UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNIA DE TELECOMUNICACIONES: TELEMÁTICA

PROYECTO FIN DE CARRERA

SERVIDOR BLUETOOTH DE EXÁMENES EN J2ME

Autora: Ana Belén Lobo Romero

Tutor: Mario Muñoz Organero

OCTUBRE 2009

Parece mentira que este momento haya llegado. Después de muchos años por fin he llegado al final del camino. Han sido años de esfuerzo, fuerza de voluntad, disgustos y alegrías, y que han sido más llevaderos gracias a personas de las cuales no me podría olvidar en estos momentos, a las que quisiera dar unas palabras de agradecimiento.

A mis padres, porque gracias a ellos sólo me tenía que preocupar de estudiar, y he tenido la vida resuelta, y porque me han intentado aconsejar aunque no siempre les haya hecho caso.

A mi hermana, estoy orgullosa de ti. Eres un ejemplo a seguir, aunque yo no sé si tengo fuerzas para hacerlo. Gracias por todos los ánimos que me has dado y por ayudarme con papá y mamá en los malos momentos.

A mis abuelos, porque siempre han tenido fe en mí y seguro que están más felices que yo en este momento, ¡os quiero mucho!

A Antonio, por ser mi persona especial. Porque siempre me has dado ánimos, has sabido darme las palabras adecuadas, y me has dado el último empujoncito en la recta final. Porque me haces reír y haces que se me olviden los problemas y sea feliz.

A Javi, por sus buenos consejos y por pelearse con el Word para que este proyecto haya quedado tan bien.

A Vane, Esther, Pablo, Rocío y Miriam con los que tan buenos momentos he pasado. Siempre recordaré nuestros días de nueve a nueve en la universidad, la mayoría de ellos peleándonos con las prácticas pero también había tiempo de risas y cervezas en el césped del campus.

A Alicia, Ruth y Elia por estar siempre ahí.

Y finalmente, a Mario Muñoz Organero por ayudarme y guiarme en este proyecto fin de carrera y por tener siempre esa sonrisa en la cara.

A todos muchas gracias y un beso.

Ana.

ÍNDICE

ÍNDICE DE FIGURAS

A continuación se expondrá con detalle el proyecto fin de carrera "Servidor Bluetooth de exámenes con J2ME".

La aplicación que desarrolla sigue el modelo cliente-servidor por lo que cuenta con dos módulos principales. El primero de ellos se corresponde con el servidor, en este caso el móvil del profesor, que dispondrá de una pila de exámenes que podrán ser publicados vía Bluetooth cuando el profesor lo considere oportuno. El segundo de los módulos será el que se instalen los alumnos en su propio teléfono móvil. Con él podrán descargar el examen publicado, resolverlo y enviar los resultados al profesor. Una vez enviados los resultados, el profesor podrá verlos en su terminal.

La tecnología utilizada para la implementación de este proyecto es J2ME, ya que ofrece grandes facilidades para el desarrollo de proyectos para móviles, ofrece la posibilidad de emular la aplicación en el ordenador durante la fase de desarrollo y es posible la instalación en distintos terminales.

Con la utilización de esta aplicación en las aulas, se conseguirá un mayor dinamismo en las clases, aumentando así el interés de los alumnos y con ello una mejora en los resultados académicos de éstos.

Octubre de 2009

With the goal to finalize the Telecommunication Degree it has studied and deployed as a final project degree the Bluetooth Server of Exams with J2ME.

The application is developed according with the customer-server's model because of it has to two main different models. The first module is corresponded with the server, in this case, the server is the telephone mobile of the teacher. This sever has a stack of exams which it can be published by Bluetooth according to the teacher's decision. The second module, has to be installed, by the student, into his/her telephone mobile. With this mechanism, the students can unload the published exams in order to resolve and come back to the teacher, by the same way. When the students finishes to come back the results, the teacher can evaluate the answer into his/ her terminal.

The technology use to implement the project is J2ME, due to is better and easier to deploy in the mobile telephone's field. Besides, it pick up the possibility to emulate the application into the computer in the deployment phase and it is possible the installation into several terminals.

The used of this application in classroom provides a higher dynamism, increases students´interest, in order to improve the academic results.

1. INTRODUCCIÓN

El presente proyecto fin de carrera tiene por objetivo el poder aplicar las nuevas tecnologías existentes a la educación.

Estamos viviendo una época de revolución tecnológica. Los móviles de nueva generación ofrecen miles de funcionalidades y permiten tener en un mismo dispositivo de tamaño reducido, conexión a internet, reproductor de música, cámara de fotos… Es un dispositivo de comunicación, ocio, trabajo y, ¿por qué no?, educación.

Cada vez más se lee en los periódicos el poco entusiasmo que tienen los jóvenes en continuar con su formación académica, y cómo el número de universitarios está disminuyendo año tras año. Según los últimos datos publicados por el Instituto Nacional de Estadística (INE) relativos al curso 2006/2007, el número de alumnos matriculados en estudios universitarios de primer y segundo ciclo disminuye un 1,6% con respecto al curso anterior, y mientras que la matrícula en universidades públicas desciende un 2,1% respecto al curso precedente, el alumnado se incrementa un 3,0% en las privadas, y por primera vez en los últimos 10 cursos se produce una disminución en la matriculación en tercer ciclo. En el curso 2006/2007 se matricula un 5,6% menos que en el curso anterior.

Con esta situación, hay que encontrar formas para incentivar a los alumnos a participar en clase y esforzarse para estudiar día a día. Este proyecto tiene esto como propósito.

Gracias a una aplicación que cada alumno se instalará en su teléfono móvil, el profesor puede, después de una clase, publicar un cuestionario a través de Bluetooth para que los alumnos se lo descarguen, contesten a una serie de preguntas, y enviarle al profesor los resultados, para así poder optar, por ejemplo, a un punto extra en la calificación final de la asignatura. De esta forma se conseguiría que los alumnos participaran más en clase, pusieran más atención y, en definitiva, que les costase menos asimilar los nuevos conceptos que se dieran.

Pero esta aplicación no sólo se ciñe al ámbito universitario. También se puede aplicar en cualquier campo de enseñanza: autoescuelas, centros de idiomas…

1.1. OBJETIVOS DEL PROYECTO FIN DE CARRERA

Hoy en día, el uso del teléfono móvil entre los estudiantes está muy extendido. Prácticamente cada estudiante tiene un teléfono móvil. Esto puede ser una ventaja a la hora de explotar las miles de posibilidades que ofrecen estos pequeños dispositivos.

El objetivo de este proyecto es precisamente eso, beneficiarse de que los alumnos tienen teléfonos móviles para implementar una aplicación que fomente la participación en clase, que los alumnos tomen más interés, evitar el que los estudiantes falten a las clases… en definitiva que el aprendizaje de los nuevos conceptos se les haga más fácil y ameno.

Teniendo claro estos objetivos, se trataba de implementar una aplicación que fuera de fácil acceso a los alumnos, y que le permitiera al profesor tener unos datos de la asimilación de los conceptos explicados durante la clase.

La aplicación debería ser de fácil manejo y que ofreciera rapidez y dinamismo, pues la idea es que, una vez dada la clase teórica, se dediquen los últimos 10-15 minutos a que los alumnos realicen un pequeño test sobre los conceptos explicados en clase, y darles la opción de que el que menos fallos haya cometido, obtenga un punto extra en la calificación final de la asignatura.

El funcionamiento sería el siguiente:

1. El profesor explica la clase.
2. El profesor conecta su aplicación Bluetooth y publica el test correspondiente a la lección explicada.
3. Los alumnos conectan su aplicación Bluetooth y se descargan el test.
4. Los alumnos realizan el test.
5. Los alumnos dan a corregir el test en su propio teléfono móvil.
6. Los alumnos, una vez visto el resultado lo envían a la aplicación del profesor.
7. Una vez finalizado el tiempo de respuesta, el profesor podrá ver para el examen correspondiente los resultados de los alumnos (número de aciertos de cada alumno) y podría dar el punto extra al que haya tenido más aciertos.

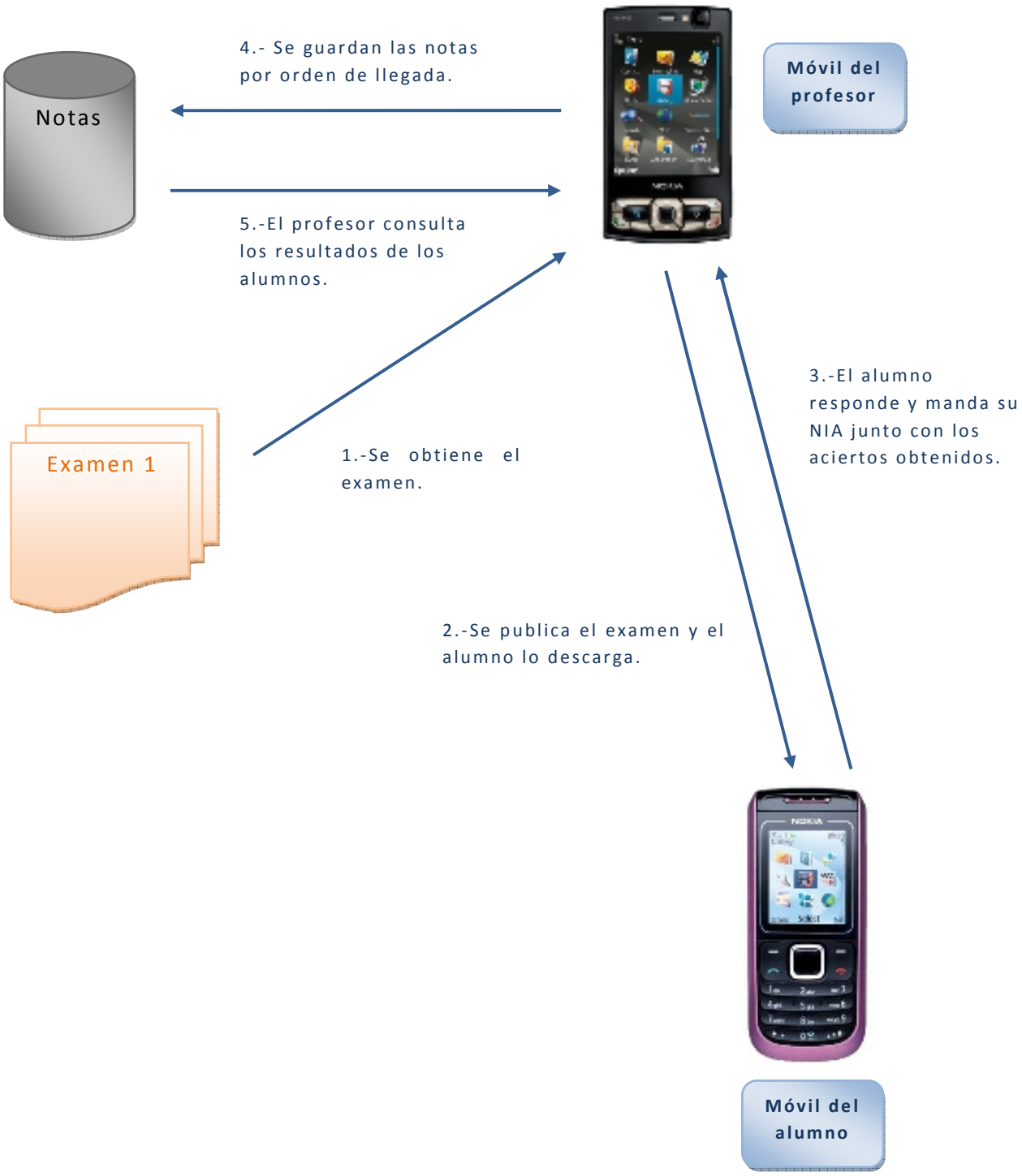


Ilustración 1. Funcionamiento de la aplicación.

2. ESTADO DEL ARTE

En la actualidad es difícil imaginarse la vida sin tener un teléfono móvil en el bolsillo. Cada día salen nuevos modelos de teléfono, más pequeños, con mejores diseños y con más funcionalidades. Aunque la función básica y más importante es la de poder realizar llamadas telefónicas a cualquier terminal, fijo o móvil, poco a poco aparecen nuevas formas de comunicación y nuevas aplicaciones.

La primera de ellas, y casi la más importante, debido al gran beneficio económico aportado a las compañías telefónicas es la mensajería SMS (*Short Message Service*). Los terminales nos permiten enviar mensajes cortos de texto (hasta un tamaño de 160 caracteres) que son enviados desde el terminal al centro servidor de mensajes cortos o SMSC (*Short Message Service Centre*), que a su vez se encarga de hacer llegar el mensaje al móvil destinatario. Esta aplicación está muy extendida entre los jóvenes lo que supone un gran volumen de facturación a las compañías telefónicas.

Lo siguiente en aparecer fue la posibilidad de navegar por Internet con estos pequeños dispositivos, mediante la tecnología WAP (*Wireless Application Protocol – protocolo de aplicaciones inalámbricas*). El inconveniente que tenía esta tecnología es que no se podían visitar todas las páginas web que nosotros quisiéramos, sino sólo las que estaban desarrolladas en WML (*Wireless Markup Language)*, esto junto con las elevadas tarifas de conexión y la baja velocidad ofrecida, hicieron que el uso de WAP no se extendiera entre los usuarios.

Pero esto era un mercado que se debía explotar, y para ello habría que ofrecer mejoras a los usuarios para que así se animaran a conectarse a Internet a través de sus móviles. Una de estas mejoras fue la introducción de GPRS (*General Packet Radio Service*) sobre la tecnología GSM (*Sistema Global para las Comunicaciones Móviles*), ya que GSM realiza la transmisión sobre conmutación de circuitos, y esto no es eficiente para la transmisión de datos. GPRS permite a las redes celulares una mayor velocidad y ancho de banda sobre el GSM, mejorando las capacidades de acceso móvil a la Internet. Con ello podemos enviar paquetes de mayor tamaño, como por ejemplo fotos, música, vídeo…

Con esto ya tenemos un teléfono móvil que podemos usar en cualquier país de Europa (ya que podemos hacer Roaming internacional) y que es además un pequeño centro multimedia, ya que poco a poco los móviles van evolucionando hasta tener cámara de fotos que graba vídeo, reproductor de música y vídeo…

A la vez que aparecen estos móviles en el mercado, aparece el nuevo servicio de mensajes cortos llamado MMS (*Multimedia Message Service*).

Gracias a MMS, además de texto, podemos enviar fotografías, sonidos, gráficos, etc.

Y estos nuevos teléfonos ofrecen muchas posibilidades. Por ejemplo, al tener mejores pantallas, los gráficos son de mejor calidad por lo que se puede dar un paso más allá e intentar introducir las funcionalidades de las videoconsolas a estos móviles, o introducir otras aplicaciones útiles para los usuarios.

Con esta situación aparece la tecnología J2ME (*Java 2 Micro Edition*) de Sun Mycrosystems. La plataforma J2ME trae el poder y las ventajas de tecnología Java a los dispositivos integrados. El objetivo principal de J2ME es permitir a los dispositivos descargar dinámicamente aplicaciones que mejoren sus capacidades iniciales. En el caso de los teléfonos móviles, por ejemplo, J2ME nos permite descargar una gran variedad de juegos.

Paralelamente se desarrolla la tecnología Bluetooth. Bluetooth es una tecnología inalámbrica de corto alcance y que originalmente se desarrolló para reemplazar los cables a la hora de conectar dispositivos. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión lo permite.

Hoy en día prácticamente todos los teléfonos móviles y ordenadores personales tienen integrada la capacidad Bluetooth y si no es así, existen transmisores/receptores de bajo coste y pequeño tamaño que se pueden conectar al ordenador mediante un puerto USB (Universal Serial Bus).

Mediante Bluetooth podemos por ejemplo conectar nuestro móvil al ordenador de una forma sencilla, y transmitir archivos entre ellos rápidamente. Otra de las aplicaciones de Bluetooth es, por ejemplo, conectar nuestro teléfono móvil a la radio del coche para así poder reproducir la música que llevamos en el teléfono en ella.

Como se puede ver, tanto J2ME como Bluetooth son dos tecnologías con mucho potencial y futuro.

2.1. J2ME

J2ME es una tecnología derivada del estándar Java que permite construir programas muy compactos, aptos para ser descargados y ejecutados sobre dispositivos con pocos recursos (memoria, procesador, interfaces de entrada y salida) informáticos.

Inicialmente J2ME se ha centrado en dispositivos móviles conectados, generalmente teléfonos móviles. Esto le ha dado una gran transcendencia debido a la enorme difusión de estos teléfonos.

CONFIGURACIONES Y PERFILES

J2ME se basa en los conceptos de configuración y perfil.

Una **configuración** describe las características mínimas en cuanto a la configuración hardware y software. No es más que una concreción del API, generalmente restrictiva, que redefine buena parte de las clases (por ejemplo, las gráficas) y añade algunas más. Una configuración determinada hace referencia a un elevado número de dispositivos, pero los describe a partir de sus características comunes.

El siguiente nivel es el del **perfil**. Una vez que se ha definido el conjunto de dispositivos, hay que entrar a concretar las características de uno de ellos o de una familia (serie de un mismo fabricante, por ejemplo), por ejemplo variaciones en el tamaño de la pantalla, número exacto de botones, o disponibilidad, por ejemplo, de funciones especiales, como la vibración del dispositivo.

MIDP (*Mobile Information Device Profile*) es el perfil que se aplica sobre la mayor parte de los dispositivos de esta clase.

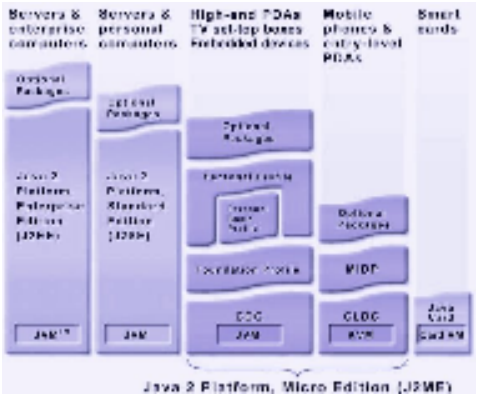


Ilustración 2. Plataforma Java.

CONFIGURACIÓN CLDC.

La configuración CLDC (*Connected Limited Device Configuration*, Configuración de Dispositivo Conectado Limitado) es la única definida a día de hoy por *Sun* para J2ME y contiene las clases e interfaces Java necesarios para desarrollar aplicaciones sobre dispositivos móviles con conexión inalámbrica (*wireless*), básicamente teléfonos móviles. Lo que contiene la configuración CLDC son:

- Un subconjunto de los elementos básicos del lenguaje Java.
- Una parte de la funcionalidad de la máquina virtual Java.
- Las APIs básicas para el desarrollo de aplicaciones.
- Requisitos hardware de los dispositivos englobados en el CLDC.

Los requisitos mínimos hardware que definen a un dispositivo englobado en la configuración CLDC son:

- 160Kb de memoria disponible para Java. Esta memoria está dividida en dos áreas: 128Kb de memoria no volátil para la máquina virtual Java y las librerías del API CLDC; y 32Kb de memoria volátil para el entorno de ejecución.
- Un procesador de 16 bits.
- Bajo consumo, por regla general por medio del uso de baterías.
- Conexión a red, normalmente con un ancho de banda de 9.600 bps o inferior.

PERFIL MIDP.

En la jerarquía definida en J2ME, los perfiles se apoyan sobre las configuraciones, CLDC en este caso.

Un perfil es una concreción aún más restrictiva de las APIs para centrarse en un conjunto muy reducido de dispositivos. Es decir, el perfil añade unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como con las de la configuración.

Las características mínimas que deben de tener los dispositivos del perfil MIDP son:

- Memoria:
  - Al menos 128Kbytes de memoria no volátil para almacenar el API MIDP.
  - Al menos 32Kbytes de memoria volátil para el sistema de ejecución Java.
  - Un mínimo de 8Kbytes de memoria persistente para el almacenamiento de información por parte de los programas.
- Pantalla:
  - Tamaño mínimo: 96x54 pixels.
  - Profundidad: 1 bit.
  - Aspecto pixel 1:1, es decir que la pantalla debe parecer cuadrada, lo que supone usar unos pixels alargados.
- Entrada de datos:
  - Debe tener un teclado o una pantalla táctil, o ambos.
  - Aunque un ratón no forma parte de los medios de entrada reconocidos, un puntero (como en las PDAs) sí es un sistema de entrada válido ya que se aplica sobre una pantalla táctil.
- Conectividad:
  - Debe contar con acceso a una red bidireccional inalámbrica.
  - Las exigencias mínimas de ancho de banda son 9.600 bps.
- Plataforma software:
  - Kernel (núcleo del sistema operativo) capaz de hacerse cargo de tareas de bajo nivel como la gestión de interrupciones, excepciones y temporizadores.
  - Un mecanismo para leer y escribir en memoria no volátil.
  - Gestión del tiempo para fijar temporizadores y añadir marcas temporales a la información persistente.
  - Acceso de lectura/escritura a la conexión inalámbrica del dispositivo.

- Medios para obtener la información introducida por el usuario a través de los dispositivos de entrada.
- Soporte a gráficos basados en mapas de bits.
- Medios para gestionar el ciclo de vida de una aplicación (inicio, interrupción, reactivación y destrucción).

MIDlet

Un MIDlet es una aplicación J2ME desarrollada sobre el perfil MIDP.

Para que un MIDlet siga la filosofía Java de "*write one, run anywhere*", la especificación MIDP ha definido los siguientes requisitos:

- Todos los dispositivos de información móviles deben contar con un módulo software encargado de la gestión de los MIDlet (cargarlos, ejecutarlos, borrarlos…). Este módulo se denomina gestor de aplicaciones (JAM, *Java Application Management*).
- Todos los MIDlet deben ofrecer la misma interfaz a los gestores de aplicaciones. Así, independientemente de la funcionalidad interna que implementen, los dispositivos pueden identificar a los MIDlet y realizar acciones sobre ellos. Este comportamiento se consigue mediante el mecanismo de herencia: todos los MIDlet heredan de la misma clase, `javax.microedition.midlet.MIDlet`.

CICLO DE VIDA DE UN MIDLET.

Los MIDlet pasan por distintos estados a lo largo de su ejecución, desde su inicio hasta su finalización. El ciclo de vida de un MIDlet describe estos estados, así como los eventos que disparan las transiciones entre ellos.

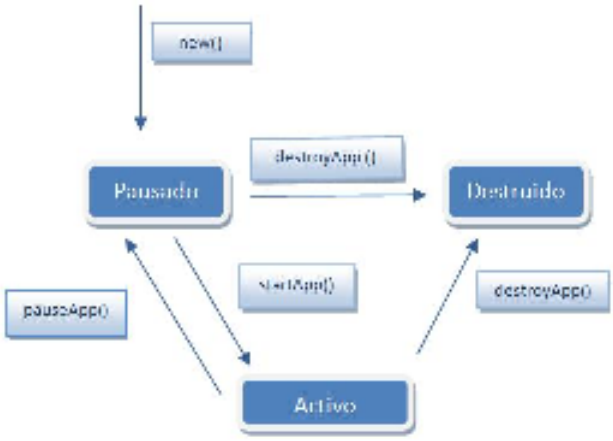


Ilustración 3. Ciclo de vida de un MIDlet

- **Pausado:** un MIDlet se encuentra en este estado cuando es interrumpido por el gestor de aplicaciones y también durante un breve periodo de tiempo al inicio de la aplicación, entre la llamada al método de creación del MIDlet y el inicio de la ejecución.
- **Activo:** un MIDlet se encuentra en este estado durante su ejecución normal. Se puede llegar aquí tanto al inicio de la ejecución como después de una interrupción (en este caso vendría del estado Pausado).
- **Destruido:** cuando un MIDlet termina su ejecución pasa a estado Destruido.

El método **`startApp()`** se ejecuta inmediatamente antes de que el MIDlet pase al estado Activo. Por esta razón es el sitio ideal para que incluyamos código que prepare al MIDlet para empezar o continuar la ejecución. Este método no es llamado únicamente al iniciarse la aplicación, cualquier interrupción que se produzca durante la ejecución de un MIDlet y que le lleve al estado Pausado desencadenará una nueva llamada a `startApp()`.

El método **`pauseApp()`** se ejecuta justo antes de que el MIDlet pase al estado Pausado. Este método se puede aprovechar para guardar información que pudiera perderse durante la interrupción del MIDlet. Por ejemplo los estados de un juego (posiciones de objetos, puntuaciones…) o progresos de acciones en segundo plano, como descargas de red. En aplicaciones donde no es necesario guardar ningún estado podemos dejar vacía la implementación de este método.

Si el cambio de estado es iniciado por el MIDlet mediante el uso del método `notifyPaused()`, es recomendable llamar explícitamente a

`pauseApp()`, como medio de asegurar que los recursos se liberan adecuadamente.

El método **`destroyApp()`** se ejecuta antes de que el MIDlet pase al estado Destruido. Aquí se debe incluir la liberación de recursos temporales reservados por el MIDlet y el almacenamiento de la información que debe perdurar hasta ejecuciones posteriores.

Como parámetro de entrada, **`destroyApp()`** recibe un boolean, que indica si el MIDlet debe terminar en cualquier caso (valor true) o si es posible evitar este cambio de estado mediante la generación de la excepción `MIDletStateException`. Este mecanismo permite al gestor dar marcha atrás en el cambio de estado en caso de que se produzca alguna situación inesperada dentro del método (guardando información o cerrando alguna conexión de red).

ESTRUCTURA GENERAL DE LA API DE INTERFAZ DE USUARIO

Dentro de MIDP tenemos dos alternativas a la hora de realizar el interfaz gráfico de nuestros MIDlets, usar componentes del API de alto nivel para interfaces de usuario o el API de interfaces de usuario de bajo nivel.

