ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FIN DE CARRERA

LOCALIZACIÓN DE MÓVILES CON GPS

AUTOR: José Luis Marín Marín
DIRECTOR: José Fernando Cerdán Cartagena

Julio 2009

Autor	José Luis Marín Marín
E-mail del Autor	Joseluisxs@hotmail.com
Director	José Fernando Cerdán Cartagena
E-mail del Director	fernando.cerdan@upct.es
Título del PFC	*Localización de móviles con GPS*

Resumen

Este proyecto consiste en el desarrollo de una aplicación J2ME, es decir, una aplicación en Java específica para dispositivos móviles (multiplataforma).

La aplicación tiene como objetivo la localización de un dispositivo móvil que incorpora/se conecta a una antena GPS, desde cualquier otro móvil y mediante el envío de mensajes de texto.

Se han implementado una aplicación cliente y otra servidor, mediante el entorno de desarrollo NetBeans, en el que se han ido desarrollando los diferentes módulos del programa, probándolos en el simulador y posteriormente su correcto y total funcionamiento con teléfonos Nokia y una antena GPS externa, conectada por Bluetooth.

Titulación	Ingeniería Técnica de Telecomunicación, especialidad Telemática
Intensificación	-
Departamento	Tecnologías de la información y las comunicaciones
Fecha de presentación	Julio 2009

Contenido

Tabla de ilustraciones.

Capítulo I – Introducción acerca del proyecto

Objetivos

Este proyecto consiste en el desarrollo e implementación de una aplicación J2ME, es decir, una aplicación en Java específica para dispositivos móviles (multiplataforma), que actúan como teléfonos móviles o lo son. La aplicación tiene que poder usarse en cualquier móvil actual.

Se pretende conseguir localizar a una persona (u otra cosa) que lleve un dispositivo móvil con nuestra aplicación instalada, mediante un usuario que disponga de otro dispositivo móvil con la aplicación (versión cliente).

La aplicación (o MIDlet) tiene como objetivo la localización de un dispositivo móvil que incorpora/se conecta a una antena GPS, desde cualquier otro móvil y mediante el único envío de mensajes de texto.

Descripción

Se han implementado una aplicación cliente y otra servidor, mediante el entorno de desarrollo NetBeans, en el que se han ido desarrollando los diferentes módulos del programa, probándolos en el simulador y posteriormente su correcto y total funcionamiento con teléfonos Nokia y una antena GPS externa, conectada por Bluetooth.

Cuando se envía una petición mediante un SMS, el dispositivo que se quiere localizar envía un mensaje de texto con la correspondiente posición dónde éste se encuentra, u otra información en caso de que fuera necesaria.

Éste es el modo de comunicarse ambas aplicaciones en sendos dispositivos, mediante el envío de mensajes de texto, que contienen la información necesaria y que será interpretada por la aplicación o el usuario final.

Partes de la aplicación:

- **Aplicación Servidor:** es la que se está ejecutando siempre en el dispositivo que vamos a localizar (en 2º plano), se inicia automáticamente al encender el móvil.
- **Aplicación Cliente:** ésta se ejecuta en otro dispositivo cualquiera, en el momento que queramos localizar a alguien. Hay una versión básica y sencilla, y otra avanzada con muchas opciones que tiene en cuenta y nos permite usar.

Funcionamiento:

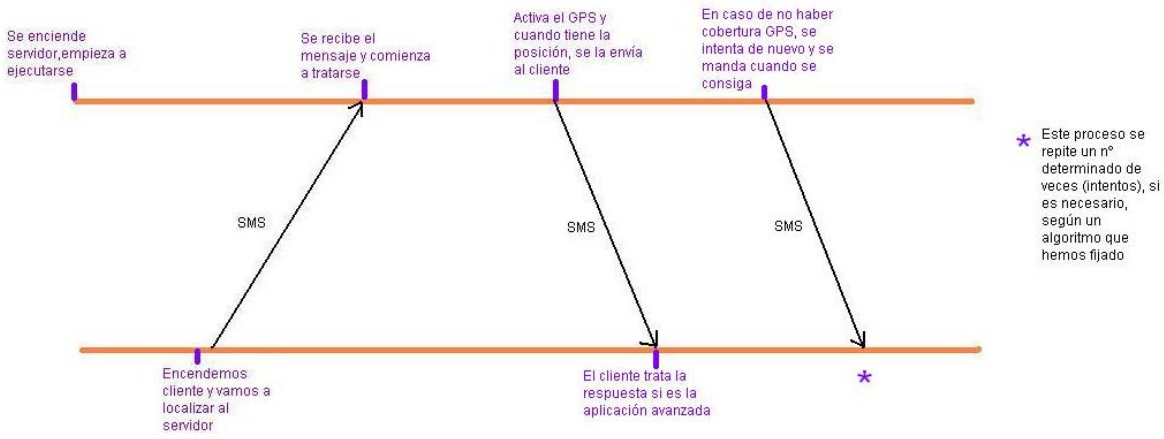


Ilustración 1 - Intercambio de mensajes

Capítulo II – Aplicaciones J2ME (Midlets)

Introducción sobre Java

Java es un lenguaje de programación orientado a objetos desarrollado por la empresa Sun Microsystems en 1995 y que se ha extendido ampliamente en 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*, descargados desde el servidor Web. Además, debido al modo de ejecución de los *Applets*, este lenguaje es muy seguro frente a la presencia y ataque de virus informáticos.

Definición J2ME

La plataforma J2ME 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)

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, J2SE o J2EE, no encajan dentro de este esquema. J2ME es por tanto, una versión “reducida” de J2SE.

J2ME contiene una mínima parte de las APIs 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, ya se

hablara de ella más ampliamente en la siguiente subsección. Otras diferencias con J2SE vienen dadas por el uso de una maquina virtual distinta de la clásica JVM denominada KVM. Esta KVM tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la JVM. Estas diferencias se verán más detenidamente cuando analicemos la KVM.

Como se puede ver, J2ME representa una versión simplificada de J2SE. Sun separó estas dos versiones ya que J2ME estaba pensada para dispositivos con limitaciones de proceso y capacidad gráfica. Se puede afirmar que J2ME es un subconjunto de J2SE exceptuando el paquete javax.microedition perteneciente a J2ME y no existente en J2SE.

Perfiles, configuraciones y máquinas virtuales de J2ME

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, 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), 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) no establece como obligatorios para incorporar en los dispositivos. Ejemplos de esto pueden ser las Apis de mensajes inalámbricos (WMAPI) y de multimedia (WMAPI).

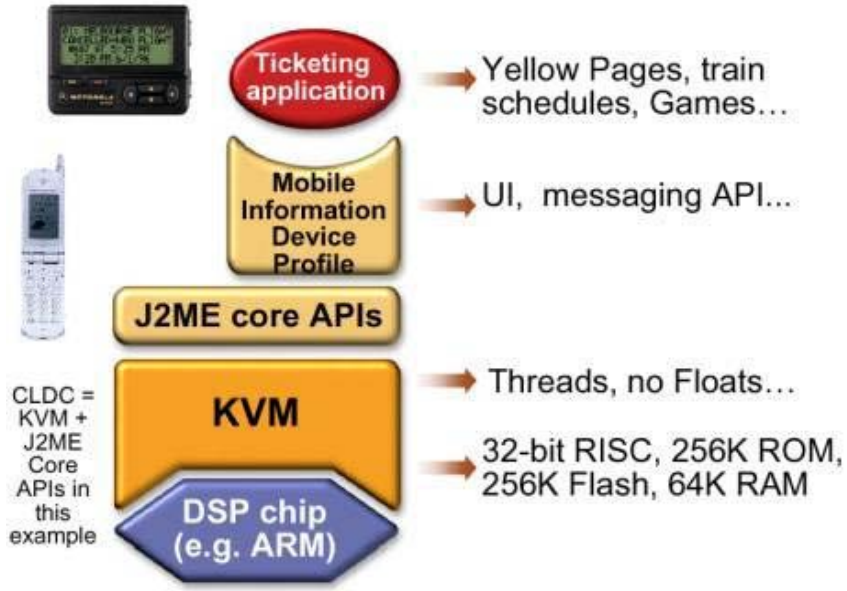


Ilustración 2 - ejemplo CLDC

CDC, configuración de dispositivos con conexión (Conected Limited 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 *Java* 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. 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.

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). 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.

La JSR185 pide como requisitos mínimos para todo equipo que implemente CLDC 1.0 o 1.1:

• Soporte mínimo para diez hilos relacionados con aplicaciones (MIDlets).
• Usar zonas de tiempo personalizables, con referencia en el formato de zonas de tiempo GMT (GMT \7:00, por ejemplo).
• Soporte para propiedades de carácter y conversiones mayúsculas-minúsculas en los bloques supleméntales Unicode para Basic Latin y Latin-1 (nuestros caracteres).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

• Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
• Un subconjunto de las bibliotecas Java del núcleo.
• Soporte para E/S básica.
• Soporte para acceso a redes.
• Seguridad.

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 amito 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. Para la configuración CDC existen los siguientes perfiles:

• Foundation Profile.
• Personal Profile.
• RMI Profile.

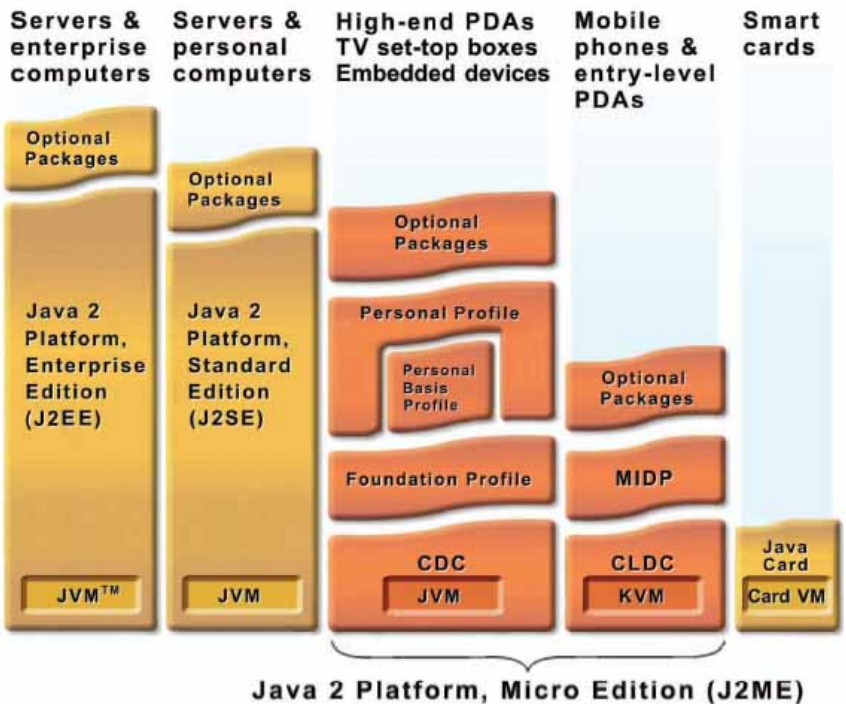


Ilustración 3 - diferentes versiones de Java

Y para la configuración CLDC:
• PDA Profile.
• Mobile Information Device Profile (MIDP).

Un perfil puede ser construido sobre cualquier otro. Sin embargo, una plataforma J2ME sólo puede contener una configuración. A continuación se explicarán los diferentes perfiles más en profundidad.

• *Foundation Profile:* Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica, como por ejemplo los decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes “java.awt” (Abstract Windows Toolkit, AWT) y “java.swing” que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación

requiriese una GUI, entonces sería necesario un perfil adicional. Los paquetes que forman parte del Foundation Profile son los siguientes:

1. Java.lang
2. Java.util
3. Java.net
4. Java.io
5. Java.text
6. Java.security

• *Personal Profile:* El Personal Profile es un subconjunto de la plataforma J2SE v1.3, y proporciona un entorno con un completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere la implementación del Foundation Profile. Los paquetes que conforman el Personal Profile v1.0 son:

1. Java.applet
2. Java.awt
3. Java.awt.datatransfer
4. Java.awt.event
5. Java.awt.font
6. Java.awt.im
7. Java.awt.im.spi
8. Java.awt.image
9. Java.beans
10. Javax.microedition.xlet

• *RMI Profile:* Este perfil requiere la implementación del Foundation Profile, se construye encima de él. El perfil RMI soporta un subconjunto de las APIs v1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido a las limitaciones de cómputo y memoria de los dispositivos. Las siguientes propiedades se han eliminado del J2SE RMI v1.3:

1. Java.rmi.server.disableHTTP.
2. Java.rmi.activation.port.
3. Java.rmi.loader.packagePrefix.
4. Java.rmi.registry.packagePrefix.
5. Java.rmi.server.packagePrefix.

• *PDA Profile:* El PDA Profile está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x100 pixels) con un factor 2:1.

• *Mobile Information Device Profile (MIDP):* Construido sobre la configuración CLDC. MIDP fue el primer perfil definido para esta plataforma. Hasta este momento es el único

perfil aplicado a los dispositivos en el mercado, aunque se están investigando algunos otros, como el especializado en PDA. MIDP 2.0 incorpora APIs de interfaz de usuario, de ciclo de vida del programa, almacenamiento persistente, juegos, trabajo en red y multimedia. Según la especificación de la tecnología inalámbrica de Java todo dispositivo que soporte MIDP 2.0 debe incluir mínimamente las siguientes características:

1. Debe permitir archivos Java (JAR) de más de 64 KB. y archivos descriptores de aplicaciones (JAD) mayores a 5 KB.

2. Se debe permitir a cada MIDlet la utilización de 30 KB de almacenamiento persistente y se recomienda que las MIDlets incluyan información acerca del almacenamiento mínimo con el que trabajan correctamente.

3. El espacio de memoria libre para una aplicación ejecutándose (Heap o del montón) debe ser por lo menos de 256 KB.

4. Soporte para pantallas de 125 x 125 pixeles, con una profundidad de color de 12 bits.

5. Se deben incluir la capacidad de que el dispositivo reaccione a eventos de tiempo (una alarma a determinada hora, los llamados ticklers o despertadores).

6. Mecanismos para tomar un número telefónico del directorio del equipo.

7. Soporte para imágenes en formato JPEG y PNG.

8. Acceso a contenidos multimedia por el protocolo HTTP 1.1.

Los paquetes que están incluidos en MIDP son los siguientes:
• Javax.microedition.lcdui
• Javax.microedition.rms
• Javax.microedition.midlet
• Javax.microedition.io
• Java.io

Java
• Java.lang
• Java.util

Además de las clases específicas de MIDP contenidas en *javax.microedition.rms*, *javax.microedition.midlet* y *javax.microedition.lcdui*; están disponibles las siguientes clases, interfaces y clases de excepción:

• *IllegalStateException.* Clase en el paquete *java.lang*.
• *Timer* y *TimerTask.* Clases en el paquete *java.util*.

• *HttpConnection.* Interfaz para acceso a una red por el protocolo HTTP contenida en el paquete *javax.microedition.io*

Las aplicaciones creadas utilizando MIDP reciben el nombre de MIDlets. Se puede decir que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

Ventajas de esta plataforma

La principal ventaja es que el lenguaje Java es multiplataforma, por lo que nuestra aplicación funciona independientemente del sistema operativo dónde la aplicación se ejecute (gran ventaja respecto a otros lenguajes).

