VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria

Sistema Multimedia para Móviles de Incidencias de Tráfico

Proyecto Fin de Carrera

Rubén Talón Argente

Tutor

Jose Javier Samper Zapater

Valencia, 2008

AGRADECIMIENTOS

Quiero dar las gracias a todas aquellas personas que me han apoyado tanto directamente como indirectamente en la realización de este proyecto que culmina el trabajo de toda una carrera de ingeniería.

Dar las gracias a mi tutor Javier Samper quien me ha aconsejado y ayudado en todo momento sacando el tiempo de “debajo de las piedras”.

Dar las gracias a Leo Van Den Berg quien me propuso el presente proyecto y que me ha ayudado en la elaboración del mismo aportándome ideas y soluciones a mis problemas.

Como no, dar las gracias a toda mi familia especialmente a mis padres que tan mal lo pasaron en mi primer año de carrera y que en estos momentos están orgullosos de mí.

Y gracias a todos mis amigos que me han soportado cada vez que quería hacer las pruebas con sus móviles.

A todos, muchas gracias.

ÍNDICE

1. Introducción

1.1 MOTIVACIONES

Desde hace varios años se ha producido una rápida evolución en lo referente al campo de la Informática y las Telecomunicaciones.
Los ordenadores personales cada vez son más pequeños, potentes y eficaces, las conexiones a la banda ancha son más rápidas, surgen nuevas formas para conectarnos a la red, ya sea desde un portátil con Wireless hasta un móvil.

Hoy en día la forma más directa para conectarnos a la red es desde un teléfono móvil, las estadísticas dicen que 9 de cada 10 españoles poseen uno y que 7 de esas 9 personas han navegado en Internet desde él. Si a esta característica añadimos que también son capaces de reproducir música, capturar fotos, grabar pequeños videos, jugar, etc. nos encontramos con una pequeña computadora que realiza las mismas operaciones que un ordenador personal.

A los dispositivos móviles actuales se les están empezando a incorporar potentes microprocesadores que, en ciertas ocasiones, se asemejan a los utilizados por videoconsolas portátiles tales como la PSP, La Nintendo DS, etc. Gracias a esta característica podemos ejecutar grandes pero pequeñas aplicaciones en nuestros móviles de última generación.

Estas son las razones por las cuales desde hace poco tiempo estamos ante un nuevo mundo de programación para dispositivos móviles. Surgen nuevos métodos de desarrollo, nuevas técnicas, aparecen Sistemas Operativos exclusivos para ciertos móviles (Symbian, Android) y nuevas herramientas de programación.

Muchas empresas ya se están dedicando exclusivamente al desarrollo de aplicaciones para móviles u otros dispositivos portátiles y la sociedad empieza a ver a estos pequeños aparatos como una nueva herramienta de trabajo.

Por otro lado, La Sociedad actual está creciendo a un ritmo vertiginoso y paralelamente al aumento de las nuevas tecnologías que nos soluciona la vida o como mínimo la resuelven de una forma rápida y eficaz.
Uno de los problemas más preocupantes está relacionado con lo que sucede en las carreteras. Por desgracia los accidentes de tráfico están a la orden del día y si añadimos que en la mayoría de ellos hay victimas, ya sean heridos o fallecidos, nos encontramos con un grave conflicto.

Los incidentes que se producen en la carretera soy muy diversos y en ocasiones es de vital importancia comunicarlos con un nivel de detalle elevado para que, si procede, se pueda intervenir rápidamente en el lugar del suceso. Este nivel de información y comunicación se puede conseguir haciendo uso de

herramientas de trabajo actuales como por ejemplos los mencionado dispositivos móviles de última generación.

1.2 DEFINICIONES IMPORTANTES

Ya que la aplicación desarrollada en este proyecto está destinada a la gestión de incidentes de tráfico debemos tener claro qué entendemos por ***Incidente de Tráfico.***
A continuación damos algunas definiciones importantes:

a) Incidente: Es el nombre asignado a la eventualidad según su tipo, que es seleccionado del catálogo con que cuenta el sistema para su registro.

b) Incidente de Tráfico Normal: Son aquellos hechos que están en grado de tentativa o que aunque estén ocurriendo no representan riesgos de mayores daños o lesiones. Generalmente son competencia de autoridades municipales.

c) Incidente de Alto Impacto: Son Incidentes cuya gravedad o nivel de violencia genera inseguridad en la población; para su atención es necesario el involucramiento de varias Corporaciones y la coordinación de estas actividades las llevará a cabo el Jefe Táctico.

d) Corporaciones: Cuerpos de Seguridad e Instituciones de Auxilio: Policía Estatal Preventiva (PEP), Policía Ministerial del Estado (PME), Dirección de Seguridad Pública Municipal (DSPM), Dirección de Tránsito Municipal (DTM), Bomberos (BO), Cruz Roja (CR), Dirección de Protección Civil Estatal (En los Municipios trabaja en coordinación con las Unidades Municipales de Protección Civil; se le denomina PC), eventualmente serán necesarias otras Corporaciones, como Policía Federal Preventiva (PFP), Procuraduría General de la República (PGR) o Secretaría de la Defensa Nacional (SEDENA).

e) Emergencias: Situaciones o eventos que ponen en riesgo la vida, la integridad o el patrimonio de la ciudadanía. Dependiendo de su gravedad, el Incidente puede ser de Alto Impacto o de Tráfico Normal.

Vemos que la definición de Incidente en si depende del sistema donde se ejecute, es decir, no tenemos una definición estándar.
Por lo tanto estamos ante una situación de incertidumbre porque no sabemos realmente si lo que vamos a registrar se trata de un incidente o de un simple problema sin importancia.
Por ejemplo, ¿la congestión en una carretera principal lo podemos relacionar con un incidente de tráfico? ¿Qué los camiones ralenticen el tráfico se considera un incidente que un agente debe registrar?...

En el presente proyecto vamos a considerar que un Incidente de Tráfico es el definido como de Alto Impacto.

1.3 SITUACIÓN ACTUAL DEL TRÁFICO EN ESPAÑA

Como bien hemos nombrado una de las principales causas de mortandad en España son los accidentes de tráfico. En la gráfica que aparece en la **Figura 1.1** podemos ver la evolución de los accidentes mortales en carretera:

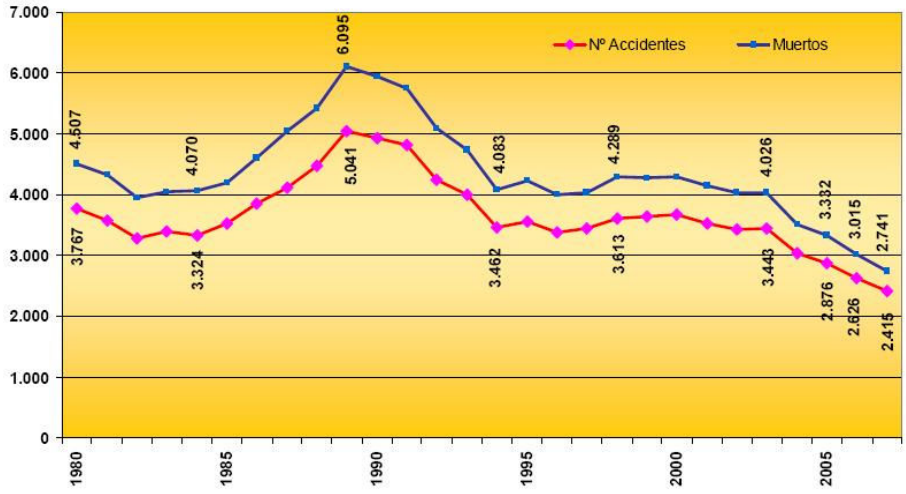


Figura 1.1 Gráfica de número de accidentes y muertos por años en España proporcionada por la Dirección General de Tráfico España [DGT08]

Observamos que a partir de 1990 se está consiguiendo disminuir tanto el número de fallecidos como el de siniestros en carretera y hoy en día va a menos.
Sin embargo el número de victimas sigue siendo muy escandaloso y cada vez más los medios intentan concienciar a la población de los peligros relacionados con el tráfico en carretera.
Gracias a estos medios (campañas publicitarias, prevención de accidentes, etc.) se consiguen salvar una cantidad considerable de vidas.

El descenso más significativo (**Figura 1.2**) se produce desde el 2004 hasta la actualidad donde vemos que año tras año la disminución de victimas es considerable.

Figura 1.2 Índice de vidas salvadas por año en España. Gráfico proporcionado por La Dirección General de Tráfico España [DGT08]

Los accidentes de tráfico por mucho que insistamos en advertir a los conductores suceden, en menor medida, pero suceden y muchos de ellos tienen como resultado muertes innecesarias.
Muchas de las vidas que se salvan son gracias a la rápida intervención de los medios tales como los bomberos, la policía, ambulancias, etc. Si una rápida intervención es capaz de salvar un porcentaje de víctimas de accidentes la meta principal de la consecución de este proyecto es el contribuir a que este objetivo se alcance.

Además, todo esto provoca unos efectos secundarios que también hay que considerar. La mayoría de los incidentes se producen en carreteras y la mayoría de ellas son muy transitadas, esto quiere decir que los incidentes equivalen a grandes atascos, embotellamientos, carriles bloqueados, etc. Todo esto supone una gran cantidad de dinero para el estado ya que tiene que volver el estado del tráfico a su normalidad y esto cuesta dinero, cuanto más tiempo dure el percance más gastos se producen.

1.4 OBJETIVOS

Cuando se produce un incidente de tráfico con un carácter importante como por ejemplo un accidente de coche o moto donde están involucradas personas es muy importante la rápida intervención de los agentes de policía.

El objetivo principal es desarrollar una aplicación donde los policías puedan plasmar con un grado de detalle alto el incidente en un informe multimedia creado en el móvil para seguidamente enviarlo a la central.

Como respuesta a este objetivo, además de actuar con mayor rapidez e informar sin perder detalle, conseguiremos subsanar en mayor proporción los efectos colaterales que surgen cuando ocurre un accidente (atascos, embotellamientos, bloqueos, etc.).

Para que se cumpla este objetivo tenemos que cumplir unos sub-objetivos que son los siguientes:

- Realización de un informe detallado con elementos multimedia (Fotos, Video) para la rápida intervención de otros medios (Bomberos, Ambulancias, Grúas…) que ya estarán informados al detalle de lo ocurrido gracias a dicho informe.

- Crear una aplicación sencilla de manejar para una rápida intervención: La interfaz de usuario tiene que ser sencilla e intuitiva para que el aprendizaje de la aplicación no sea costoso.

- La comunicación con la central tiene que ser inmediata y segura: Debemos tener contacto directo en todo momento con la central ya que será la encargada de transmitir el informe creado a los demás servicios para su rápido conocimiento de la situación.

- La aplicación tiene que amoldarse a los dispositivos móviles destinados: Tenemos que adaptarnos al dispositivo móvil donde acoplemos la aplicación, es decir, ajustar las opciones de la aplicación para que funcione sin problemas en cualquier Terminal.

- Validación y utilización de la aplicación por personas autorizadas: Es necesario controlar en todo momento quien está usando la aplicación.

- Utilización de las últimas tecnologías que disponga el dispositivo móvil: Es decir, la aplicación debe de ser capaz de acceder a las herramientas (Cámara, GPS, Video…) del móvil para su correcta utilización.

2. Estado del Arte

A lo largo de este apartado vamos a analizar y estudiar algunas de las tecnologías, herramientas y aplicaciones disponibles en la actualidad y que forman parte, tanto directa como indirectamente, del desarrollo del proyecto.

El objetivo de realizar dicho estudio es evaluar las distintas alternativas que disponemos, tanto en el aspecto tecnológico como en el de mercado para más adelante justificar las decisiones que tomaremos.

Aplicaciones actuales para móviles destinadas a la gestión directa de incidentes de tráfico son escasas o no cumplen el objetivo principal de este proyecto que, como recordatorio, es el elaborar un informe multimedia detallado para su rápida distribución entre las distintas entidades.
Un proyecto que se aproxima bastante, ya que va incorporado en un dispositivo móvil PDA, es una aplicación desarrollada en el Instituto de Robótica (LISITT-Valencia) concretamente por Pablo Jiménez Martínez y que consiste en la visualización de incidentes respaldado por un mapa cartográfico de su situación en la pantalla del PDA. Dicho proyecto lo describiremos más adelante ya que parece interesante la elaboración de un futuro proyecto que uniera estos dos proyectos.

Como referencia podemos nombrar algunas herramientas utilizadas por la DGT (Dirección General de Tráfico) para la gestión de incidentes de tráfico como por ejemplo la herramienta GIS (Sistema de Información Geográfica).

2.1 APLICACIONES ACTUALES DE GESTIÓN DE INCIDENTES DE TRÁFICO

En la introducción definimos una serie de conceptos relacionados con el tráfico en carretera, definimos lo que entendemos por incidente y llegamos a la conclusión que según el sistema que se adopte en cada sitio esta definición englobará una serie de situaciones u otras.
Las aplicaciones que vamos a nombrar a continuación están destinadas a la gestión de incidentes relacionados con el estado de las carreteras (atascos, temas climatológicos, averías, tráfico lento, etc.)

2.1.1 Sistema de Información Geográfica

2.1.1.1 ¿Qué es GIS?

Sistema de Información Geográfica (**SIG** o **GIS**, en su acrónimo inglés) es una integración organizada de *hardware*, *software*, datos geográficos y personal, diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión. También puede definirse como un modelo de una parte de la realidad

referido a un sistema de coordenadas terrestre y construido para satisfacer unas necesidades concretas de información.

2.1.1.2 Herramientas basadas en GIS

A partir de este sistema podemos desarrollar aplicaciones que tengan como finalidad información de naturaleza geográfica.

2.1.1.2.1 gvGIS

GvSIG es una herramienta orientada al manejo de información geográfica. Se caracteriza por una interfaz amigable y es capaz de acceder a los formatos mas usuales de forma ágil, tanto Raster como Vectorial. Integra en una vista tanto datos locales como remotos a través de un origen WMS, WCS o WFS.
Esta herramienta es utilizada por profesionales o administraciones públicas (ayuntamientos, diputaciones, consejerías o ministerios) de cualquier parte del mundo (actualmente dispone de una interfaz en varios idiomas como por ejemplo castellano, valenciano, inglés, alemán, euskera, gallego, francés, italiano, chino…) y además es gratuita.

Ya que se trata de código abierto es de gran interés para la comunidad internacional de desarrolladores y en concreto para el ámbito universitario por su componente I+D+I.
Además hay que subrayar su carácter extensible, es decir, podemos crear aplicaciones que añadan nuevas funcionalidad al proyecto original utilizando librerías de gvSIG (siempre y cuando se consiga la licencia GPL).

2.1.1.2.2 Herramientas GIS para dispositivos móviles

Basándose en el sistema GIS anteriormente descrito podemos encontrar programas interesantes desarrollados por alumnos universitarios como, por ejemplo, Pablo Jiménez Martínez que presentó como proyecto final de carrera una herramienta GIS para dispositivos móviles.
Concretamente diseñó una aplicación para PDAs donde se mostraban incidencias de tráfico en pantalla y mediante una interfaz sencilla y eficaz podíamos informarnos de todo lo que ocurría en nuestras carreteras al instante.

Las imágenes de la **Figura 2.1** son capturas de pantalla de la herramienta, si se desea más información al respecto sobre este proyecto se puede consultar la memoria del mismo **[JIM06]**

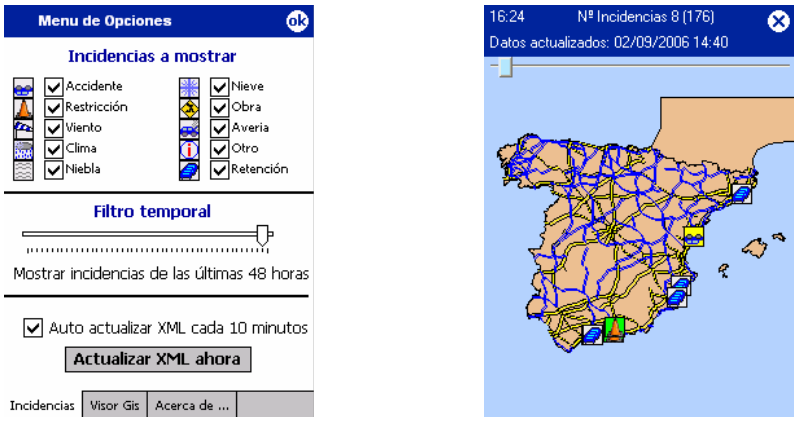


Figura 2.1 Imágenes de la Interfaz – Herramienta GIS para Dispositivos móviles

2.1.2 Servicio Web Multidispositivo

Esta aplicación **[NIÑ06]** desarrollada inicialmente como un prototipo, se ha convertido en la actualidad en una de las aplicaciones más utilizadas en el ámbito trabajo con visualización del estado de carreteras. Destacar que la aplicación se ejecuta tanto en dispositivos móviles del tipo PDA como en teléfonos móviles que tengan acceso a Internet, pues la aplicación está basada en Web.

La información que es capaz de ofrecer la aplicación va más allá de las cadenas de texto, ofrece todo un repertorio de imágenes capturadas por la red de cámaras de tráfico que posee la DGT a lo largo de todas las carreteras españolas. De esta forma, el usuario puede obtener información de, por ejemplo, un atasco en una carretera y, además, obtener en ese momento una imagen de dicho lugar.

Podemos ver en la **Figura 2.2** capturas del Servicio Web Multidispositivo en pleno funcionamiento captando incidencias y mostrando en tiempo real el estado del tráfico.

Figura 2.2 Imágenes de la Interfaz – Servicio Web Multidispositivo

La principal aportación de este servicio consiste en la adaptación de la información (visualización) a las distintas capacidades de los dispositivos.

2.2 TECNOLOGÍA ACTUAL

Existe una gran cantidad de material disponible para desarrollar aplicaciones destinadas a correr dentro de los dispositivos móviles. A continuación listaremos un conjunto de dispositivos que podrían ser candidatos para lanzar nuestra aplicación.

2.2.1 Dispositivos Móviles

Los dispositivos móviles son aparatos pequeños, con algunas capacidades de procesamiento, móviles o no, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras más generales.

Normalmente se asocian al uso individual de una persona, tanto en posesión como en operación, el cual puede adaptarlos a su gusto.

La mayoría de estos aparatos pueden ser transportados en el bolsillo del propietario y otros están integrados dentro de otros mayores, controlando su funcionalidad (como puede ser el ordenador integrado en una lavadora).

2.2.1.1 PDA y BlackBerry

La PDA, del inglés ***P**ersonal **D**igital **A**ssistant*, es un computador de mano originalmente diseñado como agenda electrónica (calendario, lista de contactos, bloc de notas y recordatorios) con un sistema de reconocimiento de escritura. Hoy día se puede usar como una computadora doméstica (ver películas, crear documentos, juegos, correo electrónico, navegar por Internet, reproducir archivos de audio, etc.).

En 1999 una compañía canadiense lanzo el llamo BlackBerry que no es más que una PDA con la principal característica de aprovechar el avance de Internet de la época para centrarse en el correo electrónico. Otra característica visible a diferencia de las PDAs normales es que integra un teclado totalmente completo ya que permite trabajar mucho más rápido.

En la **Figura 2.3** podemos ver algunos modelos de PDAs y BlackBerry, anteriormente los BlackBerrys se caracterizaban por incorporan un teclado completo pero hoy en día algunas PDAs también incluyen esta particularidad por lo que es difícil distinguirlos.

Figura 2.3 Diferentes modelos de PDA y BlackBerry

2.2.1.2 Teléfonos Móviles

Los teléfonos móviles están cambiando los hábitos de vida de las personas y actualmente casi todo el mundo cuenta con uno personal.

Existen gran cantidad de modelos, marcas, accesorios, colores… pero todos ellos tienen unas características comunes: son portátiles, pequeños y son el medio de comunicación más utilizado.

Ya desde su comienzo y hasta ahora han estado evolucionando continuamente, añadiendo a la posibilidad de llamar y enviar mensajes otras características como incluir juegos, email, transferencia de datos (BlueTooth e Infrarrojos), capturar fotos, reproducir video y hasta ver la televisión en directo.

Las compañías que se dedican a la fabricación y distribución de móviles tienen la tarea de diseñar un tipo de móvil que se ajuste a las necesidades de cada persona. Por esta razón el móvil adopta diferentes formas ideal para los gustos de cada propietario.

El mayor fabricante actualmente de móviles del mundo es Nokia (primera imagen de la **Figura 2.4**) seguido por otros también destacables como Motorola, Mitsubishi, Sony Ericsson entre otros. En la **Figura 2.4** podemos ver algunos móviles de distintos fabricantes.

Figura 2.4 De izquierda a derecha: Nokia N95, Sony Ericsson K610i, Nokia N98, Samsung SGH

2.2.1.3 Tablet PC

Un Tablet PC es una computadora a medio camino entre una computadora portátil y un PDA, en el que se puede escribir a través de una pantalla táctil. Un usuario puede utilizar un estilo (o stylus) para trabajar con el ordenador sin necesidad de teclado o mouse. Este aparato fue propugnado por Microsoft y otros fabricantes.

Esta modalidad de computadora portátil ha supuesto un avance significativo en la aplicación de los estudios en lingüística computacional.

Existen modelos que sólo aportan la pantalla táctil a modo de pizarra, siendo así muy ligeros. También hay ordenadores portátiles con teclado y mouse, llamados *convertibles*, que permiten rotar la pantalla y colocarla como si de una pizarra se tratase, para su uso como Tablet PC.

El sistema operativo que utilizan estos dispositivos es una evolución del Windows XP Profesional optimizado para trabajar con procesadores móviles (**Figura 2.5**), que consumen menos energía. El software especial que nos proporciona el sistema operativo nos permite realizar escritura manual, tomar notas a mano alzada y dibujar sobre la pantalla. Así, es útil para hacer trabajos de campo.

Figura 2.5 Fotografías del Tablet PC de Dell y Toshiba utilizando Windows XP

2.2.1.4 Ordenadores Portátiles

En paralelo con la evolución de la telefonía Móvil los portátiles personales han experimentado un gran avance. Cada vez son más pequeños, manejables, rápidos, con una gran capacidad de almacenaje y con la posibilidad, como no, de conectarse a Internet ya sea por cable o por Wireless.
Al igual que con todo, tenemos una gran variedad de tipos de portátiles configurados para desempeñar muchas actividades de ámbito profesional, ocio y personal.

A un ordenador portátil le podemos pedir la misma capacidad de procesado que a uno de sobremesa, es decir, podemos ejecutar las mismas aplicaciones sin que el rendimiento decaiga.

Es por eso que son ideales para ejecutar todo tipo de aplicaciones (los móviles tienen más restringido esta tarea) y tenemos la misma cantidad de herramientas de programación y utilidades para crear nuestras propias aplicaciones. Viendo la **Figura 2.6** observamos que son cada vez más pequeños y manejables con pantallas de alta definición y gran calidad.

Figura 2.6 Portátiles de Sony Vaio, Toshiba y ACER

2.2.1.5 UMPC

Otra variedad de dispositivos móviles son los llamados UMPC o PC Ultra Móvil. Previamente conocido por su nombre código *Project Origami* (Proyecto Origami), es un Tablet PC de factor de forma pequeño. Fue un ejercicio de desarrollo conjunto entre Microsoft, Intel, y Samsung, entre otros. Ofrece el sistema operativo Windows XP Tablet PC Edition 2005 o Windows Vista Home Premium Edition, o Linux y tiene un microprocesador Intel Pentium de voltaje ultra bajo, corriendo en el rango de 1 GHz. La portabilidad de la PC Ultra Móvil puede ser atractivo a los viajeros internacionales de negocios y a los "viajeros con mochila" (backpackers) globales.

Los PC Ultra Móviles tienen un tamaño de pantalla máximo de 20 cm. (alrededor de 7 pulgadas), la pantalla es sensible al tacto (touch screen), tiene una resolución mínima de 800 x 480 pixels, y ejecuta Windows XP Tablet PC Edition. Para hacer el interfaz adecuado para el pequeño formato del ordenador, éste ha sido ligeramente modificado. Además se incluye un paquete de software conocido como el *Touch Pack Interface*, que hace que el interfaz se adecue el uso del lápiz así como la mano.

Debido a que los UMPC ejecutan el sistema operativo Windows XP, serán capaces de trabajar con cualquier software que se haya escrito para esta plataforma, aunque el reducido tamaño de la pantalla hará necesario que se ajusten algunos elementos en el interfaz. El interfaz estándar de Windows XP viene por defecto, aunque existe la opción de tener un interfaz más adecuado para el formato reducido del ordenador utilizando el *Touch Pack Interface*. Debido al formato reducido, no ofrecerá un teclado, pero se proporciona un "teclado virtual", conocido como DialKeys, incluido en el *Touch Pack Interface*. Puesto que el dispositivo tiene conectividad estándar USB 2.0, se le pueden conectar teclados y ratones externos.

Incluirán un microprocesador de 1 GHz como el Intel Celeron M, Pentium M o VIA C7-M, con 256 MB de RAM, e incluirán discos duros de 30 - 60 y 100 GB dependiendo del fabricante y del modelo. Como ejemplo se pueden ver en la **Figura 2.7** dos tipos de UMPC con estas características. También podrían ofrecer GPS, webcam, lector de huellas digitales, sintonizador de TV y lector de tarjetas de memoria. Se soportan también Bluetooth, Wi-Fi y Ethernet.

Los UMPC tienen suficiente capacidad de proceso para soportar audio, video, y juegos, además de tener un buen soporte para navegar por Internet, así como también para otras aplicaciones de comunicación y redes. El Windows Media Player será incluido, con modificaciones menores para proporcionar una experiencia mejor en la pequeña pantalla. Los dispositivos también ofrecerán gráficos de la clase de DirectX 8. También están disponibles todas las aplicaciones estándar de los Tablet PC típicos.

Figura 2.7 Fotografías del UMPC de Gygabyte M704 y el UMPC de HTC

2.2.1.6 Otros

Podemos reservar un pequeño hueco para comentar que existen otros dispositivos móviles destinados exclusivamente al ocio y al entretenimiento, estamos hablando de las videoconsolas portátiles.
¿Porqué hacemos referencia a ellas?, pues porque actualmente podemos encontrar que estás pequeñas consolas tienen un núcleo de procesamiento muy elevado e incluyen elementos multimedia como por ejemplo pantalla táctil, reproducción de video, captura de imágenes y sonido, conexión a Internet, etc. Aunque el desarrollo de juegos y programas en estas maquinas es totalmente casero, no hay licencias ni código abierto exclusivo, cabe decir que se puede sacar mucho partido si se programasen aplicaciones que utilizasen todos sus recursos.
Como ejemplos citar las conocidas PSP (PlayStation Portable) y la Nintendo DS.
A la PSP se le puede acoplar tanto un dispositivo GPS como una cámara de fotos (**Figura 2.8**) y con su procesador de 333 Mhz podemos ejecutar aplicaciones bastante grandes y potentes.
Por otro la lado la Nintendo DS tiene la característica de su doble pantalla y la posibilidad de escribir en un de ellas (pantalla táctil).

En resumidas cuentas, vemos que cualquier dispositivo portátil ya sean móviles o videoconsolas y con una serie de conocimientos podemos utilizarlos como herramientas de trabajo.

Figura 2.8 La consola de Sony PSP con el adaptador GPS y la cámara fotográfica

2.2.2 Sistemas Operativos para Dispositivos Móviles

Cabe destacar los procedentes de Microsoft orientados en su sistema operativo principal, Windows, otros sistemas basados en Linux y el novedoso Android de mano de Apple que vio la luz hace muy poco.

2.2.2.1 Windows Mobile

Windows Mobile **[W3 MS WM]** es un sistema operativo compacto, con una suite de aplicaciones básicas para dispositivos móviles basados en la API Win32 de Microsoft. Los dispositivos que llevan Windows Mobile son Pocket PC's, Smartphones y Media Center portátil. Ha sido diseñado para ser similar a las versiones de escritorio de Windows.

Los inicios de este Sistema Operativo para móviles se remontan al año 2000 donde aún no se llamaba Windows Mobile sino PocketPC 2000.

El PocketPC 2000 apareció en Abril del 2000 y estaba basado en Windows CE 3.0. Poco después fue el debut de Windows Mobile como sistema operativo y como sucesor de los sistemas para Palm-Size PCs. Soportaba entre otras características pantallas de 240x320 QVGA de resolución y disponía de tarjetas de memoria removibles como CompactFlash y MultiMediaCard. Estéticamente el S.O del Pocket PC original era muy similar al Windows 98, ME y 2000. Las aplicaciones incluidas dentro del Pocket PC 2000 fueron herramientas de ofimática, tales como el Pocket Word y Excel, disponía de navegador de internet propio, reproductor de música, infrarrojos y reconocimiento de caracteres. En la **Figura 2.8** podemos observar el tipo de interfaz que se desarrolló para dispositivos móviles bajo Windows Mobile.

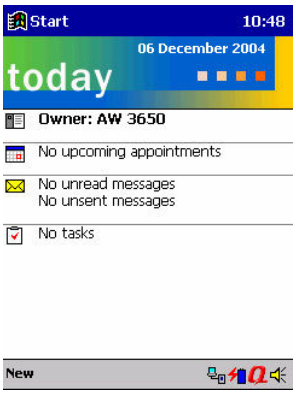


Figura 2.9 Primeras imágenes del Pocket PC 2000

El siguiente que se lanzó sobre el 2001 fue la versión 2002 de Pocket PC apodado como "Merlín". Continuó basándose en Windows CE 3.0 y en una resolución de QVGA se podía implantar en los móviles y por primera vez también en los SmartPhones, la evolución de la interfaz se aprecia en la **Figura 2.9**. Las nuevas características que aparecían en este nuevo modelo se pueden resumir en mensajería instantánea con el Messenger, Terminales de servicios, soporte para Palm OS y mejoras en las comunicaciones con el Pocket Internet Explorer.

Figura 2.10 Pocket PC con Windows CE 3.0

A mediados del 2003 surgió el nombre de Windows Mobile que venía a ser una evolución del Pocket PC actual sólo que este sistema se basaba en el nuevo Windows CE 4.2 (**Figura 2.10**). Se crearon 4 ediciones diferentes: Windows Mobile 2003 for Pocket PC Premium Edition", "Windows Mobile 2003 for Pocket PC Professional Edition", "Windows Mobile 2003 for Smartphone" y "Windows Mobile 2003 for Pocket PC Phone Edition. Se puede resaltar que en estas versiones se añadía la compatibilidad con dispositivos BlueTooth y se incorporó el Media Player 9 con optimización de streaming por internet y soporte de MIDI y mensajes

cortos SMS. Luego le sucedió una segunda versión Windows Mobile SE sobre el 2004 donde se mejoró la resolución a 640x480 y se mejoró la seguridad inalámbrica soportando el cifrado WPA.

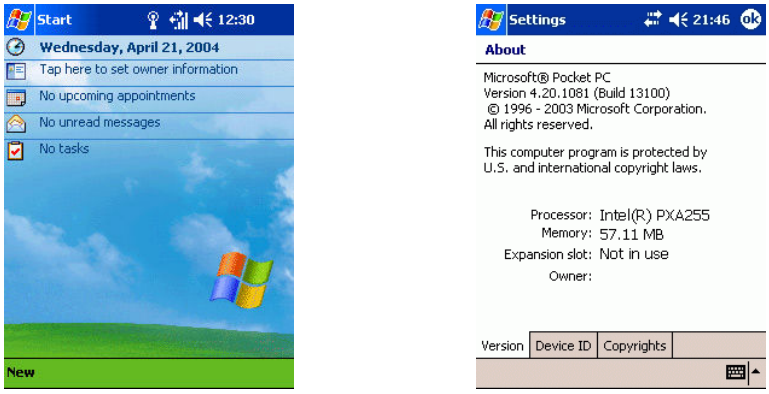


Figura 2.11 Windows Mobile SE 2004

Por el año 2005 apareció otra versión, Windows Mobile 5 (**Figura 2.11**), bajo el nombre en clave de "Magneto" impulsado por Windows CE 5.0 con .NET Compact Framework 1.0 SP2. Las mejoras en esta versión fueron el incremento de la batería, mayor almacenamiento, nueva versión del Media Player, mejora en dispositivos Bluetooth, incorporación del GPS y administración de correo electrónico.

Figura 2.12 Windows Mobile 5

Actualmente se cuenta con la versión WM 6.0 (**Figura 2.12**) lanzada en 2007 con varias ediciones para móviles con pantallas táctiles, dispositivos clásicos y sin capacidad de conexión. Inducido por la versión 5.2

de Windows CE se alcanzan resoluciones de 800x480, VoIP, Ajax, JavaScript, XMLDOM, UMA, SQL, HTML en el Outlook.

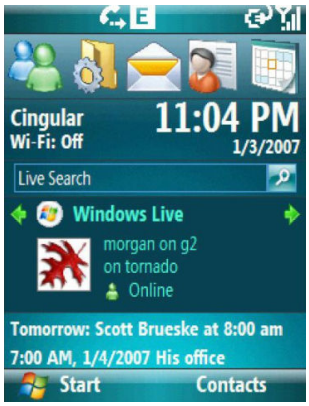


Figura 2.13 Windows Mobile 6

Para este año 2008 Microsoft está desarrollando un nuevo Sistema Operativo más atractivo estéticamente con algunos iconos sacados del propio Windows Vista y con soporte para pantallas de mayor resolución. Bajo el nombre de Windows Mobile 6.1 (**Figura 2.13**) las novedades más significativas son la posibilidad de cortar y pegar y mejoras en la interfaz del Internet Explorer.

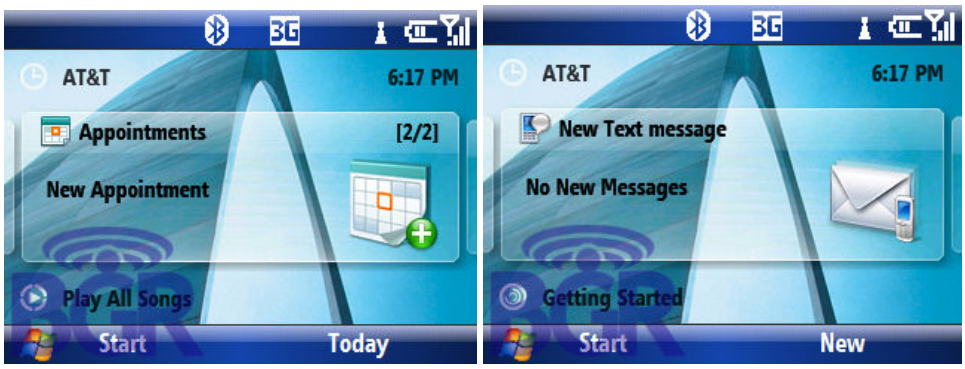


Figura 2.14 Primeras Imágenes del nuevo Windows Mobile 6.1

2.2.2.2 Symbian

Sistema operativo **[W3 SYMB]** privativo que fue producto de la alianza de varias empresas de telefonía celular, entre las que se encuentran Nokia, Sony Ericsson, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric (creador de los teléfonos FOMA junto a Fujitsu, Sharp, etc.), Panasonic, Sharp, etc. En 2003 Motorola vendió el 13% de su participación a Nokia, que se hizo con el 32.2% de la compañía

El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Windows Mobile de Microsoft. La mayoría de los móviles con Symbian son de Nokia: 3650, 3660, 6120, 6260, 6600, 6620,6630, 6680, 6681, 7650, 7610, 6670, 9300, 9500, 5700, N-Gage, E50, E60, E61, E65, E70, N70, N73, N80, N95... Es decir, todos los modelos de la serie 60 y superiores, incluyéndose toda la serie N y los denominados Communicator... Son muy pocos los modelos de móvil de otros fabricantes: Sony Ericsson P800, P802, P900, P910, P990, W950 y M600i; Siemens SX1 ; Motorola A1000, A1010, A920, A925, A728 (sólo disponible en China); todos los celulares FOMA (sólo disponibles en Japón) y Samsung con dos modelos: el SGH-D720 y el SGH-D730 y dos de Panasonic (X700, X800). La lista completa de teléfonos con este sistema operativo se encuentra actualizada en el sitio oficial de Symbian.

Symbian contempla cuatro tipos de dispositivos para su sistema operativo, los denominados Serie60, Serie80, Serie90 y UIQ. La mayoría de Nokia son Serie60, todos los de Sony Ericsson trabajan bajo UIQ, así como también Motorola. En cuanto a su interfaz (**Figura 2.14**) cabe destacar que la estructura de los iconos y de la información está bastante bien adaptada a las necesidades de cualquier usuario, es decir, acceso inmediato a las zonas más utilizadas (Agenda, Contactos, Ofimática, Configuración,…).

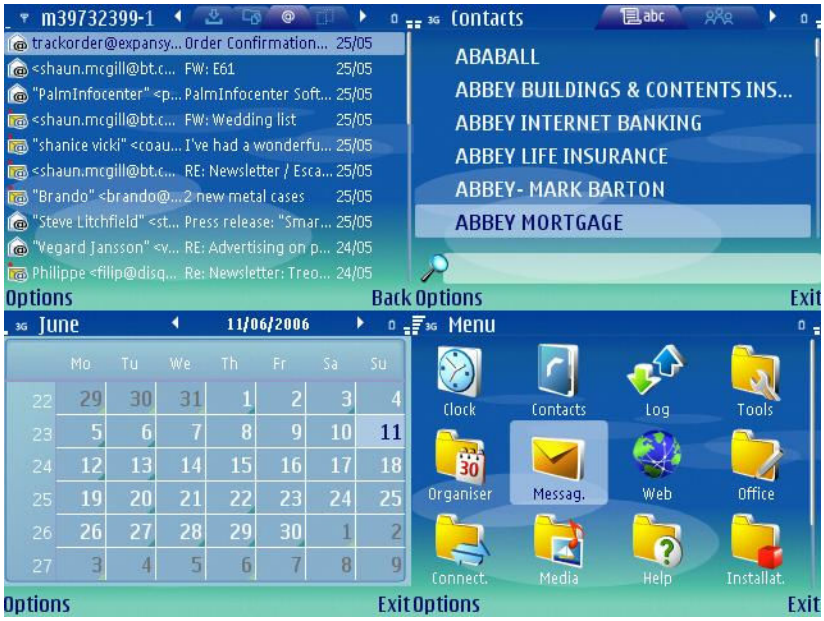


Figura 2.15 Ejemplo de una interfaz Symbian. Imagen sacada de una Web dedicada a Symbian [W3 TSYMB]

2.2.2.3 Android

Android es un stack de software para dispositivos móviles que incluye un sistema operativo, middleware y aplicaciones clave. Google está otorgando un vistazo al SDK de Android **[W3 ANDROID]**, que provee de herramientas y APIs necesarios para comenzar a desarrollar aplicaciones en la plataforma Android, utilizando el lenguaje de programación Java.

Características [W3 WIKI ANDROID]

- Framework de aplicaciones: permite reuso y reemplazo de componentes.
- Máquina virtual Dalvik: optimizada para dispositivos móviles.
- Navegador integrado: basado en el motor open source WebKit.
- Gráficos optimizados, con una librería de gráficos 2D; gráficos 3D basado en la especificación OpenGL ES 1.0 (aceleración de hardware opcional).
- SQLite para almacenamiento de datos estructurados.
- Soporte para medios con formatos comunes de audio, video e imágenes planas (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Telefonía GSM (dependiente del hardware)
- Bluetooth, EDGE, 3G, y WiFi (dependiente del hardware)
- Cámara, GPS, brújula, y acelerómetro (dependiente del hardware)
- Ambiente rico de desarrollo incluyendo un emulador de dispositivo, herramientas para debuguear, perfiles de memoria y performance, y un plugin para el IDE Eclipse.

Arquitectura de Android

Los componentes mayores del sistema operativo de Android, cada sección se describe en detalle:

- Aplicaciones: Las aplicaciones base incluirán un cliente de email, programa de SMS, calendario, mapas, navegador, contactos, y otros. Todas las aplicaciones escritas en el lenguaje de programación Java. El entorno de trabajo a diferencia de las demás se presenta en una pantalla de dimensiones similares a las de una pantalla panorámica (**Figura 2.15**).
- Framework de aplicaciones: Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar el reuso de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- Librerías: Android incluye un set de librerías C/C++ usadas por varios componentes del sistema Android. Estas capacidades se exponen a los desarrolladores a través del framework de aplicaciones de Android. Algunas son: System C library (implementación librería C standard), librerías de medios, librerías de gráficos, 3d, SQLite, entre otras.

- Runtime de Android: Android incluye un ser de librerías base que proveen la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalkiv ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalkiv ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".
- Kernel - Linux: Android depende de un Linux versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, stack de red, y modelo de drivers. El kernel también actúa como una capa de abstracción entre el hardware y el resto del stack de software.

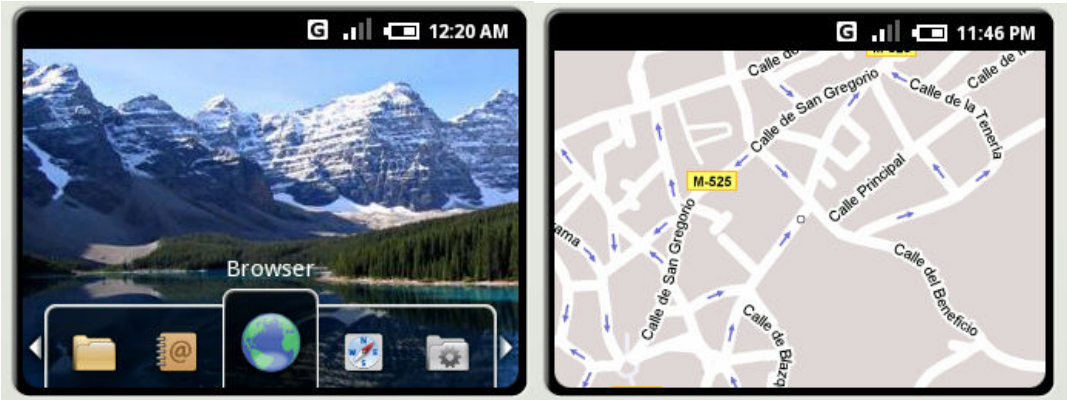


Figura 2.16 Sistema Operativo Android de Google

2.2.2.4 Familiar Linux

Familiar Linux es una distribución de Linux **[W3 FLINUX]** desarrollada con la idea de poder sustituir a otros sistemas tales como Windows CE en iPAQs y en otras PDAs. Este SO tiene como entornos gráficos el Opie y el GPE y vienen con un completo surtido de aplicaciones y de iconos utilizados en otras distribuciones para ordenadores de sobremesa como por ejemplo el icono de ir hacia atrás, cuentas de usuario, configuración, etc. En la **Figura 2.16** se presentan algunas capturas interesantes ya que aun siendo Linux se asemejan a las de cualquier otro sistema.

Esta distribución proporciona al usuario aplicaciones en paquetes iPKG, sistema parecido al de Debian, y un sistema de archivos llamado JFFS2 especializado en memorias Flash, que son las que usan estas PDAs para almacenar información.

Figura 2.17 Algunas capturas de sistemas basados en Familiar Linux

2.2.2.5 Firmware Específicos

No todos los móviles disponen de un Sistema Operativo específico, la mayoría de ellos simplemente incorporan un firmware diseñado concretamente para tareas ya predefinidas, como ejemplo en la **Figura 2.17** vemos distintos firmwares con distinta estética y distintas opciones.

El firmware es un bloque de instrucciones de programa para, como acabamos de decir, propósitos concretos, grabado en una memoria ROM y que establece la lógica de más bajo nivel que controla los circuitos electrónicos de, por ejemplo, un móvil. Al estar integrado en la electrónica del dispositivo es en parte hardware pero también es software, ya que proporciona una lógica y se dispone en algún tipo de lenguaje de programación. Funcionalmente, el firmware es el intermediario (interfaz) entre las órdenes (entradas) que recibe el dispositivo y su electrónica.

