UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

UNIVERSIDAD POLITÉCNICA DE VALENCIA
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA APLICADA

Desarrollo de un sistema integral de gestión de noticias exclusivo para teléfono móvil.

PROYECTO FIN DE CARRERA

Autor-es

Ángel Soler Cabañas.

Director-es

Diego Álvarez Sánchez

Fecha del proyecto

09 de Febrero de 2010

Código del PFC

DCADHA-3-159-B/08

2- Resumen de la memoria.

El proyecto realizado consta de una aplicación móvil en la que podremos ver y subir noticias relacionadas con la Universidad Politécnica de Valencia. El proyecto va dirigido a toda la comunidad universitaria, tanto a los alumnos como a los profesores. El producto final constará de una aplicación cliente/administrador, un proxy y un servidor de datos. La aplicación cliente será la que usarán la mayoría de personas, para subir y descargar información. La aplicación administradora, será usada por los responsables de valorar las noticias que son subidas por los usuarios, y de censurarlas si es oportuno. El servidor, únicamente es utilizado como repositorio de datos.

Para el desarrollo de las aplicaciones móviles hemos utilizado el lenguaje J2ME y para la aplicación que almacenará el servidor J2SE. Con este lenguaje conseguiremos que la aplicación sea interpretada por la mayoría de dispositivos móviles.

Las comunicaciones entre el servidor y la aplicación van a utilizar conexiones TCP/IP, bien por la tecnología Wireless o bien por paquetes de datos de la compañía telefónica de cada móvil.

El proyecto está realizado empleando programación extrema como metodología de trabajo, por tanto, no tenemos grandes objetivos a largo plazo.
Vamos a seguir con la explicación detallada de todo el proyecto, empezaremos por el índice.

1. Portada: título, estudiante, titulación, consultor, fecha.

2. Resumen de toda la memoria.

3. Índice.

    3.1 Índice de figuras.

4. Cuerpo de la memoria.

    4.1. Introducción.

        4.1.1. Justificación del PFC y contexto en el que se desarrolla: punto de partida y aportación del PFC.

        4.1.2. Objetivos del PFC.

        4.1.3. Enfoque y método seguido.

        4.1.4. Productos obtenidos.

    4.2. Fase de estudio.

        4.2.1. Justificación de la elección del lenguaje.

        4.2.2. Java.

        4.2.3. J2ME.

        4.2.4 J2SE.

        4.2.5 RMI.

            4.2.5.1 RMI Optional Packet.

        4.2.6 JDBC.

    4.3 Fase de definición.

        4.3.1 Especificación de la aplicación.

        4.3.2 Gestión Técnica.

        4.3.2.1 Dispositivos receptores y equipo servidor.

        4.3.2.2 Lenguaje y comunicación.

        4.3.2.3 Herramientas de desarrollo.

3.1 Índice de figuras.

4.1 Introducción.

4.1.1. Justificación del PFC y contexto en el que se desarrolla: punto de partida y aportación del PFC.

En la asignatura Guión Multimedia, impartida por nuestro director de proyecto, Diego Álvarez, tratamos el tema de los terminales móviles y el diseño de páginas Web y aplicaciones para éstos. Fue un tema que nos gustó, así que le propusimos a nuestro director hacer un proyecto relacionado con esto.

La propuesta realizada por Diego fue la siguiente: **Desarrollo de un sistema integral de gestión de noticias exclusivo para teléfono móvil.**

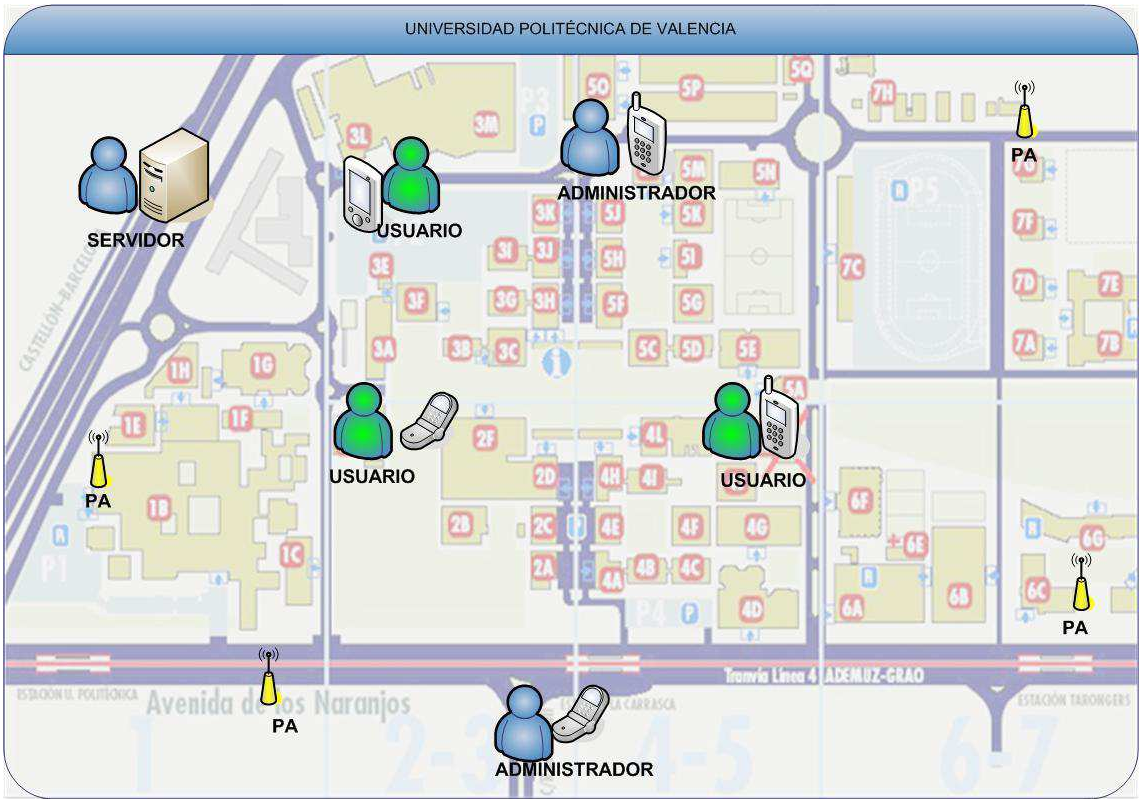
El proyecto desarrollado es una aplicación informática integrada específica para dispositivos móviles. En concreto, una aplicación para dispositivos móviles dotados con la tecnología de red Wi-fi, encargada de ofrecer comunicación con un servidor, el cual será utilizado sólo como repositorio de datos. La aplicación será la encargada de la gestión de los mismos, así como de la clasificación de los datos según los criterios establecidos.

La aplicación permite tanto subir como descargar archivos de texto. Además dispone de un visor, parecido a un lector de noticias RSS, que nos informará de los últimos archivos subidos que sean de nuestro interés, para poder visualizarlos desde nuestro terminal. Además, varios administradores se encargarán de filtrar/censurar los contenidos no aptos o inmorales. Esto implica que siempre tiene que haber varios administradores online y que cada vez que se suba algún fichero éstos reciban una notificación.

El desarrollo ha sido llevado a cabo en J2ME (más adelante detallaremos el porqué ha sido elegido este lenguaje). Previamente, teníamos algo de experiencia en programación en Java (gracias a las asignaturas de Programación y Estructuras de Datos y Algoritmos) y en J2ME (gracias a la asignatura Integración de Medios Digitales, dónde realizamos un proyecto en éste lenguaje).

Gracias al PFC, hemos adquirido mayor experiencia programando en este lenguaje, además de aprender sobre las comunicaciones Wi-fi y sobre MySql, sobre lo que previamente no teníamos ninguna experiencia. Además, también ha servido para aprender a controlar a la perfección los sockets java y la comunicación TCP entre aplicaciones. Por último, también hemos adquirido más experiencia en cuanto a la preparación, planificación y ejecución de proyectos se refiere.

Queremos destacar que las noticias que esperemos que se publiquen en nuestra aplicación, estén relacionadas con la Universidad Politécnica de Valencia, y sirva para denunciar malas conductas, defectos en las instalaciones o, por el contrario, felicitar y criticar positivamente al centro por las cosas bien hechas.



4.1- Distribución de clientes, administradores y del servidor en la Universidad.

4.1.2. Objetivos del PFC.

Los objetivos marcados para el PFC son los siguientes:

- Desarrollo de la aplicación para los terminales móviles (Cliente y Administrador.).

- Desarrollo de la aplicación servidora.

- Diseño de la base de datos.

- Diseño de un sistema para comunicar la aplicación móvil con la aplicación servidora.

- Implantación de la aplicación completa (cliente+proxy+servidor).

Estos objetivos globales que teníamos al inicio del proyecto, se han ido desglosando en pequeños objetivos que nos hemos ido marcando paso a paso.

4.1.3. Enfoque y método seguido.

La metodología de trabajo seguida para la realización del proyecto final de carrera se basa en la metodología de trabajo propuesta en la asignatura Producción Multimedia. El desarrollo del proyecto sigue una estructura de trabajo dividida en fases. A continuación se detalla el trabajo realizado en cada una de ellas, y se incluye información básica de partida.

i. Fase de iniciación.

El objetivo de esta fase es recabar y analizar información necesaria para poder realizar correctamente, y partiendo de bases sólidas, la definición del proyecto.

ii. Fase de planificación.

El objetivo de esta fase es definir completamente el proyecto y el producto. Esta fase comprende la gestión del alcance y la gestión técnica.

1. Gestión de Alcance.

En este punto se realizan las tareas necesarias para definir la aplicación a realizar y concretar aspectos relativos al proyecto.

2. Gestión Técnica.

Define y concreta las herramientas hardware y software necesarias para desarrollar y usar la aplicación.

3. Gestión de la planificación temporal.

i. Descomposición en tareas:
ii. Estimación de duraciones:
iii. Orden de ejecución:
iv. Asignación de recursos:

iii. Fase de ejecución.

Comprende una memoria explicativa de todos los pasos que se han seguido para alcanzar los objetivos propuestos.

iiii. Fase de cierre.

Análisis de los resultados obtenidos. Revisión de la documentación generada y elaboración de manuales.

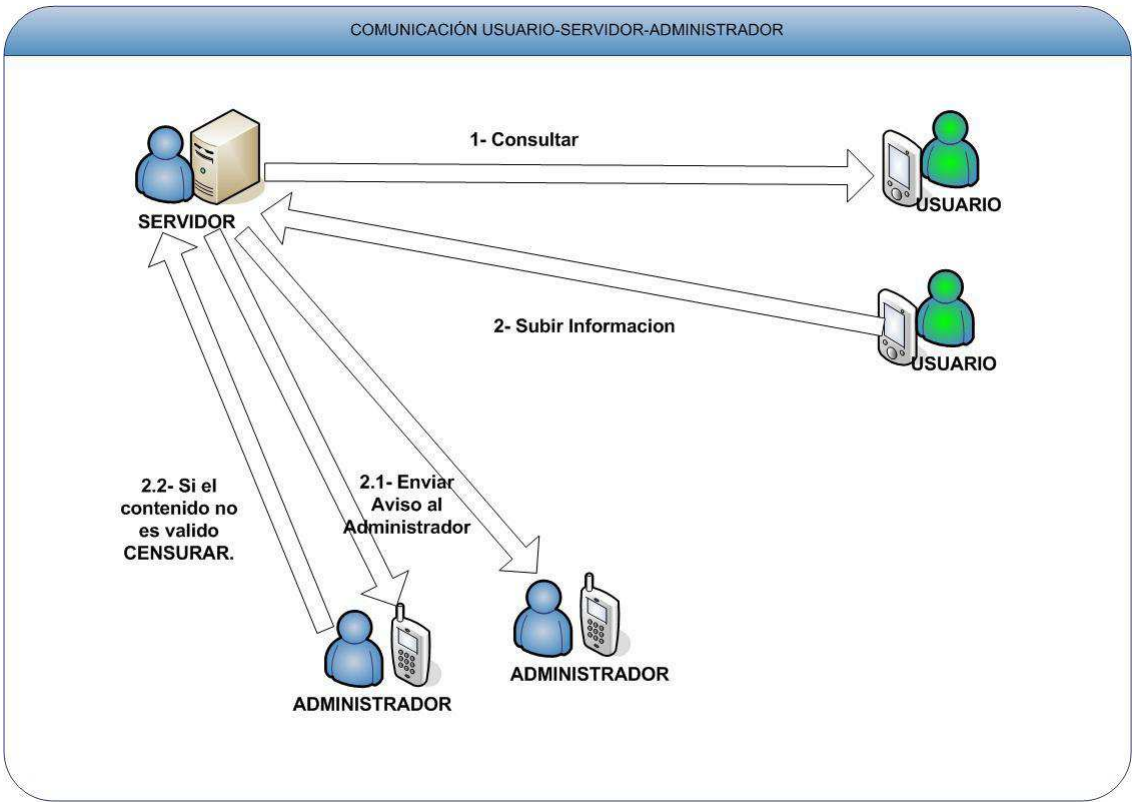
5.1.4. Productos obtenidos.

Podríamos considerar que de nuestro proyecto se obtienen tres productos distintos, aunque cada uno necesita los otros dos para funcionar correctamente. Los vamos a describir a continuación:

- Aplicación cliente: Esta será la aplicación que llevará cada cliente en su terminal móvil. Desde ella podrá registrarse, permitirá descargar las últimas noticias que han sido subidas por los distintos usuarios y que aún están pendientes de valoración. Desde aquí se permitirá aceptarlas o rechazarlas (por no mostrar contenidos dañinos o inmorales).

- Servidor: El servidor puede ser considerado también como un producto final del proyecto. Tanto la aplicación cliente como la de administrador se tienen que conectar a éste para poder obtener los datos. En dicho servidor, estará siempre en funcionamiento un programa para gestionar las conexiones a éste.

-Proxy: Actua como intermediario de los dos anteriores.



4.2- Proceso para subir/descargar información.

4.2. Fase de estudio.

4.2.1. Justificación de la elección del lenguaje.

Uno de los aspectos más importantes que había que abordar en el proyecto era la elección del lenguaje. Hoy en día, el sistema operativo más utilizado en los teléfonos móviles es el Symbian. Además, también está ampliamente extendido el Windows Mobile, con lo que podíamos decantarnos en programar nuestra aplicación bien para un sistema operativo, bien para otro, ya que existe muchísima documentación y ejemplos para estos sistemas operativos. Sin embargo, de esta forma, no sería compatible con muchísimos teléfonos móviles, con lo que nuestra aplicación no podría ser distribuida tan ampliamente como nos gustaría. Lo que se quería era desarrollar un programa pudiera utilizarse desde cualquier teléfono móvil. Es aquí donde se contempla la posibilidad de hacerlo en Java, en concreto en J2ME (Java 2 Micro Edition). ¿Y por qué Java? Java fue concebido en su origen para dar la posibilidad a cualquier aparato electrónico de consumo de tener software y poder mejorar así el rendimiento. Es posible incluir Java, por medio de su máquina virtual, en casi cualquier tipo de CPU existente, por pequeña que sea ésta. Y los teléfonos móviles cuentan con CPU, aunque de pequeña potencia computacional. Realmente es posible programar para un móvil gracias a que prácticamente, desde hace un tiempo y hasta la fecha, todos los teléfonos móviles del mercado cuentan con la máquina virtual de Java y Sun Microsystems ha puesto los medios adecuados para poder programar para ellos. Un último apunte: hoy por hoy no existe ningún otro lenguaje de programación capaz de crear aplicaciones funcionales en móviles, fuera del sistema operativo de éstos.

4.2.2. Java.

Java tiene su origen en 1991, cuando un grupo de investigadores de Sun Microsystems, liderado por James Gosling, trabajaba en técnicas para producir software capaz de ser incluido dentro de cualquier aparato electrónico (teléfonos, faxes, vídeos y electrodomésticos en general) para proveerlos de cierta “inteligencia”. La reducida potencia de cálculo y memoria de estos aparatos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido. En principio, consideraron la posibilidad de utilizar lenguajes ya existentes como C++ o Smalltalk, pero pronto se hizo patente la necesidad de definir una máquina hipotética o virtual capaz de garantizar la portabilidad de las aplicaciones debido a la existencia de distintos tipos de CPU y a los distintos cambios, a la vez que cumpliera los requisitos de seguridad derivados de su uso en dispositivos de uso común. Por tanto, Java es el lenguaje de programación y, además, la definición de la máquina virtual (*Java Virtual Machine*) encargada de ejecutar las aplicaciones.

A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje, por lo que iba a caer en desuso dado que los directivos de Sun no veían un mercado potencial y el grupo fue disuelto. Sin embargo, Gosling intuyó las posibilidades del lenguaje para proporcionar contenidos activos en las páginas Web y se embarcó en la construcción de un *browser* (navegador) basado en Java, HotJava, que

constituyó la demostración general de las capacidades del lenguaje, comenzando la historia de Java como lenguaje íntimamente ligado a Internet.
Como lenguaje de programación se presentó a finales de 1995 y desde entonces ha sido uno de los temas que más interés ha despertado en el mundo de la informática, mereciendo incluso, la atención de publicaciones no especializadas. La clave fue la incorporación de un intérprete Java en el programa Netscape Navigator 2.0 produciendo una verdadera revolución en Internet. La mayor parte de las aplicaciones Java se utilizaron para dotar de contenido dinámico e interactivo a las páginas del World Wide Web, mediante los llamados *applets*. Posteriormente el uso de Java se fue extendiendo a un gran número de sistemas y aplicaciones, en las que el modelo ofrecido por Java de un entorno distribuido y completamente transportable entre plataformas es enormemente atractivo.

Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje, encontrándose en el momento de escribir este documento por la versión 6.0 Update 16.