La versión que se usa en nuestro caso es J2ME, la versión de Java para dispositivos móviles, que incorporan todos los móviles, PDAs… en la actualidad. Por ello en cualquier dispositivo móvil actual se puede instalar y ejecutar nuestra aplicación, sea del tipo que sea, incorporará la plataforma J2ME siendo capaz de ejecutar MIDlets.

Otras alternativas

Para el desarrollo de aplicaciones ejecutables en dispositivos móviles, existen otros lenguajes de programación o plataformas a parte de J2ME, con diferentes características y propios del tipo de dispositivo, tales como:

Symbian, Windows Mobile, C++, C#, .NET…

Pero dependiendo de cual se escoja, hay que estudiar sus características y sus posibilidades, que en cada caso pueden llegar a ser bastante diferentes. En nuestro caso (J2ME) la aplicación es multiplataforma, gracias a la máquina virtual de Java, que es la que interpreta nuestro programa y se encarga de la comunicación con el sistema operativo.

Ilustración 4 - logotipo de Java

Capítulo III – Entorno de desarrollo

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.

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. A

pesar de que Netbeans ya incluye un emulador integrado se ha decidido usar esta aplicación ya que es realmente eficiente y muy útil.

Paquete de movilidad y simulador

Una vez instalado el entorno de desarrollo Netbeans, se le adjunta este módulo que hace posible el desarrollo y demás de aplicaciones móviles.

El Paquete de Movilidad de NetBeans es una herramienta para desarrollar aplicaciones que se ejecutan en teléfonos móviles; puede ser usado para escribir, probar, y depurar aplicaciones para la plataforma Java ME, tecnología existente en dispositivos móviles.

Instalación de Netbeans IDE y el modulo Mobility: Directamente desde la página de la herramienta es posible descargarla, este IDE es totalmente gratuito y su página oficial es: http://www.netbeans.org

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 del mundo que pretenda crear una aplicación en cualquier lenguaje informático existente. 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

Este enlace descargará en el ordenador la versión 5.5.1 de Netbeans con el jdk incluido. Desde el siguiente enlace habrá que descargar el Mobility Pack para CLDC y Netbeans 5.5.1. Esta herramienta es indispensable para la creación de aplicaciones en J2ME. Su dirección web es la siguiente:

http://www.netbeans.info/downloads/index.php?p=4

Como se citó anteriormente, Wireless Toolkit para CLDC, esta es una herramienta simplemente de emulación que se podría utilizar, pero que no hará falta ya que Netbeans lo incluye en sí mismo.

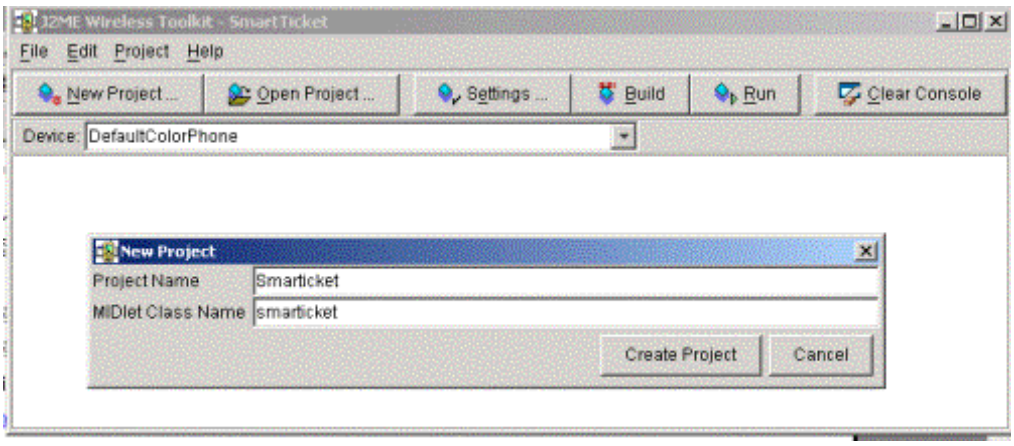


Ilustración 5 - Wireless Toolkit

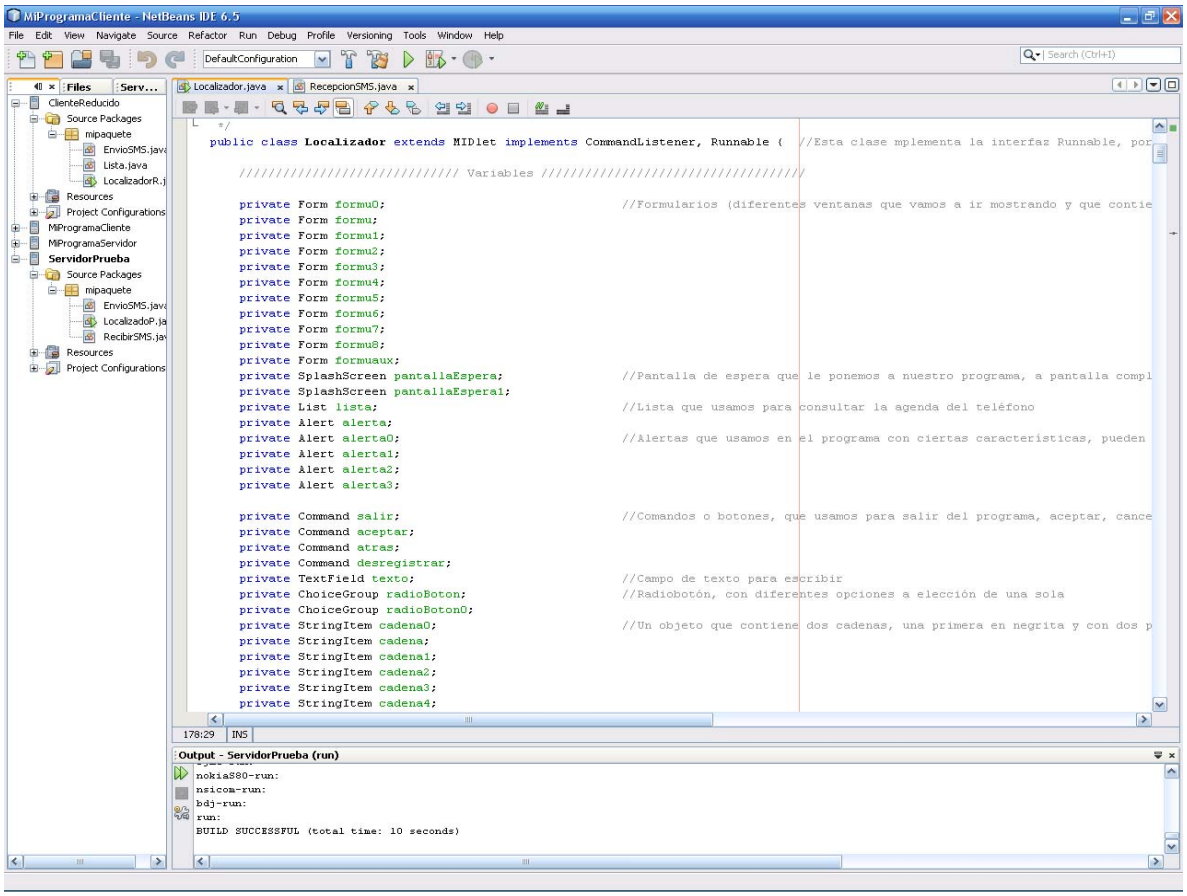


Ilustración 6 - NetBeans IDE

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 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.

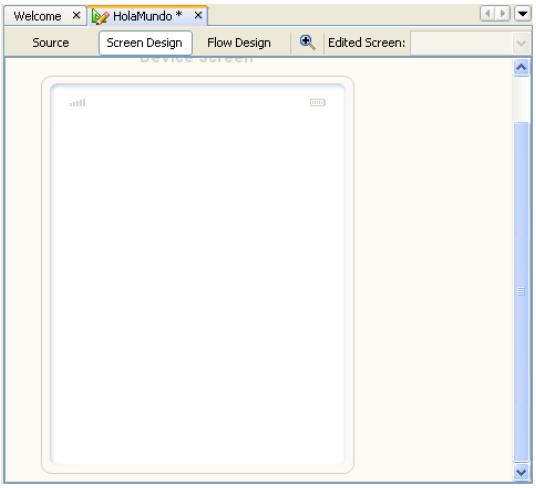


Ilustración 7 - Screen design

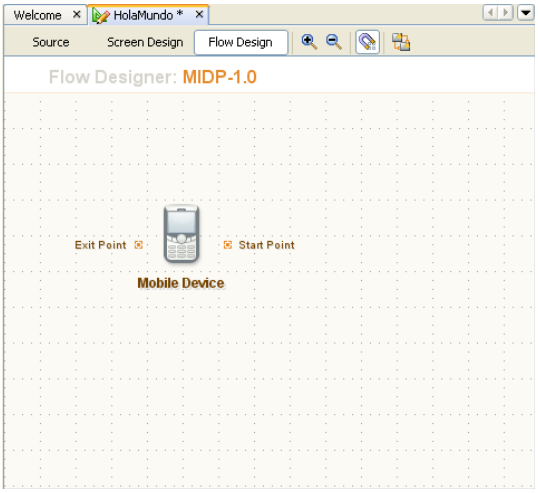


Ilustración 8 - Flow design

• 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 habrán tres opciones principales (después de poner las opciones se añadirán unas figuras ilustrativas del proceso con cada opción):

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

ii. Si se pincha sobre un paquete (package) se deberá elegir la opción Compile Package o pulsar F9.

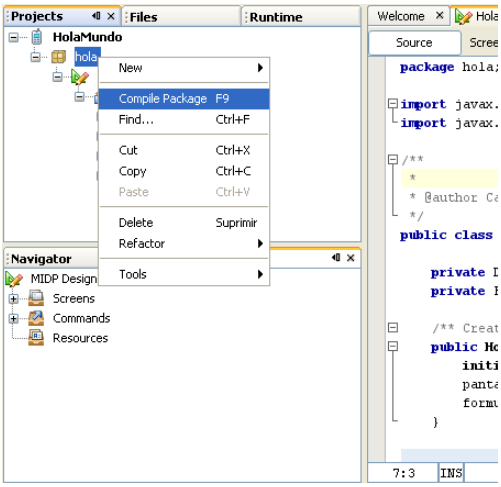


Ilustración 9 - compilar paquete

iii. Si se pincha sobre un proyecto (project) se elegirá la opción Build Project.

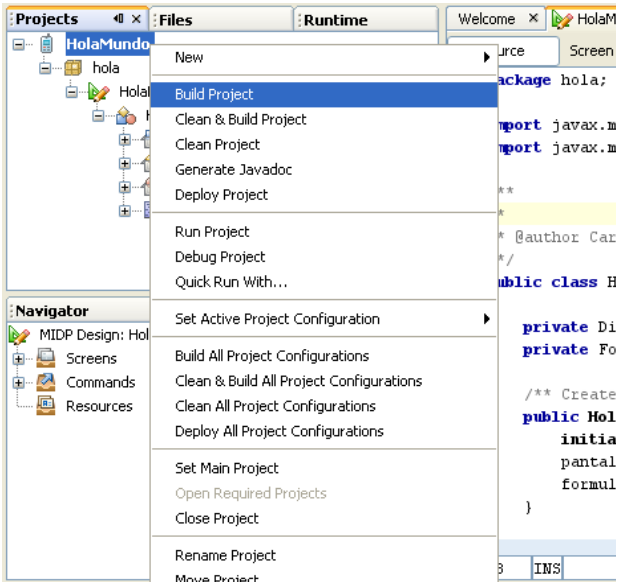


Ilustración 10 - compilar el proyecto

• El proceso de preverificación se realiza automáticamente después de la compilación.

• 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.

• Las fases de ejecución y depuración también se pueden realizar con esta herramienta, dispone de multitud de opciones y de una barra específica para ello ubicada en la parte superior izquierda (observar figura siguiente). Además de la barra también se podrán utilizar estas herramientas a través de tres pestañas situadas en la parte superior llamadas Run, CVS y Profile.

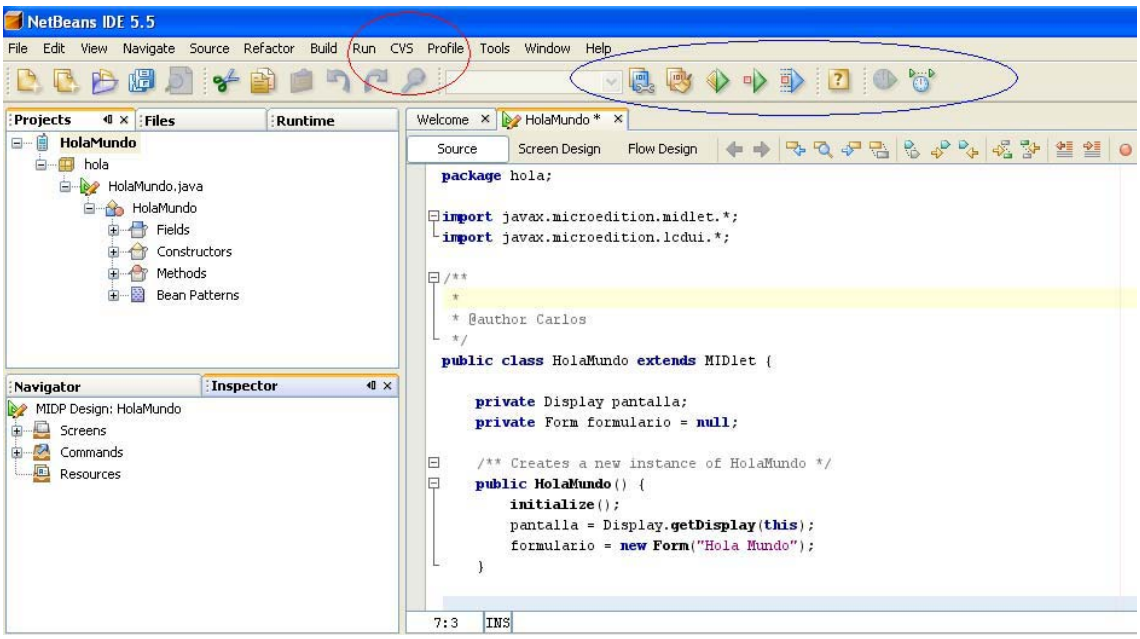


Ilustración 11 - NetBeans IDE (2)

Ilustración 12 - barra de emulación

Esta figura corresponde a la barra de emulación y debugación que había sido nombrada anteriormente. Primero se explicaran los botones de emulación ya que los de debugación serán vistos en el siguiente apartado llamado Manejo del debugador de Netbeans. Los botones correspondientes al emulador son:

• El primer desplegable se corresponde con la configuración que fue elegida en la creación del proyecto (como se puede observar es DefaultConfiguration).

• El primer y el segundo botón son para compilar el proyecto directamente (build main project) y para compilar y limpiar el proyecto (clean and build main proyect) respectivamente.

• El tercer botón es el más importante en este caso, este es el botón de emulación simple. Como antes ya se ha mencionado una vez tienes el proyecto compilado, pulsando este botón se abrirá la pantalla de emulación para poder ver la aplicación

creada ejecutándose. 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 y este reaccionara de una manera u otra según la tecla que sea clicada. Es muy realista, permite la perfecta comprobación del funcionamiento de una aplicación sin el embrollo de instalar la aplicación en el MID y disminuyendo los riesgos de la posible instalación en el MID de una aplicación inestable. En la figura de al lado se muestra una vista del emulador con la aplicación Hola Mundo ejecutándose en el.

