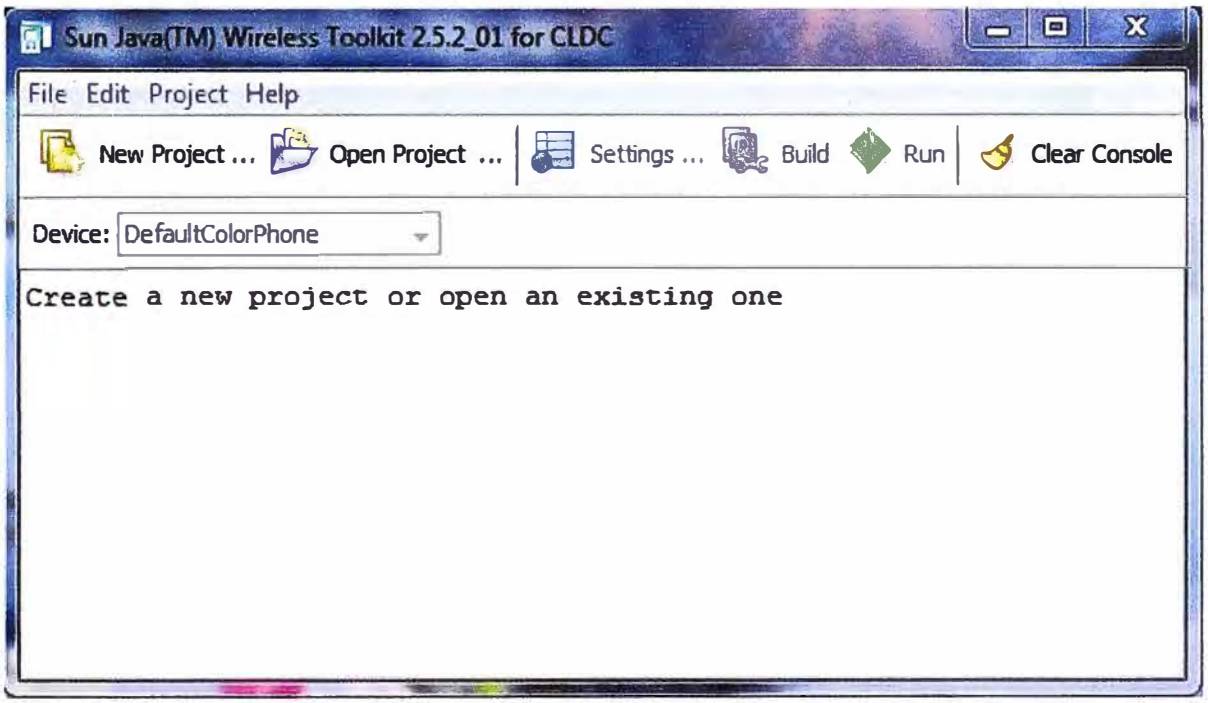


Fig. 3.2 Emulador Wireless Toolkit

3.4 Desarrollo de aplicaciones en Netbeans IDE

Como se puede observar, en esta herramienta es posible realizar todas las fases de desarrollo de aplicaciones MIDP.

En el cuadro superior derecho dispone de un editor de texto totalmente integrado para crear el código fuente (pestaña Source). Además de dos pestañas llamadas Screen Design y Flow Design, figuras 3.3 y 3.4, que sirven para poder diseñar aplicaciones mediante el agregado de elementos directamente sobre la pantalla del terminal y diseño mediante diagramas de flujo respectivamente. Las figuras que vienen a continuación reflejan estas pestañas.

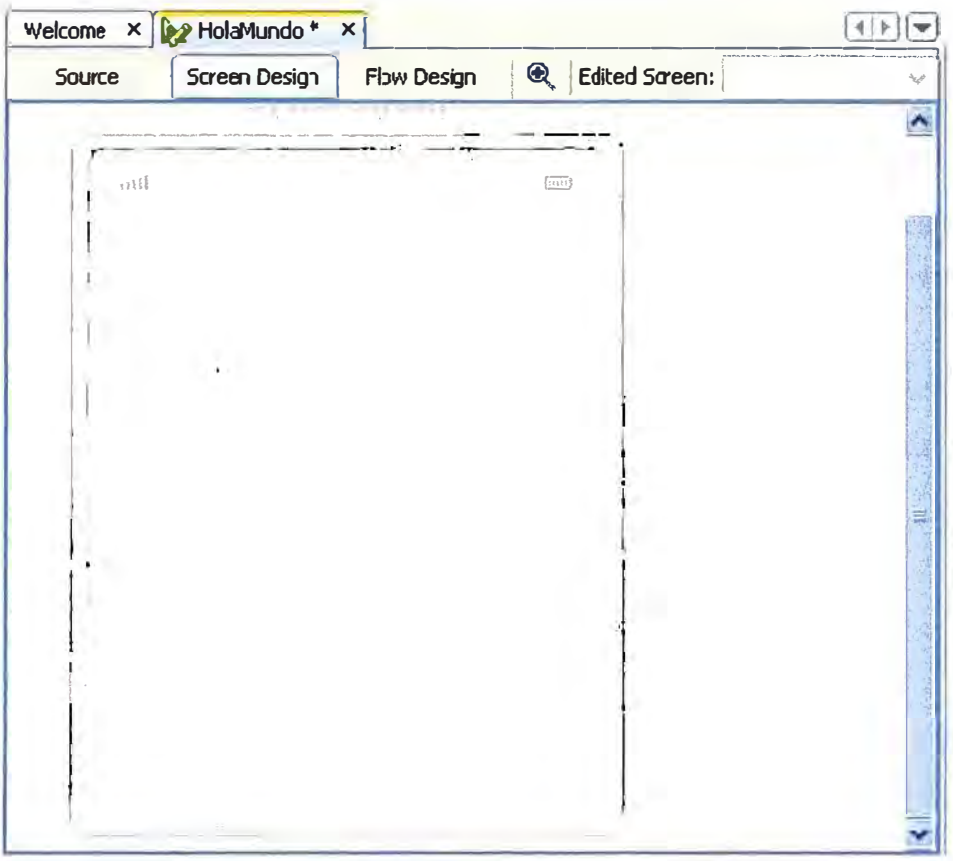


Fig. 3.3 Diseño mediante un entorno grafico

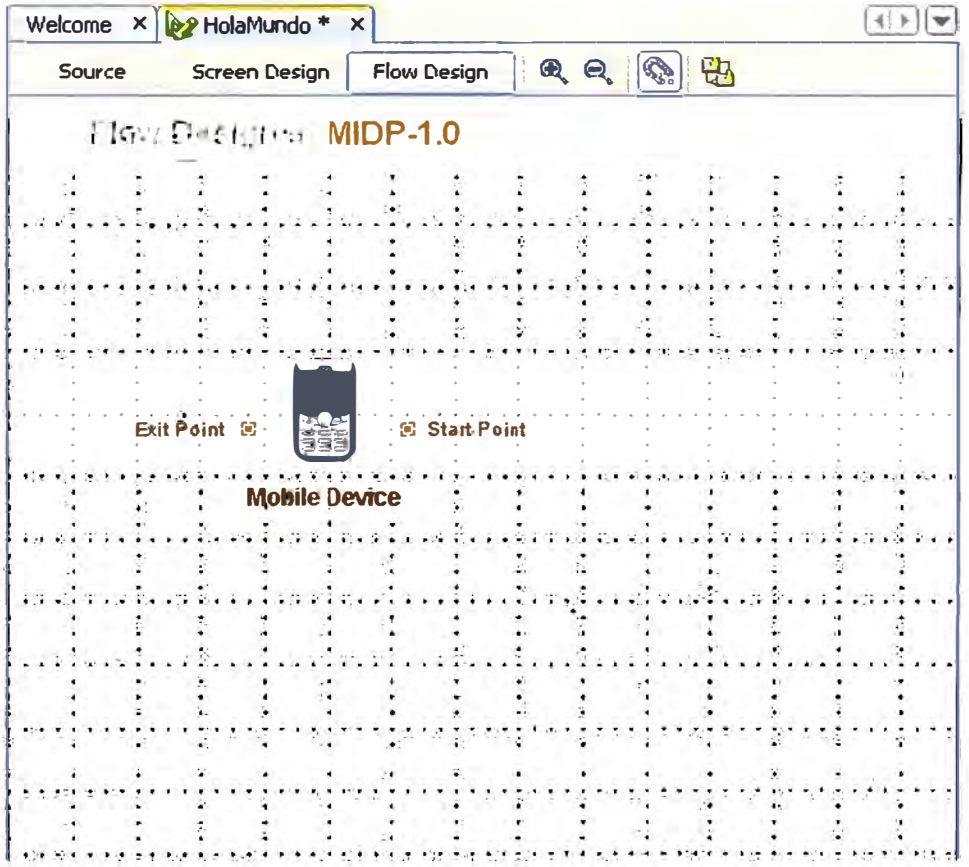


Fig. 3.4 Diseño mediante diagrama de flujos

Una vez creado el código de la aplicación es posible su compilación ya que el entorno trae un compilador (jdk) integrado en el mismo con todas las librerías necesarias. Para compilar cualquier archivo simplemente habrá que pulsar con el botón derecho del ratón sobre el archivo a compilar o proyecto (ya que es posible compilar archivos sueltos o proyectos completos) en la ventana superior izquierda llamada projects y elegir entre las opciones del desplegable la que más convenga. Según el tipo de archivo sobre el que se pinche (click derecho) habrán tres opciones:

(a) Si se pincha sobre un archivo (.java) se deberá elegir la opción Compile File pulsar F9.

(b) Si se pincha sobre un paquete (package) se deberá elegir la opción Compile Package o pulsar F9, como se inndica en la siguiente figura 3.5.

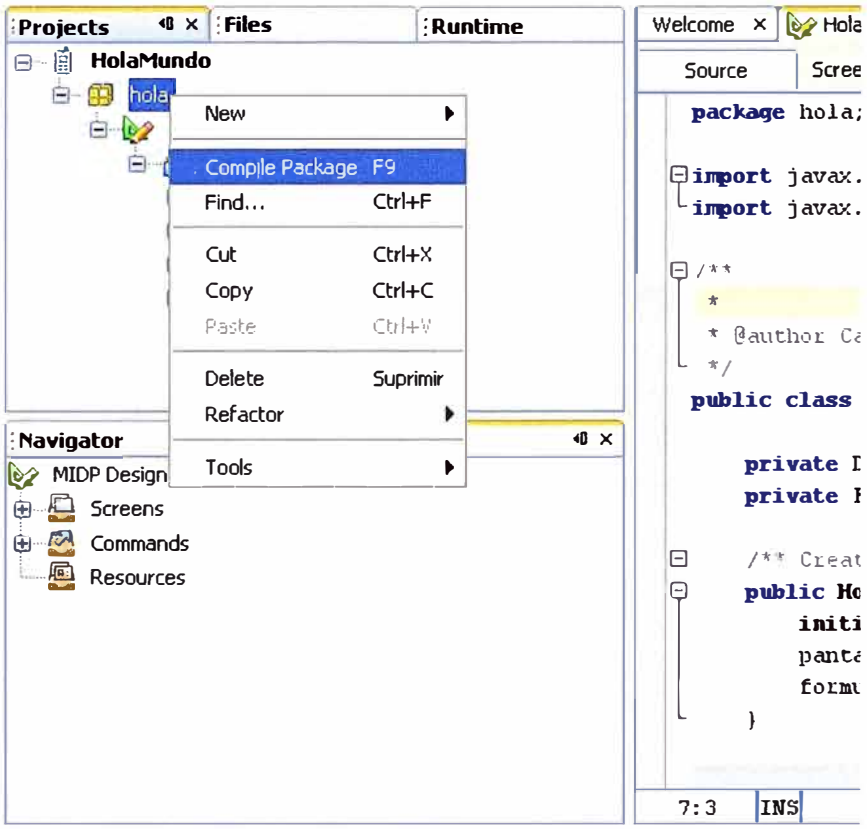


Fig. 3.5 Compilar paquete

(c) Si se pincha sobre un proyecto (19roject) se elegirá la opción Build Project, vea la siguiente figura 3.6.