4.2.3. J2ME

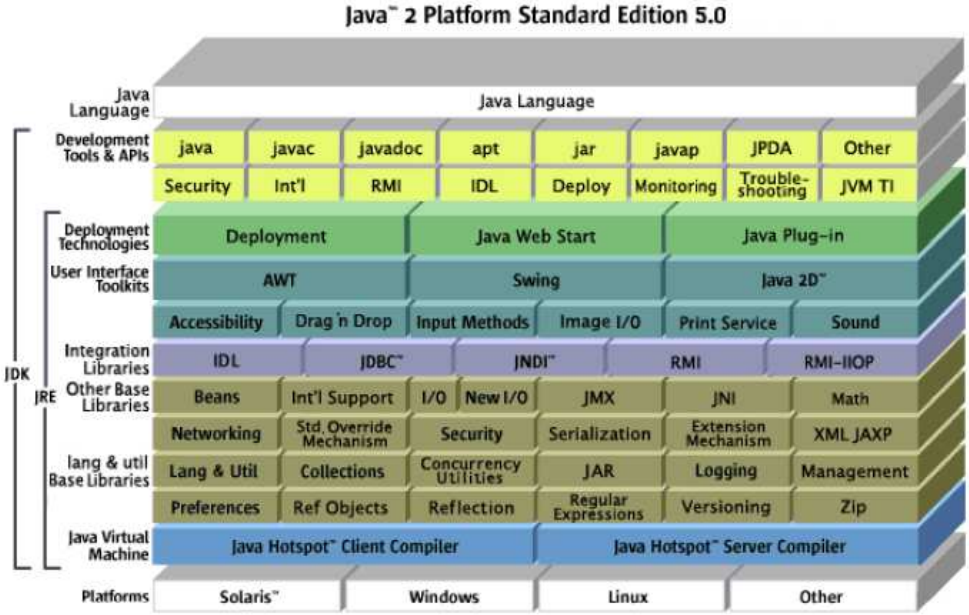
"*J2ME es un subconjunto de J2SE orientado al desarrollo de aplicaciones Java destinadas a dispositivos con pocos recursos, con capacidades restringidas, tanto con respecto a la capacidad de memoria disponible, limitaciones de la pantalla gráfica como con respecto a la capacidad de procesamiento; características típicas de cualquier equipo de electrónica de consumo; por ejemplo, teléfonos celulares, PDAs, buscapersonas, mensáfonos, dispositivos de navegación de coches, etc*."

J2ME fue presentada en 1999 por Sun Microsystems con el propósito de habilitar aplicaciones Java para pequeños dispositivos. Esta primera versión sólo nos mostraba una nueva *Java Virtual Machine* (JVM) para dispositivos Palm. La aparición de esta tecnología está íntimamente ligada a las necesidades de los usuarios de telefonía móvil que cada vez demandan más servicios y prestaciones por parte tanto de terminales como de sus propias compañías. J2ME es la tecnología del futuro para la industria de los dispositivos móviles y, actualmente, la práctica totalidad de las compañías telefónicas están incluyendo los protocolos y los medios necesarios para implantarla en sus productos.

4.2.4 J2SE

Java Platform, Standard Edition o Java SE (conocido anteriormente hasta la versión 5.0 como Plataforma Java 2, Standard Edition o J2SE), es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java. La Plataforma Java 2, Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre workstations.
Comenzando con la versión J2SE 1.4 (Merlin), la plataforma Java SE ha sido desarrollada bajo la supervisión del Java Community Process. JSR 59 la

especificación para J2SE 1.4 y JSR 176 especificó J2SE 5.0 (Tiger). En 2006, Java SE 6 (Mustang) está siendo desarrollada bajo el JSR 270.
La siguiente figura, es una descripción de lo que puede abarcar el estándar J2SE.



4.3 Plataforma J2SE.

4.2.5 RMI

El sistema de Invocación Remota de Métodos (RMI) de Java permite a un objeto que se está ejecutando en una Máquina Virtual Java (VM) llamar a métodos de otro objeto que está en otra VM diferente. Forma parte del entorno estándar de ejecución de Java y provee de un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.
Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente. Una aplicación servidor típica crea una gran cantidad de objetos remotos, hace accesibles unas referencias a dichos objetos remotos, y espera a que los clientes llamen a estos métodos u objetos remotos. Una aplicación cliente típica obtiene una referencia remota de uno o más objetos remotos en el servidor y llama a sus métodos. En el caso de nuestro proyecto, el terminal llamaría a unos métodos que están ejecutándose en el servidor, siendo estos métodos los que accederían a la base de datos. RMI proporciona el mecanismo por el que se comunican y se pasan información del cliente al servidor y viceversa. Las distintas máquinas donde se encuentran el cliente y el servidor no tienen por qué tener ni la misma arquitectura ni el mismo sistema operativo, sólo requieren tener su *JVM* (*Java Virtual Machine*) correspondiente.
Por medio de RMI, un programa Java puede exportar un objeto, lo que significa que éste queda accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de este momento, un cliente

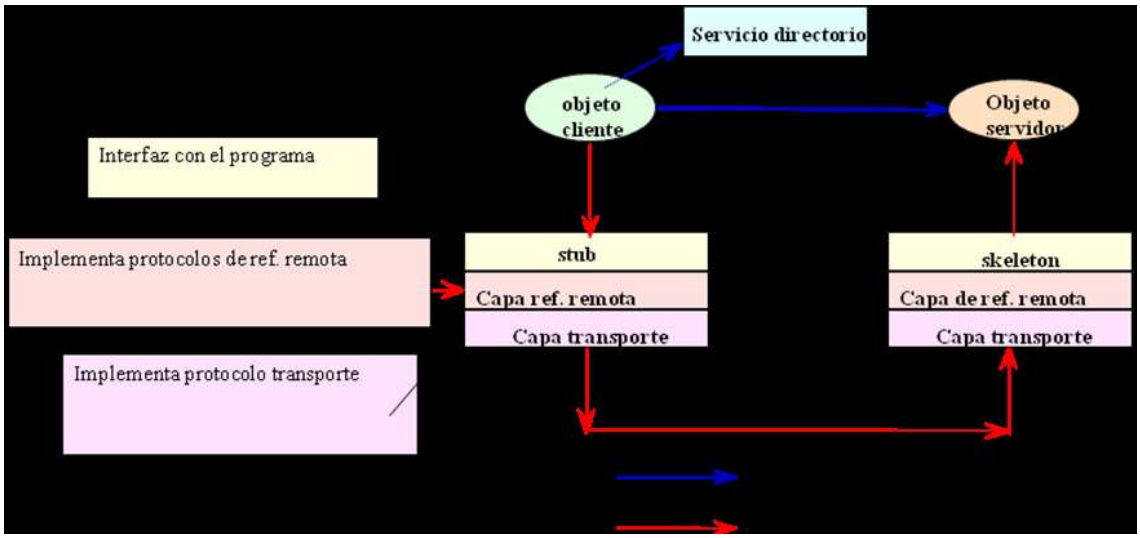
puede conectarse e invocar los métodos proporcionados por el objeto. El uso de *RMI*, entre otras cosas, aporta:

– Un recolector de basura distribuido,

– mantiene los sistemas de seguridad de Java,

– usa la posibilidad de crear hilos de ejecución (*threads*) concurrentes potenciando la idea concurrente en las aplicaciones distribuidas y

– hace un uso inteligente de la *Serialization* de los objetos *Java* con la posibilidad de transportar objetos a través de la red sin necesidad de ningún esquema extra.



4.4 Funcionamiento RMI.

4.2.5.1 RMI Optional Packet

Es un paquete que se puede utilizar sobre la configuración CDC. EL paquete RMI Optional (RMI OP) permite a dispositivos de consumo y aplicaciones embebidas interactuar cómo y con aplicaciones distribuidas. Se amplían las características de RMI existentes para que pequeños dispositivos interactúen entre sí. RMI OP es un subconjunto del API RMI de J2SE.
Desarrollado a través del Java Community Process (JCP), el paquete opcional de RMI (RMI OP) trae el mundo de las aplicaciones distribuidas de consumo y dispositivos integrados.

Durante mucho tiempo una característica de la plataforma Java 2 Standard Edition (J2SE), ha sido la invocación de métodos remotos que permite que las aplicaciones escritas en el lenguaje de programación Java interactúen de manera transparente, flexible, incluso cuando se desplieguen en sistemas muy diferentes. Con la implantación en los sistemas de RMI OP,

“dispositivos muy diferentes” ahora incluyen, no sólo dispositivos de escritorio y servidores, sino además pero pequeños aparatos portátiles y embebidos como comunicadores inteligentes y asistentes digitales personales. RMI OP es un subconjunto de J2SE RMI que se pueden utilizar en dispositivos que soportan la Connected Device Configuration (CDC) de la plataforma Java 2, Mobile Edition (J2ME).

RMI OP soporta las características fundamentales del paquete de RMI J2SE:

- La semántica completa llamada RMI.
- Soporte de objetos creados remotamente.
- Protocolo de conexión RMI.
- Exportación de objetos remotos.
- Tanto del lado del cliente como del servidor distribuido de recolección de basura.
- Interfaz de Activador y el protocolo cliente-lado de activación. Interfaces de Registro y la exportación de un objeto de registro remoto.

Algunas de las características de J2SE RMI no son compatibles con RMI OP:

- RMI a través de cortafuegos a través de proxies HTTP.
- Protocolo de multiplexación de RMI Modelo de ejecución de un objeto remoto activable.
- Métodos obsoletos, clases e interfaces.
- Stub y el compilador de esqueleto Apoyo a la JDK 1.1 talón / Protocolo de esqueleto.

La implementación de referencia OP RMI se basa enteramente en la tecnología Java y no requiere ningún esfuerzo de portabilidad. Las notas de publicación para la aplicación de referencia incluyen las instrucciones para la elaboración de las aplicaciones que utilizan RMI OP.

5.2.6 JDBC.

Java Database Connectivity, más conocida por sus siglas JDBC, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice, pudiendo ser de diferentes proveedores (Microsoft SqlServer, Oracle, MySQL, Interbase, Microsoft Access, PostgresSQL, etc…).

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos (diferentes drivers de conexión), y accede a ella estableciendo una conexión, para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos a las que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Clases y métodos de JBDC

JBDC utiliza los mismos métodos y clases independientemente del driver usado para conectar al proveedor de bases de datos, lo que cambia es el nombre del driver por lo que es bastante sencillo modificar aplicaciones al cambiar de proveedor. El código genérico de conexión a una base de datos es:

- Registrar el driver JDBC.

  Class.forname(“nombre_del_driver”);

- Conectar a la BD usando la interfaz Connection que abre una sesión o conexión con la BD especificada y, mediante el método DriverManager.getConnection, intenta seleccionar el driver apropiado entre los que JBDC tenga registrados en el sistema.

- Connection con=DriverManager.getConection(“BD_url”,”usuario”,”passw”);

- Ejecutar sentencias SQL; la interfaz Statement permite ejecutar las instrucciones SQL y devolver el resultado generado.

Statement select= con.createStatement();

- La interfaz ResultSet representa un conjunto de datos resultado de una consulta SQL, para acceder a los registros se emplea un cursor que inicialmente apunta antes del primer registro y para avanzar por los registros se emplea el método ResultSet.next(). ResultSet es solo de lectura.

ResultSet nombres=select.executeQuery(“Sentencia select”)

JBDC puede usar varios drivers, entre ellos los más importantes son:

- Microsoft SQLServer.

- JBDC-ODBC Bridge (puente con OBDC, sirve para conectar con bases de datos de Microsoft Access).

- SQL Server 2005.

En nuestro proyecto, la base de datos la hemos diseñado con My SQL . Por tanto, vamos a explicar cómo se realizaría la conexión con este tipo de driver.

En los métodos comentados anteriormente, deberemos emplear los siguientes nombres, para que el sistema entienda que estamos utilizando driver MySQL.

- Class.ForName: com.microsoft.jbdc.sqlserver.SQLServerDriver

- Connection:

- jdbc:Microsoft.sqlserver://<HOST>:<Puerto>;DatabaseName=[nombre_BD];User=[usuario];Password= [password]

4.3 Fase de definición.

4.3.1 Especificación de la aplicación.

En esta sección describiremos detalladamente todos aquellos aspectos que describen la situación de partida del proyecto. Entre otras cosas, describiremos el público objetivo al que se dirigía el proyecto, contenidos y servicios de este, el marco legal que debe tener en cuenta, así como otros aspectos relevantes como la estructura general o su navegación principal.

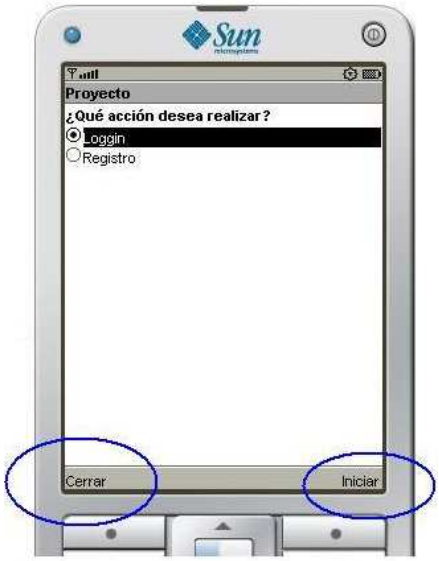
El proyecto fue diseñado pensando que el usuario principal iba a ser el propio alumno de la universidad. También hemos pensado que podría ser interesante que los profesores también puedan tener acceso a la aplicación y participar también en el contenido de noticias. Por tanto, diremos que el público objetivo al que se dirige nuestra aplicación va a ser toda la comunidad universitaria, tanto los alumnos como los profesores, que dispongan de terminales móviles con conexión internet. Pensamos que los temas que deben tratar las noticias de la aplicación deben estar relacionados con la Universidad Politécnica de Valencia. Por ello, no hemos considerado como público objetivo a personas ajenas a la comunidad universitaria.

En cuanto al marco legal de la aplicación, hay que tener en cuenta la ley de protección de datos. Debemos estudiarla nuestra aplicación y desarrollarla para que cuando el usuario se registre en nuestra ella, asuma toda la responsabilidad de los archivos que publica.

Pasamos a describir la navegabilidad de la aplicación. Como sabemos, esta consta de dos versiones, una para los usuarios normales y otra para los administradores del sistema. Pasamos a describirlas.

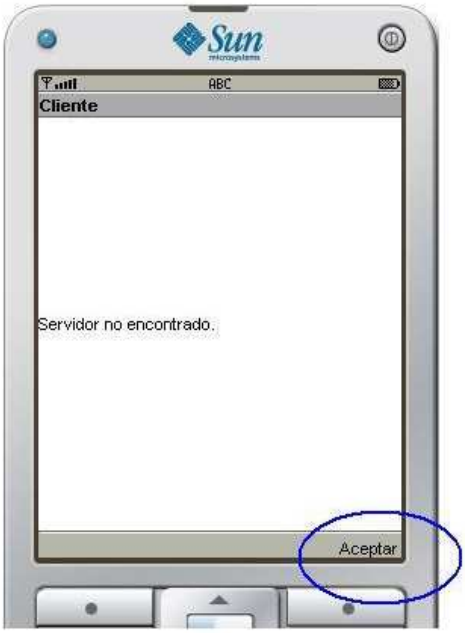
Parte común.

Al arrancar la aplicación aparecerá una pantalla como la que se muestra a continuación. Dicha pantalla tendrá un botón para saltar la presentación si se desea.



La siguiente pantalla nos dará la opción de loguearnos o registrarnos en la aplicación. Tendrá dos botones, uno para aceptar y otro para cerrar el programa.

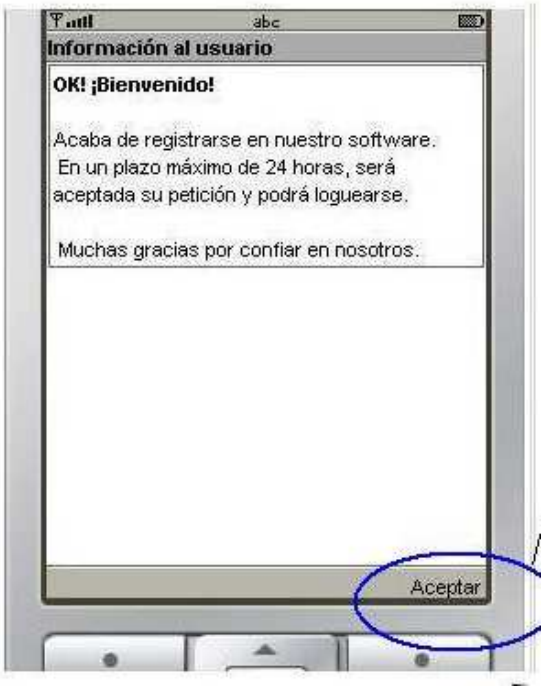
En caso de no encontrar el servidor se mostrará una pantalla informando del error como se puede ver a la izquierda y en caso de cerrar la aplicación un mensaje informativo como se puede observar a la derecha.

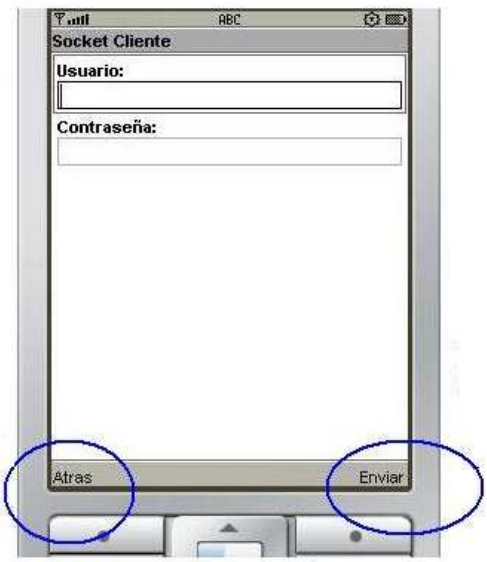


Si decidimos registrarnos

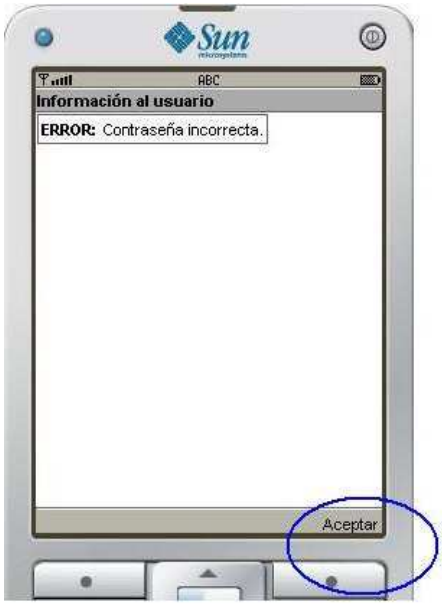
Deberemos rellenar un formulario como el que se muestra a continuación, que nos da nuevamente la posibilidad de enviar los datos al servidor o cancelar.

En caso de decidir registrarnos, se nos informará si todo ha ido correctamente mediante la pantalla que podemos observar a la derecha, aún así no podremos hacer login hasta que un administrador no decida activar nuestra cuenta. Los administradores recibirán un email al registrarse alguien y además los que estén online les vibrara el móvil.