Ilustración 13 - "hola mundo" en simulador

• El cuarto botón es para lanzar el debugador y el emulador a la vez. Este botón (Debug Main Proyect) lanzará la KVM (Kilobyte Virtual Machine) ejecutando emulador y debugador al unísono, permitiendo al programador comprobar los fallos del programa directamente sobre el emulador. Utilizando esta herramienta se puede interactuar con el emulador y el debugador ira resaltando por pantalla cada una de las incidencias o acciones que están ocurriendo.

• El quinto botón se verá en el siguiente apartado ya que es perteneciente al debugador puramente, sirve para ejecutar una aplicación configurando una conexión. Con él se podrán probar aplicaciones que requieren uso de sockets.

• El sexto botón consiste en una ayuda en tiempo de ejecución, con el se podrán consultar detalles de la implementación de la aplicación dinámicamente.

• El séptimo y octavo botón sirven para comprobar el resto de detalles de la aplicación como el consumo de recursos de la misma, la velocidad de ejecución, el modo en que se ejecuta un trozo de código especifico. Son meramente para optimización de los recursos e información sobre la aplicación.

La segunda manera de lanzar el emulador antes mencionada es con el desplegable de la parte superior de la pantalla llamado Run. Este incorpora algunas opciones más que la barra hasta ahora explicada y su uso es similar. Contiene los botones de la barra, además de algunos otros para ejecución paso a paso, introducción de Breakpoints, control de ejecución, saltos en código, etc.

Capítulo IV – Medios empleados durante la implementación y pruebas

Para la implementación y pruebas se ha empleado el entorno de desarrollo NetBeans con el paquete de movilidad, usando el simulador que incorpora para las pruebas de las funciones más básicas.

El completo funcionamiento de las aplicaciones se ha ido probando en móviles Nokia con sistema operativo Symbian (aunque los dispositivos en los que se use, no tienen por qué tener este sistema operativo).

Para el módulo GPS usamos una antena externa conectada mediante tecnología Bluetooth al móvil en el cual se ejecuta la aplicación servidor, para ello hay que vincular la antena al dispositivo, y una vez vinculada se realizan las conexiones automáticamente. Si la conexión Bluetooth del dispositivo está desactivada, al intentar comunicar con la antena el dispositivo preguntara si se desea activar esta conexión.

Ilustración 14 – Simulador **Ilustración 15 - Movil nokia**

Capítulo V – Introducción al sistema de posicionamiento GPS

El Global Positioning System (**GPS**) o **Sistema de Posicionamiento Global** (más conocido con las siglas *GPS*, aunque su nombre correcto es *NAVSTAR-GPS*) es un Sistema Global de Navegación por Satélite (GNSS) que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave, con una precisión hasta de centímetros, usando GPS diferencial, aunque lo habitual son unos pocos metros. Aunque su invención se atribuye a los gobiernos francés y belga, el sistema fue desarrollado e instalado, y actualmente es operado por el Departamento de Defensa de los Estados Unidos.

El GPS funciona mediante una red de 27 satélites (24 operativos y 3 de respaldo) en órbita sobre el globo, a 20.200 km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la posición y el reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el retraso de las señales; es decir, la distancia al satélite. Por "triangulación" calcula la posición en que éste se encuentra. La triangulación en el caso del GPS, a diferencia del caso 2-D que consiste en averiguar el ángulo respecto de puntos conocidos, se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que llevan a bordo cada uno de los satélites.

La antigua Unión Soviética tenía un sistema similar llamado GLONASS, ahora gestionado por la Federación Rusa. Actualmente la Unión Europea está desarrollando su propio sistema de posicionamiento por satélite, denominado Galileo.

Capítulo VI – Introducción al servicio de mensajería de texto (SMS)

El **servicio de mensajes cortos** o **SMS** (*Short Message Service*) es un servicio disponible en los teléfonos móviles que permite el envío de mensajes cortos (también conocidos como mensajes de texto, o más coloquialmente, textos o mensajitos) entre teléfonos móviles, teléfonos fijos y otros dispositivos de mano. SMS fue diseñado originariamente como parte del estándar de telefonía móvil digital GSM, pero en la actualidad está disponible en una amplia variedad de redes, incluyendo las redes 3G.

Un mensaje SMS es una cadena alfanumérica de hasta 160 caracteres de 7 bits, y cuyo encapsulado incluye una serie de parámetros. En principio, se emplean para enviar y recibir mensajes de texto normal, pero existen extensiones del protocolo básico que permiten incluir otros tipos de contenido, dar formato a los mensajes o encadenar varios mensajes de texto para permitir mayor longitud (formatos de SMS con imagen de Nokia, tonos IMY de Ericsson, estándar EMS para dar formato al texto e incluir imágenes y sonidos de pequeño tamaño...).

Cuando un usuario envía un SMS, o lo recibe, se incluyen con su payload (carga útil o cuerpo del mensaje) al menos los siguientes parámetros:

- Fecha de envío (también llamada *timestamp);*
- Validez del mensaje, desde una hora hasta una semana;
- Número de teléfono del remitente y del destinatario;
- Número del SMSC (centro de mensajes cortos) que ha originado el mensaje;

De este modo se asegura el correcto procesamiento del mensaje en el SMSC y a lo largo de toda la cadena.

Ilustración 16 - mensajes por el mundo

Capítulo VII – Características de la aplicación

Diferentes aplicaciones

Hemos dividido nuestra aplicación principalmente en dos partes, en la que cada una de ellas tiene una función diferente y se complementan. Se ha llamado a una de ellas "aplicación cliente" y a la otra "aplicación servidor". A su vez la aplicación cliente tiene una versión extendida con características avanzadas que se han añadido, pero que son prescindibles para el funcionamiento/cumplimiento de nuestro objetivo.

- **Aplicación Servidor:** es la que se está ejecutando siempre en el dispositivo que vamos a localizar (en 2º plano), se inicia automáticamente al encender el móvil.
- **Aplicación Cliente:** ésta se ejecuta en otro dispositivo cualquiera, en el momento que queramos localizar a alguien. Hay una versión básica y sencilla, y otra avanzada con muchas opciones que tiene en cuenta y nos permite usar.

Aplicación servidor

Esta aplicación se ejecuta en el móvil que va a ser localizado. Cuando se enciende el dispositivo, la aplicación está preparada para iniciarse automáticamente y permanecer en segundo plano sin límite de tiempo, es decir, desde que el móvil es encendido, la aplicación está activa en segundo plano (oculta a la vista para el usuario) indefinidamente, hasta que se apague el dispositivo o sea desactivada/cerrada por el usuario.

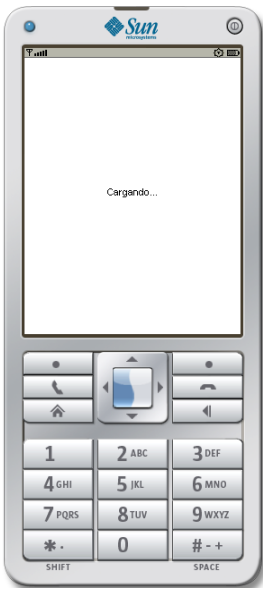


Ilustración 17- Pantalla espera de la aplicación servidor

Una vez iniciada la aplicación, ésta se prepara quedándose a la espera de recibir un mensaje de texto de forma asíncrona, es decir se queda bloqueado (parada su ejecución) en el método "receive", ya que este método es bloqueante cuando usamos la recepción de mensajes de forma asíncrona.

En un instante cualquiera, se recibe un mensaje de petición, momento en el cual la aplicación servidor reanuda su ejecución. Pero este mensaje no es un mensaje de texto cualquiera, sino uno que contiene una cadena de texto clave. Solamente si el mensaje contiene la cadena clave prefijada será tratado por nuestra aplicación, aparte el mensaje tiene que haber sido enviado con un identificador o puerto "5555", ya que si se enviara de la forma normal, el mensaje sería recibido y tratado por la aplicación del móvil que trata todos los SMS. Por lo tanto el mensaje con su identificador le llega a nuestra aplicación (que escucha en ese puerto-identificador) y empieza ser tratado. La aplicación que recibe SMS por defecto en el móvil no recibe nada, por lo que el mensaje no aparecerá reflejado en la bandeja de entrada ni en ningún sitio.

Ilustración 18 - Aplicación servidor en segundo plano

Llegado este momento, el programa tiene que comprobar la cadena de texto clave que ha recibido, ya que hay dos posibles cadenas. Esto se ha hecho así para poder diferenciar si la aplicación cliente que nos envía la petición de localización se trata de la versión normal o la versión extendida, si se trata de la normal, el mensaje o mensajes respuesta que se envíen, serán enviados sin identificador y tratados por la aplicación de mensajería por defecto en el móvil. Por el contrario si es la aplicación extendida, la respuesta será enviada con el identificador para que sea tratada por la aplicación cliente y no se reciba como mensaje por defecto.

El siguiente paso es prepararse para la comunicación con el dispositivo GPS, ya que éste es externo y se conecta mediante Bluetooth.

Ilustración 19 - Señal GPS por todo el mundo

Comprobamos si existe un medio (dispositivo) para poder realizar el posicionamiento, si no existiera enviaríamos la respuesta avisando de ello. Si existe, preparamos a la aplicación para que pueda gestionar los diferentes eventos y actualizaciones que le vaya comunicando el dispositivo GPS, momento en el cual empieza la comunicación entre ambos.

Una vez establecida la conexión entre el dispositivo y el GPS, éste puede devolver como respuesta, tres posibles situaciones diferentes:

·**Disponible**: El dispositivo GPS está encendido y además tiene cobertura, es decir, está listo para posicionarse.

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

·**Fuera de servicio***:* El dispositivo está apagado o no se puede conectar con él (por ejemplo si está demasiado lejos).

Dependiendo de la respuesta del módulo GPS, se actúa de una forma u otra. Se ha implementado un algoritmo similar al algoritmo de *back-off*.

Básicamente consiste en:

· Si la respuesta del GPS directamente es “Disponible” , entonces se procede a consultar la posición y a transformar el resultado a coordenadas del tipo

"latitud, longitud", se crea el mensaje respuesta y se le envía al cliente que nos envió la petición.

·Si la respuesta del GPS directamente es "Fuera de servicio", entonces se crea un mensaje con dicha información y se le envía al cliente como respuesta.

·Si la respuesta es "Temporalmente no disponible", entonces se entra en un bucle, en el cual se espera un tiempo fijado y se vuelve a comprobar cuál es el estado del dispositivo GPS. Si el estado ha cambiado a "Disponible" o a "Fuera de servicio", entonces se crea un mensaje de texto con la correspondiente información (coordenadas con la posición si el estado es igual a disponible o por el contrario avisar de la imposibilidad de efectuar la localización). Si el estado sigue siendo "Temporalmente no disponible", se vuelve a esperar un tiempo fijado, mayor que el anterior y este proceso lo repetimos un número determinado de veces, hasta que si al final no ha cambiado el estado, se crea un mensaje informando de que no es posible realizar el posicionamiento y se envía como respuesta al cliente.

Una vez transcurrido todo este proceso, se interrumpe/destruye la comunicación o consulta de estado con el GPS y se vuelve con el bucle al principio del programa, donde éste se prepara para recibir un nuevo mensaje y se queda a la espera, otra vez bloqueando su ejecución.

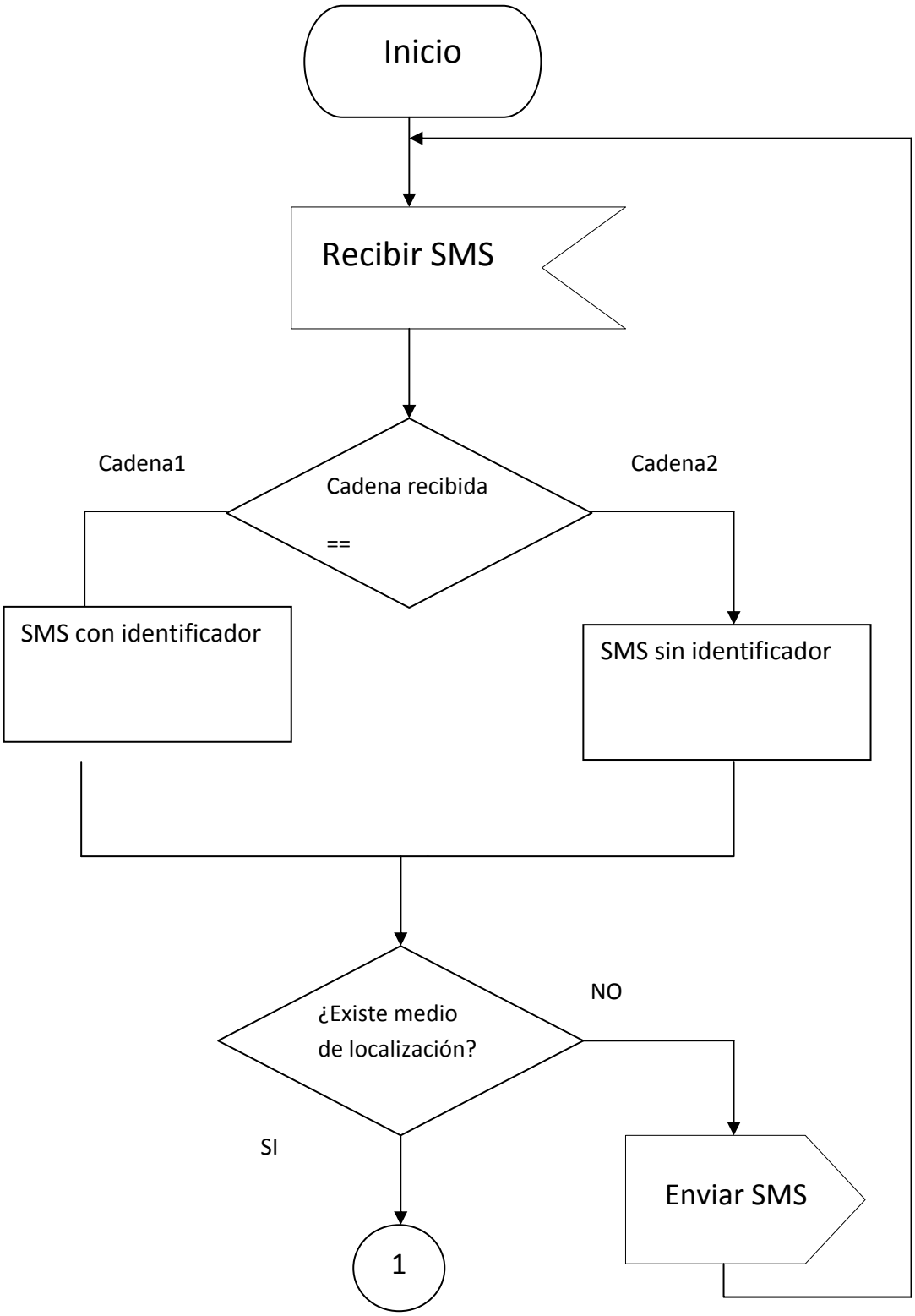
Ilustración 20 - Antena GPS externa, por BLuetooth

Ejemplo, parte del principio de la aplicación:

```
while(true){                                    //La aplicación se ejecuta
indefinidamente


    recibidor = new RecibirSMS("sms://:5555");          //Creamos recibidor,
el cuals e encargará de esperar a que le llegue un mensaje con identificador 5555

    String [] aux = recibidor.getMensaje();             //Pedimos la cadena de
texto que contiene el mensaje y el nº de tel.

    destinatario = aux[0];                          //Número de teléfono del cliente


    if(aux[1].equals("¿Donde estas?")){             //Sólo hemos de tratar el
mensaje si es nuestra cadena clave

      remitente = new EnvioSMS(destinatario, "5555");       //Preparamos el
envío del mensaje, en este caso al identificador 5555 (con aplicacion compleja)

    }

    else if(aux[1].equals("¿Donde estas?1")){

      remitente = new EnvioSMS(destinatario, null);         //Vamos a realizar el
envío del mensaje de forma que lo reciba como mensaje estándar (la aplicación simple)

    }

    else continue;


    if (!hasLocationAPI()) {

      remitente.setMensaje("Imposible activar el Navegador");    //En el caso de que
no haya dispositivo localizador, le enviamos ésto

            hilo = new Thread(remitente);

            hilo.start();
```

```
}
else{
        estado[1]=1;
        createLocationProvider();
        posicionarse();
        long tiempo = 15000;
        int i = 0;

         while(i<5){….
```

Diagrama de flujo de la ejecución, aplicación servidor.