Una vez que es ejecutada la compilación de un proyecto completo (Build Project) el proceso de empaquetado (generación archivos JAR y JAD) también es automático. Netbeans se encarga de generarlos y guardarlos en un subdirectorio perteneciente al directorio creado para el proyecto llamada dist.

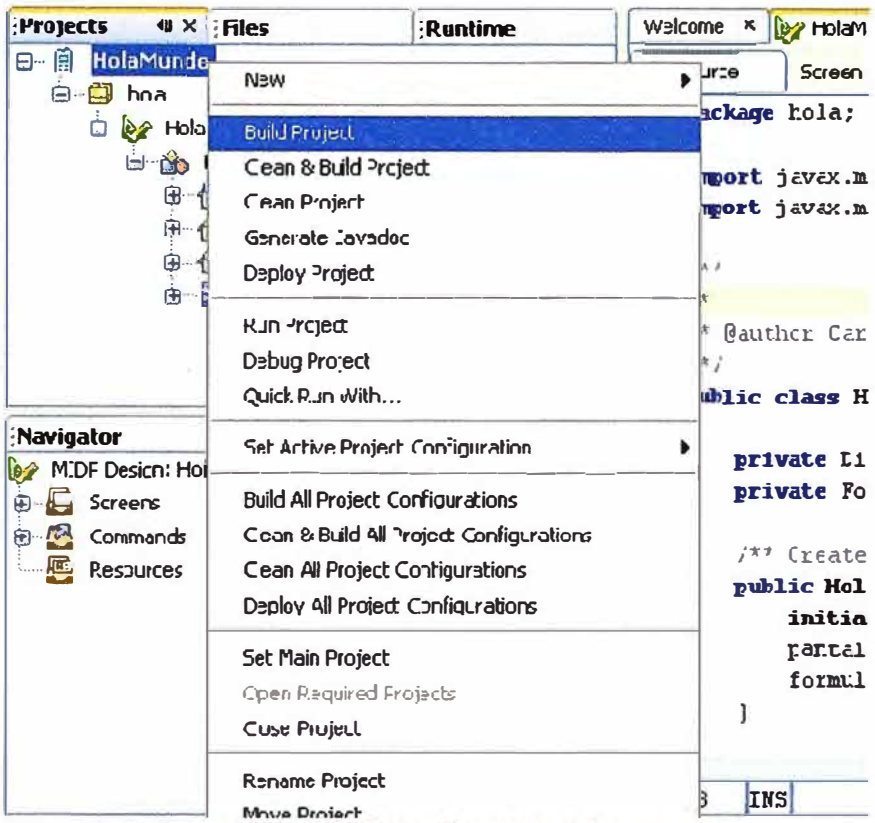


Fig. 3.6 Compilar proyecto

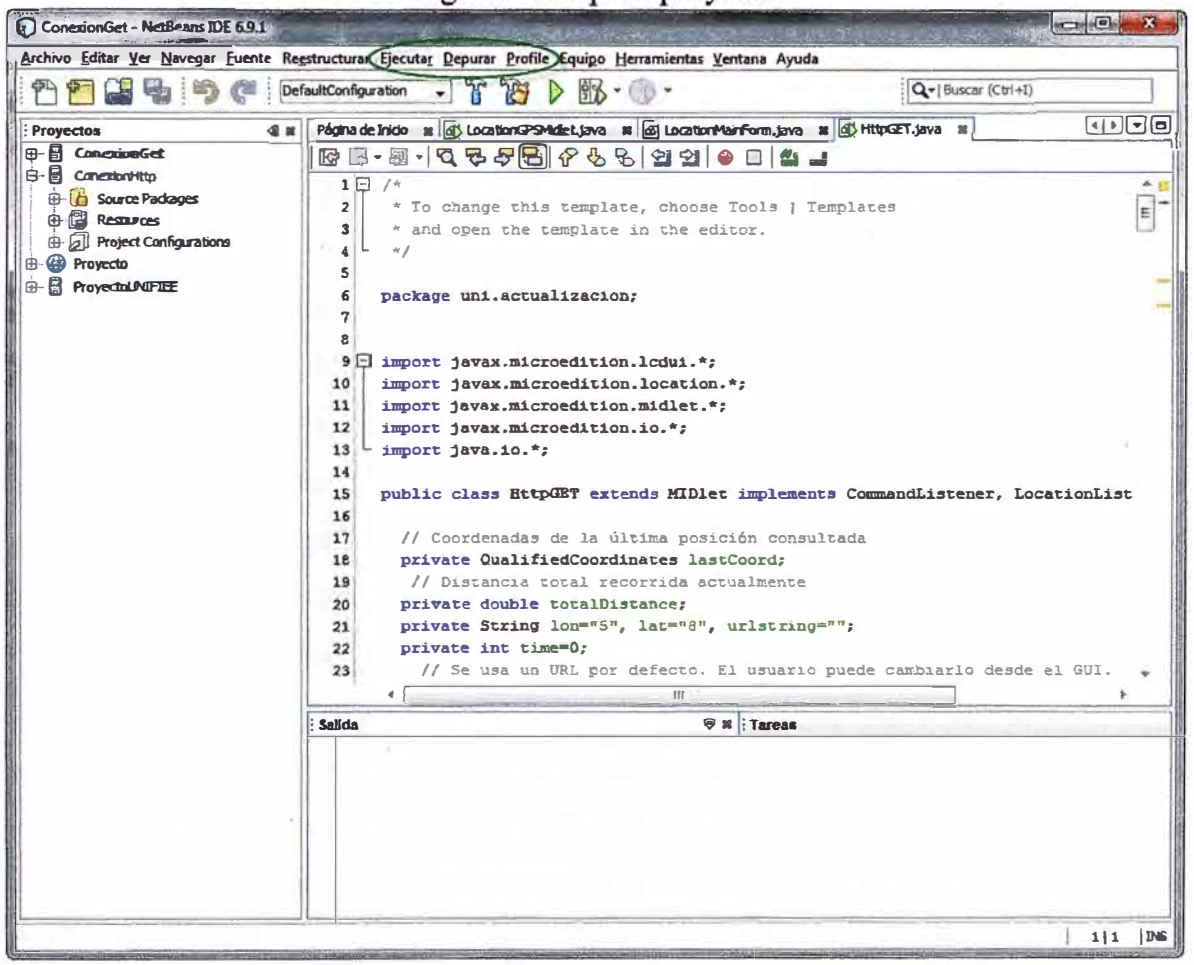


Fig. 3.7 Ejecución y depuración

Las fases de ejecución y depuración las podemos encontrar en tres pestañas situadas en la parte superior llamadas Ejecutar, Depurar y Profile (observar figura 3.7). También se dispone de una barra específica de los pasos más usados para dichas fases, esta barra se encuentra ubicada en la parte superior izquierda (figura 3.8).

Fig. 3.8 Barra de Emulación

La fig. 3.8 corresponde a la barra de emulación y depuración que había sido nombrada anteriormente. A continuación se explicara la función de cada uno de los ítems de esta barra:

El primer ítem, que es el desplegable, corresponde a la configuración con la cual se va a llevar a cabo el proyecto (por defecto este valor es DefaultConfiguration).

El primer y el segundo botón son para compilar el proyecto directamente (Generar main 21roject) y para compilar y limpiar el proyecto (Limpiar y generar main 21roject) respectivamente.

El tercer botón es el más usado, es el botón de ejecución o de emulación simple (Ejecutar main 21roject). Después de haber compilado el proyecto, pulsando este botón se abrirá la pantalla de emulación para poder visualizar la ejecución de la aplicación. Es así de sencillo, se pulsa el botón y al momento se inicia el emulador. Con el emulador se puede interactuar, o sea, se puede picar en las teclas del dispositivo que aparece tal como un dispositivo móvil real. Permite hacer pruebas del funcionamiento del midlet antes de su instalación final en un equipo o teléfono móvil. En la figura 3.9 se muestra una vista del emulador con la aplicación Hola Mundo ejecutándose en él.

El cuarto botón es para lanzar el depurador y el emulador a la vez. Este botón (Debug Main Proyect) lanzará la KVM (Kilobyte Virtual Machine) ejecutando emulador y depurador al unísono, permitiendo al programador comprobar los fallos del programa directamente sobre el emulador. El depurador nos mostrara cada una de las incidencias o acciones que están ocurriendo.

La segunda manera de lanzar el emulador antes mencionada es desde la opción Ejecutar en la barra de menús situado en la parte superior de la pantalla. Este incorpora más opciones que la barra de emulación explicada y su uso es similar. Contiene los botones de la barra, además de algunos otros para un mayor control y comodidad del programador como la ejecución paso a paso, introducción de Breakpoints, control de ejecución, saltos en código, etc.

Fig. 3.9 "Hola mundo" en el simulador

3.5 Java Servlet

Los **servlets**, son objetos que corren dentro del contexto de un servidor o contenedor de servlets (ej: Tomcat) y extienden su funcionalidad.

La palabra *servlet* deriva de otra anterior, applet, que se refería a pequeños programas que se ejecutan en el contexto de un navegador web. Por contraposición, un *servlet* es un programa que se ejecuta en un servidor.

El uso más común de los *servlets* es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

3.5.1 Ciclo de vida de un servlet

El ciclo de vida de un Servlet se divide en los siguientes puntos:

1. El cliente solicita una petición a un servidor vía URL.
2. El servidor recibe la petición.

Si es la primera, se utiliza el motor de Servlets para cargarlo y se llama al método init().

Si ya está iniciado, cualquier petición se convierte en un nuevo hilo. Un Servlet puede manejar múltiples peticiones de clientes.

3. Se llama al método service() para procesar la petición devolviendo el resultado al cliente.
4. Cuando se apaga el motor de un Servlet se llama al método destroy(), que lo destruye y libera los recursos abiertos.

CAPÍTULO IV
INTRODUCCIÓN AL SISTEMA GPS

4.1 ¿Qué es GPS?

El **Sistema de Posicionamiento Global (GPS -** Global Positioning System, más conocido con las siglas *GPS*, aunque su nombre correcto es *NAVSTAR-GPS*) es un Sistema Global de Navegación por Satélite (GNSS) que consta de 24 satélites orbitando la Tierra a una altura de 20200Km. El GPS fue desarrollado por el Departamento de Defensa de los Estados Unidos, debido a sus aplicaciones potenciales como instrumento de ubicación militar. Sin embargo en los últimos años, GPS ha probado ser una herramienta muy útil también en aplicaciones de localización no militares. Actualmente es operado por el Departamento de Defensa de los Estados Unidos.

Los satélites GPS orbitan lo suficientemente alto para evitar problemas asociados con los sistemas terrestres, no obstante puedan proveer la posición exacta de cualquier punto en el globo terrestre las 24 horas del día. Las posiciones determinadas por las señales satelitales producen un margen de error dentro del rango de 50 a 100 metros. Cuando se usa una técnica llamada corrección diferencial, los usuarios pueden conseguir una precisión de posición con un error dentro de los 5 metros o menos.