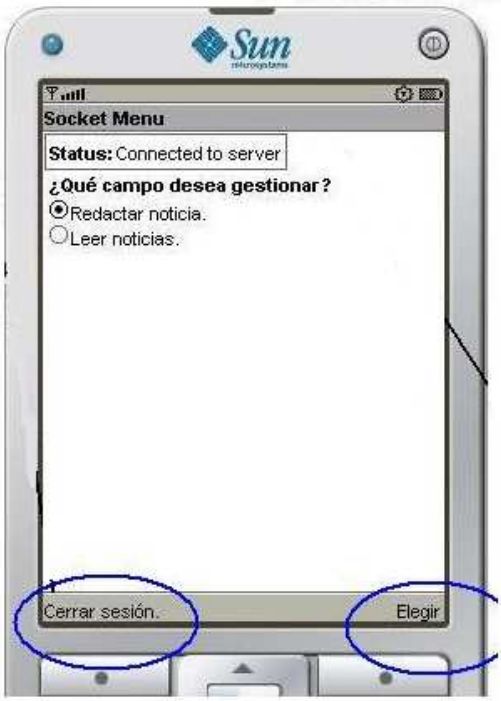


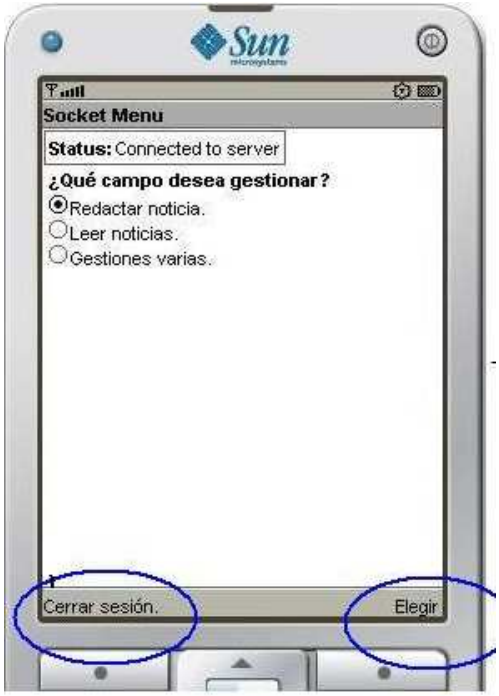
Si ya somos usuarios registrados tendremos que validarnos frente al servidor My SQL mediante un nombre de usuario y contraseña. Nuevamente tendremos la opción de cancelar o enviar los datos para validarnos.



En caso de equivocarnos en la contraseña se nos informará de ello y volveremos a la pantalla de registro o login.

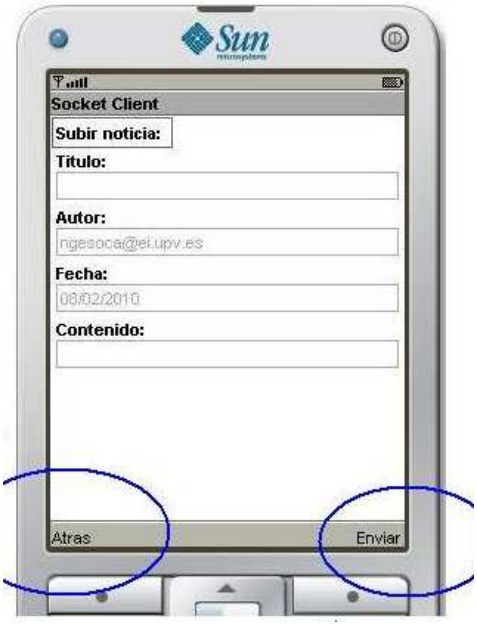
Hasta aquí la aplicación se comporta de un mismo modo tanto si tu usuario tiene privilegios de administrador como si no los tiene. Pasaremos a explicar en primer lugar La parte en común que hay entre ambas.



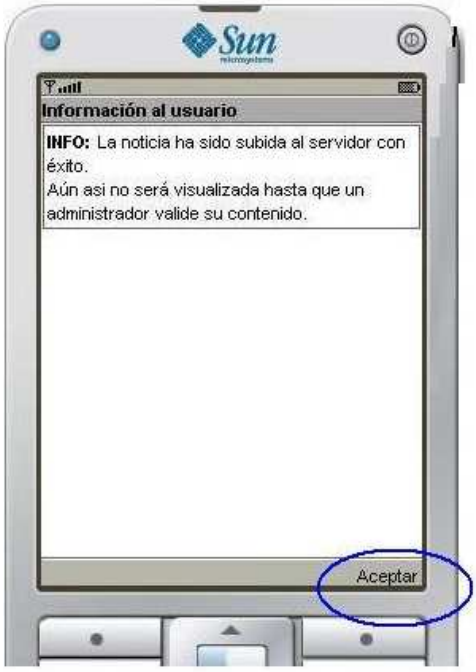


Como podemos observar la única diferencia entre ambas es que la imagen de la derecha corresponde a un usuario administrador, y tiene una opción más titulada “Gestiones varias” que procederé a explicar más adelante.

Vamos a explicar ahora la parte común entre los dos usuarios. Empecemos por redactar noticia.

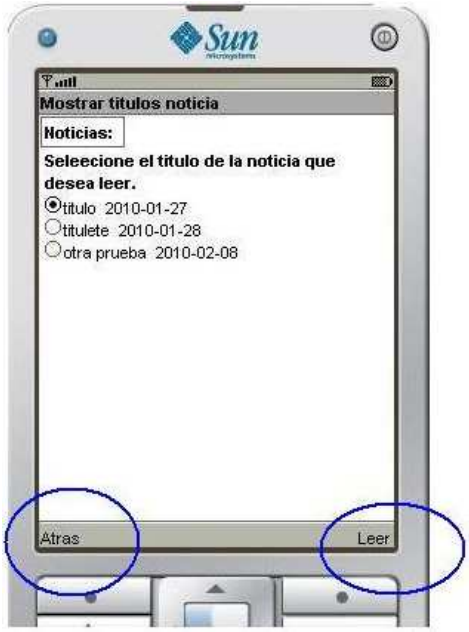


Si decidimos redactar una noticia aparecerá un formulario como el de la imagen, en el cual deberemos rellenar los datos del titulo y el contenido pues el resto de datos los coge todos del sistema. Tendremos la opción de validar nuestra noticia y enviarla por tanto al servidor o de cancelar.



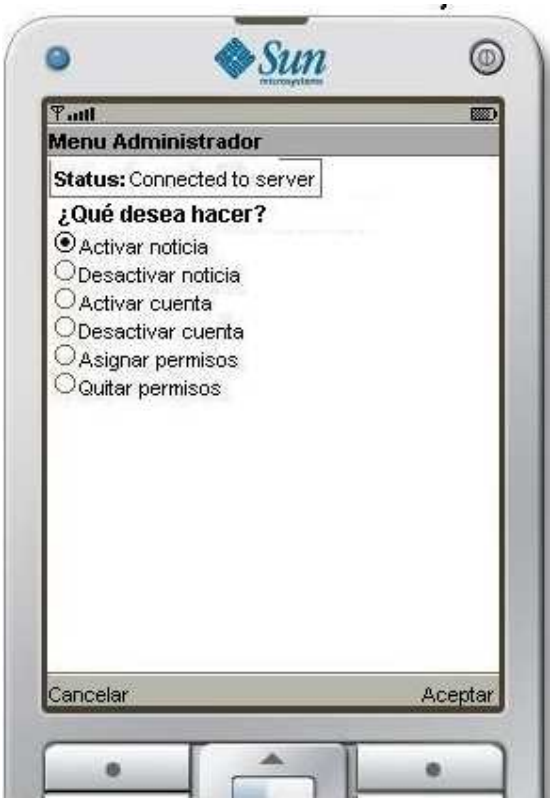
Si decidimos enviarla al servidor, ésta se almacenará, pero no podrá ser visualizada por el resto de usuarios hasta que un administrador la valide igual que ocurría cuando se registraban nuevos usuarios.

Por otra parte la otra opción en común es la lectura de noticias. Si seleccionamos esta opción, nos aparecerá en pantalla una imagen similar a la de la derecha. En la cual se nos mostrará el titulo de la noticia y la fecha de la misma. Las noticias se muestran cronológicamente en orden descendente, dejando la primera la mas vieja.



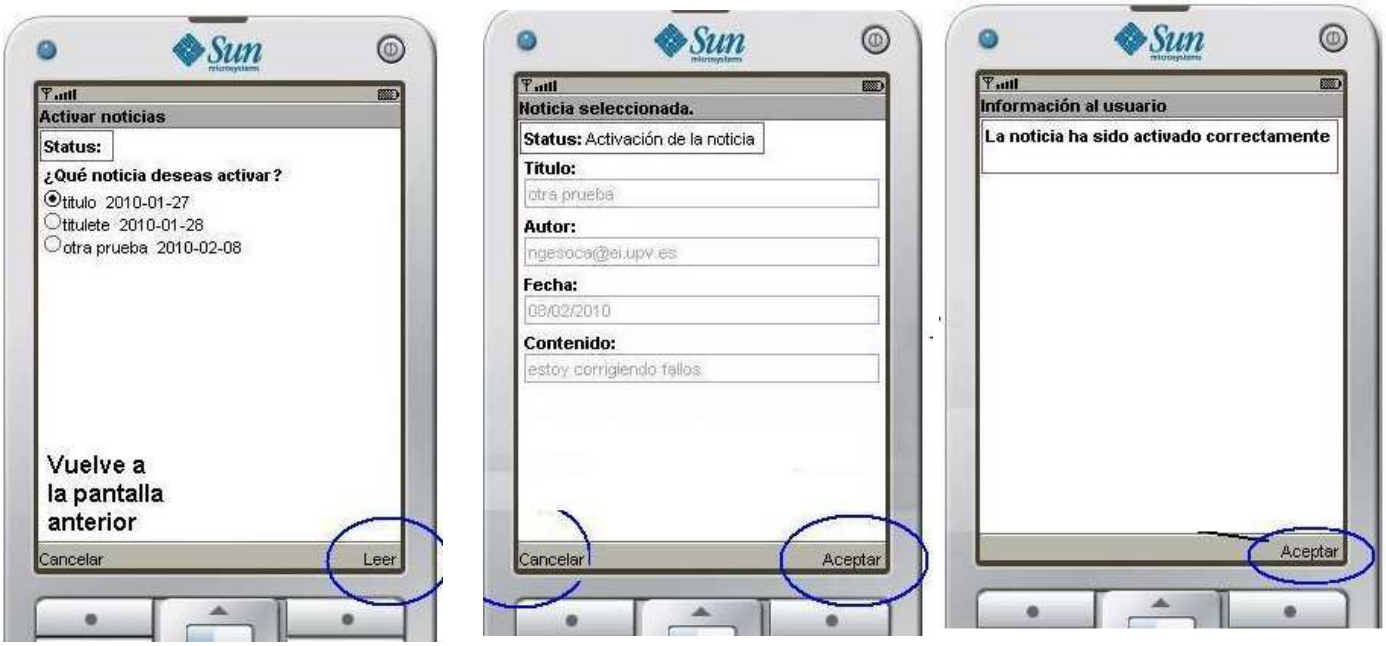
Si seleccionamos una noticia para leer en el listado anterior la podremos visualizar como se muestra en la imagen de la izquierda. Al terminar de leerla podremos volver al listado para continuar leyendo noticias pulsando el botón atrás.

Hasta aquí la parte común entre los dos tipos de usuarios, continuamos ahora explicando las opciones que tiene extras el administrador. Como podemos observar en el menú del administrador existe la opción “Gestiones varias”. Si seleccionamos esa opción, se nos presentará en pantalla un menú como el siguiente:



En dicho menú podremos seleccionar una serie de acciones:

Activar noticia:

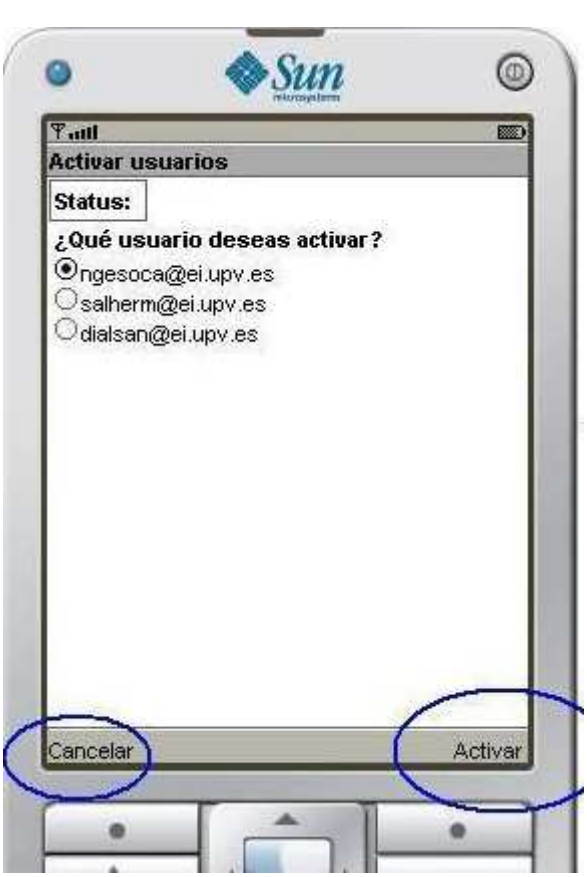


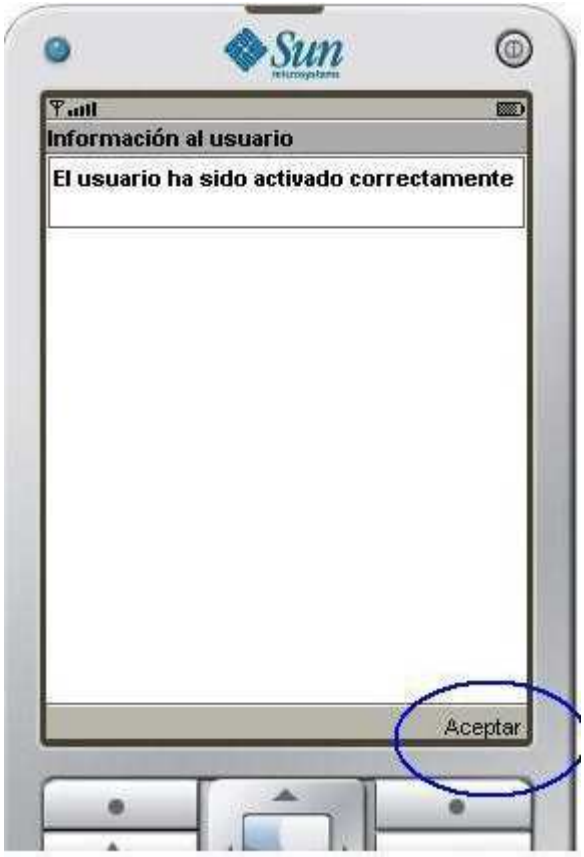
Si seleccionamos esta opción, aparecerá un listado de las noticias aún no activadas. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para activar la noticia pulsaremos al botón leer, esta nos la presentara en otro formulario, igual que el formulario central que si cancelamos volveremos al listado de noticias aún no activadas y si aceptamos activaremos la noticia para que el resto de usuarios puedan visualizarla y se nos informará de ello mediante el tercer formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

Desactivar noticia:

Si seleccionamos esta opción, aparecerá un listado de las noticias activadas. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para desactivar la noticia pulsaremos al botón leer, esta nos la presentara en otro formulario, igual que el formulario central que si cancelamos volveremos al listado de noticias aún activas y si aceptamos desactivaremos la noticia para que el resto de usuarios ya no puedan visualizarla y se nos informará de ello mediante el tercer formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

Activar cuenta:



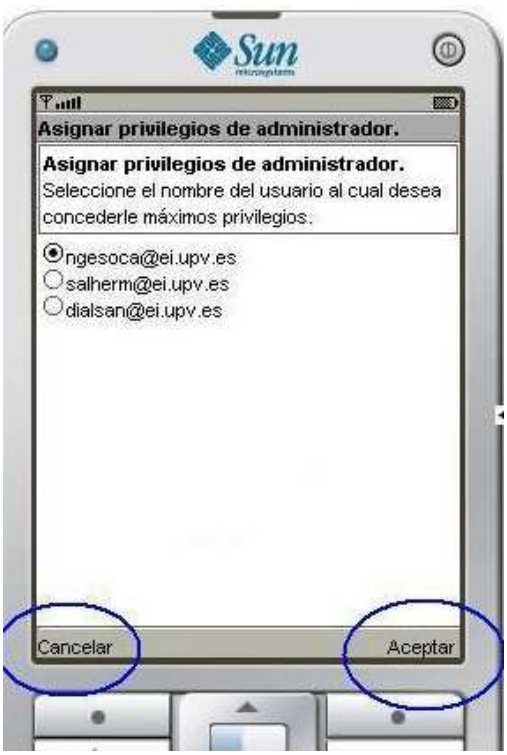


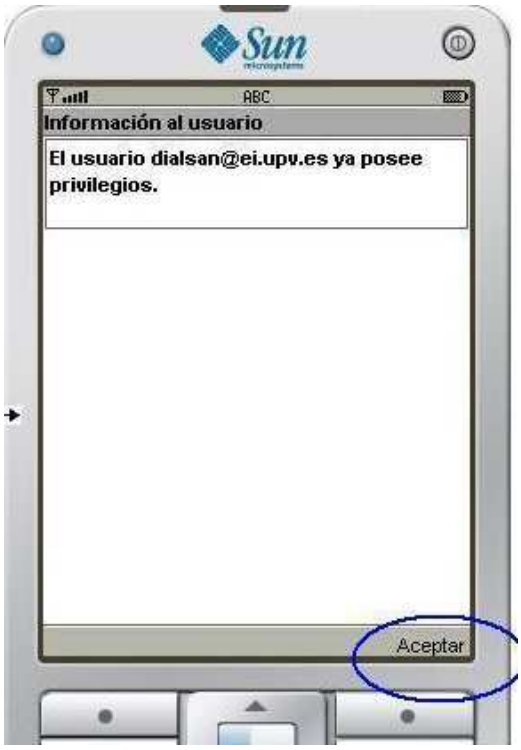
Si seleccionamos esta opción, aparecerá un listado de los usuarios no activos. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para activar un usuario pulsaremos al botón activar y el usuario se activará pudiendo desde ese momento loguearse en el sistema, además se nos informará de ello mediante un formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

Desactivar cuenta:

Si seleccionamos esta opción, aparecerá un listado de los usuarios activos. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para desactivar un usuario pulsaremos al botón desactivar y el usuario se desactivara y desde ese momento no podría loguearse en el sistema, además se nos informará de ello mediante un formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

Asignar permisos:





Si seleccionamos esta opción, aparecerá un listado de los usuarios activados que no son administradores. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para asignar permisos a un usuario pulsaremos al botón aceptar y al usuario se le asignaran privilegios de administrador, pudiendo desde ese momento loguearse en el sistema como tal. Además se nos informará de ello mediante un formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

Quitar permisos:

Si seleccionamos esta opción, aparecerá un listado de los usuarios activados que son administradores. En esa pantalla si pulsamos al botón cancelar volveremos al menú anterior. Para quitar permisos a un usuario pulsaremos al botón aceptar y al usuario se le quitaran sus privilegios de administrador. Además se nos informará de ello mediante un formulario, el de la derecha y que tras aceptar volveremos al listado otra vez.