Cabe mencionar que desde que los móviles empezaron a tener un nivel de computo aceptable las compañías se decantaron por introducir en sus firmware una maquina virtual de Java para ejecutar los juegos programados en este lenguaje (cuestión de marketing).

Figura 2.18 Algunas interfaces de Firmwares

2.2.3 Lenguajes de Programación para Móviles

2.2.3.1 Java

2.2.3.1.1 Historia

Al principio de los 90, Sun Microsystems creó un nuevo lenguaje de programación llamado Oak como parte de un proyecto de investigación para construir productos electrónicos que dependan principalmente del software. El primer prototipo para Oak fue un controlador portable llamado Star7, un pequeño dispositivo handheld con una pantalla touchscreen LCD que tenía incorporado soporte a redes inalámbricas y comunicaciones infrarrojas. Este dispositivo podría ser usado como control remoto para televisores o VCR y como guía de programas electrónicos, e incluso tenía algunas funciones que ahora son asociadas a los PDAs, como agenda de citas. El software para este tipo de dispositivos necesitaba ser extremadamente confiable y no debía hacer excesivo uso de memoria ni requerir demasiada potencia en el procesador. Oak fue desarrollado como resultado de la experiencia del equipo de desarrollo con el lenguaje C++, el cual, a pesar de tener muchas grandes características, demostró que era un lenguaje complejo y ocasionaba que los programadores comentan fácilmente errores y eso afectaba la confiabilidad del software.

Oak fue diseñado para quitar o reducir la posibilidad de que los programadores comentan errores, ¿cómo? detectando la mayoría de errores en tiempo de compilación y quitando algunas de las características del lenguaje C++ (como punteros y la administración de memoria controlada por el programador) que eran los problemas más comunes.

Desafortunadamente, el mercado para el tipo de dispositivos que el nuevo lenguaje fue creado no se desarrolló tanto como Sun Microsystems esperaba, y al final ningún dispositivo basado en Oak fue vendido a los clientes. Sin embargo, al mismo tiempo, el inicio del conocimiento público de Internet produjo un mercado para el software de navegación para Internet (los navegadores Web). En respuesta a esto, Sun Microsystems renombró el lenguaje de programación Oak a Java y lo usó para desarrollar un navegador multiplataforma llamado HotJava. También le dio la licencia de Java a Netscape, quienes lo incorporaron en su navegador que por ese entonces era el más popular en el mercado, luego fueron incorporados los Java applets.

En un par de años, las capacidades multiplataforma del lenguaje de programación Java y su potencia como plataforma de desarrollo para aplicaciones que podían ser escritas una vez y ejecutadas en diversos sistemas Windows y Unix, había despertado el interés de usuarios finales, porque vieron en ella una manera de reducir los costos del desarrollo de software.

Con el objetivo de conocer las necesidades de los experimentados desarrolladores en Windows y Motif/X-Windows para crear aplicaciones para usuarios finales sofisticados acostumbrados a usar interfaces ricas, Sun Microsystems rápidamente expandió el alcance y tamaño de la plataforma Java. Esta plataforma extendida incluyó un conjunto más complejo de librerías de interfaces de usuario que aquellos que usaran

para construir applets, además con un conjunto de características de computación distribuida y seguridad mejorada.

Con el tiempo Sun Microsystems liberó la primera versión de la plataforma Java 2, había sido necesario dividirla en varias piezas. La funcionalidad principal, estimado como el mínimo soporte requerido para cualquier ambiente Java, estaba empaquetada en el Java 2 Standard Edition (J2SE).

Muchos paquetes opcionales pueden ser agregados al J2SE para satisfacer requerimientos específicos para aplicaciones particulares, como extensiones seguras de sockets que permitan el comercio electrónico. Sun Microsystems también respondió al incremento del interés de usar Java para el desarrollo a un nivel empresarial, y ambientes de servidores de aplicaciones con la plataforma Java 2 Enterprise Edition (J2EE), el cual incorpora nuevas tecnologías como servlets, Enterprise JavaBeans, JavaServer pages, etc.

Como la mayoría de software, los requerimientos de recursos de Java tienen un incremento con cada nueva versión que aparece. A pesar que Java tiene sus raíces en el software para productos electrónicos pequeños, J2SE requiere mucha más memoria y potencia en el procesador para que sea una solución viable en el mercado.

2.2.3.1.2 Evolución

La evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa *Java Specification Requests* (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la *Java Language Specification* (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

- **JDK 1.0** (23 de enero de 1996) — Primer lanzamiento.
- **JDK 1.1** (19 de febrero de 1997) — Principales adiciones incluidas: comunicado de prensa
  - una reestructuración intensiva del modelo de eventos AWT (Abstract Windowing Toolkit)
  - clases internas (inner classes)
  - JavaBeans
  - JDBC (Java Database Connectivity), para la integración de bases de datos
  - RMI (Remote Method Invocation)
- **J2SE 1.2** (8 de diciembre de 1998) — Nombre clave *Playground*. Esta y las siguientes versiones fueron recogidas bajo la denominación **Java 2** y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Otras mejoras añadidas incluían:
  - comunicado de prensa
  - la palabra reservada (keyword) strictfp
  - reflexión en la programación
  - la API gráfica (Swing) fue integrada en las clases básicas

- la máquina virtual (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez
- Java Plug-in
- Java IDL, una implementación de IDL (Interfaz para Descripción de Lenguaje) para la interoperabilidad con CORBA
- Colecciones (Collections)

- **J2SE 1.3** (8 de mayo de 2000) — Nombre clave *Kestrel*.
- Los cambios más notables fueron:
  - la inclusión de la máquina virtual de HotSpot JVM (la JVM de HotSpot fue lanzada inicialmente en abril de 1999, para la JVM de J2SE 1.2)
  - RMI fue cambiado para que se basara en CORBA
  - JavaSound
  - se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de librerías principales (anteriormente disponible como una extensión)
  - Java Platform Debugger Architecture (JPDA)
- **J2SE 1.4** (6 de febrero de 2002) — Nombre Clave *Merlin*. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Los cambios más notables fueron: comunicado de prensalista completa de cambios
  - Palabra reservada assert (Especificado en JSR 41.)
  - Expresiones regulares modeladas al estilo de las expresiones regulares Perl
  - Encadenación de excepciones Permite a una excepción encapsular la excepción de bajo nivel original.
  - non-blocking NIO (New Input/Output) (Especificado en JSR 51.)
  - Logging API (Specified in JSR 47.)
  - API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG
  - Parser XML integrado y procesador XSLT (JAXP) (Especificado en JSR 5 y JSR 63.)
  - Seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS)
  - Java Web Start incluido (El primer lanzamiento ocurrió en Marzo de 2001 para J2SE 1.3) (Especificado en JSR 56.)
- **J2SE 5.0** (30 de septiembre de 2004) — Nombre clave: *Tiger*. (Originalmente numerado 1.5, esta notación aún es usada internamente.[2]) Desarrollado bajo JSR 176, Tiger añadió un número significativo de nuevas características comunicado de prensa
  - Plantillas (genéricos) — provee conversión de tipos (type safety) en tiempo de compilación para colecciones y elimina la necesidad de la mayoría de conversión de tipos (type casting). (Especificado por JSR 14.)
  - Metadatos — también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos. (Especificado por JSR 175.)

- Autoboxing/unboxing — Conversiones automáticas entre tipos primitivos (Como los int) y clases de envoltura primitivas (Como Integer). (Especificado por JSR 201.)
- Enumeraciones — la palabra reservada enum crea una typesafe, lista ordenada de valores (como Dia.LUNES, Dia.MARTES, etc.). Anteriormente, esto solo podía ser llevado a cabo por constantes enteras o clases construidas manualmente (enum pattern). (Especificado por JSR 201.)
- Varargs (número de argumentos variable) — El último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos (e.g. void drawtext(String... lines)). En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al método.

- Bucle for mejorado — La sintaxis para el bucle for se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Iterable, como la clase estándar Collection.
- **Java SE 6** (11 de diciembre de 2006) — Nombre clave *Mustang*. Estuvo en desarrollo bajo la JSR 270. En esta versión, Sun cambió el nombre "J2SE" por **Java SE** y eliminó el ".0" del número de versión. Está disponible en http://java.sun.com/javase/6/. Los cambios más importantes introducidos en esta versión son:
  - Incluye un nuevo marco de trabajo y APIs que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript.
  - Incluye el motor Rhino, de Mozilla, una implementación de Javascript en Java.
  - Incluye un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, como JAX-WS 2.0, JAXB 2.0, STAX y JAXP.
  - Mejoras en la interfaz gráfica y en el rendimiento.
- **Java SE 7** — Nombre clave *Dolphin*. En el año 2006 aún se encontraba en las primeras etapas de planificación. Se espera que su desarrollo dé comienzo en la primavera de 2006, y se estima que su versión RELEASE aparezca para finales del 2008.
  - Soporte para XML dentro del propio lenguaje.
  - Un nuevo concepto de superpaquete.
  - Soporte para closures.
  - Introducción de anotaciones estándar para detectar fallos en el software
- No oficiales:
  - NIO2
  - Java Module System.
  - Java Kernel.
  - Nueva API para el manejo de Días y Fechas, la cual reemplazara las antiguas clases Date y Calendar.
  - Posibilidad de operar con clases BigDecimal usando operandos.

Además de los cambios en el lenguaje, con el paso de los años se han efectuado muchos más cambios dramáticos en la librería de clases de Java (*Java class library*) que ha crecido de unos pocos cientos de

clases en JDK 1.0 hasta más de tres mil en J2SE 5.0. APIs completamente nuevas, como Swing y Java2D, han sido introducidas y muchos de los métodos y clases originales de JDK 1.0 están obsoletos.

En el 2005 se calcula en 4,5 millones el número de desarrolladores y 2.500 millones de dispositivos habilitados con tecnología Java.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

2.2.3.1.3 J2ME

J2ME es la versión de Java orientada a los dispositivos móviles. Originalmente el J2ME surgió porque los móviles tenían una potencia de cálculo muy baja y la interfaz de usuario que presentaban era muy pobre. Por ello se necesitó crear una versión de Java específica para estos dispositivos ya que el resto de versiones (J2SE, J2EE) no encajaban en este esquema (ver **Figura 2.18**).
Actualmente y como ya hemos comentado los dispositivos móviles están obteniendo una potencia muy elevada y son capaces de realizar muchas más tareas que sus predecesores. Ante este nuevo aumento J2ME fue creciendo cada vez más incorporando nuevas APIs que utilizasen las nuevas tecnologías presentes en los móviles (GPS, Cámara de fotos, Sonido…).

Ya hemos visto qué es Java Micro Edition y la hemos enmarcado dentro de la plataforma Java2. En este apartado vamos a ver cuáles son los componentes que forman parte de esta tecnología.

- Por un lado tenemos una serie de máquinas virtuales Java con diferentes requisitos, cada una para diferentes tipos de pequeños dispositivos.

- Configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen 2 configuraciones definidas en J2ME: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.

- Perfiles, que son unas bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

Un entorno de ejecución determinado de J2ME se compone entonces de una selección de:

a) Máquina virtual.
b) Configuración.
c) Perfil.
d) Paquetes Opcionales.

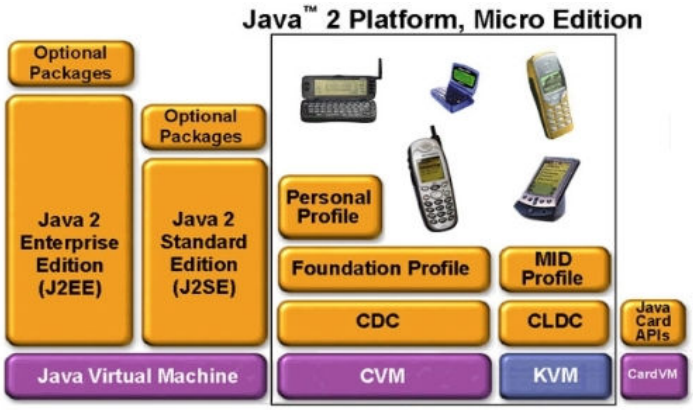


Figura 2.19 Distribución de las Versiones Java (J2EE, J2SE y J2ME) [W3 J2ME REF]

2.2.3.1.4 API

Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción **[W3 SUNAPI]**.

Una API representa un interfaz de comunicación entre componentes software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las APIs asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.

Ejemplos de API

- Microsoft WMI
- Microsoft Win32 API
- Microsoft Framework .NET

- OpenGL
- SUN J2EE APIs
- API for SCSI device interfacing
- The Carbon APIs for the Macintosh OS
- Common Object Request Broker Architecture (CORBA)
- Javascript-C de Mozilla Spidermonkey
- Symfony para PHP

2.2.3.1.5 Recolector de Basura

Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria solicitada dinámicamente de forma manual:

En C++, el desarrollador puede asignar memoria en una zona conocida como *heap* (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada puede llevar a una *fuga de memoria*, ya que el sistema operativo seguirá pensando que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual *cuelgue*. No obstante, se debe señalar que C++ también permite crear objetos en la pila de llamadas de una función o bloque, de forma que se libere la memoria (y se ejecute el destructor del objeto) de forma automática al finalizar la ejecución de la función o bloque.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

2.2.3.1.6 Independencia de la plataforma

Significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Es lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, ‘’’write once, run everywhere’’’.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode) —instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (VM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran librerías adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean “compatibles”. Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la VM es capaz de

analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en cualquier parte” por “Escríbelo una vez, depúralo en todas partes”)

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

2.2.3.2 C++

El comité para el estándar ANSI C fue formado en 1983 con el objetivo de crear un lenguaje uniforme a partir del C original, desarrollado por Kernighan y Ritchie en 1972, en la AT&T. Hasta entonces el estándar lo marcaba el libro escrito en 1978 por B. Kernighan y D. Ritchie. El lenguaje C++ se comenzó a desarrollar en 1980. Su autor fue B. Stroustrup, también de la ATT. Al comienzo era una extensión del lenguaje C que fue denominada ***C with classes***. Este nuevo lenguaje comenzó a ser utilizado fuera de la ATT en 1983. El nombre ***C++*** es también de ese año, y hace referencia al carácter del operador incremento de C (++). Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, la ATT comenzó a estandarizarlo internamente en 1987. En 1989 se formó un comité ANSI (seguido algún tiempo después por un comité ISO) para estandarizarlo a nivel americano e internacional. En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. La evolución de C++ ha continuado con la aparición de ***Java***, un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras, que se utiliza para realizar aplicaciones en Internet. Hay que señalar que el C++ ha influido en algunos puntos muy importantes del ANSI C, como por ejemplo en la forma de declarar las funciones, en los punteros a void, etc. En efecto, aunque el C++ es posterior al C, sus primeras versiones son anteriores al ANSI C, y algunas de las mejoras de éste fueron tomadas del C++.

A pesar de sus años la mayoría de los programas están escritos en este lenguaje por su gran robustez y potencia, estas son las razones por las cuales se utilice para diseñar aplicaciones para dispositivos móviles.

Por el momento, sólo podemos crear programas en C++ para móviles que incorporen Symbian como sistema operativo.

2.2.4 Comunicaciones

En el sector de las comunicaciones nos encontramos con una amplia gama de posibilidades de conectarnos a la red. A continuación describiremos los protocolos más importantes usados por la mayoría de dispositivos móviles.

2.2.4.1 Protocolo de Aplicación HTTP

El protocolo HTTP es conocido porque es el protocolo usado en cada transacción de la WEB (WWW). Sus siglas significan protocolo de transferencia de hipertexto y fue desarrollado por el consorcio W3C y la IETF (***I**nternet **E**ngineering **T**ask **F**orce*, en castellano Grupo de Trabajo en Ingeniería de Internet).

HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

Además es un protocolo sin estado, es decir, no se guarda ninguna información sobre conexiones anteriores.

2.2.4.2 Protocolo de Transporte TCP

Protocolo de Control de Transmisión, uno de los más importantes y fundamentales utilizados en Internet. Muchos programas dentro de una red de datos compuesta por ordenadores pueden usar TCP para crear *conexiones* entre ellos a través de las cuales enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto. TCP da soporte a muchas de las aplicaciones más populares de Internet, incluidas HTTP, SMTP y SSH.

2.2.4.3 Protocolo WAP

Es un protocolo de aplicaciones inalámbricas (Wireless Application Protocol) y es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas como por ejemplo el acceso a Internet mediante un teléfono móvil.

Los móviles son más potentes y livianos cada vez, permitiendo que nuestra comunicación sea cada vez más eficaz. Su gran número y sus capacidades hacen muy interesante para los proveedores de servicios y

contenidos el disponer de un entorno normalizado que permita ofrecer sus servicios a los usuarios de las redes móviles.

WAP define un entorno de aplicación y una pila de protocolos para aplicaciones y servicios accesibles a través de terminales móviles. Consiste en un conjunto de especificaciones, definidas por la Open Mobile Alliance / WAP Forum, que permiten que los desarrolladores diseñen aplicaciones de interconexión para terminales móviles, típicamente teléfonos.

La tecnología WAP permite que los usuarios de estos dispositivos puedan acceder a servicios disponibles en Internet. Sin embargo, existen algunas consideraciones a tener en cuenta al diseñar estos servicios para usuarios móviles, fundamentalmente debidas a las características de los terminales: pantalla significativamente más pequeña que la de un ordenador personal, teclados más limitados que los de un ordenador, limitaciones en la memoria disponible, tanto memoria RAM como memoria para almacenamiento persistente, y limitaciones en la capacidad del procesador, en comparación con la memoria y procesador de un ordenador personal típico. Las redes de telefonía móvil ofrecen también unas prestaciones por lo general menores que los accesos a Internet, si bien con las redes de tercera generación como UMTS las prestaciones mejoran de manera importante.

2.2.4.4 GPRS

General Packet Radio Service o GPRS es una tecnología digital de telefonía móvil.

Es considerada la generación 2.5, entre la segunda generación (GSM) y la tercera (UMTS). Proporciona velocidades de transferencia de datos superiores a las proporcionadas por la tecnología GSM (especialmente útil para conectar a Internet) y se utiliza en las redes GSM.

GPRS es una modificación de la forma de transmitir datos en una red GSM, pasando de la conmutación de circuitos en GSM (donde el circuito está permanentemente reservado mientras dure la comunicación aunque no se envíe información en un momento dado) a la conmutación de paquetes.

Desde el punto de vista del Operador de Telefonía Móvil, es una forma sencilla de migrar la red desde GSM a una red UMTS puesto que las antenas (la parte más cara de una red de Telecomunicaciones móviles) sufren sólo ligeros cambios y los elementos nuevos de red necesarios para GPRS serán compartidos en el futuro con la red UMTS.

GPRS es básicamente una comunicación basada en paquetes de datos. Los *timeslots* (intervalos de tiempo) se asignan en GSM generalmente mediante una conexión conmutada, pero en GPRS los intervalos de tiempo se asignan a la conexión de paquetes, mediante un sistema basado en la demanda. Esto significa que si no se envía ningún dato por el usuario, las frecuencias quedan libres para ser utilizadas por otros usuarios.

Que la conmutación sea por paquetes permite fundamentalmente la compartición de los recursos radio. Un usuario GPRS sólo usará la red cuando envíe o reciba un paquete de información, todo el tiempo que esté inactivo podrá ser utilizado por otros usuarios para enviar y recibir información. Esto permite a los operadores dotar de más de un canal de comunicación sin miedo a saturar la red, de forma que mientras que en GSM sólo se ocupa un canal de recepción de datos del terminal a la red y otro canal de transmisión de datos desde la red al terminal, en GPRS es posible tener terminales que gestionen cuatro canales simultáneos de recepción y dos de transmisión, pasando de velocidades de 9,6 Kbps máximos teóricos en GSM a 110 Kbps máximos teóricos en GPRS aunque en la práctica se queden en 40/60 en recepción y 20 Kbps de transmisión.

Otra ventaja de la conmutación de paquetes es que, al ocuparse los recursos sólo cuando se transmite o recibe información, la tarificación por parte del operador de telefonía móvil sólo se produce por la información transitada, no por el tiempo de conexión. Esto hace posible aplicaciones en la que un dispositivo móvil se conecta a la red y permanece conectado durante un periodo prolongado de tiempo sin que ello afecte en gran medida a la cantidad facturada por el operador.

2.2.4.5 Protocolo Jabber

Protocolo libre para mensajería instantánea, basado en el estándar XML y gestionado por XMPP Standards Foundation.

La red de Jabber está formada por miles de grandes y pequeños servidores en todo el mundo, interconectados por Internet. Habitualmente la red es utilizada por alrededor de un millón de personas.

Es el proyecto más aceptado como la alternativa libre al sistema MSN Messenger de Microsoft, al AOL o al Yahoo Messenger. Aunque es un protocolo bastante minoritario, está creciendo más cada día, gracias a los usuarios y a Google , que ha creado un cliente de mensajería instantánea que utiliza Jabber, Google Talk.

Se puede ver la noticia en las siguientes referencias web:

Comunicado Oficial del equipo de Google Talk:

http://googletalk.blogspot.com/2006/01/xmpp-federation.html

Comunicado Oficial de Google:

http://googleblog.blogspot.com/2006/01/open-federation-for-google-talk.html

Las características más relevantes son:

- **Protocolo abierto:** Con todas las ventajas del software libre, se puede programar un servidor o un cliente o ver el código, entre otras cosas.

- **Descentralizado:** Se puede crear un servidor para Jabber, y se puede interoperar o unirse al resto de la red Jabber.
- **Extensible:** Se puede ampliar con mejoras sobre el protocolo original. Las extensiones comunes son manejadas por la XMPP Standards Foundation.
- **Seguro:** Cualquier servidor Jabber está aislado del exterior. El servidor de referencia permite SSL para comunicaciones cliente-servidor y algunos clientes aceptan GPG como cifrado de las comunicaciones usando cifrado asimétrico. En desarrollo uso de claves de sesión y SASL.
- **Multiredes:** Un *transporte* o pasarela permite comunicarse con otros protocolos usados por clientes como MSN Messenger, ICQ, AOL o Yahoo!.
- **Salas de conversación**: Conocido como Multi-User Chat. Es una de las extensiones que han sido añadidas a la mensajería Jabber, la cual le permite la creación de grupos de debate como en las redes IRC, con la posibilidad de poseer usuarios con distintos privilegios (moderadores, participantes e invitados), iniciar conversaciones privadas y transferir archivos.

A pesar de que se trata de un protocolo que dará mucho de qué hablar presenta algunos problemas relacionados con la pasarela entre protocolos de mensajería instantánea. El principal problema es la estabilidad de los servidores porque nos podemos ver desconectados al tratarse de servidores particulares o de pequeñas organizaciones. Se podría agravar más este inconveniente si empezamos a desplazarnos por multiredes o multiprotocolos a través de pasarelas o transportes, ya que satura excesivamente los servidores.

La mejor solución sería utilizar clientes multiprotocolo tales como Gaim, Kopete, Trillian, etc. y de esta forma utilizar la red Jabber para Jabber y luego abrir otras conexiones para conectarse otros medios como IRC o al Messenger.

Jabber y los protocolos con núcleo XML (núcleo XMPP) y de mensajería instantánea básica con extensiones de presencia (XMPP MI) **Jabber** se han aprobado por los estrictos requisitos para la seguridad e internacionalización de la IETF. De esta forma se garantiza la compatibilidad con el software para Jabber además de aumentar el prestigio del protocolo en sí.

Se sigue avanzando en otras estandarizaciones de diferentes especificaciones para de esta forma convertirse en el estándar oficial para mensajería en Internet.

2.2.4.6 Protocolo XMPP

*E**X**tensible **M**essaging and **P**resence **P**rotocol* (*Protocolo ampliable de mensajería y [comunicación de] presencia*), es un protocolo abierto y ampliable basado en XML, originalmente ideado para mensajería instantánea. Es el protocolo principal en el que está basada la tecnología Jabber.

Con el protocolo XMPP queda establecida una plataforma para el intercambio de datos XML que puede ser usada en aplicaciones de mensajería instantánea. A parte de enviar simples mensajes de texto con este protocolo podemos, en tiempo real, utilizar pizarras colaborativas, middleware ligero, enrutamiento XML genérico y transferencia de ficheros. Las características en cuanto a adaptabilidad y sencillez del XML son heredadas de este modo por el protocolo XMPP.

A diferencia de los protocolos propietarios de intercambio de mensajes como ICQ, Y! Messenger y MSN Messenger, se encuentra documentado y se insta a utilizarlo en cualquier proyecto. Existen servidores y clientes libres para utilizar sin cargo. En la **Figura 2.20** se representa el funcionamiento de XMPP utilizándolo sobre móviles para comunicarse entre ellos.

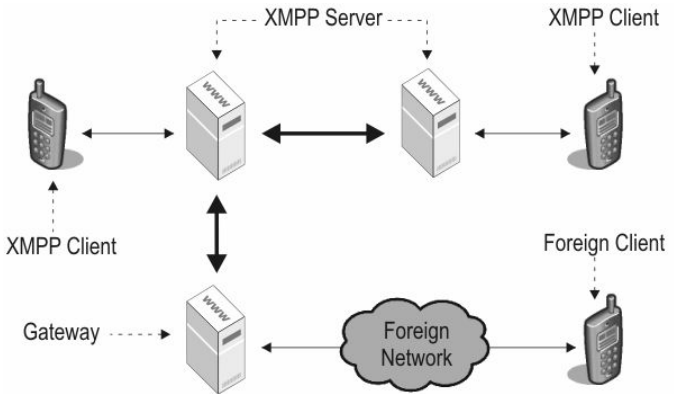


Figura 2.20 Funcionamiento del protocolo XMPP [W3 XMPP]

2.2.4.7 Tecnología 3G

3G (o 3-G) es una abreviatura para **tercera-generación** de telefonía móvil. Los servicios asociados con la tercera generación proporcionan la posibilidad para transferir tanto voz y datos (una llamada telefónica) y datos no-voz (como la descarga de programas, intercambio de email, y mensajería instantánea).

Inicialmente la instalación de redes 3G fue lenta. Esto se debió a que los operadores requieren adquirir una licencia adicional para un espectro de frecuencias diferente al que era utilizado por las tecnologías anteriores 2G. El primer país en implementar una red comercial 3G a gran escala fue Japón. En la actualidad, existen 164 redes comerciales en 73 países usando la tecnología WCDMA.

Ventajas

- IP basado en paquetes, pues solo pagas en función de la descarga lo que supone relativamente un menor costo. Aunque dependiendo del tipo de usuario también se podría calificar como desventaja.
- Más velocidad de acceso.

- UMTS, sumado al soporte de protocolo de Internet (IP), se combinan poderosamente para prestar servicios multimedia y nuevas aplicaciones de banda ancha, tales como servicios de video-telefonía y video-conferencia.

Desventajas

- Cobertura limitada.
- No orientado a conexión. Cada uno de los paquetes pueden seguir rutas distintas entre el origen y el destino, por lo que pueden llegar desordenados o duplicados.

Sin embargo el hecho de no ser orientado a conexión tiene la ventaja de que no se satura la red. Además para elegir la ruta existen algoritmos que "escogen" qué ruta es mejor, estos algoritmos se basan en la calidad del canal, en la velocidad del mismo y, en algunos, oportunidad hasta en 4 factores (todos ellos configurables) para que un paquete "escoja" una ruta.

2.2.5 Seguridad

La seguridad en todo entorno informático es fundamental y hay que tener siempre un buen sistema de protección frente a ataques externos o frente a hackers.
Para ello existen diferentes métodos de seguridad (cifrado, claves, firmas digitales, etc.) todos muy eficaces y fáciles de utilizar.

2.2.5.1 SSL

SSL (Secure Sockets Layer) proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar; la autenticación mutua requiere un despliegue de infraestructura de claves públicas (o PKI) para los clientes. Los protocolos permiten a las aplicaciones cliente-servidor comunicarse de una forma diseñada para prevenir escuchas (eavesdropping), la falsificación de la identidad del remitente (phishing) y mantener la integridad del mensaje.

2.2.5.2 GPG

GPG o GNU Privacy Guard es una herramienta para cifrado y firmas digitales, que viene a ser un reemplazo del PGP (*Pretty Good Privacy*) pero con la principal diferencia que es software libre licenciado bajo la GPL

2.2.5.3 HTTPS

Es la versión segura del protocolo HTTP. El sistema HTTPS utiliza un cifrado basado en las Secure Socket Layers (SSL) para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador

utilizado por el cliente) más apropiado para el tráfico de información sensible que el protocolo HTTP. Cabe mencionar que el uso del protocolo HTTPS no impide que se pueda utilizar http.

Ejemplo de utilización: ***https://correo.uv.es***

2.3 RESUMEN

En este apartado hemos nombrado las diferentes aplicaciones actuales relacionadas con los incidentes de tráfico. Todas estas aplicaciones están relacionadas con la gestión de incidentes de tráfico del tipo normal, es decir, no tienen un riesgo muy elevado.
El presente proyecto intenta llevar la gestión de los incidentes de forma directa, es decir, la gestión se realiza en el lugar del siniestro mediante un dispositivo móvil.

Por otra parte hemos descrito todo el material del que disponemos en estos momentos (tipos de dispositivos móviles, Sistemas Operativos y lenguajes de programación). También hemos comentado los tipos de protocolos que podemos encontrarnos cuando hablamos de conexiones mediante un dispositivo móvil. Entre ellos cabe destacar el protocolo WAP y el GPRS muy utilizados actualmente y que proporcionan velocidades de conexión optimas para realizar operaciones tales como enviar elementos multimedia (fotos, videos, etc).

Relacionado con las conexiones también hemos estudiado herramientas y protocolos de seguridad muy utilizados en otros fines como por ejemplo el cifrado SSL sobre TCP o sobre HTTP conocido como HTTPS.

3. Metodología

A lo largo de los años, conforme se profundizaba en el estudio de la ingeniería del software como parte de la disciplina informática, han surgido diversas metodologías de trabajo para el desarrollo de proyectos de tal naturaleza, incrementándose gradualmente el grado de planificación y control con el que dichos proyectos se gestionan. Desde la casi total ausencia de organización en la realización de los primeros proyectos informática hasta nuestros días, lo cierto es que la evolución de la ingeniería del software se ha producido en paralelo a la del resto de la informática, y como respuesta a la necesidad de abordar la construcción de sistemas cada más y más complejos.

Hoy en día, la práctica totalidad de proyectos informáticos de una cierta envergadura basan su desarrollo y su evolución hacia el producto final en una metodología de trabajo, generalmente probada y validada sobre multitud de desarrollos anteriores. La metodología utilizada en cada caso es dependiente tanto de la propia naturaleza del proyecto como de las preferencias y/o directrices del equipo de profesionales que se responsabiliza de su ejecución.

En cualquier caso, el desarrollo de un sistema siempre involucra, de una forma u otra, un conjunto de etapas bien definidas y establecidas en este marco de conocimiento, como son las fases de especificación de requisitos, análisis, diseño, codificación y verificación. Estas fases deben darse en el orden indicado y siempre con un cierto grado de secuencialidad; sin embargo, en función de las necesidades y del modelo de desarrollo escogido, pueden producirse progresas en paralelo y retroalimentaciones que nos lleven de nuevo a fases anteriores.

Por encima de estas fases de desarrollo se establece una planificación del proyecto, que gestiona en todo momento los recursos (materiales, temporales y humanos) necesarios y disponibles para la realización del mismo, así como su alcance y su coste (que es en función de los recursos realmente utilizados en el proyecto). La importancia relativa de las tareas de planificación es incluso superior a la de las etapas de desarrollo anteriormente citadas, y crece con el tamaño del proyecto en cuestión.

El proyecto concreto que nos ocupa presenta una componente grande de trabajo de investigación, con lo cual debemos manejar a priori un mayor grado de incertidumbre en lo relativo a su planificación, a su desarrollo y a la naturaleza exacta del producto final. Esta incertidumbre afectara sin duda también a las estimaciones realizadas durante el proceso de planificación.

De la introducción teórica obtenemos una primera descomposición del problema, que nos orientara en la elección de una u otra metodología. De este modo, estableceremos una aproximación incremental por prototipos al producto final, a medida que los diferentes subsistemas van siendo desarrollados y se integran en el conjunto de la aplicación. Para la construcción individual de cada subsistema nos bastara con un ciclo de desarrollo secuencial conformado por las correspondientes fases de análisis, diseño, codificación y verificación.

La fase de diseño se realizara con UML, lenguaje de diseño orientado a objetos de gran difusión y más que probada validez, mientras que para la codificación optaremos por el lenguaje Java más concretamente con la versión J2ME.

3.1 PLANIFICACIÓN DE RECURSOS MATERIALES Y HUMANOS

Los recursos asignables se pueden clasificar en forma de pirámide. En la base de la pirámide (**Figura 3.1**) se encuentran las herramientas existentes (hardware y software) para dar soporte al esfuerzo de desarrollo. En el nivel intermedio se encuentran los componentes de software reutilizables y, por último, en el nivel más alto se encuentra el recurso primario que se requiere siempre, las personas.

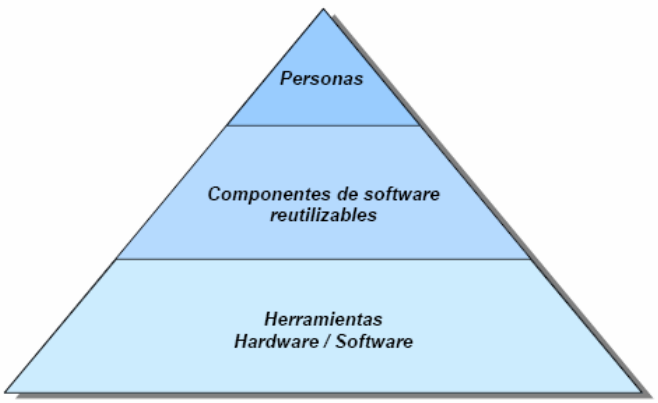


Figura 3.1 Pirámide de la planificación de recursos

Recursos de Entorno: Herramientas Hardware y Software

El entorno es donde se apoya el proyecto de software, llamado a menudo *entorno de ingeniería de software (EIS)*, incorpora hardware y software. El hardware proporciona una plataforma con las herramientas (software) requeridas para producir los productos que son el resultado de una buena práctica de la ingeniería del software.

Recursos Hardware

Los recursos de hardware utilizados en este proyecto son:

- Un ordenador AMD Athlon X2 4200+ con 2 Gigas de RAM y conexión a Internet.
- Un móvil Sony Ericsson K610i para algunas pruebas Java, concretamente para el funcionamiento de la JSR-135 (MMAPI)

- Un móvil Nokia N95 8Gb para la ejecución del proyecto.

Recursos Software

Todas las licencias del software utilizadas son propiedad de la Universidad de Valencia.

- Sistema Operativo Windows XP Professional
- Netbeans 6.0 entorno de programación gratuito que incorpora la posibilidad de desarrollar proyectos de distintos lenguajes, entre ellos Java. Además incorpora un modulo exclusivo para el diseño y la programación de aplicaciones para móviles.

NOTA: Al inicio del proyecto se empezó utilizando Netbeans 5.5 que incorpora prácticamente todo lo necesario para hacer el proyecto, el pasar a la versión 6.0 se debe a que ésta tiene la posibilidad de editar gráficos SVG.

- Sun Wireless Toolkit 2.5.1 para la emulación de móviles y prueba del proyecto.
- Microsoft Office 2003 para la elaboración de la documentación.
- Visual Paradigm for UML 6.1 Standard Edition: programa de diseño UML utilizado para crear los diagramas que representan el proyecto (casos de uso, secuencia, actividad…).
- Ikivo Animador: Esta aplicación sirve para animar gráficos SVG y darle un toque de interactividad a la interfaz del proyecto.
- Illustrator CS3: El Illustrator viene incluido en el Adobe Suite CS3 que es una recopilación de programas de diseño y retoque gráfico. Utilizaremos esta herramienta para crear y/o modificar gráficos SVG.
- XML NotePad: Para modificar cualquier fichero XML haremos uso de esta herramienta.

Componentes de software reutilizables

En el presente proyecto se observa que existen 2 partes claramente diferenciables que pueden coexistir por separado: la interfaz y la parte de comunicaciones.
Para la interfaz se ha reutilizado un ejemplo disponible en la herramienta de Sun Wireless Toolkit que simplemente era una demostración de uso de gráficos SVG para el móvil. Además se han utilizado iconos predefinidos gratuitos de Internet.
Para la parte de comunicaciones se ha utilizado el protocolo Jabber/XMPP que es de libre distribución y de código abierto. Se ha reutilizado un ejemplo de comunicación entre móvil y servidor y lo hemos adaptado para nuestras necesidades.

Personas

Para llevar a cabo el proyecto se necesita personal con conocimientos de programación en Java, concretamente J2ME para móviles, ingeniería del Software, XML para la parte de comunicaciones y los gráficos SVG, experiencia en mensajería instantánea con Jabber y su protocolo XMPP y conocimiento de estructura de incidencias de tráfico.

3.2 PLANIFICACIÓN TEMPORAL

La realización del proyecto se dividirá en seis etapas claramente diferenciadas:

a) ***Revisión bibliográfica:*** En esta etapa se procederá a la recopilación y revisión de documentos correspondientes a trabajos anteriores relativos al tema que nos ocupa. De esta etapa dependerá la visión que nos formemos del estado actual del arte, y la dirección hacia la que encaminaremos los posteriores esfuerzos de investigación. Las fuentes principales de bibliografía serán Internet y, en menor medida, el tutor del proyecto.

b) ***Análisis:*** Comprenderá las tareas propias de análisis del problema abordado y la especificación de los requisitos que deberá satisfacer la aplicación de prueba. Se realizara un análisis preliminar del problema en conjunto, para pasar posteriormente a analizar cada subsistema de manera detallada.

c) ***Diseño:*** Aquí incluiremos el diseño de la aplicación representado en diagramas UML (diagramas de casos de uso, de clases, secuencia…) y las pertinentes pruebas de código.

d) ***Codificación:*** Se efectuará dentro de esta etapa la implementación del sistema, en la forma de una aplicación codificada con el lenguaje Java orientado a dispositivos móviles (J2ME). Dentro de esta etapa se contempla la realización de pruebas de verificación del código, de acuerdo con los procedimientos normales de programación.

e) ***Evaluación:*** La etapa de evaluación comprende el diseño, la implementación y la ejecución de un conjunto de pruebas con objeto de evaluar el rendimiento y el funcionamiento correcto del sistema. Se realizarán un conjunto de pruebas que aborden todas las características del sistema comprobando que todo funciona correctamente.

f) ***Redacción:*** Se trata del proceso de redacción de la presente memoria del proyecto, que se realizará de manera sostenida durante toda la duración del proyecto, con objeto de incluir toda la información de interés y actualizar convenientemente cualquier dato que lo necesite.

3.2.1 División y Duración del Trabajo

La división de tareas debe ser lo suficientemente sencilla como para simplificar al máximo la gestión. Sin embargo, debe ser lo suficientemente elaborada y completa como para describir todas y cada una de las tareas del proyecto, segregando en paquetes distintas actividades conceptualmente diferentes. Si la descomposición es excesivamente detallada, la fragmentación resultante desperdicia recursos utilizándolos de forma poco eficiente. Así mismo deben ser lo suficientemente completas como para permitir que dicha gestión se pueda realizar de manera autónoma. Si la descomposición no es lo suficientemente detallada se diluyen los flujos de trabajo y se corre el riesgo de no dimensionar correctamente el esfuerzo o de no asignar claramente las responsabilidades. Como todo proyecto de software (o casi todo, exceptuando la programación extrema) se deben realizar fases de análisis y de diseño. Estas fases nos ayudarán posteriormente a realizar una mejor implementación del sistema, libre de errores (al menos a priori).

Una vez definidas y descritas las actividades, es preciso analizar cuánto va a durar la ejecución de cada una de ellas, y sobre todo, en qué orden se van a abordar.
Para este proyecto asumiremos que se trabaja 4 horas al día de lunes a viernes sin contar los días festivos.

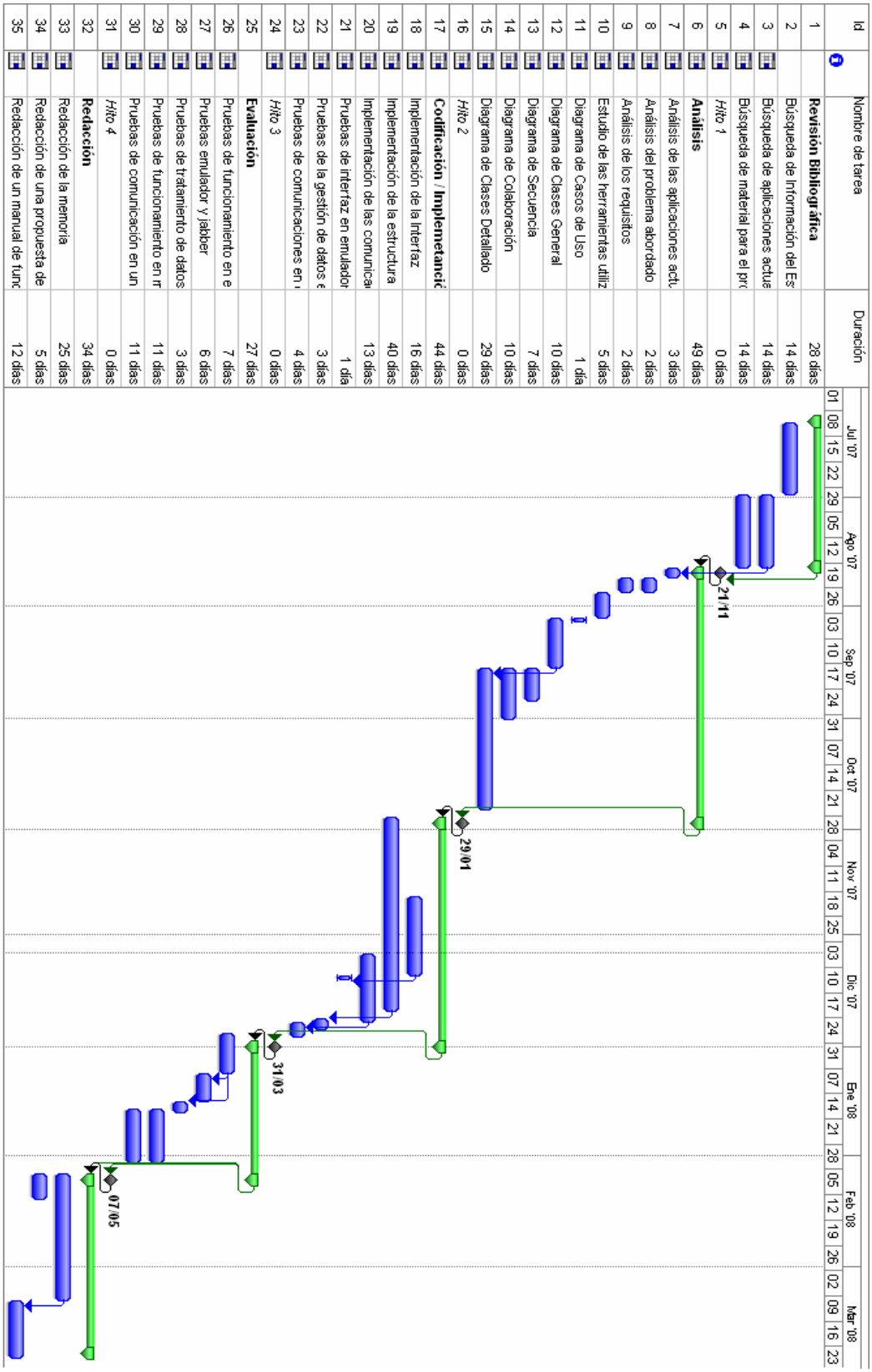
En la tabla 3.1 se muestra de forma detallada todas las partes que intervienen en el desarrollo del proyecto así como el tiempo que se dedicará a cada uno de ellas.