4.2 Latitud y Longitud

Para localizar un punto sobre la superficie de la Tierra y trasladarlo o plotearlo en un mapa o carta náutica, es necesario conocer primero las coordenadas donde se encuentra ubicado ese punto, es decir, la **latitud** y la **longitud**. Conocer el valor de las coordenadas es imprescindible para poder ubicar la posición de automóviles o coches, barcos, aviones, personas, carreteras, ciudades, puntos de interés, objetos, manchas de peces, fauna animal y hasta una piedra que se encuentre sobre la superficie de la Tierra.

Las **líneas de latitud o paralelos** están formadas por círculos de diferentes tamaños que parten de la línea del Ecuador y se expanden en dirección a los polos. La línea del Ecuador constituye el círculo de latitud de mayor diámetro de la Tierra y la divide en dos mitades: hemisferio Norte y hemisferio Sur.

La línea del Ecuador se identifica en las cartas náuticas y los mapas como latitud "0" grado (0°). A partir del Ecuador se extienden, hacia el norte y el sur, las denominadas líneas de latitud. El diámetro de los círculos que forman esas líneas se van empequeñeciendo a medida que se acercan a los polos hasta llegar a convertirse solamente en un punto en ambos polos, donde adquiere un valor de 90 grados (90°).

El Ecuador, como cualquier otro círculo, se puede dividir (y de hecho se divide) en 360 grados (360°), por lo cual pueden atravesarlo 360 líneas de **longitud o meridianos**. Estos meridianos se extienden desde el polo norte hasta el polo sur de forma paralela al eje de rotación de la Tierra. Como longitud "0" grado (0°) se designó el meridiano que pasa por el Real Observatorio Astronómico de Greenwich, cerca de la ciudad de Londres, en Inglaterra. Esa línea de longitud se conoce también por el nombre de meridiano de Greenwich a partir del cual se rigen los husos horarios que determinan la hora en todos los puntos de la Tierra.

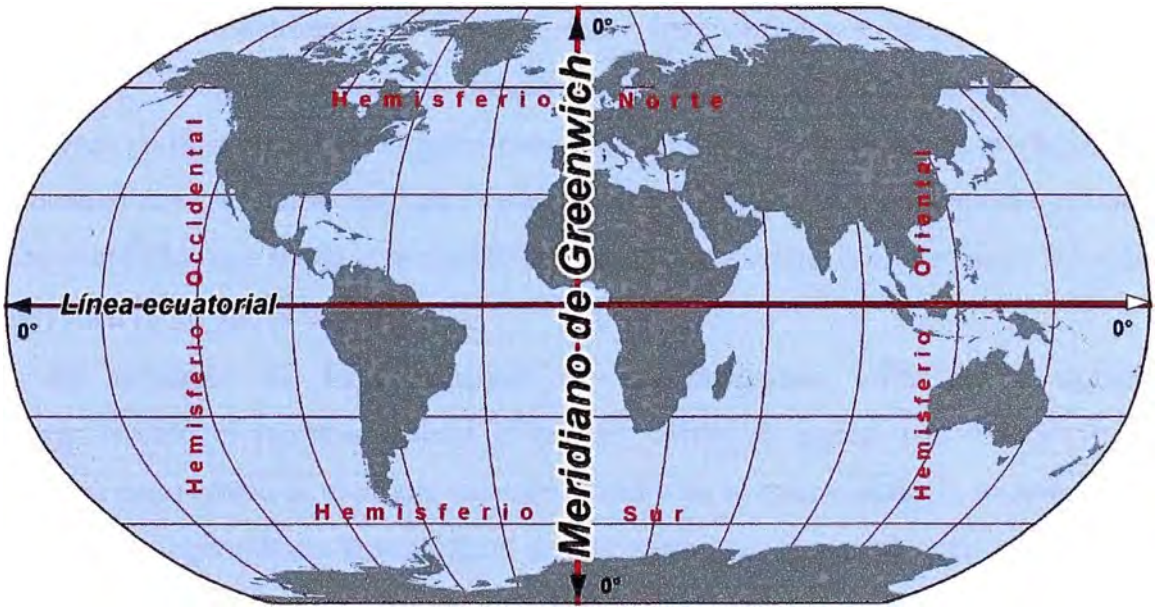


Fig. 4.1 Líneas del Ecuador y de Greenwich

Hasta hace un tiempo atrás, en navegación siempre se tomaba como referencia para todas las operaciones la hora GMT *(Greenwich Mean Time)* u hora del meridiano de Greenwich. Este meridiano divide la Tierra en otras dos mitades a partir de los polos, tomando como referencia su eje de rotación: hemisferio occidental hacia el oeste y hemisferio oriental hacia el este.

Tanto las líneas de longitud como las de latitud, además de dividirse en grados, se subdividen también en minutos y segundos. Por tanto podemos localizar un punto situado exactamente en las coordenadas 40° de latitud norte y 3° de longitud este y si nos

desplazamos unos kilómetros, el punto de localización podría ser 38° 40′ 20′′ (38 grados, 40 minutos, 20 segundos) de latitud norte y 3° 30′ 59′′ (3 grados, 30 minutos, 59 segundos) de longitud este. Si la medida anterior se repitieran en el hemisferio opuesto, por ejemplo en los 40° de latitud sur y 3° de longitud oeste, ésta sería una ubicación completamente distinta y muy alejada de la primera.

4.3 Principio de funcionamiento de los receptores GPS

Para ubicar la posición exacta donde nos encontramos situados, el receptor GPS tiene que localizar por lo menos 3 satélites que le sirvan de puntos de referencia (triangulación), ver fig 4.2. En realidad eso no constituye ningún problema porque normalmente siempre hay 8 satélites dentro del "campo visual" de cualquier receptor GPS. Para determinar el lugar exacto de la órbita donde deben encontrarse los satélites en un momento dado, el receptor tiene en su memoria un almanaque electrónico que contiene esos datos [6].

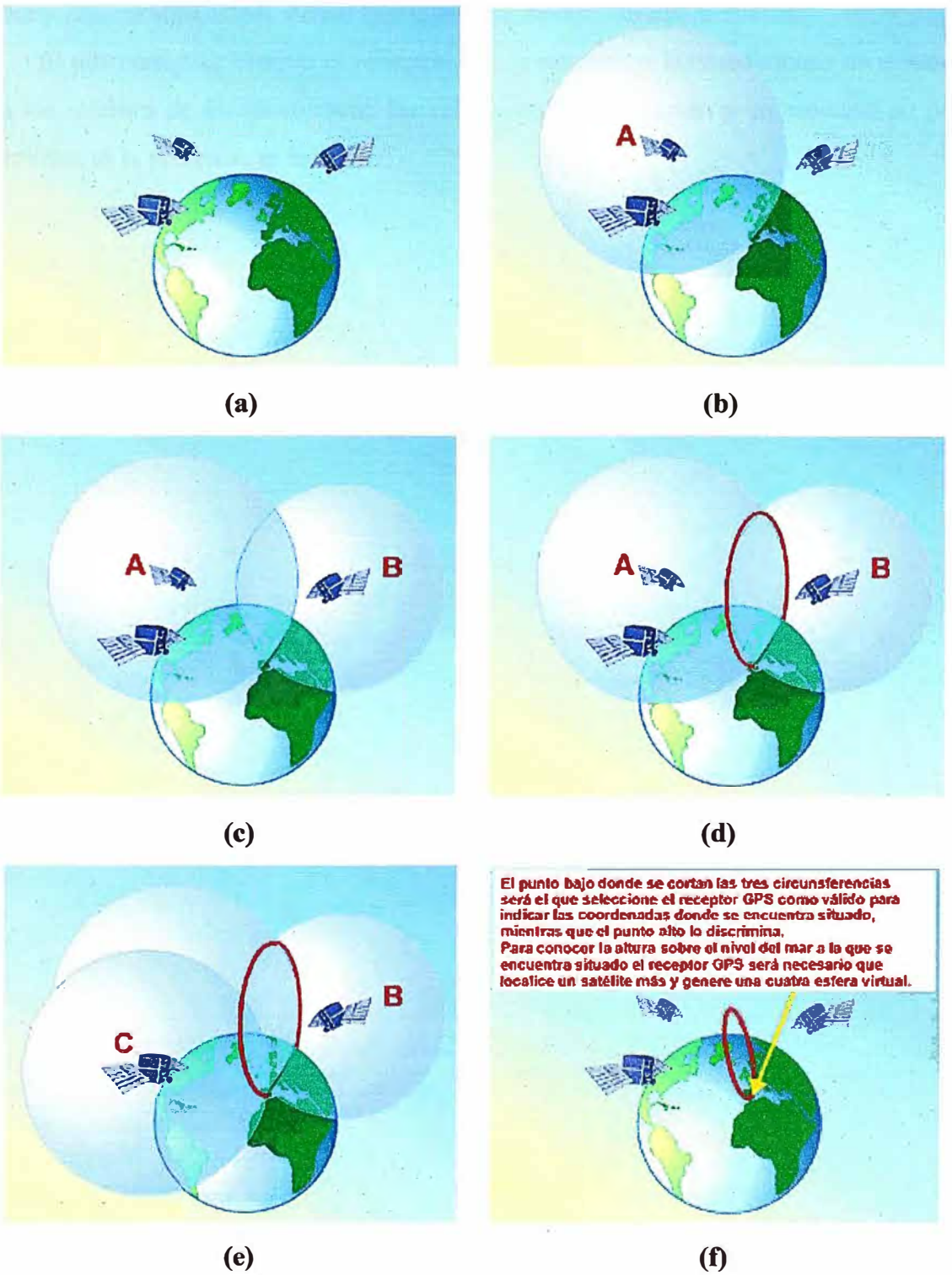
Tanto los receptores GPS de mano, como los instalados en vehículos con antena exterior fija, necesitan abarcar el campo visual de los satélites. Generalmente esos dispositivos no funcionan bajo techo ni debajo de las copas de los árboles, por lo que para que trabajen con precisión hay que situarlos en el exterior, preferiblemente donde no existan obstáculos que impidan la visibilidad y reduzcan su capacidad de captar las señales que envían a la Tierra los satélites.

El principio de funcionamiento de los receptores GPS es el siguiente: **Primero**: cuando el receptor detecta el primer satélite se genera una esfera virtual o imaginaria, cuyo centro es el propio satélite. El radio de la esfera, es decir, la distancia que existe desde su centro hasta la superficie, será la misma que separa al satélite del receptor. Éste último asume entonces que se encuentra situado en un punto cualquiera de la superficie de la esfera, que aún no puede precisar, figura 4.2 (b).

Segundo: al calcular la distancia hasta un segundo satélite, se genera otra esfera virtual. La esfera anteriormente creada se superpone a esta otra y se crea un anillo imaginario que pasa por los puntos donde se interceptan ambas esferas. En ese instante ya el receptor reconoce que sólo se puede encontrar situado en dicho anillo, figuras 4.2 (c) y (d).

Tercero: el receptor calcula la distancia a un tercer satélite y se genera una tercera esfera virtual. Esa esfera se corta con un extremo del anillo anteriormente creado en un punto en el espacio y con el otro extremo en la superficie de la Tierra. El receptor discrimina como ubicación el punto situado en el espacio utilizando sus recursos matemáticos de

posicionamiento y toma como posición correcta el punto situado en la Tierra, como se puede apreciar en las figuras 4.2 (e) y (f).



(a) **(b)**

(c) **(d)**

(e) **(f)**

Fig. 4.2 Método de la triangulación

Cuarto: una vez que el receptor ejecuta los tres pasos anteriores ya puede mostrar en su pantalla los valores correspondientes a las coordenadas de su posición, es decir, la latitud y la longitud.