4.3.2 Gestión Técnica.

4.3.2.1 Dispositivos receptores y equipo servidor.

Llamaremos receptores a los dispositivos que dispondrán tanto los clientes como los administradores. Generalmente se tratará de teléfonos móviles, aunque también podríamos estar hablando de PDAs u otros dispositivos. El servidor estará implementado sobre un PC con sistema operativo Windows 2000. Perfectamente podría estar implementado sobre otro sistema operativo, ya que Java se adaptaría, pero decidimos hacerlo en Windows para poder utilizar SQLServer para gestionar las bases de datos. Este programa es de Microsoft y, por tanto, deberemos usarlo sobre la plataforma Windows.

Se pueden hacer pruebas y mejoras en emuladores que hacen la función de un terminal móvil. Estos emuladores estarían en un PC de trabajo pero para comprobar los resultados finales se debería de utilizar dispositivos reales.

4.3.2.2 Lenguaje y comunicación.

Como hemos dicho en apartados anteriores, el lenguaje de programación utilizado ha sido J2ME. Este lenguaje permite cumplir con los objetivos marcados, además de poder ser interpretado por gran numero de dispositivos. La mayoría de dispositivos móviles soportan java, y poco a poco disponen de más capacidad para añadir más funcionalidades.

Para la comunicación entre los dispositivos receptores y el servidor hemos utilizado el protocolo TCP/IP, es decir, mediante internet. Esto nos permite utilizar la aplicación en cualquier sitio donde tengamos acceso a la red. Cada vez son más los sitios que disponen de acceso a internet, siendo ésta una forma de comunicación que está en auge. Esto es debido a la gran aceleración que ha tenido la tecnología de comunicación inalámbrica (Wi-fi o Wireless) que nos permite conectarnos a la red casi desde cualquier lugar.

Además, las compañías telefónicas, también ponen a nuestra disposición conexiones a internet mediante su línea, con lo que podemos conectarnos a la red desde cualquier lugar del mundo (eso sí, tenemos que estar dispuestos a pagar sus tarifas).

4.3.2.3 Herramientas de desarrollo.

Para poder desarrollar el producto, hemos utilizado distintas herramientas, bien para simplificar el trabajo, bien porque eran necesarias. Para la implementación en java, tanto de la aplicación del servidor como las dos para terminales móviles, hemos utilizado NetBeans IDE 6.5. Esta herramienta, además, incluye un emulador de teléfono móvil para poder probar las aplicaciones.

El desarrollo de la base de datos lo hemos realizado con SQLServer 2005. Para gestionar éstas bases de datos, hemos utilizado SQL Mannagement Studio.

Además, todo lo anterior no podría funcionar sin el JDK de Sun instalado en nuestras maquinas. Por tanto, también lo hemos descargado de la pagina de Sun y lo hemos instalado en nuestros PCs para poder desarrollar las aplicaciones.

Estas herramientas están descritas más ampliamente en los anexos del final de la memoria.

4.3.2.4 Requisitos mínimos.

Para poder llevar a buen puerto el correcto funcionamiento de estas aplicaciones, debemos de tener una serie de requisitos mínimos de software y hardware en nuestros dispositivos móviles.

En primer lugar es imprescindible que los dispositivos móviles tanto de los clientes como de los administradores puedan trabajar con programas elaborados en lenguaje Java. Es decir, deben incorporar la maquina virtual de java. Si lo móviles no disponen de esta característica, es imposible que el programa funcione correctamente. La gran mayoría de móviles de hoy en día, cumplen con este requisito.

En segundo lugar, los dispositivos móviles deben tener acceso a la red, bien por tecnología Wireless o bien por la línea de datos contratada por su compañía de telefonía móvil. Sin este acceso a la red, no se puede tener acceso al servidor de datos, por lo que tanto la aplicación cliente como la administradora no tendrían ninguna utilidad.

Para finalizar, el terminal debe de tener suficiente memoria libre para la instalación del programa. Hoy en día los dispositivos móviles disponen de mucha capacidad de memoria, no solo interna, sino también con la ayuda de tarjetas de memoria que oscilan de los 512 MB hasta unos 8 GB o más. Por lo tanto no creo que esto sea un problema.

Para las aplicaciones tanto del cliente como del administrador, deberíamos disponer en el terminal entre 50 KB y 150 KB, para el archivo de instalación del programa y para los archivos instalados en el móvil. Además, el programa podría ampliarse con nuevas funcionalidades.

Tanto en un terminal que va a utilizar la aplicación administradora, como en un terminal que va a utilizar la aplicación cliente, pensamos que debería disponer, además de la capacidad que hemos comentado, de más memoria libre para poder almacenar las noticias que vamos a ver en el visor de noticias.

Antes de ver una noticia, esta será descargada en el móvil y abierta posteriormente. Por tanto, hará falta memoria adicional para esto.

4.4 Planificación.

Para la correcta elaboración de las aplicaciones debemos de estructurar bien el proceso de producción. Por ello, antes que nada, debemos tener estudiada la planificación del proyecto y la metodología que vamos a seguir para llevarlo a cabo. Los pasos que vamos a seguir son los siguientes:

- Análisis: El análisis ya lo hemos realizado en la fase de estudio y de definición, con lo que hemos logrado tener más claras las ideas que queremos desarrollar de ahora en adelante. Aquí hemos determinado los objetivos a largo plazo del proyecto.

- Diseño: En esta sección nos basamos principalmente en el diseño del sistema de información, más que en el diseño visual que iba a tener la aplicación, ya que al ser diseñada para un teléfono móvil, el diseño no iba a ser extremadamente complicado. En esta fase decidimos como íbamos a realizar el almacenamiento de los datos (en un principio pensábamos utilizar hojas XML y posteriormente decidimos utilizar bases de datos) y también las partes que iba a tener la aplicación y como iban a ser los accesos al servidor.

- Implementación: Vamos a ir implementando paso a paso las diferentes partes de las aplicaciones. Puesto que vamos a seguir una metodología de programación extrema, iremos implementando pequeños entregables, es decir, unidades funcionales que realicen alguna función del producto final. En la parte de producción se detallará la parte de implementación.

- Pruebas: Conforme vayamos implementando partes del código, iremos haciendo pruebas y valorándolas. Según el resultado de estas pruebas, seguiremos con la siguiente funcionalidad de la aplicación o volveremos atrás para corregir la que estamos valorando.

4.5 Producción.

Antes que nada, queremos comentar que en el primer cuatrimestre cogimos algo de práctica en programación en J2ME. En la asignatura de Integración de Medios Digitales, realizamos un proyecto en este lenguaje. Consistía en programar una aplicación que permitirá, por comunicación bluetooth, comunicar dos terminales móviles a modo de chat. Este tipo de comunicación no tiene nada que ver con la que hemos utilizado en este proyecto (Wi-fi), pero nos sirvió para coger experiencia programando Midlets y manejar las funciones principales del J2ME. Además, también empezamos a manejar el entorno NetBeans, con lo que teníamos trabajo adelantado a la hora de realizar este proyecto, sobre todo en lo que respecta a la implementación.
En esta sección vamos a describir la implementación de nuestro producto. Describiremos las clases creadas y para que se han utilizado, las interfaces de usuario, como se ha hecho la comunicación y la gestión de los datos.

4.5.1 Clases.

En realidad, todo el proyecto ha constado de 4 clases. Una para el cliente, una para el administrador y otras dos para la gestión de las conexiones al servidor. Las clases que utilizarán los clientes y los administradores, mejor dicho los midlets que utilizarán, los hemos llamado Cliente y Admin. En cuanto al servidor, hemos necesitado una clase para implementar los métodos que accederán a la base de datos y hacer las gestiones oportunas en ella y otra clase ha sido utilizada para poder poner en práctica el sistema de acceso a datos remotos, RMI.

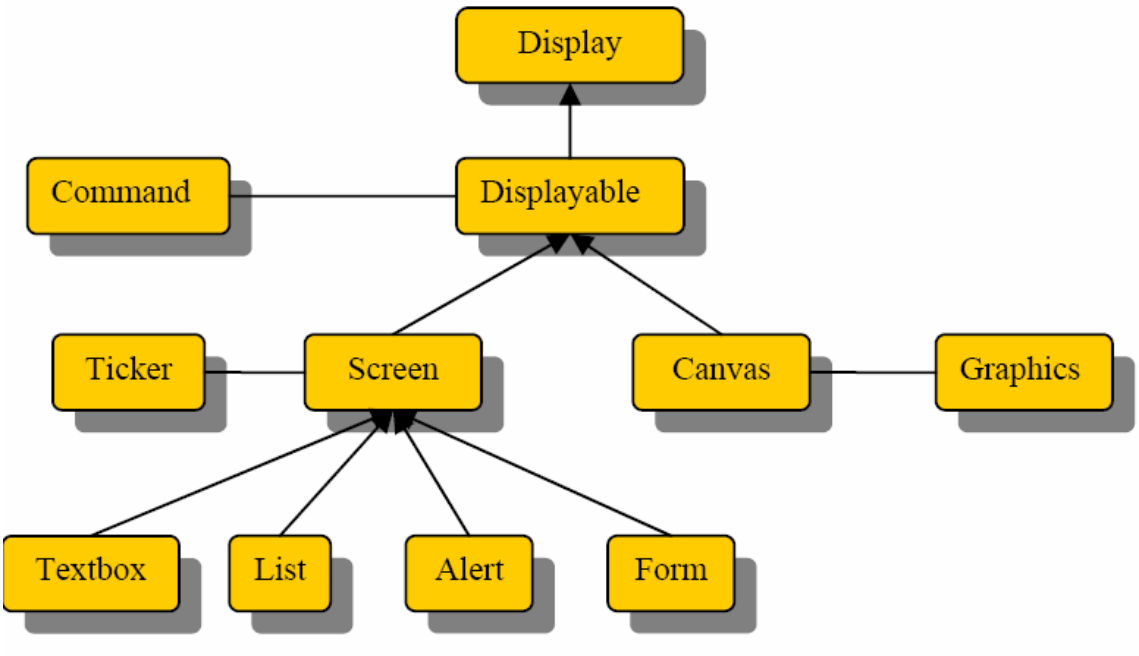
4.5.2 Interfaz y menús.

Vamos a explicar las diferentes clases y métodos que he utilizado para crear las interfaces y los menús de las aplicaciones del Cliente y del Admin. Decir que los métodos y clases utilizados han sido los mismos, aunque quizás en la aplicación del Administrador haya más elaboración de la aplicación, ya que su implementación es un poco más complicada que la del Cliente, ya que está el tema de la valoración de las noticias y del registro de otros miembros.

En este apartado voy a centrarme en el paquete javax.microedition.lcdui (Interfaz de usuario con pantalla LCD), que es donde se encuentran los elementos gráficos que vamos a utilizar en nuestras aplicaciones.
Hay que diferenciar en interfaces de usuario de alto nivel y de bajo nivel. Teniendo en cuenta la diversidad de aplicaciones se pueden realizar para los dispositivos MID (llamare así a los dispositivos que soportan MIDP), y los elementos que nos proporcionan la configuración CLDC y el perfil MIDP, se dividen estos elementos en dos grupos:
Por un lado tenemos las interfaces de usuario de bajo nivel. Al crear una aplicación usando las APIs de bajo nivel, tendremos un control total de lo que aparecerá por pantalla. Estas APIs nos darán un control completo sobre los recursos del dispositivo y podremos controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas. Generalmente, estas APIs se

utilizan para la creación de juegos donde el control sobre lo que aparece por pantalla y las acciones del usuario juegan un papel fundamental.

Para nuestras aplicaciones hemos utilizado los elementos que componen la interfaz de usuario de alto nivel. Esta interfaz usa componentes tales como botones, cajas de texto, formularios, etc. Estos elementos son implementados por cada dispositivo y la finalidad de usar las APIs de alto nivel es su portabilidad. Al usar estos elementos, perdemos el control del aspecto de nuestra aplicación ya que la estética de estos componentes depende exclusivamente del dispositivo donde se ejecute. En cambio, usando estas APIs de alto nivel ganaremos un alto grado de portabilidad de la misma aplicación entre distintos dispositivos. Fundamentalmente, se usan estas APIs cuando queremos construir aplicaciones de negocios.



4.7- Jerarquía de las clases de J2ME.

El paquete javax.microedition.lcdui incluye las clases necesarias para crear interfaces de usuario, tanto de alto nivel como de bajo nivel.

- DISPLAY

La clase Display representa el manejador de la pantalla y los dispositivos de entrada. Todo MIDlet debe poseer por lo menos un objeto Display. En este objeto Display podemos incluir tantos objetos Displayable como queramos. La clase Display puede obtener información sobre las características de la pantalla del dispositivo donde se ejecute el MIDlet, además de ser capaz de mostrar los objetos que componen nuestras interfaces.
Como hemos dicho antes un MIDlet debe de poseer una instancia del objeto Display. Además tendremos que hacer referencia a la pantalla que queramos que este activa, haciendo uso del método setCurrent().

- COMMAND Y COMMANDLISTENER

Un objeto de la clase Command mantiene información sobre un evento. Podemos pensar en él como un botón de Windows, por establecer una analogía.

Generalmente, los implementaremos en nuestros MIDlets cuando queramos detectar ejecutar una acción simple.
Existen tres parámetros que hay que definir cuando construimos un objeto

Command:

- Etiqueta: La etiqueta es la cadena de texto que aparecerá en la pantalla del dispositivo que identificara a nuestro Command.

- Tipo: Indica el tipo de objeto Command que queremos crear. La declaración del tipo sirve para que el dispositivo identifique el Command y le dé una apariencia específica acorde con el resto de aplicaciones existentes en el dispositivo.

- Prioridad: Es posible asignar una prioridad específica a un objeto Command.

Esto puede servirle al AMS para establecer un orden de aparición de los Command en pantalla. A mayor número, menor prioridad.
No basta solo con crear un objeto Command de un determinado tipo para que realice la acción que nosotros deseamos, de acuerdo a su tipo. Para ello tenemos que implementar la interfaz CommandListener.

En cualquier MIDlet que incluyamos Commands, tendremos además que implementar la interfaz CommandListener. Como sabemos, una interfaz es una clase donde todos sus métodos son declarados como abstract. Es misión nuestra implementar sus correspondientes métodos. En este caso, la interfaz CommandListener solo incluye un método commandAction(Command c, Displayable d) en donde indicaremos la acción que queremos que se realice cuando se produzca un evento en el Command c que se encuentra en el objeto Displayable d.

La clase Screen es la superclase de todas las clases que conforman la interfaz de usuario de alto nivel.
El ticker es una cadena de texto que se desplaza por la pantalla de derecha a izquierda.

- LA CLASE ALERT

El objeto Alert representa una pantalla de aviso. Normalmente se usa cuando queremos avisar al usuario de una situación especial como, por ejemplo, un error. Un Alert está formado por un título, texto e imágenes si queremos.

Además podemos definir el tiempo que queremos que el aviso permanezca en pantalla, diferenciando de esta manera dos tipos de Alert:

1. Modal: La pantalla de aviso permanece un tiempo indeterminado hasta que es cancelada por el usuario. Esto lo conseguimos invocando al método Alert.setTimeOut(Alert.FOREVER).

2. No Modal: La pantalla de aviso permanece un tiempo definido por nosotros. Para ello indicaremos el tiempo en el método setTimeOut(tiempo). Una vez finalizado el tiempo, la pantalla de aviso se eliminará de pantalla y aparecerá el objeto Displayable que nosotros definamos.

Podemos elegir el tipo de alerta que vamos a mostrar. Cada tipo de alerta tiene asociado un sonido.

- CREACIÓN DE FORMULARIOS

Un formulario (clase Form) es un componente que actúa como contenedor de un número indeterminado de objetos. Aquí tendremos que añadir todos los objetos que queremos que se muestren por pantalla, como los Command, Gauge, Ticket, …

El número de objetos que podemos insertar en un formulario es variable pero, teniendo en cuenta el tamaño de las pantallas de los dispositivos MID, se recomienda que se numero sea pequeño para evitar así el scroll que se produciría si insertáramos demasiados objetos en un formulario.

La clase ImageItem nos da la posibilidad de incluir imágenes en un formulario.

Antes hemos tenido que crear la Imagen que queremos mostrar por pantalla. Utilizamos el método createIamge de la clase Image, donde tenemos que indicar la ruta donde está la imagen creada. Las imágenes y los recursos los meteremos en la carpeta res(resources) del proyecto.

La clase Gauge implementa un indicador de progresos a través de un gráfico de barras. El componente Gauge representa un valor entero que va desde 0 hasta un valor máximo que definimos al crearlo. Un Gauge puede ser:

- Interactivo: En este modo, el usuario puede modificar el valor actual del Gauge. En la mayoría de los casos, el usuario dispondrá de unos botones con los cuales podrá modificar el valor a su antojo, pero siempre dentro del rango que se estableció al crear el componente Gauge.

- No Interactivo: En este modo, el usuario no puede modificar el valor del Gauge. Esta tarea es realizada por la aplicación que actualiza su valor a través del método Gauge.setValor(int valor). Este modo es muy útil cuando queremos indicar el progreso en el tiempo de una tarea determinada.

4.5.3 Comunicación WIFI. Ejemplo de comunicación RMI.

La comunicación entre los terminales y el servidor se realizará, como hemos dicho antes, mediante el protocolo TCP/IP. Para ello debemos tener conectividad a internet. Éste tipo de conexiones, las gestiona directamente el teléfono móvil, siendo prácticamente transparente para el usuario. Además, lo único que tendremos que pasarle a las aplicaciones por código es la dirección IP del servidor. Las conexiones a este, serán realizadas mediante el mecanismo RMI, por tanto, serán prácticamente invisibles tanto para el programador como, evidentemente, el usuario final de la aplicación.