Con el API de interfaz de usuario de alto nivel es el dispositivo (o más bien la máquina virtual del dispositivo) quien se encarga de colocar los componentes, las barras de desplazamiento, la navegación y características visuales como el color, las formas, los fuentes y los elementos a visualizar. Además tiene un sistema de entrada de alto nivel asociado.

Con el API de interfaz de usuario de bajo nivel la aplicación tiene un control mayor sobre la pantalla. El motivo de este API es para aquel tipo de aplicaciones que necesitan un control y precisión absoluto a la hora de dibujar elementos en pantalla, como son los juegos (que son el tipo de aplicaciones que copan la mayor parte del mercado del software para móviles). Un mayor control, significa también una mayor responsabilidad: hay que dibujar todo lo que aparece en pantalla e interpretar cualquier entrada del usuario. Evidentemente, este API también tiene asociado un sistema de entrada de bajo nivel. También hay que tener en cuenta que cualquier aplicación que use este API, puede no ser portable o muy poco portable, debido a que tiene que comprobar los recursos del dispositivo, por ejemplo, tenemos que saber el tamaño de la pantalla para no dibujar píxels más allá de los límites.

Las principales clases para crear interfaces de usuario en MIDP se agrupan dentro del paquete `javax.microedition.lcdui`.

Clase/ Interfaz	Descripción
Display	Gestor de recursos gráficos
Displayable	Pantalla gráfica de la interfaz de usuario
Screen	Pantalla genérica del API de alto nivel
TextBox	Pantalla de introducción de texto
List	Pantalla de selección de opciones
Alert	Pantalla de aviso
Form	Pantalla de formulario
Item	Elemento genérico de un formulario
ChoiceGroup	Elemento de formulario para seleccionar opciones
DateField	Elemento de formulario para introducir fechas
TextField	Elemento de formulario para introducir texto
Gauge	Elemento de formulario con forma de diagrama de barras, para introducir valores enteros pequeños
ImageItem	Elemento de formulario que representa una imagen descriptiva
StringItem	Elemento de formulario que representa un texto descriptivo
Canvas	Pantalla genérica del API de bajo nivel
Command	Acción genérica realizada sobre la interfaz por el usuario
Ticker	Texto deslizante disponible en todas las pantallas del API de alto nivel
Graphics	Conjunto de herramientas para dibujar dentro de una pantalla tipo Canvas
Choice	Interfaz genérico que implementan las clases que permiten seleccionar opciones

Tabla 1. Clases para crear interfaces de usuario en MIDP

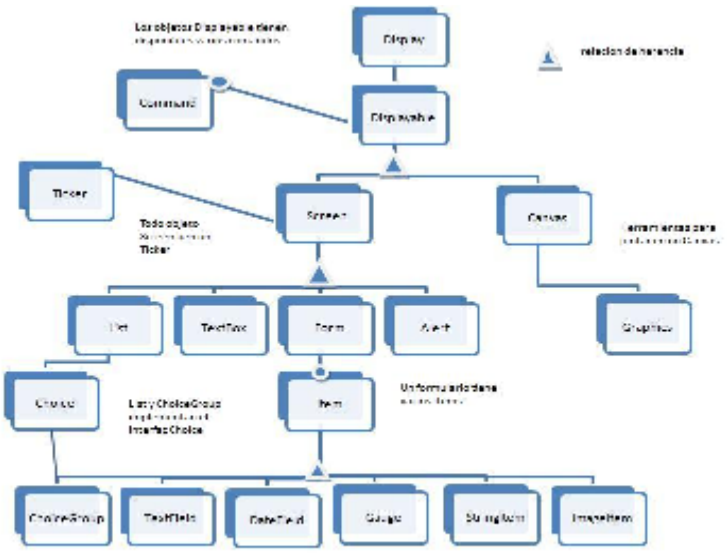


Ilustración 4. Jerarquía de clases de interfaz de usuario en MIDP.

LA CLASE DISPLAY Y LA CLASE DISPLAYABLE

La clase **Display** representa un gestor de la ventana y es la clase responsable de controlar la pantalla y la interacción con el usuario. Por cada MIDlet hay exactamente una única instancia de la clase `Display`.

Para poder obtener la instancia del objeto `Display` correspondiente a nuestro MIDlet, hay que utilizar el método de clase `Display.getDisplay()` usando como parámetro de entrada el propio MIDlet. El método `getDisplay()` se suele ejecutar una sóla vez en el constructor y almacenar la referencia al `Display`, en lugar de ejecutar continuamente el método `getDisplay()`.

Una vez disponible, el objeto `Display` se usa para obtener información de las capacidades gráficas del dispositivo (si tiene color o no, cuántos colores puede mostrar…). Pero su principal función es controlar qué pantalla se muestra en cada momento en la ventada. Para ello proporcionados métodos, `getDisplay()` y `setDisplay()`, que permiten respectivamente seleccionar y recuperar la pantalla actual.

La clase abstracta **Displayable** representa una pantalla genérica, de la que derivan todas las demás pantallas de un MIDlet (como se puede ver en la Figura 3). De esta clase proceden todos los métodos que tienen todas las pantallas para añadir y eliminar comandos y para establecer un manejador de eventos que controla la ejecución de dichos comandos. Una clase `Displayable` no será visible hasta que no se use el método `setCurrent()` de la clase `Display`.

Si se quiere obtener la pantalla que había antes se obtiene con el método `getCurrent()` de la clase `Display`.

La clase `Display` no se corresponde exactamente con la pantalla del dispositivo, ya que cuando se está en estado Pausado aunque se intente escribir en la pantalla no tendremos acceso a ella, y el resultado no será visible hasta que el MIDlet esté otra vez activo.

Podemos saber si un objeto `Displayable` está visible usando el método `isShown()` de la clase abstracta `Displayable`. Que devuelve true sólo cuando el objeto `Displayable` se ha mostrado en la pantalla del dispositivo.

GESTIÓN DE EVENTOS

Los mecanismos que proporciona la API de alto nivel para gestionar los eventos son la clase **`Command`** y la interfaz **`CommandListener`**, que permiten crear manejadores de eventos asociados a la ejecución de comandos.

La clase `Command` permite definir acciones genéricas que el usuario puede realizar durante la ejecución de un MIDlet, sería lo que el usuario llamaría *botón* o *tecla*. Un `Command` está compuesto de una etiqueta, una prioridad y un tipo.

- La etiqueta es un `String` que representa el significado del comando, es lo que ve el usuario en la pantalla del dispositivo.
- El tipo es un entero que especifica la acción del comando. Existen los tipos definidos: `BACK`, `CANCEL`, `HELP`, `EXIT`, `ITEM`, `OK`, `SCREEN` y `STOP`.
- La prioridad es un entero que indica la importancia del comando, que será mayor cuanto menor sea el número.

Una vez que el usuario aprieta un determinado *botón* (`Command`) se efectuarán las acciones que haya en el método `commandAction` perteneciente al interfaz `CommandListener`. El método `commandAction` está definido de la siguiente forma:

```
public void commandAction(Command c, Displayable s)
```

El primer argumento es el objeto `Command` actual, que ha sido presionado por el usuario y el segundo es el objeto `Displayable`.

ALMACENAMIENTO PERSISTENTE - RMS

Muchos MIDlets necesitan guardar datos de la aplicación, por ejemplo en un juego se podrían guardar los puntos obtenidos por el usuario. Para estos casos en los que es necesario guardar información y que ésta siga ahí la siguiente vez que se ejecute la aplicación, está el paquete `javax.microedition.rms`, que permite almacenar datos de forma persistente.

Los datos se guardan en un `RecordStore`. Un `RecordStore` es una pequeña base de datos que contiene datos llamados `record`. Un `record` es un array de bytes de tamaño variable.

El ámbito de un `RecordStore` está limitado a un único MidletSuite. Dicho de otra forma, un MIDlet sólo puede accerder a los RecordStores creados por un MIDlet de un mismo MidletSuite.

El nombre de un `RecordStore` debe ser único dentro del MidletSuite, y es sensible a las mayúsculas.

Para la trabajar con los `RecordStore` los métodos más destacables son los siguientes:

- `Record openRecordStore(String recordStore, boolean createIfNecessary)`: este método recibe como parámetro un String que será el nombre del `RecordStore` y un `Boolean` indicando si hay que crear un `RecordStore` con este nombre si no existía previamente, si este valor estuviera a false y el `RecordStore` no existiera, saltaría una excepción `RecordStoreNotFoundException`. Este método devuelve el objeto `RecordStore` abierto o recién creado.
- `closeRecordStore()`: este método cierra un `RecordStore` que previamente ha sido abierto. Siempre después de utilizarse un `RecordStore` debe cerrarse.
- `deleteRecordStore(String recordStoreName)`: este método borra un `RecordStore` existente y todos los `records` que contenía. Previamente se tiene que haber cerrado.
- `String[] listRecordStores()`: este método devuelve un array con los nombres de los `RecordStores` disponibles para este MidletSuite.

A la hora de trabajar con los `records` es necesario conocer los siguientes métodos:

- `int addRecord(byte[] data, int offset, int numBytes)`: este método añade un record al `RecordStore`. Recibe como parámetros un array de bytes que serán los datos que se quieren guardar, un entero (offset) que será la posición dentro de ese array desde la que se quiere empezar a guardar, y un entero con el número de bytes que se quieren guardar.
- `byte[] getRecord(int recordId)`: devuelve en un array de bytes el record con el identificador que se pasa por parámetro.
- `setRecord(int recordId, byte[] newData, int offset, int numBytes)`: este método asigna nuevos valores a un `record` determinado.
- `deleteRecord(int recordId)`: este método borra un `record` determinado.
- `int getNextRecordID()`: devuelve el identificador del siguiente record que será añadido.
- `int getNumRecords()`: devuelve el número de `records` que tiene el `RecordStore`.

El paquete `javax.microedition.rms` contiene una serie de interfaces que son de gran utilidad para las aplicaciones que contienen `RecordStores`, y que son los siguientes:

- `RecordEnumeration`.
- `RecordFilter`.
- `Recordcomparator`.

Cuando se ejecutan los MIDlets es muy fácil que el `RecordStore` cambie con rapidez. Por ello, es muy posible que los identificadores de los `records` dejen de ser consecutivos. Para poder trabajar de una forma cómoda con los `records`, se proporciona el interfaz `RecordEnumeration` y el método:

```
RecordEnumeration enumerateRecords(RecordFilter filter,
RecordComparator comparator, boolean keepUpdated)
```

Un objeto `RecordEnumeration` es una lista doblemente enlazada en la que cada nodo representa un `record`. Este interfaz contiene los métodos:

- `reset()`: puntero al primer elemento de la lista.
- `int nextRecordId()`: devuelve el identificador del siguiente elemento de la lista.
- `int previousRecordId()`: devuelve el identificador del elemento anterior de la lista.

Como se puede ver el método `enumerateRecords` del interfaz `RecordEnumeration`, recibe como parámetro un objeto tipo `RecordFilter`, este es otro de los interfaces disponibles para el almacenamiento persistente. `RecordFilter` es un interfaz que define un filtro que examina un `record` para ver si verifica el criterio de selección. Implementa el método `match()` para seleccionar los `records` que serán devueltos por el `RecordEnumeration`. Devuelve verdadero si el registro candidato es seleccionado por el `RecordFilter`.

Otro de los parámetros que recibe `enumerateRecords` es un objeto tipo `RecordComparator`. Este interfaz se usa para clasificar los `records` por algún criterio. Si se quiere implementar el interfaz `Recordcomparator` en un objeto, se tiene que escribir el método `int compare(byte[] rec1,byte[] rec2)` que compara dos `records`, el valor devuelto debe indicar el orden de los dos registros. Hay tres tipos definidos como valor de retorno:

- `PRECEDES`: el primer `record` precede al segundo en orden.
- `FOLLOWS`: el primer `record` sigue al segundo en orden.
- `EQUIVALENT`: los dos `records` son equivalentes.

Octubre de 2009

2.2. BLUETOOTH

Bluetooth es una tecnología inalámbrica de especificación abierta, de bajo coste, bajo consumo, de corto alcance. Enfocada a la comunicación tanto de datos como de voz.

Bluetooth está causando una auténtica revolución. Los fabricantes de dispositivos electrónicos lo incluyen en sus productos sin que ello eleve mucho el coste inicial del producto (el coste de los chips Bluetooth es inferior a 3$), y por ello prácticamente todos los ordenadores, tanto de sobremesa como los portátiles, disponen de Bluetooth.

Pero Bluetooth no se ciñe únicamente al campo de los ordenadores, lo podemos encontrar en los teléfonos móviles, en radios, en los dispositivos de manos libres para el coche…

Algunas de sus características más importantes:

- Está orientado a aplicaciones de voz y datos.
- El que sea una especificación hace que esté públicamente disponible y que tenga derechos libres.
- Es de corto alcance debido a que la distancia máxima de comunicación es aproximadamente 10 metros. Con una potencia de transmisión mayor se pueden llegar a alcanzar los 100 metros.
- Debido a que la comunicación es de corto alcance, el consumo también es bajo, lo que permite que lo puedan tener dispositivos portátiles que se alimenten de una batería.
- Bluetooth funciona en cualquier parte del mundo, ya que trabaja a una frecuencia de 2.4 GHz, que no precisa ninguna licencia.
- Su máxima velocidad de transmisión es de 3 Mbps.

Ilustración 5. Logotipo Bluetooth

PILA DE PROTOCOLOS

La especificación Bluetooth permite poder conectar dispositivos de diferentes fabricantes para que trabajen entre ellos. Para conseguir este propósito es necesario definir no sólo es sistema radio, sino también una pila de protocolos que permita que los diferentes dispositivos Bluetooth se encuentren entre sí dentro del área de alcance.

La pila Bluetooth de protocolos se puede dividir en dos componentes:

- El host Bluetooth, encargado de la parte relacionada con las capas superiores de enlace y aplicación.
- El controlador Bluetooth, encargado de las tareas relacionadas con el envío de información a través del interfaz de radiofrecuencia.

El interfaz de controlador de host (*Host Controller Interface, HCI*) proporciona un interfaz estandarizado entre el host Bluetooth y el controlador Bluetooth.

Al host Bluetooth es conocido también como el nivel alto de la pila y es implementado en software. Normalmente está integrado con el software o con el sistema operativo del host. Los perfiles Bluetooth están en el nivel más alto de la torre de protocolos. Son software que corre en el hardware del dispositivo host. Por ejemplo, un portátil o un móvil sería el dispositivo host, y el host Bluetooth estaría integrado en el sistema operativo del portátil o del móvil.

El controlador Bluetooth normalmente es un dispositivo hardware, como una tarjeta de ordenador. Cada vez más dispositivos tienen el controlador Bluetooth en un dispositivo.

Los niveles altos de la pila de protocolos se conectan al controlador Bluetooth mediante el HCI.

La pila de protocolos Bluetooth se basa en el modelo de referencia OSI (*Open System Interconnect*) de ISO (*Internacional Standard Organization*) para interconexión de sistemas abiertos. La especificación Bluetooth utiliza una arquitectura de protocolos que divide las diversas funciones de red en un sistema de niveles. En conjunto, permiten el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación y fomentan la interoperabilidad entre los productos de diferentes fabricantes.

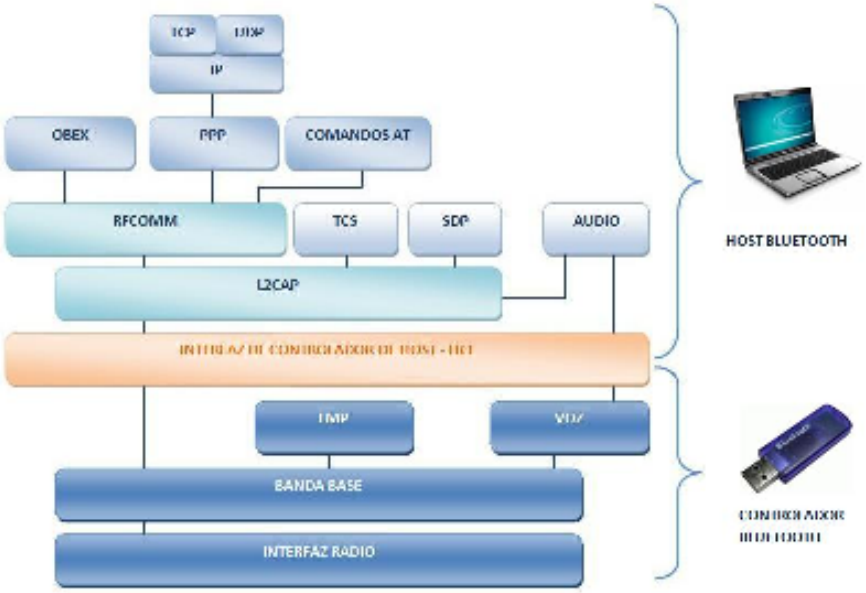


Ilustración 6. Pila de protocolos Bluetooth.

El **interfaz radio** es la capa más baja definida en la especificación Bluetooth. Define los requerimientos de transmisión Bluetooth para operar en la banda de frecuencias ISM **(***industrial, científica y médica*) de 2.4 GHz. Los dispositivos usan un transmisor de salto de frecuencia para contrarrestar las interferencias y la pérdida de intensidad.

La **banda base** junto con el enlace de la capa de control especifican o introducen los procedimientos de acceso de medios y capa física entre dispositivos Bluetooth. La banda base maneja el procesamiento y temporización de canales, y el enlace de la capa de control controla el acceso al canal. Hay dos formas diferentes de enlaces físicos: orientados a conexión síncrona (SCO) y conexión asíncrona (ACL). Un enlace ACL transporta los paquetes de datos, mientras que un enlace SCO soporta el tráfico de audio en tiempo real.

Voz/Audio realmente no es una capa de la pila de protocolos, pero se muestra aquí porque es tratada de forma exclusiva en la comunicación Bluetooth. Los datos de audio son típicamente enrutados directamente desde el nivel de banda base sobre un link SCO.

El **protocolo de gestión de enlace (LMP)** se usa para controlar y negociar todos los aspectos de funcionamiento de la conexión *Bluetooth* entre dos dispositivos. Esto incluye la configuración y el control de comunicaciones lógicas y enlaces lógicos, y el control de los enlaces físicos. El protocolo de gestión de enlace se usa para la comunicación entre los gestores de enlaces

(LM) de los dos dispositivos conectados por la comunicación lógica ACL. El LMP gestiona los aspectos de seguridad, tales como la autenticación y la encriptación, generando, cambiando y comprobando el enlace y la clave de encriptación.

El **interfaz de controlador de host (HCI)** proporciona una interfaz de comandos entre el controlador de la banda base y el gestor de enlaces, y acceso a los parámetros de configuración. Esta interfaz proporciona un método uniforme de acceso a todas las funciones de la banda base Bluetooth.

El **protocolo de adaptación y de control de enlace lógico (L2CAP)** de Bluetooth es compatible con el multiplexado de protocolos de capas superiores, la segmentación y unificación de paquetes, y la comunicación de información sobre la calidad del servicio.

SDP (**Service Discovery Protocol)** es el protocolo encargado de la búsqueda de otros dispositivos Bluetooth y servicios Bluetooth dentro del rango de alcance. A diferencia de en una conexión de LAN, en la cual se une a una red y luego encuentra dispositivos, en un entorno Bluetooth se encuentran los dispositivos antes de encontrar los servicios. SDP está construido sobre la capa L2CAP.

El **protocolo RFCOMM (***Radio Frequency Communication – Comunicación por radio frecuencia***)** proporciona una emulación de los puertos serie sobre L2CAP. Al emular los puertos serie, es compatible con aplicaciones heredadas que utilizan este tipo de conexión y admite, al mismo tiempo, el protocolo OBEX entre otros. El protocolo RFCOMM es compatible con un máximo de 60 conexiones simultáneas entre dos dispositivos *Bluetooth*. El número de conexiones que pueden usarse simultáneamente en un dispositivo *Bluetooth* depende de la aplicación en cuestión.

La **especificación de control de telefonía** (TCS binario) define la señalización de control de llamada para el establecimiento de llamadas de voz y de datos entre dispositivos Bluetooth. Está construida sobre la capa L2CAP.

OBEX (Object Exchange – intercambio de objetos) es un protocolo binario diseñado para permitir que una variedad de dispositivos intercambien datos simple y espontáneamente. La espontaneidad es importante para la tecnología Bluetooth, ya que la duración (corta) de las conexiones ad-hoc se puede configurar. Este protocolo es muy similar a HTTP y es utilizado también sobre otras tecnologías inalámbricas distintas de Bluetooth.

PERFILES BLUETOOTH

A parte de la pila de protocolos, el Bluetooth SIG (Special Interest Group) ha definido distintos perfiles Bluetooth, que deben ser conocidos por los dispositivos Bluetooth para así poder utilizar la tecnología inalámbrica Bluetooth.

Un perfil Bluetooth define los procedimientos por los que los dispositivos Bluetooth se comunican entre sí. Hay un gran número de perfiles, y cada uno de ellos define los diferentes tipos de uso y aplicaciones de la tecnología inalámbrica Bluetooth. De este modo, los desarrolladores pueden crear aplicaciones compatibles con otros dispositivos que se ajusten a este estándar.

Cada perfil incluye, como mínimo, información sobre las siguientes cuestiones:

- Dependencia de otros perfiles
- Propuestas de formato de interfaz de usuario
- Características concretas de la pila de protocolos Bluetooth utilizada por el perfil. Para realizar su función, cada perfil se sirve de ciertas opciones y parámetros en cada capa de la pila. También se puede incluir un breve resumen de los servicios requeridos si resulta necesario.

Dos perfiles pueden usar una combinación distinta de capas de protocolos y un conjunto diferente de características dentro de un mismo protocolo.

Existen cuatro perfiles básicos:

- GAP (Generic Access Profile – Perfil de acceso genérico).
- SPP (Serial Port Profile – Perfil del puerto serie).
- SDAP (Service Discovery Application Profile – Perfil de aplicación de descubrimiento de servicio).
- GOEP (Generic Object Exchange Profile – Perfil genérico de intercambio de objetos).

El **GAP** es la base para los demás perfiles ya que todos los perfiles están basados en este. GAP define los procedimientos generales para establecer conexiones entre dos terminales, incluyendo el descubrimiento de dispositivos Bluetooth, gestión y configuración, y los procedimientos relacionados con los diferentes niveles de seguridad.

El **SDAP** describe las operaciones fundamentales necesarias para el descubrimiento de servicios. Este perfil define los protocolos y los procedimientos que usarán las aplicaciones para localizar los servicios en otros dispositivos Bluetooth.

El **SPP** define los requerimientos de los dispositivos Bluetooth necesarios para configurar y emular conexiones cable RFCOMM entre dos dispositivos. SPP mapea directamente el protocolo RFCOMM y permite reemplazar un cable en aplicaciones mediante la tecnología inalámbrica Bluetooth.

El **GOEP** es un perfil abstracto que se utiliza para transferir objetos de un dispositivo a otro. Este perfil define todos los elementos necesarios para soportar OBEX.

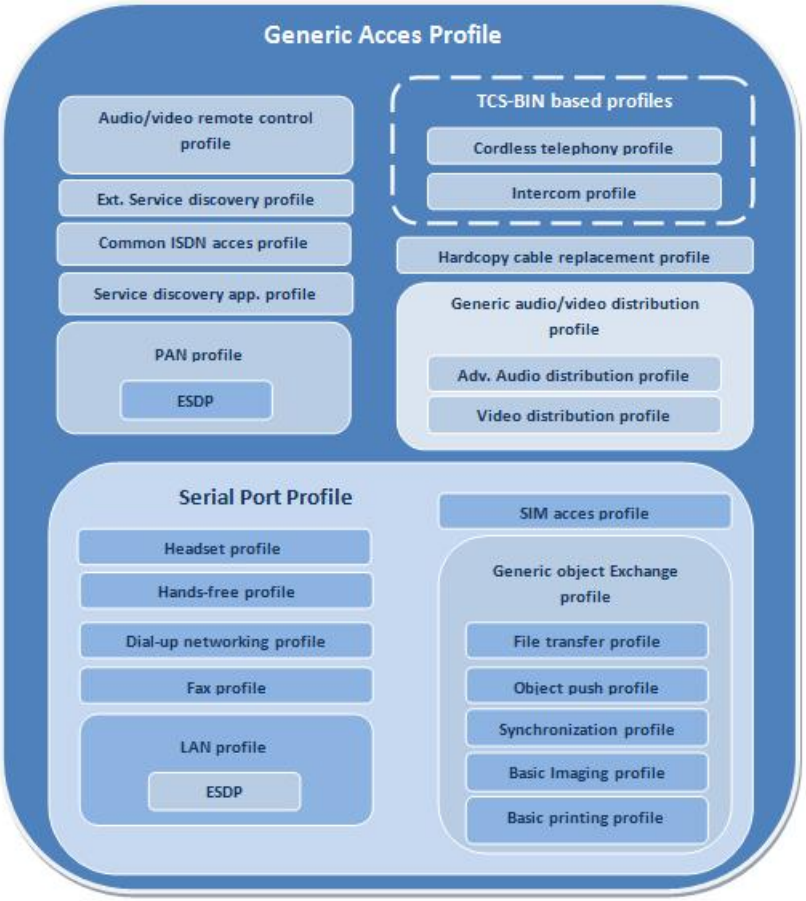


Ilustración 7. Jerarquía de los perfiles Bluetooth.

MAESTROS Y ESCLAVOS

La comunicación Bluetooth se basa en el concepto de maestro/esclavo.

El maestro establece la frecuencia de salto. Los esclavos se sincronizan en tiempo y frecuencia al maestro, siguiendo los saltos de secuencia del maestro.

Todo dispositivo tiene una única dirección Bluetooth así como un reloj Bluetooth, y es la banda base la que describe el algoritmo que calcula una frecuencia de salto de una dirección de un dispositivo Bluetooth y un reloj Bluetooth.