Quinto: para detectar también la altura a la que se encuentra situado el receptor GPS sobre el nivel del mar, tendrá que medir adicionalmente la distancia que lo separa de un cuarto satélite y generar otra esfera virtual que permitirá determinar esa medición.

Si por cualquier motivo el receptor falla y no realiza las mediciones de distancias hasta los satélites de forma correcta, las esferas no se interceptan y en ese caso no podrá determinar, ni la posición, ni la altura.

CAPÍTULO V
MEDIOS EMPLEADOS DURANTE EL DESARROLLO Y PRUEBAS

5.1 Herramientas de Desarrollo

Para la implementación y pruebas se han empleado el entorno de desarrollo NetBeans 6.9.1 conjuntamente con el emulador de equipos móviles Wireless Toolkit 2.52, vea la figura 5.1, así como también se ha empleado un servidor web local, ejecutándose en Apache Tomcat 6.0, para poder comprobar los datos que envía nuestro midlet a la aplicación web.

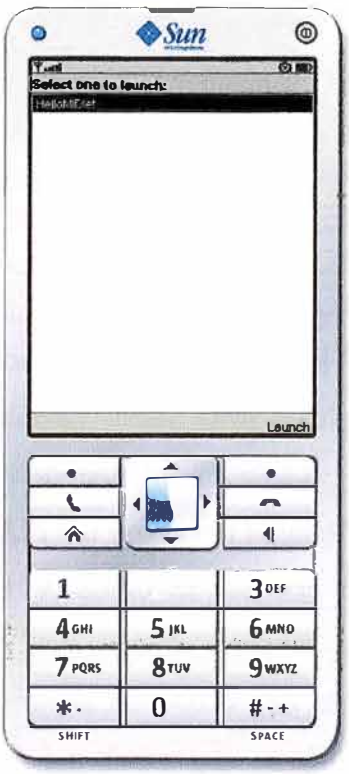


Fig. 5.1 Emulador Wireless Toolkit

5.2 Equipo móvil

El completo funcionamiento de nuestra aplicación se ha probado en teléfonos móviles Nokia con sistema operativo Symbian, en este caso se ha trabajado con el modelo E5 (aunque los dispositivos en los que se use, no tienen que ser necesariamente de esta marca y modelo, ni tener el mismo sistema operativo), como se muestra en la fig. 5.2.

El equipo cuenta con un receptor GPS interno. También es posible trabajar con un receptor externo, mediante la tecnología Bluetooth, para ello hay que vincular la conexión entre ambos dispositivos, y una vez vinculada se realizan las conexiones automáticamente.

Fig. 5.2 Móvil Nokia E5

5.2.1 Requisitos mínimos

La especificación *JSR-179 Location API for J2ME* [7] pone a nuestra disposición una serie de clases que permiten acceder desde Java a información relacionada con la posición y el movimiento de nuestro dispositivo.

Para poder trabajar con esta API sin problema alguno, necesitaremos un móvil que implemente la especificación JSR-179, como la mayoría de los Nokia o BlackBerry modernos.

Hay que tener en cuenta que no es necesario que el móvil disponga de GPS. La *Location API* nos aisla del origen de los datos, que puede ser un GPS o puede ser un servicio de pago proporcionado por la operadora.

CAPÍTULO VI
DESARROLLO Y CARACTERÍSTICAS DE LA APLICACIÓN

Hemos dividido nuestra aplicación en dos pantallas. En la primera nos pide los datos para hacer el proceso de localización, mientras que en la segunda se muestra el resultado de dicho proceso. Pasaremos a detallar el funcionamiento de nuestra aplicación.

6.1 Nuestro midlet

Nuestra aplicación es instalada en el móvil que va a ser localizado. Una vez instalada la aplicación, esta tiene que ser ejecutada manualmente para luego permanecer en segundo plano (oculta a la vista para el usuario) indefinidamente, hasta que la aplicación sea cerrada por el usuario, o hasta apagar el equipo.

Al ejecutar el midlet, aparecerá una primera pantalla, la cual nos pide que introduzcamos la dirección web del servidor con el cual se va a comunicar para enviar los datos de localización, así como el tiempo para hacer dicho proceso repetitivo (periodo). Estos valores, la dirección web y el tiempo, tienen un valor por defecto, los cuales pueden ser modificados o no a gusto del cliente.

Como se puede observar en la fig. 6.1, tenemos la dirección del servidor web *http://localhost:8090/uni/fiee/ServerCoord* y el periodo de 10s por defecto. Para efectos de prueba nos conectamos a un servidor web local, el cual esta ejecutándose en Apache Tomcat, cuya dirección con su respectivo puerto de escucha es el localhost:8090. En nuestro servidor, tenemos la aplicación web (servlet) ServerCoord que va a guardar los datos de localización en un fichero de datos para su posterior procesamiento.

Una vez introducidos los datos iniciales, podemos darle a la opción enviar. La aplicación le indicará al receptor GPS hacer las consultas de localización. Tenemos la opción de usar el servicio A-GPS [8] (Assisted-GPS) para lograr un posicionamiento inicial más rápido. Debido a que el A-GPS acarrea un costo adicional por conceptos de conectividad, en el presente proyecto con la finalidad de abaratar costos no tomaremos en cuenta dicho servicio. Ni bien son obtenidos los primeros datos de localización (latitud y longitud) estos son mostrados en la segunda pantalla, como se puede apreciar en la fig 6.2.

Fig. 6.1 Pantalla de datos iniciales

Después de haber mostrado los primeros datos de localización en pantalla, nuestro midlet tratará de comunicarse con el servidor para su posterior envío. Para este propósito, agregará dichos datos a la cadena url introducido inicialmente. La cadena resultante será más o menos la siguiente: *http://localhost:8090/uni/fiee/ServerCoord?lt=-12.05&ln=-77.05*. Notar la manera como son agregados los datos de localización a la misma url, en este caso las variables lt (latitud) y ln (longitud), por intermedio del símbolo ¿ y separados entre sí por el símbolo &. Esta forma de envío de datos es propio del método GET [9].

En la fig 6.2 también es posible observar los campos Proveedor y Metros recorridos, aparte de los de localización. Explicaremos estos campos uno a uno.

Nuestro midlet comprobará la existencia o estado del proveedor (fuentes de Localización) para poder realizar el posicionamiento, dicho estado lo mostrará automáticamente dentro del campo **Proveedor**. Si existe, preparamos a la aplicación para que pueda gestionar las actualizaciones que le vaya comunicando el receptor GPS, momento en el cual empieza la comunicación entre ambos.

Fig. 6.2 Pantalla de resultados de localización

Una vez realizada la consulta del estado del Proveedor, éste podrá adoptar cuatro posibles situaciones diferentes las cuales mostrará, como se había mencionado anteriormente, en el campo Proveedor:

·**Disponible**: El proveedor está disponible y además tiene cobertura, es decir, está listo para poder ejecutar nuestra aplicación.

·**Fuera de servicio**: Nuestro equipo no puede conectarse con él (por ejemplo si está demasiado lejos).

·**Temporalmente no disponible**: El Proveedor está conectado pero todavía no se puede posicionar debido a la ausencia de cobertura/señal de los satélites GPS.

·**Desconocido**: El dispositivo GPS en este caso no puede ser reconocido como tal.

Dichas respuestas o estados del Proveedor, son solamente indicativos, es decir, muestra el estado en Pantalla mientras que los resultados de localización mostrados siempre serán los últimos conocidos. Volverá a consultar el estado del Proveedor, así como los respectivos datos de localización después de un tiempo T.

El campo **Metros recorridos** es un adicional que se puede obtener gracias a J2ME, nos indica la distancia total recorrida desde el inicio hasta el momento de la consulta. También es posible pedir otros datos como la velocidad, la calle donde nos encontramos, etc.

Los campos **Latitud** y **Longitud** son los datos de localización que nos enviara el receptor GPS. Si después de varios periodos T, el receptor GPS adopta un estado diferente al de Disponible, estos campos se mantienen con los últimos datos conocidos. Sus formatos son -12.05 (latitud sur), -77.05 (longitud oeste).

Fig. 6.3 Servidor web no encontrado

Una vez transcurrido todo este proceso la aplicación queda en espera, es decir, vuelve con el bucle al principio del programa, donde ésta se prepara para realizar todo nuevamente después de un tiempo o periodo T.

Cabe también indicar cuando no se encuentra el servidor web disponible, en este caso la aplicación al no encontrar el servidor web para el posterior envío de datos de localización, esta simplemente mostrara un mensaje en pantalla: *"Servidor no encontrado"* y continuará con su trabajo rutinario, es decir, queda a la espera de un tiempo T para

realizar la siguiente consulta de posicionamiento, observe la fig. 6.3. Tal situación del servidor no afecta la operatividad de nuestra aplicación.

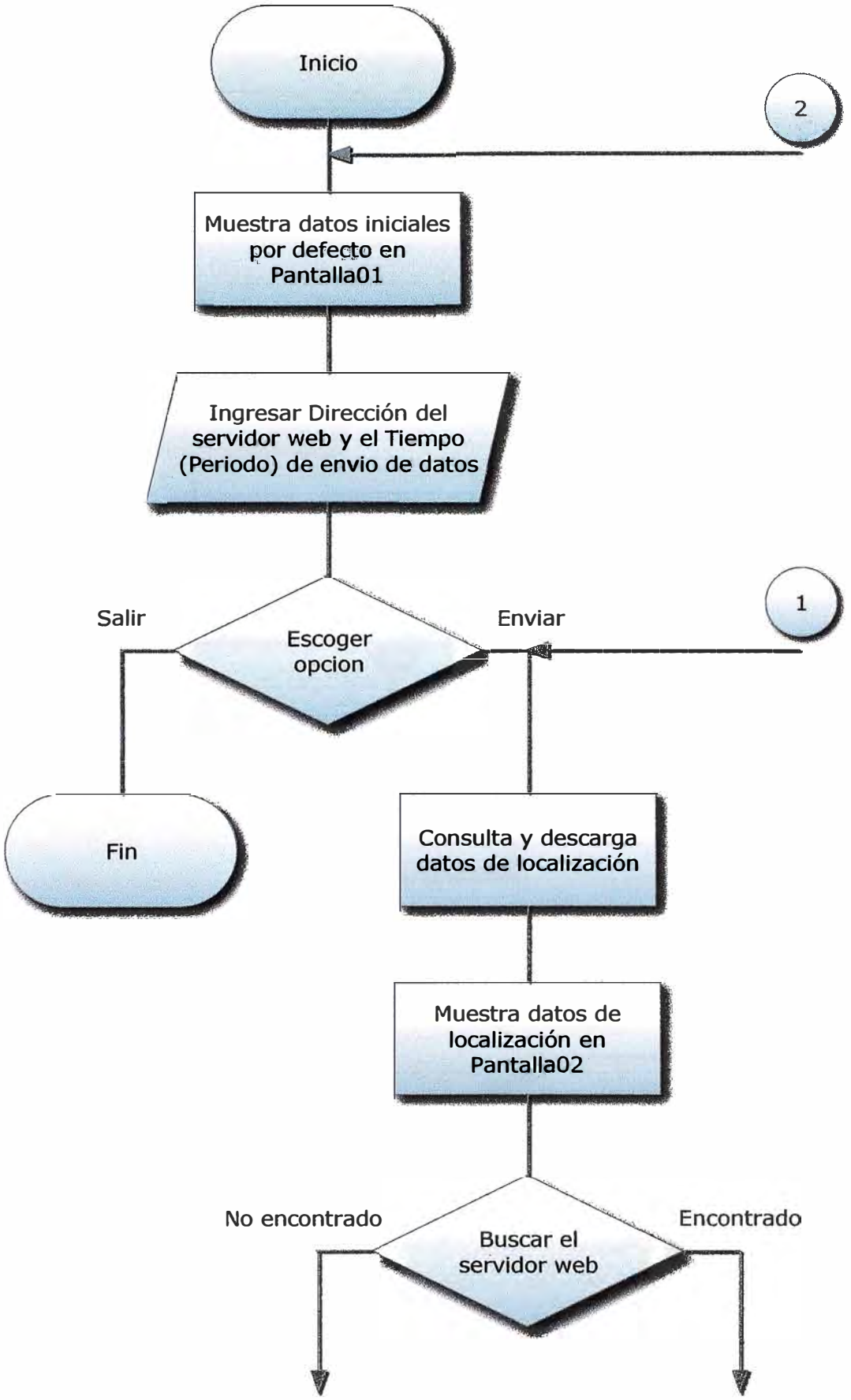
6.2 Ejemplo, parte del principio del midlet

```
public class LocalizacionGPS extends MIDlet implements CommandListener,
LocationListener {
 // Coordenadas de la última posición consultada
 private QualifiedCoordinates lastCoord;
 // Distancia total recorrida actualmente
 private double totalDistance;
 private String lat="-12.05", lon="-77.45", urlstring="";
 private int time=0;
 // Se usa un URL por defecto. El usuario puede cambiarlo desde el GUI.
 private String defaultURL ="http://localhost:8090/uni/fiee/ServerCoord";
 private String defaultTime ="10";
 // Componentes GUI para la introducción de una Web URL.
 private Display myDisplay = null;
 private Form mainScreen;
 private TextField txtdefaultURL;
 private TextField txtdefaultTime;
 // Componentes GUI para mostrar las respuestas del servidor.
 private Form resultScreen;
 private StringItem resultField;
 private TextField txtEstadoProveedor;
 private TextField txtLongitud;
 private TextField txtLatitud;
 private TextField txtDistanciaRecorrida;
 // boton "send" usado en el mainScreen
 Command sendCommand = new Command("Enviar", Command.OK, 2);
 // boton "back" usado en el resultScreen
 Command backCommand = new Command("Atras", Command.BACK, 1);
 // boton "salir" usado en el mainScreen
 Command exitCommand = new Command("Salir", Command.EXIT, 0);
 public LocalizacionGPS(){
```