Vamos a hacer un ejemplo muy tonto de RMI, para introducirnos en cómo funciona. Haremos una clase capaz de sumar ni más ni menos que dos enteros. Haremos que esta clase quede registrada como objeto remoto, por lo que podrá ser llamada desde otros ordenadores. Haremos un cliente capaz de llamar a esa clase para sumar 2 + 3.

Vamos a ver qué clases necesitamos para hacer nuestros clientes y servidores de rmi, sin extendernos demasiado. Luego veremos cada uno de estos puntos por detallado.

- InterfaceRemota. Es una interface java con todos los métodos que queramos poder invocar de forma remota, es decir, los métodos que queremos llamar desde el cliente, pero que se ejecutarán en el servidor.

- ObjetoRemoto. Es una clase con la implementación de los métodos de InterfaceRemota. A esta clase sólo la ve el servidor de rmi.

- ObjetoRemoto_Stub. Es una clase que implementa InterfaceRemota, pero cada método se encarga de hacer una llamada a través de red al ObjetoRemoto del servidor, esperar el resultado y devolverlo. Esta clase es la que ve el cliente y no necesitamos codificarla, java lo hace automáticamente para nosotros a partir de ObjetoRemoto.

En el PC/máquina servidor de rmi deben correr dos programas

- rmiregistry. Este es un programa que nos proporciona java (está en JAVA_HOME/bin, siendo JAVA_HOME el directorio en el que tengamos instalado java). Una vez arrancado, admite que registremos en él objetos para que puedan ser invocados remotamente y admite peticiones de clientes para ejecutar métodos de estos objetos.

- Servidor. Este es un programa que debemos hacernos nosotros. Debe instanciar el ObjetoRemoto y registrarlo en el rmiregistry. Una vez registrado el ObjetoRemoto, el servidor no muere, sino que queda vivo. Cuando un cliente llame a un método de ObjetoRemoto, el código de ese método se ejecutará en este proceso.

En el pc/máquina del cliente debe correr el programa Cliente. Este programa debemos hacerlo nosotros. Pide al rmiregistry del PC/máquina servidor una referencia remota al ObjetoRemoto. Una vez que la consigue (en

realidad obtiene un ObjetoRemoto_Stub), puede hacer las llamadas a sus métodos. Los métodos se ejecutarán en el Servidor, pero Cliente quedará bloqueado en cada llamada hasta que Servidor termine de ejecutar el método.

Lo primero que tenemos que hacer es una interface con los métodos que queremos que se puedan llamar remotamente.

import java.rmi.Remote; public interface InterfaceRemota extends Remote { public int suma (int a, int b) throws java.rmi.RemoteException; }

Tiene que heredar de la interface **Remote** de java, si no el objeto no será remoto. Añade además los métodos que queramos, pero todos ellos deben lanzar la excepción **java.rmi.RemoteException**, que se producirá si hay algún problema con la comunicación entre los dos ordenadores o cualquier otro problema interno de rmi o en el servidor.

Todos los parámetros y valores devueltos de estos métodos deben ser tipos primitivos de java o bien clases que implementen la interface **Serializable** de java. De esta forma, tanto los parámetros como el resultado podrán "viajar" por red del cliente al servidor y al revés. También se admiten los que implementen la interface **Remote**, pero ya lo veremos más adelante. De momento, para este ejemplo, nos conformamos con tipos primitivos, en concreto, unos int.

Hay que hacer una clase que implemente esa **InterfaceRemota**, es decir, que tenga los métodos que queremos llamar desde un cliente **rmi**. El servidor de **rmi** se encargará de instanciar esta clase y de ponerla a disposición de los clientes. Esa clase es la que llamamos "**objeto remoto**".

Esta clase remota debe implementar la interface remota que hemos definido (y por tanto implementará también la interface **Remote** de java), pero también debe hacer ciertas cosas bien. Debe definir métodos como **hashCode()**, **toString()**, **equals()**, etc de forma adecuada a un objeto remoto. También debe tener métodos que permita obtener referencias remotas, etc, etc. Es decir, una serie de cosas más o menos complejas y de las que afortunadamente no tenemos que preocuparnos. La forma sencilla de hacer esto es hacer que la clase herede de **UnicastRemoteObject**.

import java.io.Serializable; public class ObjetoRemoto extends UnicastRemoteObject implements InterfaceRemota { public int suma(int a, int b) { System.out.println ("sumando " + a + " + " + b + "..."); return a+b; } }

Lo siguiente que tenemos que hacer es ejecutar un programa java que instancie y registre el objeto remoto. Este programa java debe hacer lo siguiente ;

Indicar cuál es el path en el que rmiregistry puede encontrar la clase correspondiente al objeto remoto. En nuestro caso, el path en el que se puede encontrar **ObjetoRemoto_Stub.class**. Dicho path se da en formato URL, por lo que no admite espacios ni caracteres extraños (Quizás admita los típicos %20% que se ven en el navegador cuando hay espacios en la url, pero no he probado). El path, en formato URL, puede ser algo como esto

"**file:/D:/users/javier/prueba_servidor**/". Consiste en fijar ua propiedad de nombre **java.rmi.codebase** con el path donde se encuentran los ficheros **.class** remotos. Es imortante la barra del final, porque si no, fallará luego el servidor.

Aquí también podemos poner "**http://un_host/un_path/**" si queremos que nuestra clase **ObjetoRemoto_Stub.class** esté accesible desde un servidor web. En cualquier caso, el path que pongamos aquí debe estar accesible para el programa **rmiregistry** cuando lo lancemos.

Instanciar una clase remota y luego registrarla en **rmiregistry:**
ObjetoRemoto objetoRemoto = new ObjetoRemoto(); Naming.rebind ("ObjetoRemoto", objetoRemoto);

La instanciación no tiene problemas. Para registrarla hay que llamar al método estático **rebind()** de la clase **Naming**. Se le pasan dos parámetros. Un nombre para poder identificar el objeto y una instancia del objeto. El nombre que hemos dado debe conocerlo el cliente, para luego poder pedir la instancia por el nombre. El método **rebind()** registra el objeto en **rmiregistry**. Si ya estuviera registrado, lo sustituye por el que acabmos de pasarle.

En donde pone "ObjetoRemoto" podría ponerse "//localhost/ObjetoRemoto" o bien "//un_host/ObjetoRemoto". Si no ponemos nada se sobreentiende localhost. Si ponemos algo, es el host donde se buscará rmiregistry para registrar el objeto. El host que pongamos aquí debe ser el host en el que arranquemos **rmiregistry**.

Ya está todo lo que tiene que hacer el programa que registra el objeto.
Ahora sólo tenemos que hacer el programa que utilice este objeto de forma remota. Los pasos que debe realizar este programa son los siguientes: Pedir el objeto remoto al servidor de rmi. El código para ello es sencillo

InterfaceRemota objetoRemoto = (InterfaceRemota)Naming.lookup ("//host_servidor/ObjetoRemoto");

Simplemente se llama al método estático **lookup()** de la clase **Naming**. Se le pasa a este método la URL del objeto. Esa URL es el nombre (o IP) del host donde está arrancado **rmiregistry** y por último el nombre con el que se registró anteriormente el objeto (en mi caso **ObjetoRemoto**).

Este método devuelve un **Remote**, así que debemos hacer un "cast" a **InterfaceRemota** para poder utilizarlo. El objeto que recibimos aquí es realmente un **ObjetoRemoto_Stub**. Por ello, **ObjetoRemoto_Stub.class** debe estar en el CLASSPATH del cliente. Ya podemos llamar al método de **suma()**.
System.out.print ("2 + 3 = "); System.out.println (objetoRemoto.suma(2, 3));
Para que el código del cliente compile necesita ver en su classpath a **InterfaceRemota.class**. Para que además se ejecute sin problemas necesita además ver a **ObjetoRemoto_Stubs.class**, por lo que estas clases deben estar accesibles desde el servidor o bien tener copias locales de ellas.

4.5.4 Organización y manejo de los datos.

La organización que íbamos a seguir para ordenar los archivos dentro del servidor, ha sido una de las tareas que más quebraderos de cabeza nos ha traído. En un primer momento, pensábamos en hacer una hoja XML por cada noticia publicada y éstas hojas que fueran almacenadas dentro del servidor en carpetas por fecha. La finalidad que venía ligada al uso del XML, era el poder transmitir estas mismas hojas a los dispositivos móviles, para realizar posteriores gestiones de datos.

Esta opción, después de una serie de valoraciones, fue abandonada, ya que además de tener una complejidad excesiva, nos cerraba las puertas de usar una base de datos convencional, que era lo que más habíamos utilizado en la universidad y a lo que mejor nos podíamos adaptar. Además, nos ofrecía un abanico de opciones de búsqueda de datos mucho más amplio que el que podíamos utilizar con el XML.

Por tanto, el almacenamiento de datos estará en un servidor, y en la base de datos, para cada noticia, se van a guardar los siguientes datos:

- Nombre de la noticia: Identificador principal de la noticia.

- Usuario: Cliente que ha subido la noticia al servidor.

- Fecha de subida: Fecha en que un determinado cliente subió la noticia al servidor.

- Contenido: Campo dedicado para que los usuarios escriban sus opiniones o noticias.

Gracias a todos estos datos, tanto la aplicación administradora como la del cliente tendrán muchísimos parámetros de búsqueda de noticias.

Aún asi continuábamos teniendo un problema de bastante envergadura. La tecnología J2ME, es decir la plataforma java para dispositivos móviles no es capaz de comunicarse mediante un servidor MySQL directamente. Por lo cual tuvimos que realizar la costosa tarea de diseñar y programar un Proxy entre el servidor MySQL y nuestra aplicación. A este servidor se conecta nuestra aplicación mediante sockets TCP, elegimos TCP simplemente porque la información puede viajar cifrada de forma natural, convierte los datos que la aplicación proporciona y los envía mediante consultas SQL a la base de datos recogiendo sus respuestas y enviándolas de nuevo a la aplicación.

4.6. Conclusiones.

Las conclusiones que podemos sacar de este proyecto, pensamos que son todas positivas, al tratarse de un proyecto formativo. Si se tratase de un proyecto serio, es decir, un proyecto del que dependiera nuestra carrera laboral, ya no serían tan positivas, puesto que no hemos cumplido con las fechas y con los objetivos como se pretendía.

Sin embargo, vamos a centrarnos en la parte positiva. Hemos cogido más experiencia de la que teníamos tanto en Java con en J2ME, dos lenguajes claramente en auge, que pueden sernos de gran utilidad en un futuro. Además, hemos trabajado sobre el entorno gratuito NetBeans, que nos puede servir en más de una ocasión para realizar algún proyecto más.

También hemos adquirido experiencia sobre bases de datos, ya que en la facultad se tocan, pero no tenemos consciencia real de para qué nos van a servir realmente en un futuro. Este proyecto, sin una base de datos y sin saber gestionarla adecuadamente, sería prácticamente irrealizable. Además, al igual que con el NetBeans en Java, hemos aprendido a manejar el entorno SQLServer Management, que también nos puede servir en un futuro.

En resumen, el balance del proyecto pensamos que es positivo por la experiencia y los conocimientos adquiridos, pero tenemos que ser críticos con nosotros mismos y saber que no hemos cumplido con los plazos como se esperaba en un principio.

5. Bibliografía.

Las referencias bibliográficas que hemos utilizado para la realización del proyecto son las siguientes.

- www.infotutoriales.com: Desde aquí hemos descargado diversos tutoriales que nos han sido de gran utilidad, como por ejemplo, el manual cinco estrellas de SQL Server 2005.

- http://users.dsic.upv.es/~cgarcia/PAI/: Ésta es la página personal de Carlos García, profesor de la asignatura de Edición y Postproducción de audio. De aquí hemos obtenido gran información sobre Java y RMI.

- http://www.it.uc3m.es/nati/docencia/ittt/swcom/sesiones/7_Programa_MIDP20.pdf: Presentación que nos ha servido de gran ayuda con la programación en MIDP2.0.

- www.wikipedia.es: Como siempre, la Wikipedia. Cualquier información sobre lenguajes que podíamos utilizar, bases de datos, etc.. encontrábamos la descripción en esta enciclopedia online.

- Página Web de la asignatura Servidores Web de la Escuela de Informática del Politécnico: Aquí hemos encontrado muchísima información sobre como programar servidores, aunque posteriormente, gracias al Sql Server Management, no lo hemos utilizado mucho.

- Diversos documentos que teníamos sobre un proyecto que realizamos en la asignatura de Integración de Medios Digitales, sobre todo manuales de J2ME.

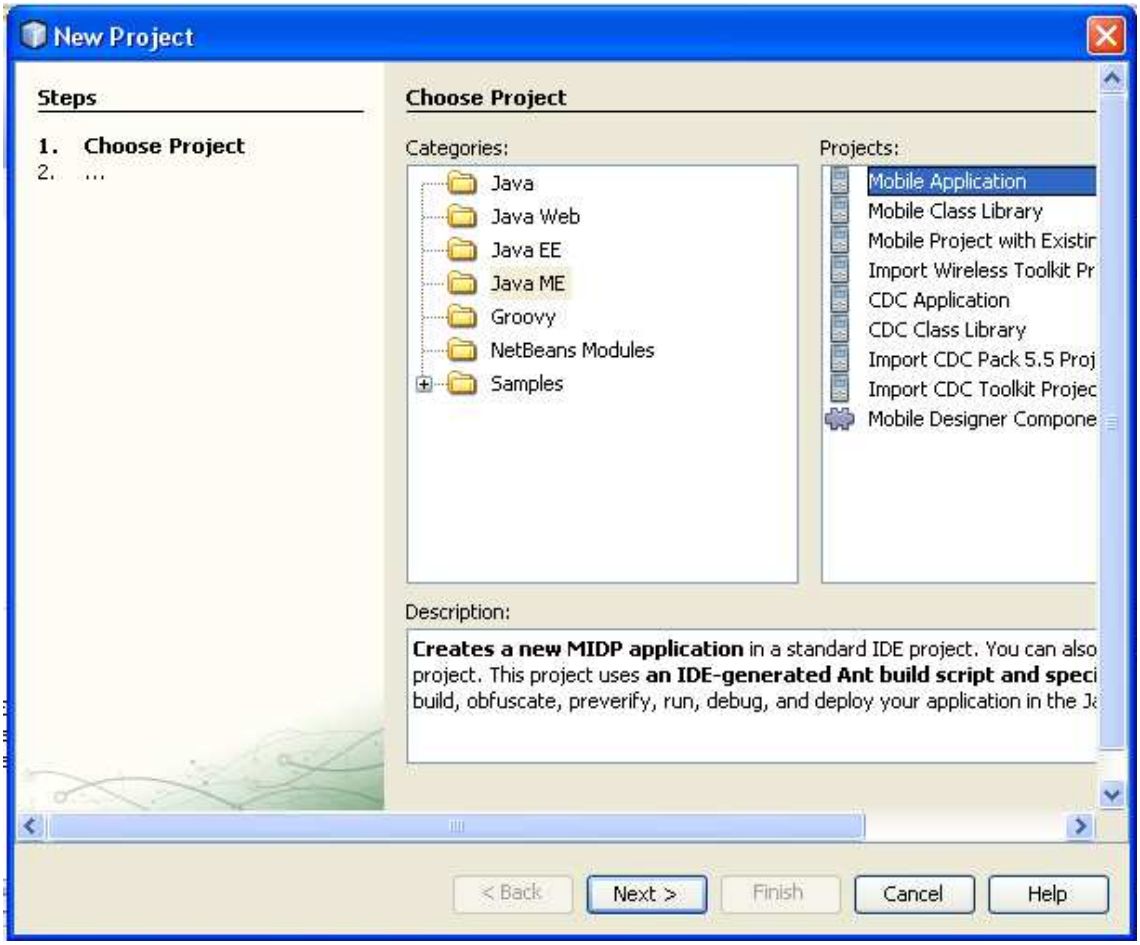
- Páginas Webs diversas como: http://www.forum.nokia.com/, http://www.j2mepolish.org/devices/devices-btapi.html, http://www.javahispano.org, http://proyectandoj2me.blogspot.com/, http://leo.ugr.es/J2ME/MIDP/aplicaciones.htm, etc...

- Dos libros sacados de la biblioteca de informática, aunque posteriormente no se les ha sacado mucho partido: “Java y XML” y “Desarrollo de soluciones XML “.