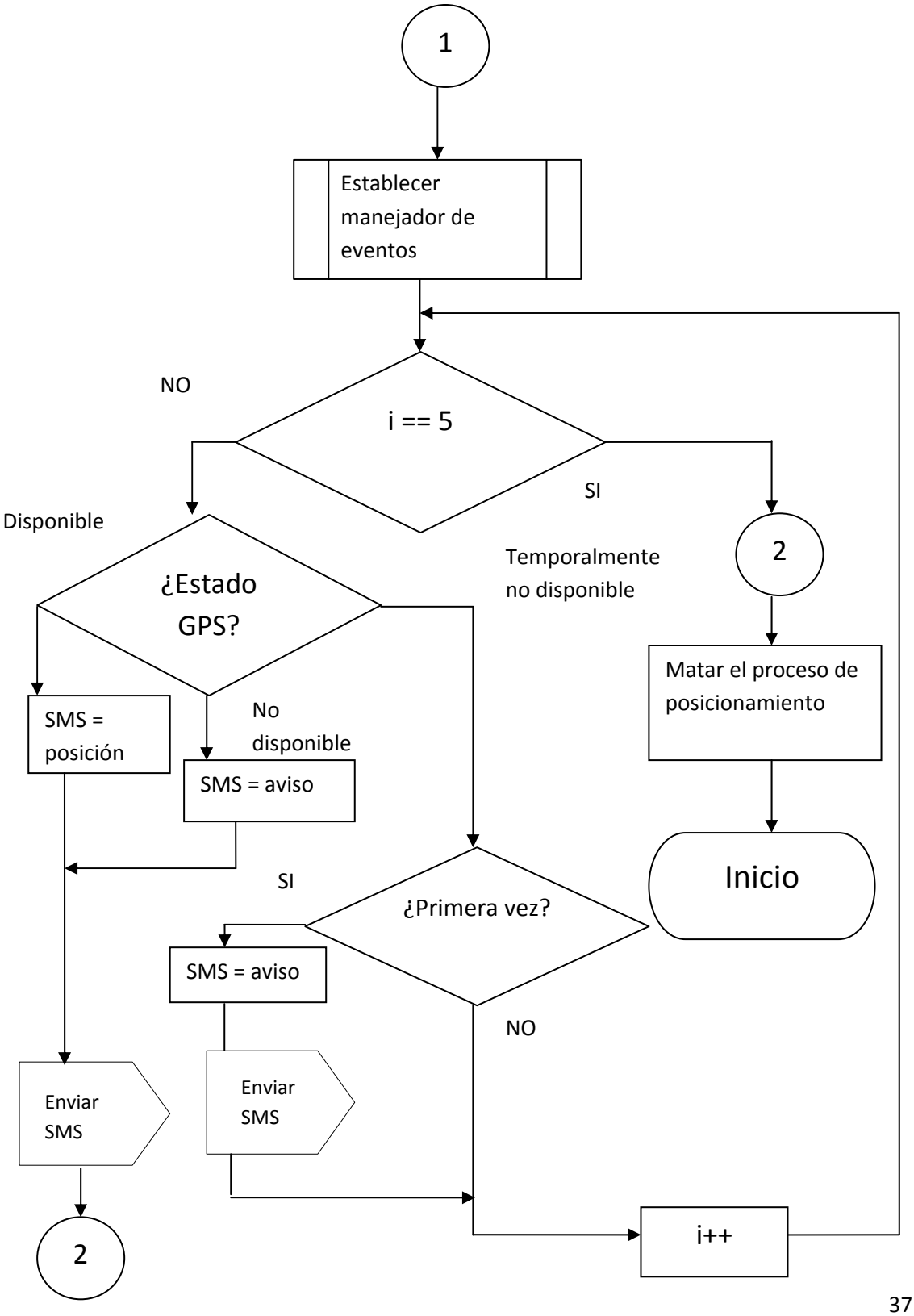
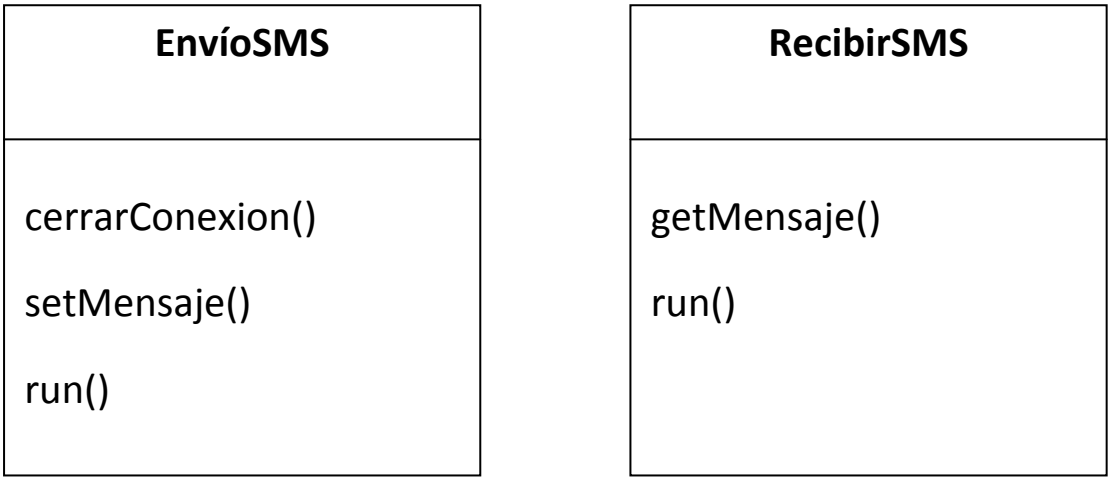

1
Establecer manejador de eventos
NO
i == 5
SI
Disponible
¿Estado GPS?
Temporalmente no disponible
2
Matar el proceso de posicionamiento
SMS = posición
No disponible
SMS = aviso
Inicio
SI
¿Primera vez?
SMS = aviso
NO
Enviar SMS
Enviar SMS
2
i++

Diagrama simple de clases UML.



Estas dos clases se incluyen en el proyecto de la aplicación servidor, son clases complementarias a la principal.

La primera de ellas, "EnvíoSMS" es la clase que se encarga de componer el mensaje y especificar que sea de texto, además de incluir el destinatario, identificador (si es necesario), la cadena de texto clave… Tiene que establecer la conexión y luego cerrarla (para ello usamos el método cerrarConexion), el método "setMensaje" se usa para poner o cambiar la cadena de texto que va a contener el mensaje. La clase tiene un método "run", es decir, crea un nuevo hilo independiente y es este hilo el que se encarga de establecer la conexión, realizar el envío y cerrarla.

La segunda de ellas, "RecepciónSMS" es la encargada de quedarse a la espera (bloqueando la ejecución del programa) de recibir el mensaje, recibirlo e identificar los diferentes parámetros que contiene, tales como la cadena de texto, el destinatario… El método "getMensaje" devuelve la cadena de texto que contiene el mensaje recibido, este método es llamado por la clase principal ya que ésta se encargará de tratar la cadena y decidir qué hacer dependiendo de su contenido. Esta clase también tiene un método "run", creando un hilo independiente y siendo éste el que se encarga de bloquear la ejecución a la espera de recibir un mensaje y recibirlo.

LocalizadoP
startApp()
startMidlet()
switchDisplayable()
getDisplay()
commandAction()
empezar_programa()
getSplashScreen()
hasLocationAPI()
createLocationProvider()
posicionarse()
providerStateChanged()
locationUpdated()
run()

"LocalizadoP" se trata de la clase principal de la aplicación, donde transcurren la mayoría de operaciones en el curso del programa. Contiene el único elemento de la aplicación que se muestra durante su ejecución: la pantalla en espera que se muestra unos segundos al principio y luego se oculta (pasa a segundo plano indefinidamente) la aplicación. Además incluye todos los métodos para establecer la conexión con el GPS y posicionarse (createLocationProvider, posicionarse…), iniciar la aplicación (startMidlet), realizar el tratamiento de eventos (commandAction), mostrar u ocultar algo en la pantalla (switchDisplayable). El método "run" que contiene esta clase, crea un nuevo hilo que se encarga de cconsultar la agenda, recorrerla y crear una lista con todos los contactos para mostrarla.

Aplicación cliente

Esta aplicación se ejecuta en el móvil que va a solicitar la localización de otro dispositivo. La aplicación se ejecuta exclusivamente cuando el usuario desea realizar dicha tarea y durante su ejecución es necesaria la interacción del usuario, una vez concluida la solicitud, se puede cerrar la aplicación y no es necesaria su nueva ejecución hasta que se desee volver a localizar un dispositivo.

Entonces todo empieza cuando cierto usuario tiene en su dispositivo instalada la aplicación cliente y decide que quiere saber dónde se encuentra otro dispositivo por algún motivo, teniendo éste la aplicación servidor.

Ilustración 21 - Pantalla en espera d aplicación cliente

Lo primero que sucede al ejecutarla es que aparece la primera pantalla de espera durante unos segundos y después automáticamente el programa nos pide que elijamos entre dos opciones, éstas son:

· **Introducir número**

· **Consultar la agenda del teléfono**

Ilustración 22 - Elegir cómo introducir el número

La primera opción nos permite introducir un número de teléfono manualmente al cual queremos localizar, mientras que la segunda opción consulta la agenda del teléfono móvil, creando una lista con todos los contactos y sus números disponibles en la agenda y nos permite seleccionar el que queramos.

Ilustración 23 - Agenda convencional

Tanto en cualquiera de los dos casos como en la mayoría de momentos en el programa, tenemos la opción de volver para atrás por si nos hemos equivocado en algo o hemos cambiado de opinión, así como la opción que permite salir directamente del programa finalizando su ejecución.

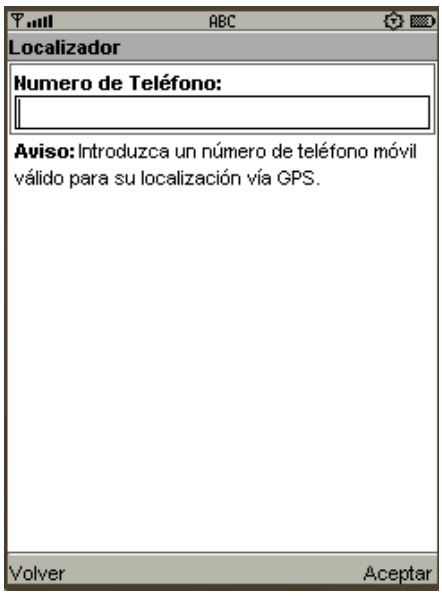


Ilustración 24 - Introducir el número de forma manual

Ilustración 25 - Se consulta la agenda y está vacia

Una vez elegido un contacto de la agenda en la lista creada para mostrarlos o escrito éste manualmente, se pide confirmación para asegurarse de que el número que vamos a localizar es el correcto y se avisa de que se va a enviar un mensaje de texto (con identificador). Si se acepta, se procede a crear el mensaje con la cadena de texto clave correspondiente y a enviarlo, mientras tanto el programa muestra durante unos

segundos la segunda pantalla de espera, la cual avisa del envío del mensaje y automáticamente después se indica que el mensaje fue enviado correctamente, dando la opción de salir de la aplicación o de volver atrás para realizar otra tarea deseada.

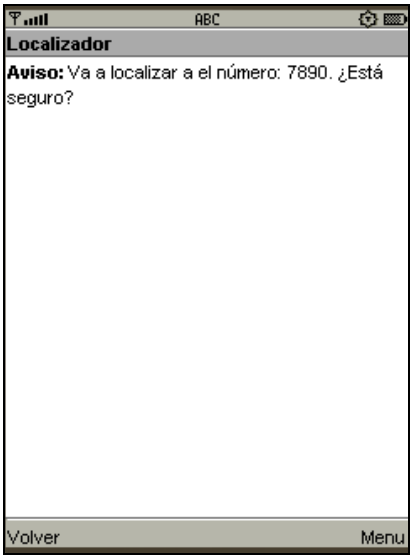


Ilustración 26 - Confirmación para localizar un número

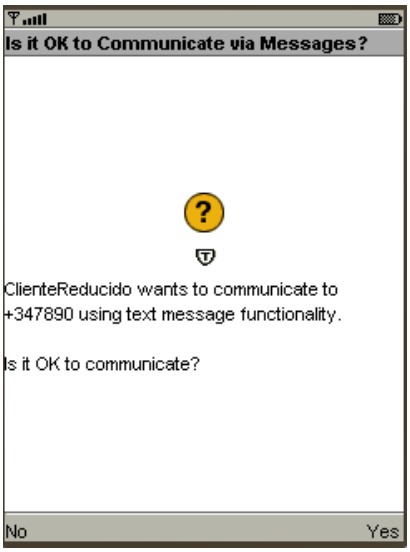


Ilustración 27 - Confirmación para el envío del mensaje

Lo normal es que el programa sea cerrado, no volviéndose a ejecutar hasta que se necesite volver a localizar a alguien. Hay que tener en cuenta que ésta versión es la normal, con lo cual la cadena clave que envía en el mensaje hacia la aplicación servidor es diferente de la que enviaría la aplicación extendida, para que ésta pueda diferenciarlas y que la respuesta nos llegue correctamente.

Ilustración 28 - Pantalla de espera mientras se envía mensaje

Es decir, el siguiente paso es esperar a que el dispositivo “servidor” nos envíe una respuesta que llegará como un mensaje de texto cualquiera (sin identificador) y será tratado por el cliente de mensajería que incorpora el móvil, por lo tanto estas respuesta(s) quedarán reflejadas como mensajes en la bandeja de entrada y podremos leerlos/consultarlos cuando queramos, todo esto sin la necesidad de usar nuestra aplicación. La respuesta puede ser la posición en coordenadas o un mensaje de aviso de que no se puede obtener la posición, si es imposible pasado un tiempo. Si la respuesta es esa, ya es decisión del usuario si quiere volver a ejecutar la aplicación e intentar localizar al mismo número de nuevo.