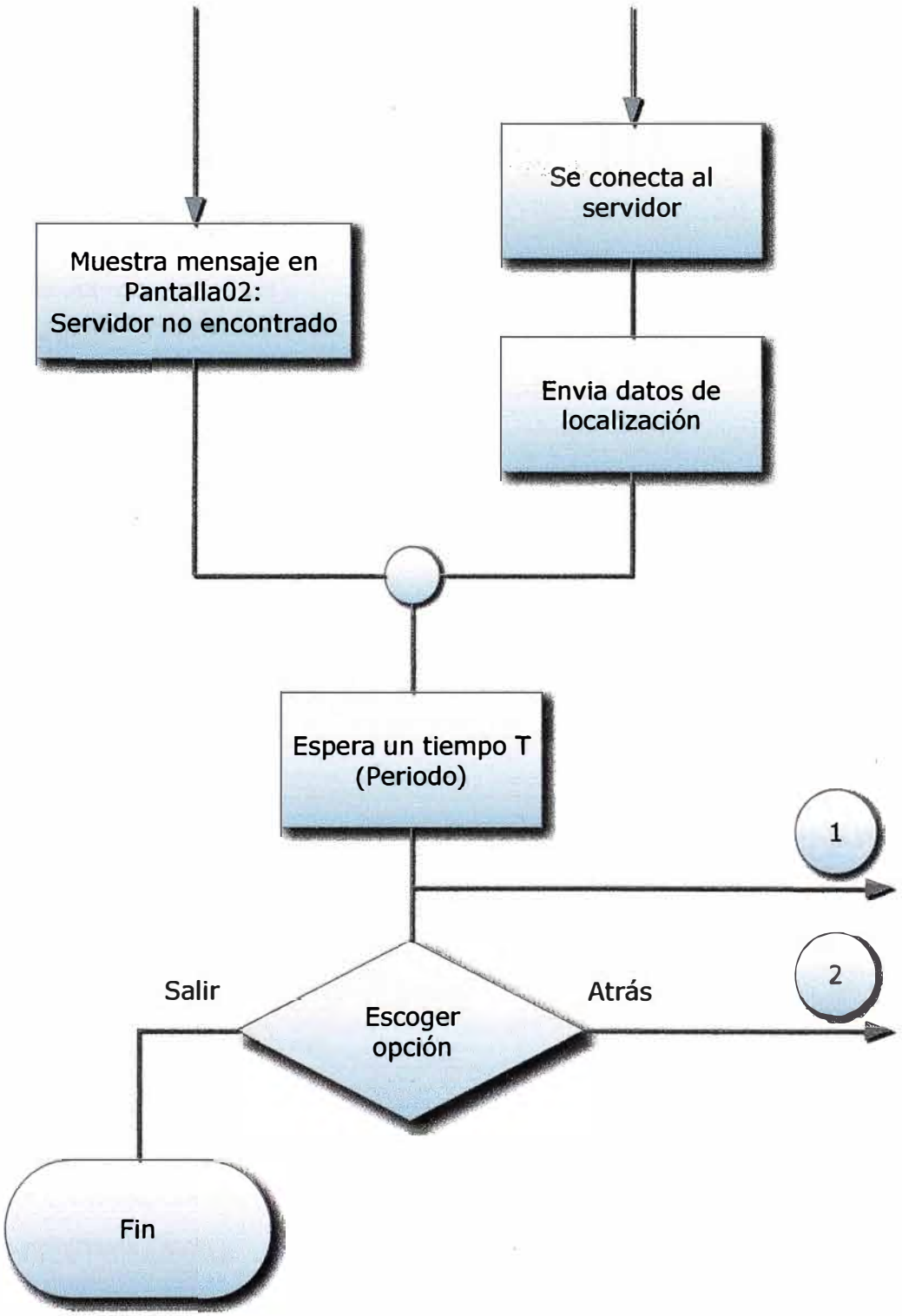
```
    //inicializamos los componentes GUI
    mainScreen = new Form("Datos iniciales:");
    resultScreen = new Form("Resultados de Localizacion");
    txtdefaultURL = new TextField("Direccion del servidor:", defaultURL, 100,
TextField.URL);
    txtdefaultTime = new TextField("Tiempo (Periodo):", defaultTime, 50,
TextField.NUMERIC);
    txtEstadoProveedor=new TextField("Proveedor:",      null, 50,  TextField.ANY);
    txtDistanciaRecorrida  = new TextField("Metros recorridos:", null, 50,
TextField.ANY);
    txtLatitud    = new TextField("Latitud:",        null, 50,   TextField.ANY);
    txtLongitud   = new TextField("Longitud:",       null, 50,   TextField.ANY);
    mainScreen.append(txtdefaultURL);
    mainScreen.append(txtdefaultTime);
    mainScreen.addCommand(sendCommand);
    mainScreen.addCommand(exitCommand);
    resultScreen.append(txtEstadoProveedor);
    resultScreen.append(txtDistanciaRecorrida);
    resultScreen.append(txtLatitud);
    resultScreen.append(txtLongitud);
    resultScreen.addCommand(backCommand);
    resultScreen.addCommand(exitCommand);
    }

public void startApp() {
    myDisplay = Display.getDisplay(this);
    mainScreen.setCommandListener(this);
    myDisplay.setCurrent(mainScreen);
  }
public void pauseApp() {
  }
public void destroyApp(boolean unconditional) {
  }
```

```
public void commandAction(Command c, Displayable s) {
  // Cuando el usuario clickea sobre el boton "send" en el mainScreen
  if (c == sendCommand) {
    // devolver el url web que el usuario introdujo
    urlstring = txtdefaultURL.getString();
    // devolver el tiempo que el usuario introdujo en numero entero
    time=Integer.parseInt(txtdefaultTime.getString());
    resultScreen.setCommandListener(this);
    myDisplay.setCurrent(resultScreen);
    try{
      ConexionGPS();
    } catch (Exception e) {
      //resultstring = "ERROR";
      resultScreen.append(e.getMessage());
    }
  }
    else if (c == backCommand) {
    //todo desde el comienzo
    mainScreen.setCommandListener(this);
    myDisplay.setCurrent(mainScreen);
  }
  else if (c == exitCommand) {
  this.notifyDestroyed();
  }
}

 public void ConexionGPS() throws LocationException   {
     // Establecemos los criterios del proveedor de localizacion deseado.
      Criteria criteria = new Criteria();
      // No queremos usar un proveedor de pago
      criteria.setCostAllowed(false);
      // No se requiere información sobre la calle, país, ciudad, etc. en la que me encuentro.
      criteria.setAddressInfoRequired(false);
```

```
        //No quiero datos de velocidad
        criteria.setSpeedAndCourseRequired(false);
        // Solicitamos el proveedor de localización
        LocationProvider provider = LocationProvider.getInstance(criteria);
        // ¿La implementación nos ha proporcionado un proveedor con esas características?
        if (provider  == null)
       {
           // Solicitamos nuevamente el proveedor de localización
           provider = LocationProvider.getInstance(criteria);
        }
        // si tenemos proveedor inicializamos
        if (provider  != null){
           try {
              // Método bloqueante
              Location location = provider.getLocation(-1);
              // Si hay datos de localización continuamos poniendo un timer que nos de  dichos
           datos  cada X segundos (llamará al método locationUpdated)
              if (location != null){
                 this.lastCoord    = location.getQualifiedCoordinates();

                 // LocationListener, interval, timeout, maxAge)
                 provider.setLocationListener(this, time, -1, -1);
              }
              // Inicializamos el UI con los datos del estado del proveedor.
              this.providerStateChanged(provider, provider.getState());
           }
              catch (java.lang.InterruptedException ex)
           {
              resultScreen.append(ex.getMessage());
           }
        }
     }
}
```


Inicio
2
Muestra datos iniciales por defecto en Pantalla01
Ingresar Dirección del servidor web y el Tiempo (Periodo) de envio de datos
1
Salir
Escoger opcion
Enviar
Fin
Consulta y descarga datos de localización
Muestra datos de localización en Pantalla02
No encontrado
Buscar el servidor web
Encontrado

(a)



(b)

Fig. 6.4 Diagrama de flujos de la aplicación móvil

6.3 Ejemplo, parte del principio del servlet

```
public class ServerCoord extends HttpServlet {
public void doPost(HttpServletRequest req, HttpServletResponse res){
try{
res.setContentType("text/html");
String latitud = req.getParameter("lt");
String longitud = req.getParameter("ln");
String res1 = "Estas son tus coordenadas:";
PrintWriter out=res.getWriter();
out.println(res1);
out.println("latitud="+latitud+" ,longitud="+longitud);
// Dirección donde se guardará el fichero Reporte
String sFichero = "C:\\Users\\jesus\\Documents\\Reporte.txt";
File fichero = new File(sFichero);
boolean append = true;
if (fichero.exists()){
   System.out.println("El fichero " + sFichero + " ya existe");
   System.out.println("Se agregaran los datos al fichero existente");
   try{
      BufferedWriter bw = new BufferedWriter(new FileWriter(sFichero, append));
      bw.write("latitud: " + latitud  + "\t");
      bw.write("longitud: " + longitud  + "\t");
      bw.newLine();
         // Hay que cerrar el fichero
      bw.close();
      }
      catch (IOException ioe){
      ioe.printStackTrace();
      }
         }
         else {
               try{
                BufferedWriter bw = new BufferedWriter(new FileWriter(sFichero));
```