Cuando los esclavos se conectan al maestro, se les comunica la dirección del dispositivo Bluetooth y el reloj del maestro, y los utilizan para calcular la frecuencia de salto. Además de controlar la frecuencia de salto de los dispositivos esclavos, el maestro controla el momento en el que los dispositivos esclavos pueden transmitir.

El maestro permite a los esclavos transmitir asignándoles ranuras para el tráfico de voz o de datos (TDM, *Time Division Multiplexing – Multiplexación por División en el Tiempo*). La cantidad de ranuras de tiempo que se asigna a cada dispositivo depende de los requerimientos de transmisión.

En el caso de transmisión de datos, los esclavos sólo pueden transmitir en contestación a una transmisión del maestro. Las transmisiones de voz requieren transmitir regularmente en ranuras reservadas, aun que no sea contestación a una transmisión del maestro.

El maestro controla también cómo dividir el ancho de banda entre los esclavos y cuándo y con qué regularidad se comunica con cada uno de ellos.

Los enlaces inalámbricos Bluetooth se forman en el contexto de una **piconet**. Una piconet se compone de dos o más dispositivos que ocupan el mismo canal físico, por lo que están sincronizados con un mismo reloj y secuencia de salto.

PICONETS Y SCATTERNET

Una **piconet** se puede definir como un grupo de esclavos operando con un maestro común, y siguiendo la frecuencia de salto y el temporizador del maestro.

Un dispositivo Bluetooth puede utilizarse simultáneamente en dos o más piconets mediante un multiplexado por división de tiempo. Ahora bien, este dispositivo Bluetooth no actuará nunca como maestro en más de una piconet. Esto se debe a que la piconet está determinada por la sincronización con el reloj Bluetooth del dispositivo maestro. En cambio, este dispositivo Bluetooth sí podrá hacer las veces de esclavo en diversas piconets.

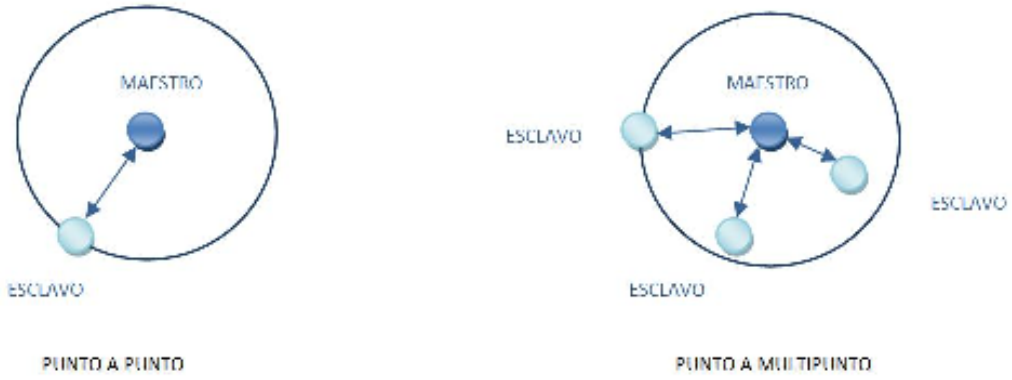


Ilustración 8. Ejemplo de Piconets.

La especificación limita el número de esclavos en una piconet a siete, comunicándose cada esclavo con un maestro (que es compartido). Para conseguir un mayor número de esclavos posibles y una mayor área de cobertura, se deben unir varias piconets en lo que se conoce como una **scatternet**, donde algunos dispositivos son miembros de más de una piconet. Cuando un dispositivo pertenece a más de una piconet, debe hacer reparto de las ranuras de tiempo, asignado unas a una piconet y otras a la otra piconet.

No es posible tener un dispositivo que sea maestro de dos piconets diferentes, ya que todos los esclavos están sincronizados a la frecuencia de salto del maestro.

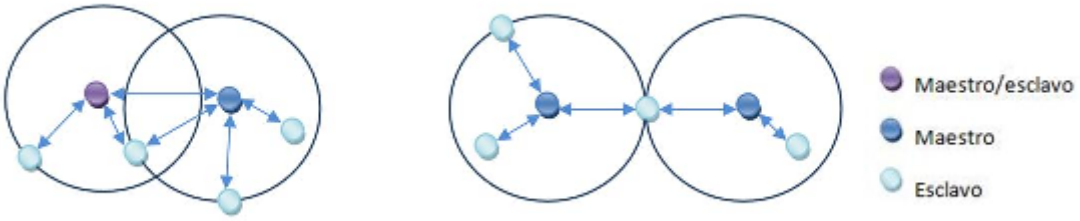


Ilustración 9. Ejemplo de Scatternets

Existen problemas de interferencia, porque aun que los dispositivos que comparten una piconet, deben sincronizarse para "evitarse" unos a otros, otras piconets que se encuentren en el área de cobertura podrían tener la misma frecuencia y por tanto colisionar con ellos. Si hubiera una colisión en un determinado canal, los paquetes se perderían pero serían retransmitidos en el caso de paquetes de datos, en el caso de que fueran paquetes de voz, serían ignorados. Por lo tanto, a mayor número de piconets en un área determinada, mayor número de retransmisiones y por lo tanto peores estadísticas.

2.3. JSR 82

JSR-82 es la forma de unir el mundo J2ME con la tecnología Bluetooth, es un estándar definido por la Java Community Process para poder desarrollar aplicaciones Bluetooth en Java.

JSR-82 consta de dos paquetes independientes entre sí y que dependen de javax.microedition.io:

- javax.bluetooth: define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación.
- javax.obex: permite manejar el protocolo de alto nivel OBEX.

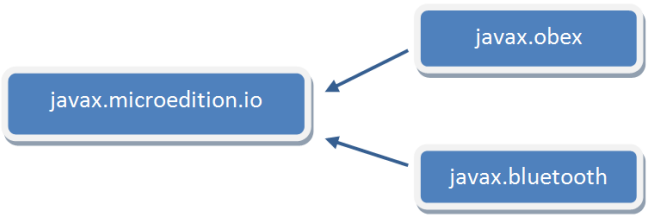


Ilustración 10. Paquetes de JSR-82

REQUERIMIENTOS

Este API está diseñado para dispositivos que tienen las siguientes características:

- Mínimo 512K de memoria disponible para la plataforma Java (ROM, Flash y RAM). Los requerimientos para la memoria de las aplicaciones son adicionales.
- Hardware para la comunicación Bluetooth.
- Implementación de la configuración CLDC de J2ME.

Los requerimientos para el sistema Bluetooth son los siguientes:

- El sistema deberá tener validado por el Programa de calificación de Bluetooth al menos los perfiles GAP, SDAP y SPP.
- Los siguientes niveles serán soportados como definidos en la versión 1.1 de la especificación Bluetooth, y la implementación de este API da acceso a ellos:
  - SDP
  - RFCOMM
  - L2CAP
- El sistema debe proporcionar una entidad denominada Centro de Control de Bluetooth (BCC), que es una especie de "panel de control" que permite a un usuario o a un fabricante definir valores específicos para ciertos parámetros de configuración en una pila.

EL CENTRO DE CONTROL BLUETOOTH

El centro de control Bluetooth forma parte de la especificación JABWT (Java APIs for Bluetooth wireless technology – APIs de Java para la tecnología inalámbrica Bluetooth) pero no hay ninguna API Java que proporcione acceso directo a él.

La necesidad de un BCC viene de la necesidad de prevenir que una aplicación afecte negativamente a otra aplicación.

La autoridad principal para los ajustes de los dispositivos Bluetooth es el BCC.

Los detalles del BCC se dejan para la implementación, que puede ser una aplicación interactiva o no con el interfaz del usuario. El BCC debe ser una

aplicación nativa, con un API separada o simplemente una serie de ajustes definidos por el fabricante.

Las principales tareas que tiene el BCC son:

- Resolver los conflictos que se puedan dar entre peticiones de distintas aplicaciones.
- Permitir modificar las propiedades del dispositivo Bluetooth.
- El manejo de las operaciones de seguridad que puedan requerir la interacción con el usuario.

EL PAQUETE JAVAX.BLUETOOTH

La comunicación Bluetooth sigue el modelo cliente-servidor. Un servicio Bluetooth es una aplicación actuando como un servidor que ofrece algún tipo de asistencia para los dispositivos clientes vía comunicación Bluetooth. Esta asistencia generalmente toma la forma de una capacidad o función que no está disponible localmente en el dispositivo cliente.

Los desarrolladores pueden definir sus propias aplicaciones de servidor Bluetooth más allá de los especificados en los perfiles de Bluetooth y hacer que estos servicios estén a disposición de los clientes remotos. Para ello, deben definir un servicio de registro (*service record*) que describa el servicio y añadirlo a la base de datos de descubrimiento de servicios (SDDB) del dispositivo local.

Los pasos que debe realizar un servidor Bluetooth son los siguientes:

- Crear una conexión servidora.
- Especificar los atributos de servicio.
- Abrir las conexiones cliente.

Un cliente Bluetooth deberá realizar las siguientes operaciones para comunicarse con un servidor Bluetooth:

- Búsqueda de dispositivos.
- Búsqueda de servicios.
- Establecimiento de la conexión.
- Comunicación.

Existen dos mecanismos de conexión en el paquete `javax.bluetooth`:

- SPP: la URL comenzará por “btspp://” y se usará un `InputStream` y un `OutputStream`.
- L2CAP: la URL comenzará por “btl2cap://” y se enviarán y recibirán arrays de bytes.

DESCUBRIR DISPOSITIVOS Y SERVICIOS

Para poder realizar la comunicación Bluetooth, los dispositivos deben encontrarse, para ello el paquete `javax.bluetooth` ofrece una serie de clases que permiten que esto sea posible. La búsqueda de dispositivos es tarea del cliente.

LA CLASE LocalDevice

La clase `LocalDevice` representa al dispositivo local, y permite obtener información sobre el dispositivo: modo de conectividad, dirección Bluetooth y nombre ("friendly-name").

Se debe tener una única instancia de esta clase, para obtenerla existe el método `getLocalDevice()`.

Para definir la visibilidad de este dispositivo está el método `setDiscoverable(int mode)` que puede recibir como parámetro:

- `DiscoveryAgent.`**`GIAC`** (General/Unlimited Inquiry Access Code): Conectividad ilimitada.
- `DiscoveryAgent.`**`LIAC`** (Limited Dedicated Inquiry Access Code): Conectividad limitada.
- `DiscoveryAgent.`**`NOT_DISCOVERABLE`**: El dispositivo no será visible para el resto de dispositivos.

La case `LocalDevice` proporciona también el método `getRecord()` que hace que la aplicación obtenga su propio `ServiceRecord`, de este modo el servidor podría modificar los atributos del `ServiceRecord`.

LA CLASE DeviceClass

La clase `DeviceClass` tiene métodos para recuperar los valores de las clases que describen las propiedades de un dispositivo. Describe el tipo de dispositivo que es, un teléfono móvil, una pda…

LA CLASE DiscoveryAgent

Esta clase proporciona métodos para el descubrimiento de dispositivos y de servicios. Este objeto es único y lo obtendremos a través del método `setDiscoveryAgent()` del objeto `LocalDevice`.

Para el descubrimiento de dispositivos, esta clase proporciona el método `startInquiry()` que hace que el dispositivo local en modo búsqueda (no bloqueante). Este método requiere dos argumentos. El primer argumento es un entero que especifica el modo de conectividad que deben tener los dispositivos a buscar. Este valor deberá ser `DiscoveryAgent.GIAC` o bien `DiscoveryAgent.LIAC`. El segundo argumento es un objeto que implemente `DiscoveryListener`. A través de este último objeto serán notificados los dispositivos que se vayan descubriendo.

El método `retrieveDevices()` devuelve los dispositivos remotos encontrados o conocidos (encontrados en una búsqueda previa).

EL INTERFAZ DiscoveryListener

Esta interfaz permite que una aplicación especifique un listener que escuche y notifique eventos a la aplicación cada vez que se descubre un dispositivo, un servicio, o se finaliza una búsqueda.

El método `deviceDiscovered()` se llama cada vez que un dispositivo es encontrado durante una búsqueda. Cuando la búsqueda se ha completado o se cancela, se llamará a `inquiryCompleted()`. Este método puede recibir como parámetro `INQUIRY_COMPLETED`, `INQUIRY_ERROR` o `INQUIRY_TERMINATED`.

LA CLASE UUID

La clase `UUID` encapsula enteros sin signo que son de 16 bits, 32 bits o 128 bits de longitud. La clase se utiliza para representar un identificador único universal.

Sólo los atributos de servicio representados por un UUID son buscados por el Bluetooth SDP.

La especificación Bluetooth define algunos UUID (de 16-bit o 32-bit) y describe cómo un UUID de 16-bit o 32-bits se convierte en un UUID de 128 bits. Requieren esta conversión porque normalmente se comparan UUIDs de 128 bit.

LA CLASE DataElement

Esta clase contiene los distintos tipos de datos que un valor de atributo de servicio Bluetooth puede asumir.

Tipos de datos válidos son:

- Enteros de diferente longitud con o sin signo.
- Cadenas de texto.
- Booleanos.
- UUID
- Secuencias de cualquiera de estos tipos.

La clase también presenta una interfaz para construir y recuperar el valor de un atributo de servicio.

EL INTERFAZ ServiceRecord

Un `ServiceRecord` describe un servicio Bluetooth a los clientes. Se componen de un conjunto de atributos de servicio, donde cada atributo es un par: un atributo ID y un valor de atributo.

El `ServiceRecord` se obtiene a través de la clase `LocalDevice` con el método `getRecord()`. Una vez tenemos el `ServiceRecord` podremos establecer sus atributos mediante el método `setAttributeValue()` al que le pasaremos un identificador numérico y un objeto `DataElement` que representará el valor del atributo de servicio.

REGISTRO DEL SERVICIO

Se llama registro del servicio a las tareas del servidor relacionadas con la notificación al cliente de los servicios disponibles. Estas tareas son:

- Crear un ServiceRecord que describa el servicio ofrecido por la aplicación.
- Añadir el ServiceRecord al SDDB del servidor para avisar a los clientes potenciales de este servicio.
- Actualizar el ServiceRecord en el SDDB del servidor si las características del servicio cambian.
- Quitar o deshabilitar el ServiceRecord en el SDDB del servidor cuando el servicio no está disponible.

Los métodos `open()`, `acceptAndOpen()` y `close()` se usan en muchas aplicaciones J2ME para operaciones Entrada/Salida.

- **`open()`**: se le pasa la URL como argumento, la implementación crea un `ServiceRecord`.
- **`acceptAndOpen()`**: escucha y procesa las conexiones cliente. En una conexión servidora SPP este método devuelve un `javax.microedition.StreamConnection`, y en una conexión servidora L2CAP devuelve un objeto del tipo `javax.bluetooth.L2CAPConnection`.
- **`close()`**: cuando se llama a este método el `ServiceRecord` es eliminado de la SDDB.

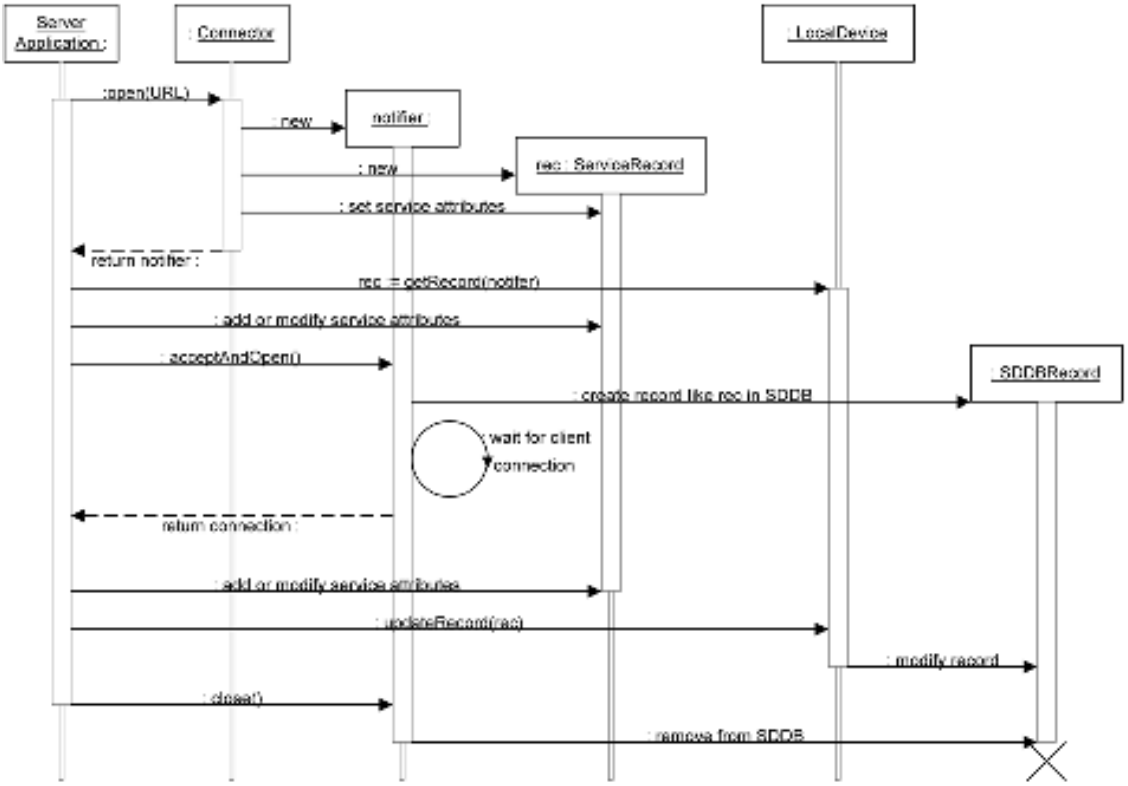


Ilustración 11. Colaboración entre la implementación y la aplicación servidora para el registro del servicio.

3. ENTORNO DE DESARROLLO

La aplicación que se desarrolla en el presente proyecto fin de carrera, está desarrollada en lenguaje Java. Para este lenguaje, existen varios entornos de desarrollo que ayudan al programador a hacer las aplicaciones. En mi caso, he elegido como entorno de desarrollo Eclipse. Eclipse es un entorno de desarrollo de código abierto multiplataforma.

También es necesario tener instalado el Sun Wireless Toolkit, que es un conjunto de herramientas para crear aplicaciones Java que se ejecutan en dispositivos compatibles con la especificación de la tecnología Java para la industria inalámbrica (JTWI, JSR 185) y la especificación de la Arquitectura de Servicios Móviles (MSA, JSR 248). Se trata de herramientas de construcción, utilidades, y un emulador de dispositivos.

También se necesita tener instalado EclipseME que es un plugin de Eclipse para ayudar a desarrollar MIDlets J2ME. EclipseME hace el "trabajo sucio" de los kits de herramientas de conexión inalámbrica para el entorno de desarrollo Eclipse, permitiendo que nos centremos en el desarrollo de la aplicación, en lugar de preocuparnos por las necesidades especiales de desarrollo J2ME.

Los pasos que hay que seguir para poder programar en J2ME utilizando Eclipse en nuestro ordenador son los siguientes:

- Ir a la página http://java.sun.com/products/sjwtoolkit/download.html descargar e instalar el Sun Wireless Toolkit.
- Desde la página www.eclipse.org se descarga y se instala el Eclipse.
- En http://EclipseME.org/ descargar EclipseME.

A continuación se explica con más detalle la instalación y configuración de EclipseME.

Los puntos que se desarrollan a continuación se ha extraído del documento ***"Manual de Eclipse para el desarrollo de aplicaciones Java ME"*** de **Manuel Caballo Gil** (Diciembre 2007) de la Universidad Carlos III de Madrid.

3.1. INSTALACIÓN DE EclipseME

En primer lugar, para instalar el *plugin* deberemos buscar nuevas actualizaciones para nuestro IDE de Eclipse, tarea que se hace en el menú *Help/Software Updates/Find and Install*:

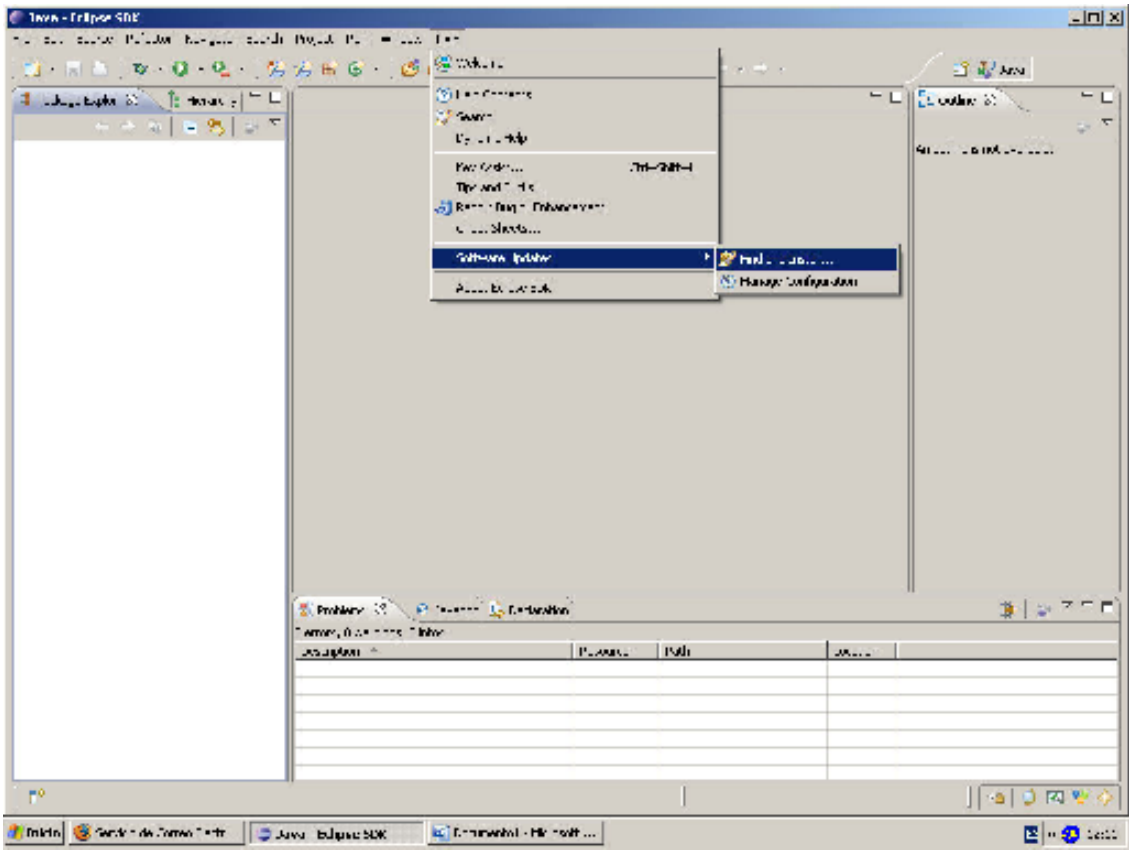


Ilustración 12. Búsqueda de nuevas actualizaciones del IDE Eclipse.

Seleccionaremos la opción "*Search new features to install*" y daremos a siguiente.

Nos aparecerá una ventana con todos los sitios remotos en los que se buscarán actualizaciones. El sitio del plugin no estará seleccionado por defecto así que tendremos que añadirlo pinchando en "*New Remote Site*" y completando la ventana que se abrirá del siguiente modo:

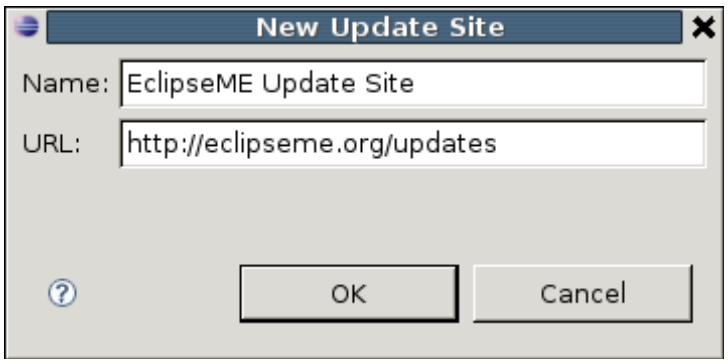


Ilustración 13. Añadiendo el sitio remoto del plugin EclipseME.