Ilustración 29 - El mensaje se ha enviado correctamente

Ejemplo, parte del principio de la aplicación:

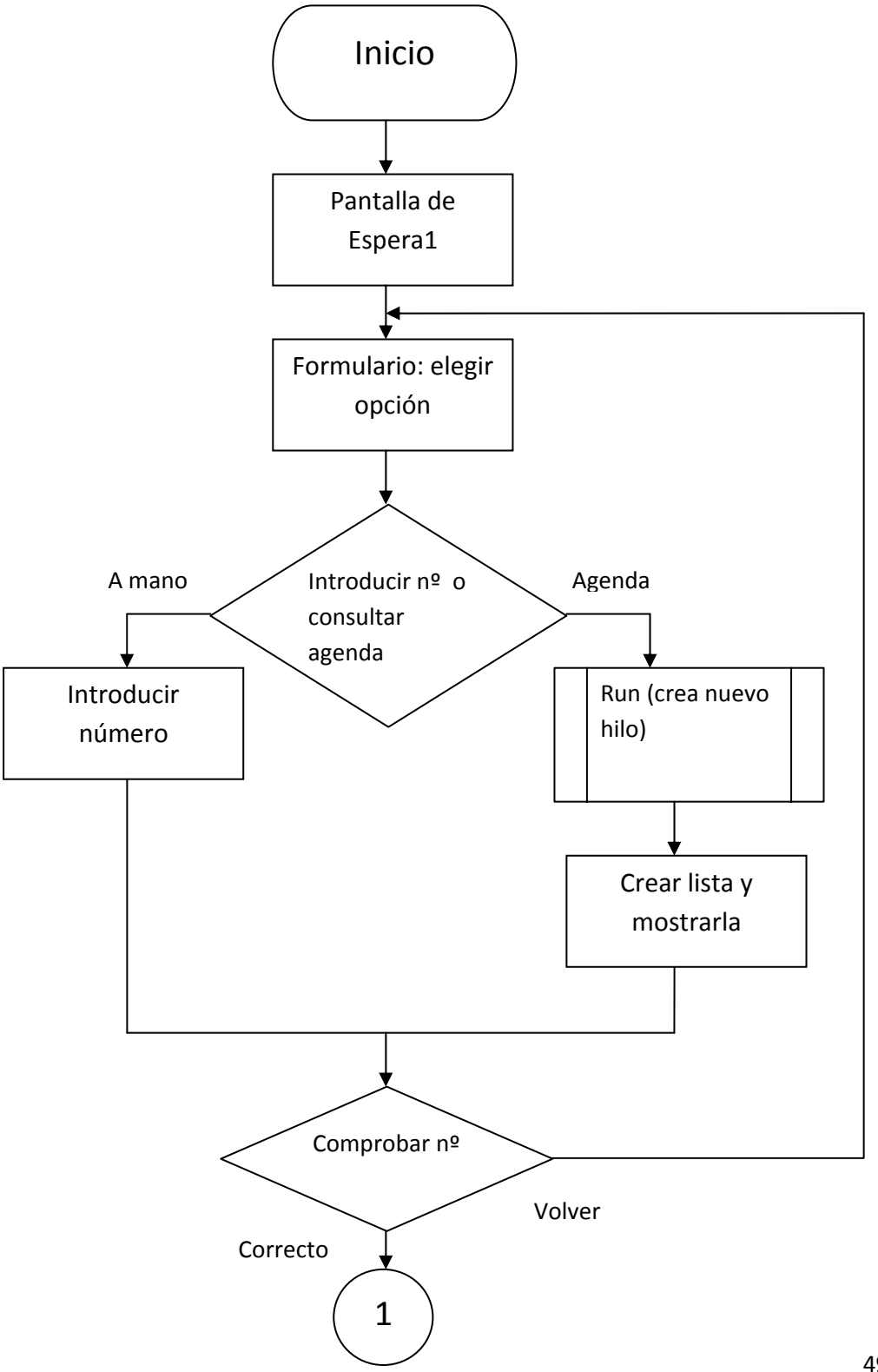
```
public void commandAction(Command command, Displayable displayable){
//Método encargado de manejar los eventos que puedan darse

    if (displayable == pantallaEspera) {                    //Tras pasar unos segundos,

      if (command == SplashScreen.DISMISS_COMMAND) {                    //vamos a
mostrar el primer formulario

        switchDisplayable(null, getFormu());

      }

    }

    else if (displayable == pantallaEspera1) {

      if (command == SplashScreen.DISMISS_COMMAND) {

        switchDisplayable(null, getFormu5());                    //Mostramos el formu5,
que avisa del envío correcto del mensaje

      }
```

```
            }
            else if (displayable == formu){
              if (command == salir){
                exitMIDlet();
              }
              else{                                   //En caso de que aceptemos
                int elegido=radioBoton.getSelectedIndex();          //Consultamos la opción marcada
                if(elegido==0){
                  switchDisplayable(null, getFormu1());          //Si es la primera, vamos a formu1, que permite introducir el nº a localizar
                }
                else if(elegido==1){
                  new Thread(this).start();                      //Si es la 2ª, arrancamos el hilo que se trata en esta clase, que consulta la agenda del móvil
                }
              }
            }
            else if(displayable == formu1){
              if (command == atras){                          //Atras nos lleva a formu, dónde podemos elegir de nuevo una opción
                cadena1.setText("Introduzca un número de teléfono móvil válido para su localización vía GPS."); //Nos aseguramos de que aquí en formu1, se va a mostrar lo correcto si volvemos
                switchDisplayable(null, getFormu());
              }
```

```
            else if(command == aceptar){                    //Si aceptamos,
consultamos el nº escrito en el cuadro de texto

              try{

                num= Integer.parseInt(texto.getString());   //Transformamos lo que
hay en el cuadro de texto a entero

              }

              catch(NumberFormatException e){

                cadena1.setText("¡ERROR¡ no has escrito un número válido.");   //Si se
produce esta excepción, es porque lo introducido no se corresponde con un nº, por lo que
mostramos el mensaje de error

                return;

              }

              comprobante = true;                           //Ponemos a true el booleano,
indicando que el nº lo hemos introducido manualmente, no consultándolo de la agenda

              switchDisplayable(null, getFormu3());  }      //Mostramos formu3,
que es el que pide la confirmación para la localización del nº

            }

            else if(displayable == formu2){                 //Si se muestra este
formulario, es porque el acceso a la agenda del móvil está restringido

              if(command == atras){

                switchDisplayable(null, getFormu());        //Atrás nos lleva a formu
para elegir de nuevo

              }

              else if(command == salir){                    //Salir, sale del midlet

                exitMIDlet();

              }

            }

            else if(displayable == formu3){
```

```
            if (command == atras){                    //Atrás nos lleva a formu para
elegir de nuevo
               switchDisplayable(null, getFormu());
            }
            else if(command == salir){                //Salir, sale del midlet
               exitMIDlet();   }
            else if(command == aceptar){              //Si aceptamos, se procede
al envio del mensaje localizador
               remitente = new EnvioSMS("" + num, "5555");          //Creamos la
instancia de la clase que envia el mensaje, al nº que hay en "num" y con identificador "5555"
               new Thread(remitente).start();         //Arrancamos su hilo
               switchDisplayable(null, getSplashScreen1());        //Mostramos pantalla
de espera1, mientras se envía el sms
            }
         }
```

Diagrama de flujo de la ejecución, aplicación cliente.



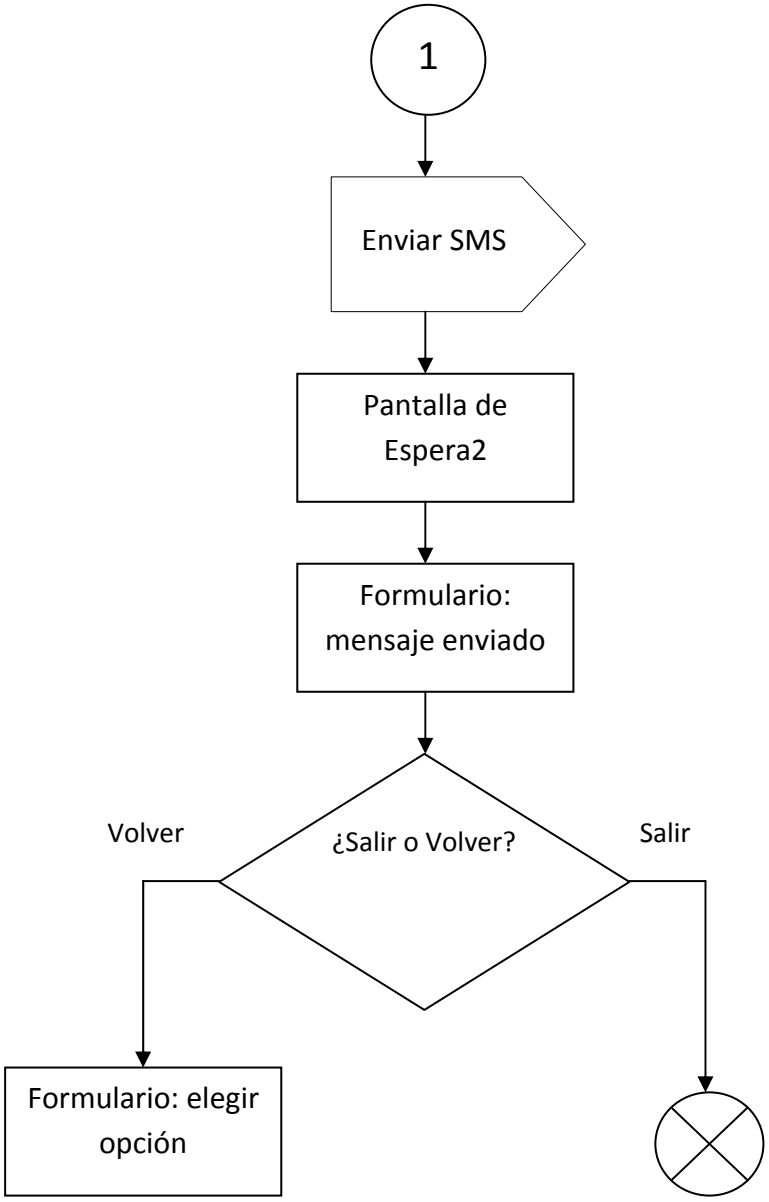

1
Enviar SMS
Pantalla de Espera2
Formulario: mensaje enviado
¿Salir o Volver?
Volver
Salir
Formulario: elegir opción

Diagrama simple de clases UML.

EnvíoSMS
cerrarConexion() run()

Lista
nuevo() buscarValor()
Clase Nodo

Nodo
Nodo siguiente
getValor() setValor() getSiguiente() setSiguiente()

La primera de ellas, “EnvíoSMS” es la clase que se encarga de componer el mensaje y especificar que sea de texto, además de incluir el destinatario, identificador (si es necesario), la cadena de texto clave… Tiene que establecer la conexión y luego cerrarla (para ello usamos el método cerrarConexion). La clase tiene un método “run”, es decir, crea un nuevo hilo independiente y es este hilo el que se encarga de establecer la conexión, realizar el envío y cerrarla.

La segunda clase, “Lista” es la encargada de crear una lista enlazada simple donde guardaremos los contactos de la agenda del móvil y los números de teléfono. Esta lista contiene los punteros necesarios para su uso y recorrido (inicio y fin), así como un método que se encarga de crear un nuevo elemento en la lista con la información requerida y añadirlo: método “nuevo()”, también el método “buscarValor()” que se usa para pasarle un valor y que recorriendo la lista te devuelva la posición del elemento en la lista que lo contiene o si no está en la lista.

La tercera clase, “Nodo” está incluida dentro de la clase “Lista”, ya que son complementarias porque un Nodo es cada elemento de los que forman la clase Lista. Cada Nodo contiene una cadena de texto con la información y un puntero a otro nodo, en nuestro caso al nodo “siguiente” (el último nodo siempre apuntará a null). Además cada nodo contiene unos métodos que sirven para consultar o modificar su información, estos métodos son:

·getValor y setValor: son los métodos que se usan para consultar/modificar el valor o contenido de un nodo (en nuestro caso un campo de texto).

·getSiguiente y setSiguiente: son los métodos que se usan para consultar/modificar puntero a siguiente, es decir el nodo al que apunta nuestro nodo.

Aplicación cliente extendida.

Esta versión de la aplicación cliente contiene todo lo que ya tiene la normal, pero además incluye ciertas características y funciones especiales que se explicarán a continuación.

La aplicación puede iniciarse de dos formas, mediante la interacción del usuario y mediante la activación automática del Midlet. La primera de ellas es la forma normal de ejecutar la aplicación y la segunda sucede cuando llega un mensaje de respuesta de la aplicación servidor que anteriormente solicitamos localizar. Al principio solamente el usuario puede iniciar la aplicación, hasta que se solicite localizar a otro móvil, entonces se registra una conexión activa que iniciará automáticamente el Midlet y se espera la respuesta, que al llegar ésta, se inicia la aplicación y comienza a tratarse la respuesta. Una vez hecho esto se des registra la conexión, por lo que hasta que no se solicite localizar a otro dispositivo, no volverá a ejecutarse automáticamente la aplicación (aunque recibiera un mensaje del mismo tipo).

Ilustración 30 - Mensaje recibido

En realidad el proceso es más complejo, se lleva una cuenta de los dispositivos pendientes de responder a nuestras peticiones y hasta que ésta no sea cero, no se des registra la conexión que permite activar automáticamente la aplicación mediante un

mensaje de texto entrante. La primera vez que se solicita localizar algún dispositivo, se pone a uno la cuenta y se registra la conexión, si se solicitan más localizaciones se va incrementando el contador y cada vez que se recibe un mensaje de contestación válido, se decrementa en uno, hasta que si es cero se anula y se vuelve a estar como al principio.

Esta cuenta se lleva a cabo en la memoria no volátil del dispositivo (registros), es decir, la información que es almacenada allí no se pierde aunque el programa finalice su ejecución o se apague el dispositivo. La primera vez que se ejecuta el programa se crea el almacén de registros donde guardaremos la información y las demás veces, solamente se abre éste.

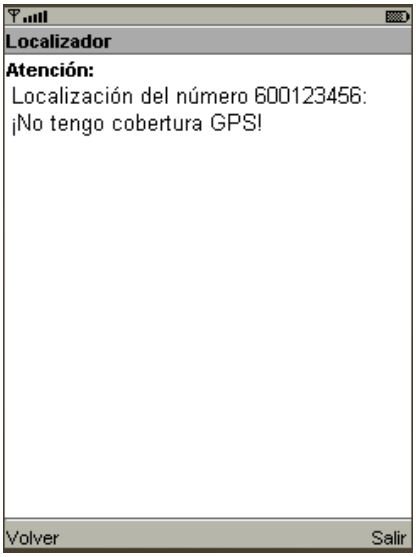


Ilustración 31 - Se muestra la localización

Se ha establecido un tiempo límite para el que se espera mientras no llegan las respuestas de las peticiones, siendo éste almacenado también en el almacén de registros (memoria no volátil). Cada vez que se envía una petición nueva se reinicia otra vez el tiempo, mientras que si llegan las respuestas a todas las peticiones, se cancela esto para que no se tenga en cuenta. Cuando el usuario ejecuta la aplicación, lo primero que se comprueba es si tenemos peticiones pendientes de responder y si el tiempo límite de espera se ha alcanzado o superado, si estas dos cosas se cumplen entonces se avisa de ello al usuario y se le da a elegir entre dos opciones:

· Borrar estas peticiones, con lo que se pondría a cero el contador, se cancelaría el tiempo de espera y se des registraría la conexión activa para la activación automática del Midlet.

·No hacer nada respecto a las peticiones y reiniciar el tiempo límite para esperarlo completamente de nuevo, por si en ese tiempo se obtuviera una respuesta por parte del servidor/es.