		Días
Revisión Bibliográfica	*Total*	**28**
Búsqueda de información del estado del arte	10	
Búsqueda de aplicaciones actuales	4	
Búsqueda de material para el proyecto	4	
		Días
Análisis	*Total*	**49**
Análisis de las aplicaciones actuales	3	
Análisis del problema abordado	2	
Análisis de los requisitos	2	
Estudio de las herramientas utilizadas en el proyecto	5	
Análisis y diseño de diagramas	*Total*	**37**
Diagrama de Casos de Uso	1	
Diagrama de clases general	5	
Diagrama de Secuencia	7	
Diagrama de Colaboración	10	
Diagrama de clases final detallado	14	
		Días
Codificación / Implementación	*Total*	**44**
Implementación de la interfaz	14	
Implementación de la estructura y gestión de datos	11	
Implementación de las comunicaciones	12	
Pruebas de interfaz en emuladores	1	
Pruebas de la gestión de datos en emuladores	3	
Pruebas de comunicaciones en emuladores	3	
		Días
Evaluación	*Total*	**27**
Pruebas de funcionamiento en el emulador	7	
Pruebas de comunicaciones entre emulador y Jabber	6	
Pruebas de tratamientos de datos en el emulador	3	
Pruebas de funcionamiento en un móvil real	3	
Pruebas de comunicación en un móvil real	8	
		Días
Redacción	*Total*	**34**
Redacción de la memoria	25	
Redacción de una propuesta de protocolo de utilización	5	
Redacción de un manual de funcionamiento	4	
Total tiempo empleado	***Días***	**182**
	Meses*	**8**

Tabla 3.1 División del proyectos en bloques con una estimación de la duración

(*) No se han tenido en cuenta los días festivos

3.2.2 Diagrama de Gantt



3.3 PLANIFICACIÓN DE COSTES

3.3.1 El Coste del Software

Al principio, el coste del software constituía un pequeño porcentaje del coste total de los sistemas informáticos basados en computadora. Un error considerable en las estimaciones del coste del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos. Un gran error en la estimación del coste puede ser lo que marque la diferencia entre beneficios y pérdidas. Sobrepasarse en el coste puede ser desastroso para el equipo de desarrollo.

Los costes se pueden dividir en dos tipos: costes directos y costes indirectos. Los costes directos son aquellos que se pueden imputarse directamente al proyecto como consecuencia del mismo, estos son: costes de personal, material y equipo, viajes y estancias y gastos adicionales. Los costes indirectos son aquellos que la empresa debe ingresar a través de los proyectos que realice, son también llamados costes generales, como por ejemplo: alquiler del inmueble, costes de suministros (luz, agua, etc.), costes de limpieza, secretariado, etc.

En el estudio del coste que vamos a realizar no incluiremos estos costes indirectos ya que son irrelevantes.

3.3.2 Estimación del coste del personal

Para la estimación de costes se utilizó el modelo COCOMO 2 **[W3 MCOCOMO]** que es un modelo de tres niveles que permite estimaciones cada vez más detalladas y que pueden realizarse a la vez que progresa el desarrollo del proyecto.

MODELO DE COMPOSICIÓN DE APLICACIONES (ACM)

Las primeras fases o ciclos en espiral utilizados en proyectos software de Generador de aplicaciones, Integración de Sistema e Infraestructura implicarán generalmente prototipado y utilizarán las utilidades del Módulo de Composición de Aplicaciones. El Modulo de Composición de Aplicaciones COCOMO II, soporta estas fases y cualquier otra actividad de prototipado que se realice más adelante en el ciclo de vida.

Este modelo se dirige a aplicaciones que están demasiado diversificadas para crearse rápidamente en una herramienta de dominio específico, (como una hoja de cálculo) y que todavía no se conocen suficientemente como para ser compuestas a partir de componentes ínter operable. Ejemplos de estos sistemas basados en componentes son los creadores de interfaces gráficas para usuario, bases de datos ó gestores de objetos, middleware para proceso distribuido ó transaccional, manejadores hipermedia, buscadores de datos pequeños y componentes de dominio específico tales como paquetes de control de procesos financieros, médicos ó industriales.

Dado que el Modelo de Composición de Aplicaciones incluye esfuerzos de prototipado para resolver asuntos potenciales de alto riesgo tales como interfaces de usuario, interacción software/sistema, ejecución ó grado de madurez tecnológica, los costes de este tipo de esfuerzo se estiman mejor mediante dicho modelo.

MODELO DE DISEÑO INICIAL (EDM)

Hemos visto que las primeras fases o ciclos en espiral utilizados en proyectos software de Generador de Aplicaciones, Integración de Sistema e Infraestructura se ajustaban mejor al modelo de Composición de Aplicaciones. Las siguientes fases ó ciclos espirales normalmente incluirán la exploración de arquitecturas alternativas ó estratégicas de desarrollo incremental. Para sostener estas actividades COCOMO II proporciona un modelo de estimación anticipado: el Modelo de Diseño Anticipado. El nivel de detalle de este modelo puede ser consistente con el nivel general de información disponible y con el nivel general de aproximación de la estimación requerida en esta etapa. El Diseño Anticipado incluye la exploración de arquitecturas de software/sistema alternativas y conceptos de operación. En esta fase no se sabe lo suficiente como para dar soporte a la estimación de grano fino. La correspondiente capacidad de COCOMO II incluye el uso de Puntos de Función y un conjunto de siete drivers de coste de grano grueso (por ejemplo, dos drivers de coste para capacidad del personal y experiencia del personal en lugar de los seis drivers de coste del Modelo Post-Arquitectura que cubren varios aspectos de capacidad del personal, continuidad y experiencia). El modelo de Diseño Anticipado usa Puntos de Función No Ajustados como métrica de medida. Este modelo se utiliza en las primeras etapas de un proyecto software, cuando se conoce muy poco sobre el tamaño del producto que se va a desarrollar, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto ó especificaciones detalladas del proceso que se va a usar. Este modelo puede aplicarse a cada uno de los sectores de desarrollo de Generador de Aplicaciones, Integración de sistemas ó Infraestructura. (Ver apartado 5.2.1, Situación del software en el mercado futuro).

MODELO DE POST-ARQUITECTURA (PAM)

Hemos visto que las primeras fases o ciclos en espiral usados en proyectos software de Generador de Aplicaciones, Integración de Sistema e Infraestructura se ajustaban mejor al modelo de Composición de Aplicaciones y que las siguientes fases ó ciclos espirales normalmente serán sostenidas por el modelo de Diseño Anticipado. Una vez que el proyecto está listo para desarrollar y sostener un sistema especializado, debe haber una arquitectura de ciclo de vida que proporcione información más precisa de los drivers de coste de entradas y permita cálculos de coste más exactos. Para apoyar esta etapa COCOMO II proporciona el Modelo Post-Arquitectura.

El Modelo Post-Arquitectura incluye el actual desarrollo y mantenimiento de un producto software. Esta fase avanza rentablemente si se desarrolla una arquitectura de ciclo de vida software válida con respecto a la misión del sistema, al concepto de operación y al riesgo, y establecido como marca de trabajo del producto. El modelo correspondiente de COCOMO II tiene aproximadamente la misma granularidad que los anteriores

modelos COCOMO y AdaCOCOMO. Utiliza instrucciones fuente y/ó Puntos de Función para medir, con modificadores para reutilización y objetos software; un conjunto de 17 drivers de coste multiplicativos; y un conjunto de 5 factores que determinan el exponente de escala del proyecto. Estos factores sustituyen los modos de desarrollo (Orgánico, Semilibre y Rígido) del modelo original COCOMO y refina los 4 factores de exponente-escala en Ada COCOMO.

Una vez explicados los diferentes modelos de estimación para el presente proyecto utilizaremos el modelo de diseño inicial, calcularemos lo necesario para estimar el coste que tendría nuestro trabajo y la duración del mismo.

El modelo de COCOMO 2 nos aporta las siguientes fórmulas para calcular el esfuerzo personas/mes:

ESFUERZO NOMINAL
$PM_{nominal} = A \times (Tamaño)^{B}$

ESFUERZO AJUSTADO
$PM_{ajustado} = PM_{nominal} * M$

Tabla 3.2 Fórmulas del COCOMO 2000 para el Esfuerzo

3.3.2.1 Obtención de los puntos de función desajustados

Calculamos las funcionalidades de nuestra aplicación. Para esto se evalúa la complejidad de los componentes individualmente, mediante unas tablas preestablecidas que proporcionan el factor de complejidad (sencillo, medio o complejo). Los componentes son los siguientes:

- **Entradas de usuario**. Son aquellos grupos de datos que entran en nuestra aplicación y añaden o cambian información de la misma, como formularios, diálogos.

- **Salidas de usuario**. Son aquellos grupos lógicos que salen de la aplicación, como las pantallas, los informes, etc.

- **Grupos lógicos de datos internos**. Son aquellos grupos lógicos que se generan y que mantiene la aplicación, como un fichero o una tabla de una

- **Grupos lógicos de datos interfaz**. Son grupos lógicos compartidos con otra aplicación, recibidos o enviados a ella.

- **Consultas externas**. Son entradas de usuario que generan salidas inmediatas.

	Cantidad	Complejidad	Valor/Elemento	Total
Entradas de Usuario	14	Baja	1	**14**
	2	Alta	10	**20**
Salidas de Usuario	1	Baja	1	**1**
	1	Media	2	**2**
Datos Internos	2	Media	5	**10**
Datos Interfaz	1	Media	5	**5**
Salida Inmediata	2	Baja	2	**4**

Ptos de Función Desajustados
56

Paso a líneas de código				
56 PF No Ajustado	**X**	**23**	**JAVA**	**1288**

Tabla 3.3 Puntos de Función No Ajustados Obtenidos y líneas de código estimadas

3.3.2.2 Obtención del Exponente B

El término B refleja el esfuerzo creciente al incrementarse el tamaño del proyecto. Depende de 5 factores de escala que a continuación estimaremos.

Factor de Escala	Descripción	Escala	Valor
PREC	Experiencia en este tipo de proyectos	normal	**3,72**
FLEX	Grado de flexibilidad	alta	**2,03**
RESL	Amplitud del análisis del riesgo	baja	**5,65**
TEAM	Cohesión del equipo	extra alta	**0**
PMAT	Madurez del proceso	muy baja	**7,8**

Factores de Escala	
Suma	**19,20**
B	**1,10**

Tabla 3.4 Obtención del Factor B

3.3.2.3 Obtención de Factores de Ajuste

Factor de Ajuste	Descripción	Escala	Valor
RCPX	Fiabilidad y complejidad del producto	alta	**1,33**
RUSE	Requerimientos de reusabilidad	muy alta	**1,91**
PDIF	Dificultad de la plataforma	normal	**1**
PERS	Capacidad del personal	alta	**0,83**
PREX	Experiencia del personal	alta	**0,83**
FCIL	Facilidades para el desarrollo	extra alta	**0,62**
SCED	Esfuerzo de calendario	muy baja	**1,3**

Factores de Ajuste	
M	**1,41**

Tabla 3.5 Obtención del Factor de Ajuste

3.3.2.4 Cálculo del Esfuerzo

ESFUERZO NOMINAL
$PM_{nominal} = A \times (Tamaño)^{B}$

ESFUERZO NOMINAL		
A	**CTE calibración**	**2,94**
Tamaño	**KIFE**	**1,288**
B	**Factores Escala**	**1,10**
$PM_{nominal}$	**Esfuerzo**	**3,89**

ESFUERZO AJUSTADO
$PM_{ajustado} = PM_{nominal} * M$

ESFUERZO AJUSTADO		
$PM_{nominal}$	**Esfuerzo Nom**	**3,89**
M	**Factores Ajuste**	**1,41**
$PM_{ajustado}$	**Factores Escala**	**5,48**

Tabla 3.6 Cálculo del Esfuerzo

3.3.2.5 Cálculo del Tiempo Estimado

TIEMPO ESTIMADO
$TDEV = 3{,}67 \times PM_{ajustado}^{0{,}28+0{,}2x(B-1{,}01)}$

Tiempo estimado		
TDEV	6	meses

Tabla 3.7 Cálculo del tiempo estimado

3.3.2.6 Cálculo del Coste del Personal

COSTE DEL PERSONAL
$Coste = sueldo/mes \times PM_{ajustado} \times TDEV$

Coste		
Sueldo al mes, 4 horas	1000	€
Total	*33419*	*€*

Tabla 3.8 Coste del Personal

Tal y como habíamos previsto la duración estimada de realización del proyecto es de unos 6 meses trabajando 4 horas diarias 5 días a la semana.

El sueldo especificado corresponde al salario medio que puede cobrar un ingeniero informático trabajando a media jornada.

Al coste total personal calculado habría que sumarle el coste de los recursos materiales empleados descritos a continuación.

3.3.3 Coste de Recursos Materiales

Se estimarán los costes de utilización de los recursos empleados en relación a su precio de adquisición, la amortización anual a la que se someten y del tiempo de utilización de los mismos.

Material	Coste	Amortización Anual 20%	Utilización / meses	Coste Utilización
Hardware				
PC AMD X2 4200+, 2 GB RAM	545 €	109 €	6	54,50 €
Móvil				
Nokia N95 8Gb LIBRE	600 €	120 €	6	60,00 €
Licencias				
NetBeans 6.0	0 €			0,00 €
Sun Wireless Toolkit 2.5.2	0 €			0,00 €
Illustrator CS3	450 €			450,00 €
Ikivo Animator 90 días	0			0,00 €
XML NotePad	0 €			0,00 €
Microsoft Office 2003	300 €			300,00 €
Windows XP Professional	150 €			150,00 €
Otros				
Consumibles (Tinta, CDs, DVDs…)	50 €			50,00 €
			Total	1.364,50 €

Tabla 3.9 Coste de Recursos Materiales

4. Análisis y Diseño del Sistema

En todo proyecto de Ingeniería del Software se requiere una fase para el análisis y diseño del sistema que queremos desarrollar de forma que se consiga una visión general del mismo.
Mediante un análisis exhaustivo del problema a abordar extraeremos los requisitos que debemos que satisfacer para solucionar dicho problema.

4.1 ANÁLISIS DEL SISTEMA

El problema que queremos solucionar es la situación que se presenta cuando los agentes de policía gestionan una incidencia de tráfico en el mismo lugar del suceso.
Cuando sucede un incidente se requiere la rápida intervención de las fuerzas de seguridad vial para, entre otros motivos, salvar vidas. Esta vertiginosa intervención es realiza con mucha precisión, informando de los hechos a otros servicios (bomberos, ambulancias, etc.), controlando que la situación no vaya a peor, investigando las causas del siniestro, etc. Pero desgraciadamente a veces la información con la que tratan tiene un grado de detalle bajo y en ocasiones no es suficiente para mostrar la magnitud de lo sucedido.
Con este proyecto se pretende mejorar este nivel de detalle para que la intervención de las UVI móviles, grúas, bomberos, etc. sea más rápida, eficaz y controlada.

4.1.1 Análisis de Requisitos

Los requisitos que tiene que cumplir nuestro proyecto son principalmente:

- Sistema de información detallado, rápido, sencillo de utilizar y fiable

Basándonos en estas 4 características diseñaremos un programa para tecnología móvil que cumpla las siguientes características:

INTERFAZ

- **Sencilla:** La interfaz del usuario tiene que ser sencilla e intuitiva de manejar por personal que carezca de nociones de informática o de manipulación de aparatos electrónicos/digitales.
- **Adaptable**: A cualquier dispositivo móvil, es decir, que se pueda acoplar a cualquier tamaño de pantalla.
- **Vistosa:** El móvil se convierte en una herramienta pequeña pero potente, tenemos que tener en cuenta las reducidas dimensiones para elaborar un entorno que no cueste de ver.
- **Rápida:** La interacción con la interfaz tiene que ser rápida, es decir, las transiciones entre pantallas tiene que resultar veloz y utilizar cuantas menos mejor.
- **Flexible:** Que se pueda cambiar sin ningún problema para añadir nuevas funcionalidades o nuevos menús.

COMUNICACIONES

- **Rápidas:** En una situación crítica el envío urgente de información es de vital importancia. El sistema tiene que utilizar las últimas novedades en transmisión de datos vía móvil.
- **Flexibilidad:** El cambio del dispositivo móvil o de la tecnología usada para transmitir datos no debe afectar a la información que queremos enviar.
- **Mensajes Entendibles:** Cuando el informe enviado llegue a su origen dicho mensaje tiene que ser descifrado en un corto periodo de tiempo.

INFORMES

- **Detallados:** Los reportes del incidente tienen que tener hasta el más mínimo detalle.

SEGURIDAD

- **Cifrado:** Los informes tienen que tener algún tipo de cifrado mientras navegan por la red.
- **Autenticación:** Los usuarios tienen que identificarse a la hora de utilizar el sistema.
- **Histórico:** El sistema debe de guardar un histórico de todo lo que realice.
- **Backup:** Posibilidad de realizar copias en archivos internos de los incidentes creados.

4.2 DISEÑO DEL SISTEMA

4.2.1 Casos de Uso

Figura 4.1 Diagrama de Casos de Uso

Hemos hecho una distinción del tipo de acciones que puede realizar el usuario. Cada tonalidad de color agrupa un determinado concepto entre *Crear Incidentes, Gestionar Usuarios, Configuración y Multimedia.* Además, los grupos hacen tener una idea de la estructura que tendrá la interfaz pues ésta también estará dividida en secciones que corresponden a las tonalidades de color del diagrama de casos de usos y que se verá más adelante.

4.2.2 Diagrama de Clases General

Los paquetes son los módulos de Java. Recipientes que contienen clases y que se utilizan tanto para mantener el espacio de nombres dividido en compartimentos más manejables como un mecanismo de restricción de visibilidad. Se pueden poner clases dentro de los paquetes restringiendo la accesibilidad a éstas en función de la localización de los otros miembros del paquete.

La estructura general que presenta nuestra aplicación es la siguiente:

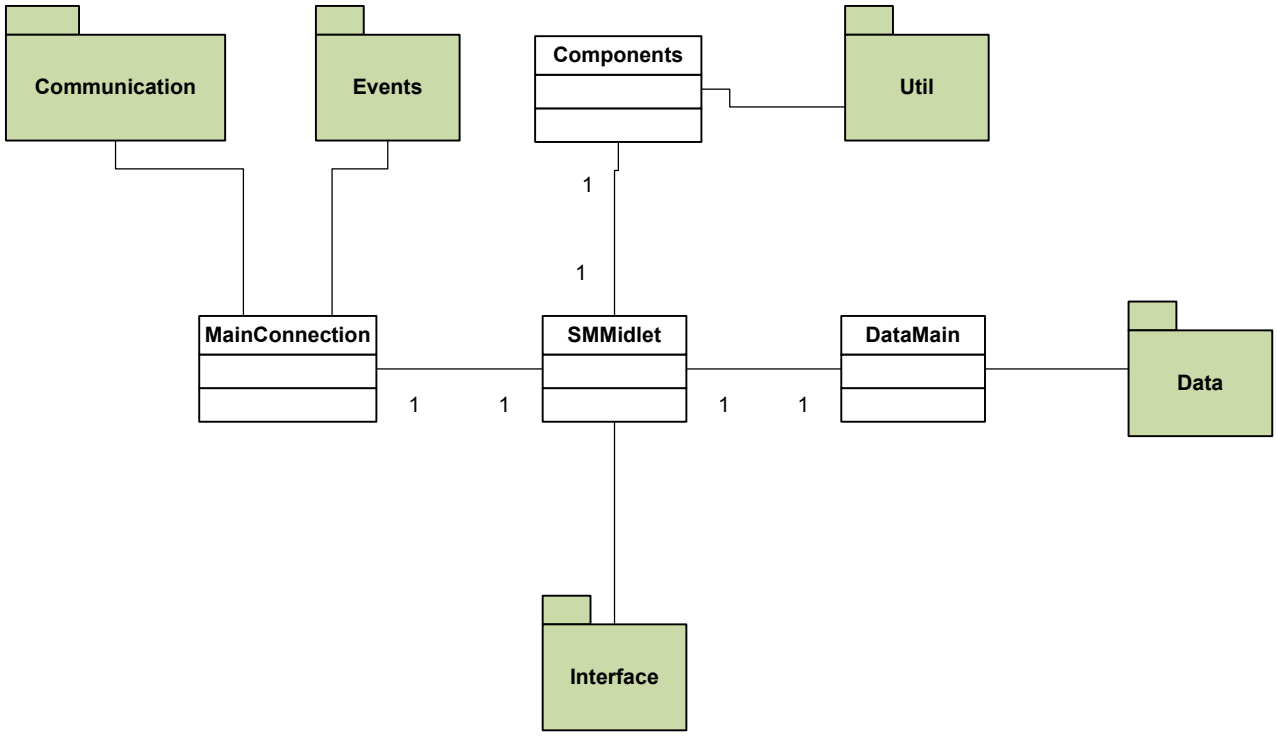


Figura 4.2 Diagrama de Clases General

Se puede apreciar que ya en el diseño del diagrama de clases hemos dividido el programa en tres partes que son la parte de la interfaz, comunicaciones y la parte de datos y utilidades.
La clase principal **SMMidlet** es la encargada de controlar todas las demás clases, es decir, todo el programa necesita acceder a ella para acceder a otros módulos, sirve como de pasarela. Así mismo, la clase **DataMain** es la segunda más importante ya que contiene toda la información que usará la aplicación (configuración de la interfaz, datos de la conexión, formularios, etc). Por el momento y como primera muestra este sería el diagrama de clases principal donde no se han desarrollado todos los paquetes para mejor comprensión.

4.2.3 Diagrama de Clases Detallado

Podemos ir abriendo los paquetes del diagrama general e ir viendo las dependencias que surgen entre distintas clases o distintos paquetes.

Por ejemplo, la sección de la interfaz la podemos desarrollar de la siguiente manera:

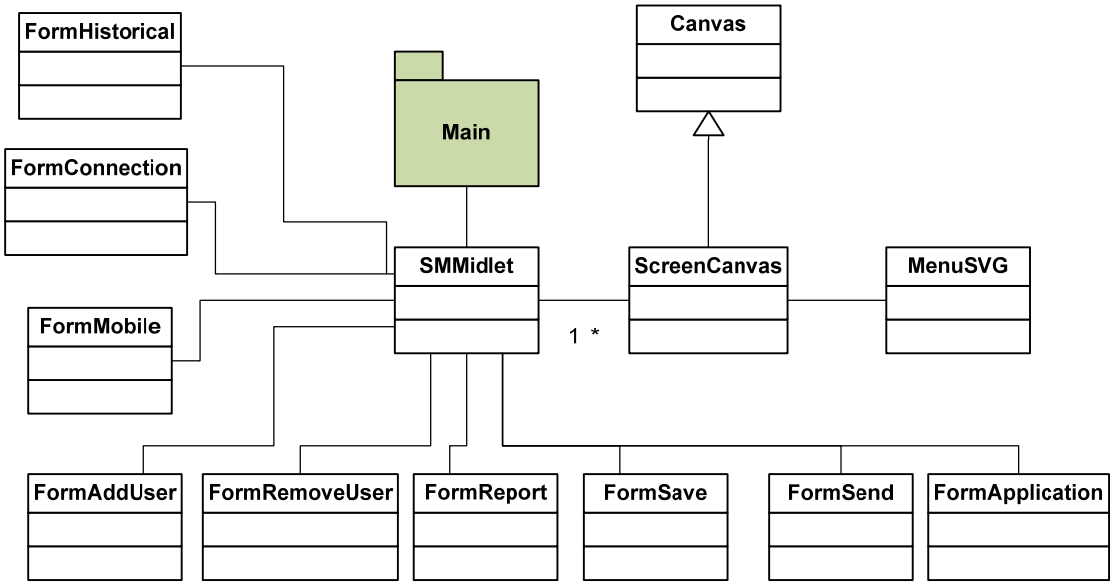


Figura 4.3 Diagrama de Clases para de la Interfaz

Todos los formularios están supervisados por la clase principal ya que es ésta la que se encarga de visualizarlos o no. La parte de gráficos SVG se realiza mediante una serie de **ScreenCanvas** que heredan de la clase **Canvas** proporcionada por J2ME.

Otro ejemplo de implementación lo vemos en el diagrama de la **Figura 4.2.3.1** donde aparece el componente RMSData utilizado para el almacenamiento de los distintos usuarios que hacen uso de la aplicación y el histórico de acciones. Los formularios que hacen uso de **RMSData** pueden acceder a esta clase pasando por la clase principal SMMidlet.

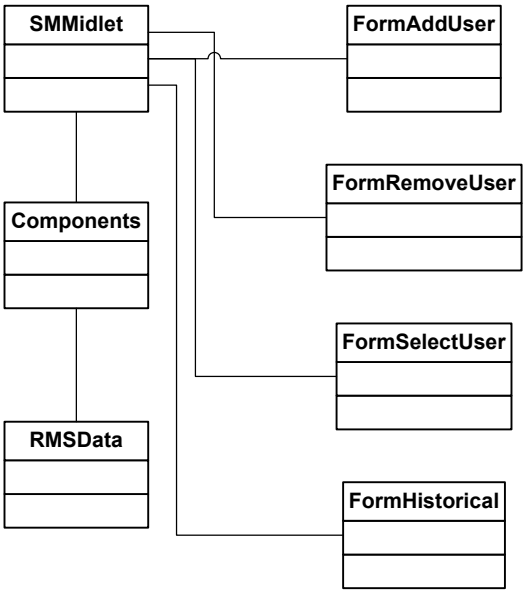


Figura 4.4 Diagrama de Clases Gestión RMS

Para el módulo de conexiones es interesante analizar aparte el diseño del diagrama para que tengamos claro de forma general el funcionamiento de una conexión Jabber. En la **Figura XXX** vemos como el núcleo del sistema, SMMIdlet, accede a una clase que hemos diseñado separadamente llamada **MainConnection** y que se encarga de lanzar las llamas a las clases encargadas de la conexión.

Las conexiones las establece el **CommunicationManager** que utiliza un parseador programado exclusivamente para Nodos Jabber y una serie de herramientas de codificación y creación de identificadores aleatorios.

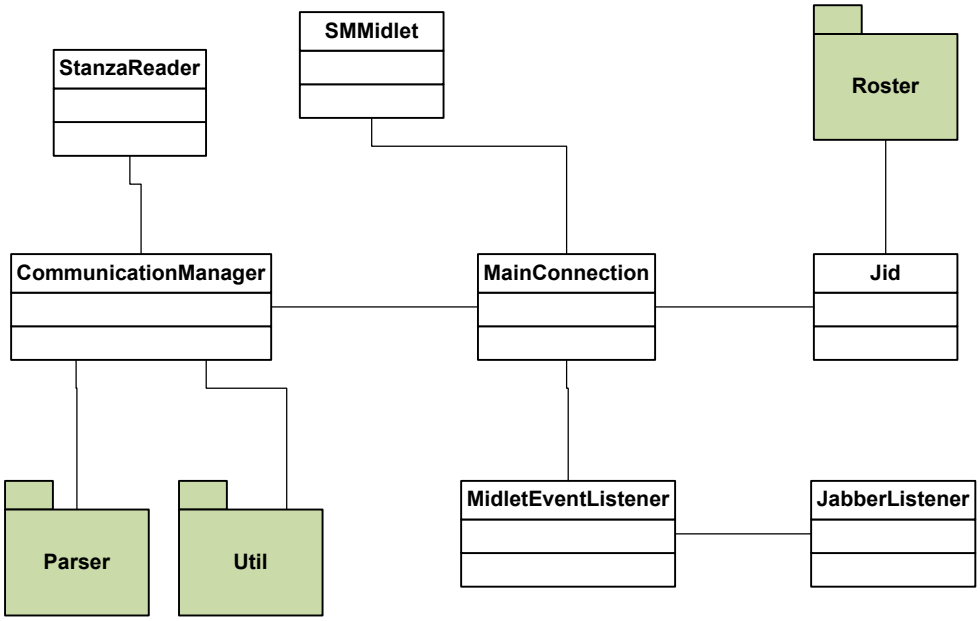


Figura 4.5 Diagrama de Clases Conexiones

El envio de datos se realiza mediante la ejecución de varios hilos concurrentes que son creados y gestionados por la clase **HttpBindThread** que, como su nombre indica, se encarga de hacer las conexiones vía HTTP y que deriva de la clase base **Thread.**

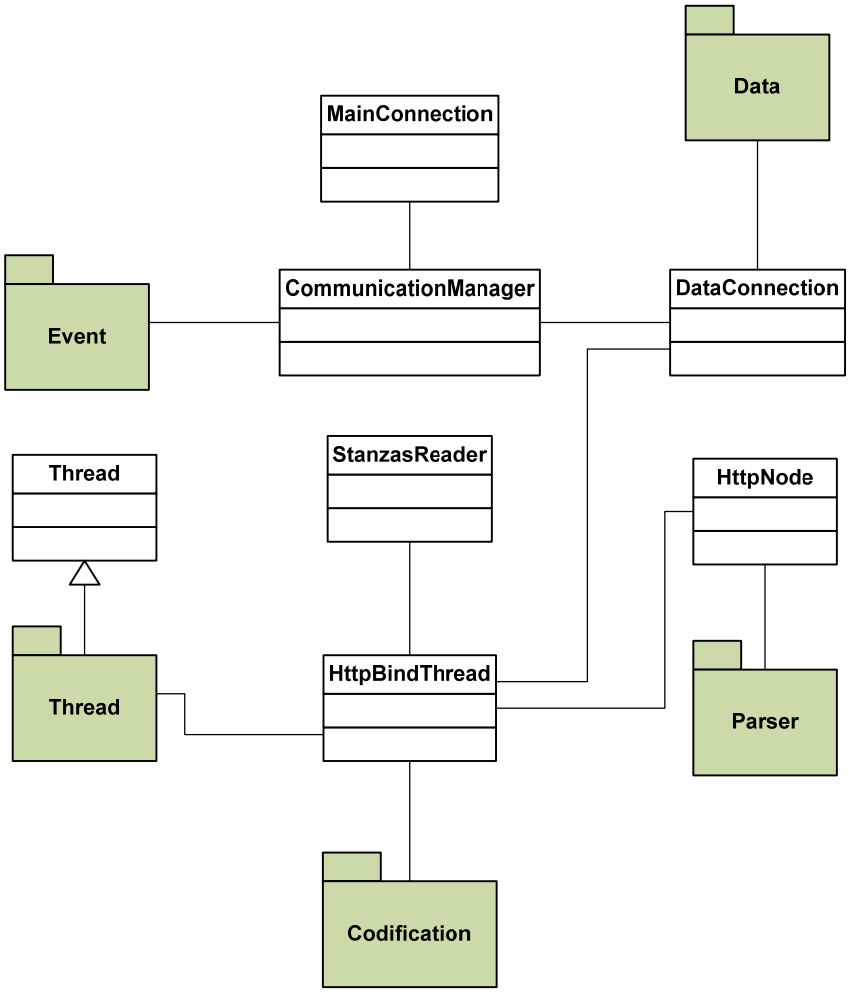


Figura 4.6 Diagrama de Clases parte del Envío de Información

Por último se ha creado una clase encargada de gestionar todos los eventos que se produzcan (conexiones, mensajes enviados, mensajes recibidos, errores, etc.) y de transmitir esas acciones al midlet para que se visualicen por pantalla.

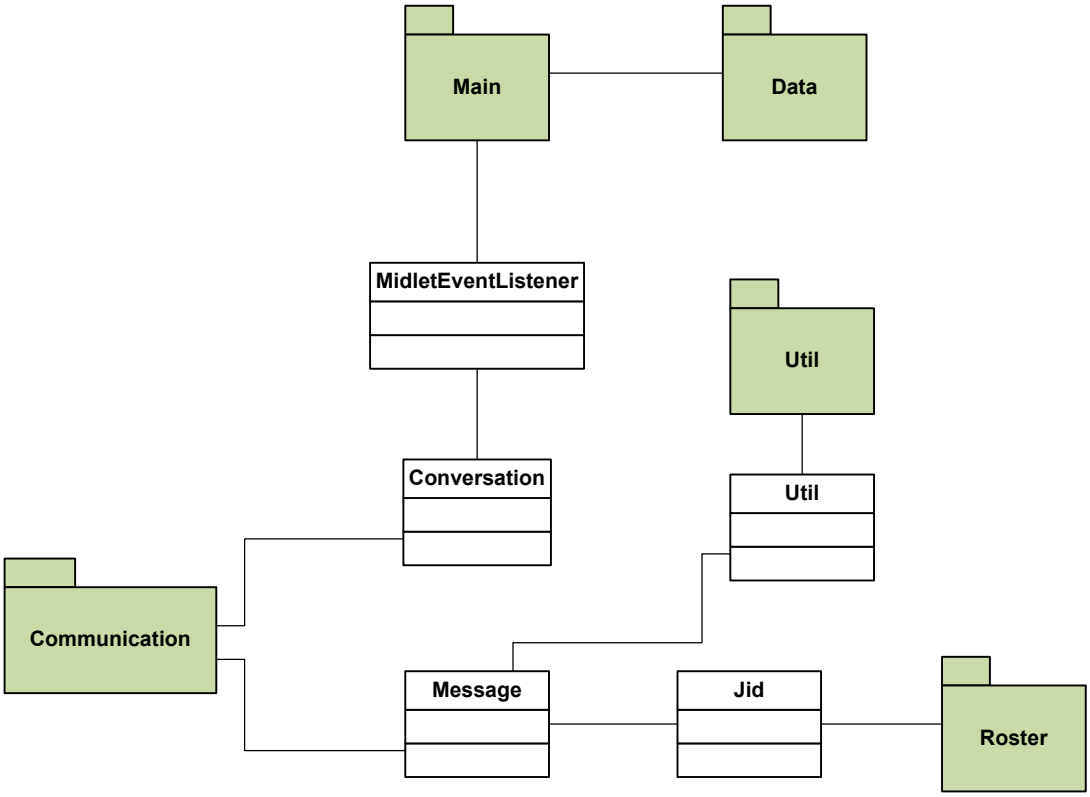
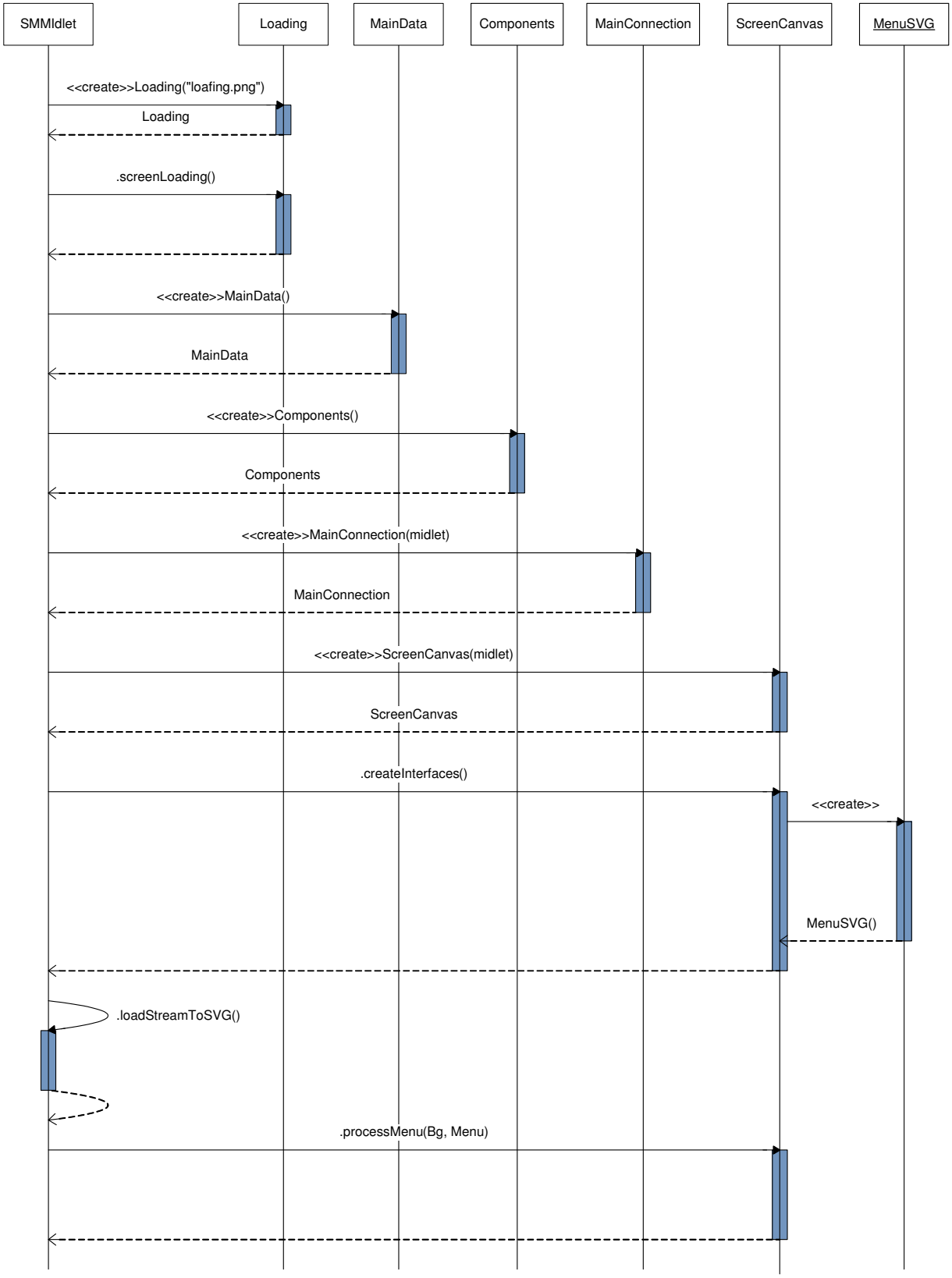


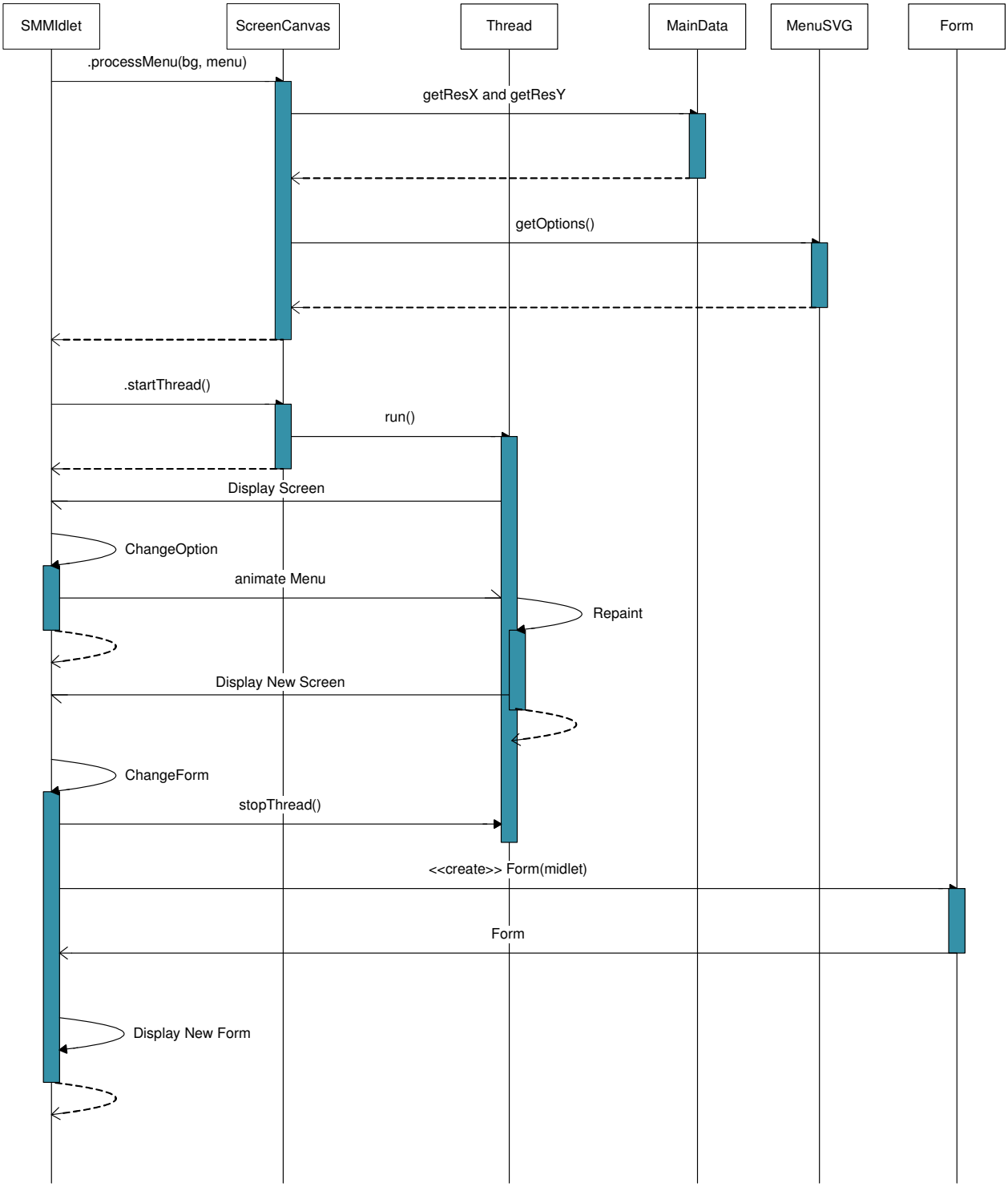
Figura 4.7 Diagrama de Clases Gestión de Eventos y Mensajes Recibidos