Una vez hecho esto comprobaremos que el sitio se ha añadido correctamente si en la ventana de sitios de actualizaciones aparece el que acabamos de crear:

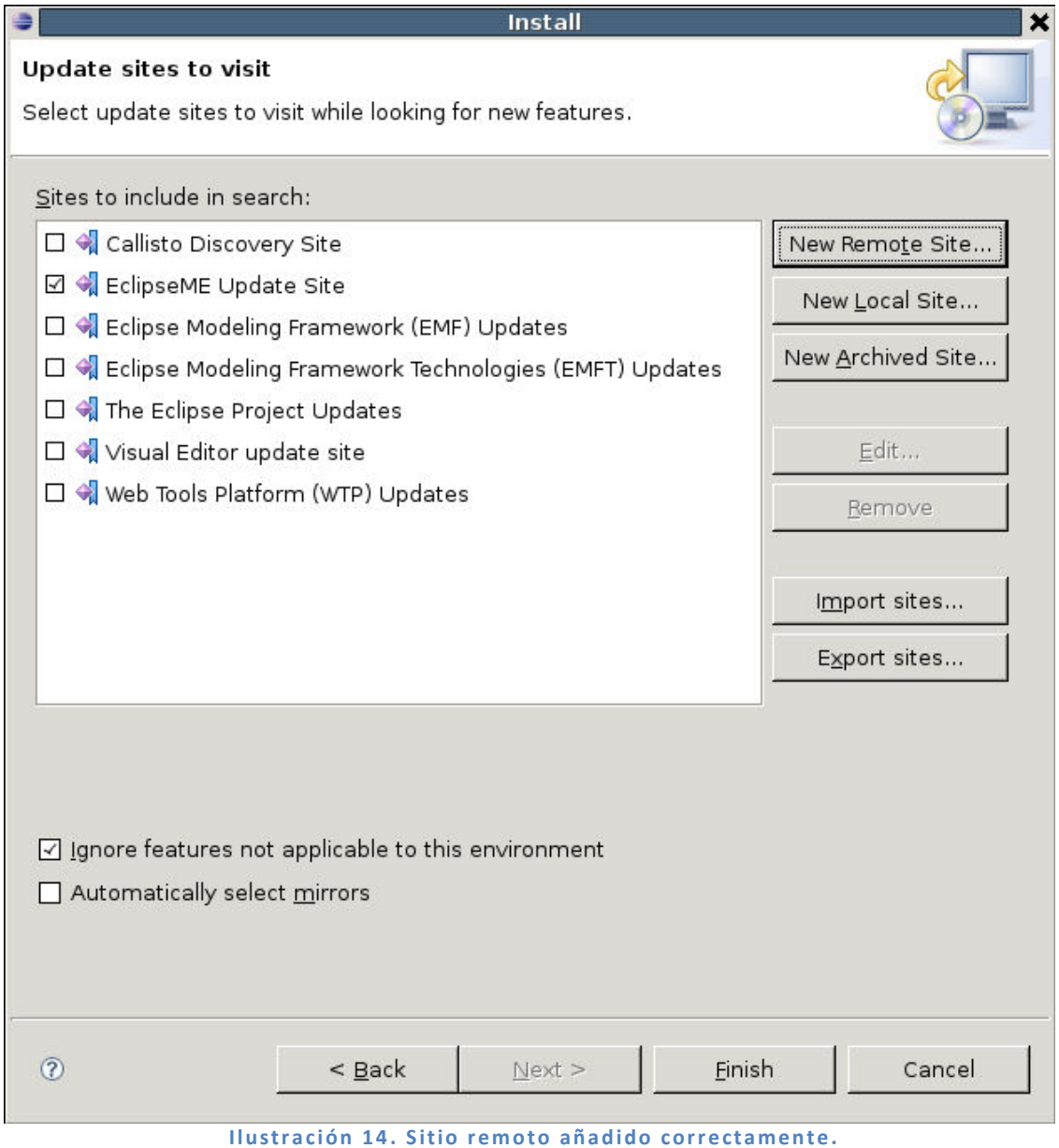


Ilustración 14. Sitio remoto añadido correctamente.

Pincharemos en *Finish* para iniciar la búsqueda y posteriormente seleccionaremos todas las actualizaciones disponibles. Pulsaremos en siguiente y aceptaremos los términos de la licencia para comenzar la descarga:

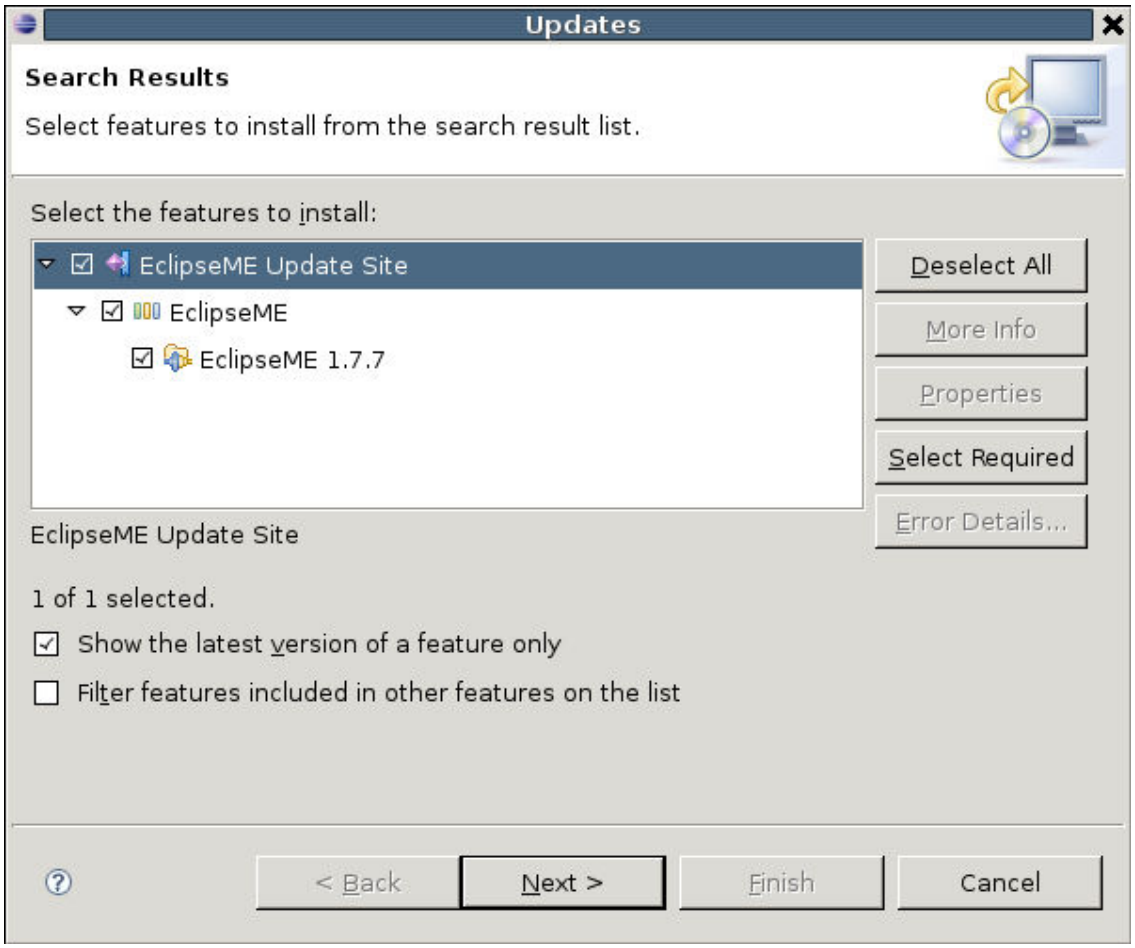


Ilustración 15. Descarga de acutalizaciones.

Una vez finalizada la descarga pulsaremos el botón siguiente (*Next*) para confirmar los términos de la licencia para la instalación del *plugin*:

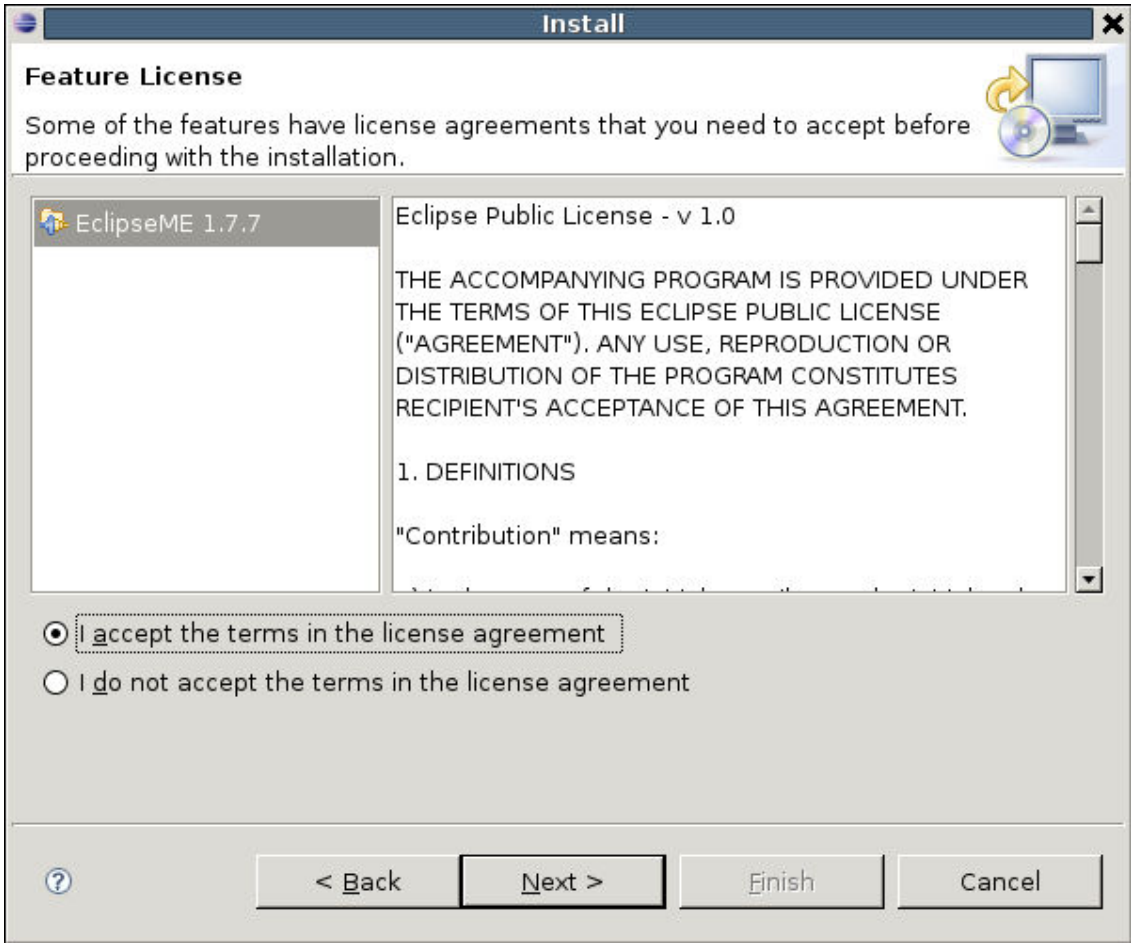


Ilustración 16. Licencia de instalación de EclipseME.

En este punto pulsaremos en el botón *Next*, y aparecerá en pantalla información detallada del *plugin* a instalar y el directorio de instalación.

Eclipse intentará, por defecto, instalar EclipseME en el directorio de los *plugins* habitual, por lo que deberemos seleccionar un directorio de nuestra cuenta para que la instalación sea completada satisfactoriamente. Esto se hará haciendo *click* en el botón "*change location*".

Cuando hagamos esto aparecerá una ventana en la que podremos añadir un nuevo directorio adicional donde instalaremos nuestras nuevas utilidades. Pincharemos en "*Add Location*" y selecionaremos en la ventana que aparecerá el directorio deseado. Pulsamos *Aceptar* y la ventana quedará como se muestra a continuación:

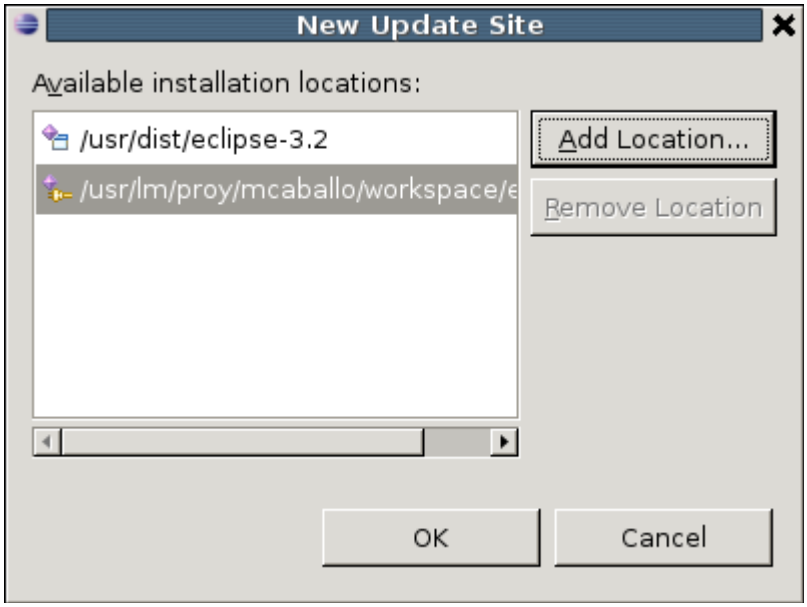


Ilustración 17. Directorio de instalación.

Una vez cambiado esto en la ventana de instalación aparecerá algo parecido a esto:

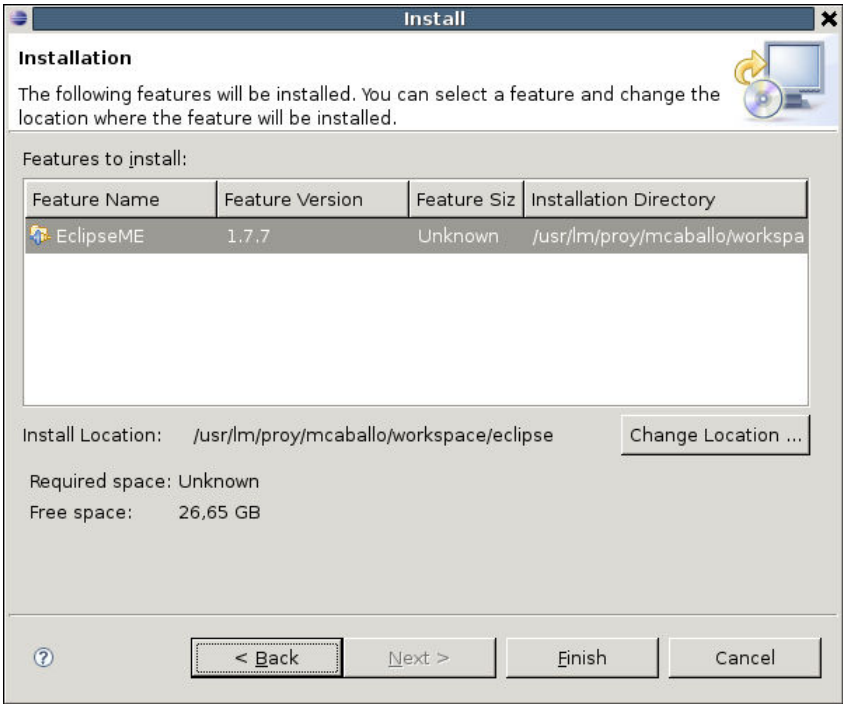


Ilustración 18. Instalación EclipseME.

Pulsamos *Finish* y comenzará la descarga de las utilidades de EclipseME. Una vez termine la descarga comenzaremos con la instalación pulsando el botón "*install all*" de la ventana que se muestra a continuación:

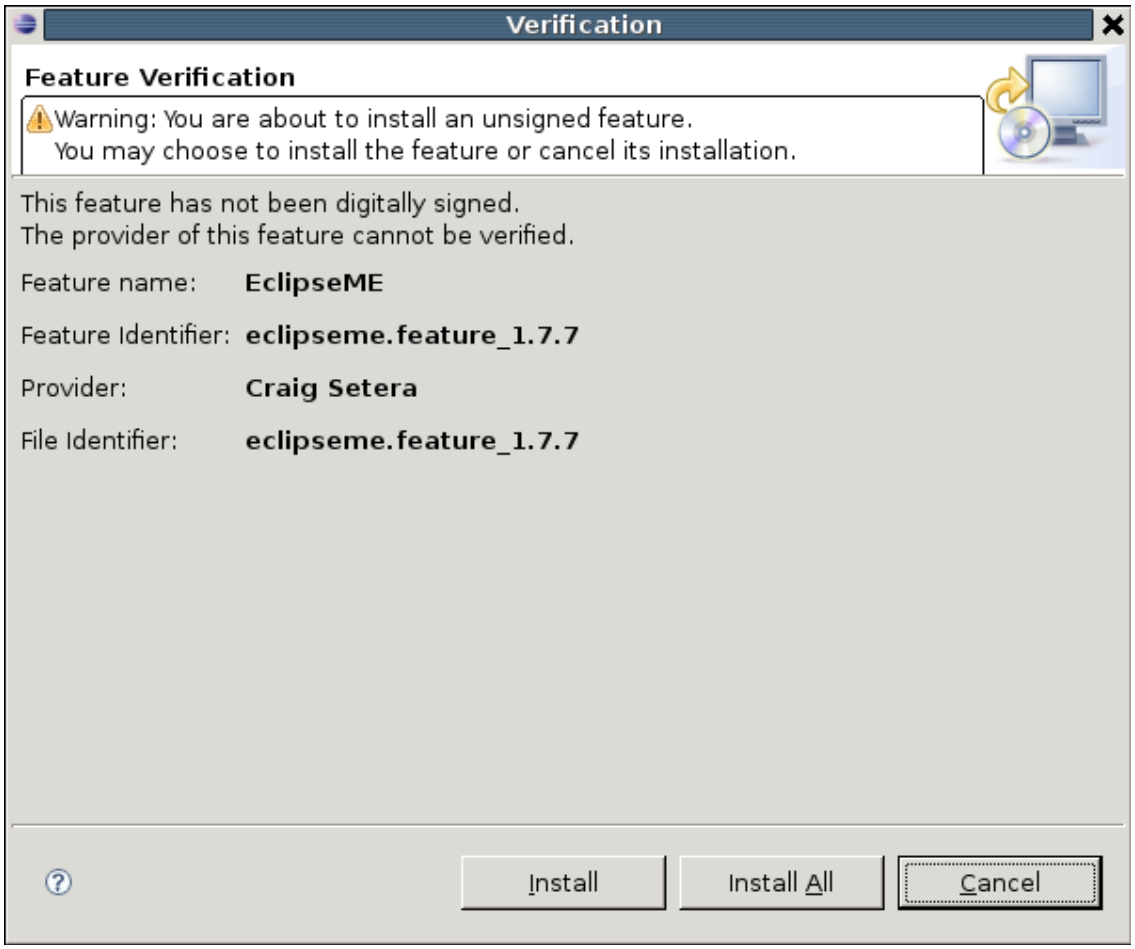


Ilustración 19. Verificación de instalación.

Cuando finalice la instalación eclipse mostrará un mensaje diciendo que para que los cambios surtan efecto se debe reiniciar el IDE. Pulsaremos *Yes*.

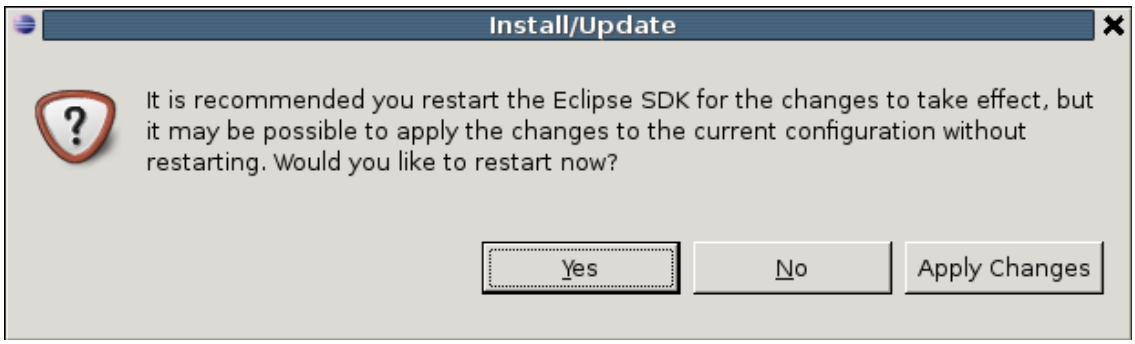


Ilustración 20. Instalación finalizada.

Para comprobar que la instalación se ha realizado con éxito deberemos seleccionar el menú "*File/New/Other…*" donde se nos dará la posibilidad de crear nuevos proyectos de J2ME:

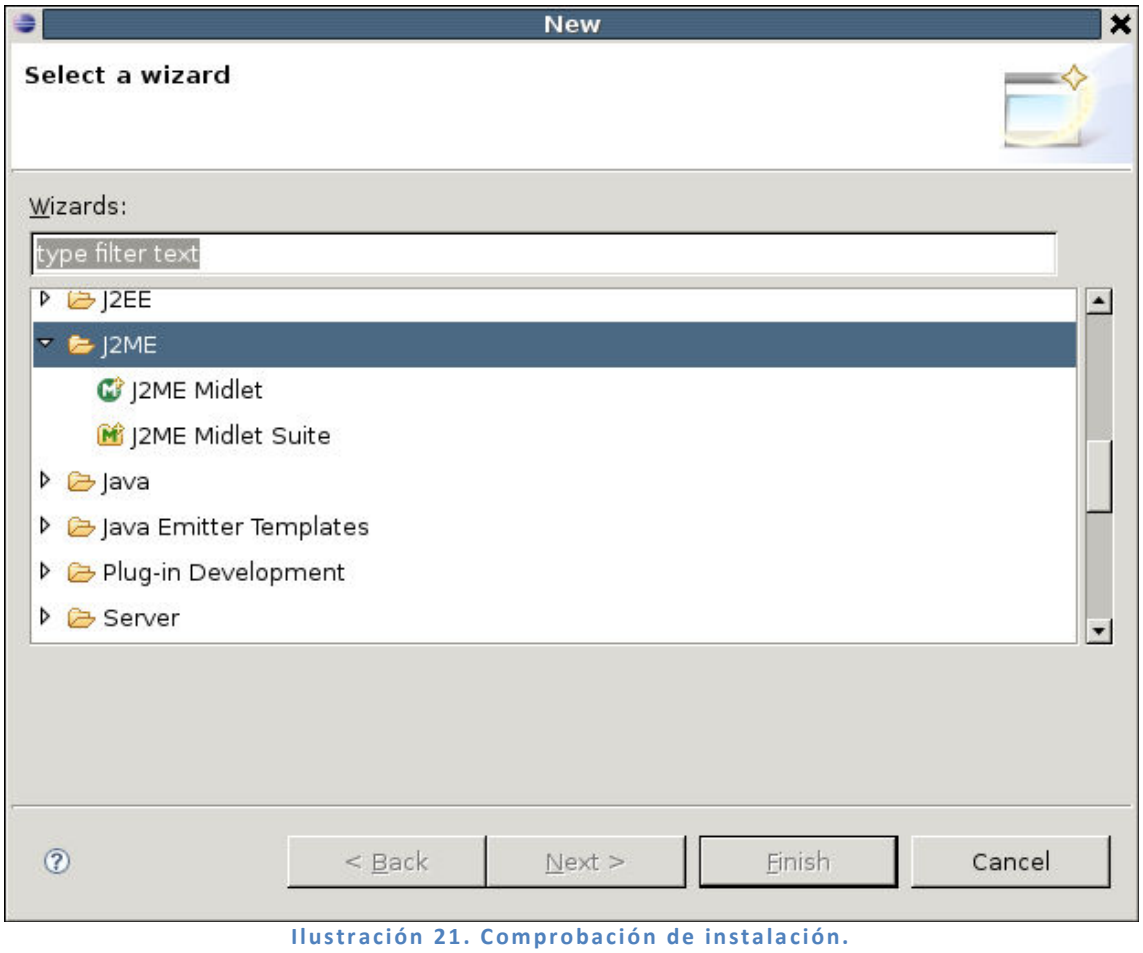


Ilustración 21. Comprobación de instalación.

3.2. CONFIGURACIÓN DE EclipseME

Antes de crear nuestro primer proyecto de J2ME en eclipse deberemos configurar el plugin EclipseME para poder conectarlo al Wireless Toolkit (en adelante WTK) que previamente deberemos tener instalado en nuestra cuenta. Para ello deberemos acudir al menú "*Window/Preferences*"

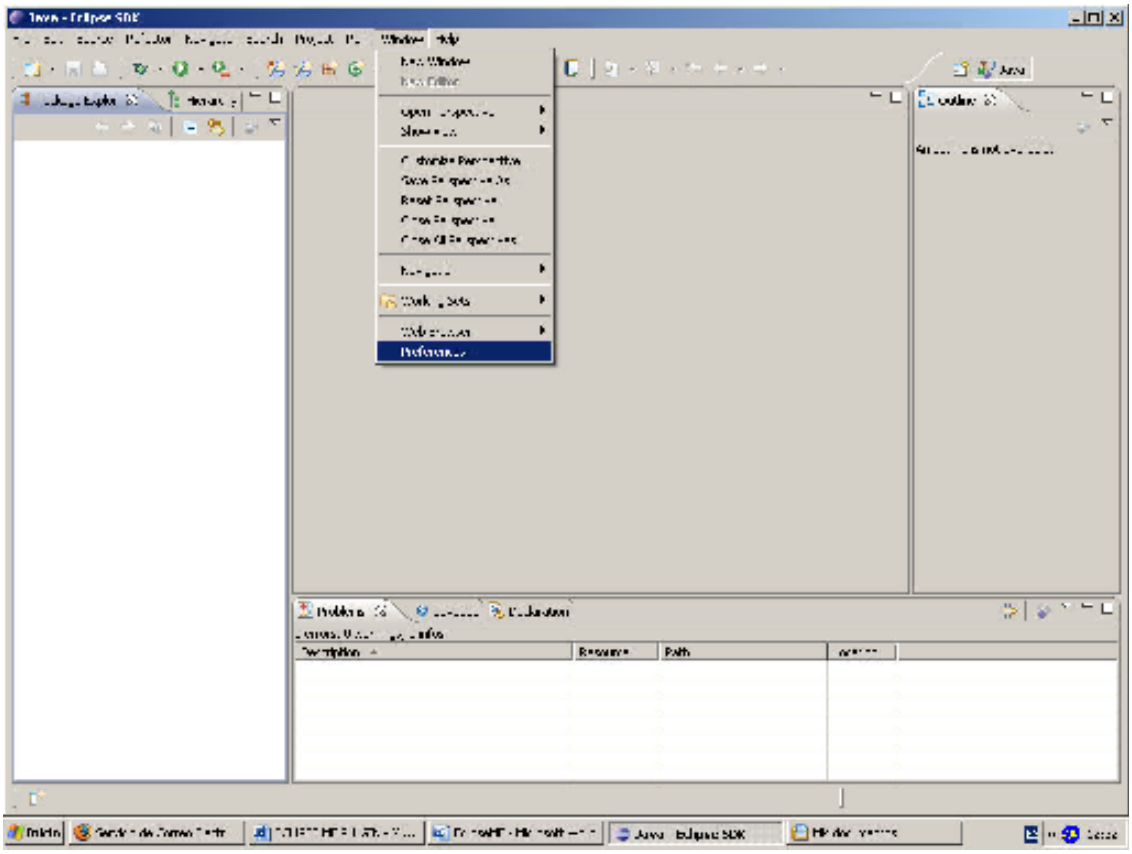


Ilustración 22. Configuración EclipseME.