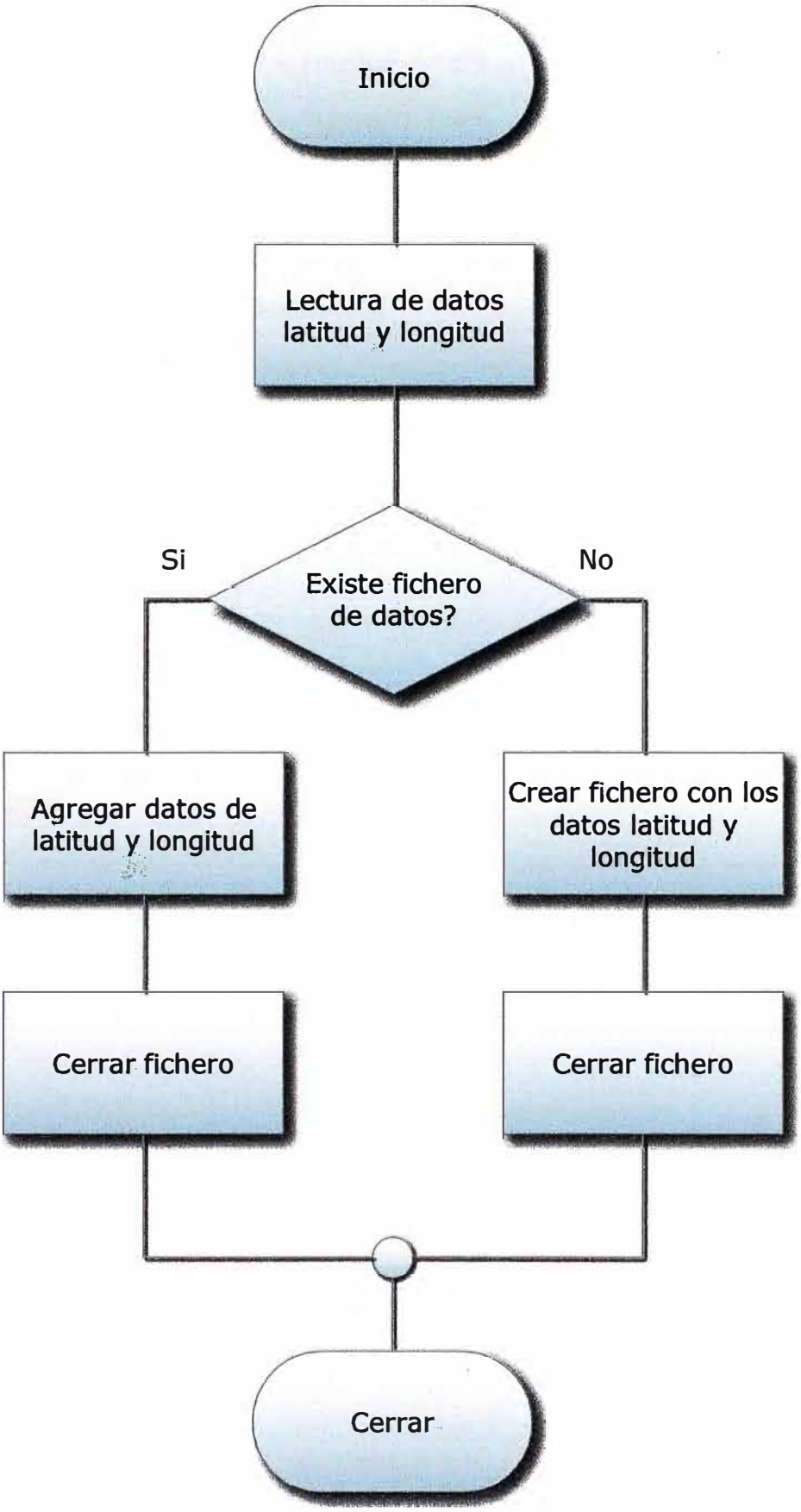


Fig. 6.5 Diagrama de flujos de la aplicación web

```
                bw.write("latitud: " + latitud  + "\t");
                bw.write("longitud: " + longitud  + "\t");
                bw.newLine();
                // Hay que cerrar el fichero
                      bw.close();
                } catch (IOException ioe){
                            ioe.printStackTrace();
                            }
                  }
out.close();
}
catch (Exception e){
System.out.println(e);
}
}
public void doGet(HttpServletRequest req, HttpServletResponse res){
doPost(req,res);
}
}
```

CAPÍTULO VII
PRUEBAS Y RESULTADOS

En el capítulo 5 se habló de los medios empleados durante el desarrollo, el cual consistía de NetBeans 6.9.1 conjuntamente con el emulador de equipos móviles Wireless Toolkit 2.52, y de Apache Tomcat 6.0 que pueda alojar la aplicación web (servlet) que va a recepcionar los datos de localización. Así mismo, también se menciono del equipo con el cual se estuvo haciendo las pruebas, en este caso, el Nokia E5.

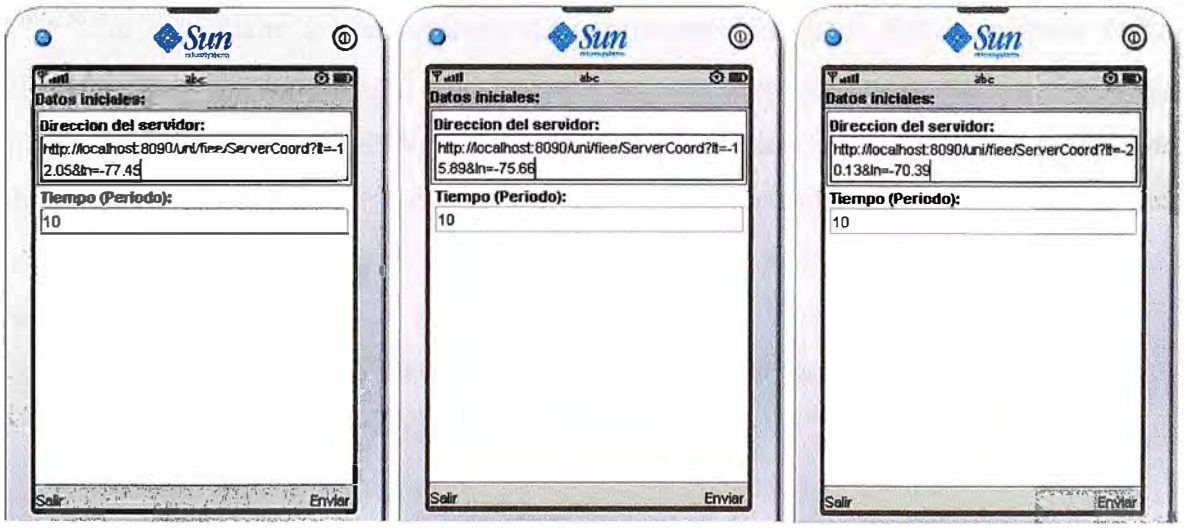
En este capítulo mostraremos las pruebas y resultados de nuestro proyecto, justamente usando dichos medios mencionado líneas arriba. Empezaremos probando el funcionamiento de nuestro midlet primero a nivel del emulador wireless, luego en el mismo equipo movil.

7.1 Nuestro midlet en el emulador

Para efectos de simulación, para poder comprobar que nuestro midlet se está comunicando correctamente con el servidor (en este caso el localhost), introduciremos tres coordenadas geográficas manualmente como datos de localización en la cadena url:

http://localhost:8090/uni/fiee/ServerCoord?lt=-12.05&ln=-77.45

http://localhost:8090/uni/fiee/ServerCoord?lt=-15.89&ln=-75.66



(a) **(b)** **(c)**

Fig. 7.1 Introducción de las coordenadas geográficas

http://localhost:8090/uni/fiee/ServerCoord?lt=-20.13&ln=-70.39

Notar que están en el formato necesario para que el servidor pueda obtener los datos que le interesa a partir de la cadena url. En condiciones normales, solo habría que introducir la dirección web del servidor, es decir la cadena url hasta justamente antes del símbolo ?, el resto es introducida por nuestro midlet después de haber descargado los datos de localización del receptor GPS. Dichas urls completas las podemos observar en la fig 7.1. El periodo lo dejamos en su valor por defecto, 10s.

Le damos al botón enviar, seguidamente se mostrara los resultados de localización en la siguiente pantalla, como se muestra en la fig 7.2. Los campos Proveedor y Metros recorridos se muestran en blanco, porque para esta prueba no está trabajando el receptor GPS, recordemos que los datos de localización han sido introducidos manualmente.

(a) **(b)** **(c)**

Fig. 7.2 Muestra de las coordenadas geográficas introducidas

Lo que viene a continuación es comprobar que estos datos se están enviando correctamente al servidor. Nuestro servlet, el cual se encuentra en la siguiente dirección:http://localhost:8090/uni/fiee/, creara un fichero llamado Reporte donde almacenará los datos de localización que continuamente le está enviando nuestra aplicación midlet. Esto queda evidente si analizamos la siguiente parte del código del servlet:

```
String sFichero = "C:\\Users\\jesus\\Documents\\Reporte.txt";
File fichero = new File(sFichero);
boolean append = true;
if (fichero.exists()){
   System.out.println("El fichero " + sFichero + " ya existe");
```

```
System.out.println("Se agregaran los datos al fichero existente");
try{
   BufferedWriter bw = new BufferedWriter(new FileWriter(sFichero, append));
   bw.write("latitud: " + latitud  + "\t");
   bw.write("longitud: " + longitud  + "\t");
   bw.newLine();
     // Hay que cerrar el fichero
   bw.close();
   } catch (IOException ioe){
                        ioe.printStackTrace();
                        }
     }
     else {
              try{
          BufferedWriter bw = new BufferedWriter(new FileWriter(sFichero));
               bw.write("latitud: " + latitud  + "\t");
          bw.write("longitud: " + longitud  + "\t");
          bw.newLine();
          // Hay que cerrar el fichero
               bw.close();
          } catch (IOException ioe){
                        ioe.printStackTrace();
                        }
       }
```

El fichero Reporte lo crea en la dirección según lo indica la cadena sFichero, en este caso en la carpeta Mis Documentos. El fichero Reporte lo podemos apreciar en la figura 7.3.

En caso no se encontrase el servidor activo por alguna razón, nuestro midlet supera este inconveniente mostrando un mensaje en pantalla, el de “Servidor no encontrado”, para luego seguir su funcionamiento normal, quedando a la espera de un periodo T y realizar la próxima descarga de datos del receptor GPS. Dicha situación lo podemos apreciar en la figura 7.4.