4.2.4 Diagramas de Secuencia

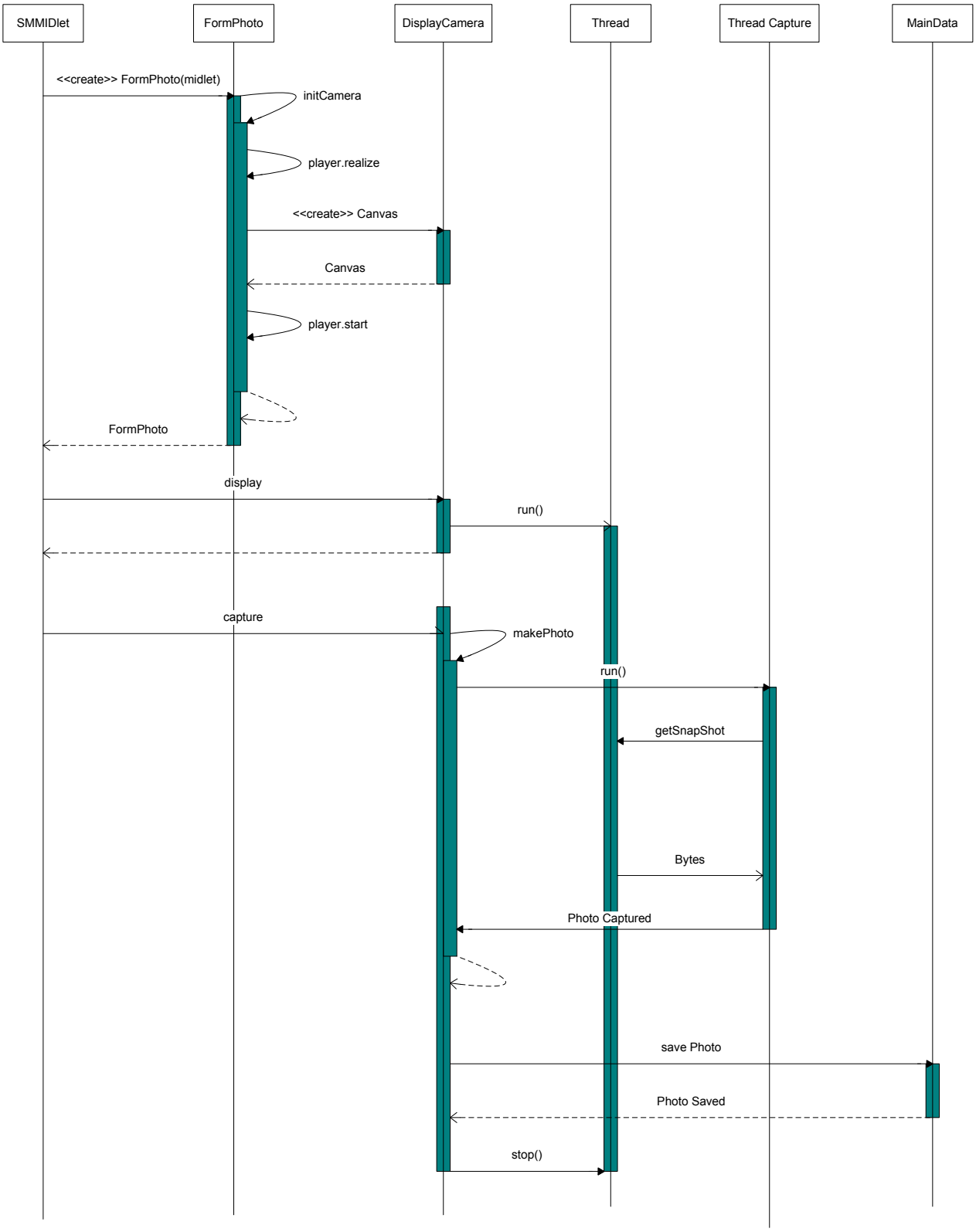
- Inicio de la aplicación



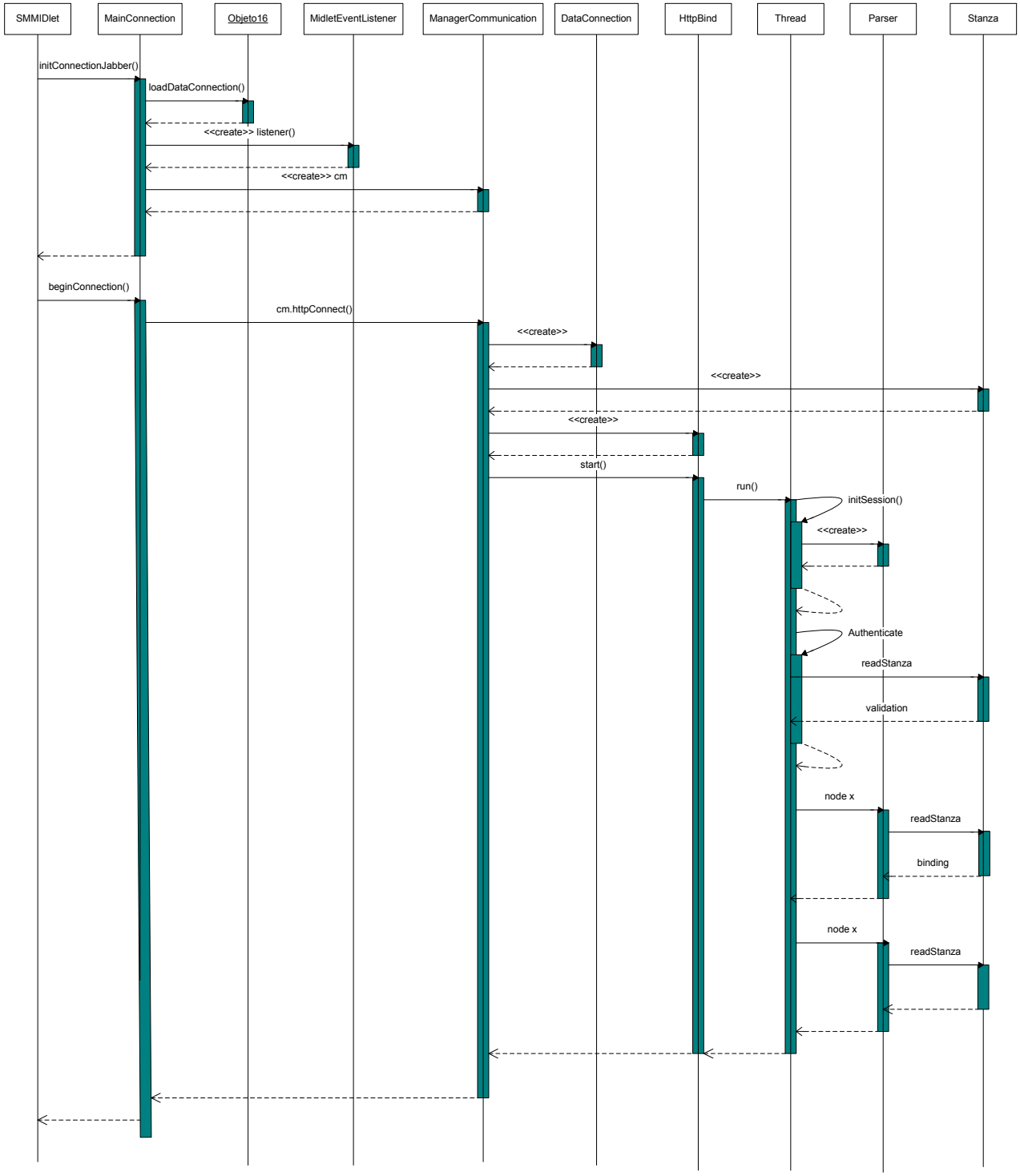
- Generación de los menús SVG



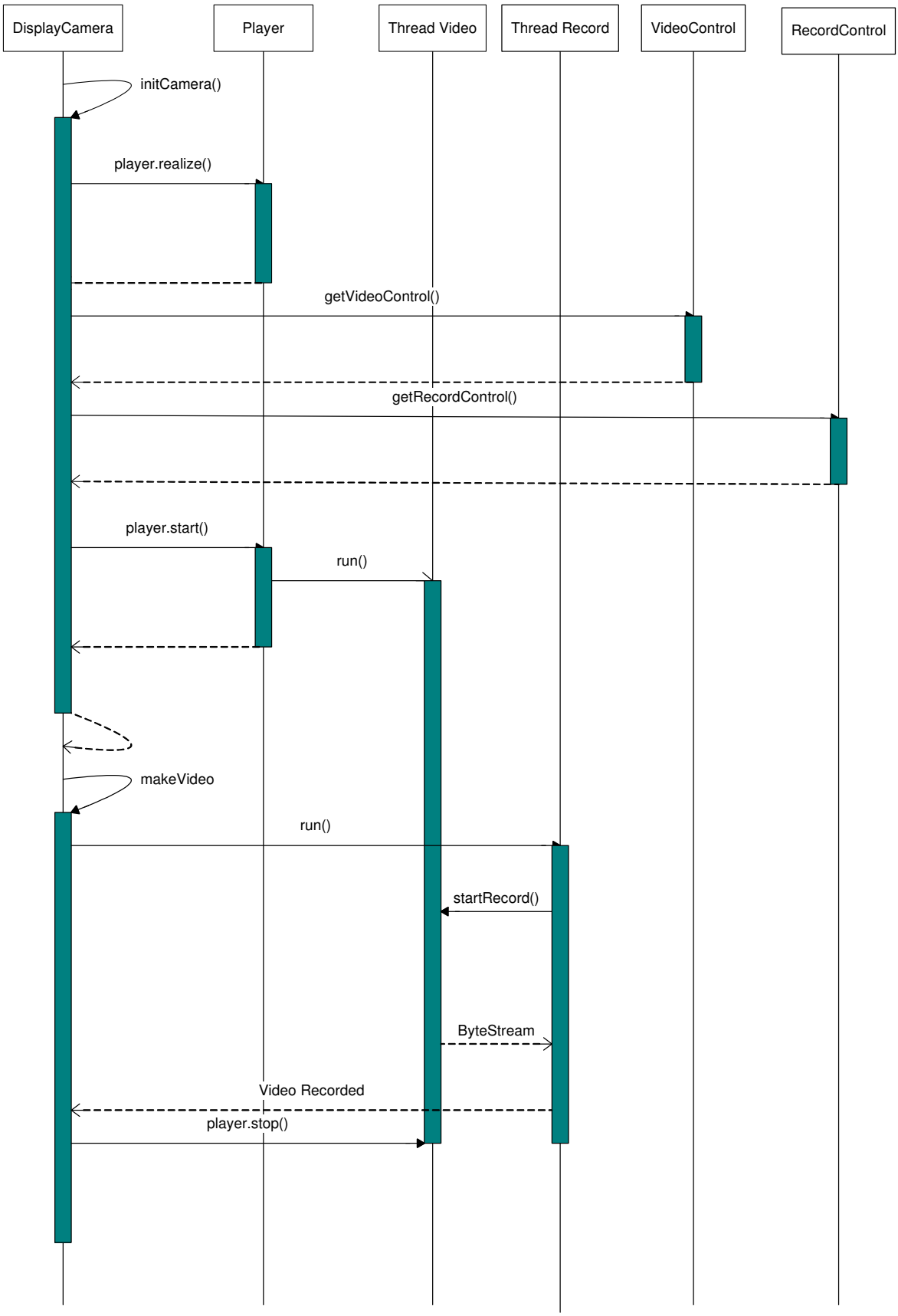
- Lanzar cámara y capturar fotos



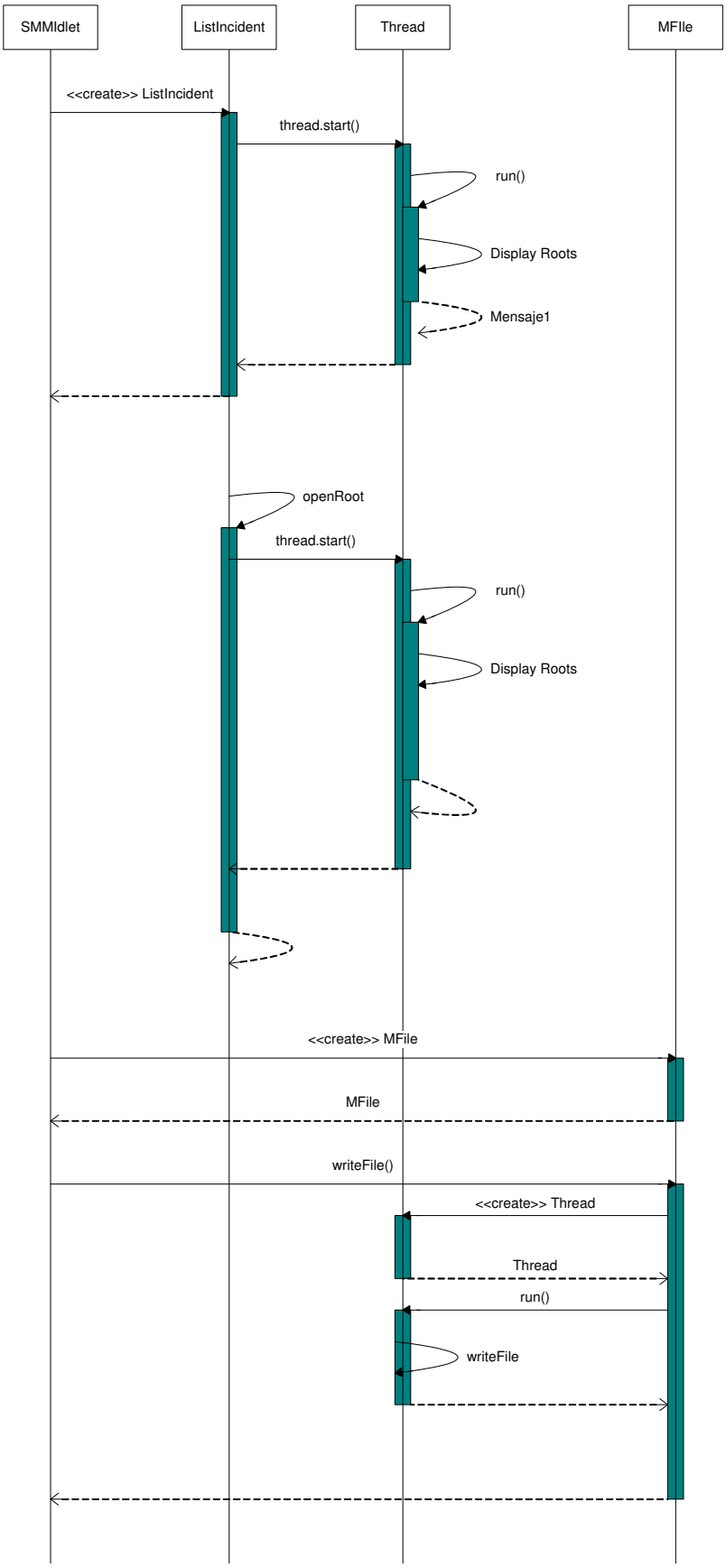
- Iniciar una conexión Jabber



- Grabar Video



- Tratamiento de Ficheros



4.2.5 Diseño de la Interfaz

El diseñar una buena interfaz es una de las partes más delicadas ya que, en esta ocasión, tenemos que considerar más aspectos, como por ejemplo, que vamos que crear un entorno para un móvil y por lo tanto su tamaño será reducido. Además de tener en cuenta su tamaño hay que darse cuenta que tiene que ser un sistema sencillo de manejar y amigable, sin muchas pantallas y con iconos que representen la información mostrada.

En la **Figura 4.7** vemos lo que sería el ciclo de vida de la interfaz, desde que se inicia la aplicación pasando por los diferentes menús, formularios, avisos y elementos multimedia hasta su finalización.

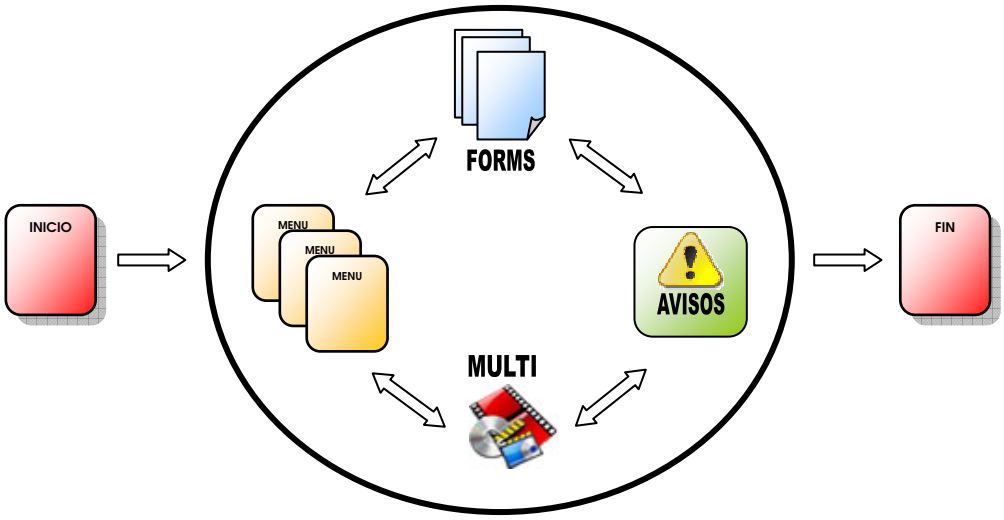


Figura 4.8 Ciclo de Vida de la Interfaz propuesta

Estructura principal de una pantalla de menú, como vemos en la **Figura 4.8** es muy sencilla y, a pesar de ser lanzada en una pantalla reducida, los elementos son bastantes grandes para distinguirlos.

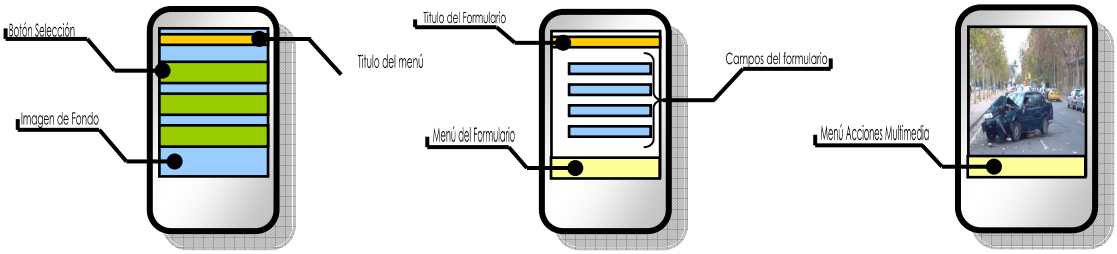


Figura 4.9 Prototipos de la Interfaz

Esta sería la estructura general de una pantalla de formulario. Cada formulario tiene un título, unos elementos como campos de texto, casillas, desplegables… y por último una barra de menú con diferentes acciones (guardar, volver atrás, capturar, etc.).

Para el caso de las opciones multimedia (Foto y Video) tenemos otra estructura muy simple que básicamente consiste en la imagen captada por la cámara y un menú de acciones tales como capturar foto, grabar video, guardar, volver atrás, etc.

4.2.6 Diseño de la Estructura de los Mensajes

El diseño y la estructura que damos a los paquetes que enviamos al servidor es una tarea importante ya que, aunque la recolección de datos sea satisfactoria, si falla, todo el trabajo habrá sido en vano. Además, nos pusimos como requisito que dichos mensajes fueran fácilmente y rápidamente descifrados en el otro lado.
La Figura 4.9 muestra una organización de los datos que recibirá el servidor cuando el cliente envíe el informe.

Estructura de los mensajes

Figura 4.10 Estructura de un paquete de información

DR	Datos de Registro	**DL**	Datos de Localización
DU	Datos de Usuario	**DM**	Datos Multimedia
DI	Datos de Incidente	**FOTOS/VIDEO**	Capturas de fotografías y videos

5. Implementación

En esta parte de la memoria vamos a empezar a programar el sistema previamente estudiado y diseñado. Para ello, primero tenemos que hacer una elección minuciosa de todos los componentes y herramientas que vamos a utilizar, y segundo procederemos a su codificación.

5.1 DESIGNACIÓN DE COMPONENTES

Una vez estudiado el problema que queremos solucionar y los requisitos que nos hemos impuesto es la hora de seleccionar qué componentes nos harán falta y qué herramientas de programación utilizaremos para satisfacer nuestras necesidades.

5.1.1 Elección del Dispositivo Móvil

Lo primero que tenemos que escoger antes que nada es dónde vamos a instalar nuestro sistema, es decir, en qué dispositivo móvil puede funcionar correctamente y al mismo tiempo cumplir todas las exigencias impuestas.

En el Estado del Arte hemos estudiado los diferentes tipos de dispositivos móviles que podemos encontrarnos hoy en día en el mercado pero, a pesar de que todos comparten la misma característica (movilidad), muchos de ellos no se adaptan a nuestros propósitos.

5.1.1.1 Comparativa de Dispositivos Móviles

Existen muchos tipos de dispositivos pero para este análisis hemos seleccionado un pequeño grupo de diferentes características para compararlos y ver qué terminales se adaptarían mejor, a priori, a nuestro sistema.

Hemos seleccionado los siguientes:

- En la categoría de portátiles: Vaio TZ y LG R500
- En la categoría de Tablet PC: LG Tablet PC
- En la categoría de móviles: Nokia N95 y N73
- En la categoría de UMPC: HTC SHIFT
- En la categoría de PDA: HP IPAQ y BlackBerry Pealt 8100

Una vez seleccionados los posibles candidatos procederemos al análisis detallado (**Tabla 5.1**) de las características más relevantes como por ejemplo el tamaño, peso, tamaño de la pantalla, etc.

	Características Técnicas									
Dispositivo Móvil	**Dim**	**Peso**	**Pantalla**	**CPU**	**RAM**	**ALM**	**S.O**	**Java**	**Internet**	**Precio**
Portátil Vaio TZ	27x30x2	1,2 Kg	11,1"	1,2 Ghz x2	2048 MB	120 GB	Windows - Linux	Si	Si	2.399
LG Tablet PC	27x20x2,3	1 Kg	10,6"	1,2 Ghz x2	1024 MB	80 GB	Windows - Linux	Si	Si	1.599
Portátil LG R500	36x26x3,3	2,8 Kg	14,4"	2,2 Ghz x2	1024 MB	160 GB	Windows - Linux	Si	Si	899
PDA HP IPAQ	7x12x1,3	0,164 Kg	3,5"	300 Mhz	32 MB	64 MB	Windows Mobile	Si	SI	400
Móvil N95	5,3x10x2,1	0,120 Kg	2,6"	300 Mhz	128 MB	8 GB	Symbian	Si	Si	699
Móvil N73	4,9x11x2	0,116 Kg	2,4"	211 Mhz	44 MB	42 MB	Symbian	Si	Si	199
UMPC HTC SHIFT	21x13x2,5	1,7 Kg	7"	800 Mhz	1024 MB	40 GB	Windows - Linux	Si	Si	1.244
BlackBerry Pearl	5x11x1,5	0,089Kg	2,6"	300 Mhz	16 MB	64 MB	Symbian	Si	Si	249

Tabla 5.1 Comparativa de Dispositivos Móviles

Después de este pequeño estudio llega la hora de seleccionar en qué dispositivos se acoplaría mejor nuestro proyecto. Para ello recordaremos que nuestra aplicación viajará dentro de un dispositivo con una movilidad elevada, es decir, de poco peso, que tenga una potencia de procesado considerable, abundante memoria y alta capacidad de almacenamiento sin olvidar que debe cámara de fotos, conexión a Internet y buen precio.
Si estos requerimientos los aplicamos a la tabla anterior nos quedará una nueva tabla donde veremos la valoración de cada aparato.

	TA	PE	PA	RE	ME	CA	VI	JA	WM	SY	PR	IN	Valoración
Dispositivo Móvil													
Portátil Vaio TZ	Regular	Regular	Bueno	Bueno	Bueno	Malo	Malo	Bueno	Malo	Malo	Malo	Bueno	6
LG Tablet PC	Malo	Regular	Regular	Bueno	Bueno	Malo	Malo	Bueno	Malo	Malo	Malo	Bueno	5
Portátil LG R500	Malo	Malo	Bueno	Bueno	Bueno	Malo	Malo	Bueno	Malo	Malo	Regular	Bueno	5,5
PDA HP IPAQ	Bueno	Bueno	Bueno	Malo	Malo	Malo	Malo	Bueno	Bueno	Malo	Bueno	Bueno	7
Móvil N95	Bueno	Bueno	Bueno	Malo	Malo	Bueno	Bueno	Bueno	Malo	Bueno	Regular	Bueno	**8,5**
Móvil N73	Bueno	Bueno	Bueno	Malo	Malo	Bueno	Bueno	Bueno	Malo	Bueno	Bueno	Bueno	**9**
UMPC HTC SHIFT	Regular	Malo	Regular	Regular	Regular	Malo	Malo	Bueno	Bueno	Malo	Malo	Bueno	5
BlackBerry Pearl	Bueno	Bueno	Bueno	Malo	Malo	Bueno	Malo	Bueno	Malo	Bueno	Bueno	Bueno	**8**

TA	Tamaño
PE	Peso
PA	Pantalla
RE	Rendimiento
ME	Memoria
CA	Cámara
VI	Video
JA	Java
SY	Symbian
PR	Precio
IN	Internet
WM	Windows Mobile

Malo

Regular

Bueno

Tabla 5.2 Elección del Dispositivo Móvil Ideal

CONCLUSIÓN

Según el estudio realizado vemos que los candidatos correspondes a dispositivos de telefonía móvil y a una PDA. La PDA la tenemos que descartar forzosamente debido a que no dispone de grabación de video. Nos queda por tanto dos móviles de última generación que disponen de todos los requisitos exigidos, la diferencia de potencia con respecto a otros dispositivos se compensa utilizando programación adaptada para móviles.
La pequeña diferencia entre estos dos móviles reside en la relación calidad precio, el N95 tiene más potencia que el N73 pero su precio es prácticamente el doble.
Esta diferencia de precio comparada con el precio total que tiene el producto es imperceptible por lo que ganaremos en potencia eligiendo el **Nokia N95 en su versión de 8GB**.

5.1.2 Elección del Lenguaje de Programación

Después de la elección del dispositivo donde irá la aplicación urge decidir qué lenguaje de programación utilizaremos para la implementación.

El abanico de posibilidades se ve reducido al haber seleccionado previamente el terminal ya que éste dispone únicamente del Sistema Operativo Symbian, por lo que tendríamos que utilizar C++, o usar la maquina virtual de Java.

C++ y Java prácticamente tienen las mismas características (programación orientada a objetos, declaración de clases, utilizan los mismos tipos de datos con pequeñas diferencias, etc.) pero también tienen diferencias que serán las que nos digan cuál elegir.

Mientras que C++ dispone de destructores en Java esta característica no existe porque dispone de un **reciclador de memoria** que la libera automáticamente cuando no la utiliza.
Esta es una diferencia importante a la hora de programar para móviles ya que no tendremos tanta libertad y tendremos que tener cuidado con no sobrecargar el sistema.
A pesar de que los móviles de última generación llevan incorporada bastante memoria RAM no es motivo para pasar por alto este detalle.

Por otro lado si queremos utilizar C++ estaríamos obligados, por así decirlo, a utilizar móviles que dispongan de un sistema operativo que entienda este lenguaje, como por ejemplo Symbian o Windows Mobile. No sería una buena elección ya que nos reduce el rango de dispositivos donde poner la aplicación.

Por lo tanto, y a pesar de que Java tiene una gestión de memoria bastante mala, para a lo que a móviles se refiere, será el lenguaje elegido por dos razones: es multiplataforma y dispone de APIs que agilizará el desarrollo de ciertas partes del proyecto.

5.1.3 Elección del Tipo de Arquitectura

Elegir una buena arquitectura de comunicaciones es importante a la hora de entender el funcionamiento real de la aplicación. Dependiendo de elegir una u otra la información viajará de una forma diferente por lo que primero tenemos que tener claro qué es lo vamos a desarrollar.

Tal y como se muestra en la **Figura 5.1** el flujo de información viaja en las direcciones de las flechas y en nuestro caso implementaremos la parte de envío/recepción desde un móvil hasta su servidor.

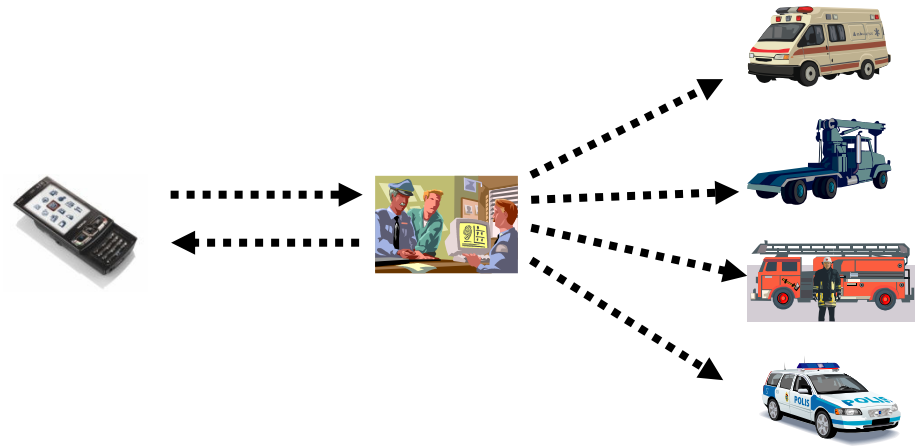


Figura 5.1 Arquitectura de Comunicaciones – Flujo de Datos

Vemos que existe una comunicación reciproca entre el móvil y la central de policía, significa que la aplicación de móvil necesita de información proporcionada por la central para, por ejemplo, validar un informe creado. En este caso vemos que la arquitectura empleada corresponde a una tipo **Cliente-Servidor** donde el cliente es el móvil que recibirá una serie de datos la central. De la misma forma, el móvil también proporciona datos a la central enviando un informe de un incidente pero este envío no es solicitado por la central.

Por el otro lado vemos que la central proporcionará datos a los diferentes servicios públicos avisando de un posible incidente.

Resumiendo, la arquitectura más adecuada a esta situación sería la arquitectura basada en **Cliente-Servidor.**

5.1.4 Elección del Tipo de Comunicación

Llegamos a la parte que dará sentido a todo el trabajo que estamos realizando. Se trata de un pilar fundamental en nuestra aplicación ya que es el encargado de que todos los datos recogidos por la terminal móvil lleguen a su destino de una forma rápida, segura e íntegra.
Aquí lo tenemos más fácil para elegir pues estamos limitados por las características que disponga nuestro móvil. En principio un Midlet (programa Java para dispositivos embebidos) puede establecer diversos tipos de conexiones: Sockets, http, Https, datagramas, y otras, sin embargo, el Standard sólo obliga a la implementación del protocolo http.

Hay que tener presente que la elección de protocolos y métodos de comunicación tiene que adaptarse a las particularidades que presente el servidor. Además, ya comentamos que una necesidad de la aplicación es que los mensajes sean fácilmente descifrables en el otro lado.

Surge una necesidad de que tanto el cliente como el servidor se pongan de acuerdo con el método de comunicación y de que, además, se adapte con las pautas que rige el móvil.

Por lo tanto y siguiendo el modelo de referencia de Interconexión de Sistemas Abiertos (OSI, Open System Interconnection) los protocolos que utilizaremos son los descritos en la **Tabla 5.1**.

Nivel de Aplicación
HTTP, Jabber, XMPP
Nivel de Presentación
XML
Nivel de Sesión
SSL
Nivel de Transporte
TCP
Nivel de Red
IP
Nivel Enlace de Datos
WiFi
Nivel Físico
Radio

Tabla 5.3 Estructura en Niveles de la parte de comunicación

Como novedad a los protocolos más conocidos se encuentran el protocolo **Jabber** y el **XMPP.**

Jabber

Con Jabber solucionamos el problema que surgía entre el cliente y el servidor ya que este protocolo es gratuito, dispone de varios servidores fácilmente instalables y de un protocolo de comunicación basado en XML (**Figura 5.2**) lo que agilizará el descifrado, por así decirlo, del paquete que enviemos.

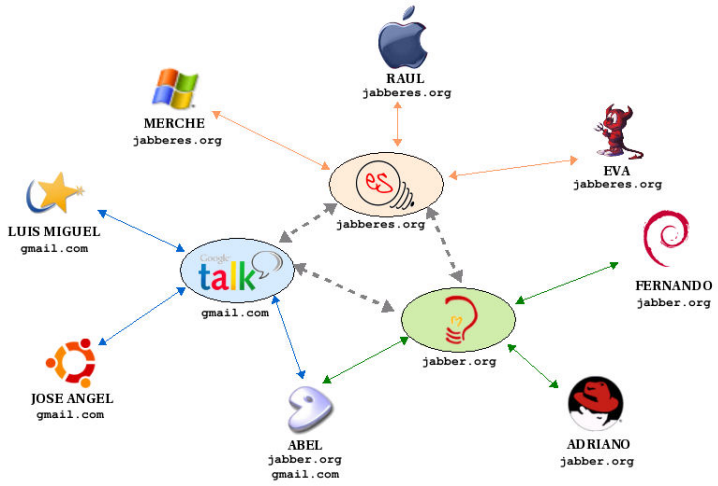


Figura 5.2 Comunicación de Jabber con otros sistemas de mensajería instantánea utilizando XML [W3 JABBERES]

5.1.5 Herramientas Seleccionadas

Prácticamente ya tenemos todo el material necesario para empezar a implementar nuestro proyecto, solo falta seleccionar entre cientos de herramientas de programación y diseño gráfico cuáles nos aprovecharán más y cuáles nos permitirán programar de una manera cómoda, sencilla y rápida.

Aquí también influye mucho los conocimientos del programador y la experiencia que tenga en el desarrollo de nuevo software, concretamente para móviles. Después de probar varios compiladores de Java nos quedamos con el entorno NetBeans que viene acompañado por una herramienta de emulación de móviles y PDAs (Wireless Toolkit).
En la parte gráfica, como trabajamos con gráficos SVG, nos decantamos por utilizar el Illustrator que es un potente creador de gráficos vectoriales. Para la parte de animaciones de menú utilizaremos el Ikivo Animator apoyado de otras herramientas como editores XML.

5.1.5.1 NetBeans 6.0

El NetBeans es una plataforma para el desarrollo de aplicaciones de escritorio usando Java y un entorno de desarrollo integrado IDE **[W3 NETBEANS]**.
La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java

escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole otros nuevos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Actualmente el NetBeans incorpora elementos esenciales para la programación de dispositivos móviles. A partir de la versión 5.5 se añadió esta funcionalidad debido al gran aumento de la demanda de aplicaciones Java para los móviles. En su versión 6.0 este aspecto ya viene incluido con retoques asignados al desarrollo de juegos Java para terminales.
Además, cosa que nos viene muy bien, en la versión 6.0 se empieza a dar más importancia a los gráficos SVG e incorpora un editor XML interno para modificar dichos archivos.

Cabe decir que a partir de la versión 3.5 se fue adaptando para proyectos escritos en C++ y actualmente esta plataforma soporta Ruby, SOA y UML.

5.1.5.2 Sun Wireless Toolkit 2.5.2

Cuando se desarrolla una aplicación tenemos la necesidad de ir viendo poco a poco como está quedando. Cuando se trata de programas diseñados para dispositivos externos, por ejemplo móviles, placas FPGA o memorias ROM, tenemos que instalar el software en ellos.
Sucede que no podemos estar continuamente cambiando código e instalándolo nuevamente en las máquinas y probarlo, se necesitan de emuladores que simulen el dispositivo que estamos utilizando.

Tratándose en este proyecto de móviles existen muchos emuladores, la mayoría de ellos proporcionados por el fabricante, que emulan casi a la perfección su comportamiento.

No obstante nosotros utilizaremos un emulador genérico al cual podemos modificar para que se comporte como lo haría nuestro móvil real. El Wireless Toolkit de Sun nos proporciona un emulador de distintos dispositivos móviles (móviles a color, en B/N, PDAs…) el cual se puede acoplar a la plataforma NetBeans para hacer las pruebas.
En la versión 2.5.2 tenemos la posibilidad de cambiar especificaciones internas del móvil como por ejemplo el tamaño de la RAM, tolerancia del GPS o añadir y quitar APIs.

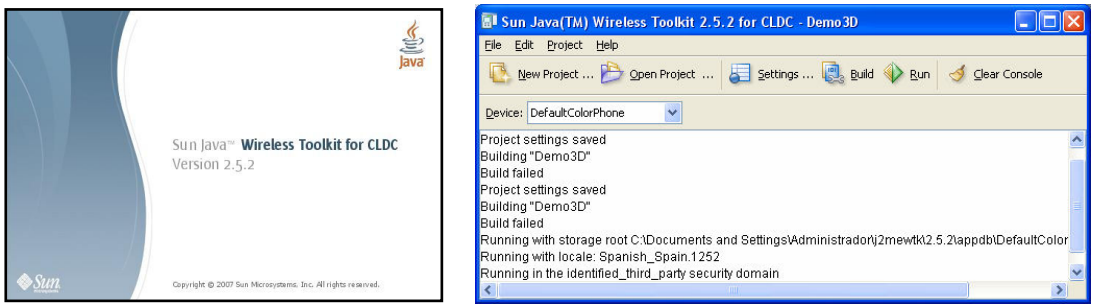


Figura 5.3 Imágenes del Wireless Toolkit 2.5.2

Como ya hemos nombrado, Nokia nos ofrece también su emulador (**Figura 5.4**) particular que emula todos sus dispositivos, tan sólo tendríamos que seleccionar el tipo de emulador según la versión de Symbian que incorporen. Dicho emulador es integrable en NetBeans lo que ocurre es que es excesivamente lento y no incorpora la posibilidad de cámara fotográfica y, por lo tanto, sólo lo utilizaremos para las pruebas finales.

Figura 5.4 Emulador de Nokia Forum para los S60 aportado de Nokia Forum [W3 FORO NOKIA]

5.1.5.3 Illustrator CS3

Entre un amplio surtido de instrumentos de diseño de gráficos vectoriales SVG hemos escogido el Illustrator versión CS3 propiedad de Adobe Systems. Una de las razones por la cual lo hemos escogido es porque tiene la posibilidad de guardar los gráficos SVG para el entorno Tiny 1.1.

N O T A

Los gráficos SVG tienen una serie de perfiles que definen el tipo de documento XML, en nuestro caso el perfil SVG Tiny 1.1, subconjunto del SVG 1.1, esta adecuado para archivos que van a ver en dispositivos pequeños. Hay que tener en cuenta que no todos los teléfonos móviles soportan perfiles SVG Tiny y por lo tanto los gráficos no se verán correctamente.

5.1.5.4 Ikivo Animator

Los gráficos SVG también se pueden animar, es decir, se les pueden aplicar transformaciones como translaciones, escalados o degradados simplemente añadiendo las etiquetas adecuadas en sus archivos XML.

Pocas aplicaciones hay para la animación de gráficos vectoriales y una de las más sencillas es el **Ikivo Animator** que tiene una similitud, en cuanto a animar objetos, muy parecida a Adobe Flash como puede apreciarse en la **Figura 5.5**.

Para nuestra aplicación nos bastará animar un poco los menús principales para darle un toque de interactividad y movimiento.

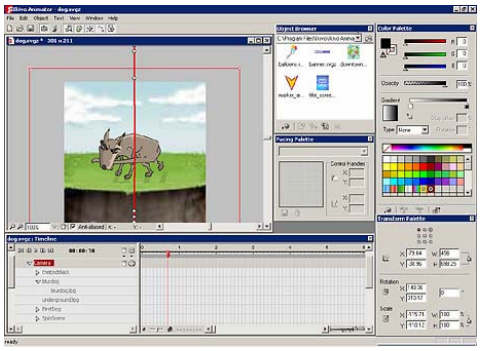


Figura 5.5 Interfaz del programa Ikivo Animator

5.1.5.5 Otras Herramientas de Apoyo

Cabe nombrar algunas utilidades usadas para la edición de archivos XML como por ejemplo el **NotePad XML** que respeta la estructura de los SVG Tiny 1.1.

En cuanto a las pruebas de comunicaciones hemos utilizado 2 clientes Jabber y un servidor todos ellos totalmente gratuitos y de código abierto.
Spark es un cliente Jabber que hace la función de emular la central de policía, es decir, a este cliente le llegarían los mensajes enviados por el móvil (esta técnica se explicará más adelante).
PidGin es otro cliente que además de Jabber soporta otros protocolos como IRC, Messenger, etc. y cuya ayuda incorpora una consola de depuración donde se observan los mensajes XML que circulan por la red.
Como servidor Jabber hemos utilizado **OpenFire** que está escrito en Java con licencias comerciales y GNU. Además es uno de los mejores servidores de comunicaciones en tiempo real.

5.2 PROGRAMACIÓN Y DISEÑO

Empezar a implementar una aplicación es una tarea costosa y a veces si no se tiene una buena planificación y una buena estructuración de lo que se quiere hacer puede desembocar en un producto mal construido, en un plazo de tiempo superior al estimado y con muchos defectos.
Previamente hemos diseñado y analizado la aplicación que queremos instaurar separándola en distintos módulos claramente diferenciados.

5.2.1 Núcleo del Sistema Principal

El núcleo del sistema, o también llamado Midlet, es el programa principal donde la aplicación empezará a funcionar y donde terminará. El Midlet es una aplicación Java realizada usando la especificación MIDP, es decir, utiliza la funcionalidad aportada por MIDP y por CLDC.
MIDP es el acrónimo de **Perfil para dispositivos de información móvil** que se apoya en CLDC el cual contiene aspectos básicos para ejecutar Java.

Actualmente la versión de MIPD más utilizada para programación móvil es la MIDP-2.0 siendo no compatible con la antigua, la MIDP-1.0.
Existe una versión más actualizada, la MIDP-2.1, con pequeñas diferencias con respecto la 2.0 pero no está siendo muy usada.

Un midlet siempre estará compuesto, al menos, por una clase principal que hereda directamente de la clase *javax.microedition.midlet.MIDlet*. Dicha clase con tiene 3 funciones básicas de funcionamiento:

- startApp(): inicio de la aplicación.
- pauseApp(): pausa en el hilo principal de la aplicación.
- destroyApp(): destrucción de la aplicación en memoria.

Estos métodos pasan de unos a otros como si fueran una maquina de estados.

Diagrama de Estados de un Midlet Figura 5.6

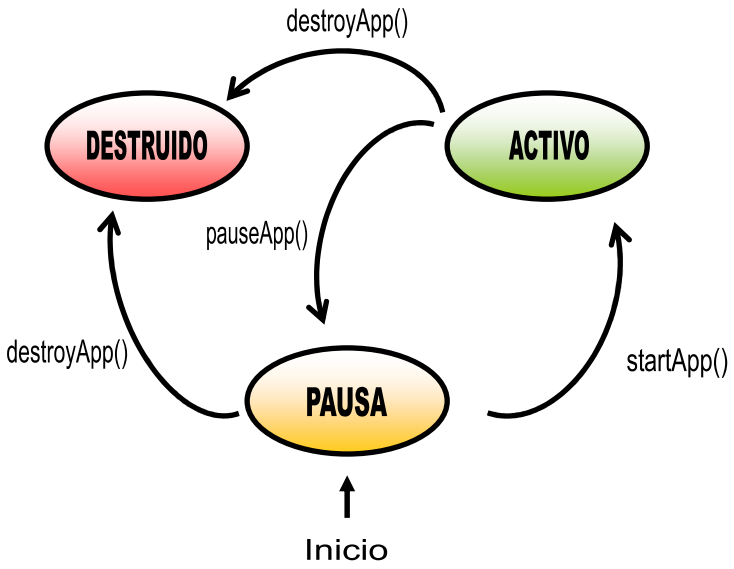


Figura 5.6 Ciclo de Vida de un Midlet

El MIDlet principal (**SMMIDlet**) llevará el control absoluto de todos los módulos que intervienen en este proyecto (interfaz, comunicaciones, datos, ficheros…) y gestionará todos los datos que pasen por él.

5.2.1.1 Inicialización de Componentes

Creación e inicialización de todas las variables de entorno y todos los componentes

Al tratarse de una aplicación bastante amplia es recomendable agrupar y organizar todas las variables, componentes, etc. que se vayan a utilizar.

Al inicio de la aplicación todas las partes de las que consta el proyecto estarán inicializadas, los formularios estarán vacíos y se establecerá una configuración por defecto con todos los atributos necesarios para la puesta en marcha.

Los valores por defecto son los siguientes:

EL INFORME

No existirá ningún usuario, se asignará uno anónimo que habrá que cambiar cuando se inicie la aplicación por primera vez. Todos los campos, fotos, videos, opciones y demás estarán vacios, puestos a cero y con todas las casillas sin marcar.

CONFIGURACIÓN DEL MOVIL

La configuración inicial consta de una resolución por defecto de 320x240

CONFIGURACIÓN DE LA APLICACIÓN

Los valores iníciales de la aplicación será de:

Resolución de fotos: 160x120

Resolución de Videos: 160x120

Precisión del GPS de 60 metros horizontal y vertical con 60 seg de TimeOut

CONFIGURACIÓN DE LA CONEXIÓN

Todas las variables, Listener y Eventos están vacios desde el principio. Cuando se produzca una conexión cogerán los valores definidos por defecto o los marcados por el usuario.

5.2.1.2 Gráficos e Interfaz

Creación de los menús SVG

Cuando ya tengamos todos los componentes SVG cargados y listos en memoria es momento de empezar a crear la interfaz de usuario en el dispositivo móvil.