En la ventana que se abrirá seleccionaremos las opciones de J2ME, donde tendremos que configurar la ruta del WTK así como la de una función de preprocesamiento que ofrece EclipseME.

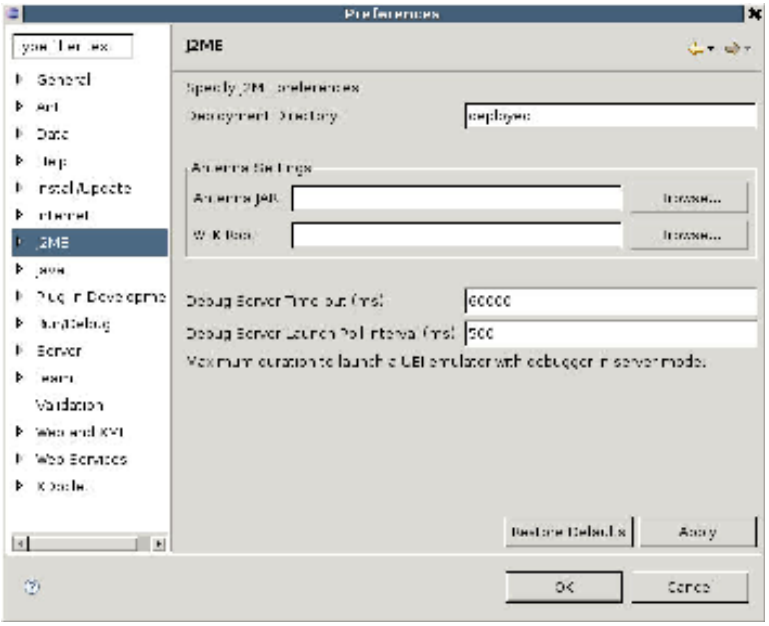


Ilustración 23. Preferencias en la configuración EclipseME.

La ruta de instalación del WTK la conoceremos de antemano, así como el directorio en el que está ubicado el preprocesador. En el caso de Linux se encontrará en el directorio de instalación de plugins de eclipse que hemos especificado anteriormente en nuestra cuenta y la librería tendrá un nombre parecido a "*antenna.preprocessor.v2_1.7.7.jar*".

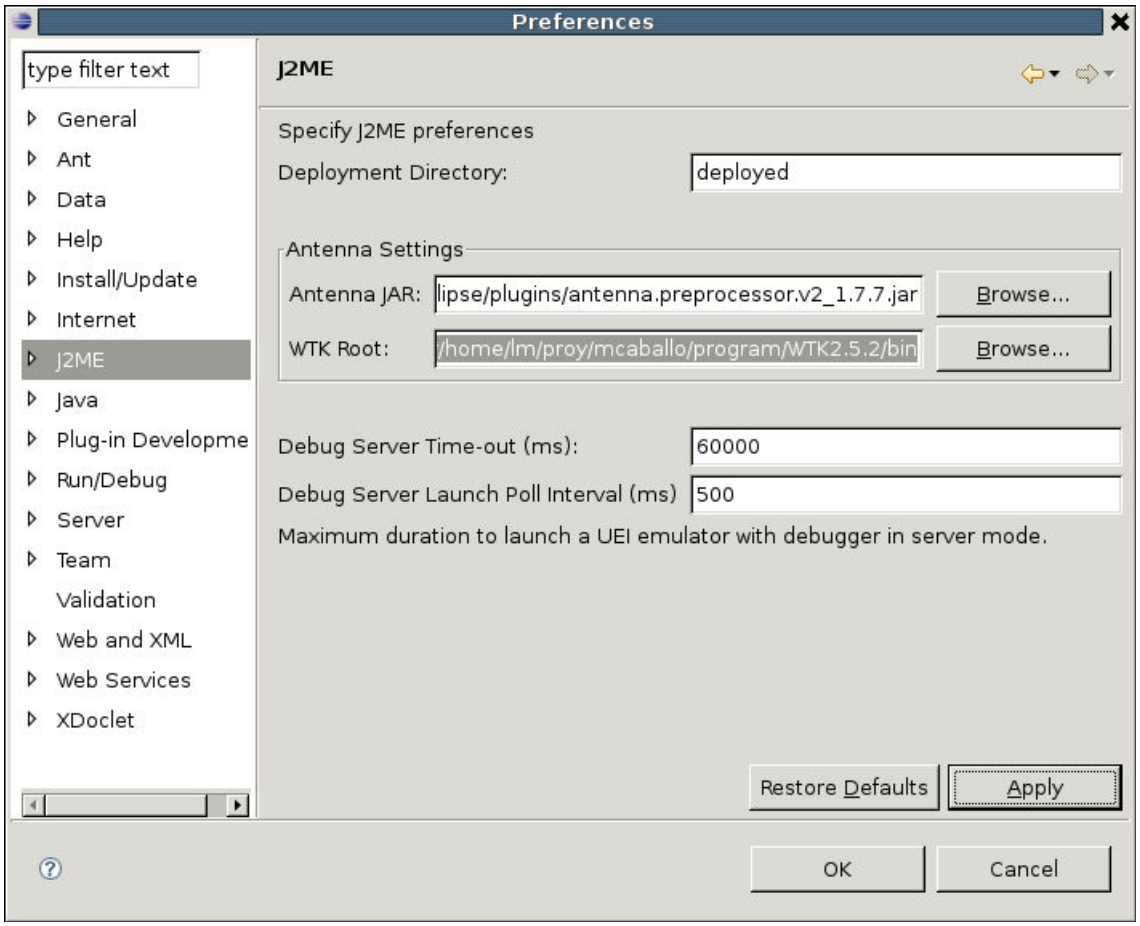


Ilustración 24. Configuración de las preferencias J2ME en EclipseME.

Una vez hecho esto deberemos añadir un nuevo dispositivo que será aquel en el que desplegaremos nuestros futuros proyectos. Para esto deberemos seleccionar la opción "*Devices Management*" que en principio aparecerá vacía:

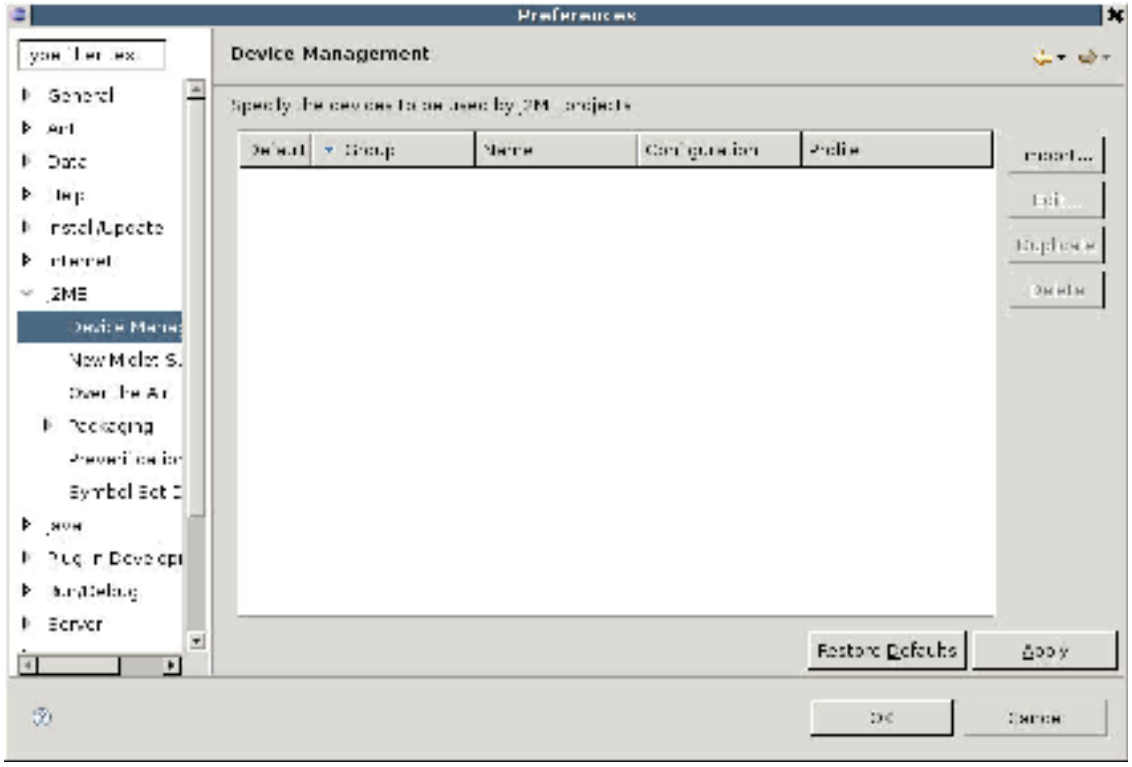


Ilustración 25. Añadiendo un dispositivo para ejecutar los proyectos J2ME.

Pincharemos sobre el botón "*import*" y se nos pedirá un directorio en el que buscar. De nuevo aquí seleccionaremos el directorio de instalación del WTK y pincharemos sobre el botón "*Refresh*" para que busque los dispositivos disponibles. Si todo va bien obtendremos una ventana como esta:

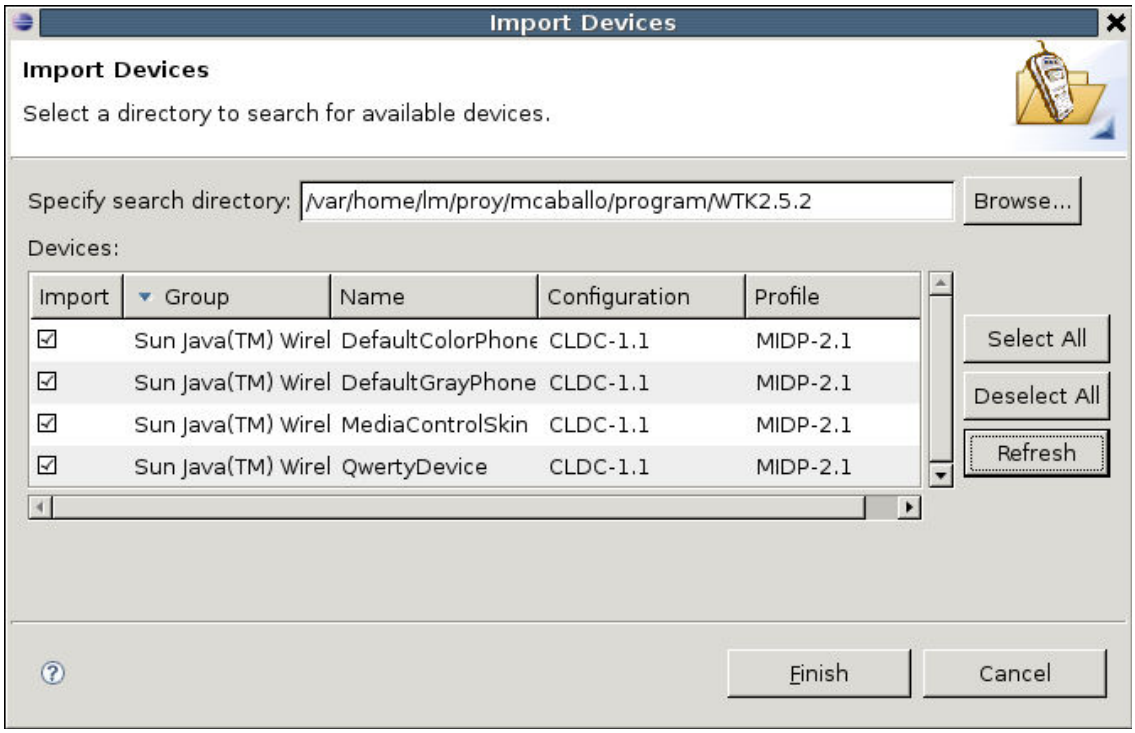


Ilustración 26. Dispositivos disponibles.

Pincharemos sobre "*Finish*" y seleccionaremos el dispositivo que queramos en la ventana "*Device Management*" que antes nos aparecía vacía:

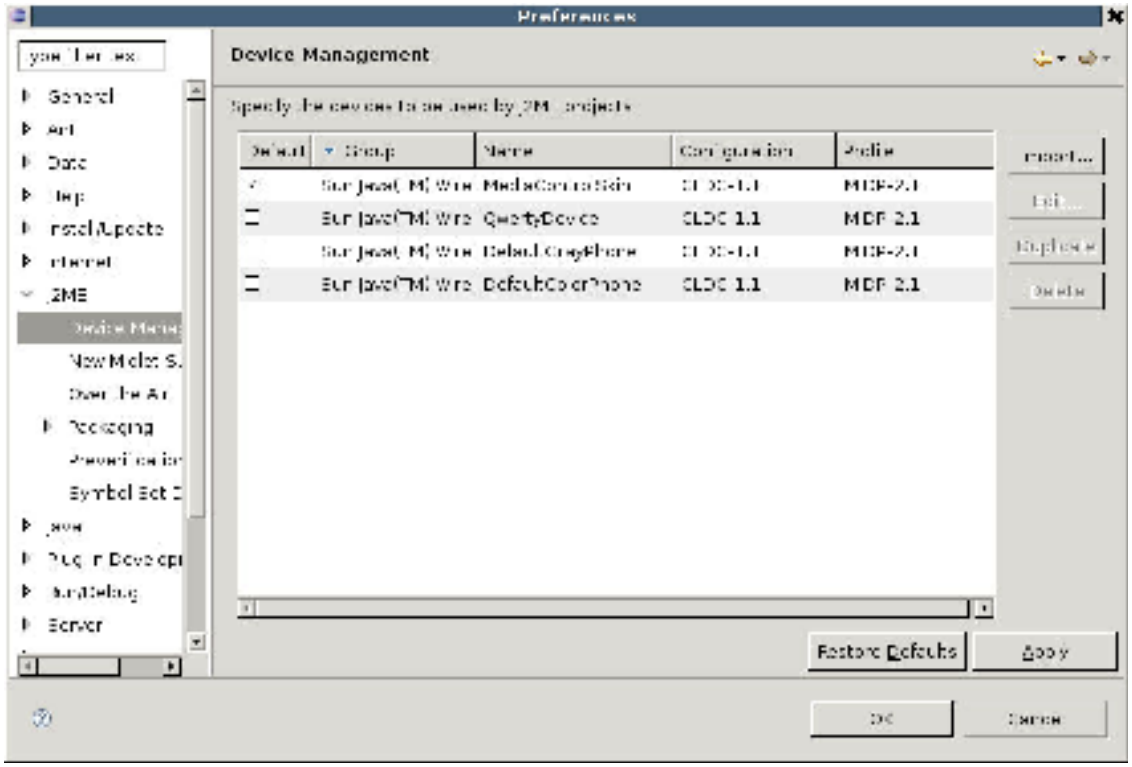


Ilustración 27. Selección del dispositivo.

En este punto ya estaríamos en condiciones de crear nuevos proyectos pero si nos fijamos en la página de EclipseME (http://eclipseme.org) se especifica que para un correcto funcionamiento del *plugin* deberemos hacer un último cambio.

En el mismo menú "*Window/Preferences*" Seleccionaremos la opción "*Java/Debug*":

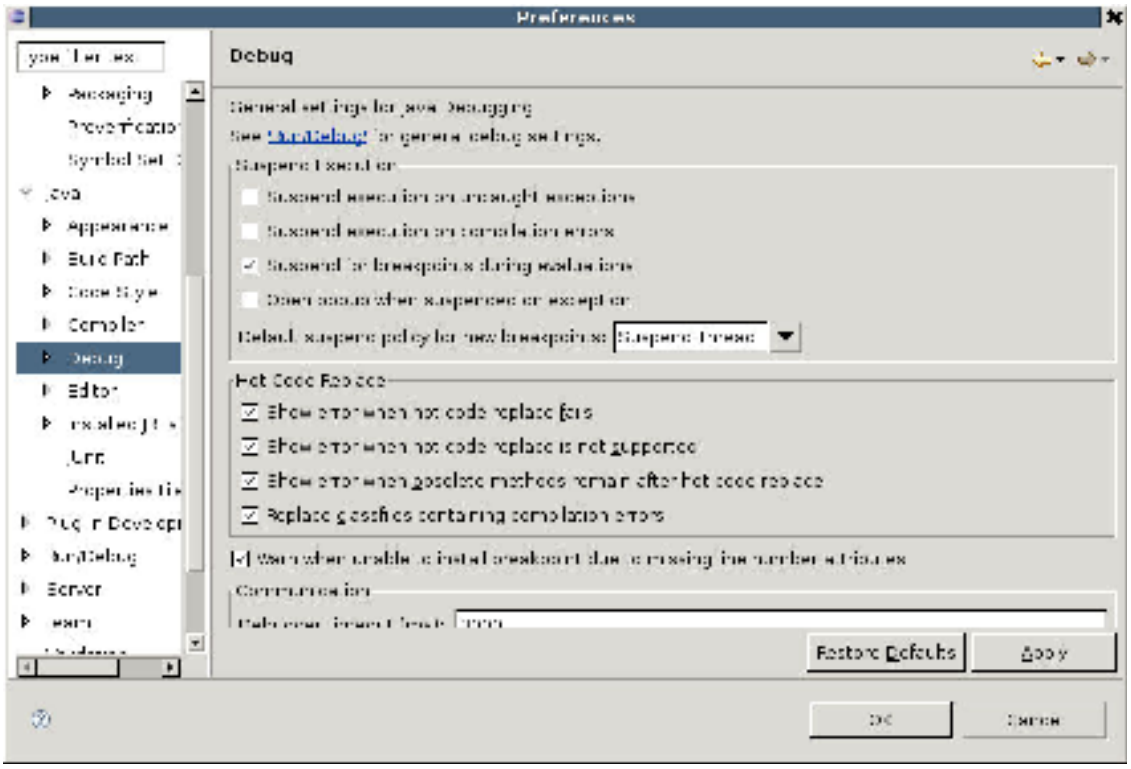


Ilustración 28. Debug.

Donde deberemos deseleccionar las opciones "*Suspend execution on uncaught exceptions*" y "*Suspend execution on compilation errors*", tal y como aparece en la captura anterior.

4. TRABAJO REALIZADO

4.1. INTRODUCCIÓN

El presente proyecto fin de carrera se basa en la arquitectura cliente-servidor.

El servidor será el móvil del profesor, que publicará mediante Bluetooth exámenes tipo test para que los alumnos (clientes) los resuelvan.

Los clientes, los móviles de los alumnos. Una vez que el profesor les comunica que el test está publicado vía Bluetooth, los alumnos ejecutarán su aplicación cliente para descargarlo, contestarlo y enviarle las soluciones (número de aciertos obtenidos) al profesor (servidor).

4.2. ARQUITECTURA DE LA APLICACIÓN

BLUETOOTHSERVER

Este es el módulo servidor, y tiene las siguientes funciones:

- Mantener el servicio ofrecido.
- Publicar vía Bluetooth los test que el profesor considere oportuno.
- Aceptar conexiones Bluetooth por parte de los alumnos.
- Enviar los test solicitados por los alumnos.
- Recibir y gestionar los resultados obtenidos por los alumnos.

Una vez arrancada la aplicación servidor, aparece en la pantalla del teléfono móvil una lista con los exámenes disponibles para ser publicados.

Ilustración 29. Servidor – Pantalla de exámenes disponibles.

Cuando se selecciona uno (simplemente bajando con la tecla hasta llegar al examen que se desea) hay varias posibilidades:

Ilustración 30. Servidor – Pantalla de opciones de un examen.

1. Publicar test: esta opción se selecciona cuando se quiere poner visible el test para los clientes Bluetooth, en este caso los alumnos. Cuando se publica el test, la imagen de estado cambia a color verde.

Ilustración 31. Servidor – Estado de un examen: publicado.

2. Dejar de publicar test: esta opción se selecciona cuando el profesor quiere que el test publicado deje de estar visible vía Bluetooth. La imagen de estado pasará a color morado.

Ilustración 32. Servidor – Estado de un examen: no publicado.

3. Ver Resultados: se selecciona cuando el profesor quiere ver los resultados enviados por los alumnos. Los resultados se mostrarán según orden de llegada

Ilustración 33. Servidor – Pantalla de resultados de un examen.

4. Ayuda: seleccionando esta opción se muestra una pantalla de ayuda.

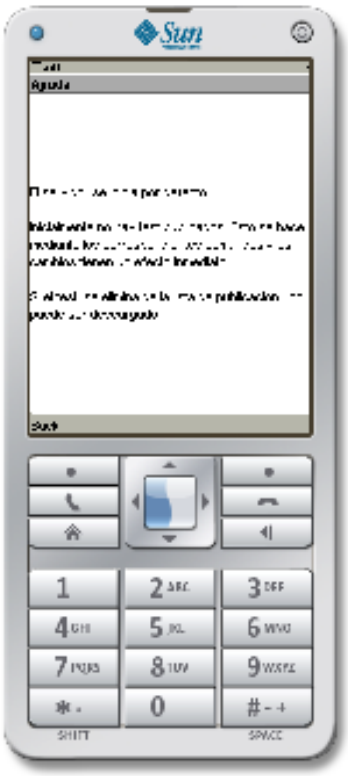


Ilustración 34. Servidor – Pantalla de ayuda.

Las clases pertenecientes a este módulo son las siguientes:

Clase TestServer:

Es la clase MIDlet del módulo, por ello contiene todos los métodos relativos al ciclo de vida de un MIDlet, además de los métodos específicos de este módulo.

- `completeInitialization(boolean isBTReady)`: se llama después de la inicialización Bluetooth. Si se ha inicializado bien, mostrará la lista de los exámenes que pueden ser publicados, sino muestra un mensaje de error.
- `getTestFileName(String teName)`: este método se llama para obtener un examen a partir de su nombre.
- `setupTestList()`: prepara la lista de los test disponibles para presentarlos al usuario.
- `getNotas(String test)`: devuelve el objeto Notas.

- `crearListaNotas(String test)`:muestra una pantalla al usuario (profesor) con los resultados asociados a ese test de los alumnos, ordenados según orden de llegada.
- `borrarNotas(String test)`: borra los resultados de los alumnos asociados a ese test.
- `show()`: método para mostrar el menú principal.
- `esperando(int milisegundos)`: Método que para la ejecución del hilo un instante para poder visualizar ciertos elementos o cambios.

Ilustración 35. Clase TestServer.

Clase BluetoothServer:

Esta es la clase encargada de gestionar toda la comunicación Bluetooth. Acepta y abre conexiones para procesar las peticiones de exámenes por parte de los clientes. Manda los exámenes y recibe los resultados de los alumnos.

Contiene la clase interna **`ClientProcessor`** que se encarga de gestionar una a una las peticiones.

Cada una de ellas es un hilo diferente. A medida que llegan nuevas peticiones al hilo.

Los métodos más destacables de la clase `BluetoothServer` son los siguientes:

- `changeTestInfo(String name, boolean isPublished)`: Actualiza el service record con la información publicada acerca de la disponibilidad de los test. Este método se invoca después de que el llamante haya chequeado que la acción se debe hacer.

- `processConnection(StreamConnection conn)`: Lee el nombre del test de la conexión especificada (petición de test) y lo manda, siempre que haya sido publicado, a través de la conexión. Después cierra la conexión.
- `sendTestData(Byte[] tData, StreamConnection conn)`: este método se encarga de enviar los bytes de datos a través de la conexión especificada.
- `readTestName(StreamConnection conn)`: lee el nombre del test a través de la conexión especificada.
- `getTestData(String testName)`: lee los datos del test especificado de la clase Examen y los pasa a array de bytes

Los métodos más destacables de la clase `ClientProcessor` son:

- `addConnection (StreamConnection con)`: añade la nueva conexión a la cola y notifica al hilo.
- `destroy(boolean needJoin)`: cierra la conexión y notifica al hilo.
- `gestionarNotas(String resultado)`: este método se encarga de guardar en un RecodStore los resultados enviados por los alumnos en el formato adecuado.

Ilustración 36. Clase BluetoothServer y ClientProcessor.

Clase Examen:

Esta clase contiene los atributos y métodos necesarios para crear el examen tipo test que será publicado.

Cada examen constará de diez preguntas con tres posibles respuestas, de las cuales sólo una será correcta.

Esta clase contiene como atributos una batería de preguntas con sus correspondientes posibles respuestas, de la combinación de estas preguntas se obtendrán los distintos exámenes.

Los métodos a destacar de esta clase son:

- `getPregunta(int p)`: devuelve en un String la pregunta con ese identificador.
- `getRespuestas(int pregunta)`: devuelve en un array de Strings las posibles respuestas a esa pregunta.
- `getExamen(String numero)`: devuelve en un array de bytes el examen identificado por el parámetro que recibe el método.
- `concatena(byte[] array, byte[] array2)`: método para concatenar dos arrays.
- `getSoluciones(String examen)`: método que dado el nombre de un examen, devuelve las soluciones a sus preguntas.
- `getSolucion(int pregunta)`: método que dada una pregunta, devuelve su solución.

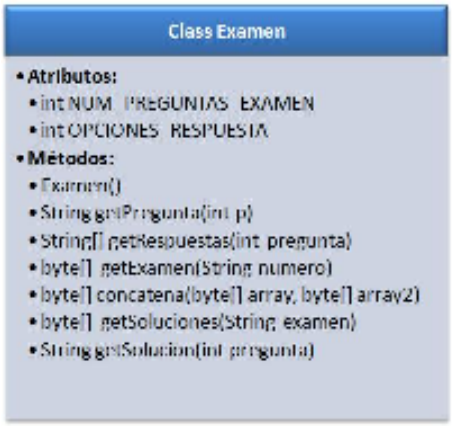


Ilustración 37. Clase Examen.