6.1 Apéndice NetBeans 6.5

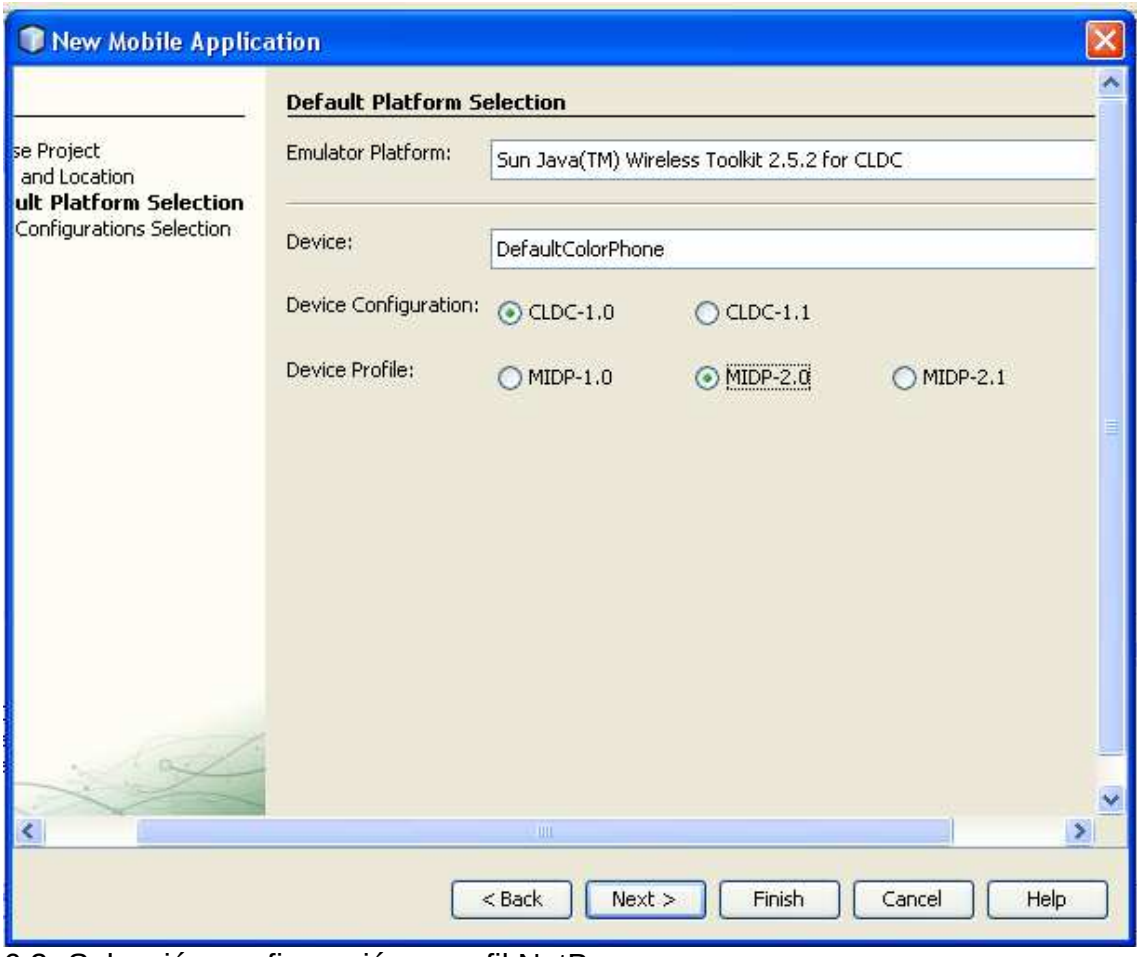
Net Beans 6.5 nos provee de las librerías básicas para la utilización del jsr-82, además de las librerías javax.microedition. Además, también contiene el emulador Wireless Toolkit integrado, con lo que podemos probar las aplicaciones que programamos sin necesidad de instalarlas en ningún dispositivo móvil. Podemos descargar NetBeans 6.5 de forma totalmente gratuita desde http://www.netbeans.org/downloads/. Es importante descargar la versión que contenga java ME. Además, en el ordenador a instalar, debemos tener instalada una versión actualizada de J2SE, que como se ha dicho antes es una versión que contiene J2ME. En caso de querer instalar el J2SE, lo podemos descargar de http://java.sun.com/j2se/downloads.html, donde todas las descargas son gratuitas. Una vez instalado, ya podemos empezar a utilizar NetBeans para crear nuestras aplicaciones.

En primer lugar, debemos crear un nuevo proyecto. En nuestro caso seleccionaremos un proyecto de tipo Java ME y la opción Mobile Aplication.



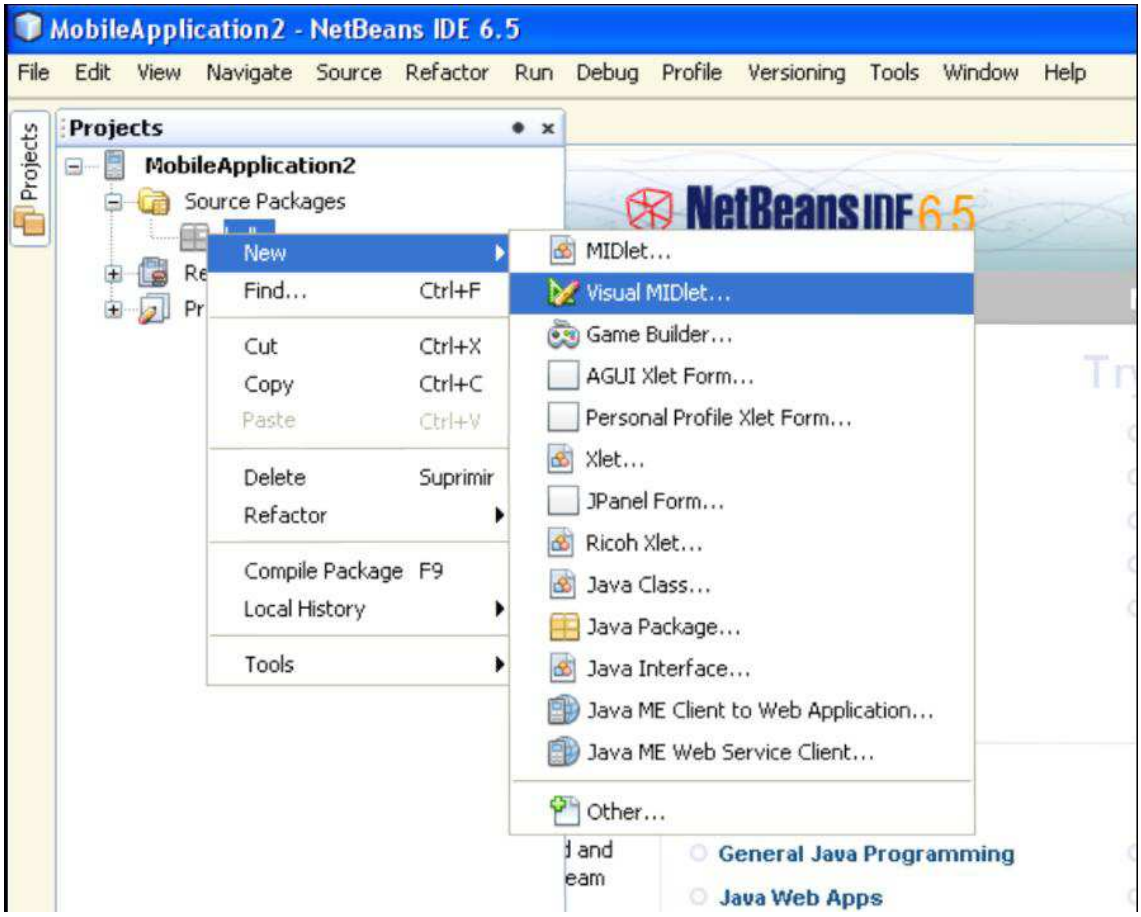
6.1- Selección lenguaje aplicación NetBeans

Seguidamente, seleccionaremos las opciones CDLC 1.0 (para que pueda ser ejecutado en mayor cantidad de dispositivos móviles) y MIDP 2.0 para que nos deje manejar las librerías de RMI.



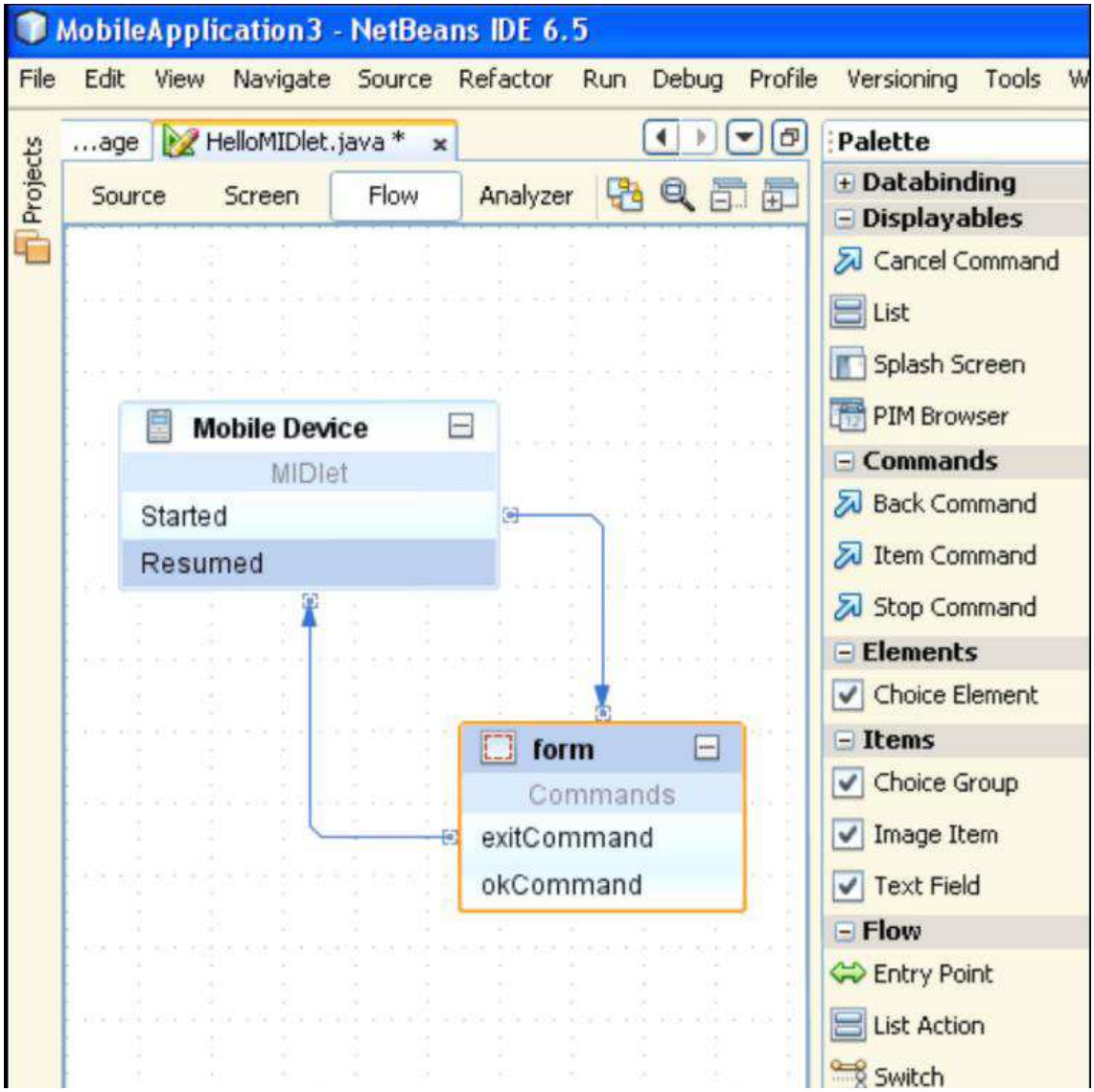
6.2- Selección configuración y perfil NetBeans

Una vez hecho esto, ya podemos empezar nuestro proyecto. Podemos empezar nuestro proyecto escribiendo el código desde cero o bien utilizar la ayuda que nos proporciona NetBeans para crear aplicaciones. Si queremos empezar el proyecto desde cero, simplemente borramos el archivo que nos genera automáticamente NetBeans y creamos un nuevo MIDlet. A partir de este creamos más clases o lo que se desee.



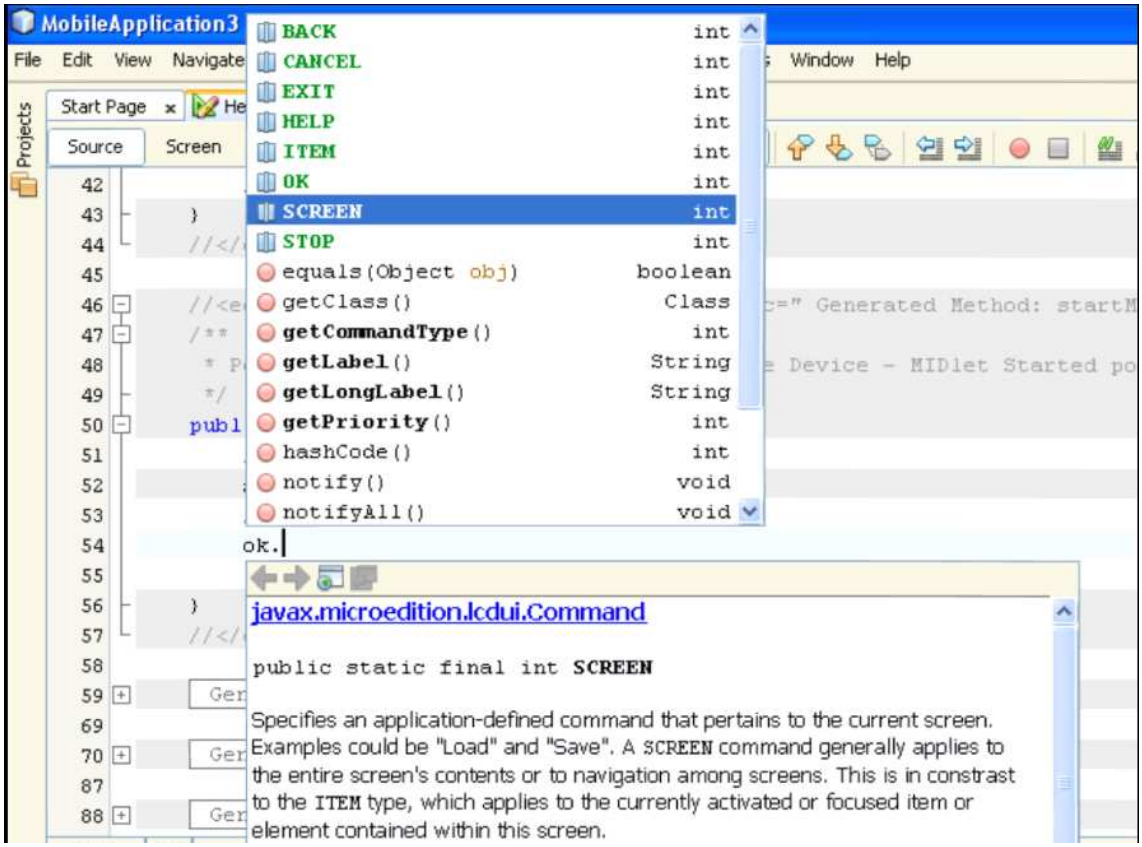
6.3- Nuevo MIDLET NetBeans

Sin embargo, NetBeans nos puede servir de gran ayuda a la hora de crear aplicaciones de una forma más visual e interactiva. Podemos empezar seleccionando la opción “flow” del menú i agregar los objetos y las clases que deseemos a nuestro MIDlet. Se les puede dar también cierta funcionalidad a comandos u otras funciones, aunque la gran parte de la funcionalidad se la tendremos que dar por código. Sin embargo, NetBeans también nos generará cierto código automáticamente, facilitándonos y quitándonos el trabajo.



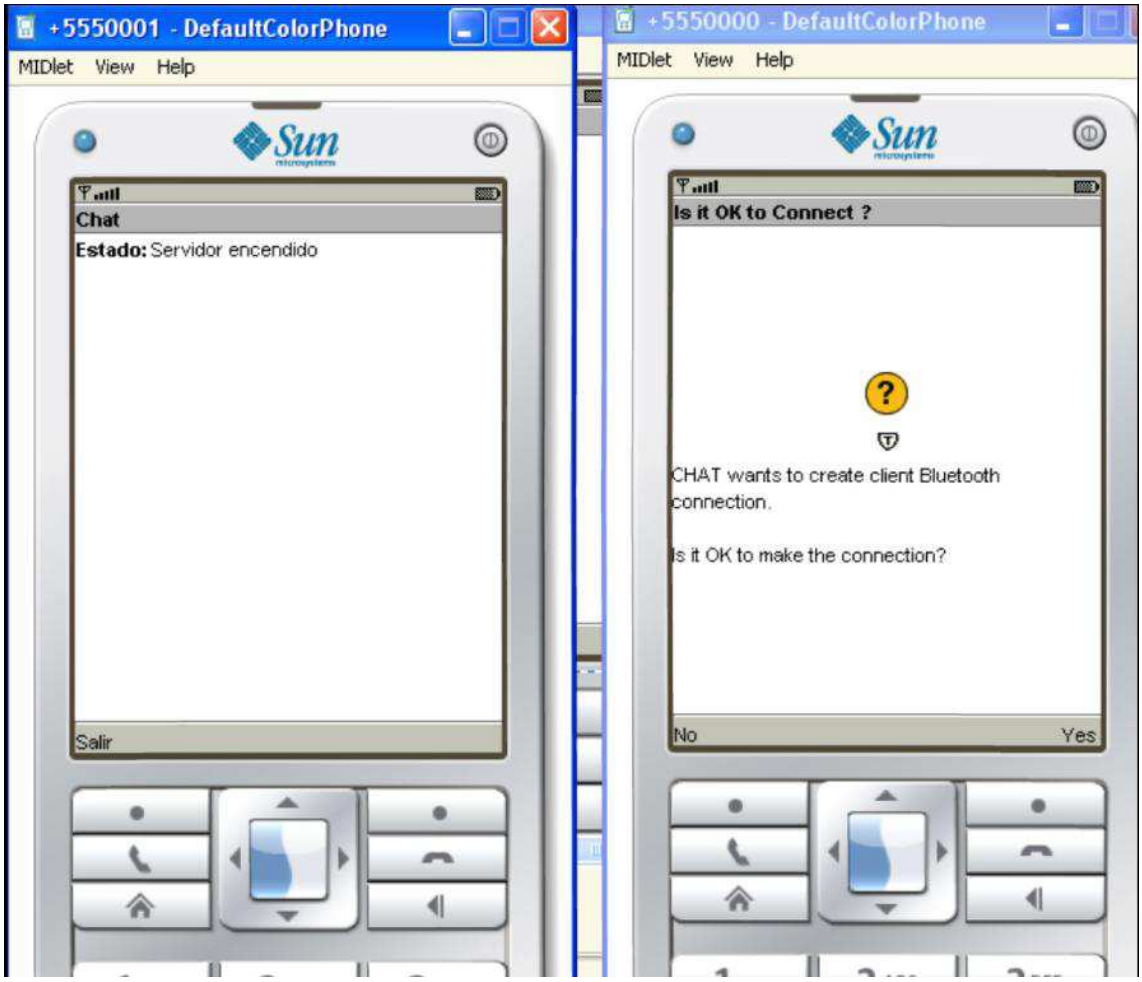
6.4- Modo Flow NetBeans

Otra función que estaría bien destacar es la de predicción de texto que tiene NetBeans. Después de escribir un punto detrás de una variable o método, sale un menú con las posibles alternativas a utilizar. Esto es muy aprovechable cuando no se conocen ampliamente las clases sobre las que se está programando o simplemente para no perderse ni cometer errores. Podríamos decir que utiliza la llamada predicción de Microsoft Visual Studio.



6.5- Predicción de texto NetBeans.

Por último, NetBeans lleva integrado el simulador Wireless Toolkit, como se ha comentado anteriormente. Podemos realizar simulaciones de nuestro código fácilmente simplemente dándole al boto “Run”. Nos sale un teléfono móvil en nuestra pantalla del ordenador y podemos usarlo tal como si fuera uno real. Además, podemos utilizarlo para simular comunicaciones bluetooth entre dos o más dispositivos móviles, comunicaciones Wireless con un servidor e incluso mandar mensajes de texto a otros dispositivos. Es una herramienta muy valiosa.