Si se elige la primera opción, se cancela todo y es como si no se hubiera realizado ninguna petición o hubieran llegado las respuestas correspondientes, por lo que podemos volver a intentar localizar de nuevo a algún dispositivo, cosa que volvería a registrar la conexión y poner el contador d peticiones a uno, activar el tiempo límite…. Por otro lado si elegimos la segunda opción, se vuelve a esperar el mismo tiempo y si pasado éste la segunda vez sigue sin haber respuesta de las peticiones, se vuelve a avisar y dar opción de elegir.

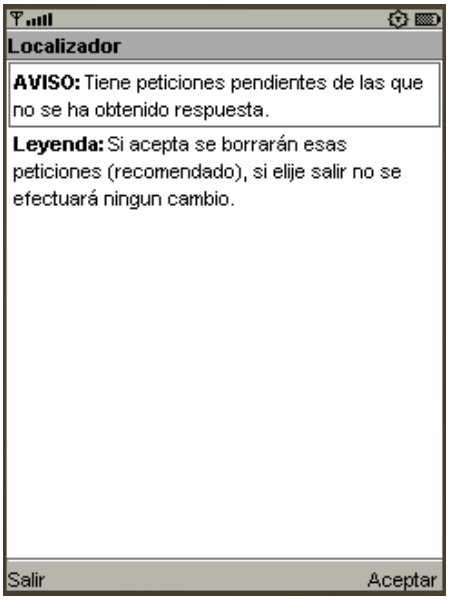


Ilustración 32 - El tiempo de espera para las peticiones ha expirado

Al encender la aplicación, se muestra la pantalla en espera 1 y después se comprueba si hay alguna conexión registrada en el registro de conexiones (push registry). Si no hay ninguna, iniciamos el programa normalmente, pero si hay alguna entonces se comprueba si esa conexión ha sido la que ha activado el Midlet, si esto

sucediera se inicia la aplicación por la parte en que se recibe y trata un mensaje de repuesta (porque acaba de llegarle uno) y si no pues se ejecuta normalmente.

Ilustración 33 - Confirmación de que la espera de respuesta se ha cancelado

Una vez que se ejecuta el programa normalmente y segundos después de mostrar la pantalla en espera 1, la aplicación muestra un formulario en el que se da a elegir dos opciones:

·Localizar un móvil: es la opción que permite introducir un número a localizar o consultar la agenda y posteriormente funciona como en la versión normal del cliente.

·Activar escucha de respuesta: una opción especial y avanzada que permite al usuario modificar según desee, el registro de la conexión para activar automáticamente el midlet.

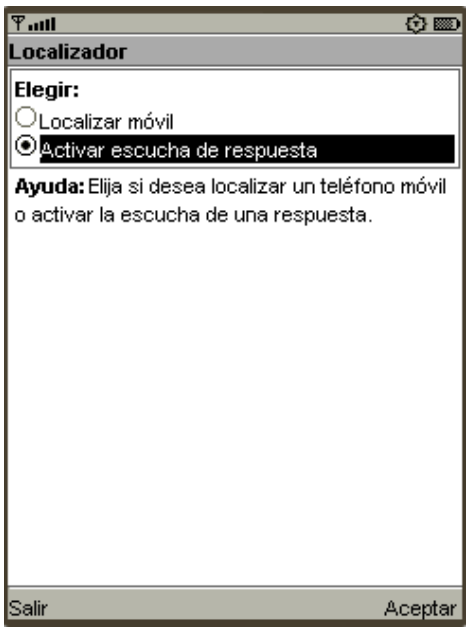


Ilustración 34 - Modificar manualmente la conexión

En la segunda opción, el usuario puede elegir entre des registrar la conexión activa o registrar esta conexión. Si elige la des registrarla, si no había ninguna no pasa nada y estaba presente, entonces se desactiva y se pone a cero el contador de peticiones a la espera de respuesta y se desactiva el tiempo límite de espera. Si por el contrario elige registrar la conexión, puede pasar que ya estuviera activa, con lo que se avisa de ello y no se produce ningún cambio, pero si la conexión no estaba registrada entonces se registra y se pone a uno el contador de peticiones a la espera de respuesta. Estas opciones son avanzadas y se pueden usar si se produce algún problema o se quiere realizar pruebas, comprobaciones de funcionamiento… pero no es necesario su uso para el completo/correcto funcionamiento de la aplicación.

La cadena de texto clave que envía la aplicación difiere de la que envía la versión cliente normal, para que la aplicación servidor las diferencie.

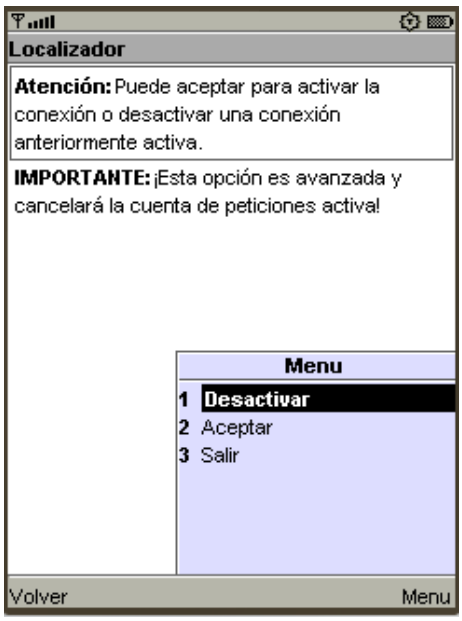


Ilustración 35 - posibles opciones avanzadas

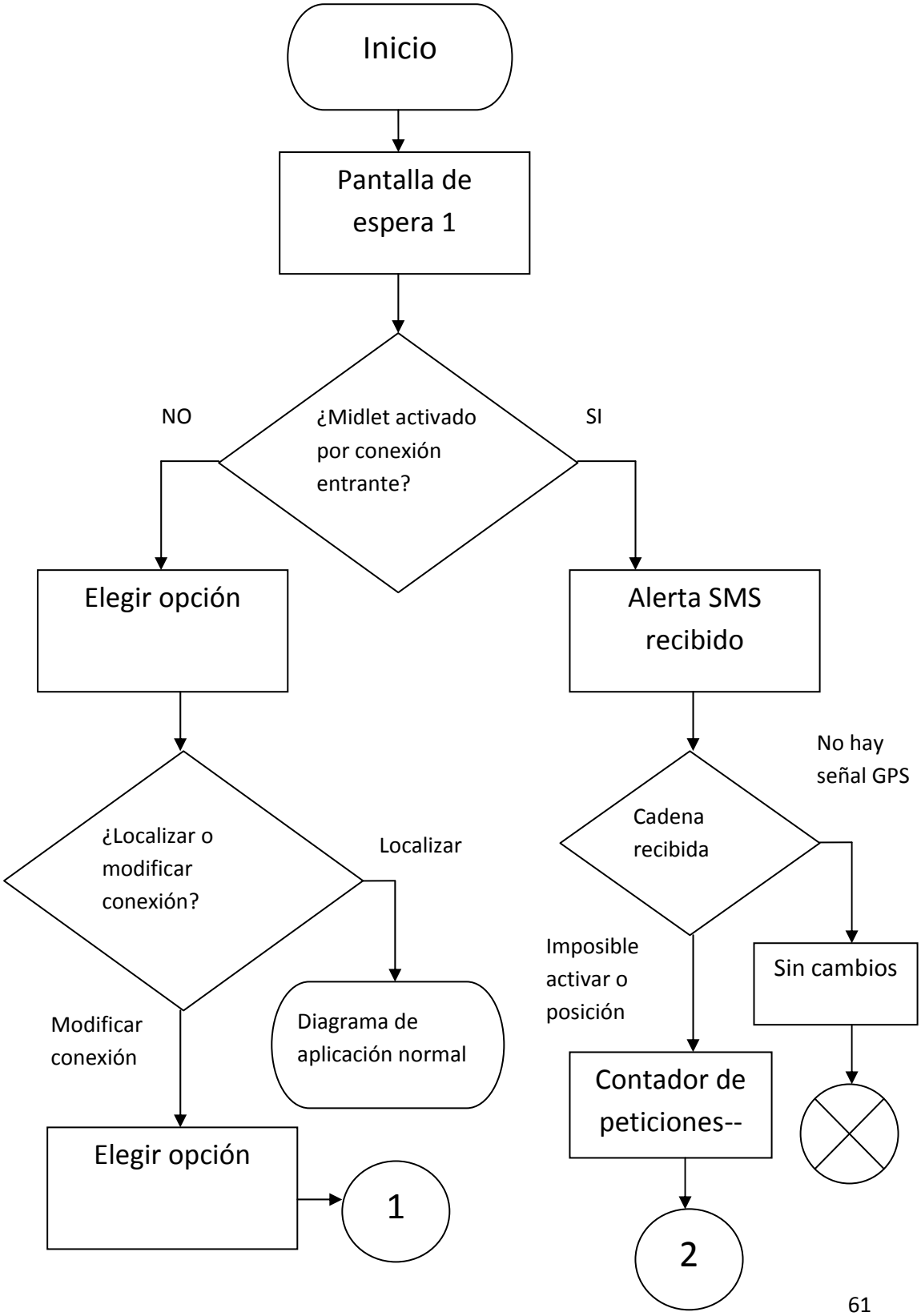
Ilustración 36 - activar manualmente la conexión

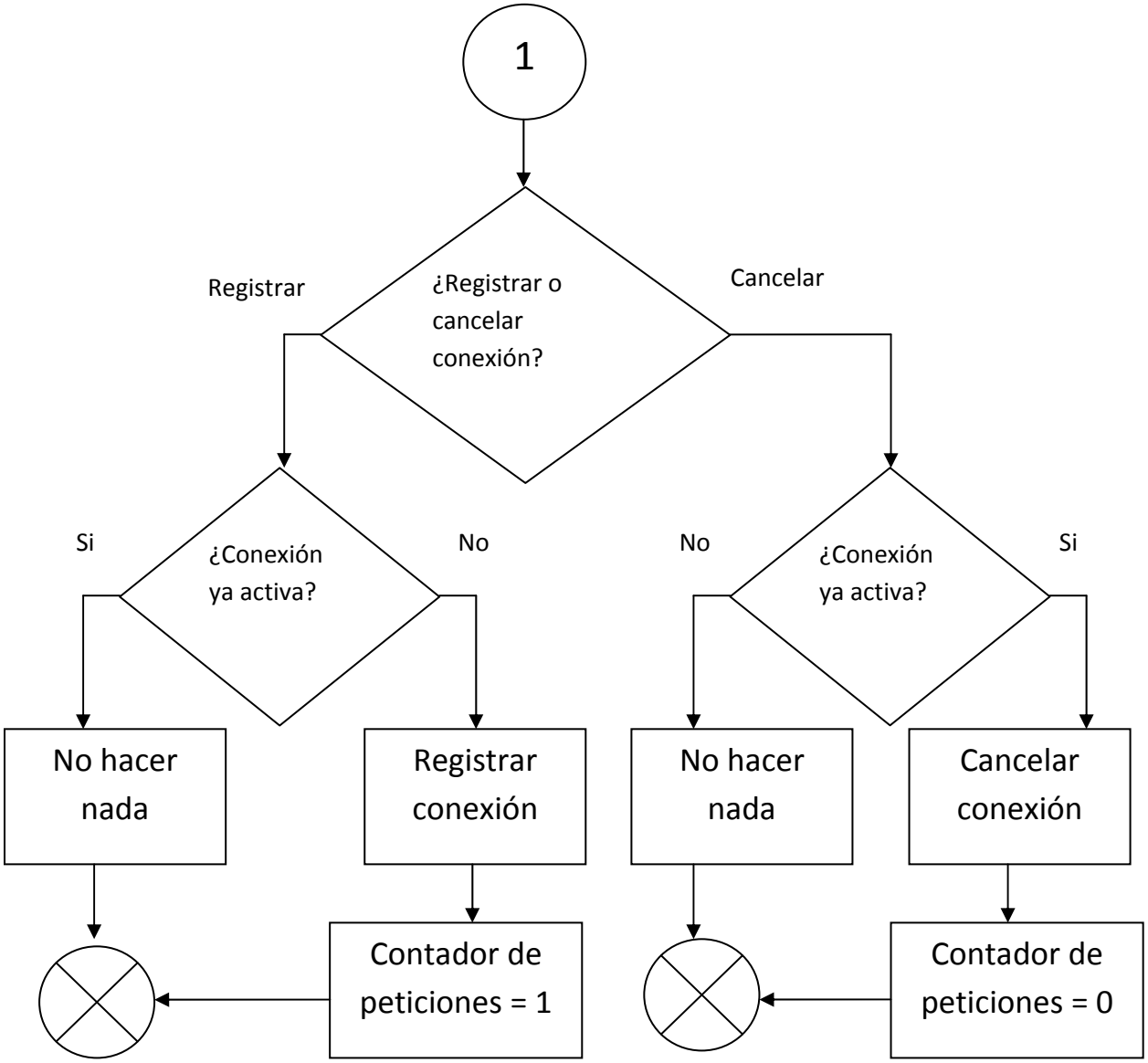
Ejemplo, parte exclusiva de la aplicación extendida:

```
package mipaquete;                                  //Clase incluida dentro de
mipaquete


import java.io.IOException;

import javax.microedition.io.PushRegistry;          //Importamos las librerías
que vamos a usar

/**

 *

 * @author José Luis

 */

public class Registro implements Runnable {        //Esta clase implementa
runnable, por lo que crea un nuevo hilo mediante el método run


    String conexion;                                //Cadena con la conexión que vamos a
registrar

    String midlet;                                  //Nombre del midlet que se encarga de
tratar esta conexión

    String filtro;                                  //Si queremos especificar algún filtro para
la activación de este midlet,ej. ciertos nº de tel.

    boolean registrar;                              //Boleano que usamos para indicar si
queremos registar la conexión o desregistrarla


    public Registro(String c, String m, String f, boolean r){      //Constructor de la clase

      conexion = c;

      midlet = m;
```

```
        filtro = f;
        registrar = r;
    }
    public void run(){                          //Método que crea el nuevo hilo
      registrarConexion();
    }

    public void registrarConexion(){
      if(registrar){                            //Si registrar vale true, se registra una
nueva conexión para este midlet
        try {
          PushRegistry.registerConnection(conexion, midlet, filtro);
        } catch (ClassNotFoundException ex) {
          ex.printStackTrace();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
      }
      else{
        PushRegistry.unregisterConnection(conexion);          //Si por el contrario
vale false, se desregistra la conexión existente
      }
    }
  }
```

Diagrama de flujo de la ejecución, aplicación cliente extendida.