Clase Notas:

Esta clase es la encargada de almacenara los resultados de los alumnos. Para ello utiliza el paquete RMS – Record Management System que permite a las aplicaciones almacenar datos de forma persistente.

Cada examen tendrá un `RecordStore`, que a su vez tendrá una colección de `records` que serán el identificador del alumno junto con el número de aciertos que ha tenido en el examen.

Los métodos de esta clase son:

- `openNotas(String examen)`: abre el `RecordStore` correspondiente al examen que se le pasa por parámetro.
- `closeNotas()`: cierra el `RecordStore`.
- `deleteNotas(int pos)`: borra un record determinado del `RecordStore`.
- `addNotas(String alumno, int aciertos)`: añade un nuevo `record` al `RecordStore`.
- `devolverNota(int id)`: método que devuelve una nota.
- `getIdsNota()`: devuelve en un array todos los identificadores de los `records` que contiene ese `RecordStore`.
- `obtenerNumeroRecords()`: devuelve el número de records que contiene ese `RecordStore`

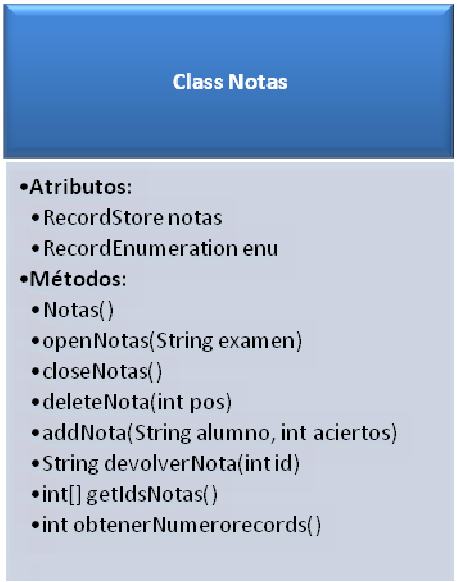
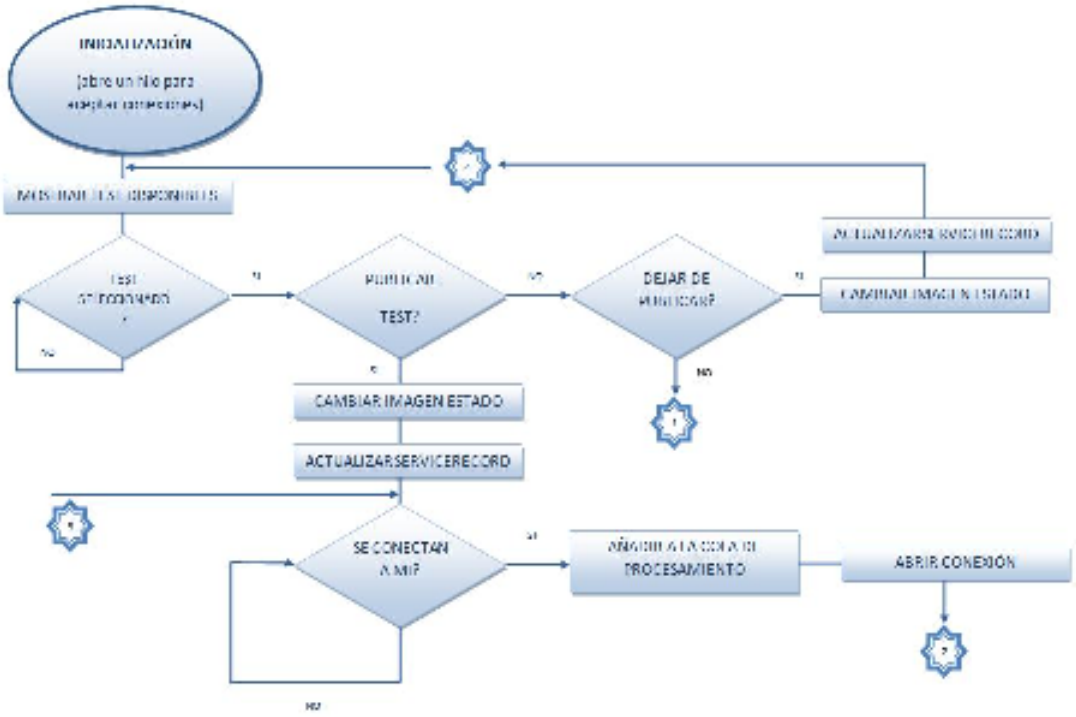
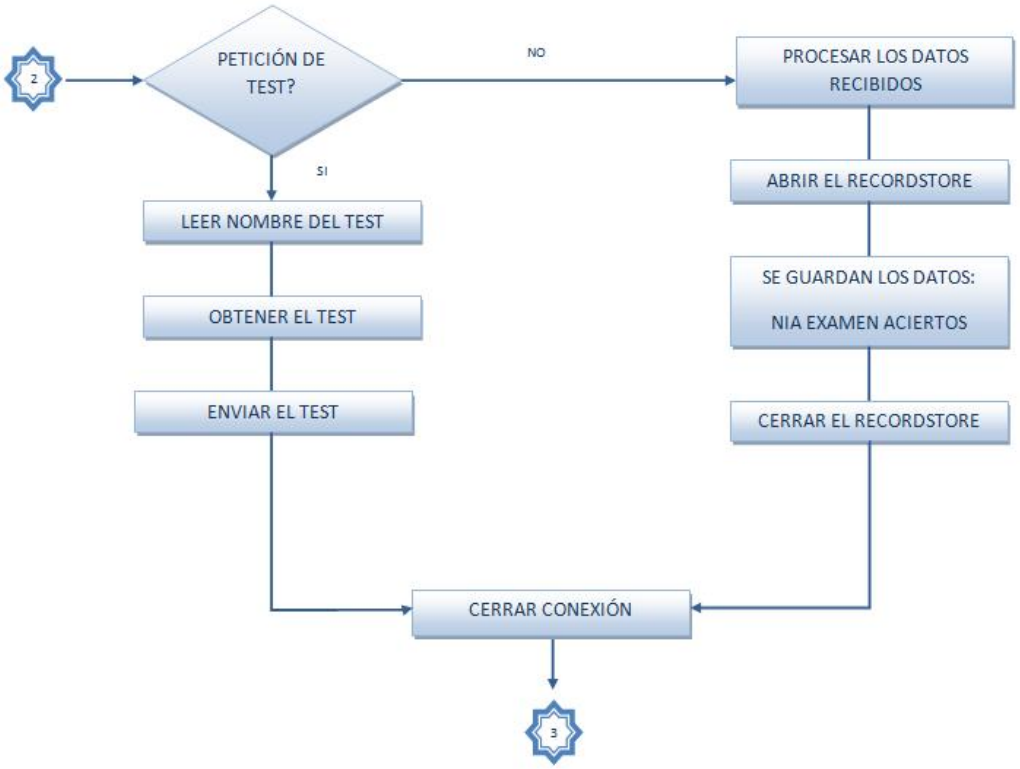
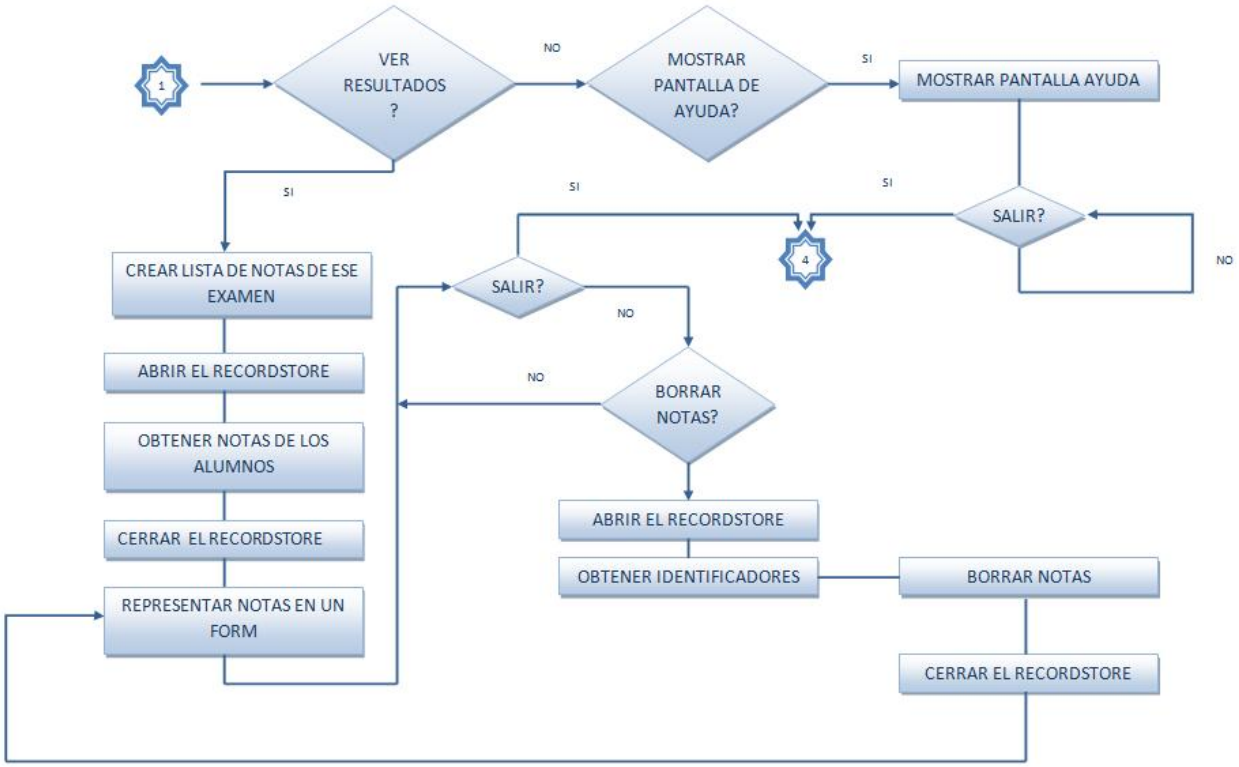


Ilustración 38. Clase Notas.

El siguiente diagrama, explica el funcionamiento de este módulo.







BLUETOOTHCLIENT

Es el módulo cliente, el que ejecutarán los alumnos cuando tengan que descargarse el examen tipo test que publique, vía Bluetooth, el profesor. Este módulo se encarga de:

- descubrir nuevos dispositivos
- descubrir nuevos servicios
- establecer la conexión
- comunicarse

Una vez arrancada la aplicación cliente, aparece la siguiente pantalla:

Ilustración 39. Cliente – Pantalla de búsqueda de test.

Seleccionando el botón “Back” aparecerá la siguiente pantalla:

Ilustración 40. Cliente – Pantalla para salir de la aplicación.

En la que encontramos el botón de “Exit” para salir de la aplicación, y el botón de “Ok” que nos llevaría otra vez a la pantalla de búsqueda.

Si por el contrario se pincha sobre el botón de “Find” aparecerá la siguiente pantalla:

Ilustración 41. Cliente – Pantalla que aparece durante la búsqueda de test.

Si el dispositivo servidor (móvil del profesor) no hubiera arrancado , y por lo tanto no estuviera visible mediante Bluetooth, aparecería la siguiente pantalla:

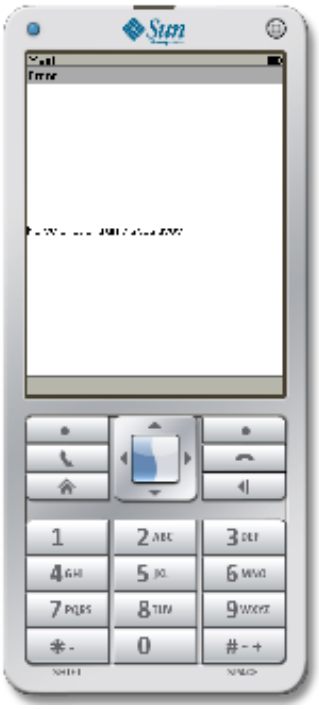


Ilustración 42. Cliente – Pantalla que aparece cuando no se han encontrado test.

En el caso de que si hubiera arrancado y hubiera publicado un examen aparecería:

Ilustración 43. Cliente – Pantalla con los test encontrados.

Seleccionando “Back” se vuelve a la pantalla de búsqueda de test. Si se selecciona “Load” preguntará al usuario si autoriza la conexión Bluetooth:

Ilustración 44. Cliente – Pantalla para permitir la conexión Bluetooth.

Y pinchando “Yes” automáticamente cargará el test descargado:

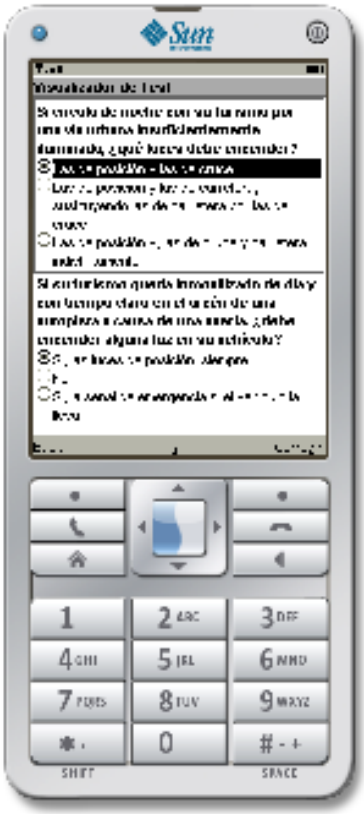


Ilustración 45. Cliente – Pantalla visualizando un test.

Seleccionando “Back” se vuelve a la pantalla de Test Encontrados. Una vez solucionado el test, se selecciona “Corregir” apareciendo una pantalla donde se especifican los aciertos que se han tenido, y las preguntas que se han fallado (de esta forma los alumnos pueden saber qué conceptos tienen que estudiar mejor) como la siguiente:

Ilustración 46. Cliente – Pantalla que muestra los resultados obtenidos.

Una vez comprobadas las notas, se envían los resultados al móvil servidor. Esto se hace pulsando sobre el botón “Enviar”, apareciendo una pantalla para que el alumno introduzca su número identificativo, el NIA (Numero de Identificación del Alumno):

Ilustración 47. Cliente – Pantalla para introducir los datos del alumno.

Una vez hecho esto, el alumno pulsará el botón “Mandar” , le aparecerá una pantalla avisándole de que va a abrir una conexión Bluetooth y pidiendo su autorización para abrirla:

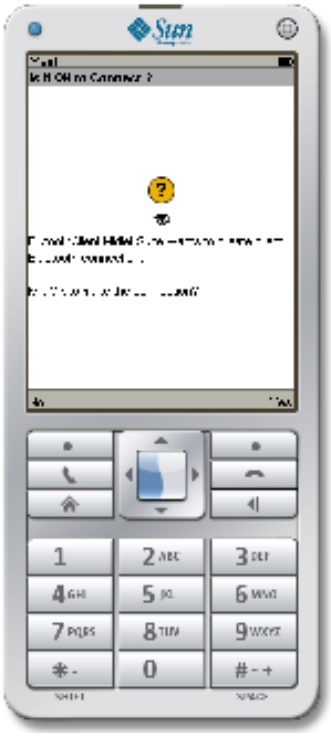


Ilustración 48. Cliente – Pantalla para permitir la conexión Bluetooth.

Pulsando sobre el botón de "Yes" los resultados se mandarán al móvil servidor, y volverá a la pantalla del test por si quiere repetirlo o simplemente comprobar qué preguntas había fallado.

Las clases pertenecientes a este módulo son las siguientes:

Clase TestClient:

Es el MIDlet de este módulo. Los métodos más importantes son los siguientes:

- `completeInitialization(boolena isBTReady)`: Este método es llamado por `BluetoothClient` después de la inicialización Bluetooth y cuando la siguiente pantalla está lista para aparecer.
- `informSearchError(String resMsg)`: se encarga de informar de los posibles errores ocurridos durante la búsqueda de test.
- `informLoadError(String resMsg)`: se encarga de informar de los posibles errores ocurridos durante la carga del test.
- `showTest()`: muestra el test descargado por pantalla. Internamente llama a `representarTest(byte[] dat, Form f)`.
- `showTestNames(Hashtable base)`: muestra los nombres de los test que se han encontrado vía Bluetooth.

- `representarTest(byte[] dat, Form f)`: representa el test descargado en un Form.
- `getTestSolicitado()`:método que devuelve el test solicitado, guardado previamente.
- `getTestSolicitado(byte[] t)`: método que guarda en un atributo el test solicitado (descargado).
- `aciertos()`:método que devuelve un array con las respuestas acertadas.
- `numAciertos(int[] respuestasAcertadas)`: método que devuelve el numero de aciertos que ha tenido el alumno en el test realizado.
- `getRespuestasAlumno()`: método que obtiene las respuestas que ha dado el alumno en las preguntas del test.
- `esperando(int milisegundos)`: método que para la ejecución del hilo un instante para poder visualizar ciertos elementos o cambios.

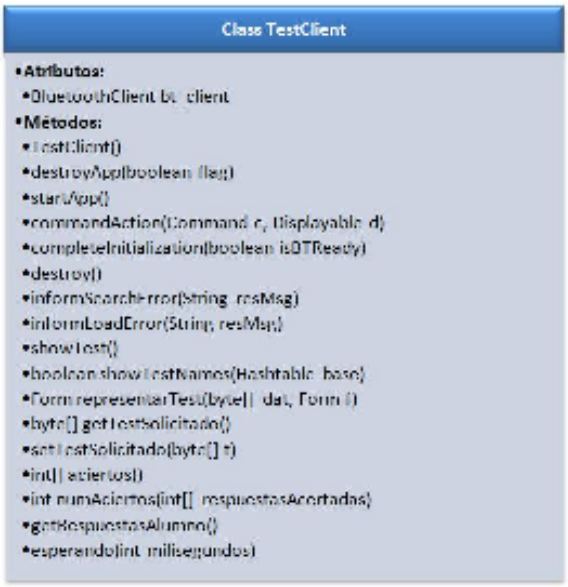


Ilustración 49. Clase TestClient.

Clase BluetoothClient:

Esta es la clase que se encarga de gestionar toda la comunicación Bluetooth, deberá realizar las siguientes operaciones para comunicarse con el servidor Bluetooth:

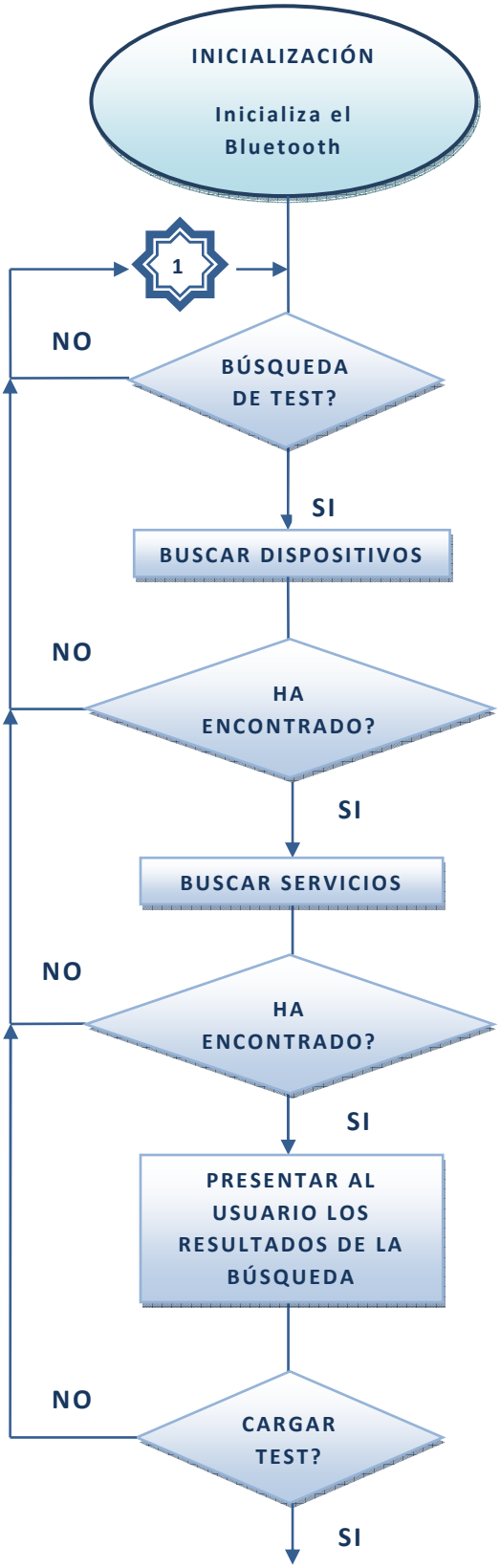
- Búsqueda de dispositivos
- Búsqueda de servicios
- Establecimiento de la conexión
- Comunicación

Los métodos más destacables de esta clase son:

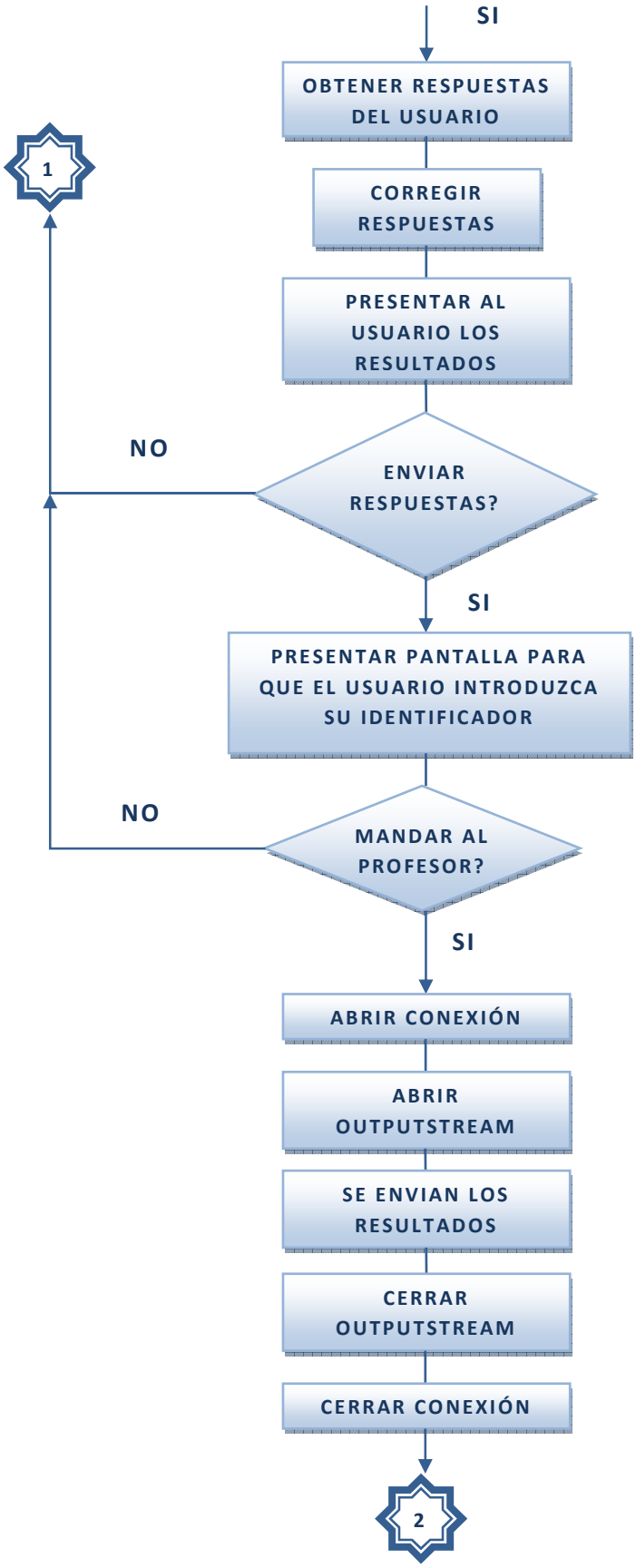
- `deviceDiscovered(RemoteDevice btDevice,DeviceClass cod)`: se invoca por el sistema cuando un dispositivo remoto es encontrado, se mantienen en memoria los dispositivos encontrados.
- `inquiryCompleted(int discType)`: Se invoca por el sistema cuando se ha encontrado el dispositivo. Únicamente guarda el discType y se procesa en otro hilo. El discType será INQUIRY_COMPLETED si la búsqueda se ha terminado normalmente o INQUIRY_TERMINATED si la búsqueda fuera cancelada por una llamada a `DiscoveryAgent.cancelInquiry (DiscoveryListener)`. El discType será INQUIRY_ERROR si un error ocurriera procesando la búsqueda que hace la búsqueda terminarse de modo anormal.
- `servicesDiscovered (int transID, ServiceRecord[] servRecord)`: guarda los servicios encontrados.
- `serviceSearchCompleted(int transID, int respCode)`: Llamado cuando una búsqueda de servicio es completada o fue terminado debido a un error. Valores válidos del código de respuesta pasado como argumento son:
  - `SERVICE_SEARCH_COMPLETED`
  - `SERVICE_SEARCH_TERMINATED`
  - `SERVICE_SEARCH_ERROR`
  - `SERVICE_SEARCH_NO_RECORDS`
  - `SERVICE_SEARCH_DEVICE_NOT_REACHABLE`
- `requestSearch()`:Establece la solicitud de búsqueda de los dispositivos y servicios.
- `cancelSearch()`:cancela la búsqueda de dispositivos/servicios.
- `requestLoad(String name)`: Establece la petición de carga del test específico.
- `cancelLoad()`: cancela la descarga del test.
- `destroy()`: destruye el trabajo Bluetooth.
- `processTestSearchDownload()`:Procesa la búsqueda o descarga de test hasta que el componente está cerrado o ha habido un error en el sistema.
- `searchDevices()`: busca los dispositivos Bluetooth.
- `searchServices()`: busca los servicios Bluetooth.
- `presentUserSearchResults()`:Obtiene la colección de los títulos de los test (nombres) de los servicios, prepara una HashTable para que coincida el nombre del test con una lista de servicios, se presentan los nombres de los test al usuario final.
- `loadTest(String name)`: carga el test seleccionado.
- `enviarRespuesta(String respuesta)`: envía las respuestas (soluciones) del test al servidor.