La creación de interfaces en J2ME es bastante sencilla y podemos mostrar 3 tipos de pantallas

- Formularios: donde las aplicaciones pueden llenar este tipo de pantalla con texto, campos típicos de los formularios, imágenes estáticas, etc.
- Screen predefinidos: como por ejemplo pantallas de ALERTA, TEXTBOX, LIST que pueden contener otros elementos como texto e imágenes.
- Canvas: donde las aplicaciones tienen control total sobre la aparición de componentes en el display y pueden acceder directamente a eventos de bajo nivel.

En este proyecto usaremos todos los tipos pero ahora lo que nos interesa es saber cómo incorporar gráficos SVG en el tipo Canvas. Las clases que utilizaremos se encuentran en **javax.microedition.lcdui.Canvas**

donde utilizaremos la clase base **Canvas** y para la manipulación de los gráficos haremos uso de los paquetes **javax.microedition.m2g.SVGImage** y **javax.microedition.m2g.ScalableGraphics.**

Una vez localizado todo el material necesario procederemos a crear una clase que será la encargada de crear, dibujar en pantalla y animar todos los gráficos (**ScreenCanvas**).
ScreenCanvas se divide en una parte que será la encarga de recoger todos los datos, imágenes y gráficos, otra parte encargada de crear la interfaz y una última que será la que haga el redibujado y las animaciones de movimiento.

La Función ***processMen(SVGImage bg, SVGImage menú)*** será la encargada de recoger lo que habíamos creado al iniciar la aplicación, es decir, las imágenes de fondo y todos los gráficos escalares.
Cuando se ejecuta lo primero que hace es parsear los archivos SVG y clasificar por nodos o elementos su contenido.
Recordemos que un archivo SVG es básicamente un archivo XML con etiquetas claramente diferenciadas y que nuestro programa distingue por nodos.

CODIGO

```
Document doc = svgMenu.getDocument();
SVGSVGElement svg = (SVGSVGElement)doc.getDocumentElement();

Vector iconVector = new Vector();
SVGElement[] iconVector = new SVGElement[getMenuSVG().getNumButtons()];
```

Después de identificar todos los nodos hay que buscar cuáles de ellos son los que corresponden a los botones u opciones que forman el menú. A la hora de diseñar los gráficos SVG con el Illustrator ya tuvimos en cuenta el separar lo que serían los botones del fondo y ayudándonos de un editor XML, como por ejemplo el NotePad XML, añadimos un atributo identificador **id** para que el parseador lo encuentre enseguida.

CODIGO

```
for (int i = 0; i < getNumOptions(); i++)
{
    String identificador = "boton_" + (i+1);

    SVGElement iconElt = (SVGElement)doc.getElementById(identificador);

    // Oculta todos los iconos inicialmente
    iconElt.setTrait("display", "none");

    iconVector[i] = iconElt;
}
```

Ya tenemos la parte de los botones localizada, ahora lo que tenemos que hacer es decirle a la aplicación el tamaño y el área que ocupan tanto los botones como el fondo.
Antes que nada hay que tener en cuenta que cuando utilicemos áreas de un gráfico svg las coordenadas de estas cajas hay que adaptarlas a las coordenadas que maneja Java.
Para sacar los puntos exactos de cada **box** y aplicarlos a las coordenadas de Java tenemos que hacer una serie de cambios y trabajar con matrices de transformación.
Dichas matrices son calculadas por Java y lo único que tenemos que hacer es aplicar los resultados a los gráficos que queremos mostrar por pantalla.
Cuando ya tengamos ajustada las nuevas coordenadas de cada tenemos que volver a convertirlas en coordenadas SVG solo que ahora estarán ajustadas a la pantalla

CODIGO

```
SVGLocatableElement icon = (SVGLocatableElement)iconVector[i];
SVGRect bbox = icon.getBBox();

// icon -> svg -> screen
SVGMatrix iconCTM = icon.getScreenCTM();

// icon -> svg
SVGMatrix iconToSvg =
svg.createSVGMatrixComponents(svgICTM.getComponent(0), svgICTM.getComponent(1),
                              svgICTM.getComponent(2), svgICTM.getComponent(3),
                                svgICTM.getComponent(4), vgICTM.getComponent(5));
iconToSvg.mMultiply(iconCTM);
float x0 = bbox.getX();
float y0 = bbox.getY();
float x1 = x0 + bbox.getWidth();
float y1 = y0 + bbox.getHeight();
float[] pointsX = { x0, x0, x1, x1 };
float[] pointsY = { y0, y1, y0, y1 };
float minX = Float.MAX_VALUE;
float minY = Float.MAX_VALUE;
float maxX = -Float.MAX_VALUE;
float maxY = -Float.MAX_VALUE;
float a = iconToSvg.getComponent(0);
float b = iconToSvg.getComponent(1);
float c = iconToSvg.getComponent(2);
float d = iconToSvg.getComponent(3);
```

Con las áreas puestas en su sitio ya sólo falta dibujar dentro de ellas. Como ya hemos comentado, java puede dibujar primitivas básicas (líneas, círculos, cuadrados) y imágenes externas (jpeg, png, gif) sin ningún problema y complicación. Lo difícil es decirle que me redibuje un gráfico svg ya que no se trata de una imagen de mapa de bits ni nada por el estilo, se trata de un archivo plano de texto. Ante tal problema SVG de J2ME dispone de su propio renderizado, es decir, una función que llamada desde **paint()** redibuja en pantalla un gráfico svg.

CODIGO

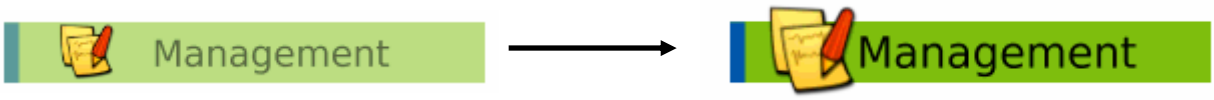
```
protected void paint(Graphics g)
{
    int fi = 0;

    if ( isRenderizado() == false )
    {
        sg.bindTarget(g);
        sg.render(0, 0, svgBg);
        sg.releaseTarget();

        setRenderizado(true);
    }
 . . .
 . . .
```

Por último queda la interacción de la interfaz y la animación del menú que es la parte más delicada. Todo sistema de renderizado necesita de un bucle que se repita (redibujar) una y otra vez hasta que el sistema termine o ya no sea necesario. Como estamos trabajando con gráficos svg y además queremos una cierta animación conviene que todo este trabajo se realice en un hilo aparte para evitar cuelgues y ralentizaciones inoportunas.

La nueva forma que adquirirán los botones cuando se activen simplemente será un pequeño escalado e iluminación.



La transición de la forma inicial a la final viene definida por una serie de parámetros privados incluidos en la clase **ScreenCanvas** que por defecto son: el número de frames que tendrá la animación, el tiempo entre frames, y el espaciado entre ellos.
Estos atributos los utilizaremos en el hilo para controlar el escalado y ver si se hace rápido, lento, a golpes o si se sobrepone encima de otro botón.

Para crear sensación de movimiento tenemos que usar la técnica de utilizar varias imágenes relacionadas y mostrarlas una detrás de otra con una cierta velocidad para engañar al ojo humano y dar el efecto de movimiento. Los archivos svg incorporan etiquetas especiales para añadir animaciones a sus gráficos vectoriales, estas etiquetas son también clasificadas por el parseador de Java. Renderizar 16, 25 o 29 imágenes por segundo con svg en Java es prohibitivo y por eso tenemos que recurrir al renderizado básico de Java, es decir, redibujar imágenes simples. Usaremos un pequeño bucle de 16 ciclos (frames) para capturar imágenes svg y renderizarlas en variables tipo **Image** y guardarlas en un vector. Cuando tengamos

todas las imágenes haremos uso de un hilo de ejecución que será el encargado de recorrer el vector de imágenes y de redibujarlas (**repaint()**) dando el efecto como si el botón se agrandara o menguara.

NOTA

La función de Canvas **repaint()** hace que se vuelva a llamar a la función **paint(Graphics g).**

CODIGO

```
for (int fi = 0; fi < numFrames; fi++)
{
    menuIcons[ri][fi] =  Image.createImage(midlet.getDataMain().getResMobileWidth(),
                                    midlet.getDataMain().getResMobileHeight()*50)/320);

    Graphics g = menuIcons[ri][fi].getGraphics();

    g.setColor(255, 255, 255);
    g.fillRect(0, 0, midlet.getDataMain().getResMobileWidth(),
              (midlet.getDataMain().getResMobileHeight() * 40)/320);

    sg2.bindTarget(g);
    sg2.render(0, 0, svgMenu);
    sg2.releaseTarget();

    svgMenu.incrementTime(frameLength);
}
```

Por último añadiremos interacción del usuario usando los cursores del móvil para desplazarnos por el menú y seleccionar opciones. El hilo del redibujado contiene una variable auxiliar llamada **needUpdate** que, como su nombre indica, será la encargada de decirnos si el usuario se está moviendo por el menú y por lo tanto necesitaremos actualizar la pantalla. Si el usuario no hace nada la variable se encontrará a "*false*" y evitaremos volver a dibujar, a pesar de que el bucle siga ejecutándose.

NOTA

El hecho de usar variables booleanas para controlar el redibujado de gráficos, y más todavía si se tratan de gráficos animados svg, es porque no existe una continuidad de interacción por parte del usuario. Si hubiéramos utilizado una interfaz donde los elementos están en continuo cambio la utilización de estos flags podría ocasionar que viéramos pararpadeos o sensación de lentitud en algunas animaciones.

CODIGO

```
public void createThread()
{
    final long frameLengthMs = (long)(frameLength * 1000);

    setTh(new Thread()
    {
         boolean ngc = true;
         public void run()
         {
             isThread = true;
             long start = 0;
             long end = 0;
             long sleep = 0;

             while (!interrupted)
             {
                 start = System.currentTimeMillis();
                 int cr = focusRow;
                 boolean needUpdate = false;

                 for (int ri = 0; ri < getNumOptions(); ri++)
                 {
                     int curFrame = currentFrame[ri];

                     if (cr == ri)
                     {
                         if (curFrame < (numFrames-1))
                         {
                             curFrame += 1;
                             needUpdate = true;
                         }
        . . . . . . .
        . . . . . . .
```

En resumen, el dibujado de gráficos SVG animados en J2ME es una tarea compleja pero con resultados muy vistosos y muy adaptables para cualquier tamaño de pantalla. Todo este proceso que parece tan complejo se puede ver en la **Figura 5.7** donde la idea queda muy clara.

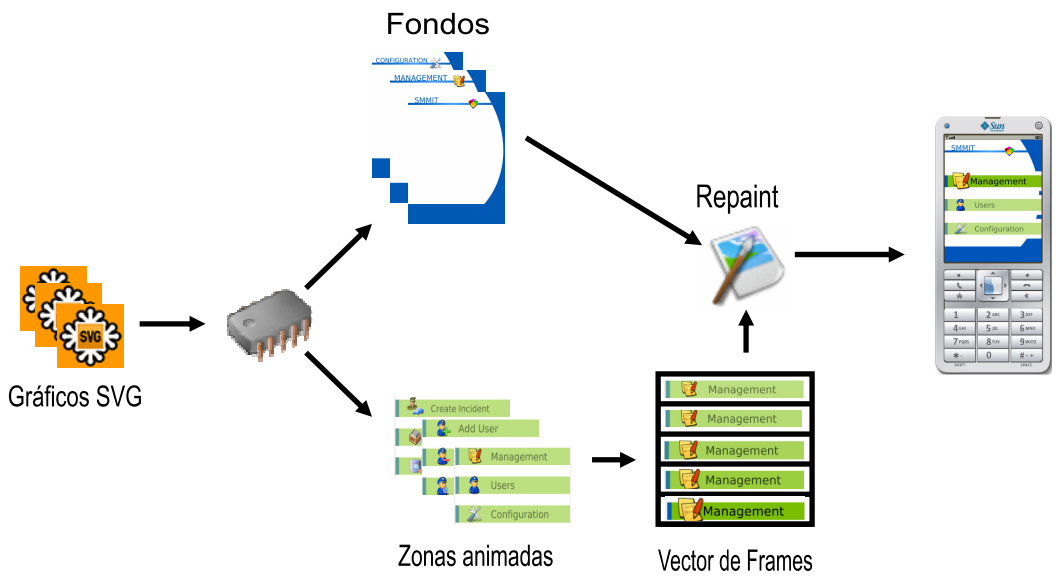


Figura 5.7 Pasos para crear una interfaz interactiva con SVG

Carga en segundo plano de las imágenes de fondo e interfaces SVG

El tratamiento de gráficos SVG en dispositivos móviles mediante Java es una tarea que requiere una supervisión extra de los recursos que vamos gastando.
La interfaz consta de 5 pantallas de gráficos SVG animados que sumados no llegan a las 220 KB y 5 fondos de pantalla de imágenes SVG estáticas de apenas 50 KB. Son tamaños bastante reducidos para la cantidad de información que se maneja hoy en día pero el problema no está en el espacio que abarcan sino en la puesta en escena, es decir, en el redibujado en pantalla.

Para dibujar gráficos Java incorpora APIs encargadas del redibujado en pantalla, concretamente se hace uso de la clase **Canvas** utilizada comúnmente para la creación de videojuegos Java en móviles. Esta clase crea un hilo de ejecución donde se llama a la rutina **paint()** que es la que realmente se encarga de hacer el redibujado.
Dicho redibujado está optimizado para mostrar por pantalla primitivas gráficas y algún que otro componente relacionado con los juegos (capas, sprites, movimiento de dibujos, etc.) pero no lo está tanto a la hora de pintar gráficos SVG.

El renderizado Java de gráficos vectoriales es lento si no se vigila la carga de los mismos, es por ello que tenemos que repartir el trabajo y empezar a cargarlos en el inicio de la aplicación.

Por lo tanto una de las primeras tareas del proceso principal será leer todos los archivos SVG y cargarlos en memoria para su rápida intervención.

CODIGO

```
private void createInterfaces()
{
  System.out.println(" ### - Creando el vector de memoria de pantallas...");

   try
   {
       for ( int i = 0; i < getSmmitComponents().getTableScreens().length; i++ )
       {
        MenuSVG menu = new MenuSVG();
        canvasMemory[i] = new ScreenCanvas(this);


        menu.setIdMenu(getSmmitComponents().getDescriptionMenu((String)getSmmitComponents
        ().getTableScreens()[i]));

        menu.setNumButtons(getSmmitComponents().getOptionsMenu((String)getSmmitComponents
        ().getTableScreens()[i]));

        menu.setSvgFile(getSmmitComponents().getUrlMenus((String)getSmmitComponents().get
        TableScreens()[i]));

         canvasMemory[i].setMenuSVG(menu);
     ......................................
     ......................................
```

5.2.1.3 Transiciones

Cambio de pantallas y menús

La gestión de pantallas que dispone J2ME es bastante sencilla ya que únicamente se basa en dibujar la imagen activa en ese momento.
Las clases básicas que se encargan de esta tarea las podemos encontrar en el paquete **javax.microedition.lcdui**

Display.getDisplay(this).setCurrent(canvasRT);

De esta forma podemos visualizar, en cualquier instante, las pantallas que tengamos cargadas en la memoria.

Para cambiar de una forma ordenada la información en pantalla se crean dos funciones, una para el cambio de formularios y la otra para los menús SVG.
A continuación se muestra la función encargada de sustituir un menú SVG por otro.

CODIGO

```
public void changeOfScreen(String nextScreen)
{
     //Inicializamos la variable que contiene el pantallazo

     if ( canvasRT != null )
     {
         canvasRT.stopThread();
         canvasRT.setNullThread();
         canvasRT = null;
         System.gc();
     }

     canvasRT = new ScreenCanvas(this);

     for ( int i = 0; i < numScreens; i++)
     {
         if ( canvasMemory[i].getMenuSVG().getIdMenu().compareTo(nextScreen) == 0 )
         {
              canvasRT = canvasMemory[i];
              canvasRT.processMenu(svgB[i], svgM[i]);
              canvasRT.startThread();
              canvasRT.setMidlet(this);
              canvasRT.setRenderizado(false);
              canvasRT.setFullScreenMode(true);

              Display.getDisplay(this).setCurrent(canvasRT);
              break;
         }
     }
  }
```

Vemos en el código anterior que, aun cambiando de pantallas, tenemos un control exhaustivo de la memoria que vamos gastando. Antes de efectuar el cambio paramos el hilo de ejecución del menú actual y liberamos memoria apuntando a NULL y ejecutando el ***Collector Garbage.***

NOTA

El **Collector Garbage** o recolector de basura se encarga de liberar memoria de aquellos recursos que no la necesiten evitando que este trabajo sea realizado por el mismo programador.

Lo mismo pasa con el cambio de los formularios, la rutina recoge el nombre del formulario que será mostrado en pantalla.

CODIGO

```
public void changeOfForm ( String nextForm )
{
     //Limpiamos la pantalla Screen SVG anterior
     canvasRT.stopThread();
     canvasRT.setNullThread();
     canvasRT = null;
     System.gc();

     if ( nextForm.compareTo("FORM_SEND") == 0 )
     {
         //Creamos la cadena que vamos a enviar al servidor
         prepareReport();

         //Iniciamos la conexion con el servidor para que nos de una
         //identificacion para el incidente que vamos a mandar
         //initConexion();
         getMainConn().initConnectionJabber();
         getMainConn().beginConnection();
     }
     else if ( nextForm.compareTo("FORM_ADD_USER") == 0 )
     {
         formAddUsers = new FormAddUsers(this);
         Display.getDisplay(this).setCurrent(getFormAddUsers());
     }
    else if ( nextForm.compareTo("FORM_REMOVE_USER") == 0 )
     {
         formRemoveUsers = new FormRemoveUsers(this);
         Display.getDisplay(this).setCurrent(getFormRemoveUsers());
     }
    . . .
    . . .
```

Al igual que ocurría antes tenemos que parar el hilo de ejecución del menú SVG e intentar liberar memoria.

5.2.1.4 Control de la Aplicación

Controles de la aplicación

La interacción en la aplicación se realiza con el propio teclado incorporado en el móvil no utilizaremos ningún otro medio para movernos por la interfaz.
Hacer que se capturen eventos de teclado es trivial en J2ME ya que cualquier Midlet lleva incorporado automáticamente el **Listener** del teclado para su rápida incorporación.
Lo único que tenemos que programar es qué hará cada tecla pulsada pero esta tarea está también implementada si nos ayudamos de la clase **Canvas** que dispone de funciones de manejo del teclado. Concretamente se dispone de la función ***KeyPressed(int KeyCode)*** que, como su nombre indica, será la encargada de asignar a cada tecla pulsada una acción.

Para el manejo de las interfaces de menú y todos los formularios se hará uso del JoyStick presente en todos los móviles de última generación, dicho mando está presente en la implementación J2ME por lo que no tenemos que averiguar en cada móvil con quien se corresponde.

NOTA

Cuando nos referimos al JoyStick estamos hablando de las teclas típicas de movimiento horizontal/vertical, izquierda/derecha. Además casi todos los móviles disponen de otro botón de acción usado para la selección de elementos.

CODIGO

```
public void keyPressed(int keyCode)
{
     int r = focusRow;

     switch (getGameAction(keyCode))
     {
         case UP:
             r--;

             if (r < 0)
                 r = getNumOptions() - 1;

             break;

         case DOWN:
             r++;

             if (r == getNumOptions())
                 r = 0;

             break;

         case FIRE: //Centrol del Joystick

            if ( r == 0 && this.getMenuSVG().getIdMenu().compareTo("Main Screen") == 0 )
                   midlet.changeOfScreen("Management Screen");
```

5.2.1.5 Multimedia

Captura de fotografías

La captura de fotografías es una de las tareas más importantes a la hora de gestionar un incidente de tráfico pues todo lo que no se puede explicar con palabras se podrá ver en la instantánea.
Manipular los elementos multimedia tales como las fotografías, el video y el sonido desde Java se consigue con la API **JSR-135,** también conocida como **MMAPI** (Mobile Media API), la cual provoca una gran expectación entre los programadores de móviles porque se puede sacar mucho partido y se pueden hacer virguerías. En la especificación de esta interfaz existen multitud de funciones e información, de hecho existen muchos manuales y documentos de referencias dedicados exclusivamente a dicho API (dichas referencias pueden consultarse en la sección de Bibliografía).

El paquete responsable es el **javax.microedition.media** y utilizaremos las clases principales de **Player**, **VideoControl**, **RecordControl** y **Manager.** Para entender el funcionamiento cada clase tiene su propio mecanismo pero sin embargo todas están relacionadas entre sí.

Cuando se crea una instancia de **Player** está tiene un ciclo de vida (**Figura 5.8**) por donde va pasando entre varios estados.

Figura 5.8 Fases por las que tiene que pasar la instancia Player para lanzar la cámara del móvil

El primer estado (**Unrealized**) es el estado inicial que se crea cuando instanciamos un objeto de la clase Player y luego invocamos al método *createPlayer().* Es un estado que no hace nada, simplemente porque aun no tiene la información necesaria para activar la cámara digital. Es necesario pues pasar a un siguiente estado (**Realized**) para obtener información del dispositivo donde se está ejecutando y qué buffers de memoria se le asignarán para adquirir el contenido multimedia. Cuando tengamos la información necesaria para seguir pasaremos al siguiente estado (**Prefetched**) que es una fase temporal de preparación para activar la cámara en el móvil y ver las imágenes que captura. En esta parte se decodifica las imágenes capturas en el dispositivo antes de ser visualizadas en la pantalla, esta es una medida para que no se produzca una latencia entre lo que me vemos y lo que capturamos. Ahora que tenemos la imagen capturada en memoria es momento de visualizarla, trabajo que realiza el siguiente paso (**Started**). Aquí ya vemos en la pantalla lo que observamos en la realidad y ya luego se procede a capturar las imágenes.
El ciclo de vida se cierra cuando paramos el *PlayBack* y vamos al último paso (**Closed**).

Aunque este ciclo tenga bastante lógica, a veces, y si nuestro móvil lo permite, podemos obviar algunos estados (unrelized y prefetched) y pasar inmediatamente al siguiente, ejecutándose en un segundo plano estos pasos saltados o incluso no invocarlos.
Para trabajar con las imágenes que estamos obteniendo en tiempo real se hace uso del control de interfaz **VideoControl** que será el encargado de manejarlos.
Esta interfaz puede controlar una variedad muy extensa de protocolos multimedia tanto de imágenes y video como de sonido.

Ya sólo nos falta especificar cómo mostraremos las imágenes recibidas por la cámara digital en la pantalla del móvil. Anteriormente explicamos que existen tres formas de mostrar información por pantalla y una de ellas no tenía restricciones, la clase **Canvas**. Cabe decir que lo podríamos poner en un formulario simple como si se tratasen de imágenes que se van actualizando pero el tamaño del marco quedaría muy reducido.

Uniéndolo todo parece que el código a implementar será muy extenso y complejo sin embargo todo viene reducido a unas pocas líneas.

CODIGO

```
videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);

 try
 {
     videoControl.setDisplayLocation(0, 0);
     videoControl.setDisplaySize(getWidth(), getHeight());
 }
 catch (MediaException me) {}

 videoControl.setVisible(true);

 try
 {
     setPlayer(Manager.createPlayer("capture://video"));
     getPlayer().realize();

     videoControl = (VideoControl) (getPlayer().getControl("VideoControl"));
     if (videoControl != null)
     {
         canvas = new DisplayCamera(midlet, videoControl);
         getPlayer().start();
     }
  }
```

De esta forma tan fácil hemos puesto en marcha el aparato de video de nuestro móvil y ahora ya sólo nos falta coger fotografías.

Dentro de **VideoControl** existe un método que permite tomar capturas de las imágenes que se están reproduciendo en la pantalla del dispositivo. Este método devuelve los datos de la imagen en un *vector de bytes* que podremos manejar a nuestro antojo usando la clase **Image** permitida por J2ME. La forma con la que queremos capturar la imagen, resolución y formato, hay que especificarla previamente y tiene que estar tolerada por el móvil si no nos lanzará una excepción de medios no soportado.

A modo de ejemplo el emulador utilizado Wireless Toolkit 2.3.2 soporta la captura de imágenes tipo jpeg con una resolución de 120x160 píxeles.

jpeg&width=120&height=160

La implementación no tiene dificultad sólo que hay que volver a tener cuidado con no bloquear el móvil e introducir el código de captura dentro de un hilo de ejecución:

CODIGO

```
 class createPhoto extends Thread
 {
     public void run()
     {
         try
         {
             System.out.println(" ### - Capturando la foto...");
            imageCaptured = videoControl.getSnapshot("encoding=jpeg&width=" +
midlet.getDataMain().getCurrentPhotoResWidth() + "&height=" +
midlet.getDataMain().getCurrentPhotoResHeight());
             imagesCaptured.addElement(imageCaptured);
             System.out.println(" ### - Foto capturada con exito ");

             //Guardamos la fecha de la captura de la foto
             datesOfPhotos.addElement(new Date().toString());
             System.out.println(" ### - Fecha creada y guardada correctamente");
         }
         catch (MediaException me)
         {
             System.out.println(" ??? - Error capturando foto ");
         }
     }
  }
```

Lógicamente se espera que el emulador disponga de una especie de emulación secundaria que represente la ejecución de una cámara digital para hacer las pruebas. Para ello Wireless Toolkit dispone de los medios necesarios para lanzar, cuando activemos la cámara, un pequeño video de muestra que hará el papel de una escena que estamos observando en un momento dado (**Figura 5.9**). Dicho video es lanzando automáticamente por el propio emulador y nosotros no tenemos que preocuparnos por nada porque se lanza en un hilo aparte.

Figura 5.9 Imágenes Reproducidas por el emulador de SUN simulando la cámara de un móvil

Grabación de Video

La grabación de video es un poco más delicada que capturar fotos aunque el sistema es muy parecido. El proceso de activar la cámara y mostrarla en pantalla es el mismo lo único que cambia es la captura de imágenes que ahora tiene que ser continua.
Como es de esperar no podemos hacer un bucle de 12, 25, o 29 capturas de imágenes para conseguir un segundo de video, tenemos que recurrir a los métodos que nos ofrece MMAPI.
En esta ocasión vamos a utilizar el **RecordControl** que es una interfaz de control del proceso de grabación de imágenes en tiempo real. Esta internaz tiene los controles básicos de captura de video como el *startRecord()*, *pauseRecord()* y *stopRecord()*.
Lo que nos devuelve esta interfaz es un vector de bytes, como se esperaba, y lo debemos volcar a una variable del tipo **ByteArrayOutputStream** para su posterior manipulación.

Las pruebas en esta versión del emulador son imposibles porque, como hemos dicho, se lanza en un segundo plano un hilo con la reproducción de un video de muestra y por motivos de implementación del emulador no podemos reproducir el video y grabarlo al mismo tiempo.
Podríamos utilizar el emulador de Nokia pero éste ni siquiera dispone de un video de muestra que simule una cámara digital.
En esta situación no hay más remedio que hacer las pruebas en el propio móvil y ver si realmente graba lo que se está viendo.
Hay que tener en cuenta que la grabación de video está soportada en muy pocos móviles y aun así en alguno de ellos no se comporta como debería (1 min. de grabación, formatos propios de móviles, etc.).

5.2.1.6 Gestión de Informe Multimedia

Creación del Informe Multimedia

Todo el trabajo perpetrado por la aplicación tiene que quedar reflejado en un informe multimedia para mandarlo a un servidor específico.

La estructura del informe tiene que ser como la de la **Tabla 5.1**

ID Incidente	110783
Día Registro	10/01/2008
Hora Registro	22:10

Agente de Servicio	
Identificación	AG12345
Nombre	Germán
Apellidos	Granero Payás

Localización			
Tipo de Vía	Autopista	**Nº de Vía**	A7
Provincia	Valencia		
Kilometro	100		
GPS	39.5º N - 0.112º E		

Datos del Incidente			
Tipo	Accidente de Moto	**Hora Incidente**	21:45
Vehículos	1	**Comp Meteorológico**	No
Heridos	Si	**Vehículo Removible**	Si
Obstáculo Carril	Si	**Personas atrapadas**	No
Vehículo en Cuneta	No		

Factores de Riesgo	
Fuego	No
Sustancias Peligrosas	No

Elementos Multimedia	
Fotografías	2
Videos	1

Fotografías			
Foto 1	22:10	**Resolución**	240x120
Foto 2	22:11	**Resolución**	240x120

Tabla 5.4 Informe Multimedia recogido por el móvil

5.2.1.7 Gestión de Registros, Archivos y RMS

Almacenamiento de datos RMS

Hasta ahora hemos usado la memoria principal del móvil (la RAM) para almacenar datos temporales, pero al salir del midlet, estos datos son eliminados. Esto plantea algunos problemas. Por ejemplo, ¿cómo podemos almacenar pequeños grupos de datos sin que se pierdan?
Este problema tiene dos posibles soluciones: guardar datos en un archivo creado en el móvil en tiempo real o usar un pequeño sistema de base de datos llamado **RMS (Record Management System).**

La necesidad de poder almacenar algún dato en el móvil surge cuando el sistema puede ser usado por más de un usuario. La aplicación contiene una gestión simple de usuarios que se pueden validar y eliminar pero si no se guarda un registro cada vez que se inicie la aplicación tendríamos que crear el nuevo usuario. Esta tarea es minúscula para utilizar ficheros y se ha optado por utilizar el método RMS.

La API que gestiona este sistema es la JSR-75 y contiene todo lo necesario para su manipulación, desde crear una nueva base datos hasta insertar o eliminar registros de forma sencilla. No obstante se ha creado una clase llama **RMSData** con rutinas básicas (openRMS(), closeRMS(), sizeRMS(), addDataRMS()…) para controlar las excepciones que se produzcan.

CODIGO

```
public void openRMS()
{
    try
    {
        rs = RecordStore.openRecordStore("LIST OF USERS", true);
    }
    catch (RecordStoreException ex)
    {
        System.out.println(" ??? - Error abriendo la Base de Datos RMS");
        ex.printStackTrace();
    }
}

public void closeRMS()
{
    try
    {
        rs.closeRecordStore();
    }
    catch (RecordStoreNotOpenException ex)
    {
        System.out.println(" ??? - No se ha podido cerrar porque no estaba abierta");
        ex.printStackTrace();
    }
    catch (RecordStoreException ex)
    {
        System.out.println(" ??? - Error cerrando la Base de Datos RMS");
        ex.printStackTrace();
    }
}
```

CODIGO

```
public void addDataRMS( String data )
{
    //Abrimos RMS
    openRMS();

    byte[] dat = data.getBytes();

    try
    {
         if ( rs.getNumRecords() != 0 )
         {
             for ( int i = 0; i < rs.getNumRecords(); i++ )
             {
               try
               {
               String tmpA = new String(rs.getRecord(i+1), 0, rs.getRecord(i+1).length);
               String tmpB = new String(dat, 0, dat.length);

                 if ( tmpA.compareTo(tmpB) == 0 )
                 {
                     rs.setRecord(i+1, dat, 0, dat.length);
                 }
                 else
                 {
                     rs.addRecord(dat, 0, dat.length);
                     break;
                 }
             }
             catch (RecordStoreNotOpenException ex)
             {
                 System.out.println(" ??? - Error comparando, RMS no abierta ");
                 ex.printStackTrace();
             }
             catch (InvalidRecordIDException ex)
             {
                  System.out.println(" ??? - Error comparando, Record invalido");
                  ex.printStackTrace();
             }
             catch (RecordStoreException ex)
             {
                 System.out.println(" ??? - Error comparando, Record mal guardado");
                 ex.printStackTrace();
             }
         }
       }
       else
       {
           try
           {
               rs.addRecord(dat, 0, dat.length);
           }
           catch (RecordStoreNotOpenException ex) { ex.printStackTrace(); }
           catch (RecordStoreException ex) { ex.printStackTrace(); }
       }
     }
     catch (RecordStoreNotOpenException ex)
     {
         System.out.println(" ??? - Error guardando, RMS no abierta");
         ex.printStackTrace();
     }

     //Cerramos RMS
    closeRMS();
}
```

Almacenamiento de datos en archivos físicos

Si anteriormente hemos usado el sistema RMS para gestionar una pequeña base de datos de usuarios ahora vamos a utilizar los otros métodos también disponibles en la interfaz JSR-75 y que son exclusivamente para la creación, modificación y borrado de ficheros lógicos en la memoria física del móvil.

Crearemos una clase **MFile** que implementará los métodos de la interfaz **FileSystemListener** encargada de manipular la estructura del árbol de directorios y ficheros del móvil.

N O T A

Todas las tareas de I/O en J2ME deben de ser en segundo plano porque este proceso es muy delicado y bloquea al móvil, de hecho, **FileSystemListener** implementa obligatoriamente la interfaz **Runnable**.

La jerarquía de archivos en un móvil es similar a la de cualquier Sistema Operativo, se cuenta con un nodo inicial que es la "raíz" y de aquí van surgiendo distintas "ramas" que contienen ficheros u otros directos que contiene otras ramas y así sucesivamente como muestra la **Figura 5.10**.

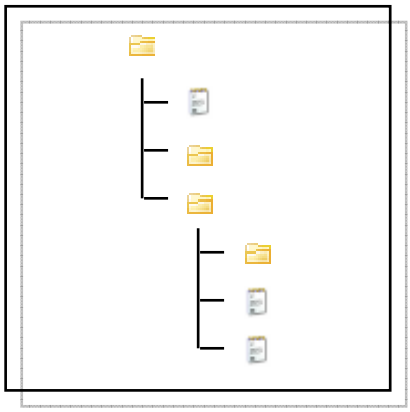


Figura 5.10 Estructura típica de árbol de directorios y archivos

Cada nivel está separado por un separador que puede variar según el sistema de ficheros que se esté utilizando siendo en nuestro caso el símbolo "/" quien delimitará los niveles.

Las operaciones sobre ficheros que vamos a hacer serán:

Carga del árbol de ficheros

La clase **FileSystemRegistry** dispone de un método muy útil llamado *listRoots()* que proporciona de una forma estructurada todas las raíces del árbol de directorios de la memoria interna del móvil.

NOTA

En la mayoría de móviles que disponen de esta API el árbol de directorios que se extrae contiene directorios ocultos por los fabricantes para evitar su manipulación ya que se si hace un mal uso de ellos el firmware podría resultar inservible.

La lista de directorios la guardamos en una variable *Enumeration* proporcionada por Java que hace la función de un *Iterator* recorriendo un vector de elementos. Para mostrar por pantalla las diferentes raíces guardamos los elementos del iterador en una cadena de caracteres.

CODIGO

```
 private void loadRoots()
{
    // Borramos los elementos que tuviese previamente
    roots.removeAllElements();

    Enumeration roots = FileSystemRegistry.listRoots();
    String root  = null;

    while (roots.hasMoreElements())
    {
        root  = (String) roots.nextElement();
        root  = FILE_SEPARATOR + root;
        this.roots.addElement(root);
    }
}
```

Abrir directorios o ficheros

La acción de abrir dependerá de si lo que queremos abrir se trata de un fichero o de una nueva carpeta. Para ello primero tenemos que tener claro que símbolo de separación entre elementos (ficheros y carpetas) existe, es decir, para algunos sistemas se una el "/" para otros el "\\" y para la mayoría "//". Una vez identificado el símbolo que hará de guía comprobaremos si lo que hemos seleccionado se trata de un fichero, una carpeta o la propia raíz para volver hacia atrás. La siguiente función realiza las tres acciones comentadas.

CODIGO

```
private void openSelected() throws java.io.IOException
{
    int selectedIdx  = this.getSelectedIndex();
    String url = null;
    String selectedFile = null;

    if (selectedIdx < 0)
        return;

    selectedFile = this.getString(selectedIdx);

    // ¿ Desea abrir un directorio?
    if (selectedFile.endsWith(FILE_SEPARATOR) || selectedFile.endsWith("/"))
    {
        if (currentRoot == null)
            currentRoot = (FileConnection) Connector.open("file:///" + selectedFile,
Connector.READ);
        else
            currentRoot.setFileConnection(selectedFile);

        displayCurrentRoot();
        return;
    }

    // ¿ Desea ir al directorio anterior?
    if (selectedFile.equals(UPPER_DIR))
    {
        this.currentRoot.setFileConnection(UPPER_DIR);
        this.displayCurrentRoot();

        return;
    }

    // Es un fichero, lo mostramos.
    url = currentRoot.getURL() + selectedFile;
    showTextFile(url);
}
```

Leer Ficheros

Como hemos visto anteriormente a la hora de abrir elementos de un árbol de directorios si lo que queremos abrir es un fichero llamamos a una función *showTextFile(String url)* que básicamente crea una variable tipo **DataInputStream** a la cual le asignamos el flujo de datos que contiene el archivo que queremos leer y luego llama a una función *readFully(input)* que transforma lo que leemos del archivo en un array de bytes para posteriormente mostrarlos por pantalla.

CODIGO

```
private void showTextFile(String url) throws java.io.IOException
 {
     FileConnection  conn  = null;
     DataInputStream input = null;
     byte[] bytes = null;

     try
     {
         conn  = (FileConnection) Connector.open(url, Connector.READ);
         input = conn.openDataInputStream();
         bytes = readFully(input);

         deleteAll();

         this.append(new String(bytes), null);
         conn.close();
     }
     finally
     {
         if (conn != null)
            conn.close();
     }
 }
```

CODIGO

```
 public byte[] readFully(DataInputStream input) throws java.io.IOException
 {
     ByteArrayOutputStream bo = null;

     try
     {
         bo = new ByteArrayOutputStream(); // 6Kb

         while (true)
             bo.write(input.readByte());
     }
     catch (java.io.IOException ex){}

     byte[] bytes = bo.toByteArray();

     // Liberamos recursos
     try { bo.close(); } catch (Exception ex){}
     try { input.close(); } catch (Exception ex){}

     return bytes;
 }
```

Escribir Ficheros

Crear un nuevo fichero de texto requiere la utilización extra de otro hilo de ejecución encargado de abrir el sitio donde irá e implantar el archivo que queremos.
A la hora de manipular un fichero tenemos dos opciones: la primera es que el archivo no exista por lo que tendremos que crearlo desde cero y rellenarlo con los datos que le pasemos y la segunda que el archivo ya exista y por lo tanto añadamos más información a la que ya existe.
Utilizaremos las clases **Connector** y **FileConnection** en modo READ_WRITE para averiguar en qué estado nos encontramos y actuar en consecuencia creando un nuevo fichero *FileConnection.create()* o modificando uno *FileConnection.truncate(0).* A partir de aquí los dos modos siguen el mismo camino, es decir, abrimos un flujo de datos de entrada *OutputStream* y lo llenamos con los bytes que serán la información que queremos guardar para, por último lugar, escribirlos "*flush()"* en el fichero.
Cuando acabe todo el proceso hay que cerrar el flujo de datos y el **Connector** asociado al fichero.

CODIGO

```
public void writeFile (final String path, final byte[] data)
 {
     write = new Thread()
     {
         public void run()
         {
             Connection c = null;
             OutputStream os = null;
             try
             {
                 c = Connector.open(path, Connector.READ_WRITE);
                 FileConnection fc = (FileConnection) c;

                 if ( !fc.exists())
                     fc.create();
                 else
                     fc.truncate(0);

                 os = fc.openOutputStream();
                 os.write(data);
                 os.flush();

                    //return true;
                }
                catch ( Exception e ){ }
                finally
                {
                    try
                    {
                        if ( os != null )
                            os.close();
                        if ( c != null )
                            c.close();
                    }
                    catch ( Exception ex ){ ex.printStackTrace(); }
                }
            }
```

5.2.1.8 Componentes para la Conexión Jabber

Inicializar los componentes para la comunicación Jabber

Dedicaremos un apartado exclusivamente para explicar las conexiones Jabber y todos los elementos que intervienen pero conviene exponer previamente qué componentes están asociados a la clase principal **SMMidlet.**

La clase que se encarga de enlazar las comunicaciones Jabber con el sistema principal de la aplicación se llama **MainConnection** y contiene los siguientes atributos:

a) cm (**CommunicationManager**): básicamente es el administrador que gestiona el tipo de conexión que se va a realizar (http, TCP, SSL).

b) listener (**MidletEventListener**): Esta clase recoge todos los eventos producidos durante la comunicación desde que se inicia hasta que se cierra pasando por las etapas de enlace con el servidor, envío y recibimiento de información XMPP y desconexión por ambas partes.

c) midlet (**SMMidlet**): Núcleo principal del sistema.

Cuando se desee establecer una conexión la función *beginConnection()* llamará a **CommunicationManager** para empezar a crear los distintos elementos que componen Jabber y que se explicarán en un tema aparte.

5.2.2 Interfaz

5.2.2.1 Gráficos SVG

Unos de los requisitos que nos hemos impuesto es que la interfaz se adapte a cualquier tamaño de pantalla. Para ello J2ME dispone, en sus últimas actualizaciones, de unas APIs especiales dedicadas al manejo y diseño de gráficos escalares **SVG** idóneos para la finalidad que buscamos.

Los gráficos escalares, como su nombre bien indica, son gráficos que se pueden escalar al tamaño que necesitemos sin perder calidad. Esto es debido a que los gráficos SVG son gráficos compuestos por ecuaciones matemáticas y no por un mapa de bits como la mayoría de los formatos gráficos que conocemos. De esta forma si dibujamos, por ejemplo, un círculo y lo queremos ampliar simplemente modificaríamos las variables de la ecuación.

Por otro lado, los ficheros SVG son ficheros planos de texto, concretamente son archivos XML fácilmente editables, que los podemos modificar en cualquier momento para adaptar los gráficos a cualquier nueva funcionalidad. En nuestro proyecto esta ventaja se traduciría a quitar o añadir nuevos botones animados en los menús, simplemente abriríamos el archivo con un simple editor de texto y guiándonos por sus etiquetas modificaríamos lo que fuera necesario.

Características

- Tiene todas las ventajas asociadas a un formato vectorial: es escalable, compacto, con formas siempre editables a través de curvas Bézier, con contornos suavizados, transparencias, y capaz de incluir, si es preciso, bitmaps.
- El tamaño de los SVG es muy compacto.
- El texto que incluyen es editable: admite las fuentes escalables más comunes, como TrueType o Type 1. Esto supone una diferencia enorme con los actuales GIF o JPG: el texto que contienen se puede editar, seleccionar, ser indexado por los buscadores…
- La calidad de color es excelente; el color del gráfico se puede calibrar con los sistemas estándar de gestión de color.
- El fichero SVG no es binario: se trata de un fichero de texto normal y corriente. Esto significa que se puede editar incluso en el Bloc de notas, y sus contenidos se pueden indexar, buscar.
- Es compatible con los estándares actuales de la web y –lo que es más importante– con los futuros.
- Soporte de hojas de estilo CSS. Esto significa que con las hojas de estilo podemos modificar, de la forma más poderosa y fácil, ¡incluso los gráficos de las páginas web! El control y poder creativo que esto supone es insuperable.
- Será posible crear páginas con una riqueza tipográfica y de layout sin precedentes, sin sacrificar la accesibilidad del contenido escrito.
- Puede incluir código (scripts) que modifican el gráfico dinámicamente en función de las necesidades.
- Al ser XML, es un formato extensible: los fabricantes podrán adoptarlo como formato nativo de sus aplicaciones, añadiendo las características específicas que deseen, pero siempre mantendrá la compatibilidad básica y universal con toda aplicación que reconozca el formato.
- Admite efectos como sonido, efectos visuales al hacer clic o mover el ratón, etiquetas informativas.
- Puede generarse dinámicamente en un servidor web como respuesta a instrucciones de Java, JavaScript, Perl o XML. Por ejemplo, pueden crearse al momento gráficos de excelente calidad con las cotizaciones de bolsa en tiempo real; un reloj analógico, con minutos y segundos, requiere sólo 2K de código.
- SVG puede llegar a simplificar extraordinariamente el “workflow” para la web. En una aplicación única se podrá generar casi todo el contenido de las páginas, y convertirse en un formato universal: todos los programas podrán abrir todo tipo de ficheros. Los gráficos SVG no serán, como hasta ahora, una versión mutilada de un gráfico que ha pasado por varias aplicaciones. Se ahorraran conversiones, pasos de un programa a otro, tareas de optimizar, cambiar de tamaño…

Ejemplo de archivo SVG

CODIGO

```
<svg width="400" height="350" viewBox="0 0 400 350">
    <text x="108" y="160" style="font:italic 48pt Georgia">
        E x p o
    </text>

    <text x="108" y="230" style="font:italic 48pt Georgia">
        Vision
    </text>

    <rect x="90" y="100" width="225" height="150" rx="32" ry="64" fill="orange"/>
</svg>
```

Las APIs que necesita JAVA para manejar estos tipos de gráficos son la **JRS-226** para los gráficos SVG y la **JSR-172** para el parseado XML.