1
¿Registrar o cancelar conexión?
Registrar
Cancelar
¿Conexión ya activa?
Si
No
No hacer nada
Registrar conexión
Contador de peticiones = 1
¿Conexión ya activa?
No
Si
No hacer nada
Cancelar conexión
Contador de peticiones = 0

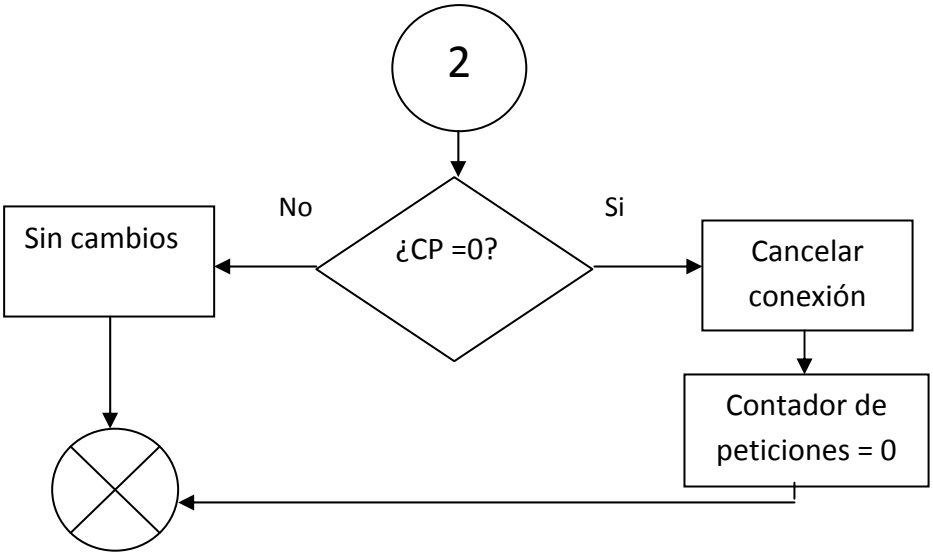

2
¿CP =0?
No
Si
Sin cambios
Cancelar conexión
Contador de peticiones = 0

Diagrama simple de clases UML.

EnvíoSMS
cerrarConexion() run()

Lista
nuevo() buscarValor()
Clase Nodo

Nodo
Nodo siguiente
getValor() setValor() getSiguiente() setSiguiente()

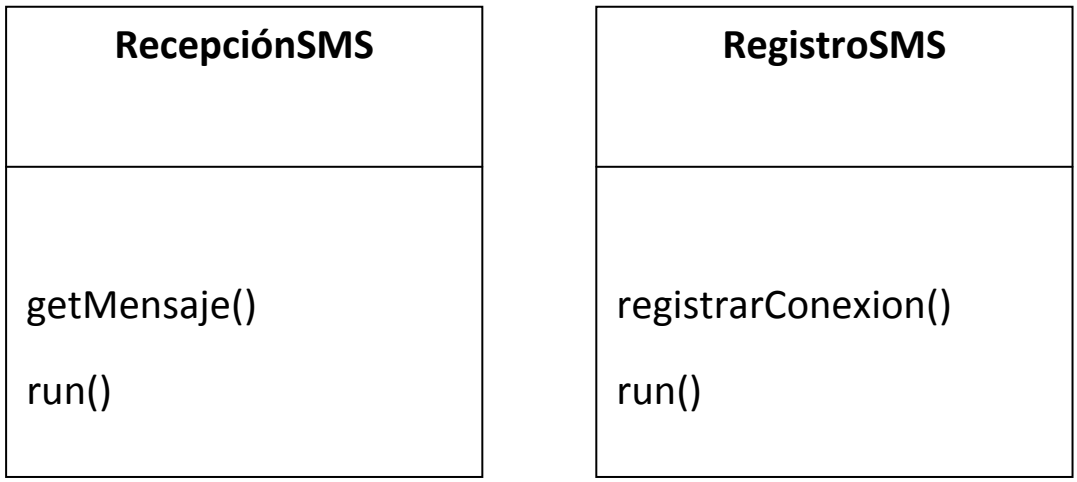
La primera de ellas, “EnvíoSMS” es la clase que se encarga de componer el mensaje y especificar que sea de texto, además de incluir el destinatario, identificador (si es necesario), la cadena de texto clave… Tiene que establecer la conexión y luego cerrarla (para ello usamos el método cerrarConexion). La clase tiene un método “run”, es decir, crea un nuevo hilo independiente y es este hilo el que se encarga de establecer la conexión, realizar el envío y cerrarla.

La segunda clase, “Lista” es la encargada de crear una lista enlazada simple donde guardaremos los contactos de la agenda del móvil y los números de teléfono. Esta lista contiene los punteros necesarios para su uso y recorrido (inicio y fin), así como un método que se encarga de crear un nuevo elemento en la lista con la información requerida y añadirlo: método “nuevo()”, también el método “buscarValor()” que se usa para pasarle un valor y que recorriendo la lista te devuelva la posición del elemento en la lista que lo contiene o si no está en la lista.

La tercera clase, “Nodo” está incluida dentro de la clase “Lista”, ya que son complementarias porque un Nodo es cada elemento de los que forman la clase Lista. Cada Nodo contiene una cadena de texto con la información y un puntero a otro nodo, en nuestro caso al nodo “siguiente” (el último nodo siempre apuntará a null). Además cada nodo contiene unos métodos que sirven para consultar o modificar su información, estos métodos son:

·getValor y setValor: son los métodos que se usan para consultar/modificar el valor o contenido de un nodo (en nuestro caso un campo de texto).

·getSiguiente y setSiguiente: son los métodos que se usan para consultar/modificar puntero a siguiente, es decir el nodo al que apunta nuestro nodo.



La clase "RecepciónSMS" se encarga de recibir el mensaje que ha llegado y tratarlo, ver su contenido, número del emisor… y crear la cadena con la información correspondiente a mostrar al usuario. El método "getMensaje()" es llamado desde el hilo principal (clase Localizador) y le devuelve a éste el número del emisor y la cadena formada con la información necesaria. Además tiene un método "run()" que crea un nuevo hilo y es éste el que se encarga de recibir el mensaje y tratar su información, en este caso la recepción es síncrona y no bloqueante, ya que solo se ejecuta esta recepción justo cuando ha llegado un mensaje.

La clase "RegistroSMS" es la que usamos para registrar o cancelar una conexión o alarma para nuestro Midlet, en el caso concreto de nuestra aplicación se usa para registrar la conexión que activará automáticamente el Midlet mediante la llegada de un mensaje de texto con un identificador especial y también para cancelar (desactivar) esta misma. El método "run()" se encarga de crear un nuevo hilo que llama al método "registrarConexion()" y éste comprueba los parámetros que se le han pasado, por ejemplo un booleano que le indica si tiene que registrar o desactivar la conexión en el *Push Registry* de nuestro dispositivo.

Capítulo VIII - RMS (Record Management System)

Introducción.

Un dispositivo móvil (al menos por ahora) no dispone de disco duro donde almacenar información permanentemente. J2ME resuelve el problema mediante el RMS (Record Management System). RMS es un pequeño sistema de bases de datos muy sencillo, pero que permite añadir información en una memoria no volátil del móvil. RMS no tiene nada que ver con JDBC debido a las limitaciones de los dispositivos J2ME, por lo tanto, el acceso y almacenamiento de la información se hace a mucho más bajo nivel. RMS no puede ser consultado con sentencias SQL ni nada parecido. En una base de datos RMS, el elemento básico es el registro (record). Un registro es la unidad de información más pequeña que puede ser almacenada. Los registros son almacenados en un recordStore que puede visualizarse como una colección de registros. Cuando almacenamos un registro en el recordStore, a éste se le asigna un identificador único que identifica unívocamente al registro.

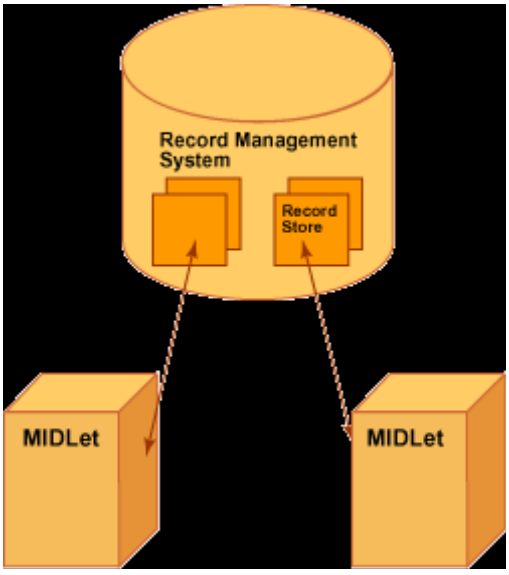


Ilustración 37 - RMS

El mecanismo básico de almacenamiento de RMS es denominado record store. Un record store es un conjunto de registros, y un registro es un byte array de datos de tamaño variable. Un record store está representado por un objeto de la clase RecordStore.

Existen reglas importantes sobre los record store:

• El nombre de un record store consiste en una combinación de hasta 32 caracteres (sensible a las mayúsculas).

• Los record stores son creados por MIDlets. Los MIDlets de un mismo MIDlet suite están almacenados en el mismo espacio de nombres, y por lo tanto, pueden compartir y ver sus contenidos.

• Los record stores creados por MIDlets en un MIDlet suite, no son accesibles para los MIDlets de otros MIDlets suite.

• El nombre de un record store debe ser único en un MIDlet suite.

Un *Record Store* tal como su nombre indica es un almacén de registros. Estos registros son la unidad básica de información que utiliza la clase RecordStore para almacenar datos. Cada uno de estos registros está formado por dos unidades:

• Un número identificador de registro (*Record* ID) que es un valor entero que realiza la función de clave primaria en la base de datos.

• Un *array* de bytes que es utilizado para almacenar la información deseada.

En la siguiente figura se muestra la estructura de un Record Store:

Record ID Datos
1 Byte [] arrayDatos
2 Byte [] arrayDatos
… …

Además de un nombre, cada *Record Store* también posee otros dos atributos:

• Número de versión: Es un valor entero que se actualiza conforme vayamos insertando, modificando o borrando registros en el *Record Store*. Podemos consultar este valor invocando al método RecordStore.getVersion().

• Marca temporal: Es un entero de tipo long que representa el número de milisegundos desde el 1 de enero de 1970 hasta el momento de realizar la última modificación en el *Record Store*. Este valor lo podemos obtener invocando al método RecordStore.getLastModified().

Operaciones con un Record Store

La clase RecordStore proporciona los siguientes métodos para crear, abrir, cerrar y borrar un record store:

• Para crear/abrir un objeto RecordStore se utiliza el método *public static RecordStore openRecordStore(String recordName, boolean createIfNecessary)*

· Para crearlo se puede utilizar: *RecordStore.openRecordStore (recordStoreName, true)*, de tal forma que si el record store con nombre recordStoreName no existe, se crea. Si existe, se "abre" el existente para trabajar sobre él.

· Para abrirlo también se puede utilizar:
RecordStore.openRecordStore(recordStoreName, false), de tal forma que si el record store no existe, se produce una excepción RecordStoreNotFoundException.

• Para cerrar un objeto RecordStore se utiliza el método *public void closeRecordStore ()*. Después de utilizar un record store siempre debe cerrarse.

• Para borrar un objeto RecordStore se utiliza el método *public static void deleteRecordStore (String recordStoreName)*. Antes de borrar un record store es necesario cerrarlo.

Cada record store mantiene al menos un campo de cabecera. Si no existe espacio suficiente para almacenar la cabecera, el record store no se creará y se producirá una RecordStoreFullException. Si ocurre otro tipo de problema, como que el nombre del record store es demasiado largo, o el record store está corrupto, se produce una RecordStoreException.

Capítulo IX – Diseño de la aplicación

Introducción a la interfaz gráfica de usuario

La interfaz gráfica de usuario (en idioma inglés **Graphical User Interface**, **GUI**) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

En el contexto del proceso de interacción persona - ordenador, la *interfaz gráfica de usuario* es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

Surge como evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplo de interfaz gráfica de usuario podemos citar el escritorio o 'desktop' del sistema operativo Windows y el entorno X-Window de Linux y también Aqua de Mac OS X.

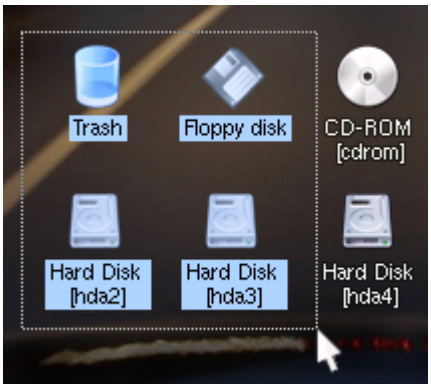


Ilustración 38 - escritorio

Interfaz gráfica en Java

En Java SE existen principalmente dos paquetes que incorporan las librerías necesarias para crear/dibujar interfaces gráficas, estas son: AWT y Swing.

·**AWT**: (Abstract Window Toolkit) contiene rutinas para soportar operaciones básicas GUI y utiliza ventanas básicas desde el sistema nativo subyacente. Muchas implementaciones independientes de la API Java implementan todo excepto AWT, el cual no es usado por la mayoría de las aplicaciones de lado de servidor. Este paquete también contiene la API de gráficos Java 2D.

·**Swing**: Swing es una colección de rutinas que se construyen sobre java.awt para suministrar un toolkit de widgets independiente de plataforma. Swing usa las rutinas de dibujado 2D para renderizar los componentes de interfaz de usuario en lugar de confiar en el soporte GUI nativo subyacente del Sistema operativo.

Swing es un sistema muy rico por sí mismo, soportando pluggable looks and feels (PLAFs) para que los controles (widgets) en la GUI puedan imitar a aquellos del sistema nativo subyacente. Los patrones de diseño impregnan el sistema, especialmente una modificación del patrón modelo-vista-controlador, el cual afloja el acoplamiento entre función y apariencia. Una inconsistencia es que (para J2SE 1.3) las fuentes son dibujadas por el sistema nativo subyacente, limitando la portabilidad de texto. Mejoras, tales como usar fuentes de mapas de bits, existen. En general, las layouts (disposiciones de elementos) se usan y mantienen los elementos dentro de una GUI consistente a través de distintas plataformas.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

La apariencia externa (el ‘’’look and feel’’’) de las aplicaciones GUI (Graphical User Interface) escritas en Java usando la plataforma Swing difiere a menudo de la que muestran aplicaciones nativas. Aunque el programador puede usar el juego de herramientas AWT (Abstract Windowing Toolkit) que genera objetos gráficos de la

plataforma nativa, el AWT no es capaz de funciones gráficas avanzadas sin sacrificar la portabilidad entre plataformas; ya que cada una tiene un conjunto de APIs distinto, especialmente para objetos gráficos de alto nivel. Las herramientas de Swing, escritas completamente en Java, evitan este problema construyendo los objetos gráficos a partir de los mecanismos de dibujo básicos que deben estar disponibles en todas las plataformas. El inconveniente es el trabajo extra requerido para conseguir la misma apariencia de la plataforma destino. Aunque esto es posible (usando GTK+ y el Look-and-Feel de Windows), la mayoría de los usuarios no saben cómo cambiar la apariencia que se proporciona por defecto por aquella que se adapta a la de la plataforma.

Ejemplos muy simples de ventanas hechas con Swing en Java:

Ilustración 39 - Hola mundo GUI

Ilustración 40 - Hola mundo con botón

Características de la interfaz en J2ME

Desde la creación de la especificación J2ME (Java 2 Platform, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores específicamente diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos. El modelo de desarrollo de estas aplicaciones es muy semejante a las *applets* de los navegadores salvo que en este caso se denominan *MIDlets*.

Como J2ME es una versión reducida o simplificada de J2SE, los paneles, contenedores, objetos… gráficos que ésta posee son también reducidos respecto a los de su versión estándar para PC. Además los dispositivos móviles, PDA´s… no tienen los mismos dispositivos e interfaces de entrada y salida que pueda tener un ordenador convencional aunque poseen algunas de sus características y otras que se asemejan a pesar de que se ven limitadas. También hay que tener en cuenta que el teclado de un dispositivo móvil en la mayoría de los casos difiere del de un ordenador personas, ya que puede ser táctil, o de diversas formas pero por lo general es un teclado bastante reducido.