Ilustración 50. Clase BluetoothClient.

El siguiente diagrama, explica el funcionamiento de este módulo.




ABRIR CONEXIÓN
ABRIR OUTPUTSTREAM
ENVIAR NOMBRE DE TEST QUE SE QUIERE DESCARGAR
CERRAR OUTPUTSTREAM
ABRIR INPUTSTREAM
LEER LONGITUD DE DATOS
LEER Y GUARDAR DATOS
CERRAR INPUTSTREAM
CERRAR CONEXIÓN
GUARDAR TEST DESCARGADO
MOSTRAR TEST AL USUARIO
CORREGIR TEST?
NO
SI
2


SI
OBTENER RESPUESTAS DEL USUARIO
1
CORREGIR RESPUESTAS
PRESENTAR AL USUARIO LOS RESULTADOS
NO
ENVIAR RESPUESTAS?
SI
PRESENTAR PANTALLA PARA QUE EL USUARIO INTRODUZCA SU IDENTIFICADOR
NO
MANDAR AL PROFESOR?
SI
ABRIR CONEXIÓN
ABRIR OUTPUTSTREAM
SE ENVIAN LOS RESULTADOS
CERRAR OUTPUTSTREAM
CERRAR CONEXIÓN
2

4.3. PRUEBAS

Para comprobar el correcto funcionamiento de la aplicación, se le ha sometido a una serie de pruebas, con el fin de detectar posibles fallos.

Situación 1: El servidor arranca y publica un test. El cliente arranca y detecta este test. Antes de que el cliente seleccione la opción de descargar, el servidor termina.

En este caso, aparecerá en la pantalla del cliente, un mensaje informando de que no se puede cargar.

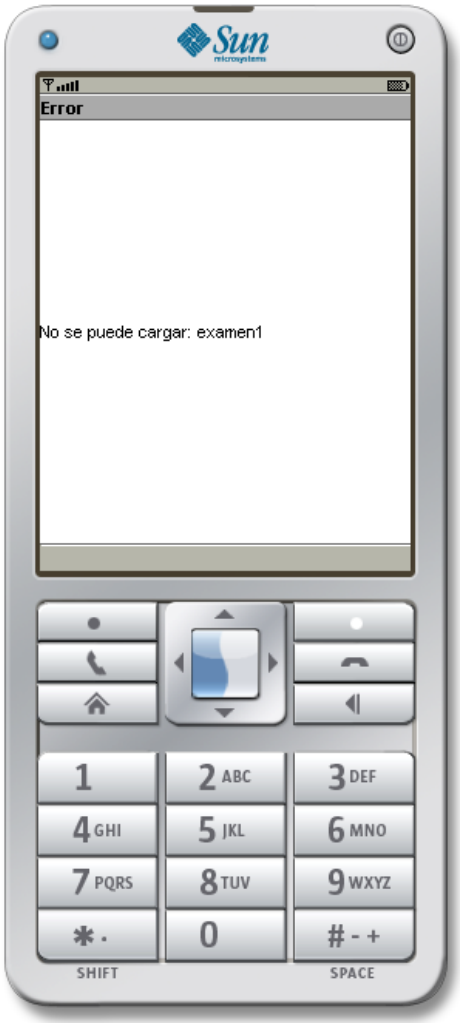


Ilustración 51. Pruebas: Situación 1

Situación 2: El servidor arranca y publica un test. El cliente arranca y detecta este test, lo descarga y el usuario responde las preguntas y completa los datos de envío. Antes de que el cliente seleccione la opción de enviar los resultados al servidor, el servidor termina.

Aparecerá en la pantalla del cliente, un mensaje informando de que no se han podido enviar los resultados.

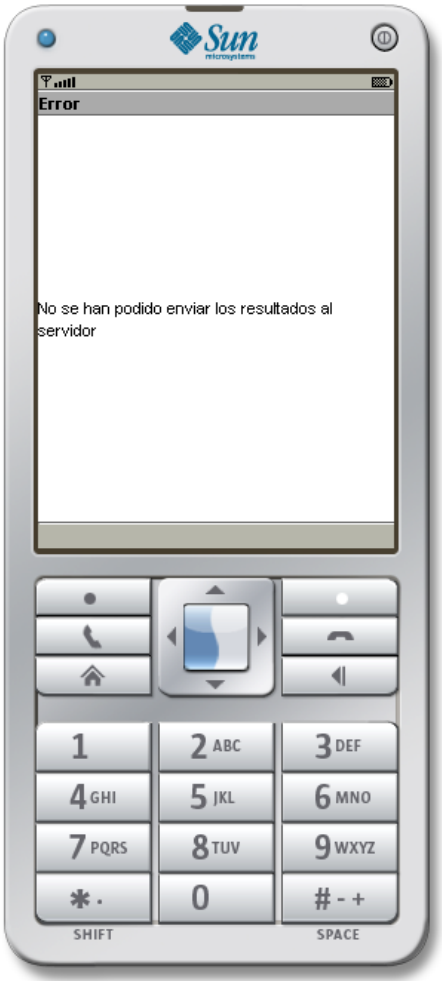


Ilustración 52. Pruebas: Situación 2

Situación 3: A la hora de enviar los resultados al servidor, si el usuario, cuando la aplicación le pide que introduzca el NIA, teclea caracteres no numéricos, comprobará que no aparecen, ya que estos no son permitidos.

5. CONCLUSIONES

Lo que se buscaba con este proyecto fin de carrera es la aplicación de las nuevas tecnologías al sistema educativo para la mejora de éste.

Instalando esta aplicación en los terminales de los profesores y alumnos y, utilizándola con regularidad durante el curso escolar, se puede conseguir que los alumnos aumenten su grado de interés hacia la asignatura, adquieran nuevos conocimientos de una forma más amena y que la asimilación de los mismos se haga de una forma menos tediosa.

Por otra parte, ofrece al profesor la posibilidad de conocer de una forma regular el nivel que tienen los alumnos en la asignatura, pudiendo profundizar más en los temas que los alumnos encuentren más difíciles, esto lo conoce gracias a que guarda los resultados de los alumnos para cada test, siendo los test con peores resultados los más difíciles para los alumnos, lo que conlleva una mejora en la enseñanza.

A la vez que aumenta el grado de interés en la asignatura por parte de los alumnos, aumentará el grado de motivación del profesor, al ver mayor participación e interés en sus clases.

En el futuro, tal y como estamos viendo, la educación tenderá a utilizar más las nuevas como la implantación de pizarras multimedia en los centros educativos, el que los alumnos posean su propio ordenador portátil con el que trabajar en aulas informatizadas con acceso a internet… Pero este es un proceso lento, con un alto coste económico, por lo que hasta ese momento, esta aplicación Servidor Bluetooth de exámenes es una buena herramienta a implantar ya que el impacto económico sería muy bajo debido a que la mayoría de los alumnos y profesores poseen un teléfono móvil en el que se podría instalar la aplicación, además de que estos terminales son más económicos que los ordenadores portátiles.

5.1. POSIBLES MEJORAS DE LA APLICACIÓN

Como se ha visto, esta aplicación ofrece muchas posibilidades para el aprendizaje, sin embargo le podrían añadir nuevas funcionalidades como son:

- Se podría realizar una aplicación para el módulo servidor que permitiera al profesor, desde su propio ordenador, añadir nuevos exámenes. Esta aplicación podría ofrecer un interfaz gráfico en el que se le fuera pidiendo el enunciado de cada cuestión, sus posibles respuestas y la respuesta correcta. También podría ofrecer la posibilidad de que el profesor elija una combinación de cuestiones de las ya existentes, para crear nuevos exámenes.
- Siguiendo con la idea de un programa instalado en el ordenador del profesor, se podría hacer un módulo que guardara los datos relativos a cada examen (resultados obtenidos por los alumnos) y sacara estadísticas y gráficas importantes para conocer el desarrollo de la asignatura, como pueden ser:
    - La evolución en el número de aciertos de cada alumno.
    - Diferencias en el número de aciertos en un examen según el año académico, podría sacar gráficas en las que se viera la evolución. De esta forma si un año se producen cambios en la forma de impartir un curso, se podría ver si con esta nueva forma los alumnos entienden mejor o no los conceptos.
    - Participación del alumnado según el examen. Esto daría una información de qué tema es el más difícil para los alumnos, ya que muchas veces, al no ser obligatoria la asistencia, los alumnos no van a clase si saben que van a tener un examen de un tema complicado.
- En la aplicación del profesor, se podría imponer que antes de guardar los resultados de un alumno, se compruebe que es un NIA válido, que sea de un alumno matriculado en esa asignatura, y que a su vez este NIA no haya llegado con anterioridad (en la misma sesión) a la aplicación servidora para este mismo examen.
- Se podría imponer que para descargar un examen, se necesite enviar una contraseña válida, y diferente para cada examen.

6. BIBLIOGRAFÍA

Libros:

- Álvarez García, A. y Morales Grela, J.A. (2002). *J2ME* . Ed. Anaya Multimedia - Anaya Interactiva. ISBN: 8441513961.
- Barbagallo, Ralph. (2004). *Wireless game development in Java with MIDP 2.0*. Ed. Wordware Publishing, Inc. ISBN: 1-55622-998-4
- Knudsen, Jonathan. (2001). *Java™: Developing with Java 2, Micro Edition.* Ed. Apress™. ISBN: 1-893115-50-X
- Kumar, C. Bala, Kline, Paul J. y Thompson, Timothy J. (2004). *Bluetooth application programming with the Java APIs*. Ed. Morgan Kaufmann. ISBN: 1558609342.

Páginas Web:

- Bluetooth SIG (2009) http://spanish.bluetooth.com/Bluetooth/Technology/Works/
- García Castellano, F.J. (2006) *Tutorial de MIDP* http://leo.ugr.es/J2ME/MIDP/ Dpto. de CCIA, Universidad de Granada.
- Sun Microsystems, Inc. (2006) *API JSR82* http://java.sun.com/javame/reference/apis/jsr082/
- Sun Microsystems, Inc. y Motorola, Inc. (2006) *API MID Profile* http://java.sun.com/javame/reference/apis/jsr118/

Otros documentos:

- Caballo Gil, Manuel (2007) *Manual de Eclipse para el desarrollo de aplicaciones Java ME*. Universidad Carlos III de Madrid.
- Campo, Celeste. (2005) *Apuntes de Software de Comunicaciones. 3º I.T.T: Telemática*. Universidad Carlos III de Madrid.
- Gimeno Brieba, Alberto *Programación de dispositivos Bluetooth a través de Java.*
- Gimeno Brieba, Alberto (2004) *JSR-82: Bluetooth desde Java™.*
- Sun Microsystems, Inc. (2001, 2002) *JavaTM APIs for BluetoothTM Wireless Technology (JSR 82) Specification Version 1.1.1 JavaTM 2 Platform, Micro Edition*.

Octubre de 2009

7. ANEXO I : TABLA DE SIGLAS

SIGLAS	SIGNIFICADO
ACL	Asynchronous Connectionless
BCC	Bluetooth Control Center
CLDC	Connected Limited Device Configuration
GAP	Generic Access Profile
GIAC	General/Unlimited Inquiry Access Code
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HCI	Host Controller Interface
INE	Instituto Nacional de Estadística
IR	Interfaz Radio
ISM frequency	Industrial-Scientific-Medical frequency
ISO	Internacional Standard Organization
J2ME	Java 2 Micro Edition
JABWT	Java APIs for Bluetooth wireless technology
JAM	Java Application Management
L2CAP	Logical Link Control and Adaptation Protocol
LIAC	Limited Dedicated Inquiry Access Code
LM	Link Managers
LMP	Link Manager Protocol
MIDP	Mobile Information Device Profile
MMS	Multimedia Message Service
NIA	Número de Identificación del Alumno
OBEX	Object Exchange
OSI	Open System Interconnect
RFCOMM	Radio Frequency Communication
SCO	Synchronous Connection-Oriented
SDAP	Service Discovery Application Profile
SDDB	Service Discovery Database
SDP	Service Discovery Protocol
SIG	Special Interest Group
SMS	Short Message Service
SMSC	Short Message Service Centre
SPP	Serial Port Profile
TCS Binary	*Bluetooth* Telephony Control protocol Specification Binary
TDM	Time Division Multiplexing
USB	Universal Serial Bus
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WTK	Wireless Toolkit

Tabla 2. SIGLAS

8. ANEXO II: CREACIÓN DE UN PROYECTO J2ME EN ECLIPSE

A continuación se explica cómo crear un proyecto J2ME en Eclipse.

Los puntos que se desarrollan a continuación se ha extraído del documento ***"Manual de Eclipse para el desarrollo de aplicaciones Java ME"*** de **Manuel Caballo Gil** (Diciembre 2007) de la Universidad Carlos III de Madrid.

Se selecciona el menú "*File/New/Other…*" y dentro de las opciones J2ME Midlet Suite:

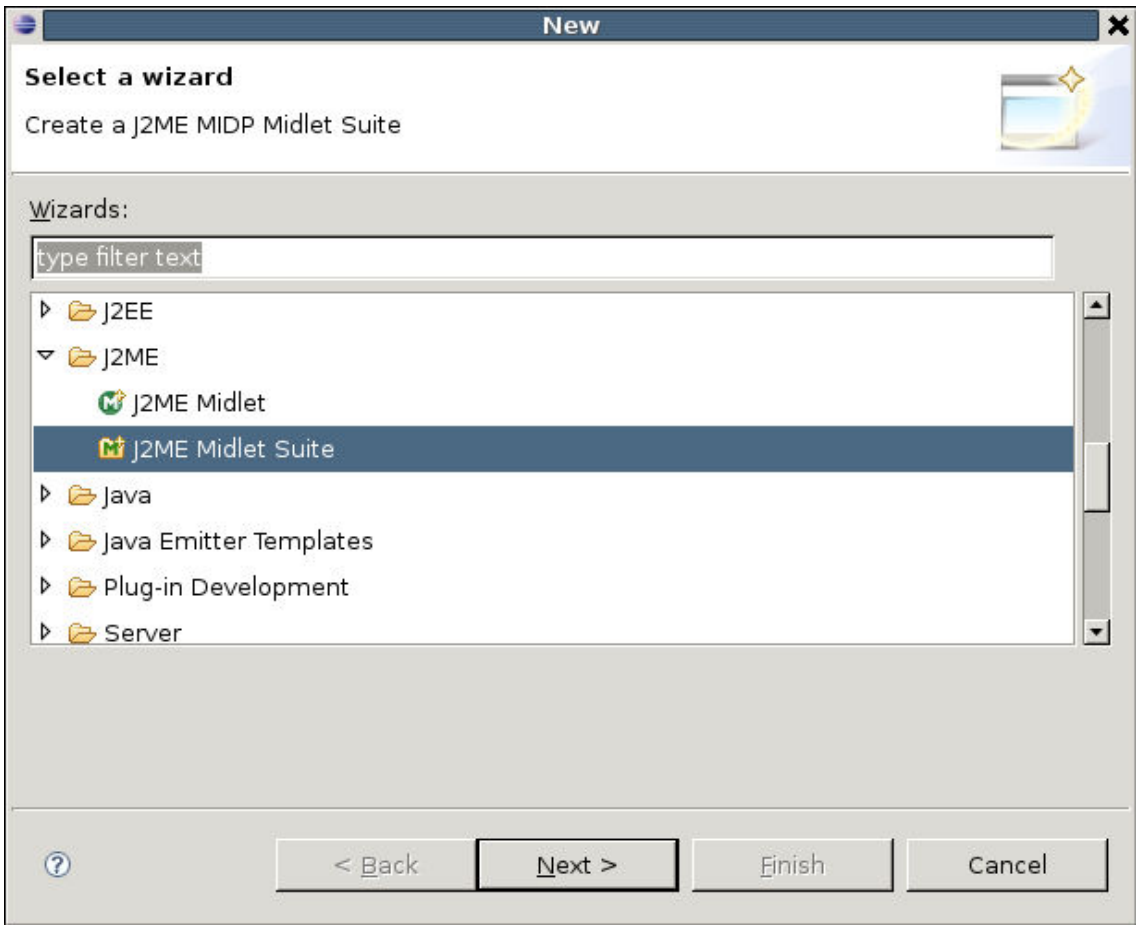


Ilustración 53. Creando un proyecto J2ME en Eclipse.

Daremos a siguiente y seleccionaremos el nombre de nuestro proyecto:

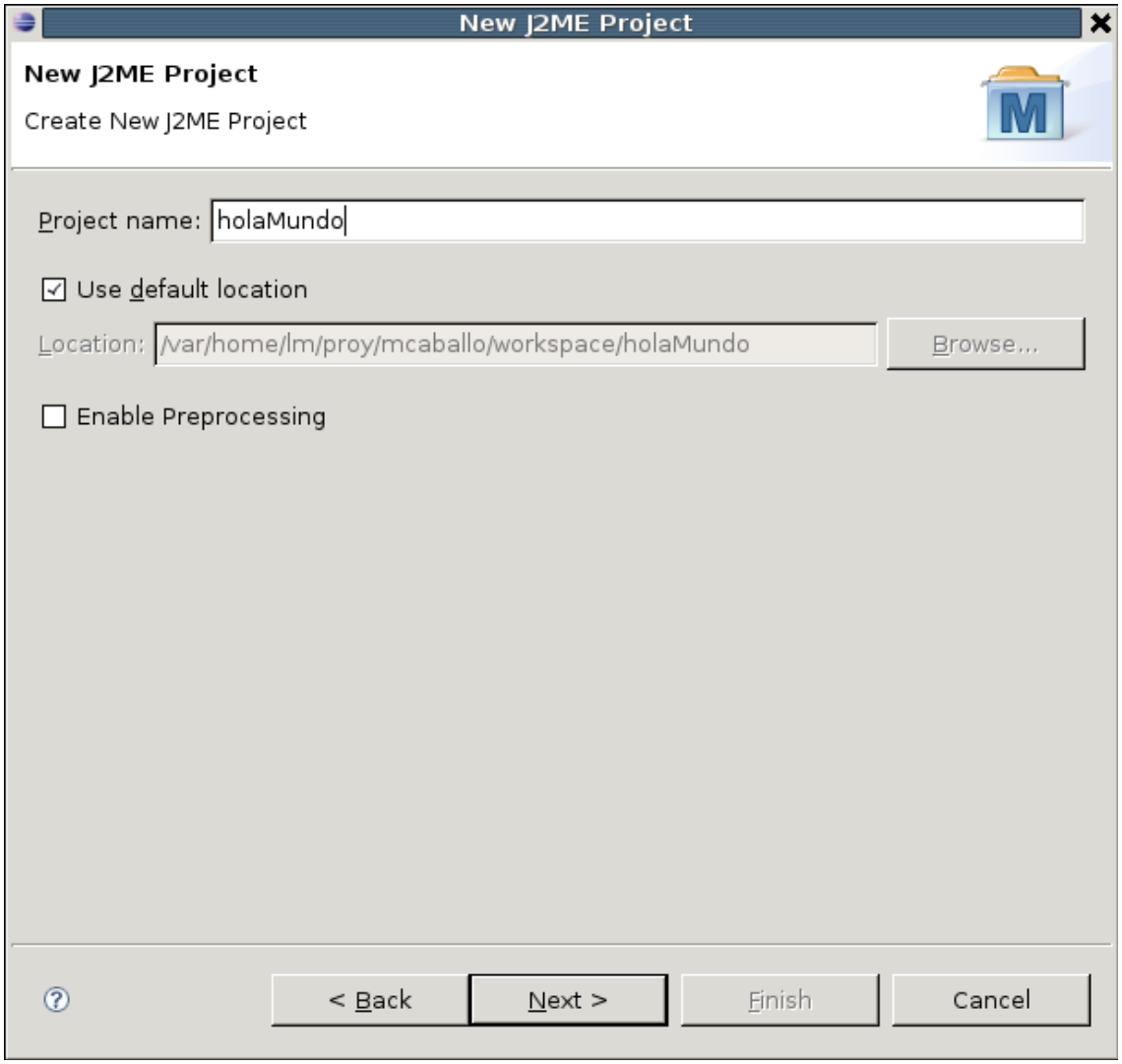


Ilustración 54. Asignando un nombre al proyecto.

En principio no nos interesan las opciones de preprocesamiento por lo que no seleccionaremos la casilla correspondiente. En caso de necesitar estas funcionalidades las podremos habilitar más adelante.

En la pantalla siguiente no deberemos cambiar nada, ya que se nos muestran las opciones de configuración referentes al WTK y al dispositivo en el que se emulará al MIDlet, y que ya habremos configurado previamente durante la instalación de EclipseME.

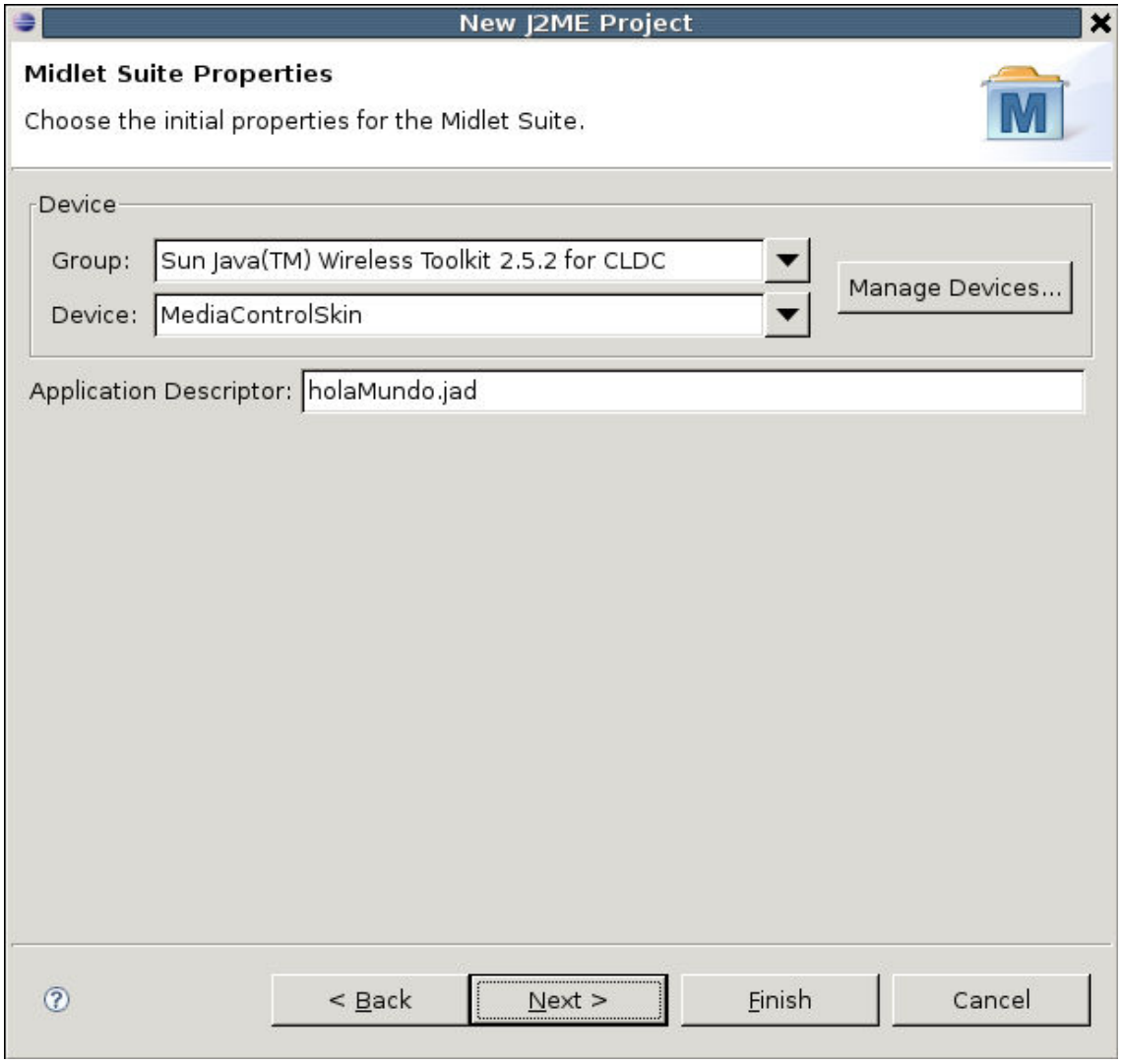


Ilustración 55. Propiedades del MIDlet.

Como podemos ver se puede personalizar el nombre del descriptor jad del MIDlet pero lo dejaremos por defecto, para evitar posibles confusiones. Pulsamos el botón *siguiente* y aparecerá una pantalla con los distintos directorios con los que contará nuestra aplicación, las librerías utilizadas y los proyectos relacionados con nuestra aplicación.

A no ser que deseemos alguna configuración especial de nuestro proyecto las
opciones seleccionadas por defecto en esta pantalla serán válidas suficientes para el correcto despliegue de la aplicación en el WTK.