5.2.2.2 Formulario J2ME

J2ME nos proporciona una serie de elementos predefinidos exclusivos para dispositivos móviles que usaremos para programar todos los formularios presentes.
Estos elementos tienen la misma tónica que los usados en cualquier interfaz, estamos hablando de los campos de texto, botones, menús desplegables, casillas de verificación, etc.

La forma de utilizarlos es exactamente igual que los que solemos usar sólo que están adaptados a las pequeñas dimensiones que puede tener un dispositivo móvil.
Para incluirlos simplemente tenemos que crear una clase formulario **Form,** que sería el contenedor, y luego ir añadiendo elementos **append**.

Ejemplos de formularios disponibles en J2ME son los incluidos en la **Figura 5.11**

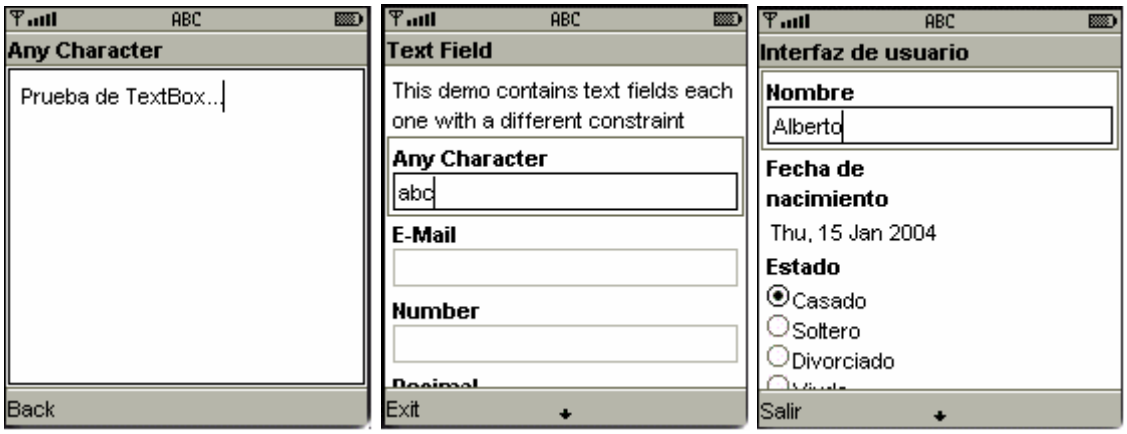
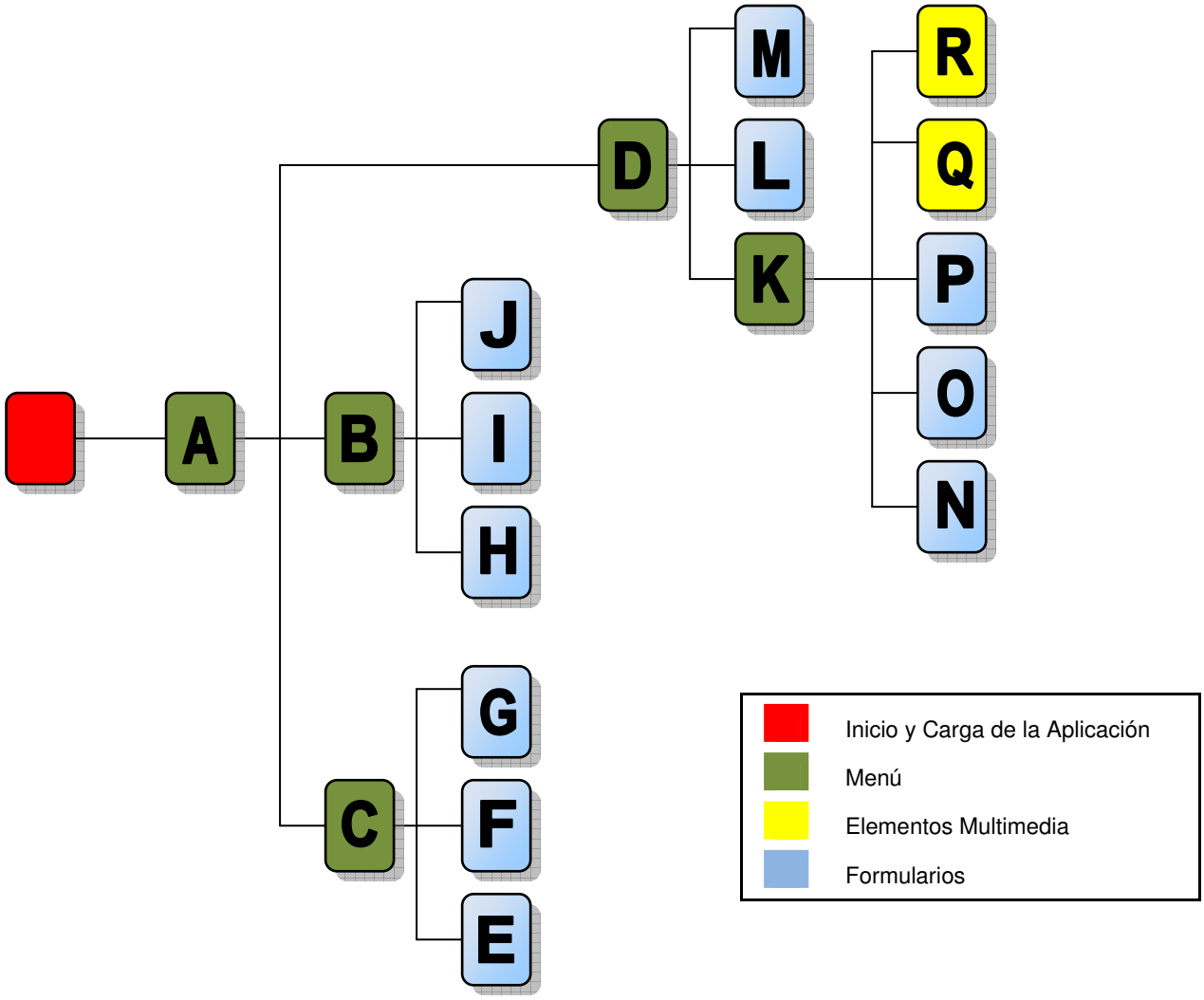


Figura 5.11 Formularios y Campos que J2ME tiene por defecto

5.2.2.3 Mapa de la Interfaz



A	Menú Principal	**J**	Añadir Usuarios
B	Menú Gestión de Usuarios	**K**	Menú Multimedia
C	Menú Configuración	**L**	Ver Archivos de Incidentes
D	Menú Gestión de Incidentes	**M**	Ver Histórico
E	Configuración de la aplicación	**N**	Enviar Informe
F	Configuración del móvil	**O**	Guardar Informe
G	Configuración de conexiones	**P**	Escribir Informe
H	Selección de usuario actual	**Q**	Grabar Video
I	Eliminar Usuarios	**R**	Capturar Fotografías

5.2.2.4 Capturas de Pantalla

Figura 5.12 Capturas de Pantalla de la Interfaz SMMIT

5.2.3 Comunicaciones Jabber

Como ya se iba comentando a lo largo de esta memoria la comunicación es la clave para el éxito de este proyecto, si falla el enlace entre el cliente y el servidor habremos perdido el tiempo inútilmente.
Desarrollar un sistema de comunicación propio para un teléfono móvil conlleva el estudio de las diferentes tecnologías que se usan para las comunicaciones inalámbricas y de qué herramientas disponemos para su elaboración. Dicho estudio se realizó anteriormente dando como resultado la utilización de un protocolo novedoso llamado **Jabber** acompañado por su protocolo de mensajería instantánea **XMPP.**

5.2.3.1 Elementos Básicos de Jabber

El principal objetivo que tenia Jabber era implantar un nuevo sistema de mensajería instantánea con la particularidad de que fuera en *Open Source* y totalmente gratuito para todo aquel que quisiera hacer uso de sus servicios. Al utilizarse XML como lenguaje de comunicación fueron surgiendo distintos tipos de clientes y servidores con un determinado factor común, el protocolo XMPP.
Todo este servicio estaba destinado a cualquier dispositivo ya fuera un ordenador de sobremesa o un portátil con conexión a internet pero al aparecer dispositivos móviles con estas características y añadiendo que cada vez son más potentes se está aplicando esta tecnología sobre ellos.

La utilización de Jabber en un dispositivo móvil es totalmente igual que si lo utilizáramos en un PC normal sólo que habría que adaptar la conexión que realiza el móvil con el nuevo protocolo.
Para entender el funcionamiento de Jabber en un móvil utilizando Java primero vamos a definir los elementos que lo constituyen.

5.2.3.1.1 Roster

Para no enviar los cambios de presencia entre todos los usuarios del sistema, Jabber ha creado el concepto de suscripción de presencia. Como su nombre lo indica, la suscripción de presencia determina los suscriptores que recibirán las actualizaciones de presencia de cada usuario. Los suscriptores piden una suscripción a un usuario, y el usuario acepta o deniega dicha suscripción. Cada usuario se suscribe a los usuarios que desea y puede aceptar que dichos usuarios u otros se suscriban a los cambios de su presencia. Para realizar esta tarea, Jabber ha definido unas estructuras de datos estándar conocidas como Jabber Roster, la cual, no es más que una lista de otros usuarios identificados por su Jabber ID.

5.2.3.1.2 Jid y Jud

Para estructurar de una forma ordenada el flujo de datos XML que enviamos por la red Jabber especifica dos tipos de estructuras llamadas JID (Identificador de usuario Jabber) y JUD (Identificación de sesión Jabber) y que nosotros las agruparemos en dos clases.

La clase JID identifica a un usuario Jabber y al servidor al cual se conecta, la estructura básica de esta clase se puede encontrar en el anexo de descripción de clases.

La clase JUD especifica la relación que hay entre el JID y el servidor Jabber asignando un identificador sesión. Es una clase que no tiene constructor, simplemente se utilizan las funciones que dispone para enviar información directamente al servidor y para recoger datos. La estructura básica de esta clase se puede encontrar en el anexo de descripción de clases.

5.2.3.1.3 XML Stanzas

La comunicación entre dos puntos Jabber, como por ejemplo un cliente y un servidor, se realiza mediante un socket TCP y un XML es transferido entre un punto y el otro. Este XML no tiene que estar fragmentado, tiene que tener una estructura clara y lógica que puedan entender las dos partes. Para ello se definió una estructura de etiquetas incluidas en un documento XML y que en XMPP se conocen como XML Stanzas.
XML Stanzas se define como una estructura de información discreta que se envía de una entidad a otra sobre un flujo de datos XML.
En el protocolo Jabber hay tres XML Stanzas definidos en XMPP y son: <message/>, <iq/>, <presence/> que serían los necesarios para establecer una comunicación simple entre dos puntos.

5.2.3.1.3.1 Protocolo Message

La mensajería es el núcleo de todo sistema que incorpore Jabber. El trabajo que realiza este protocolo es un simple envio y recepción de mensajes. Un ejemplo de utilización sería el siguiente.

```
<message to='romeo@example.net' from='juliet@example.com/balcony' type='normal'>
      <subject> Hello World! </subject>
      <body> Hello </body>
      <thread> hilo_01 </thread>
</message>
```

Este ejemplo muestra como un usuario, Juliet, envía un mensaje, a Romeo, usando como dominio Jabber *balcony*. La etiqueta <message> contiene una serie de atributos característicos usados cuando enviamos un email (from, to) y un tercer atributo llamado *type* que utiliza Jabber para describir el tipo de mensaje que se quiere enviar. Se pueden ver en la **Tabla 5.2** los tipos que soporta Jabber. En este ejemplo el mensaje sería del tipo Normal pero hay otros tipos que se utilizan para cuando la ocasión lo requiera:

Estilo del Mensaje	Tipo	Ejemplo	Interface
Normal	normal	mensaje tipo email	Editor de texto
Chat	chat	conversación online uno a uno	Línea por línea
Grupo chat	groupchat	conversación en un chatroom	Línea por línea
Cabecera	headline	mensaje scrolling	Stock Ticker
Error	error	mensaje de error	Dialogo Alerta

Tabla 5.5 Tipos de comunicación por texto de Jabber

La etiqueta <body> se usa habitualmente para estructurar el contenido de un mensaje y aunque se considera opcional se debería de utilizar dentro de <message>. Las otras dos etiquetas que quedan, <subject> y <thread> son totalmente opcionales.

5.2.3.1.3.2 Protocolo Presence

El protocolo **presence** es utilizado en dos contextos principales:

Presence update: que informa a los usuarios conectados en un servidor Jabber sobre el estado actual de cualquier usuario conectado.

Presence subscription management: permite a los usuarios de Jabber suscribir en su cuenta a otros usuarios Jabber y de esta forma controlar el estado de su lista de contactos.

Para el protocolo *presence update* se envía un simple mensaje. Un cliente envía una actualización de su presencia al servidor y éste informa a los demás usuarios conectados del estado actual de este cliente. Se puede entender este mecanismo como una estructura básica de multicast.
A continuación se muestra un ejemplo de *presence update.*

```
<presence from='juliet@example.com/balcony'>
      <show> away </show>
      <status> one momento please </status>
      <priority> 1 </priority>
</presence>
```

En esta ocasión el significado de este ejemplo sería que el usuario Juliet cambia su estado a **no disponible** y muestra un mensaje de su ausencia, la etiqueta prioridad significa que este mensaje se mandará con una prioridad alta para que el servidor actué enseguida y muestre el nuevo estado.
Los distintos tipos de estado que podemos adoptar son los mostrados en la **Tabla 5.3**:

Estado	Tipo
chat	Usuario en línea
away	no disponible
xa	ausente
dnd	no molestar

Tabla 5.6 Estados por defecto de presencia de un usuario

Para el protocolo de subscripción de un nuevo usuario se necesita la intervención del servidor ya que será este el que reenvíe la petición al usuario destino.

```
ENVIO:  <presence from='juliet@example.com/balcony' to='romeo@example.net'
type='subcribe'>
RECIBE: <presence from='romeo@example.net/orchard'
to='juliet@example.com/balcony' type='subscribed'>
```

Los tipos disponibles para señalar el tipo subscripción se explican en la **Tabla 5.4**:

Tipo Presence	Tipo Protocolo	Significado
unavailable	update	Usuario no disponible
subscribe	petición de subscripción	Se desea subscribir
subscribed	respuesta de subscripción	Subscripción aceptada
unsubscribe	petición de subscripción	Una entidad se no quiere subscribirse
unsubscribed	respuesta de subscripción	Petición denegada
error	Error estándar Jabber	error enviado el presence stanza
probe	petición de servidor	presence generada sólo por el servidor

Tabla 5.7 Tipos de Subscripciones

5.2.3.1.3.3 Protocolo IQ (Info/Query)

El tercer y último elemento básico de toda comunicación Jabber es la etiqueta </iq> que significa “información y consultas” y que básicamente consiste en la gestión y consulta de información.

Recoge un amplio conjunto de casos complejos de utilización como por ejemplo creando, actualizando y quitando cuentas de usuario, autenticación de usuarios y gestión del roster.
El roster no es más que una especie de identificador que contiene toda la información de un usuario como por ejemplo su JID, JUD, PRESENCE, etc.

El mecanismo de este protocolo es similar al HTTP donde hay dos principales participantes, el cliente y el servidor, donde el cliente pide información al servidor y éste se la envía.

La sintaxis básica de IQ es la siguiente:

```
<iq
       to='handler_jid'
            from='originator_jid'
      type='get|set|result|error'
      id='unique'>
      <query xmls='iq-extension.namespace'>
            <query-field1/><query-field2/>
      </query>
</iq>
```

El atributo que parece más importante es el tipo "*type*" que nos indica el formato del paquete a enviar. En la **Tabla 5.5** vemos una relación de los distintos tipos y su descripción.

IQ Type	Dirección	Descripción
get	requester-to-responder	Petición de información o requerimientos
set	petición de subscripción	Aplicamos o reemplazamos valores
result	responder-to-requester	Respuesta a una exitosa petición
error	petición de subscripción	Error en la petición o respuesta

Tabla 5.8 Tipos de Información y Consulta

5.2.3.2 Establecimiento de la Conexión Jabber

Ya tenemos definidos todos los elementos básicos que constituyen Jabber, ahora es el turno de adaptarlos a la aplicación y al dispositivo que lo contiene.
Para empezar hay que decir que todo el núcleo Jabber que vamos a utilizar no está programado desde cero, está sacado de una aplicación base muy sencilla llamada **MicroJabber** que está disponible en código abierto

a la cual hemos añadido algunos métodos prácticos de otra aplicación de libre distribución llamada **JMC Jabber** creada por Gabriele Bianchi (En la sección de Bibliografía se podrá encontrar una referencia a este autor).
Basándonos en esta estructura inicial hemos ido modificando el módulo de conexión quitando todos los métodos que no necesitamos y quedándonos con las funciones que realizan el parseado de los documentos XML. Además, en este proyecto no vamos a desarrollar una comunicación continua entre un cliente y un servidor, simplemente queremos que el cliente se conecte al servidor y le vaya mandando información y que la participación del servidor sólo se requiera para obtener la identificación de los informes y el registro de usuarios.

También hay que decir que, a pesar de utilizar XML, y por lo tanto poder crear nuestras propias etiquetas, vamos a utilizar los tipos básicos de Jabber y para el envío de información se usará el tipo "chat" aunque, como ya hemos dicho, el presente proyecto no se basa en una conversación cliente-cliente. De esta forma nos adaptaremos a cualquier servidor Jabber disponible ya que si modificáramos alguna etiqueta tendríamos que tenerla en cuenta en el otro lado.

5.2.3.2.1 Tipos de Conexión

La conexión que se establece con Jabber es la misma para cualquier sistema que la utilice como medio de comunicación. Tenemos dos formas de conectarnos, mediante http que es la forma más habitual y por TCP que sería establecer un enlace de bajo nivel.

5.2.3.2.1.1 Conexión HTTP

Para este tipo de conexión **CommunicationManager** crea las instancias adecuadas y un hilo de ejecución encargado de iniciar, autenticar y establecer la conexión. Dicho hilo se ejecuta en un segundo plano para no entorpecer la ejecución del hilo principal y de esta forma continuar la ejecución de la aplicación por si hubiera excepciones.
La secuencia que sigue este hilo de conexión claramente se puede ver en los *diagramas de secuencia* mostrados anteriormente en el apartado de diseño y por lo tanto sólo comentaremos que es lo que hace cada parte.

La primera fase consiste en iniciar una sesión con el servidor Jabber, para ello necesitamos la dirección y el nombre de la máquina destino y el usuario que deberá estar registrado en la cuenta Jabber del servidor.

CODIGO

```
protected void initSession(String addr, String domain, String user,
                    String pass, String resource, String Status) throws Exception
    {
        try
        {

            System.out.println("Opening first stream");

            generateRequestId(); // create rid

            writeStream("<body content=\"text/xml; charset=utf-8\" to=\""
                            + domain
                            + "\" hold=\"1\" wait=\""
                            + wait
                            + "\" rid=\""
                            + rid
                            + "\" "
                            + "xml:lang=\"en\" "
                            + ""
                            + "route=\"xmpp:" + addr + "\" "
                            + "xmlns=\"http://jabber.org/protocol/httpbind\" "
                            +"/>", 0);

            HttpNode x = readResponse(0);

        . . . . . . .
        . . . . . . .
```

No se ha mostrado todo el código de *initSession(…)* sólo la parte más interesante que es el primero flujo de datos que vamos a enviar al servidor y que consta de una serie de parámetros tales como el dominio de la conexión, un identificador “rid” aleatorio de petición de conexión, el tipo de texto que vamos a mandar, dirección del servidor y tipo de protocolo que estamos usando.
El servidor nos contestara con otro flujo de información para ver si hemos sido admitidos, si todo va bien nos devolverá otro identificador de sesión “sid” el cual utilizaremos para distinguir entre diferentes conexiones que puede sufrir el servidor. Además de la identificación el servidor también puede darnos datos sobre él, es decir, puede informarnos de sus características (métodos de codificación aceptados, protocolos admitidos, etc).

La segunda fase consiste en autenticar al usuario, para ello mandamos una cadena con el nombre de usuario y su contraseña. Si el servidor lo permite podemos establecer un codificado MD5 para proteger la cuenta de usuario mientras llega al servidor. Haremos uso de una librería de codificación de datos MD5 y Base64 diseñada para Java útil para esta ocasión.

CODIGO

```
void Authenticate(HttpNode x, String user, String pass, String  domain) throws Exception
  {
     if (x.child("mechanisms").hasValueOfChild("PLAIN"))
     {
         String resp = "\0" + user + "\0" + pass;
         writeWithBody("<auth xmlns=\"urn:ietf:params:xml:ns:xmpp-sasl\"
          mechanism=\"PLAIN\">" + MD5.toBase64(resp.getBytes()) + "</auth>");

          x = readStanza();
          if (x.getName().equals("failure"))
              throw new Exception("PLAIN authorization error");
      }
      else throw new Exception("Only PLAIN authorization supported");
  }
```

En la tercera y última fase y después de identificarnos en el servidor le enviaremos un último mensaje de sesión para registrarse como un usuario conectado. A partir de aquí habremos establecido una conexión satisfactoria con nuestro servidor y cada un cierto periodo de tiempo mandaremos un mensaje de reconocimiento para estar siempre actualizados.

El proceso de comunicación se puede resumir en la **Figura 5.13** donde un cliente establece una conexión con el servidor Jabber.

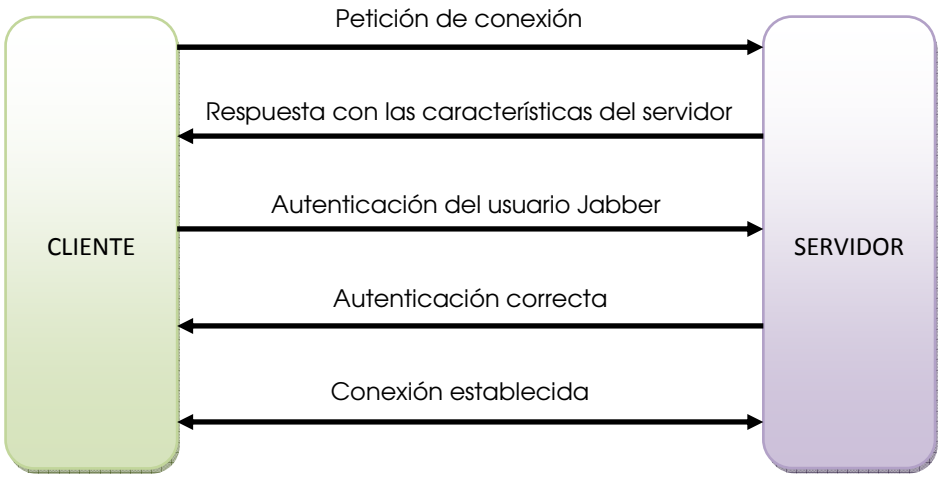


Figura 5.13 Comunicación básica entre Cliente y Servidor Jabber

5.2.3.2.1.2 Conexión TCP

Aunque la conexión HTTP es la que más utilizaremos por no decir la única es interesante implementar otro método secundario que para efectos Jabber es lo mismo, es decir, esta conexión enviará igualmente mensajes de presencia, notificación e IQ al servidor.
Para este tipo de conexión **CommunicationManager** hace uso de otra función implementada que invoca a una nueva clase llamada **meConnector** que es la encargada de realizar conexiones por sockets con o sin seguridad SSL. Al igual que la conexión HTTP se hace uso de un nuevo hilo de conexión el cual crea un nuevo flujo de información llamado **StreamConnection** que será el que usemos para las conexiones TCP. Este nuevo flujo de información vuelve a llamar a otro hilo de notificación al servidor para decirle que se va a producir una conexión que en este caso es TCP.
La notificación es exactamente igual a la que sucedía en conexiones HTTP ya que al servidor le da igual la forma con la que nos conectamos, él simplemente espera un mensaje con un puerto y una identificación para crear el enlace cliente-servidor.

5.2.4 Localización GPS

La utilización de la tecnología GPS estaba reservada antiguamente para propósitos militares pero vemos que actualmente este sistema se encuentra prácticamente en cualquier dispositivo.
La finalidad siempre es la misma, localizar exactamente donde nos encontramos y utilizar ese punto de referencia para guiarnos a través de mapas y rutas. La meta que se pretende conseguir en este proyecto es utilizar el GPS para determinar el punto geográfico correcto donde ha ocurrido un incidente. Esta información unida a las indicaciones del usuario del móvil hace que se consiga una situación prácticamente perfecta del suceso.

Programar un sistema que utilice el GPS en Java es bastante sencillo, tan solo hay que usar la API correspondiente que en este caso se trata de la JSR-179.
Esta API, también conocida como **Location API**, contiene todo lo necesario para diseñar un pequeño módulo que hará de localizador GPS.

Para empezar hemos añadido una nueva clase **GPS** que contiene:

- Un Hilo que será el encargado de ejecutar el GPS del móvil.
- Una función que nos devolverá las coordenadas en formato Grados/Minutos/Segundos
- Y otra función que nos devolverá las coordenadas en Grados

N O T A

Las coordenadas por defecto que da Location son de la forma LAT_61.51d pero podría interesarnos devolverla de la forma LAT_61:20:36

Dentro de la clase hacemos uso de los componentes **Location,** que nos devolverá las coordenadas **Coordinates** y **Criteria** donde especificaremos los criterios de búsqueda, es decir, la precisión con la que queremos que nos de las coordenadas.

Usar un hilo aparte para lanzar el GPS tiene como finalidad no bloquear el móvil y de esta forma, aunque la inicialización del GPS no funcione, podamos seguir trabajando.

C O D I G O

```
class LaunchGPS extends Thread
{
        public void run()
        {
            System.out.println(" ### - Lanzando GPS");
            crite.setHorizontalAccuracy(accuracyH);
            crite.setVerticalAccuracy(accuracyV);

            try
            {
                lp = LocationProvider.getInstance(crite); //Proveedor mas cercano
                location = lp.getLocation(timeOut); //Localizacion
                coord = location.getQualifiedCoordinates();
                System.out.println(" ### - Coordenadas GPS adquitidas con exito");
            }
            catch (LocationException ex)
            {
                System.out.println(" ??? - Error en la localizacion");
                ex.printStackTrace();
            }
            catch (InterruptedException ex)
            {
                System.out.println(" ??? - Error Interrupcion");
                ex.printStackTrace();
            }
        }
}
```

Se puede observar que se toman en cuenta los valores de **accuracyH** y **accuracyY** que es la precisión en metros que nos darán las coordenadas.

Por otro lado las funciones que nos devuelven la cadena de caracteres con las posiciones GPS son muy parecidas.

CODIGO

```
public String getCoordinatesDDMMSS()
{
    String s_lat = "";
    String s_alt = "";
    String s_lon = "";

    if ( coord != null ){
        double lat = coord.getLatitude();
        float alt = coord.getAltitude();
        double lon = coord.getLongitude();

        s_lat = "_LAT_" + Coordinates.convert(lat, Coordinates.DD_MM_SS);
        s_alt = "_ALT_" + Coordinates.convert(alt, Coordinates.DD_MM_SS);
        s_lon = "_LON_" + Coordinates.convert(lon, Coordinates.DD_MM_SS);
    }

    return "_GPS_" + s_lat + s_alt + s_lon;
}

public String getCoordinatesNormal()
{
    double lat = 0;
    float alt = 0;
    double lon = 0;

    if ( coord != null ){
        lat = coord.getLatitude();
        alt = coord.getAltitude();
        lon = coord.getLongitude();
    }

    return "_GPS_" + "_LAT_" + lat + "_ALT_" + alt + "_LON_" + lon;
}
```

6. Pruebas y Resultados

En todo buen desarrollo de software no debe de falta la realización de una batería de pruebas y ensayos para ver el correcto funcionamiento de la aplicación. Es lógico pensar que mientras se estaba programando cada una de las partes del sistema se han ido probando una por una y corrigiendo los fallos o deficiencias que nos hemos ido encontrando. Ahora lo que vamos a realizar son pruebas conjuntas para evaluar la integridad y consistencia que tiene nuestro proyecto.
Examinaremos minuciosamente todas las partes y haremos un pequeño estudio del rendimiento obtenido así como verificar que los resultados que obtenemos son satisfactorios.

6.1 PRUEBAS GENERALES

La primera tarea que estudiaremos será cómo se tiene que ejecutar la aplicación, qué necesitamos y cómo lo tenemos que hacer.

6.1.1 Ejecución de la Aplicación en un Emulador

A medida que hemos ido creando la aplicación se ha podido ir observando pequeños resultados que se mostraban en una emulación proporcionada por el mismo software de programación, NetBeans. La emulación corre a cargo del programa **Wireless Toolkit 2.5.2** que es un excelente emulador de móviles con soporte Java y que además al pertenecer a “Sun” al igual que el NetBeans ambos trabajan conjuntamente.

Las pruebas generales que vamos a ejecutar se realizarán directamente en el emulador por lo que el NetBeans no nos hará falta. Hay que mencionar que al tratarse de una aplicación exclusiva para móviles el NetBeans crea automáticamente dos archivos imprescindibles para que la aplicación funcione tanto en el emulador como en un móvil real. Estos dos archivos son los siguientes:

Archivo.JAD ➔ contiene información sobre la aplicación que queremos ejecutar
Archivo.JAR ➔ contiene los paquetes comprimidos que forman la aplicación Java

La estructura de estos dos archivos se muestra en la **Figura 6.1** donde se ha añadido un ejemplo de archivo JAD y la descripción general de un paquete JAR.

Figura 6.1 Estructura de un archivo JAD y otro JAR

Cuando tengamos localizados estos dos archivos tenemos que utilizarlos en el WTK 2.5.2 de la siguiente manera:

- Abrimos el WTK 2.5.2 y crearemos un nuevo proyecto

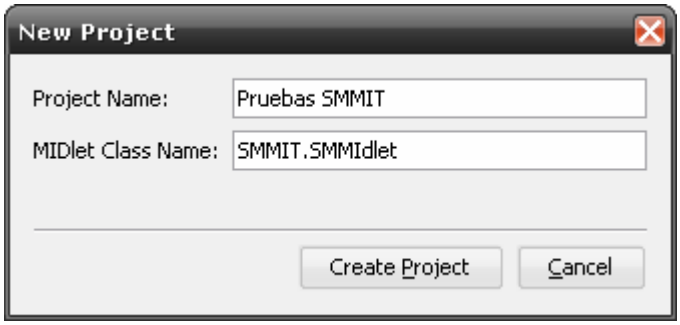


Figura 6.2 Creación de un proyecto con WTK252

Hay que especificar el nombre de la clase principal del programa, en nuestro caso se encuentra dentro del paquete SMMIT y se llama SMMIDlet.

- Aparecerá una nueva ventana de configuración del tipo de aplicación que queremos, dándole a "CUSTOM" seleccionaremos los APIs que se adapten a nuestras necesidades: gráficos SVG, GPS, Parseador de XML, etc.

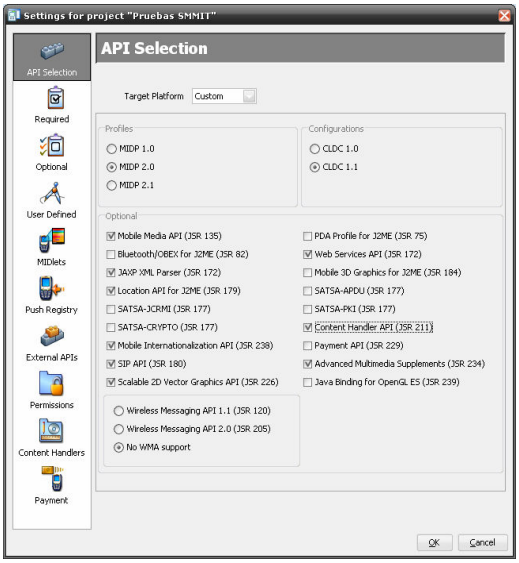


Figura 6.3 Selección de la configuración básica de un proyecto J2ME

- Para una prueba rápida aceptaremos las modificaciones ahora y ya tendremos generado un proyecto vacio en el WTK2.5.2. Ahora hay que seleccionar los archivos .JAR y .JAD e introducirlos en la carpeta que el programa habrá creado en su directorio de instalación:

C:/WTK252/apps/"nombre del proyecto"/bin

Hay que tener en cuenta que la ruta puede variar según la versión del emulador y del sistema operativo.

- Cuando tengamos los dos archivos en su sitio daremos a la **RUN** para lanzar la aplicación. Si todo ha ido bien tendremos la interfaz del proyecto visualizada en pantalla y ya podremos realizar acciones.

Esta sería la forma rápida de lanzar la aplicación en el WTK2.5.2 pero hay a veces que los archivos JAR y JAD creados por NetBeans no suelen corresponderse con la configuración aceptada a la hora de crear un nuevo proyecto. Por lo tanto y para asegurarnos al 100% tenemos la posibilidad de compilar los .JAVA y crear los dos archivos esenciales.

Los dos primeros pasos son iguales y en el tercero en vez de copiar los archivos creados por NetBeans copiaremos todo el proyecto (gráficos, paquetes, archivos punto Java) a la carpeta:

C:/WTK252/apps/"nombre del proyecto"/

Los elementos considerados como recursos o gráficos deben ir a la carpeta **RESOURCE.**
Después pincharemos sobre la opción **BUILD** para generar los .CLASS compilados y luego crearemos los otros dos archivos de ejecución yendo a **PROJECT-PACKAGE-CREATE PACKAGE** y por último dándole a **RUN** para ver los resultados.

De esta forma nos aseguraremos que la aplicación funciona sin la información que NetBeans añade para que funcione sobre él.

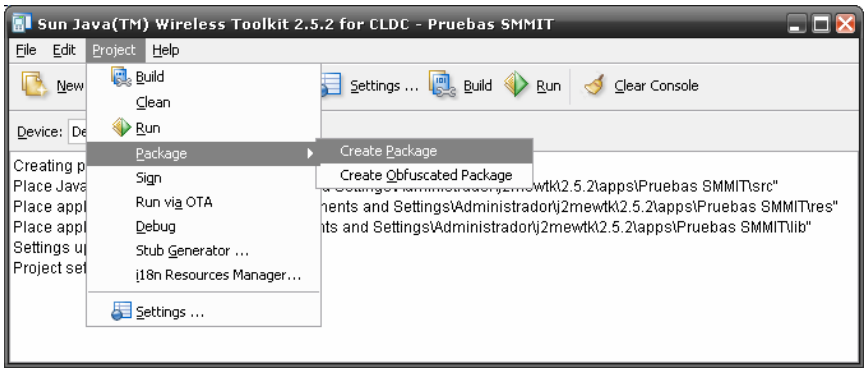


Figura 6.4 Creación del paquete JAR y JAD

6.1.2 Permisos

Cuando la aplicación es lanzada y durante su ejecución deben salir unos mensajes de advertencia (**Figura 6.5**) sobre la manipulación de elementos internos de nuestro móvil como por ejemplo la captura de fotografías, la activación del GPS, guardado de ficheros, etc.

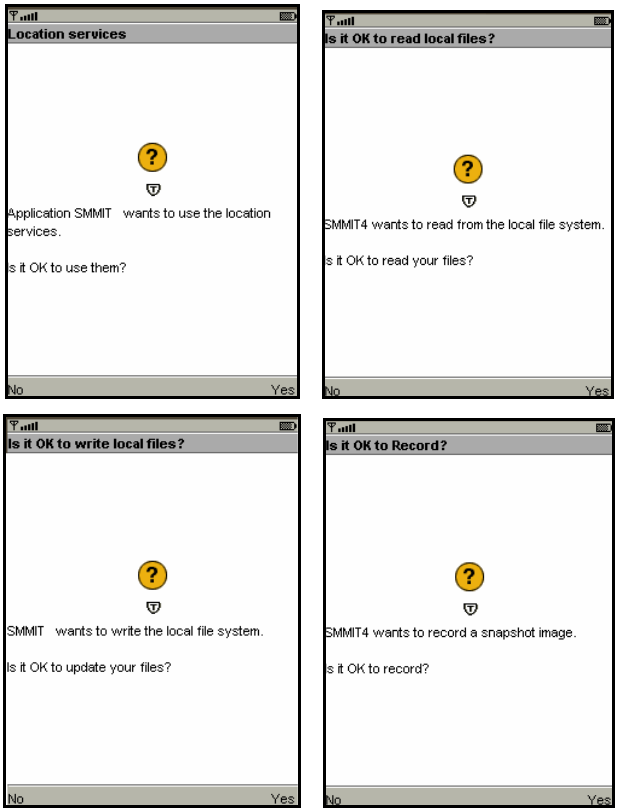


Figura 6.5 Mensajes de Permisos que aparecen en una aplicación cuando se accede a zonas delicadas

Estos mensajes nos preguntarán si realmente damos permiso a nuestra aplicación para que active ciertas partes de nuestro móvil.
Se pueden entender estas advertencias como una medida de seguridad para evitar el mal uso de ciertas aplicaciones que pueden realizar acciones fraudulentas sin que el usuario se dé cuenta.
Sin embargo, para aplicaciones que sabemos de antemano que son seguras o están diseñadas por nosotros mismos, estos mensajes suelen ser molestos y en ciertas ocasiones, como por ejemplo en este proyecto que se usan bastantes hilos, pueden bloquear la aplicación.
Por lo tanto tenemos que, previamente, conceder esos permisos antes de distribuir la aplicación.

Lo podemos hacer de dos formas: dar permisos al emulador o dar permisos a un móvil real.

Para seguir usando el emulador y evitar que aparezcan los avisos haremos lo siguiente:

- Abriremos el WTK252 y cargaremos el proyecto que tenemos
- Luego seleccionaremos **SETTINGS** y seguido pincharemos sobre **PERMISSIONS,** aparecerá una nueva sección donde podremos dar permisos cómodamente.

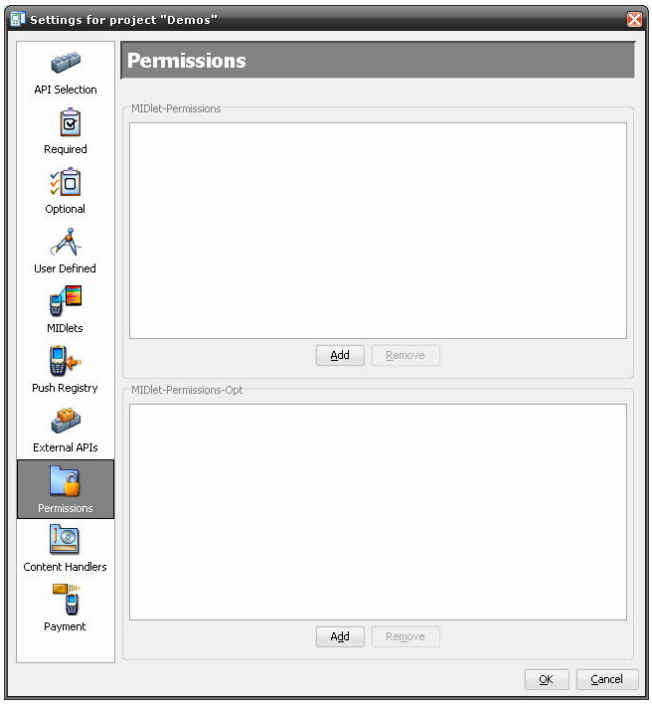


Figura 6.6 Adjudicación de permisos en un proyecto

- Vemos que tenemos dos secciones para introducir los distintos permisos, la primera "Midlet-Permissions" sirve para especificar los permisos que damos a nuestra aplicación y que sin ellos ésta no funcionaría y la segunda sirve para dar permisos opcionales, es decir, permisos que no afectarán a la ejecución de la aplicación. Si seleccionamos **ADD** nos mostrará una nueva ventana con todas las Apis que requieren de permisos.

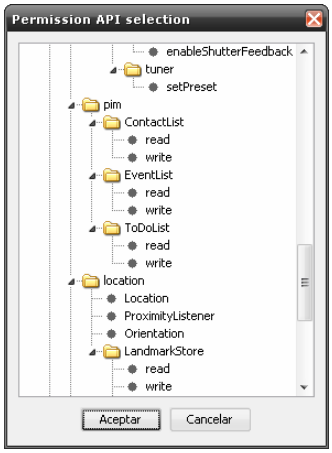


Figura 6.7 Listado de posibles permisos que podemos dar

La otra opción de dar permisos, a otros dispositivos y otros emuladores que no son de java, es un poco más complejo porque nos metemos en un área más delicada, se explica mejor en el siguiente punto.

6.1.2.1 Creación de una Firma Certificada

Si queremos que nuestra aplicación sea aceptada por otras plataformas tiene que estar debidamente firmada o bien por nosotros mismo en un ámbito de pruebas y testeo o bien por otras entidades más importantes que aporten un certificado de autoridad del dispositivo base.

Para crear una firma se procede de la siguiente forma:

- Tenemos que seguir todos los pasos anteriores que hacíamos para aplicar permisos y ejecutar la aplicación en un emulador de *SUN MICROSYSTEM*.
- En la ventana principal de WTK252 vamos al menú **PROJECT** y seleccionamos **SIGN,** aparecerá la siguiente ventana:

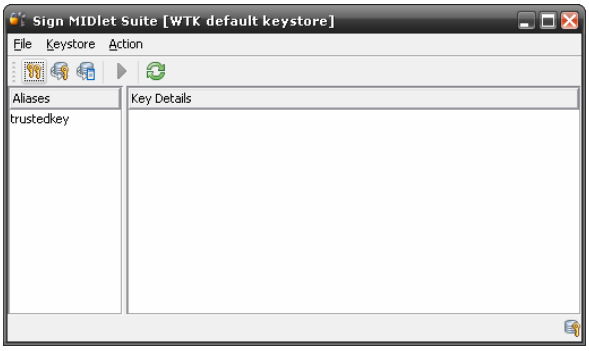


Figura 6.8 Panel de Asignación de Firmas

Existen 4 tipos de firmas digitales que podemos aplicar:

-**untrusted**: Proporciona un nivel alto de seguridad para aplicaciones cuyo origen no se puede determinar. Se pedirá confirmación por parte del usuario para operaciones sensibles.

-**trusted**: Todos los permisos son concedidos en este dominio

-**minimum:** Todos los permisos son denegados en este dominio

-**maximum:** Todos los permisos son concedidos en este dominio (tiene la misma validez que trusted)

Por defecto está creada la firma llama **TrustedKey** que incorpora el modo trusted. Si no deseáramos especificar más datos porque queremos hacer simples pruebas con esta firma nos bastaría. Si por el contrario queremos crear otro tipo de firma procederemos al paso 3.

- Para crear una nueva firma seleccionaremos **KEYSTORE** y **NEW KEY PAIR** nos mostrará una pequeña ventana donde introduciremos algunos datos y cuando aceptemos nos saldrá la opción de elegir el tipo de firma que deseamos.

Figura 6.9 Creación de una firma personalizada

- Dando a **OK** ya tendremos una firma creada por nosotros, ahora tan sólo falta seleccionar en el menú **ACTION SIGN MIDLET** y ya tendremos nuestra aplicación firmada.