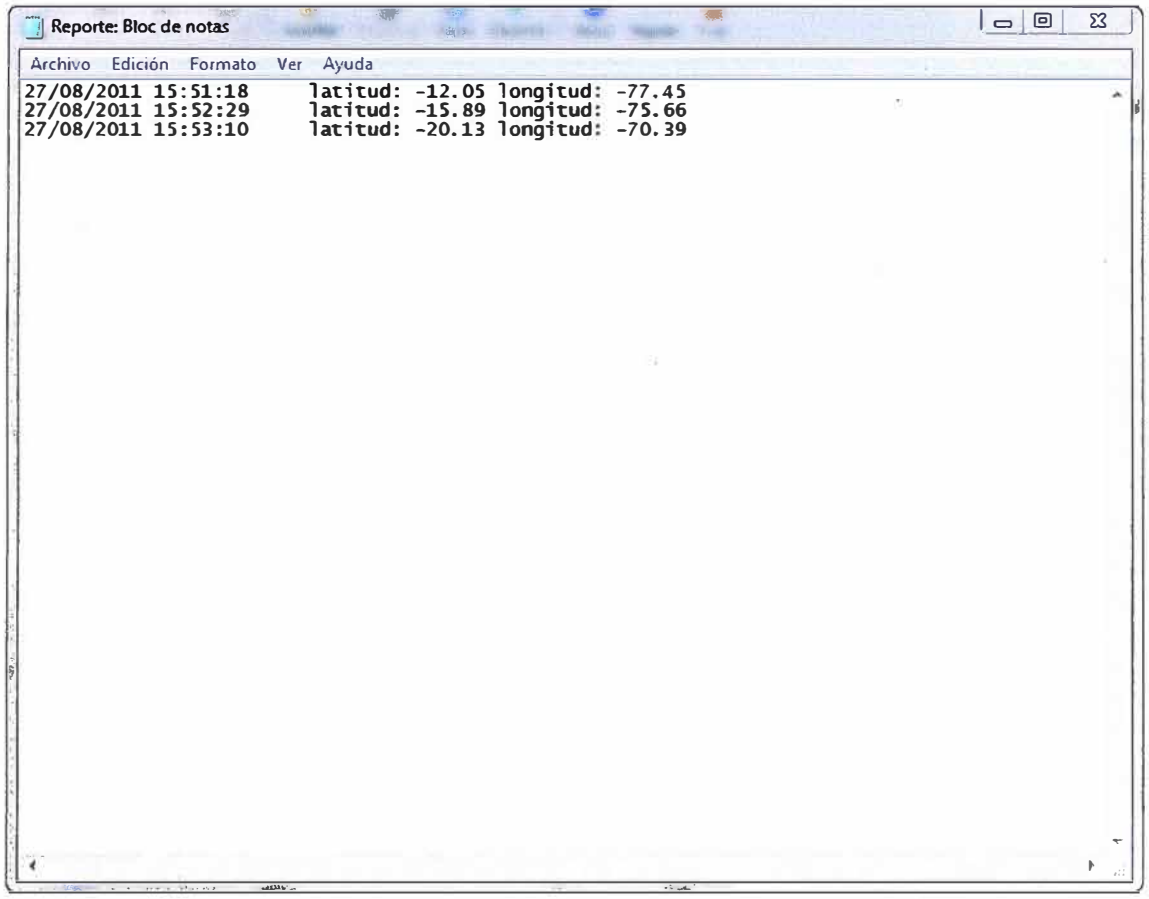


Fig. 7.3 Fichero Reporte creado por el servidor web

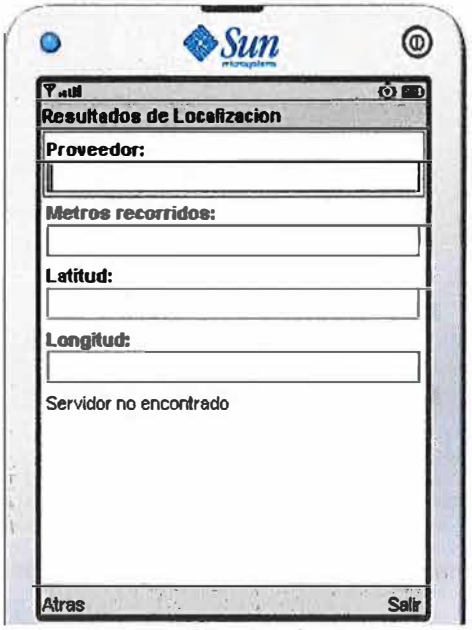


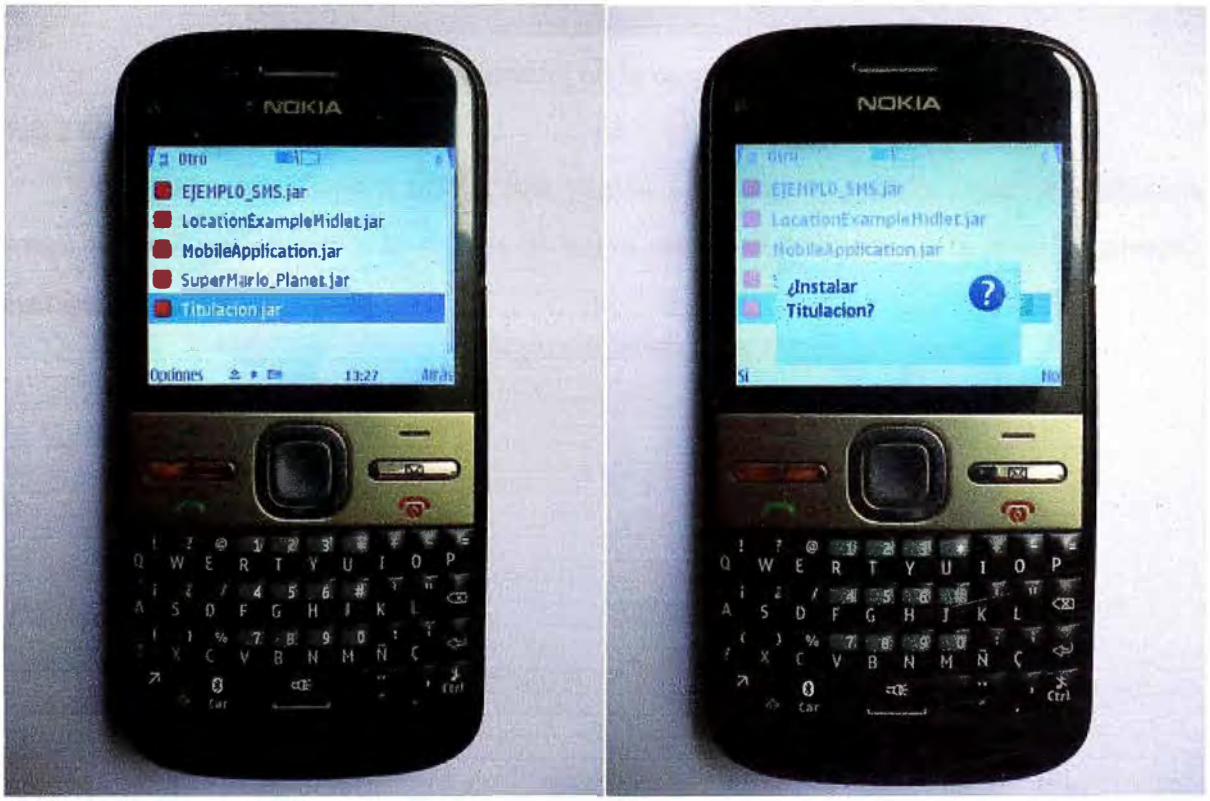
Fig. 7.4 Pantalla indicando que no se ha encontrado el Servidor

7.2 Nuestro midlet en un equipo real

Para este caso, como se había explicado anteriormente, vamos a realizar nuestras pruebas de funcionamiento en el equipo móvil Nokia E5.

7.2.1 Instalación (instrucciones para Nokia)

Para poder instalar nuestro midlet necesitamos su archivo ejecutable, de la forma *.jar. Para este trabajo o proyecto, nuestro midlet tiene el nombre de LocalizacionGPS, mientras que su archivo ejecutable el de Titulacion.jar. Guardamos el archivo Titulacion.jar en alguna carpeta de la memoria externa (también podría ser en la memoria interna) del teléfono. Nos ubicamos en esa carpeta y vamos al archivo, observe la fig. 7.5(a). Le damos al botón enter o aplicar, automáticamente el teléfono nos preguntará por la instalación, fig. 7.5(b), respondemos que si para que la instalación empiece inmediatamente.



(a) **(b)**

Fig. 7.5 Instalación de nuestra aplicación

Por defecto las aplicaciones que se instalan en el móvil, aparecen en la carpeta "Instalaciones", a su vez dentro de la carpeta "Aplicaciones" que se encuentra en el menú principal. Pero podemos mover nuestra aplicación a la carpeta que queramos, seleccionándola y pulsando "opciones" y "mover a carpeta", la opción "mover" sirve para cambiar la posición de la aplicación dentro de la carpeta en la que se encuentra. En la fig. 7.6 se puede apreciar el icono de nuestra aplicación LocalizacionGPS dentro de la carpeta Instalaciones.

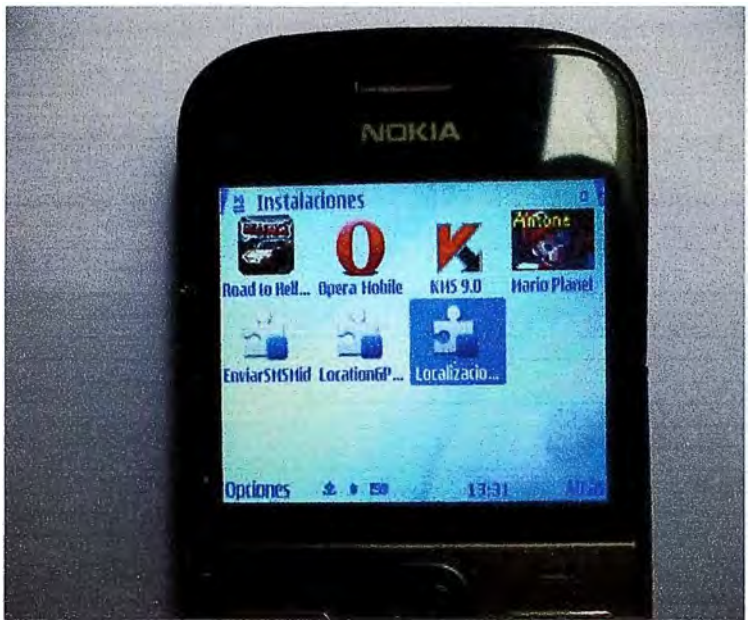


Fig. 7.6 Nuestro midlet en la carpeta de Instalaciones

7.2.2 Ejecución

Para ejecutar nuestro midlet nos vamos a la carpeta Instalaciones, nos ubicamos sobre nuestra aplicación y le damos al botón enter. Otra forma sería ir a "opciones" y escogemos la opción "abrir".

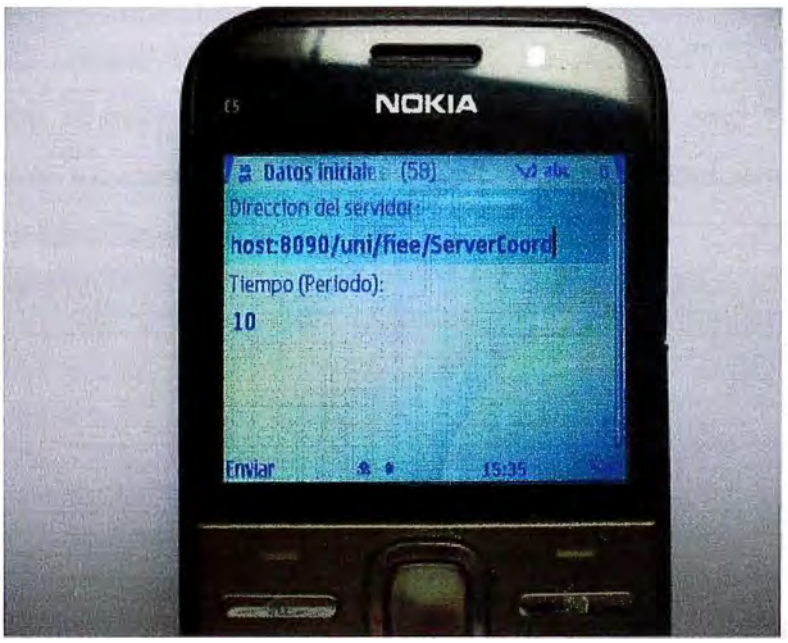


Fig. 7.7 Pantalla "Datos Iniciales"

Aparece la primera pantalla, fig. 7.7, que nos pide los datos iniciales para empezar a ejecutar la aplicación, en este caso la dirección web del servidor al cual enviara los datos de localización, así como el periodo en segundos para realizar dicho envío. Note los valores que aparecen por defecto, el campo Periodo solo admite valores numéricos enteros. En la parte inferior de esta primera pantalla se encuentran los botones Enviar, para

empezar la consulta de datos GPS y enviarlos al servidor, y Salir si desea abandonar la aplicación.