A parte de las características anteriormente nombradas, hay una que es de las más importantes y es que: el sistema operativo que los pequeños dispositivos como móviles, PDA´s o similares, es muy diferente y limitado respecto al sistema operativo que un ordenador maneja (Windows, MAC…), por ello las características de la maquina virtual de Java en estos sistemas es más reducida y las ventanas de las aplicaciones tienen ciertas limitaciones. En general hay que decir que los componentes gráficos para los Midlets están reducidos respecto a sus hermanos mayores y adaptados a las necesidades y capacidades de los móviles.

Diferentes ventanas/contenedores usados.

A lo largo del desarrollo de nuestra aplicación se han ido utilizando diferentes componentes gráficos según las necesidades de cada situación. No se han usado un gran número de ellos, sino que hemos ido utilizando para todo el proyecto unos cuantos que son los principales. El manejador de eventos para estos componentes se encuentra en la clase principal. A continuación se van a describir cada uno de los componentes gráficos usados:

· **Form** (formulario): es el componente más usado en éste proyecto, es un contenedor principal vacío en el cual se insertan otros componentes, tales como cadenas de texto u otros. También contiene una serie de opciones que se le pueden asignar y se suelen situar en un botón de selección derecho y otro izquierdo que poseen gran cantidad de teléfonos (como es nuestro caso), si se asignan más de dos opciones, al menos en uno de los botones de selección se asigna un menú con las diversas opciones.

Ejemplo de formulario (form) con dos campos de texto y tres opciones para los botones (upload, exit y back):

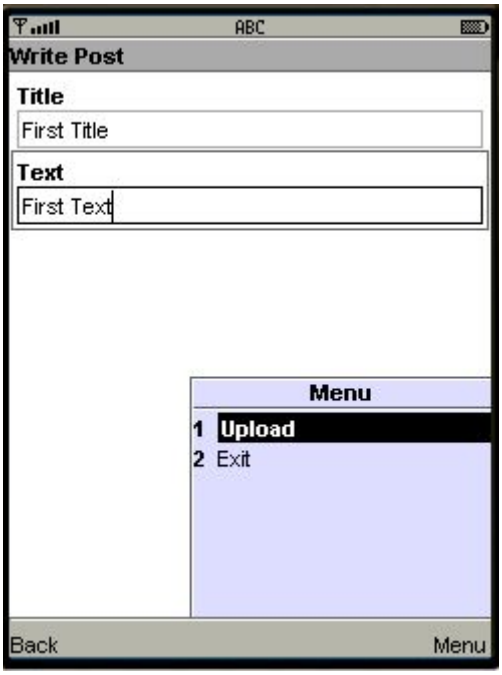


Ilustración 41 - formulario

·**TextField** (campo de texto): es un campo rectangular en el cual se permite introducir texto en general, ya sean letras o números, se le puede poner un nombre y en cualquier momento se puede consultar su contenido o modificarlo desde el código del programa. Generalmente se inserta en un formulario.

·**ChoiceGroup** (conjunto de radio botones): es un conjunto formado por dos o más botones circulares, en el que cada uno de ellos representa una opción a elegir y lleva un texto asociado. Se puede crear de forma que solo pueda elegirse una opción o varias d ellas.

·**StringItem** (cadena de texto compuesta): es un objeto que básicamente contiene una cadena de texto principal y otra secundaria, separadas por dos puntos y remarcando en negrita la principal. Lo insertamos en los formularios y en las alertas.

·**Alert** (Alerta): ésta es una ventana especial a la que se le puede asignar un título, una cadena de texto y unas opciones (botones), esto último exactamente igual que a un formulario. Puede establecerse un tiempo para que finalice la alerta o dejarla indefinidamente a la espera de un evento.

·**SplashScreen** (pantalla de espera): se trata de una pantalla especial a la que se le puede asignar un título, una imagen y un texto, además de un manejador de eventos (como ocurre con el formulario y la alerta). Puede establecerse a pantalla completa y una vez mostrada, transcurridos unos segundos o pulsando algún botón, se trata su evento.

·**Image** (imagen): es un objeto que se usa para cargar una imagen desde nuestro disco duro y poder incorporarla posteriormente donde se desee.

·**List** (lista): una lista ordenada de elementos en la cual se puede ir avanzando o retrocediendo de uno en uno, hasta llegar al deseado y seleccionarlo, sobre el cual queda constancia y se puede actuar.

·**Command** (Comando): es una opción de selección que se asocia a algunos componentes, es decir, es como un botón del tipo “aceptar”, “cancelar”… Puede ser de tres tipos: OK (aceptar), BACK (volver), EXIT (salir). Cuando una de estas opciones es seleccionada pulsando la correspondiente tecla del móvil, salta el manejador de eventos que se le ha asociado al crear el comando.

Ejemplos de diferentes componentes anteriormente descritos:

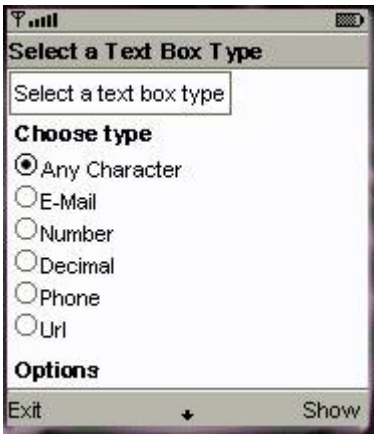


Ilustración 42 - choice group (con opciones show y exit)

Ilustración 43 - SplashScreen (a pantalla completa y con imagen):

Ilustración 44 - Alerta (con comando done)

Anexo – Documentación técnica

El más importante de los detalles es que para realizar ciertas tareas u operaciones, la aplicación pide confirmación al usuario en vez de realizarlas automáticamente, esto depende de cada operación y los permisos que tenga asociados. Esta norma actual de requerir autentificación y controlar los permisos para ciertas tareas, se impuso con Java MIDP 2.0 y este es usado actualmente y también MIDP 2.1 (una pequeña actualización de 2.0), nuestra aplicación esta implementada con la especificación 2.0.

En las opciones del teléfono móvil, en aplicaciones, aparece una lista de operaciones y los diferentes permisos que se le pueden asignar, estos son: Permitir siempre, preguntar siempre, preguntar la primera vez y no permitir. Algunas operaciones que requieren de estos permisos son: Enviar y recibir mensajes, acceder a la agenda del teléfono, acceder a algún directorio de la memoria, usar la conexión Bluetooth, usar la cámara, iniciar automáticamente la aplicación…

Esto es un problema si se quiere que la aplicación haga o ejecute todo automáticamente sin la interacción del usuario, debido a que a algunas operaciones no les está permitido el permiso de "preguntar siempre" e incluso a otras ni si quiera el de "preguntar sólo la primera vez". La mejor solución a esto, si se necesita tener una aplicación consistente sin interacción alguna del usuario, es firmar la aplicación digitalmente, lo que conlleva obtener una firma o certificado digital por parte de una empresa autorizada que supervise nuestra aplicación y nos conceda dicho certificado, con el cual la aplicación tendría vía libre para poner cualquier operación con el permiso "siempre permitir".

El GPS Bluetooth dispone de una batería que ha de cargarse, tiene tres luces, una naranja fija a la izquierda que indica que se está cargando (cuando está enchufado a la corriente) o que queda poca batería (si no está enchufado), una luz azul a la derecha que parpadea cuando se está comunicando vía Bluetooth y por último una luz en el centro amarilla que si es fija indica que el dispositivo GPS está encendido y si parpadea, que además capta la señal de los satélites.

La aplicación dispone de un pequeño icono que se muestra en el menú del teléfono móvil, es una pequeña imagen que se adjunta a nuestro proyecto, en nuestro caso usando las opciones del entorno de desarrollo Netbeans.

Conclusión

Tras el desarrollo de nuestro proyecto, nos hemos dado cuenta de la gran cantidad de ventajas y algunas desventajas de utilizar la plataforma Java, que desde nuestro punto de vista, son muchas más las ventajas y consideramos que este lenguaje nos ofrece una gran cantidad de herramientas y capacidades para el desarrollo de nuestra aplicación, tanto como otras muchas que podrían usarse en diversas aplicaciones de mayor o menor índole e importancia. Gracias a las API que J2ME nos ofrece para usar la mayoría de recursos, seríamos capaces de implementar una aplicación de casi cualquier tipo y que cumpla las necesidades deseadas en función de la demanda.

En nuestro caso concreto, la posibilidad de enviar y recibir mensajes de texto, acceder a la agenda del teléfono, usar la conexión Bluetooth para comunicarnos con el GPS… han hecho posible cumplir los objetivos propuestos que deseábamos resolver mediante el desarrollo de nuestro proyecto. Combinando todas estas posibilidades, se podrían resolver infinidad de problemas, siendo la capacidad del manejo de un dispositivo GPS una gran ventaja que podría ir mucho más allá de los objetivos de nuestra aplicación.

Pensamos que la aplicación aquí desarrollada es muy útil para localizar a una persona o a un objeto en cualquier momento deseado mediante un teléfono móvil (dispositivo que la mayoría de personas tienen) que pueda ejecutar aplicaciones Java y enviar mensajes de texto (todos los móviles en la actualidad realizan ambas cosas), mediante una interfaz sencilla y fácil de manejar por cualquier usuario.

El servicio que se ofrece tendría especial utilidad en personas en peligro, o que se las necesite tener localizadas en algún momento, ya sea por algún problema o necesidad e incluso estas personas pueden ser localizadas sin ser conscientes de ello, si en su teléfono móvil está funcionando oculta nuestra aplicación y éste lleva GPS incorporado.

Manual de usuario

Instalación (ejemplo para nokia)

Tanto la aplicación cliente como la servidor se instalan de la misma forma, para ello es necesario primero pasar de alguna forma el archivo *.jar de nuestra aplicación al móvil. Las diferentes formas pueden ser:

·Mediante el cable de transmisión de datos entre el móvil y el ordenador.

·A través de la conexión Bluetooth, desde un ordenador u otro dispositivo.

·Mediante la inserción de una tarjeta de memoria con el archivo en su interior.

Si usamos el cable o la tarjeta de memoria, para llevar a cabo la instalación tendremos que, desde el menú principal ir a “ajustes”, después entrar en “gestor de datos” y por último en “gestor de archivos”. Una vez hecho esto buscamos el archivo y lo abrimos, lo cual nos pedirá confirmación, preguntara si queremos instalar en la memoria del dispositivo o en la de la tarjeta e iniciará la instalación de la aplicación.

Si utilizamos la conexión Bluetooth, el archivo recibido llegará como un mensaje dl tipo Bluetooth, con lo que iremos al menú principal, a “mensajes” y a “buzón de entrada”. Al abrir el mensaje, se va a iniciar la instalación procediendo igual que en el caso anterior.

Por defecto las aplicaciones que se instalan en el móvil, aparecen en la carpeta “Mis cosas”, 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.

Desinstalación (ejemplo para nokia)

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". Otra alternativa es ir desde el menú principal a "ajustes", "gestor de datos" y "gestor de aplicaciones", dónde veremos la lista de todas las aplicaciones instaladas en nuestro teléfono y podremos elegir la que queremos desinstalar o modificar alguno de sus parámetros.

Requisitos mínimos

Para que la aplicación pueda ser instalada es necesario que el dispositivo incorpore la plataforma J2ME de Java, versión MIDP 2.0 y que no tenga restringido el permiso para la instalación de aplicaciones. El dispositivo que ejecuta la aplicación servidor, ha de disponer de conexión Bluetooth para poder comunicarse con la antena GPS (si esta es externa). Para poder acceder a la agenda del teléfono, éste ha de tener el permiso para aplicaciones Java que permita este acceso.

Problemas y soluciones.

· No hay cobertura GPS: hay que esperar a que el dispositivo se encuentre en otro lugar donde la señal sí llegue.

· No se recibe respuesta del dispositivo que ejecuta la aplicación servidor: pueden ocurrir dos cosas, que el dispositivo se encuentre apagado o fuera de cobertura, o que no disponga de saldo para enviar una respuesta (este caso sólo si el móvil es de tarjeta).

· No se puede enviar el mensaje en la aplicación cliente: no se dispone de saldo para el envío de un mensaje o no hay cobertura en ese momento.

· No se puede acceder a la agenda: nuestro móvil no le da el permiso de acceso a las aplicaciones J2ME. Si la agenda está vacía entonces veremos una alerta que lo indica.

· No se puede instalar la aplicación: el archivo .jar estará corrupto, es decir, se habrá grabado mal o el móvil no tiene permiso para la instalación de aplicaciones.

· No sé si la aplicación servidor está ejecutándose (en segundo plano) o no: mantener pulsada la tecla de menú principal y aparece una lista con las aplicaciones en segundo plano.

· La aplicación servidor no se ejecuta automáticamente al encender el dispositivo: hay que especificarlo en los permisos/opciones del móvil respecto a la aplicación y si el móvil no lo permite el permiso, se usa un programa aparte que gestione el encendido automático de aplicaciones al inicio del dispositivo. (En nuestro caso usamos el programa Powerboot).

Glosario de términos utilizados.

GPS – Global Position System

J2ME – Java 2 Micro Edition

CDC – Configuración de Dispositivos con Conexión

MIDP - Mobile Information Device Profile

SMS – Short Message Service

J2SE – Java 2 Standard Edition

WWW – World Wide Web

Applet - es un componente de una aplicación que se ejecuta en el contexto de otro programa

PDA – Personal Digital Assistant

API - Application Programming Interface

JVM – Java Virtual Machine

JSR - Java Specification Request

CLDC - Connected Limited Device Configuration

RISC - Reduced Instruction Set Computer

RMI - Remote Method Invocation

AWT - Abstract Windows Toolkit

GUI - Graphical User Interface

JAR – Java ARchive

HTTP - HyperText Transfer Protocol

IDE - Integrated Development Environment

JDK – Java Development Kit

MID - Mobile Internet Device

KVM - Kilobyte Virtual Machine

GSM - Groupe Special Mobile, sistema global para las comunicaciones

JDBC - Java Database Connectivity

SQL - Structured Query Language

recordStore – almacén de registros

PC – Personal Computer

Referencias

[1] http://es.wikipedia.org/wiki

[2] Jon Byous, Java technology: The early years. Sun Developer Network, sin fecha [ca. 1998]. Recuperado 21 de abril de 2005.

[3] James Gosling, A brief history of the Green project. Java.net, sin fecha [ca. Q1/1998]. Recuperado 22 abril de 2005.

[4] James Gosling, Bill Joy, Guy Steele, y Gilad Bracha, The Java language specification, tercera edición. Addison-Wesley, 2005. ISBN 0-321-24678-0.

[5] Tim Lindholm y Frank Yellin. The Java Virtual Machine specification, segunda edición. Addison-Wesley, 1999. ISBN 0-201-43294-3.

[6] JSR 45 – Especifica cambios al formato de fichero class para soportar depuración a nivel de fuente de lenguajes tales como JSP y SQLJ que son traducidos a Java.

[7] http://www.programacion.com/java/tutorial/ags_j2me/

[8] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=wma

[9]http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=pushRegistry

[10] http://www.java.com/es/about/

[11] http://sourceforge.net/projects

[12] http://java.ciberaula.com/articulo/introduccion_j2me/

[13] http://www.javahispano.org/forum/j2me/es/

[14]http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=j2me

[15] http://www.netbeans.org/index_es.html

[16] http://www.nokia.es/

[17] http://www.telecable.es/personales/agmart1/j2meMiddlet.htm