Ahora bien, si lo que queremos es que nuestra aplicación esté firmada por una firma real y autentica reconocida por alguna autoridad de certificados (CA) tendremos que efectuar estos pasos:

1) Crearemos una firma como lo hacíamos anteriormente
2) Generaremos después una petición de certificado de firma, GENERATE CSR.
3) Enviar el archivo generado a una autoridad de certificados (CA) y algunos datos de identidad.
4) Una vez la autoridad verifica tus datos personales y cobre el precio del certificado mandará un certificado que garantiza tu clave pública.
5) Ahora podemos importar de Sun Java™ Wireless Toolkit for CLDC la firma que certificará nuestro proyecto.

6.2 PRUEBAS DE INTERFAZ

Lo primero que se contempla cuando se lanza la aplicación por primera vez es la interfaz que hemos diseñado. Se procuró que fuera una interfaz sencilla, amigable, fácil de manejar y muy intuitiva sin perder en calidad y velocidad. Las pruebas que se han ido realizando según se iban incorporando nuevos gráficos SVG han demostrado que efectivamente el uso de imágenes vectoriales consume muchos recursos. Con la configuración estándar que tiene NetBeans para hacer las pruebas en el emulador no se podían cargar más de dos pantallas en memoria y la aplicación se cerraba por falta de recursos.
Por defecto, el emulador de WTK252 viene con 2 Megabytes de memoria interna (simulada) que apenas cubre un 10% del proyecto. Es necesario cambiar la capacidad del emulador y para ello accederemos al **MANAGER EMULATORS** donde podemos cambiar el aspecto del emulador, de la pantalla, etc. pero lo que nos interesa está dentro de **TOOLS & EXTENSIONS** y **PREFERENCES.** Aparecerá la ventana que se muestra en la **Figura 6.10** con una serie de opciones para configurar el emulador.

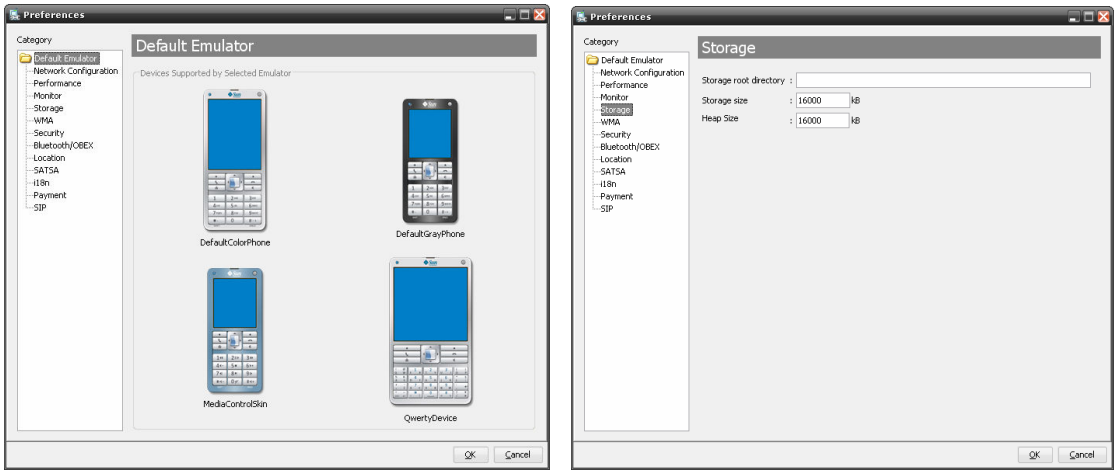


Figura 6.10 Configuración de las características del emulador

En las preferencias del emulador podemos hacer que el móvil simule diferentes características que puede tener un móvil real como por ejemplo decirle que las coordenadas GPS son unas determinadas, que el Bluetooth funcione a diferentes modalidades, etc. Nos interesa el **STORAGE** que en este caso simula la cantidad de memoria interna de que disponemos, con 16 MBytes tenemos más que suficiente (recordemos que el Nokia N95 tiene 128 MBytes).

Con este problema resuelto ya se pueden cargar en memoria todos los gráficos necesarios sin faltar recursos, no obstante el que tengamos más recursos no influye para nada en la velocidad con la que fluya la interfaz.

Se han hecho pruebas en diferentes ordenadores con distintas velocidades de procesador y hay que decir que la fluidez de la interfaz varía mucho entre velocidades.

N O T A

El que haya tanta diferencia entre procesadores viene a que el emulador se ejecuta en un entorno donde hay más aplicaciones activas. En un móvil real la aplicación tiene todos los recursos para él solo.

6.2.1 Optimización de Gráficos SVG

La utilización de gráficos no debería ser un inconveniente a la hora de diseñar y programar una interfaz cualquiera pero en este caso se recomienda la utilización de gráficos no muy grandes en cuanto a resolución y tamaño por el simple hecho de que van a visualizarse en una pantalla reducida.
A los gráficos SVG se les puede incorporar otros formatos gráficos como por ejemplo BMP, JPG, GIF o PNG donde se incluyen codificados en Base64 dentro del propio XML, es decir, es un conjunto de caracteres.

N O T A

Base 64 es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Todas las variantes famosas que se conocen con el nombre de Base64 usan el rango de caracteres A-Z, a-z y 0-9 en este orden para los primeros 62 dígitos, pero los símbolos escogidos para los últimos dos dígitos varían considerablemente de unas a otras

Utilizaremos concretamente el formato PNG que consigue ya de por sí una compresión asequible para reducir un poco el tamaño de las imágenes y permite una profundidad de hasta 24 bits.

Podríamos reducir aún más el tamaño de las imágenes (conseguir 5Kb menos en un archivo SVG significa ganar velocidad) realizando una doble conversión de formatos pasando primero de PNG a GIF y luego de GIF a PNG.
La explicación de por qué este cambio viene ligada a que los gráficos GIF soportar hasta 256 colores y al realizar el cambio de PNG a GIF la información de colores se reduce y al volver a pasarla a PNG volvemos a comprimir consiguiendo, a veces, arañar unos pocos Kbytes.

También hay que tener en cuenta que las resoluciones que debemos manejar tienen que ser bastante pequeñas. Un simple icono debería tener una resolución de 16x16 y aunque se deformara la imagen en la pantalla del ordenador en el móvil sale bastante nítida y no se aprecian los fallos.

6.2.2 Gráficas de Rendimiento

Netbeans incorpora una serie de monitores de seguimiento para ver la evolución y la forma con que Java libera memoria cuando es necesario. Este método es útil para que, sin recurrir al aumento de la memoria interna, veamos en qué momento podemos liberarla de forma manual.
La siguiente gráfica muestra los recursos que se gastan cuando la aplicación es lanzada, la línea verde representa los recursos gastados y la roja el máximo en ese momento.

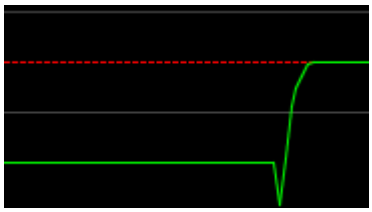


Figura 6.11 Recursos utilizados al iniciar el programa

Inicialmente la aplicación en reposo ya gasta unos pocos Kbyte y cuando se inicia se produce un aumento considerable de 1,5 Mbyte porque en ese momento lanzamos ya varios hilos de ejecución correspondientes a la interfaz, cámara de fotos y manejador de archivos.
En las primeras pruebas que se hicieron para estimar los recursos necesarios se utilizaba una opción disponible en el monitor que llamaba al Collector Garbage y de esta forma decidir si era eficiente llamarlo nosotros mismo manualmente en código para, por si fuera necesario, tener más recursos.
Si ahora le diéramos a esa opción ocurriría lo siguiente:

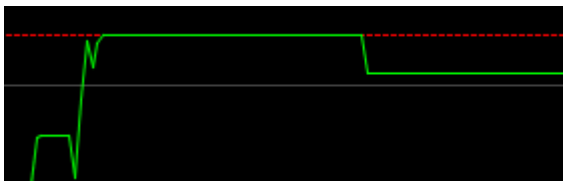


Figura 6.12 Liberación de recursos manual con Collector Garbage

Hemos liberado casi medio Mega y para nuestro caso es bastante considerable si tuvieras que usar un móvil con pocos recursos. Este tipo de estudio en un móvil de hoy en día no haría mucha falta pero es bueno saber las limitaciones con las que trabajamos para no llevarnos sustos.

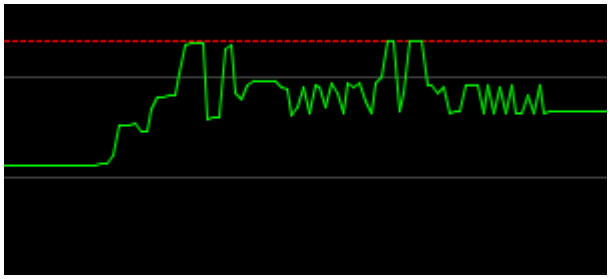


Figura 6.13 Oscilación de recursos durante la interactuación de la interfaz

Esta gráfica muestra la interacción entre los menus que tiene la interfaz y los formularios. Cada vez que se accede a un menú SVG consumo recursos y cuando llamamos a un formulario se reducen.
La línea roja en este caso llega hasta los 2,5Mbytes sobrepasando el umbral que pone por defecto NetBeans. Podríamos en este momento llamar al "CG" a ver qué resultados nos da.

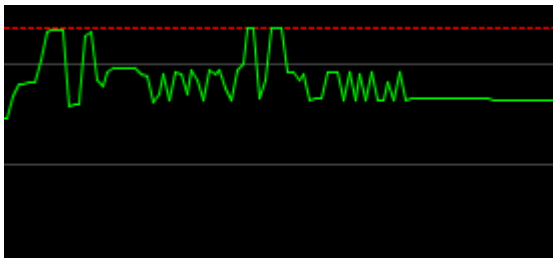


Figura 6.14 Llamando al "CG" después de utilizar la interfaz. Recursos conseguidos mínimos.

Apenas se aprecia la liberación de recursos porque la aplicación ya tiene llamadas al GC en sitios estratégicos optimizando los recursos.

Por último cabe comentar qué sucede cuando capturamos fotos ya que éstas se guardan en la memoria del móvil antes de enviarlas (esta prueba no ha podido realizar con el Video por la razón de incompatibilidad de hilos mencionada en el capítulo de Multimedia).

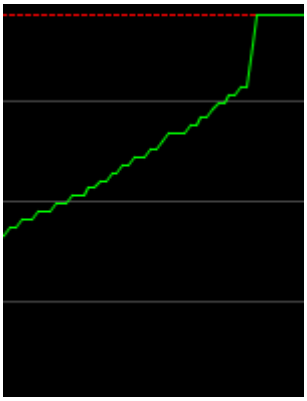


Figura 6.15 Incremento de los recursos cuando capturamos fotos.

Los pequeños escalones simbolizan cada vez que damos a capturar una foto y la guarda en un array de bytes. Se han realizado unas 20 fotos y el último pico es el producido al guardarlas en memoria. Vemos que ahora la línea roja marca los 4 MBytes que está bastante bien para haber hecho 20 fotos de 160x120.

Resumiendo, el control de los recursos a la hora de programar para móviles y más si es con Java es esencial para hacer un buen programa capaz de correr en cualquier plataforma.

6.3 PRUEBAS DE ALMACENAMIENTO

Una de las opciones que disponemos en esta aplicación es la posibilidad de guardar un resumen del informe que se haya creado en la memoria física del móvil. Comentamos que manipular ficheros lógicos en la memoria física es un proceso sencillo pero a la vez muy delicado ya que estamos tocando puntos sensibles tanto del propio móvil como de la aplicación.
Es necesario pues efectuar unas cuantas pruebas para ver la integridad de nuestro sistema.

Las pruebas son bien sencillas, se trata de ir haciendo varios informes y ver como efectivamente se van guardando en la memoria. Los archivos generados no ocupan más de 1 Kbyte por lo que no nos tenemos que preocupar en excesivo del espacio disponible en el disco, si por alguna razón faltara espacio se mostraría un mensaje de alerta comunicándonoslo. Lo que tenemos que tener cuidado es a la hora de escribir el fichero porque este proceso lanza varios mensajes de permisos que ya se explicaron anteriormente y si no se les da a todos una confirmación bloquean el programa.

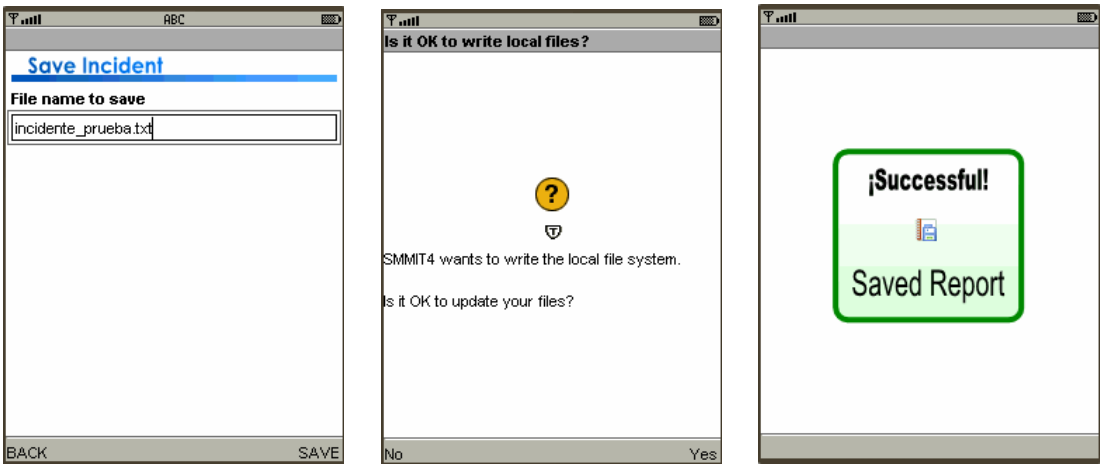


Figura 6.16 Pantallas que se visualizan cuando guardamos un ficheros

La ruta de almacenamiento dependerá de la distribución interna que tenga cada móvil o cada emulador, en nuestro caso el WTK252 guarda los archivos en el directorio **filesystem\\root1**\ y se puede comprobar perfectamente que el fichero se ha creado con existo como por ejemplo el mostrado en la **Figura 6.12**.

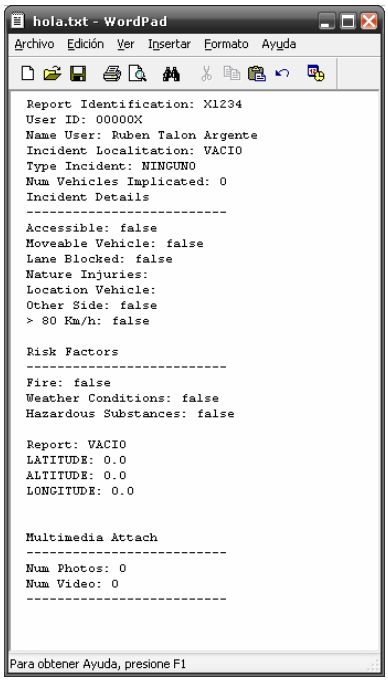


Figura 6.17 Ejemplo de Fichero guardado en disco

6.4 PRUEBAS DE CREACIÓN DEL INFORME

Conforme se iba desarrollando poco a poco el proyecto una de las preguntas que incluso en estos momentos nos podemos plantear es qué elementos identificativos añadimos a un informe, en pocas palabras, qué información debemos contemplar.
No está claro al 100% la información que queremos plasmar en un informe y continuamente el proyecto ha tenido cambios en los formularios para adaptarlos a las nuevas exigencias que iban surgiendo. Pero esto no es un problema grave, los formularios en J2ME son simples campos de textos que se pueden modificar libremente y hacer pruebas sobre ellos es bastante fácil. Lo único que hay que tener en cuenta es inicializar esos campos para que a la hora de crear el informe no se disparen excepciones por campos nulos o conversiones de caracteres extrañas. Por esta razón una prueba prudente sería crear un formulario totalmente en blanco y guardarlo o enviarlo y ver como efectivamente el sistema responde con satisfacción. Se debe de probar a la inversa, es decir, rellenar al máximo todas las posibilidades que nos dan los formularios y añadir unas cuantas fotos y videos. Un ejemplo de formulario vacio se puede ver en la figura inmediatamente anterior a este punto.

6.5 PRUEBAS DE CONEXIÓN CON UN SERVIDOR JABBER (OPENFIRE)

Estas pruebas se pueden considerar como las más importantes y a la vez las más interesantes de estudiar ya que aquí reside la parte estrella del proyecto. Volvemos a recordar que las conexiones se realizan mediante un protocolo gratuito de código abierto llamado Jabber respaldado por XMPP como la estructura de los mensajes.

Para las pruebas hemos recurrido a un servidor Jabber también gratuito y de código libre llamado **OpenFire** que hará las funciones de recepción y envío de mensajes y otro programa de mensajería instantánea llamado **Spark** de la misma compañía que OpenFire y que hará la función de monitorizar lo que le llega al servidor (este proceso lo explicaremos en breve).

Podemos bajarnos dichos programas de su página web en la sección de descargas:
http://www.igniterealtime.org/downloads/index.jsp

CONFIGURACIÓN OPENFIRE

Una vez descargado e instalado en nuestro ordenador personal automáticamente se lanzará un asistente en formato página web donde lo configuraremos rápidamente.

a) Seleccionamos el idioma
b) Damos el nombre del servidor y los puertos de conexión

N O T A

El dar un nombre a nuestro servidor Jabber sirve para hacer las pruebas localmente, es decir, sobre la misma máquina. Para conexiones exteriores tendríamos que conectarnos con la IP del ordenador donde resida el OpenFire

c) Seleccionamos que el asistente cree una nueva base de datos interna para llevar el registro de los usuarios. Todo este proceso es automático y no tenemos que preocuparnos por nada.
d) Escribimos el nombre del administrador y su contraseña y acto seguido nos identificaremos para entrar. Saldrá una pequeña ventana de depuración donde veremos el nombre de nuestro servidor y el puerto de consola.

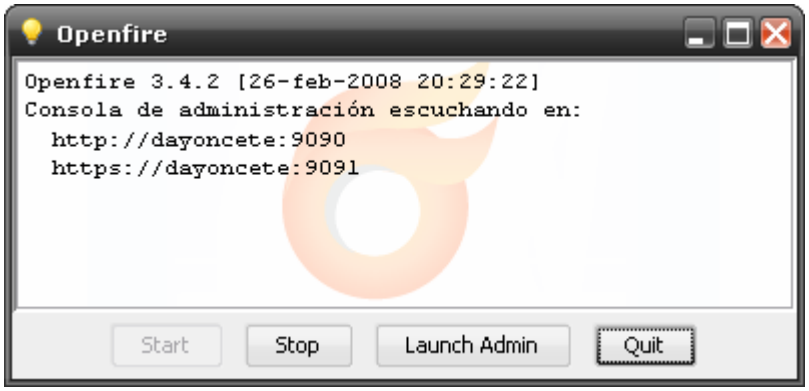


Figura 6.18 Inicio del Servidor Jabber OpenFire

e) Una vez registrados entraremos dentro de la consola de administración del servidor. Este panel incorpora varias opciones como por ejemplo la monitorización del tráfico que procesa y la configuración de las conexiones. Seleccionaremos la opción de crear nuevos usuarios.

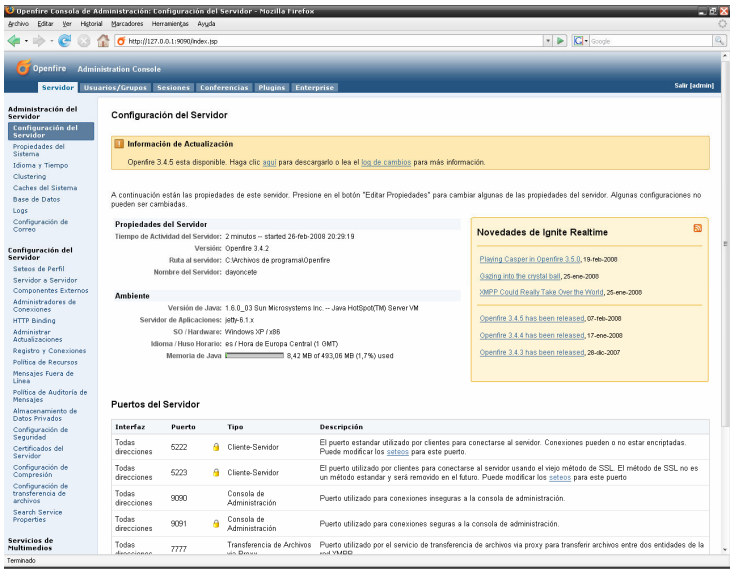


Figura 6.19 Web Administrativa del Servidor OpenFire

f) La creación de nuevos usuarios es vital para llevar un control exhaustivo, en nuestro caso esta parte es importante porque aquí el servidor actúa de la misma forma que actuaría un servidor real (por ejemplo el de una central de policía). Como prueba crearemos un usuario llamado Usuario1 que hará el papel de Cliente y otro llamado Usuario2 que será el Ayudante, por así decirlo, del Servidor.

CONFIGURACIÓN SPARK

La idea de utilizar un segundo programa de mensajería instantánea basado en Jabber es para ver los datos que el programa manda al servidor. El usuario del móvil manda todos los datos al servidor pero no vemos que es lo que realmente le llega, podríamos hacer un debug de todo lo que lee pero veríamos más código XML que lo que nos interesa. Por esta razón es fácil crear un usuario ficticio al que le llegará todo y de esta forma ver en su pantalla los datos que enviamos.

Cuando tengamos el programa instalado tenemos que registrarnos (**Figura 6.15**), para eso el servidor nos dio de alta con el nombre **Usuario2** y password **usu2.** Solo faltaría escribir el nombre del servidor, si somos un usuario local bastaría con poner el nombre del servidor y si no (externos) pondríamos la IP.

Figura 6.20 Inicio de sesión con Spark

Cuando ingresemos y todo sea correcto veremos cómo nos aparece una ventana con todos los contactos registrados por ese usuario, en nuestro caso todavía no tenemos ninguno. De momento añadiremos al **Usuario1** que será el usuario del móvil. Vamos al menú **CONTACTOS** y seleccionamos **AGREGAR CONTACTO,** seguidamente aparecerá una ventana como la de la **Figura 6.16** donde podremos introducir al nuevo contacto**.**

Figura 6.21 Agregación de un nuevo usuario con Spark

A partir de ahora el **Usuario1** pasa a formar parte de la lista de contactos y podremos comunicarnos con él.

Ahora que ya tenemos ambos programas instalados, configurados y comunicándose entre ellos procederemos a enviar datos desde el móvil.

a) Iniciaremos la aplicación en un emulador (da igual si lo lanzamos desde NetBeans o directamente desde el WTK252)
b) Configuraremos las conexiones accediendo al menú **CONFIGURATION** y **CONNECTION.** Nos aseguraremos de que conectaremos con el servidor que toca por el puerto correspondiente que por defecto es el 8080 HTTP.
c) Vamos a la sección de gestión de incidentes y crearemos uno nuevo. En el menú incidente, por ejemplo, haremos una foto de prueba y la guardaremos. Luego entramos en la opción de **REPORT** para añadir todos los detalles del incidente.
d) Cuando tengamos el informe daremos a la opción **SEND** y automáticamente nos conectaremos con el servidor. Si la conexión fuera errónea siempre podemos volver hacia atrás y modificar los parámetros de configuración. Cuando la conexión es exitosa saldrá una pequeña ventana de confirmación y un formulario donde se indicará el número de fotos capturadas y si hay o no video.
   En este momento el servidor deberá mandar un identificador para el informe multimedia que va a recibir, dicho identificador se guardará automáticamente mostrando por pantalla una imagen de identificación recibida. Si no hay identificación no podremos mandar el informe.

e) Confirmamos y damos en la opción **SEND** nuevamente y, si todo ha ido bien, el informe le llegará al usuario2 en formato de texto.

El usuario2 recibirá la foto en un formato codificado Base64, es decir, recibirá una pila de caracteres que en conjunto forman la imagen o imágenes (**Figura 6.17)**.
Lógicamente no se pretende que el usuario2 reconozca estos caracteres, esto solo lo hacemos de prueba para ver el funcionamiento. En realidad el servidor recibirá los caracteres y automáticamente los transformará en una foto decodificando en Base64.
Después de recibir la foto más abajo están todos los demás datos que estos sí que se pueden identificar (**Figura 6.18**).

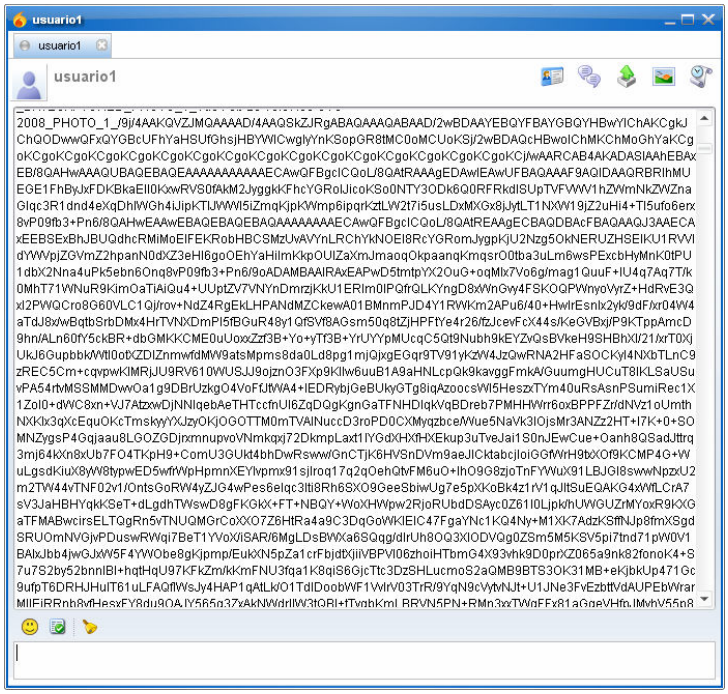


Figura 6.22 Mensaje recibido codificado en Base64 - Fotografía

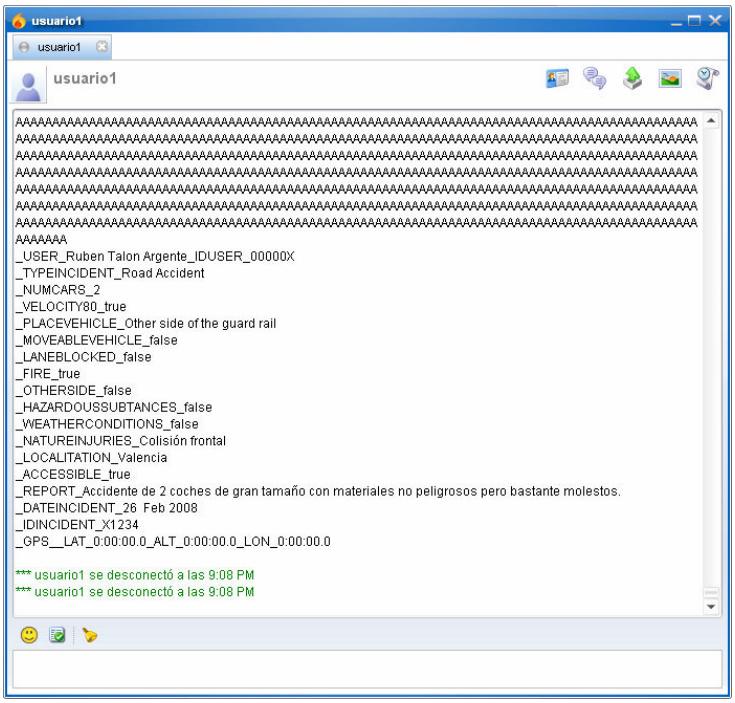


Figura 6.23 Mensaje recibido codificado en Base64 – Informe

6.6 RESULTADOS OBTENIDOS

Los resultados que hemos obtenido encajan con todos los requisitos y necesidades que propusimos al inicio de este documento.

Se trata de una aplicación sencilla de manejar gracias a su interfaz de usuario desarrollada con gráficos SVG que se adaptan a cualquier tamaño de pantalla sin perder calidad.

Se adapta a cualquier dispositivo que soporte Java y que hoy por hoy son casi el 100% de los móviles.

La comunicación entre móvil y servidor es segura y rápida utilizando un protocolo gratuito y de código abierto que podemos amoldar a nuestras necesidades al estar implementado en XML.

Hacemos uso de todas las herramientas que pueden proporcionarnos un dispositivo móvil desde capturar fotografías con una alta resolución hasta decirnos la posición GPS en el mismo momento.

Por otro lado se ha probado la aplicación en el emulador de Nokia dando resultados totalmente adversos ya que este emulador no incorpora la utilidad de simular una cámara digital por lo que las pruebas sin elementos multimedia son incompletas.

7. Conclusiones

Los accidentes de tráfico son algo que ya existía anteriormente, están ocurriendo en estos momentos y desgraciadamente seguirán sucediendo una y otra vez. No podemos hacer prácticamente nada para evitar que se produzcan, se está concienciando a la gente de los distintos peligros que hay en la carretera e incluso el código penal ha cambiado para definir que una infracción grave es un delito que se paga con la cárcel. Los agentes de policía gestionan decenas de accidentes que suponen a largo plazo unos costes bastantes elevados para la sociedad. Estas gestiones, a veces, son consideradas muy importantes porque el agente tiene que describir con mucho detalle qué es lo que ha ocurrido y como se puede solucionar. Este proyecto es una herramienta muy útil para crear un informe virtual de todo lo sucedido ya que podemos hacer fotos, tenemos posiciones GPS y lo más importante es que se trata de una utilidad que se guarda en el bolsillo.
Aportamos pues nuestro pequeño granito de arena en mejorar la eficiencia de los agentes a la hora de intervenir en un incidente ahorrando un tiempo que en ocasiones puede salvar vidas y una cantidad considerable de presupuesto.

Viendo los objetivos impuestos al principio de la memoria vamos a recordarlos y a decidir si se han cumplido tal y como esperábamos.

- Creación de un informe multimedia específico y detallado con elementos multimedia (fotografías, videos, GPS) donde se plasme de forma precisa todas las características y detalles de un incidente producido.
    - Objetivo Cumplido: Como ya se ha explicado anteriormente el programa dispone de todos los medios necesarios para realizar fotos, videos y localizar el punto exacto con la ayuda del GPS. A parte de la información vital que pueda aportar el propio usuario del móvil se han definido algunos campos que agilizarán los trámites.

- La aplicación a manejar tiene que ser sencilla y muy intuitiva, el tiempo de aprendizaje tiene que ser mínimo y la aplicación no debe de requerir en mayor medida la intervención del usuario excepto para introducir datos y configurar algún parámetro básico.
    - Objetivo Cumplido: Se ha diseñado una interfaz con un alto grado de usabilidad donde aparecen muchos iconos identificativos y menús vistosos. Al ser una aplicación con pocas pantallas se hace más rápido el aprender a utilizarlas.

- La velocidad de la aplicación en si tiene que adaptarse a las exigencias del objetivo principal, es decir, tiene que desenvolverse con soltura y fluidez.
    - A pesar de optimizar al máximo toda la aplicación, exclusivamente lo referente a la interfaz, hay que decir que se requiere de dispositivos móviles potentes.

- Alto grado de adaptabilidad en cualquier dispositivo preparado para soportarlo (soporte JAVA, inclusión de todas las APIs utilizadas, Cámara integrada, GPS). La gran cantidad de móviles de diferentes modelos, marcas, tecnologías, resoluciones… hacen que la aplicación se adopte a cualquiera de ellos.
    - Objetivo Cumplido: La mayor preocupación de cambiar de soporte es que la interfaz no se ajuste a cada pantalla y tengamos problemas a la hora de visualizar contenidos. Este problema se soluciona utilizando gráficos vectoriales SVG que se escalan sin perder calidad sobre cualquier tamaño de ventana.

- Los usuarios permitidos para la utilización y gestión de esta aplicación tienen que estar identificados en una base de datos. Si la aplicación corresponde con incidencias de tráfico los usuarios válidos serán todos aquellos que tengan que ver con dicha finalidad.
    - Objetivo Cumplido: Se ha incluido una gestión sencilla del personal que puede interactuar con la aplicación. Dicha gestión se realiza con RMS y con la supervisión del servidor central.

- Como continuamente estamos evolucionando tecnológicamente en lo que a móviles se refiere hay que aprovecharse y utilizar todas las nuevas incorporaciones que se añadan a los dispositivos móviles. Por ejemplo, últimamente están apareciendo móviles con pantalla táctil que hacen que su control sea más rápido y sencillo.
    - Este objetivo lo cumplimos ahora pero a medida que pase el tiempo la aplicación se quedará anticuada con respecto a las nuevas tecnologías que vayan apareciendo.

- En la parte de las comunicaciones con un servidor impusimos la necesidad de que las conexiones fueran rápidas y que fueran controladas tanto por el cliente como por el servidor. La aplicación sólo actúa de cliente mandando un informe completo y recibiendo confirmaciones e identificadores del servidor.
    - Objetivo Cumplido: Utilizando el protocolo Jabber hacemos que la conexión con un servidor sea sencilla y además la podamos modificar a nuestras necesidades. A pesar de ser una tecnología joven está dando muy buenos resultados y cada vez más está siendo utilizada por una gran cantidad de usuarios y corporaciones.

No todo acaba aquí, esta herramienta aparte de ser útil a los agentes de policía puede ser útil a cualquier persona que necesite transmitir cualquier cosa con mucho detalle y que sea barata.
Se puede modificar muy rápidamente los formularios e incluso se pueden añadir más sin problemas y adaptar la herramienta a cualquier otro servicio. Recordemos que incluso la interfaz está hecha en gráficos SVG o lo que es lo mismo con archivos XML que podemos editar sin complicaciones.

Resumiendo, el objetivo principal está claramente definido pero no obstante se ha diseñado una aplicación bastante flexible capaz de adaptarse en un corto periodo de tiempo a otras finalidades por un precio muy reducido.

7.1 TRABAJO FUTURO

Observando con detenimiento el trabajo realizado, somos conscientes de determinados aspectos que podrían sin duda ser mejorados, y que por lo tanto formarían parte de lo que consideraríamos como un trabajo futuro.

Ya que hemos concluido que el proyecto se podría reprogramar para otras finalidades, teniendo como base el principio de crear unos informes de calidad, se podría utilizar un archivo XML donde se definiesen los campos de los formularios que queremos que disponga el programa y de esta forma evitar entrar en el código y cambiarlos a mano.

Una de las finalidades secundarias que podríamos dar a este proyecto es que colaborase con otros proyectos relacionados con incidencias de tráfico como el que nombrábamos al principio y conseguir un sistema híbrido entre gestión de incidencias de tráfico a distancia y a tiempo real.

Por otra parte el sistema de envio de mensajes al servidor está diseñado bajo un protocolo que todavía está en desarrollo aparte de que no está optimizado para ser usado en sistemas de telefonía móvil. Actualmente dicho protocolo dispone de estructuras pensadas para la transferencia de ficheros <file-transfer> que para sistemas de sobremesa funciona sin problemas pero para móviles aun contiene algunos errores y por esta razón aun no se ha usado en este proyecto. Cabe esperar que más adelante se adapte este protocolo a la telefonía móvil ahora que está surgiendo el boom de la mensajería instantánea en los celulares.

Para concluir con este apartado decir que cuando hablamos de telecomunicaciones, informática, electrónica, etc. estamos hablando de temas que estarán continuamente evolucionando, saldrán nuevos aparatos que revolucionen a las personas, nuevos sistemas más rápidos, nuevos componentes y todo esto hay que aprovecharlo y trabajar sobre ellos. Este proyecto está pensado para que se pueda adaptar en un futuro a lo que se avecina y no quedarse nunca atrás.

BIBLIOGRAFÍA

LIBROS

- **[EIS02]** J.David Eisenberg, **SVG Essentials**, O'Reilly, 2002.
- **[GO06]** Vikram Goyal, **Pro Java ME MMAPI**, Apress, 2006.
- **[WES03]** Addison Wesley, **Programming Wireless Devices with the J2ME 2nd Edition**, 2003.
- **[SHI02]** Iain Shigeoka, **Instant Messaging in Java, Manning**, 2002.
- **[TO02]** Kim Topley, **J2ME In A Nutshell,** O'Reilly, 2002.
- **[WHHE02]** James White and David Hemphill, **J2ME IN SMALL THINGS**, Manning, 2002.
- **[NIÑ06]** Proyecto Fin de Carrera de Ingeniería Informática.Servicio Web multidispositivo de información de incidencias de tráfico con imágenes de cámaras. Vicente Niñerola Molina. Valencia, España. 2006

MANUALES Y PRESENTACIONES

- **[TUTSUN04] Tutorial Mobility Modules for NetBeans**, Sun Microsystem, Inc 2004.
- **[KAI04]** Mai Kozakai. **Jabber/J2ME**. University Stuttgart, 2004.
- **[LA05]** Oscar Lage Serrano, **Jabber/XMPP**, 2005.
- **[CHIINHAR05]** Suresh Chitturi, Michael Ingrassia and Vincent Hardy, **Building Scalable Mobile Application Using JSR 226**, Sun Microsystem and Nokia Center, 2005.
- **[CAHAR05]** Tolga Capin and Vicent Hardy, **JSR-226**, Sun Microsystem and Nokia, 2005.
- **[ARI05]** J.Ramon Arias Garcia, **Almacenamiento Persistente de datos con RMS**, Universidad de Oviedo, 2005
- Java Community Process, **Location API for J2ME**, JSR-179, Nokia, 2003
- **[PRI02]** Manuel J.Prieto, **Introducción a J2ME,** 2002.
- **[GALU04]** Sergio Gálvez y Lucas Ortega, Java a tope: J2ME, Universidad de Málaga, 2004

SITIOS WEB

- **[W3 JABBERES]** JabberES, comunidad hispana de Jabber: www.jabberes.org
- **[W3 JCOMM]** Jabber Community: www.jabber.org
- **[W3 JINC]** Jabber Inc: www.jabber.com
- **[W3 JPOW]** Jabber Powered: www.jabberpowered.org
- **[W3 J2ME]** ¿Qué es J2ME?: http://www.java.com/es/download/faq/whatis_j2me.xml
- **[W3 J2ME REF]** Referencia J2ME: http://java.sun.com/javame/index.jsp

- **[W3 SUNAPI]** Referencia a API y documentación: http://java.sun.com/javame/reference/apis.jsp
- **[W3 XMPP]** XMPP Standards Foundation: http://www.xmpp.org/
- **[W3 XMPP SCHE]** XMPP Schemas: http://www.xmpp.org/schemas/
- **[W3 MJABBER]** Programa Base de Jabber para móviles: http://sourceforge.net/projects/micro-jabber
- **[W3 JMIX BIA]** Beta del programa Jabber Mix Client (Gabriele Bianchi): http://jabbermixclient.sourceforge.net/
- **[W3 JSRVCLI]** Cliente y Servidor Jabber: http://www.igniterealtime.org/projects/openfire/index.jsp
- **[W3 MS WM]** Windows Mobile en Microsoft: http://www.microsoft.com/windowsmobile/default.mspx.
- **[W3 SYMB]** Sistema Operativo Symbian: http://www.microsoft.com/windowsmobile/default.mspx
- **[W3 TSYMB]** Foto sobre todo lo relacionado con Symbian: http://www.microsoft.com/windowsmobile/default.mspx
- **[W3 ANDROID]** Página Oficial de Android – Google: http://code.google.com/android/
- **[W3 WIKI ANDROID]** Definición por Wikipedia de Andrid: http://es.wikipedia.org/wiki/Android
- **[W3 WIKI SYMB]** Definición por Wikipedia de Symbian: http://es.wikipedia.org/wiki/Symbian
- **[W3 FLINUX]** Proyecto Familiar Linux para PDAs: http://familiar.handhelds.org/
- **[W3 WIKI FLINUX]** Definición por Wikipedia de Familiar Linux: http://es.wikipedia.org/wiki/Familiar_Linux
- **[W3 MCOCOMO]** Modelo de Cocomo 2000: http://trevinca.ei.uvigo.es/~cfajardo/tema4.htm
- **[W3 ILLU]** Página de Adobe Illustrator: http://www.adobe.com/es/products/illustrator/
- **[W3 FORO NOKIA]** Foro de desarrolladores de Nokia: http://www.forum.nokia.com/
- **[DGT08]** Balance de Siniestralidad del Año 2007, Dirección General de Tráfico - Madrid 2008
- **[W3 NETBEANS]** Página Oficial de NetBeans: http://www.netbeans.org/ http://www.dgt.es/was6/portal/contenidos/documentos/prensa_campanas/notas_prensa/notaprensa033.pdf

OTROS DATOS DE INTERÉS

- Balance de Siniestralidad Año 2007, Dirección General de Tráfico: http://www.dgt.es/was6/portal/contenidos/documentos/prensa_campanas/notas_prensa/notaprensa033.pdf
- ITS España: http://www.directorioits.com/index_a.asp

Anexo

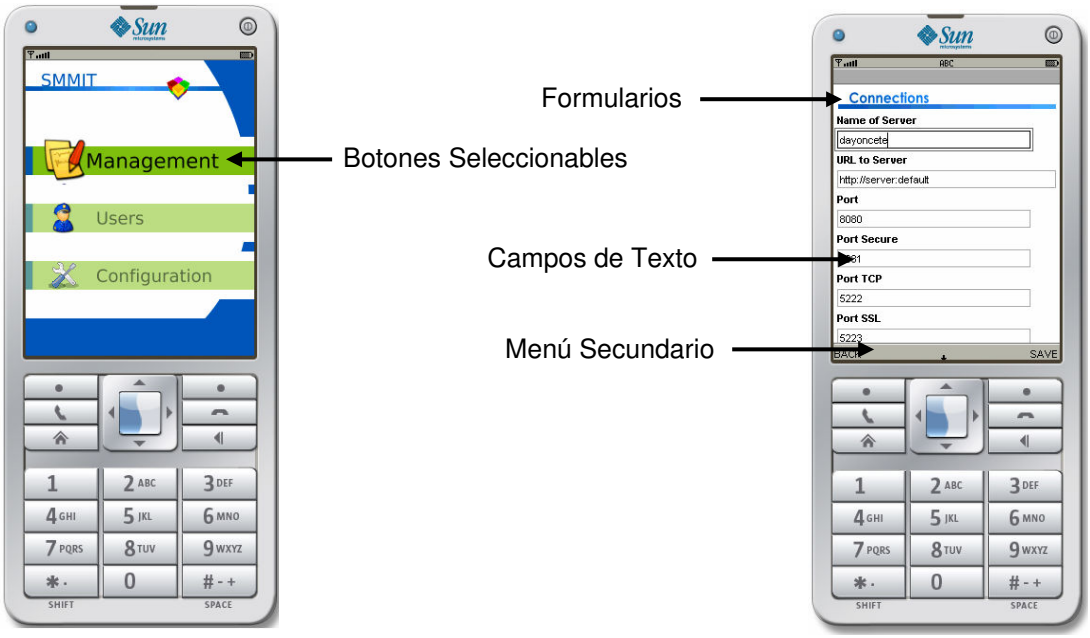
A.1 MANUAL DE USUARIO

INDICE DE CONTENIDOS

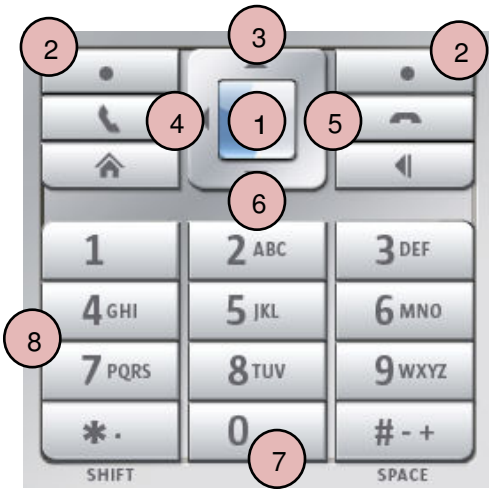
M.1 DESCRIPCIÓN

SMMIT (Sistema Multimedia para Móviles de Incidencias de Tráfico) es una aplicación Java diseñada para dispositivos móvil que puedan soportar esta plataforma. Está destinada a ser utilizada como una herramienta de apoyo para agentes de policía que necesiten una máxima precisión elaborando sus informes de atestados, accidentes, incidentes, altercados, etc. La novedad de este sistema es que utiliza prácticamente todas las funcionalidades de los móviles como por ejemplo su cámara de fotos o GPS y como forma de

comunicación hace uso de un sistema llamado Jabber. Así mismo cuenta con una interfaz sencilla e intuitivamente fácil de controlar por lo que no se necesita un excesivo aprendizaje y hay que añadir que gracias a sus gráficos vectoriales se puede visualizar en cualquier pantalla sin perder calidad.