Una vez comprobados los datos de la primera pantalla, entonces le damos al botón Enviar para continuar con la aplicación, aparecerá la siguiente pantalla, mostrada en la fig. 7.8.

Fig. 7.8 Pantalla "Resultados de Localización"

Si en algún momento el estado del Proveedor deja de estar "Disponible", los valores mostrados en los campos Latitud y Longitud no podrán actualizarse cada tiempo T, estos valores serán los últimos conocidos.

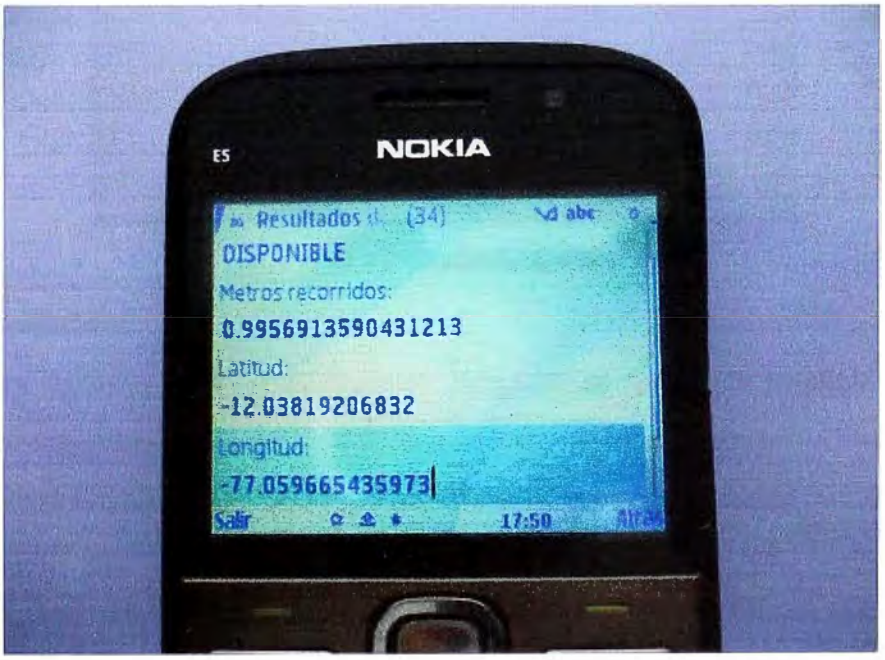


Fig. 7.9 "Resultados de Localización" después de un tiempo T

Cabe recordar que los valores negativos de las coordenadas geográficas mostradas se refieren a valores de latitud sur y de longitud oeste: Latitud: -12.038208748301 (latitud sur), Longitud: -77.059679769022 (longitud oeste).

En la parte inferior de esta segunda pantalla se encuentran los botones Atrás, para volver a la pantalla anterior si se quiere corregir algunos datos iniciales, y Salir si desea abandonar la aplicación.

La fig. 7.9 nos muestra los resultados de localización para la siguiente consulta hecha después de un tiempo T.

7.2.3 Desinstalación

Nuestra aplicación, al igual que cualquier otra, puede ser desinstalada buscándola en la carpeta o directorio que se encuentre, seleccionándola y dándole a "opciones" y a "eliminar", observe la fig. 7.10. Otra alternativa es ir a la carpeta "Archivos instalados" usando el "Administrador de archivos" desde "Oficina" en el menú principal, donde veremos la lista de todas las aplicaciones instaladas en nuestro teléfono y podremos elegir la que queremos desinstalar o modificar alguno de sus parámetros.

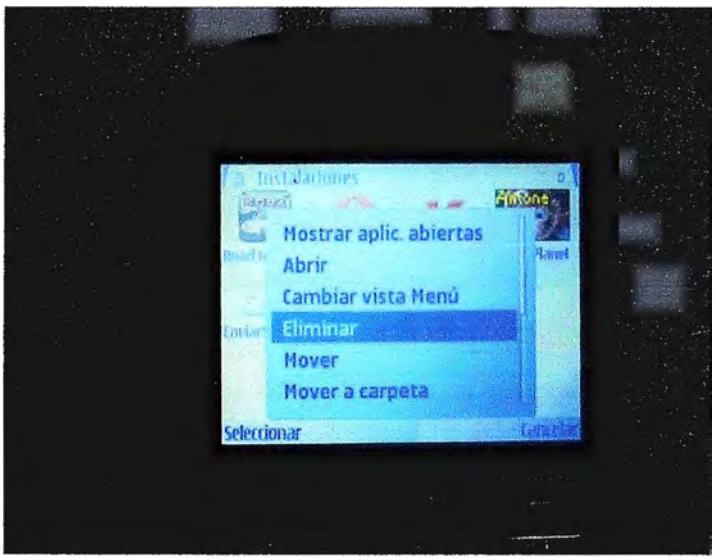


Fig. 7.10 Desinstalación

7.3 Gastos incurridos

Los gastos incurridos en el presente proyecto se limitan a la adquisición de un teléfono móvil con receptor GPS incluido más la conectividad a internet debido al envío de información entre el móvil y el servidor web, como se puede apreciar en la Tabla 7.1.

Tabla 7.1 Gastos incurridos

Equipo móvil (S/)	Plan Internet Total (S/)	Gasto inicial (S/)
600.00 (promedio)	95.00 (a la fecha)	695.00

CONCLUSIONES Y RECOMENDACIONES

Objetivos iniciales propuestos

En nuestro caso concreto, las posibilidades de controlar el receptor GPS (ya sea uno interno o externo) con J2ME así como el de establecer enlaces inalámbricos (internet inalámbrico) han hecho posible prácticamente cumplir los objetivos propuestos inicialmente, con las siguientes particularidades:

- Precisión menor a 10m.
- Tiempo real
- En todo momento
- Cobertura

Los dos primeros puntos fueron cubiertos cabalmente, llegando a una precisión inclusive por debajo de los 10m. Salvo el primer intento de comunicación con el sistema satelital GPS, que por lo general le toma alrededor de 1 minuto (GPS convencional), la consulta y descarga de datos de localización se realiza efectivamente en tiempo real.

El factor En todo momento depende de la disponibilidad de ambas señales, de las del operador móvil y de los satélites GPS. Si falla el primero nuestra aplicación móvil o midlet trabajará parcialmente, si falla el segundo simplemente no funcionará.

La cobertura queda prácticamente limitada al área de servicio del operador móvil debido a las conexiones inalámbricas necesarias para el envío de datos de localización. Entonces el uso del presente proyecto es factible en áreas donde haya cobertura celular, por ejemplo en la gran mayoría de la costa peruana.

Posicionamiento inicial

Debido a que el GPS convencional presenta dificultades a la hora de proporcionar posiciones precisas en condiciones de baja señal, se recomienda el uso del servicio A-GPS (Assisted-GPS) aprovechando que estamos usando precisamente un teléfono móvil. No se trata de una nueva tecnología, sino de un añadido a GPS, el cual justamente sirve para disminuir el tiempo de inicialización.

Hay que tener en cuenta que el principal inconveniente de este servicio es que la información que nos llega desde el servidor externo lo hace por tráfico de datos lo cual puede resultar en un coste adicional.

Agregaciones o cambios al presente proyecto

La posibilidad de enviar y recibir mensajes de texto también abre una serie de oportunidades en la resolución de situaciones o problemas. Una mejora o cambio al presente proyecto, sería el de hacer consultas de localización de algún objeto móvil en cualquier momento desde otro móvil a través de los mensajes de texto sin la necesidad de estar respondiendo cada petición, ya que se trata de una aplicación de segundo plano.

Trabajo futuro

La funcionalidad de nuestro servidor web es almacenar la data en su base de datos, en este proyecto trabajamos con ficheros de datos. Es posible utilizar dicha data para la generación de informes y análisis, visualización sobre mapas digitales y otras funcionalidades del servicio, mediante la creación de software que integre una cartografía detallada a nivel nacional para la consulta por parte del cliente o el centro de control. Dicho software no es parte del presente proyecto, pero si puede ser considerado como una extensión o parte de un trabajo futuro.

J2ME y sus potencialidades

Con el desarrollo del presente proyecto, hemos notado que gracias a las diversas API que posee J2ME, y con un poco de imaginación, es prácticamente posible implementar toda aplicación que nos propongamos dentro de los limites de hardware ya conocidos (capacidades computacionales y gráficas muy reducidas).

Además J2ME, tiene cada vez más relevancia para el uso y control de consumibles electrónicos y dispositivos embebidos (embedded), lo cual lo hace de un futuro bastante prometedor.

ANEXO
GLOSARIO DE TÉRMINOS UTILIZADOS

J2ME – Java 2 Micro Edition

CDC – Connected Device Configuration

CLDC – Connected Limited Device Configuration

MIDP – Mobile Information Device Profile

SMS – Short Message Service

J2SE – Java 2 Standard Edition

WWW – World Wide Web

Applet – es una aplicación pequeña hecha en Java que se ejecuta en el contexto de otro programa mas grande

Servlet – es un programa que se ejecuta en un servidor orientado a petición – respuesta y extiende la funcionalidad de este

PDA – Personal Digital Assistant

API – Application Programming Interface

JVM – Java Virtual Machine

GPS – Global Position System

A-GPS – Assisted-Global Position System

JSR – Java Specification Request

RISC – Reduced Instruction Set Computer

GUI – Graphical User Interface

JAR – Java ARchive

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

JDK – Java Development Kit

KVM – Kilobyte Virtual Machine

GSM – Groupe Special Mobile, Sistema global para las comunicaciones

PC – Personal Computer

CGI – Common Gateway Interface

POS – Point Of Sale (JavaPOS)

BIBLIOGRAFÍA

[1] Agustín Froufe Quintas, Patricia Jorge Cárdenes. Java 2 Micro Edition – Manual de usuario y tutorial. RA-MA Editorial, Madrid 2004.

[2] http://www.ssimail.com/WAP_gateway.htm.

[3]Sun Microsystems. J2ME Building blocks for mobile devices – White paper on KVM and the Connected Limited Device Configuration (CLDC). Palo Alto, CA USA. http://java.sun.com/products/cldc/wp/KVMwp.pdf

[4] Sun Microsystems. Java Technology for the Wireless Industry Version 1.0 , Status: FCS. Network Circle, Santa Clara California, June 26 2003.

[5] J. Muchow, Core J2ME Technology and MIDP. Prentice Hall, 2002.

[6] http://www.asifunciona.com/electronica/af_gps/af_gps_1.htm.

[7] Java Community Process, Location API for Java 2 Micro Edition, version 1.0.1. http://jcp.org/aboutJava/communityprocess/final/jsr179/index.html.

[8] http://javiercancela.com/2008/01/07/leyendo-nuestro-gps-desde-java-con-lajavame-location-api-jsr-179-parte-i/

[9] Sergio Gálvez Rojas, Lucas Ortega Díaz. Java a tope: J2ME (Java 2 Micro Edition). Edición electrónica. ETS de Ingeniería Informática – Universidad de Málaga – España.