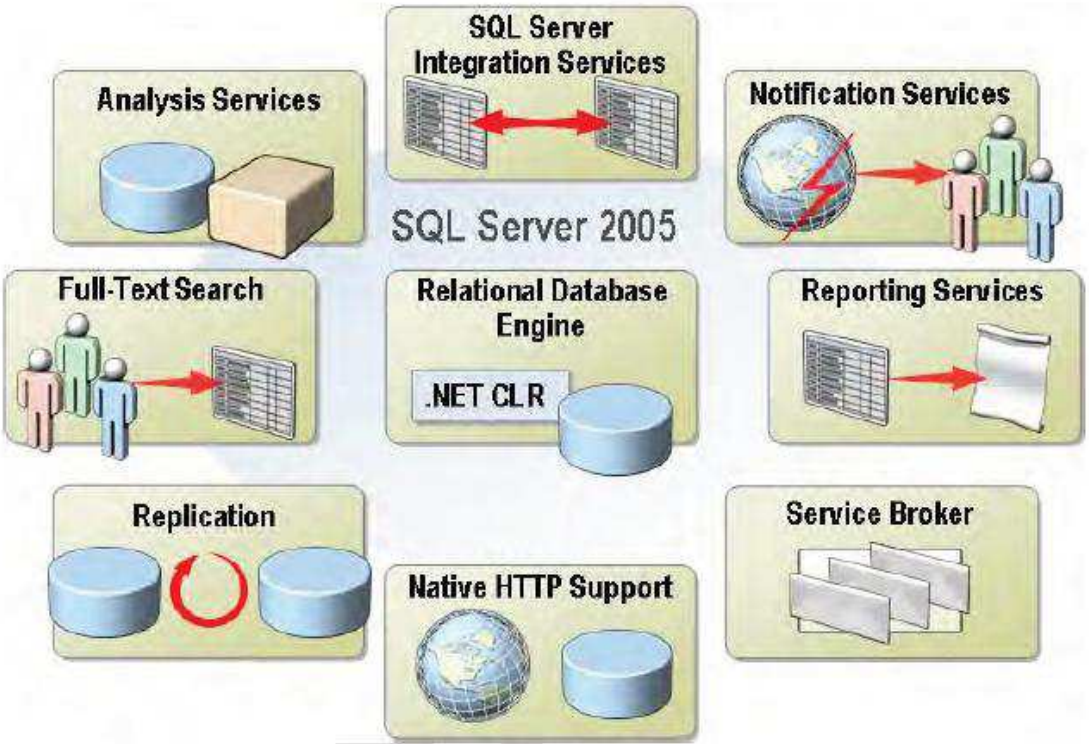


6.6- Wireless Toolkit, Simulador de NetBeans.

6.2 SQL Server 2005.

En este apéndice, vamos a describir brevemente la aplicación SQL Server, en concreto, la versión 2005. Existen más versiones, entre ellas la 2008, más actualizada, pero en nuestro proyecto hemos creído más conveniente utilizar la 2005 por estar más extendida y haber más información de ella. Pero, ¿Qué es SQL Server?

SQL Server es más que un sistema de administración de bases de datos. Incluye componentes múltiples y servicios los cuales la hacen una plataforma comprensiva para programas de empresa.



6.7- Microsoft Sql Server 2005.

SQL Server 2005 se compone de los componentes siguientes.

Componente	Descripción
Motor de Base de datos Relacional	El motor de base de datos relacional de SQL Server es el corazón de SQL Server 2005 y proporciona un ambiente de alto rendimiento, scalable, seguro para almacenar y recuperar dator de modificación relacional o formato Extensible Markup Language (XML)
Analysis Services	Proporciona la base de una solucion business intelligence para soporte de Online Analytical processing (OLAP) aplicaciones y data mining.
SQL Server Integration Services (SSIS)	Un motor para importar y exportar datos soluciones y transformaciones de datos mientras que se transfieren.
Notification Services	Un framework para las soluciones en las cuales se envían a los suscriptores las notificaciones cuando ocurren los acontecimientos específicos. Las notificaciones se pueden generar eficientemente y enviar a dispositivo múltiples de diferentes tipos.
Reporting Services	Se utiliza para extraer datos desde SQL Server y generar reportes.
Service Broker	Un mecanismo confiable de queuing, y comunicación transaccional basada en mensajes entre los servicios de software.
.NET common language runtime(CLR)	Incluido adentro de SQL Server, permitiendo poner soluciones de base de datos en ejecución usando el código manejado escrito en .NET language por ejemplo Microsoft Visual C#® .NET o Microsoft Visual Basic® .NET.
Native HTTP Support	Permite a programas de cliente conectarse con HTTP endpoints dentro de SQL Server sin requerir Internet Information Services (IIS).
Replicación	Un sistema de tecnologías para el copiado de datos y distribucion de base de datos a partir de una base de datos o servidor a otro y sincronizando entre las bases de datos para asegurar consistencia.

El motor de la base de datos es el componente principal de SQL Server. Proporciona almacenaje de datos, recuperación, y servicios de modificación que pueden escalar desde soluciones personales hasta el nivel de empresa.
Es importante, antes de instalar SQL Server, saber para qué vamos a utilizar la plataforma. Dependiendo del uso que vayamos a darle, tendremos que instalar

una versión determinada u otra. En la tabla siguiente, se muestra una descripción de las distintas versiones disponibles.

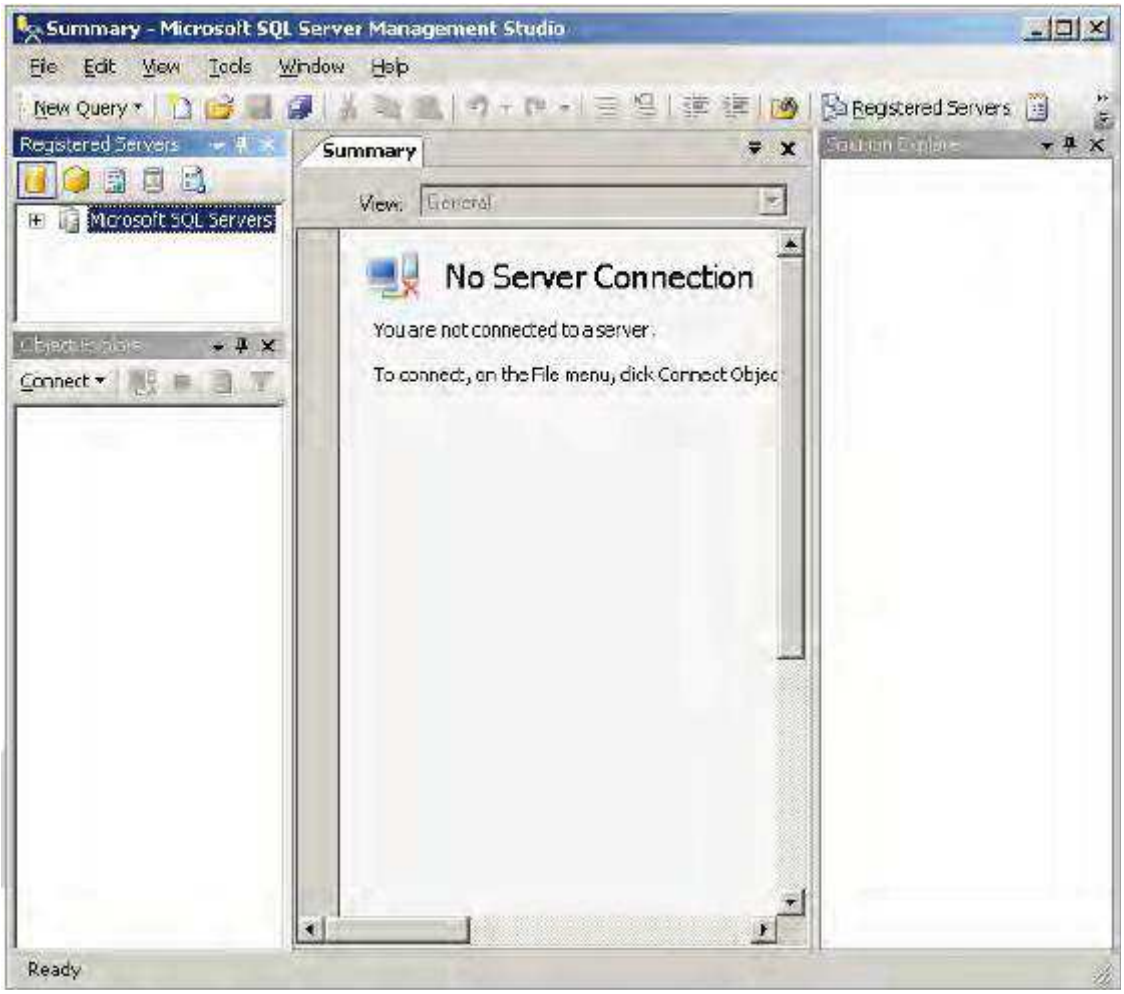
Edicion SQL Server 2005	Descripción
Enterprise Edition (Disponible en versiones 32-bit y 64-bit)	Una versión comprensiva de SQL Server diseñada para altos niveles de performance. Use esta edición para grandes escalas, niveles empresarios, aplicaciones criticas. La Enterprise Edition contiene todos las ventajas de la edición Standard, como así también las de Enterprise, incluyendo: Failover clustering Database mirroring Snapshot databases Mirrored backups Online page and file restore Distributed partitioned views Heterogeneous replication Peer-to-peer replication
Standard Edition (Disponible en versiones 32-bit y 64-bit versions)	Diseñado para aplicaciones de trabajo de grupo y departamentales. Use esta edición si no necesita los niveles de performance y disponibilidad ofrecidos por la Enterprise Edition.
Express Edition (Disponible en 32-bit)	Una versión de SQL Server 2005 para clientes que no están conectados o aplicaciones que corren solas.
Mobile Edition	Una edición compacta que provee administración de datos para equipos inteligentes. Esta edición es capaz de replicar datos con SQL Server 2005 y SQL Server 2000, permitiendo a los usuarios mantener un mobile data store que puede ser sincronizado con enterprise data.
Developer Edition (Available in 32-bit and 64-bit versions)	Incluye todas las funcionalidades de la Enterprise Edition, pero esta licenciado para ser usado como un sistema de testeo y desarrollo, no como un servidor de producción. Use esta edición para desarrollar y testear soluciones para base de datos.

Queremos destacara que Microsoft SQL Server es una herramienta privada, es decir, de pago. Para llevar a cabo nuestro proyecto hemos usado una versión de evaluación de 120 días, disponible en la Web de Microsoft. Por último, añadir que para la gestión de las bases de datos de SQL Server, haremos servir la herramienta SQL Mannagement Studio, descrita en el anexo siguiente.

6.3 SQL Server Management Studio.

El SQL Server Management Studio es un entorno integrado para acceder, configurar, administrar y manejar todos los componentes de SQL Server 2005. Combina las capacidades de Enterprise Manager, Query Analyzer

y Analisis Manager, las cuales eran provistas por las versiones previas de SQL Server.



6.8- Sql Server Management

Las características principales del SQL Server Management son:

- Administración completa de bases de datos relacionales.

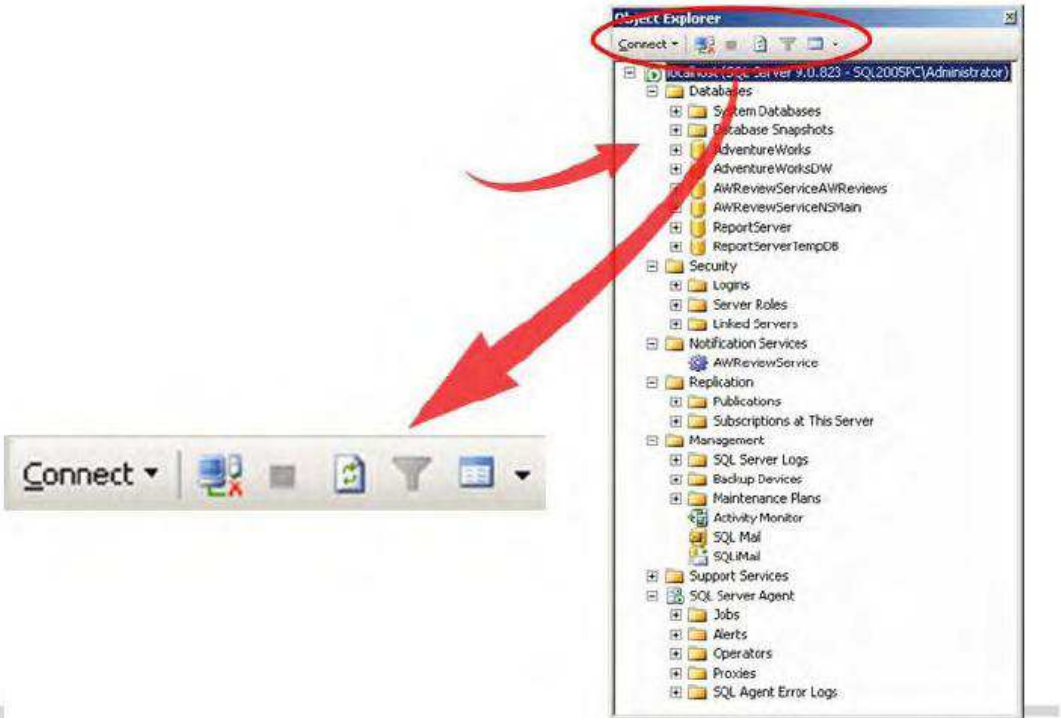
- Herramientas visuales para crear Transact-SQL, XMLA, MDX…

- Incluye funcionalidades de SQL Visual Studio Framework para crear consultas o Scripts.

Ahora bien, ¿qué pasos tenemos que seguir para registrar un servidor en nuestro sistema?

6.9- Registrar Servidor Sql Management.

¿Qué es el Object Explorer?



6.10- Object Explorer Sql Server Management.

El Object Explorer presenta los objetos de una manera jerarquica, en modo de arbol. Nos permite navegar y administrar todos los objetos i características de nuestra base de datos. Gracias al Object Explorer podemos:

- Construir y manejar bases de datos y objetos.

- Ver y modificar propiedades de las bases de datos.

- Generar Scripts para reconstruir los contenidos de las bases de datos.

- Crear fuentes de datos.

- Controlar privilegios y permisos.

- Configurar replicación.

Además, el Object Explorer consta de una serie de botones que pueden dan cierta funcionalidad también a nuestra base de datos:

- Connect: Permite conectarse a una instancia de SQL Server.

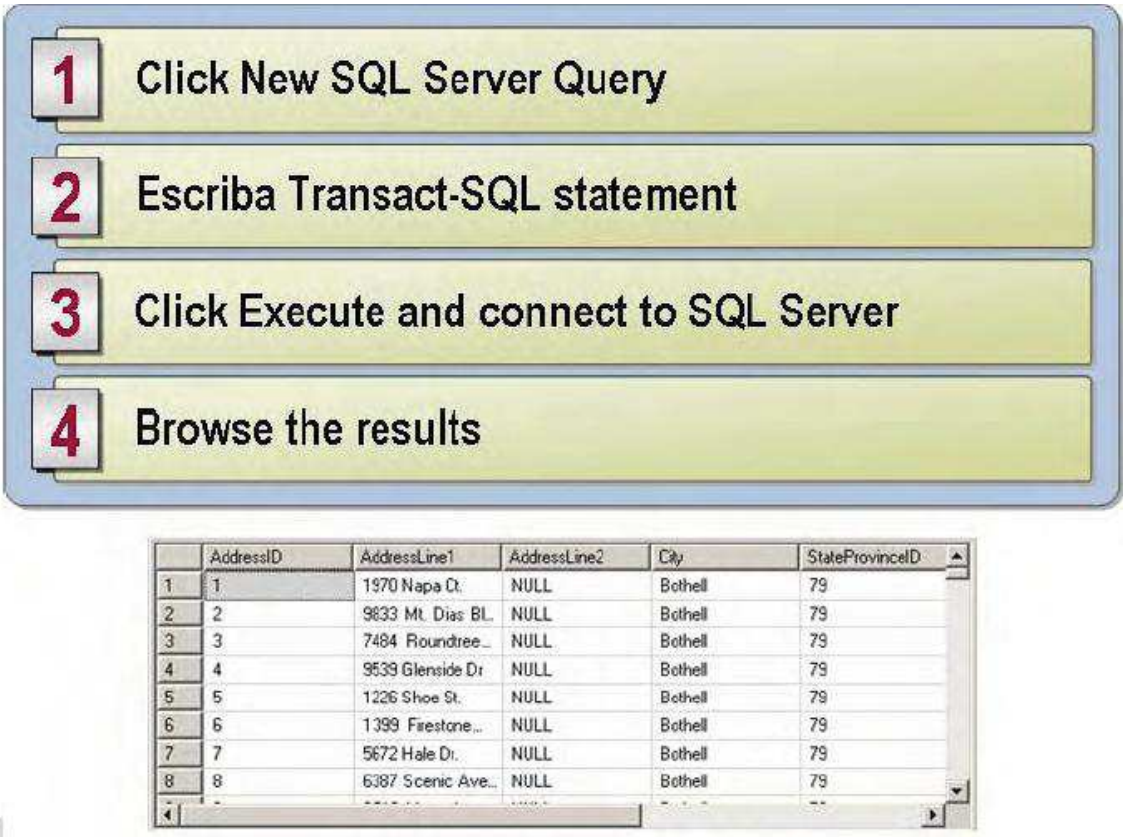
- Disconnect: Como su nombre indica, se desconecta de la instancia a la que nos encontramos conectados.

- Stop: El control de vista de árbol expande carpetas dinamicamente mientras son seleccionadas. Una carpeta que contiene gran número de objetos puede tardar tiempo en ser mostrada. Debemos presionar este botón para detener una operación que está tardando mucho tiempo en ser ejecutada.

- Refresh: Refresca la info del Object Explorer.

- Filter: Filtro para seleccionar lo que deseamos ver.
- Schema: Permite agrupar los objetos según nos convenga.

¿Cómo ejecutamos las consultas SQL?



6.11- Consultas SQL con Sql Server Management.