M.2 CONTROLES



1 Confirma selección

2 Selecciona la opción que se visualice arriba

3 Mueve/Selecciona hacia arriba una opción . o texto

4 Vuelve al menú principal
Desplaza el cursor de texto hacia la izquierda

5 Desplaza el cursor de texto hacia la derecha

6 Mueve/Selecciona hacia abajo una opción o texto

7 Cierra la aplicación

M.3 UTILIZACIÓN

El manejo básico de esta aplicación consiste en seleccionar determinadas opciones e introducir datos por los botones del móvil como si de un SMS se tratase. Cualquier acción errónea o no permitida se informará mediantes mensajes de alerta informativos.

M.3.1 Puesta en Funcionamiento

En las aplicaciones del móvil elegimos nuestra aplicación, SMMIT, y pulsamos en aceptar, veremos cómo seguidamente sale una ventana de carga.

Figura 8.1 Ventana de Carga de la Aplicación

Cuando el sistema acabe de cargarse saldrá el menú principal y a partir de aquí ya tendremos nuestra aplicación funcionando. A continuación explicaremos las diferentes secciones.

M.3.2 Configuración

Si seleccionamos la opción Configuración nos saldrá un submenú interactivo donde podemos elegir el tipo de cambios que queremos hacer: de conexiones, del propio móvil o de la aplicación en general.

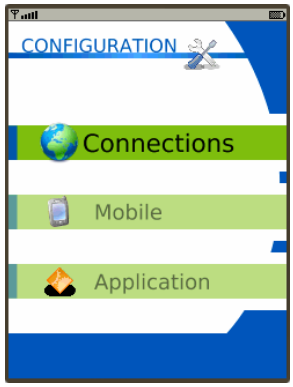


Figura 8.2 Menú de Configuración

M.3.2.1 Conexiones

Aquí podemos definir manualmente el tipo de conexión que queremos y por qué puertos se realizará. Los campos que contiene son:

- **Name of Server**: Nombre del servidor Jabber
- **URL to Server**: Dirección completa del servidor Jabber
- **Port**: Puerto de conexiones HTTP
- **Port Secure**: Puerto de conexiones HTTP seguro
- **Port TCP**: Puerto de conexiones TCP
- **Port SSL**: Puerto de conexiones SSL
- **Connection's Options**: Opciones de conexión
- **Security's Options**: Si activamos la seguridad SSL

M.3.2.2 Móvil

Selección de la resolución principal del móvil que contiene la aplicación. Se especifica el ancho **Width** y el alto **Height.**

M.3.2.3 Aplicación

Configuración de todo lo relacionado con la creación de informes multimedia. Podemos definir la resolución de las fotos y del video y la precisión del GPS.

- **Width Photo**: Resolución de la foto, ancho.
- **Height Photo**: Resolución de la foto, alto.
- **Width Video**: Resolución del video, ancho.

- **Height Video**: Resolución del video, alto.
- **Accurecy Horizontal:** Precisión GPS horizontal
- **Accurecy Vertical**: Precisión GPS vertical
- **TimeOut**: Tiempo de espera para la conexión GPS

M.3.3 Usuarios

Podemos crear y eliminar usuarios que pueden utilizar el móvil así como seleccionar el que va a utilizar la aplicación en un momento dado. La interfaz deja bien claro las tres opciones:

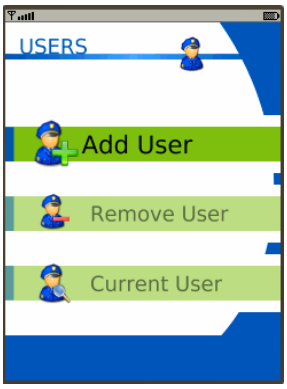


Figura 8.3 Menú de gestión de usuarios

Añadir un Usuario

Se introducen los datos del usuario, contraseña y salvamos, la información será recibida por el servidor quien confirmará que el usuario ha sido registrado.

Figura 8.4 Formulario para agregar a un nuevo usuario

Eliminar un Usuario

Seleccionamos un usuario existente en nuestra base de datos RMS y eliminamos.

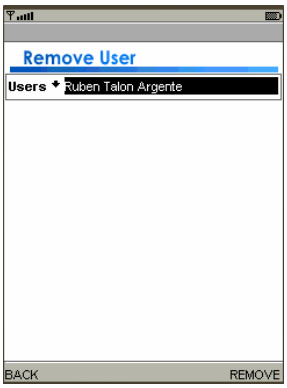


Figura 8.5 Formulario para eliminar a un usuario

Seleccionar Usuario Actual

Seleccionamos el usuario que va a tomar el control de la aplicación en ese momento.

Figura 8.6 Formulario para seleccionar al usuario actual

M.3.4 Gestión de Informes

En la gestión de informes tenemos tres posibilidades: Crear uno nuevo, ver registros guardados y visualizar una especie de historial de utilización de la aplicación.

Figura 8.7 Opciones de Gestión de Incidentes

M.3.4.1 Ver Historial

Se visualiza en pantalla un histórico de todas las acciones producidas en el móvil. Se especifica la fecha y la hora cuando se realizó un evento (inicio de aplicación, guardado de ficheros, eliminación de usuarios…).

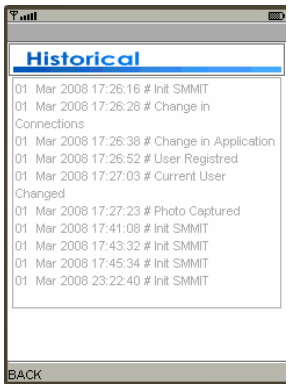


Figura 8.8 Histórico de la aplicación

M.3.4.2 Ver Archivos Guardados

Como si se tratase de un explorador de ficheros similar al de un sistema operativo podemos desplazarnos por la memoria física del móvil buscando archivos creados para visualizarlos por pantalla. Por defecto los archivos se guardan en el directorio **filesystem\\root1.**

Figura 8.9 Árbol de directorios de un móvil

El menú inferior dispone de las opciones **BACK** que vuelve hacia atrás en la ruta de directorios y **OPEN** que abre tanto una carpeta como un archivo de texto.

M.3.4.3 Crear un nuevo incidente

Para crear un nuevo incidente tenemos tres posibilidades que consisten en capturas fotos, grabar un video y escribir el reporte de forma digital, además luego podremos guardarlo y enviarlo en el mismo instante.

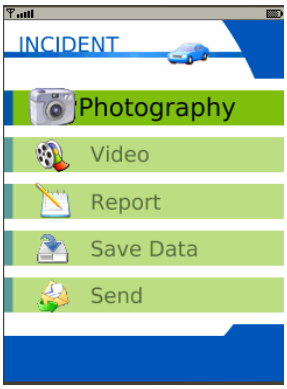


Figura 8.10 Menú de elementos multimedia

M.3.4.3.1 Capturar una Fotografía

Cuando seleccionemos esta opción veremos cómo la cámara de nuestro móvil se activa y estará habilitada la opción de capturar en el menú inferior.

Figura 8.11 Activación de la Cámara de Fotos

En estos momentos es cuando podemos capturar instantáneas, todas las que queramos y nos permita el móvil. Cuando ya tengamos las que necesitemos en el mismo menú seleccionaremos la opción **SAVE.**
Si no se pudiera realizar las fotos bien por la resolución escogida o por otras causas el sistema nos lanzará un mensaje de advertencia.

M.3.4.3.2 Grabar un Video

Es el mismo sistema usado en la captura de fotos solo que ahora tenemos la posibilidad de grabar los videos que deseemos y guardarlos. Los comandos son los básicos: grabar y detener.

M.3.4.3.3 Escribir un Informe

Aparte de poder escribir libremente un informe personalizado por el usuario podemos seleccionar varias opciones creadas por defecto y que son las siguientes:

- **Agent**: Usuario actual. Este campo se rellena automáticamente y no puede ser editado.
- **Data Incident**: Fecha de creación del incidente. Tampoco se puede modificar
- **Time Incident**: Hora en la que se produjo el incidente.
- **Country**: País.
- **Location**: Localización del incidente.
- **Type Incidet**: Entre 4 posibilidades podemos escoger el tipo de accidente.
- **Nature Injuries**: Naturaleza u origen de las heridas o lesiones.
- **Nº Vehicles**: Número de vehículos involucrados.
- **Location Vehicle**: Lugar donde está situado en estos momentos el vehículo siniestrado.
- **Characteristics**: Más detalles del suceso: muertos, carriles bloqueados, víctimas atrapadas…
- **Details**: Detalles de cómo se produjo el accidente: velocidad, condiciones…
- **Risk Factores**: Factores de riesgo como fuego, sustancias peligrosas o condiciones meteorológicas.

M.3.4.3.4 Guardar Informe

Una vez creado el informe tenemos la posibilidad de guardarlo en un archivo lógico en la propia memoria del móvil para posteriormente visualizarlos o descargarlo.

Figura 8.12 Formulario de Guardado de Ficheros

M.3.4.3.5 Enviar Informe

Si damos a esta opción la aplicación se conectará automáticamente con la central y si la conexión se realiza saldrá una ventana de información adicional como por ejemplo el número de fotos realizadas, videos, etc.
Si todo es correcto y la central nos envía una identificación del incidente podemos seleccionar **SEND** y enviarlo.

M.4 MENSAJES Y ALERTAS

Mensajes de Alertas que pueden ir surgiendo en el transcurso de la ejecución de la aplicación.

¡Warning! User Not Selected	Aviso, no se ha seleccionado el usuario actual que está utilizando la aplicación
¡Error! Failed Connection	Error, conexión con el servidor errónea o no se puede conectar.
¡Error! Resolution Not Supported	Error, la resolución actual no está permitida en este móvil.
¡Error! User Not Registered	Error, usuario no registrado en el servidor.
¡Successful! Received Identification	Identificación de informe recibida del servidor central.
¡Successful! Registered User	Usuario registrado con éxito.
¡Successful! Saved Report	Informe guardado con éxito.
¡Successful! Sent Incident	Informe enviado con éxito.

¡Successful! Established Connection	Conexión con el servidor realizada correctamente.
¡Wait! Connecting	Conectando con el servidor Jabber

M.5 SOLUCIÓN DE PROBLEMAS

Durante el inicio o el transcurso de la ejecución de la aplicación pueden surgir errores inesperados o avisos que no comprendemos. Se listan una serie de cuestiones y su posible solución.

Problema	Posibles Causas	Posibles Soluciones
No se inicia la aplicación	No hay soporte Java No hay soporte para alguna API Instalación defectuosa	Instalación en otro móvil Verificar si tu móvil dispone de las APIs necesarias Asegúrate que cuando instalas dispones de los archivos JAD y JAR
Lentitud de la interfaz	Móvil con pocos recursos	Aunque la aplicación está optimizada al máximo es posible que en ciertos móviles se aprecie una lentitud molesta de la aplicación. La mejor solucionar es instalar el software en un móvil con mayor capacidad de cómputo.
Interfaz pequeña con respecto a la pantalla del móvil	Mala configuración de la aplicación	Hay que seleccionar en configuración la opción móvil y especificar en Width y Height las dimensiones exactas del móvil.
No se inicia la cámara del móvil	API no soportada Cámara iniciada por otra aplicación	Verificar que se dispone de la API JSR-135 Cerrar otras aplicaciones que usen la cámara
No se puede capturar la foto	Resolución no permitida por el móvil Memoria agotada	Cambiar la configuración de la aplicación a una resolución permitida Dependiendo del móvil se podrán hacer más o menos fotos
No se puede grabar video	No todos los móviles soportan Video Resolución no permitida	A pesar de que se disponga de la API adecuada muchos móviles no soportan la grabación de video en tiempo real.
No se puede leer un fichero	Acceso no permitido	La estructura interna de algunos móviles impiden ver los archivos
No se establece la conexión	Puertos incorrectos Nombre o IP del servidor incorrecto Servidor no disponible Usuario no seleccionado	Entrar en configuración de conexiones y cambiar los puertos Entrar en configuración de conexiones y cambiar el servidor Esperar a que el servidor esté operativo En el menú usuarios seleccionar a un usuario actual
No se envía el informe	Informe demasiado grande Conexión con el servidor perdida	Puede ocurrir si se hacen demasiadas fotos y no hay memoria. Volver hacia atrás y volver a enviar para que vuelva a conectar.
No se guarda el fichero	Espacio insuficiente en el móvil No hay permiso para acceder	Liberar espacio físico en la memoria del móvil El sistema de archivos del móvil es inaccesible

A.2 INDICE DE FIGURAS

A.3 INDICE DE TABLAS

A.4 ESPECIFICACIONES TÉCNICAS DEL NOKIA N95

Pantalla y sonido

La pantalla es de 2.6 pulgadas y 16 millones de colores. Desplazándola hacia arriba se ve el teclado y desplazándola hacia abajo aparecen unas teclas multimedia y automáticamente la pantalla se visualiza en horizontal. Aparece el menú multimedia con opciones como Galería, RealPlayer, Reproductor de música, Visual Radio, etc. Para desplazarnos por estas funciones, se usan las teclas principales de la tapa. Para el sonido (estéreo), cuenta con dos altavoces en los laterales de la unidad. Se pueden conectar audífonos estándar directamente al teléfono o al dispositivo manos libres, ya que la conexión es del tipo 3,5 mm...

Cámara

Posee dos cámaras, la principal de 5 megapíxeles y lentes Carl Zeiss y la secundaria frontal, de menor resolución para videoconferencia. La cámara principal se visualiza en horizontal. Las fotos guardadas se pueden transmitir a una impresora compatible o a un kiosco de impresión mediante la tecnología Bluetooth, una tarjeta de memoria, una LAN inalámbrica (UPnP) o directamente a una impresora compatible con PictBridge a través de un cable USB. También se pueden exhibir en un televisor o a través de un proyector mediante un cable incluido con el teléfono. También captura videos en calidad DVD (según el fabricante) con sistema de auto estabilización de la cámara. Como curiosidad, la cámara se puede utilizar para leer código de barras del tipo Block Matrix.

Batería

- Batería: BL-5F
- Capacidad Batería: Batería de Ion Litio 950mAh BL-5C
- Tiempo de conversación: hasta 190 min. (WCDMA), hasta 390 min. (GSM)
- Tiempo de espera: hasta 233 horas (WCDMA), hasta 228 h. (GSM)
- La autonomía de las baterías puede variar en función de la tecnología de acceso de radio, la configuración de la red del operador y el uso.

Cobertura

Cobertura WCDMA HSDPA 2100 y EGSM 850/900/1800/1900 MHz Cambio automático de una banda a otra y de un modo a otro

Radio

Es posible escuchar música e interactuar con emisoras de radio. Además, aprovechando las capacidades de conectividad, incorpora *Visual Radio*, característica con la que se permite el acceso a información relativa a la canción que se está emitiendo, de su cantante, y otros datos de interés relacionados.

GPS

Este teléfono posee un dispositivo GPS integrado y, a partir de la versión 12.00.013 del firmware, se añade funcionalidad A-GPS o GPS Asistido. Incluye el navegador de carreteras nokia maps, la posibilidad de descargar los mapas de forma gratuita vía internet y, mediante suscripción de pago, guías y voces en varios idiomas para la navegación

Acelerómetro

N95 incluye un acelerómetro. Originalmente se usaba sólo para la estabilización del vídeo y la orientación de las fotos (para mantener las fotos orientadas en vertical u horizontal tal cual fueron tomadas).

Una vez que Nokia Research Center permitió interactuar directamente con el acelerómetro, para hacer uso de sus datos, empezaron a aparecer aplicaciones de terceros como RotateMe, la cual cambia la orientación del contenido de la pantalla de forma automática cuando el teléfono se inclina, y Lightsaber, la cual provoca que el teléfono emita sonidos del sable de luz de la Guerra de las Galaxias cuando el teléfono se mueve.

Internet

El acceso a internet se puede realizar, bien a través de las posibilidades 3G que ofrecen las operadoras o mediante la conectividad Wi-Fi integrada en el teléfono. Se puede hacer prácticamente lo mismo que con un navegador para un ordenador, pero teniendo en cuenta las limitaciones existentes para internet móvil-Pda.

Conectividad

- LAN inalámbrica integrada (802.11 b/g) y UPnP (Universal Plug and Play)
- Tecnología Bluetooth integrada v.2.0
- USB 2.0 a través de interfaz Pop-Port™ (cable mini USB estándar), compatible con la clase de almacenamiento masivo con función “arrastrar y soltar”
- Toma de 3,5 mm para auriculares estéreo y salida de TV (PAL/NTSC)
- Conexión mediante Nokia PC Suite con USB, infrarrojos y tecnología Bluetooth
- Sincronización local de los contactos y la agenda con un PC por medio de una conexión compatible
- Sincronización inalámbrica a distancia
- Envía y recibe imágenes, secuencias de vídeo, gráficos y tarjetas de visita a través de la tecnología Bluetooth

Memoria

Memoria NAND de 256 MB, Memoria SDRAM de 64 MB (para aplicaciones en ejecución), Memoria flash de 160 MB para almacenamiento (mensajes, tonos de llamada, imágenes, secuencias de vídeo, notas de la agenda, lista de tareas, aplicaciones, etc...)

Memoria ampliable: ranura para tarjeta de memoria microSDHC de hasta 8 GB (incluye una tarjeta microSD de 1 GB)

A partir de la versión 20.0.015 del firmware, se implementa la memoria de intercambio (swap) que, haciendo uso de la Memoria flash , proporciona mayor espacio libre en la memoria SDRAM, mejorando así el rendimiento de las aplicaciones.

Funciones básicas

Admite video-llamada, multiconferencia, PPH, número fijo de marcación (sólo permite las llamadas a los números predefinidos). En cuanto a la mensajería de texto admite SMS concatenados, postales electrónicas y listas de distribución de SMS. Para la mensajería multimedia combina imágenes, vídeo, texto y secuencias de audio para enviarlas como MMS a un PC o teléfono compatible.

Funciones para el trabajo

Además del correo, lee documentos en formato .doc, .xls, .pps y PDF y dispone de compresor de archivos ZIP. Dispone de función de copia de seguridad del contenido de la memoria principal en la tarjeta de expansión o alternativamente hacia el PC conectado.

Revisiones

N95 8GB (N95-2)

El 29 de agosto de 2007 se anunció una revisión del N95, llamada N95 8GB (N95-2 internamente conocido como RM-320).

Los cambios en comparación con la versión regular son:

- Memoria flash interna de 8 GB
- La pantalla es algo más grande (de 2.6" pasa a 2.8"), manteniendo la misma resolución 240x320 pixels.
- Se elimina la ranura para MicroSD.
- Memoria SDRAM se amplía de 64 a 128 MB
- Batería de 1200mAh (BL-6F)
- Se elimina el protector de la lente de la cámara para dar cabida a la batería más grande.

- Software para GPS en español. (Actualización para México y España)
- Flash Lite 3.0
- Teclas frontales más pequeñas

N95-3

Nokia N95-3 es la revisión del N95 diseñado específicamente para el mercado norteamericano (internamente conocido como RM-160). También está disponible en Australia en la red NextG de Telstra. El resto de características es similar al N95-2[2] . El n95-3 muestra algunas variaciones como la duración de batería de 1200 mAh comparada a los 950 mAh del primer N95.

A.5 RESUMEN DE LAS APIs MÁS UTILIZADAS

JSR	Descripción
JSR-135	También conocido como MMAPI contiene los paquetes multimedia para imagen y sonido
Referencia	**http://jcp.org/en/jsr/detail?id=135**

JSR	Descripción
JSR-179	API que gestiona el paquete Location encargado de la utilidad de GPS
Referencia	**http://jcp.org/en/jsr/detail?id=179**

JSR	Descripción
JSR-172	Procesos básicos de parseado XML
Referencia	**http://jcp.org/en/jsr/detail?id=172**

JSR	Descripción
JSR-75	API para la gestión RMS y ficheros residentes en los propios dispositivos
Referencia	**http://jcp.org/en/jsr/detail?id=75**

JSR	Descripción
JSR-226	API para Gráficos Vectoriales SVG
Referencia	**http://jcp.org/en/jsr/detail?id=226**

JSR	Descripción
JSR-238	API para la internacionalización de Aplicaciones MIDP
Referencia	**http://jcp.org/en/jsr/detail?id=238**

JSR	Descripción
JSR-234	Define un paquete opcional de funcionalidades multimedia avanzadas para el MMAPI
Referencia	**http://jcp.org/en/jsr/detail?id=234**

A.6 ACRÓNIMOS Y SIGLAS

3G	3rd Generation Partnership Project
AAC	Advanced Audio Coding (AAC)
ACM	Modelo de Composicion de Aplicaciones
AMR	Adaptive Multi-Rate
API	Application Programming Interface
ATT	American Telephone and Telegraph
AWT	Abstract Windowing Toolkit
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
EDM	Modelo de Diseño Inicial
GCJ	GNU Compiler for Java
GIF	Compuserve GIF o Graphics Interchange Format
GIS	Sistema de Información Geográfica
GNU	Acrónimo recursivo - GNU No es Unix
GPG	GNU Privacy Guard
GPL	Licencia pública general de GNU
GPRS	General Packet Radio Service
GPS	Sistema de Posicionamiento Global
GSM	Sistema Global para las Comunicaciones Móviles
HTTPS	protocolo de transferencia de hipertexto seguro
ICQ	"IGUAL COMO QUIEN " o "te busco"
IDL	Interfaz para Descripción del Lenguaje
IETF	Internet Engineering Task Force
IRC	Internet Relay Chat
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAAS	Java Authentication and Authorization Service
JCE	Java Cryptography Extension
JCP	Java Community Process
JDK	Java Development Kit
JIT	Just in Time
JLS	Java Language Specification
JNI	Java Native Interface
JPDA	Java Platform Debugger Archictecture
JPG	Joint Photographic Experts Group

JSR	Java Specification Requests
JSSE	Java Secure Socket Extension
LCD	Pantalla de cristal líquido - Liquid crystal display
MIDP	Perfil para Dispositivos de Información móvil
MMAPI	Mobile Multimedia API
MP3	MPEG-1 Audio Layer 3
PAM	Modelo de Post-Arquitectura
PDA	Personal Digital Assistant
PERL	Practical Extraction and Report Language
PGP	Pretty Good Privacy
PHP	PHP Hypertext Pre-processor
PNG	Portable Network Graphics
PSP	PlayStation Portable
RMI	Java Remote Method Invocation
RMS	Record Management System
SASL	Simple Authentication and Security Layer
SMTP	Simple Mail Transfer Protocol
SOA	Servicio Orientado a Arquitectura
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UML	Unified Modeling Language
UMPC	Personal Computer Ultra Móvil
UMTS	Sistema Universal de Telecomunicaciones móviles
VCR	Video Cassette Recorder
VM	Maquina Virtual
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access
WCS	Web Coverage Service
WFS	Web Feature Service
WMI	Windows Management Instrumentation
WMS	Servicio Web Map Service
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

A.7 DESCRIPCIÓN DE LAS CLASES MÁS IMPORTANTES

CommunicationManager	
Atributos	***Descripción***
MidletEventListener midlet;	Midlet encargado de recoger eventos
meConnector cinit;	Inicio de conexiones TCP
IWriterThread writerThread;	Hilo de escritura de flujos XML
StanzaReader stanzaReader	Reader de elementos Stanza
StreamConnection inConn;	Variable de conexión TCP
Constructores	
CommunicationManager(_midlet)	Constructor con parámetro _midlet
Métodos	
connect(int state)	Conexión TCP
httpConnect()	Conexión HTTP
disconnect()	Termina la ejecución del hilo de conexión
nodeStart(Node _node)	Primer nodo que enviamos al servidor
nodeEnd(Node _node)	Ultimo nodo y fin de sesión
notifyConnect(_inConn,_is, _os)	Notifica conexiones TCP

RMSData	
Atributos	***Descripción***
RecordStore rs	RMS de usuarios
RecordStore historical	RMS del histórico
Constructores	
RMSData()	Constructor sin parámetros RMSData
Métodos	
openRMS()	Abre la base de datos de usuarios
openRMSHistorical()	Abre la base de datos del histórico
sizeRMS()	Número de registro de la RMS de usuarios
closeRMS()	Cierra la base de datos de usuarios
closeRMSHistorical()	Cierra la base de datos del histórico
deleteRegRMS(int id)	Borra el registro identificado por id
getRegHistorical()	Devuelve una cadena con todos los registros
addDataRMS(String data)	Añade los datos 'data' al registro de usuarios
addRegHistorical(String data)	Añade los datos 'data' al registro de histórico
getAllRegs()	Devuelve un vector de String de usuarios
getAllIds()	Devuelve todos los identificadores de los usuarios
getAllNameUsers()	Devuelve todos los nombre de usuarios
getIdByNameUser(String name)	Devuelve el identificador por nombre de usuario

Jid

Atributos	*Descripción*
String servername	Nombre del Servidor Jabber
String username	Nombre del Usuario Jabber
String presence	Estado del Usuario
String resource	Aplicación que utiliza Jabber
String mail	Correo del Usuario
Constructores	
Jid(String _jid)	Constructor con parámetro Jid
Jid(String _jid, String presence)	Constructor con parámetro Jid y presencia
Métodos	
String getLittleJid()	Devuelve el Jid sin el resource
String getFullJid()	Devuelve el Jid completo
String getLittleJid(String _jid)	Devuelve el Jid pasado sin el resource

Jud

Atributos	*Descripción*
Métodos	
setRegistration(Jid jid, String name, String	Registra un usuario Jid
getJidfromResponse(Node node)	Devuelve el Jid de una respuesta
searchJid(Vector params, String jud_server)	Busca el Jid en un servidor Jabber

MidletEventListener

Atributos	*Descripción*
Display display	Muestra la pantalla activa principal
Hashtable infopool;	Tabla Hash de información
MainConnection mainConn;	Contiene las variables de inicio de conexión
Constructores	
MidletEventListener(_mainConn)	Constructor con parámetro _mainConn
Métodos	
connectedEvent()	Evento de conexión establecida
unauthorizedEvent(String reason)	Evento de conexión de autorizada
disconnectedEvent()	Evento de desconexión
notifyError(_conversation, _errorMessage)	Notificación de error
newConversationEvent(_conv)	Nueva conversación entrando
newMessageEvent(Conversation _conv)	Evento nuevo mensaje
notifyPresence(Jid _roster, String _presence)	Notificar presencia Jabber
notifyRoster()	Notificar usuario conectado

MDate	
Atributos	***Descripción***
String date	Fecha y Hora completa actual
Constructores	
MDate(String _date)	Constructor con parámetro MDate
Métodos	
getTime()	Devuelve la hora actual
getDateLarge()	Devuelve la fecha en formato completo DD/MM/AA
intoPiecesDate()	Devuelve la fecha estructurada en partes

SMMIdlet	
Atributos	***Descripción***
Int numScreens	Número de pantallas
Components smmitComponents	Componentes de la aplicación
ScreenCanvas canvasRT	Pantalla visualizada en tiempo real
ScreenCanvas canvasMemory[]	Vector con todas las pantallas en memoria
SVGImage svgB[], svgM[]	Flujo de datos de gráficos SVG
String toSend	Cadena de texto que se enviara al servidor
String toSave	Cadena de texto para guardar en archivo
DataMain dataMain	Datos generales de la aplicación
MainConnection mainConn	Objetos para la conexión Jabber
Constructores	
startApp()	Inicio del Midlet
Métodos	
initElements();	Inicialización de variables y objetos
createInterfaces();	Creación del vector de memoria de screens
loadStreamToSVG();	Cargamos los Streams en gráficos SVG
changeOfScreen(String nextScreen)	Cambia la pantalla de gráficos SVG
changeOfForm (String nextForm)	Cambia el formulario
prepareReport()	Crea el informe para enviar
prepareSaveFile()	Crea el informe para guardar

Util	
Atributos	***Descripción***
Constructores	
Métodos	
sha1(String _str)	Devuelve una cadena codificada en sha
escapeCDATA(String _str)	Escapa el valor de un xml CDATA
formatGtwAddress(String jid)	Devuelve el formato a Gateway address

MenuSVG

Atributos	*Descripción*
int numButtons	Número de opciones que tiene el menú
String svgFile	Nombre del fichero que contiene el menú
String idMenu	Identificación del menú

Constructores	
MenuSVG()	Constructor sin parámetros
MenuSVG(_numButtons, _svgFile, _idMenu	Constructor con el Nº de opciones, fichero e Id

Métodos

ScreenCanvas

Atributos	*Descripción*
SMMidlet midlet	Acceso al midlet principal
String bgImage	Nombre de la imagen de fondo
SVGImage svgBg	Imagen SVG de fondo
SVGImage svgMenu	Imagen SVG de menú
ScalableGraphics sg, sg2	Objetos de la clase base gráficos escalares
int numFrames	Numero de frames de la animación
float frameLength	Tiempo entre frames
float padding	Espaciado de los botones de menú
int[] currentFrame	Vector con el frame en visualización
MenuSVG menuSVG	Objeto de la clase MenuSVG
InputStream streamMenu	Flujo de datos del menú
InputStream streamBg	Flujo de datos de la imagen de fondo
Thread th	Hilo del redibujado
boolean renderizado	Comprobación de redibujado
Image[][] menuIcons	Vector con todos los frames de la animación
int dimWidth	Anchura del menú SVG
int dimHeight	Altura del menú SVG

Constructores	
ScreenCanvas(SMMidlet _midlet)	Constructor con parámetro midlet

Métodos	
processMenu(SVGImage svgBg, SVGImage	Crea un menú SVG con un fondo
createThread()	Crea el hilo de redibujado
keyPressed(int keyCode)	Controla el menú por teclado
paint(Graphics g)	Función que hace el redibujado en pantalla

Users

Atributos	*Descripción*
String nameUser	Nombre del usuario
String surnameUser	Apellidos del usuario
String idUser	Identificación del usuario
String nick	Nick del usuario
String password	Password del usuario

Constructores	
Users()	Crea un usuario por defecto
Users(name, surname, id, pass))	Constructor con parámetros

ScreenCanvas

Atributos	*Descripción*
Jid currentjid	Identificación Jabber actual
Hashtable infopool	Información de conexiones en tablas Hash
int internal_state	Estado actual de la conexión
SMMidlet midlet	Midlet principal
CommunicationManager cm	Objeto CommunicationManager
MidletEventListener listener	Listener de eventos

Constructores	
MainConnection(SMMidlet midlet)	Constructor con midlet

Métodos	
initConnectionJabber()	Inicializa variables de conexión
beginConnection()	Comienza a establecer la conexión con el servidor

MFile

Atributos	*Descripción*
Vector roots	Vector con la localización de los roots del directorio
FileConnection currentRoot	Objeto que establece unión con el móvil
Thread write	Hilo para visualizar contenidos
Thread task	Hilo para escribir un archivo

Constructores	
MFile()	Constructor sin parámetros

Métodos	
stop()	Cerramos la conexión AMS y liberamos recursos
rootChanged(int state, String rootName)	Cambiamos el directorio donde estamos
deleteAll()	Borra todos los elementos en pantalla
loadRoots()	Carga roots actuales
openSelected()	Abre una carpeta o fichero
showTextFile(String url)	Muestra el texto en pantalla
readFully(DataInputStream input)	Carga en memoria la información del fichero
displayCurrentRoot()	Muestra el estado actual del directorio
showItems(Vector items)	Muestra los elementos en pantalla
writeFile (final String path, final byte[] data)	Escribe un fichero en la memoria física

Presence	
Atributos	***Descripción***
String[] string_presence	Modos básicos de presencia de Jabber
Constructores	
Métodos	
changePresence(_presence, message)	Cambia el modo de presencia en el Servidor
String getPresence(int _presence)	Devuelve el modo de presencia actual

StanzaReader	
Atributos	***Descripción***
String stanzaId	Identificador del mensaje Stanzas
String stanzaType	Tipo de mensaje Stanzas
String stanzaFrom	Quien envía el mensaje
String stanzaTo	Hacia quién va dirigido
ExceptionListener exceptionListener	Listener de excepciones
JabberListener jabberListener	Listener de Jabber
Constructores	
StanzaReader(_exceptionListener,…	Constructor con parámetros
Métodos	
read(Node _node)	Lee un mensaje especifico recibido
readIq(Node _node)	Lee un mensaje del tipo IQ
readPresence(Node _node)	Lee un mensaje del tipo Presence
readRoster(Node _node)	Lee mensaje tipo Roster
readMessage(Node _node)	Lee cualquier mensaje proveniente del servidor

A.8 IMÁGENES DE SMMIT EN UN MÓVIL REAL (NOKIA N95)

Imágenes adjuntas de cómo SMMIT (Sistema Multimedia para Móviles de Incidencias de Tráfico) trabaja en un móvil real, que en este caso se trata del Nokia N95 que propusimos para su utilización.

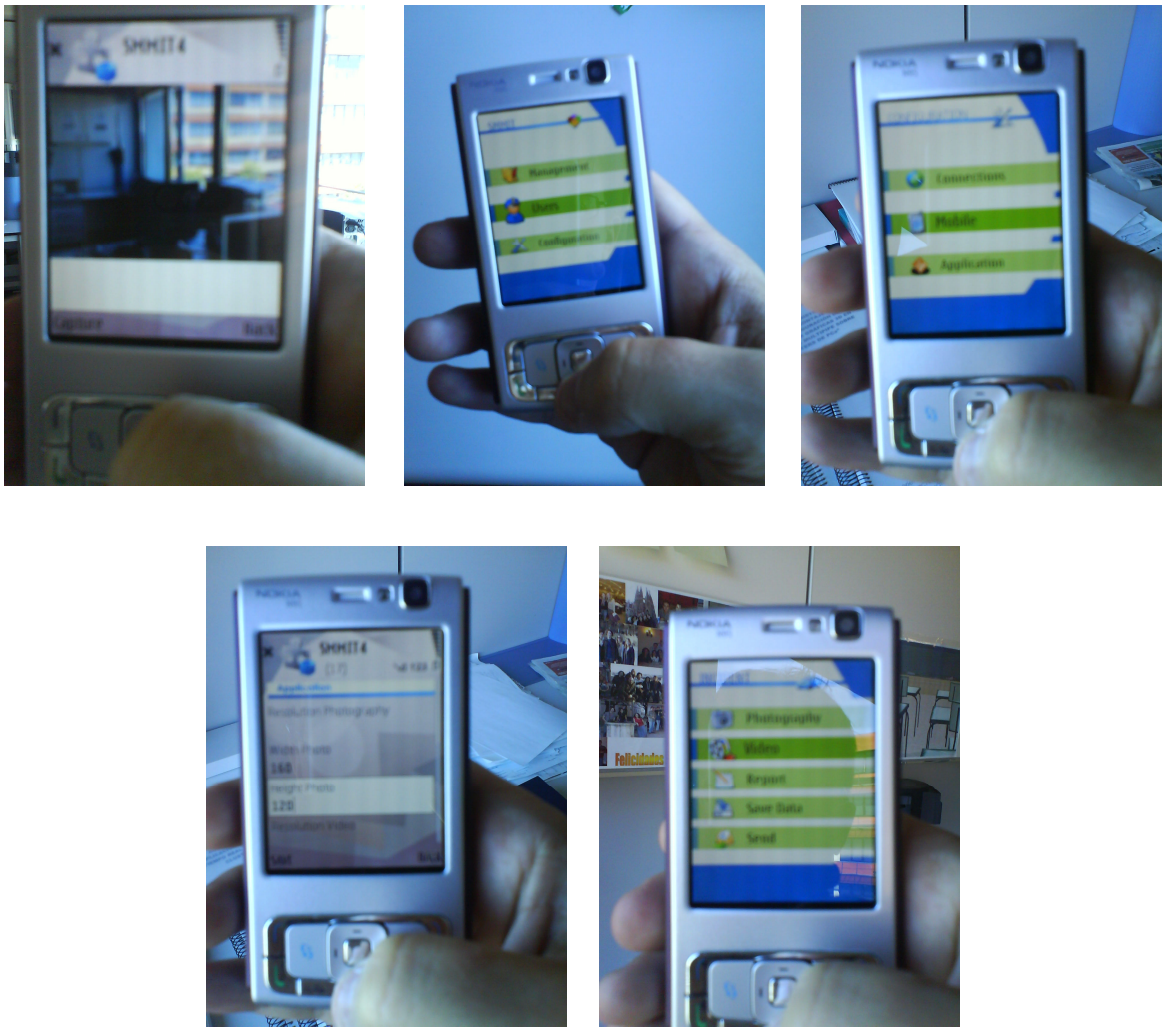


Figura A.1 Imágenes de SMMIT en funcionamiento en el Nokia N95

A.9 PROPUESTA DE PROTOCOLO DE INTERVENCIÓN POLICIAL CON SMMIT

Actualmente todos los sistemas de seguridad se rigen por unas reglas o protocolos que se deben de seguir a rajatabla para llevar a cabo una determinada acción. La aplicación que hemos desarrollado es una herramienta que será utilizada por agentes del orden en su labor de asegurar el bienestar y seguridad de las personas, es por ello que dicha herramienta necesite tener unas pautas de utilización.

Tomando como ejemplo los protocolos de intervención policial de accidentes de tráfico urbano **[PRO06]** propondremos un protocolo de utilización de SMMIT. Dicho protocolo está desarrollado pensando principalmente en la seguridad de las personas y sólo utilizaremos la aplicación cuando la situación esté controlada o cuando se necesite describir lo sucedido con urgencia.

El inicio del protocolo se produce inmediatamente después de llegar al lugar del suceso donde se evaluará la situación (leve, moderada, grave, muy grave) y, según el tipo, se procederá a realizar las diferentes acciones. A modo de resumen si la situación del incidente no es grave y no hay vidas en juego se procederá a despejar y señalizar el lugar del suceso. Después de restablecer el tráfico (si éste estuviera parado) se procederá a tomar parte del incidente y enviarlo a la central.
Si por otro lado la situación se considera grave antes que nada hay que socorrer a los heridos (si los hubiera) y valorar si necesitan atenciones médicas por lo que inmediatamente llamaríamos a los servicios de urgencias. Si no se pudieran atender a las víctimas por razones como estar atrapadas, mal heridas, inconscientes, etc. se procedería a realizar un informe con fotografías que describan lo ocurrido para posteriormente enviarlas a la central y luego enviarlas a los servicios de urgencias.

Cuando se estabilice la situación de las victimas hay que preocuparse luego de controlar la situación y que no adquiera mayores proporciones. Si no se puede controlar es necesario hacer un rápido informe que, al igual que con las víctimas, describa el incidente y de esta forma decidir si se mandan servicios de bomberos, grúas, etc.

Tanto si la situación es grave o leve al final hay que realizar un informe detallado con todo lo necesario y enviarlo a la central para su gestión.

Viendo la **Figura A.2** se puede tener una idea de cómo y cuándo utilizar la aplicación

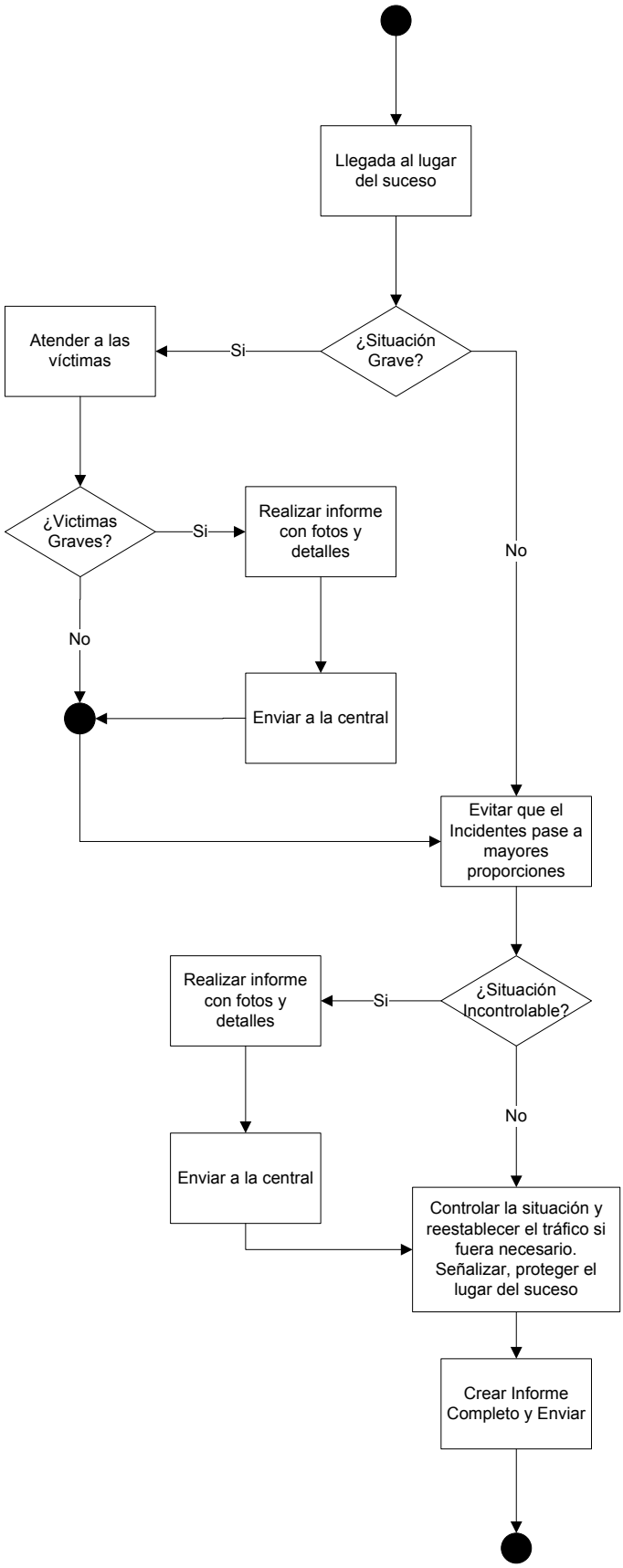


Figura A.2 Propuesta de Protocolo de Utilización de SMMIT

A.10 CONTENIDO DEL CD ADJUNTO

Junto con la memoria se adjunta un CD cuyo contenido es el siguiente:

- Carpeta **Proyecto Final de Carrera** donde hay dos subcarpetas que contienen una de ellas el código con comentarios en consola y otra con comentarios eliminados.
- Carpeta **Documentación** con la presente memoria.
- Carpeta **Diagramas** con proyectos en Microsoft Visio y Visual Paradigm.
- Carpeta **Cliente y Servidor Jabber** con los programas utilizados para las pruebas.
- Carpeta **NetBeans** con el entorno de programación Java empleado.
- Carpeta **Imágenes** con algunas imágenes reales del proyecto.
- Carpeta **Otros** con manuales y otros documentos.