Hemos explicado brevemente las funcionalidades de SQL Server Management Studio que mas vamos a utilizar. Adjunto a la memoria tenemos un archivo.pdf que explica ampliamente tanto el SQL Server 2005 como el SQL Server Management.

6.4 Características de Java.

Java es un lenguaje de programación orientado a objetos, con una sintaxis similar a C o C++ pero ofreciendo una mayor simplicidad y robustez en el ciclo de desarrollo: las construcciones y características más complicadas de C y C++ han sido eliminadas y el lenguaje contiene mecanismos implícitos para garantizar la seguridad de las aplicaciones construidas con él.
También incorpora dos mecanismos a la hora de escribir programas simples, potentes y robustos: un tratamiento interno de multitarea y un sistema de excepciones que normaliza el procesado de errores por parte del programador.
La principal característica de Java es que es independiente de la plataforma, pudiendo ejecutarlo sobre distintas arquitecturas y sistemas operativos sin que sea necesario modificar el código del programa. Esta independencia se logra gracias a que el lenguaje está soportado por dos elementos fundamentales: el compilador y la máquina virtual. El compilador traduce los programas a un formato especial llamado *bytecodes,* que es el formato que se le pasa a la máquina virtual. Tanto el compilador Java como la máquina virtual son

específicos para cada plataforma, por lo que para poder ejecutar un programa Java en una determinada plataforma debe existir previamente una máquina virtual para ella, por lo que Sun Microsystems dispone de un entorno de ejecución para la mayoría de las plataformas.

Otra ventaja es que cuenta con un gran número de clases preexistentes que no deja de aumentar, presentando una gran riqueza en cuanto al tipo de funciones que permiten realizar. A continuación, se muestran algunos ejemplos:

- *Math:* Proporciona métodos para realizar las operaciones matemáticas más habituales.

- *Date:* Se utiliza para manejar fechas y horas.

- *Integer:* Tiene como miembro una variable de tipo entero, y proporciona los métodos para trabajar con ella.

- *String:* Proporciona a Java la capacidad para el manejo de cadenas de caracteres.

- *Vector:* Representa un array de objetos que puede crecer y reducirse. Además, permite acceder a los elementos con un índice.

- *Frame:* Es una ventana de la que pueden depender otras ventanas y que puede tener una barra de menús.

Sun describe el lenguaje Java como "*simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico*":

Simple: Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Los lenguajes más difundidos eran C y C++, pero adolecen de falta de seguridad, característica muy importante para los programas que se usan en Internet, por ello Java se diseñó para ser parecido a ellos y así facilitar un rápido y fácil aprendizaje. Java elimina muchas de las características de otros lenguajes para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles, como el *garbage collector* (reciclador de memoria dinámica), que se encarga de liberar memoria no usada. Java reduce en un 50% los errores más comunes de programación respecto a lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- No se admite la aritmética de punteros.

- No existen referencias.

- No existe la definición de registros (*struct*).

- No existe la definición de tipos (*typedef*).

- No existe la definición de macros (*#define*).

- No existe la necesidad de liberar memoria (*free*).

Orientado a objetos: Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta encapsulación, herencia y polimorfismo. Hace uso de la definición de entidades formadas por métodos y variables que reciben el nombre de clases, la instancia de una clase recibe el nombre de objeto.

Distribuido: Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales. Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan serlo.

Interpretado: la máquina virtual Java es un programa que se ejecuta sobre el sistema operativo del ordenador (por lo que es dependiente de la plataforma) y ejecuta directamente el código objeto mediante la interpretación de los *bytecodes*, aunque no se trata de un intérprete tradicional pues éstos ya han pasado por las etapas de validación del compilador Java.

Robusto: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los *bytecodes*, que son el resultado de la compilación de un programa Java.

Seguro: La seguridad en Java tiene dos facetas. Por una parte se eliminan características de C y C++ para prevenir el acceso ilegal a la memoria como, por ejemplo, punteros o el *casting* implícito. Por otra parte, el código Java pasa muchos test antes de ejecutarse en la máquina virtual. Pasa a través de un verificador de *bytecodes* que comprueba el formato de los fragmentos de código y aplica un comprobador de teoremas para detectar fragmentos de código que falsee punteros, viole derechos de acceso sobre objetos o intente cambiar el tipo o clase de un objeto. Si los *bytecodes* pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

- El código no produce desbordamiento de operandos en la pila.

- El tipo de los parámetros de todos los códigos de operación son conocidos y correctos.

- No ha ocurrido ninguna conversión ilegal de datos.

- El acceso a los campos de un objeto se sabe que es legal.

- No hay ningún intento de violar las reglas de acceso y seguridad.

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo Caballo de Troya, ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior. Las clases importadas de la red se almacenan en un espacio de nombres privado, asociado con el origen. Cuando una clase del espacio de nombres privado accede a otra clase, primero se busca en las clases predefinidas (del sistema local) y luego en el espacio de nombres de la clase que hace la referencia. Esto imposibilita que una clase suplante a una predefinida. En resumen, las aplicaciones Java resultan extremadamente seguras.

Arquitectura neutral: El compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará (este fichero contiene los *bytecodes*). Cualquier máquina que tenga el sistema de ejecución (*run-time*, que sí es dependiente de la máquina) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.

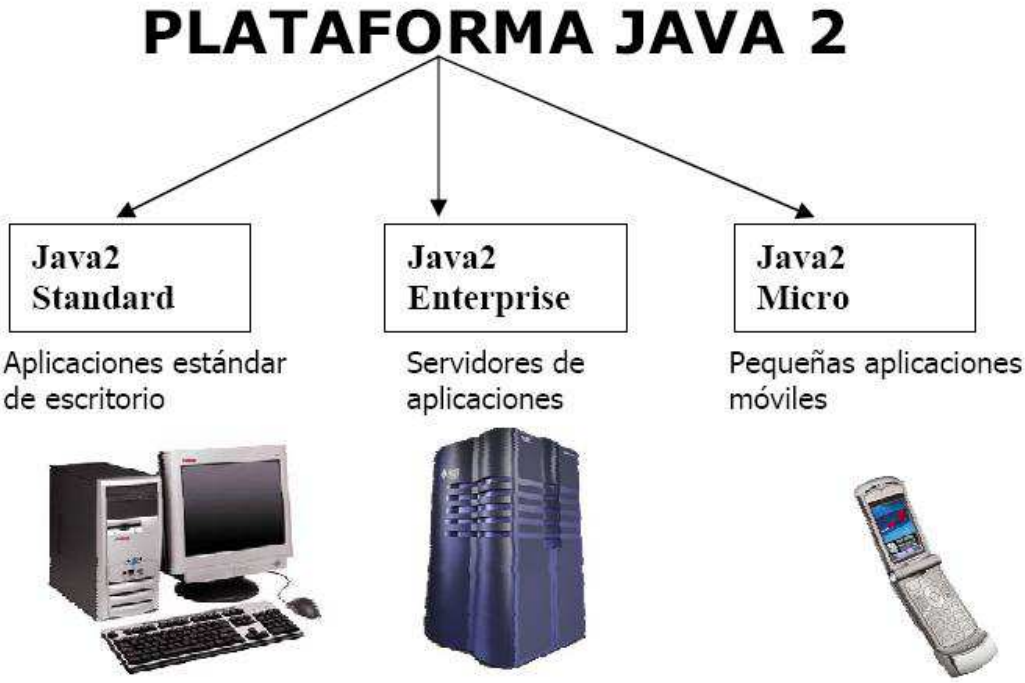
Portable: Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros de 32 bits en complemento a 2 y las cadenas de caracteres utilizan Unicode (no ASCII). Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos diferentes (Unix, Pc, Mac, etc.).

Altas prestaciones: Para los casos en que la velocidad del intérprete Java no resulte suficiente, existen mecanismos como los compiladores **JIT** (*Just In Time*), que se encargan de traducir, a medida que va siendo necesario, los *bytecodes* a instrucciones de código máquina. También existen otros mecanismos como los compiladores incrementales y sistemas dedicados para tiempo real.

Multitarea: Java permite muchas actividades simultáneas en un programa. Los hilos (*threads)* son pequeños procesos o piezas independientes de un gran proceso. Al estar los *threads* construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en otros lenguajes que no los soportan de manera nativa. Esta característica permite mejorar el rendimiento interactivo y el comportamiento en tiempo real.

Dinámico: Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una

aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior). Esto permite actualizar el código “en caliente” y facilita el mantenimiento del software. Por otro lado, Java proporciona mecanismos para cargar dinámicamente clases desde la red, de manera que nuevos contenidos de información podrán ser tratados por manejadores específicos.



6.12- Plataforma Java 2.

6.5 Características J2ME.

- No se pueden crear aplicaciones J2ME sin conocer de antemano la *configuración* y el *perfil* al que van destinados, o sin conocer qué *paquetes opcionales* están disponibles. En estos conceptos se encuentran las definiciones que determinan las características de Java que se pueden emplear, qué APIs usar y cómo se empaquetan, así como sobre cómo se realiza la distribución de las aplicaciones.

- *Configuraciones:*

-Una configuración consiste en un entorno de ejecución Java completo que define el entorno de ejecución básico de J2ME. Su objetivo es adecuarse a las necesidades de una familia de dispositivos con capacidades similares [12]. Cualquier configuración está formada por tres elementos, a saber:

- Una máquina virtual Java para ejecutar el *bytecode* de la aplicación.

- Código nativo para realizar la interfaz entre Java y el sistema operativo del dispositivo.

- Un conjunto de clases Java que constituyen el entorno de ejecución.

Aunque una configuración proporcione un entorno Java completo, su conjunto base de clases es reducido y debe ser ampliado por medio de *perfiles* o bien por clases propias definidas por el fabricante.

Las configuraciones no son pensadas como una solución completa, sino como un punto de partida, una base común para crear APIs adicionales. Actualmente están definidas dos configuraciones: *Connected Device Configuration* (CDC) y *Connected Limited Device Configuration* (CLDC).

CDC

La *Configuración para Dispositivos Conectados* (CDC), definida en la especificación JSR-36, está orientada a dispositivos con cierta capacidad computacional y con memoria, además deben poder conectarse a la red, de modo generalmente inalámbrico. CDC usa una máquina virtual Java similar a la de J2SE pero con limitaciones gráficas y de memoria. Esta máquina virtual es llamada CVM (Compact Virtual Machine). Dispositivos de este tipo son los Palmtop-PC, los Set-Top (televisión digital interactiva), mensáfonos, terminales digitales de video o fotografía, impresoras, reproductores de sonido MP3, etc.

El API CDC es un subconjunto muy amplio de J2SE. Incluye todas las clases del API CLCD, a las que añade algunas características adicionales:

- Soporte para coma flotante.
- Soporte para procesos nativos.
- Soporte para multihilo (*multithread*), incluso con soporte para grupos de hilos.
- Soporte para la manipulación de sistemas de ficheros.
- Soporte para la serialización de objetos.
- Soporte para conexiones HTTP.
- Soporte para la mayoría de las colecciones del API *Collections* de J2SE.
- Cargador de clases definido por el usuario.
- Soporte de red.
- Soporte para los paquetes J2SE.

CLDC

Esta *Configuración para Dispositivos con Conexión Limitada* (CLDC) fue creada para poder mantener las características de Java sobre dispositivos de capacidad muy limitada. Para este dispositivo se dispone de una máquina virtual Java específica denominada *Kilo Virtual Machine* (KVM), donde Kilo corresponde a Kilobyte. CLDC fue definida en la especificación JSR-30 del programa JCP. CLDC no requiere muchos recursos de la máquina:

- Entre 160 Kb y 512 Kb (incluida RAM y memoria flash o ROM)

- Conexión a red, intermitente, de baja velocidad, 9600 bps o menor.

La descripción API CLDC tiene ciertas limitaciones. Las más importantes son:

- No hay soporte para operaciones en coma flotante.

- Las constantes y tipos en coma flotante no están soportadas. Las clases y métodos que necesitaban argumentos *Float* o *Double* han sido eliminadas. No obstante, la revisión 1.1 de la especificación sí que incluye cierto soporte para operaciones en coma flotante.

- No hay finalización de objetos. El método *finalize()* ha sido eliminado de la clase Object. El recolector se limita a recuperar objetos no referenciados y a eliminarlos.

- Los errores en tiempo de ejecución dependerán de la implementación. En CLDC sólo se definen estos errores: Error, OutOfMemoryError, VirtualMachineError y, posteriormente NoClassDefFoundError. Cualquier error no asimilado por la implementación será tratado por la máquina virtual concluyendo la aplicación.

- No hay soporte para *Java Native Interface* (JNI) o cualquier interfaz de bajo nivel que dependa de un mecanismo de reflexión.

- No hay soporte para la serialización de objetos, así como grupos de tareas, hilos de ejecución, ni tareas demonio. No obstante, las tareas por sí solas si están soportadas.

- No se permiten cargadores de clases específicos. La aplicación no puede influir en cómo se cargan las clases.

- La verificación de clases es previa a la ejecución. Se usa la preverificación que es la verificación de las clases en tiempo de desarrollo. De este modo en el dispositivo móvil sólo se validan los resultados del paso previo de preverificación realizado al crear la aplicación.

Requisito importante es la obligada compatibilidad hacia arriba en relación a J2SE. Esto implica que todas las clases estén heredadas de J2SE y sean un subconjunto de ellas. Otro dato a tener en cuenta es que en el paquete *javax.microedition* se encuentran todas las clases definidas para CLDC en exclusiva. Finalmente, la configuración CLDC debe proporcionar soporte para la internacionalización.

A continuación se muestra una breve descripción de los paquetes del API CLDC.

java.lang

Clases No Soportadas:

Float, Double, ClassLoader, Compile, InheritableThreadLocal, Number, Package, Process, RuntimePrecission, SecurityManager, StrictMath, ThreadGroup, ThreadLocal y Void.

Clases Soportadas:

Class, Object, Runtime, System, Thread, Throwable, Runnable, Boolean, Byte, Character, Integer, Long, Short, Float, Double, String, StringBuffer, Math.

java.io

No están soportados los métodos que realizan operaciones en coma flotante. Consta de las siguientes clases:

DataInput, DataInputStream, InputStream, InputStreamReader, Reader, ByteArrayInputStream, DataOutput, DataOutputStream, OutputStream, OutputStreamReader, PrintStream, Writer, ByteArrayOutputStream.

java.util

Las clases más usadas del paquete J2SE, son estas:
Enumeration, Hashtable, Stack, Vector, Calendar, Date, Random.

javax.microedition.io

Además de la clase genérica Connector, se incluye:
ContentConnection, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier, HttpConnection.

javax.microedition.ui

Contiene a las clases Canvas y Screen, extendidas de la clase Displayable, encargada de representar cualquier objeto en pantalla.

javax.microedition.rms

Este paquete contiene las clases necesarias para implementar espacios temporales de almacenamiento en el dispositivo.

javax.microedition.midlet

Contiene la clase MIDlet, que es la que implementa, ejecuta y controla el ciclo de vida del midlet.

- *Perfiles*

“Un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado”.

Mientras que la configuración define las características de las APIs de una familia de dispositivos, los perfiles definen las características de las APIs de cada dispositivo. Por eso, a la hora de construir una aplicación se necesitan tanto las APIs del perfil del dispositivo como de la configuración. Entonces, queda claro que un perfil dado dota de funcionalidad específica a una configuración, apuntamos directamente al objetivo: un dispositivo.

Para cada configuración, CDC – CLDC, tendremos unos perfiles determinados. A continuación, mostraremos combinaciones configuración-perfil y su correspondiente descripción:

CDC - Foundation Profile

APIs orientadas a dispositivos sin interfaz gráfica (descodificadores de televisión digital). Excluye los paquetes *java.awt* y *java.swing*. Incluye *java.lang*, *java.util*, *java.net*, *java.io*, *java.text*, *java.security*.

CDC - Personal Profile

APIs que proporcionan soporte gráfico AWT (*Abstract Windows Toolkit*). Requiere del perfil *Foundation Profile*. Incluye: java.applet, java.awt, java.awt.datatransfer, java.awt.event, java.awt.font, java.awt.im, java.awt.im.spi, java.awt.image, java.beans, javax.microedition.xlet.

CDC - RMI Profile

Construido sobre *Foundation Profile*, RMI soporta un subconjunto de las APIs J2SE v1.3 RMI. Ciertas características de las APIs anteriores han sido eliminadas debido a limitaciones de cómputo y memoria.

CLDC- PDA Profile

Perfil en fase de definición. Pretende abarcar PDAs de gama baja, con pantalla y ratón o lápiz óptico. Resolución de 200x100 píxel al menos, factor 2:1.

CLDC-Mobile Information Device Profile (MIDP)

Este es el perfil que nos interesa en este proyecto. Está orientado a dispositivos móviles como los teléfonos móviles, buscapersonas, PDAs de gama baja con conectividad.

El perfil MIDP construido sobre CLDC fue el primer perfil definido para J2ME.

Orientado a dispositivos de estas características:

- Reducida capacidad computacional y de memoria.
- Conectividad limitada.
- Capacidad gráfica reducida.

- Entrada alfanumérica de datos reducida.
- 128 Kb de memoria no volátil para componentes MIDP.
- 8 Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

El perfil MIDP incluye las siguientes librerías:

- javax.microedition.lcdui: Clases e interfaces para GUIs.
- Javax.microedition.rms: Record Management Storage. Soporte para almacenamiento persistente en el dispositivo.
- javax.microedition.midlet: Clases de definición de la aplicación.
- javax.microedition.io: Clases e interfaces de conexión genérica.
- java.io: Clases e interfaces de E/S básica.
- java.lang: Clases e interfaces de la Máquina Virtual.
- java.util: Clases e interfaces de utilidades estándar

Acabo esta sección haciendo un último apunte: las aplicaciones realizadas usando el perfil MIDP son llamadas MIDlets en clara alusión a los APPlets de Java.

- *Paquetes Opcionales*

Cuando una serie de clases o interfaces no se ajustan a un solo dispositivo, sino que podemos incluirla en varios de ellos, creamos un *paquete opcional*. Un claro ejemplo de esto sería el soporte para la tecnología *Bluetooth.*

Los paquetes opcionales no definen un entorno de ejecución completo sino un conjunto de APIs para características determinadas.
La *Java Comunity Process* (JCP) es la organización encargada de definir los paquetes.

Por último añadir que existen varios paquetes opcionales actualmente en desarrollo como RMI *Optional Package*, los APIs para Bluetooth o el JDBC *Optional*

Package. En nuestro proyecto, el paquete opcional que va a ser utilizado es el RMI Optional Package, que será explicado posteriormente.