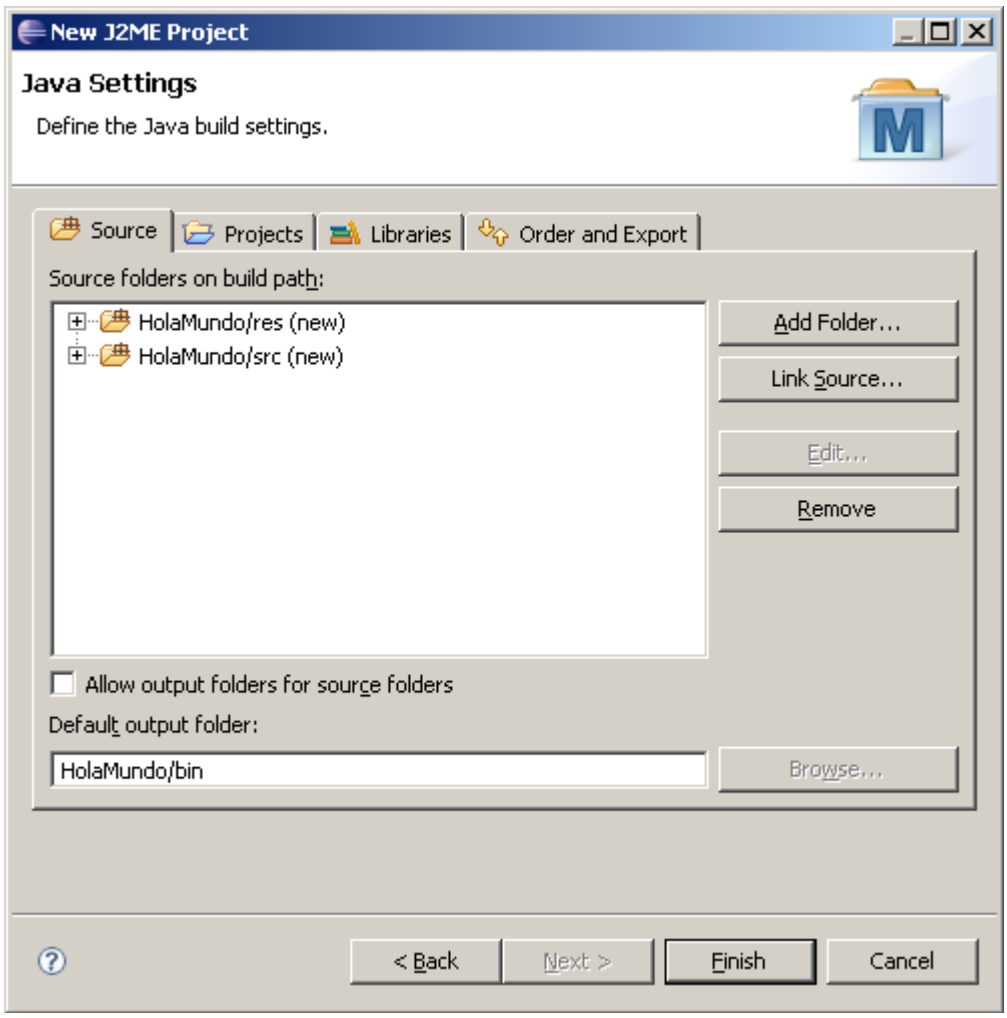


Ilustración 56. Configuración del proyecto.

Estamos solo a un paso de finalizar la creación de nuestro proyecto, si pulsamos el botón "*Finish*" se cerrará la ventana y volveremos al espacio de trabajo de nuestro IDE de Eclipse, mostrándose en la ventana de proyectos nuestro nuevo proyecto J2ME, tal y como se muestra en la siguiente captura:

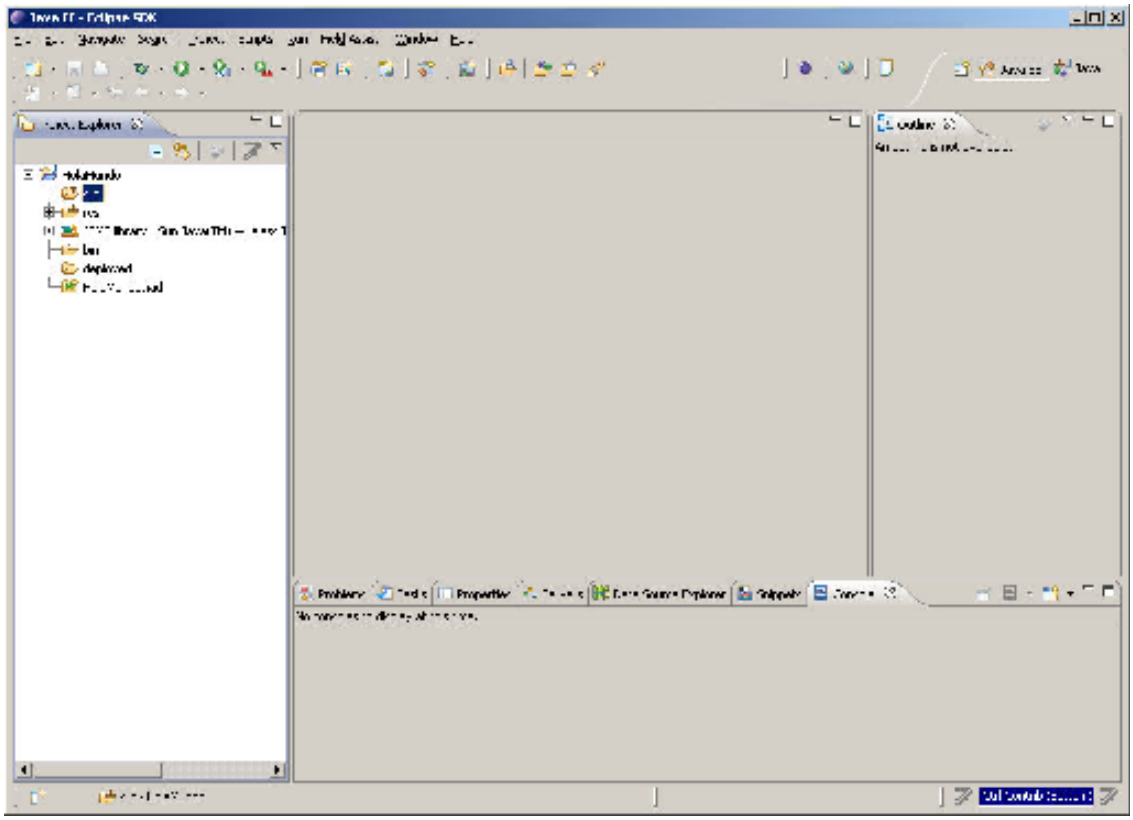


Ilustración 57. Espacio de trabajo de Eclipse.

Hasta ahora solo hemos creado el proyecto general, es decir, hemos especificado el conjunto de directorios y librerías que utilizará nuestro MIDlet, pero todavía no hemos explicado cómo se crearán las clases correspondientes al mismo. Este será el siguiente punto a abordar.

Si desplegamos el proyecto tal y como aparecía en la captura anterior podremos ver una carpeta con el nombre "*src*". Pulsamos con el botón derecho esta carpeta y añadimos un nuevo "*J2ME MIDlet*":

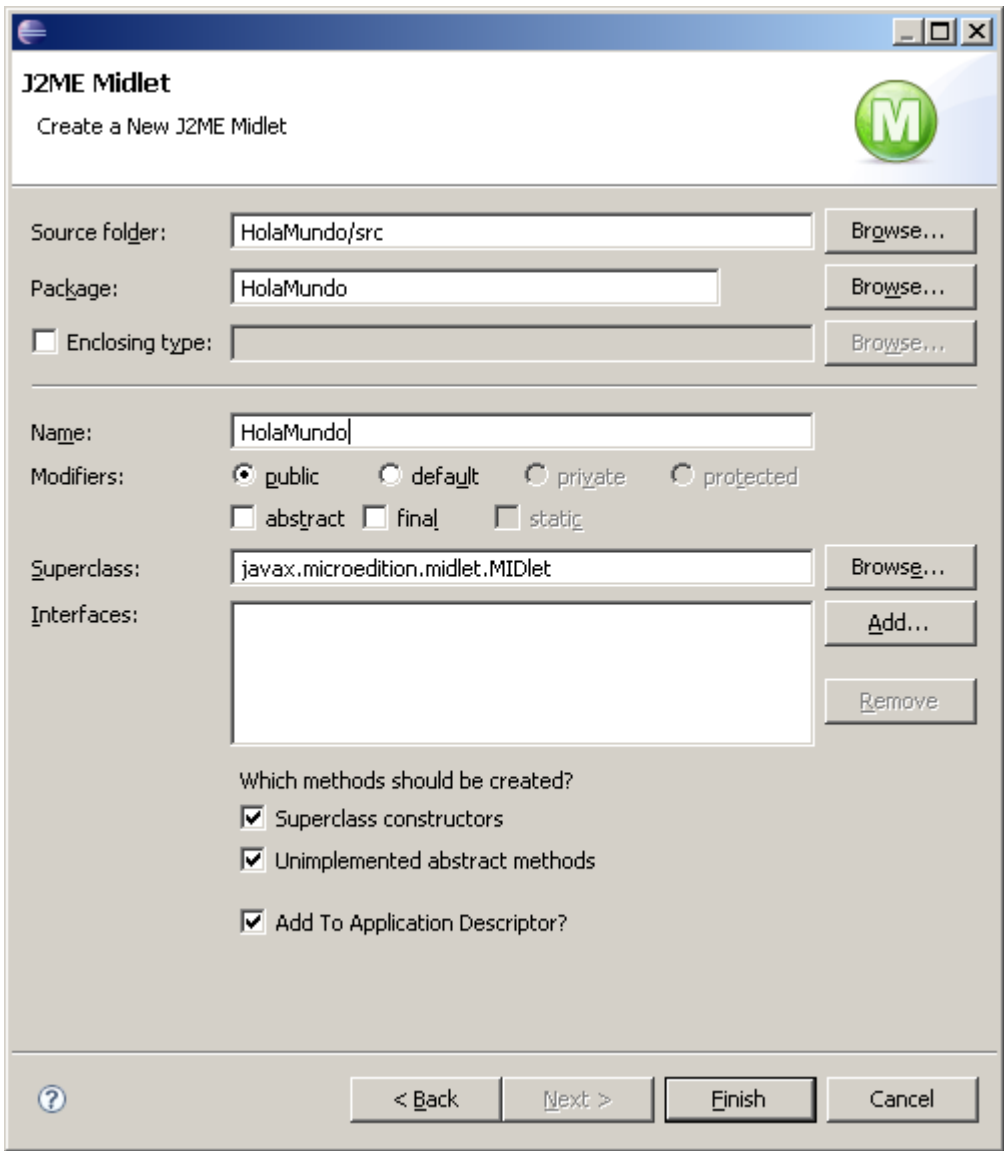


Ilustración 58. Creación de un MIDlet.

Si seguimos estos pasos se abrirá esta ventana, en la que podremos elegir aspectos como dónde se almacenarán las clases java del MIDlet (por defecto será *Nombre_Proyecto/src*), el nombre del paquete que contendrá todas las clases (muy útil si son varios los archivos .java) y el nombre de la clase principal del MIDlet, que en nuestro caso será "*HolaMundo.java*".

Pulsamos "*Finish*" y se abrirá automáticamente la clase que acabamos de crear dentro de la carpeta *src*. Tan solo tendremos que implementar los métodos correspondientes de esta clase (los heredados de la superclase `javax.microedition.midlet.MIDlet`).

En nuestro caso, la aplicación a crear es un MIDlet simple que mostrará por la

pantalla del terminal móvil el mensaje "*Hola Mundo*", por lo que tendremos que modificar la clase tal y como se muestra en la siguiente captura:

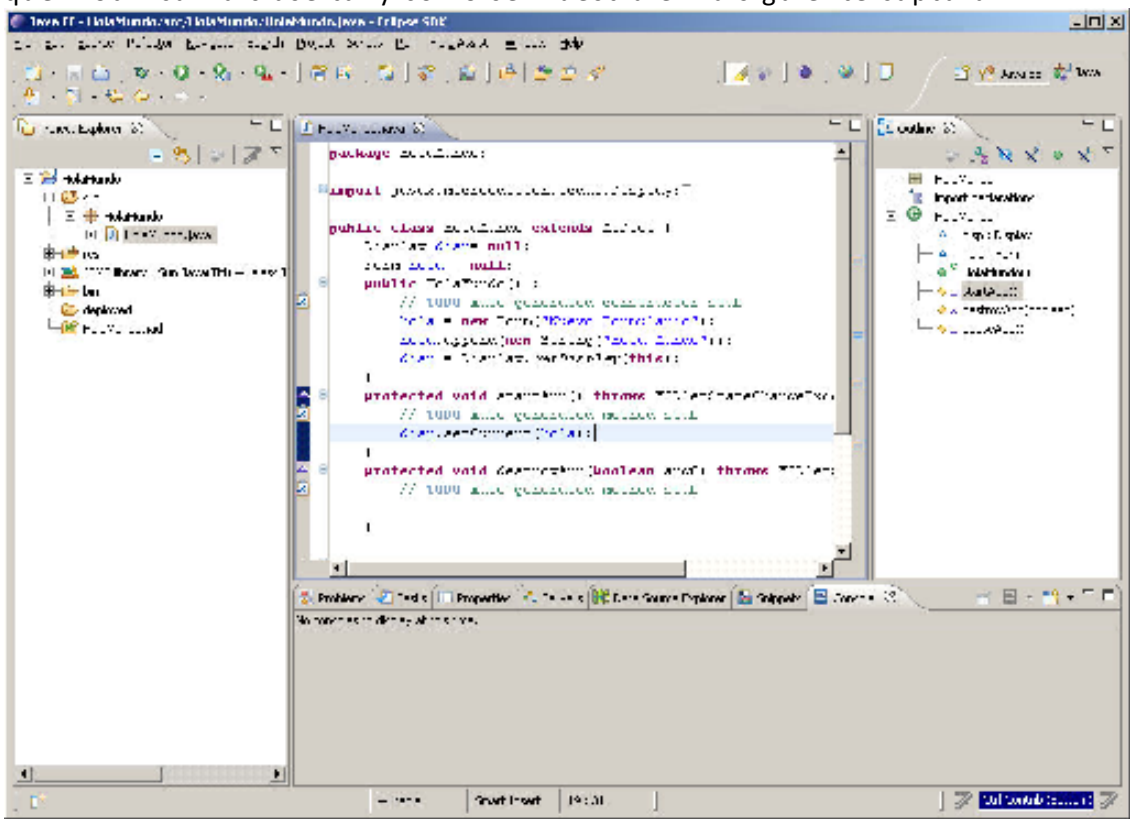


Ilustración 59. Clase HolaMundo

Una vez hayamos guardado todas las clases del proyecto y puesto todos los archivos adicionales en la carpeta *res* (en caso de que los haya) ya tenemos todo preparado para el despliegue de nuestra aplicación.

Para ejecutar el MIDlet en el WTK deberemos pinchar con el botón derecho sobre la clase principal del proyecto, en nuestro caso `HolaMundo.java` y seleccionaremos la opción "*Run As/ Emulated J2ME MIDlet*", cuya ubicación podemos apreciar en la siguiente captura:

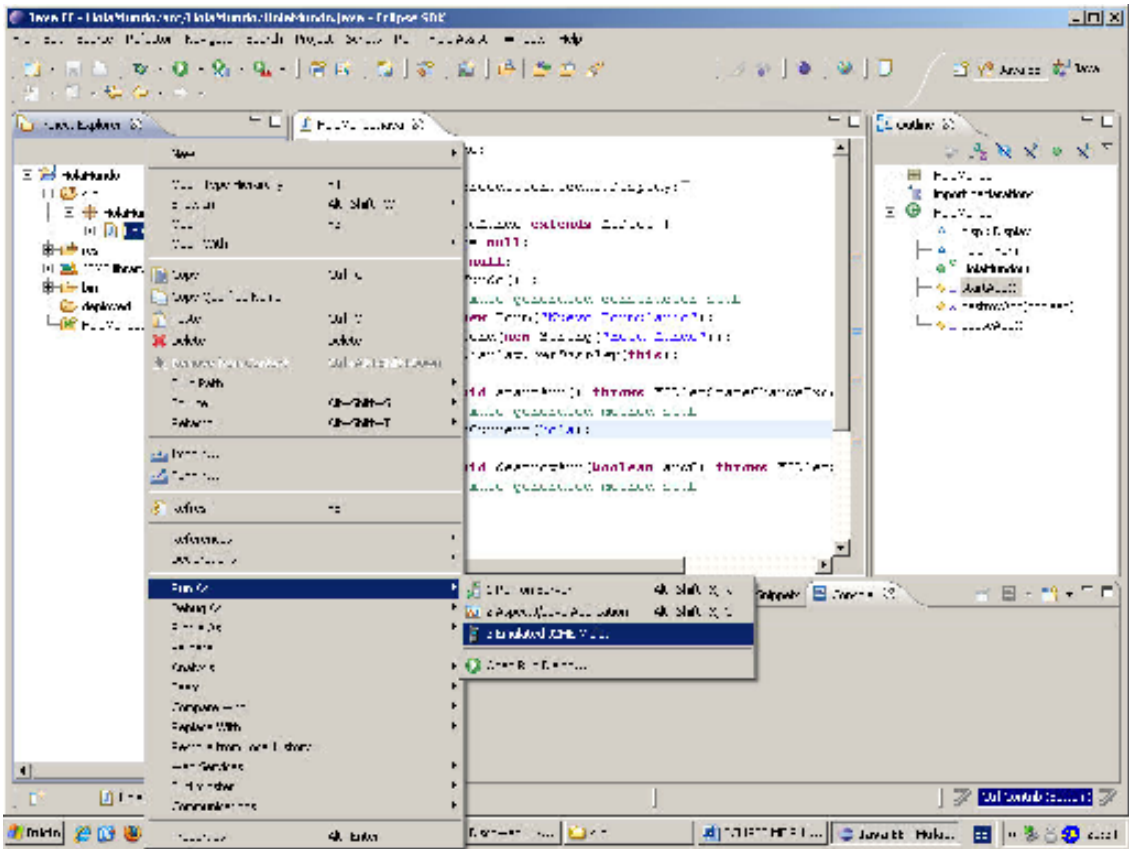


Ilustración 60. Ejecutando el MIDlet.

Si el proceso se ha seguido y como hemos explicado, al ejecutar este último paso obtendremos la siguiente salida por pantalla:

Ilustración 61. Resultado de la ejecución de HolaMundo.

9. ANEXO III: INSTALACIÓN DE UN MIDlet EN EL TELÉFONO MÓVIL

Para la instalación de aplicaciones J2ME en el teléfono móvil, necesitaremos tener instalado la aplicación del fabricante del teléfono que nos permita instalar aplicaciones en él.

A continuación se va a explicar con detalle la instalación de la aplicación en un **teléfono Nokia**.

En primer lugar se tiene que tener instalado en el ordenador personal el programa ***Nokia PC Suite***. Desde la web del fabricante, podemos descargarlo http://www.nokia.es/soporte/software/pcsuite .

Una vez instalado correctamente arrancamos el programa:

Ilustración 62. Nokia PC Suite

Una vez hecho esto, conectaremos nuestro teléfono móvil al ordenador. Para ello hay varias medios, por un cable USB, vía Bluetooth… Al hacerlo, veremos que nuestro teléfono nos pide que seleccionemos el modo de conexión, elegiremos *Modo Nokia*. El programa identificará nuestro modelo teléfono ya aparecerá la siguiente pantalla:

Ilustración 63. Nokia PC Suite, teléfono conectado

Nota: en esta explicación usaremos un teléfono Nokia 7390, como puede verse en la imagen anterior. La instalación sería igual para cualquier otro modelo Nokia que permita aplicaciones Java.

Seleccionaremos el icono  *Instalar aplicaciones* y aparecerá la siguiente pantalla:

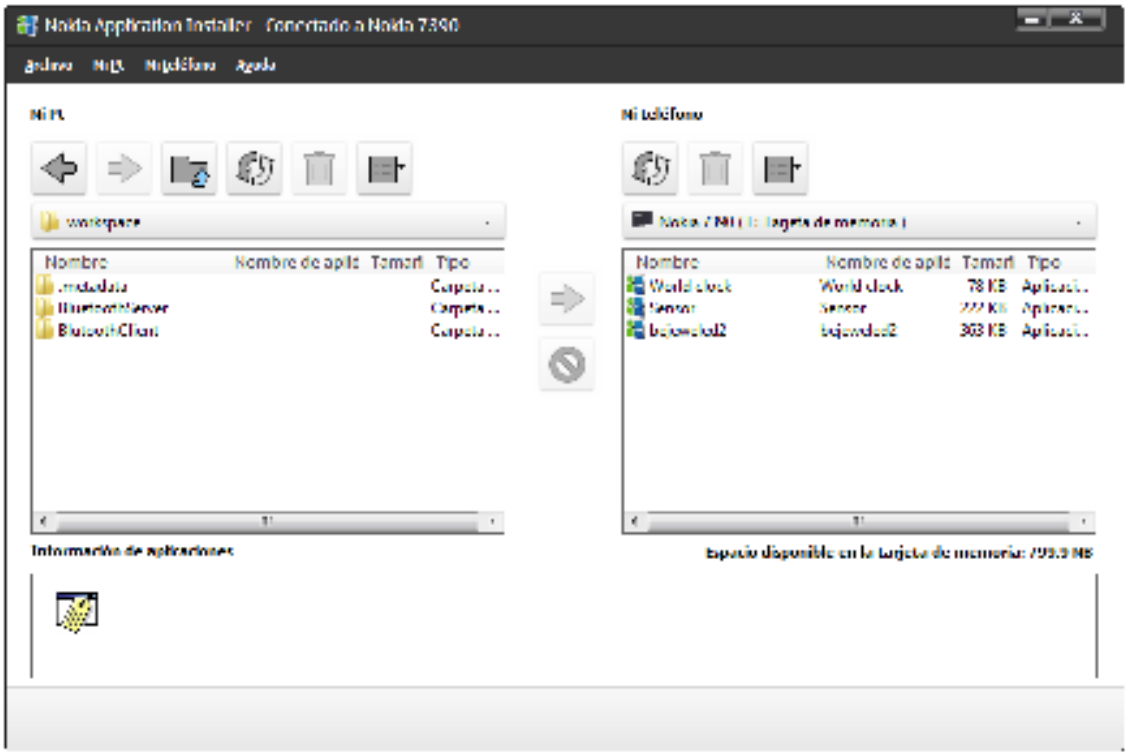


Ilustración 64. Nokia PC suite - Nokia Application Installer

Vemos que aparecen dos cuadros, el de la derecha para navegar en el sistema de archivos del ordenador personal y el de la izquierda para navegar en el sistema de archivos del teléfono móvil.

En el cuadro de navegación por el sistema de archivos del ordenador, nos iremos a la carpeta donde tengamos guardado el archivo **.jar** (ejecutable) que queramos instalar.

En este caso, instalaremos por ejemplo el módulo servidor de exámenes, llamado *BluetoothServer*, y como lo hemos desarrollado en Eclipse tendremos que ir a la carpeta *C:\directorio_de_instalacion\workspace\BluetoothServer\.eclipseme.tmp\emulation\emulation*

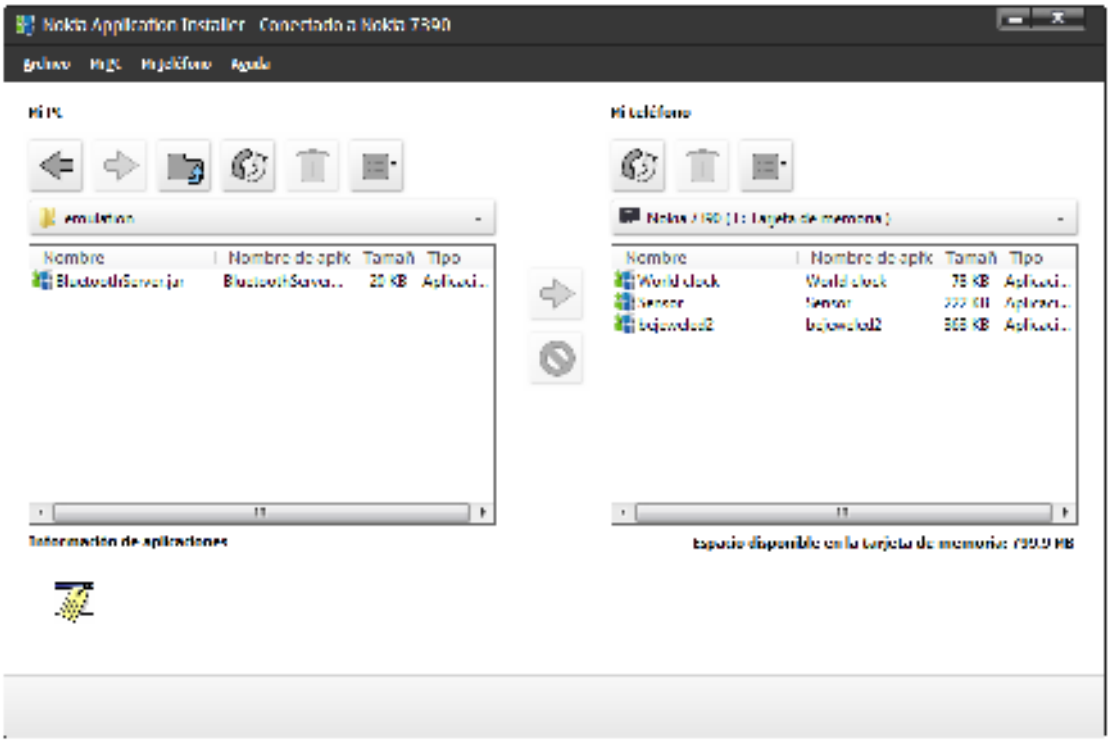


Ilustración 65. Nokia Application Installer - archivo *.jar

A continuación seleccionamos en el cuadro de navegación de archivos del móvil el directorio donde queremos instalar la aplicación.

Pinchamos sobre el ejecutable que queremos instalar, iluminándose este en azul:

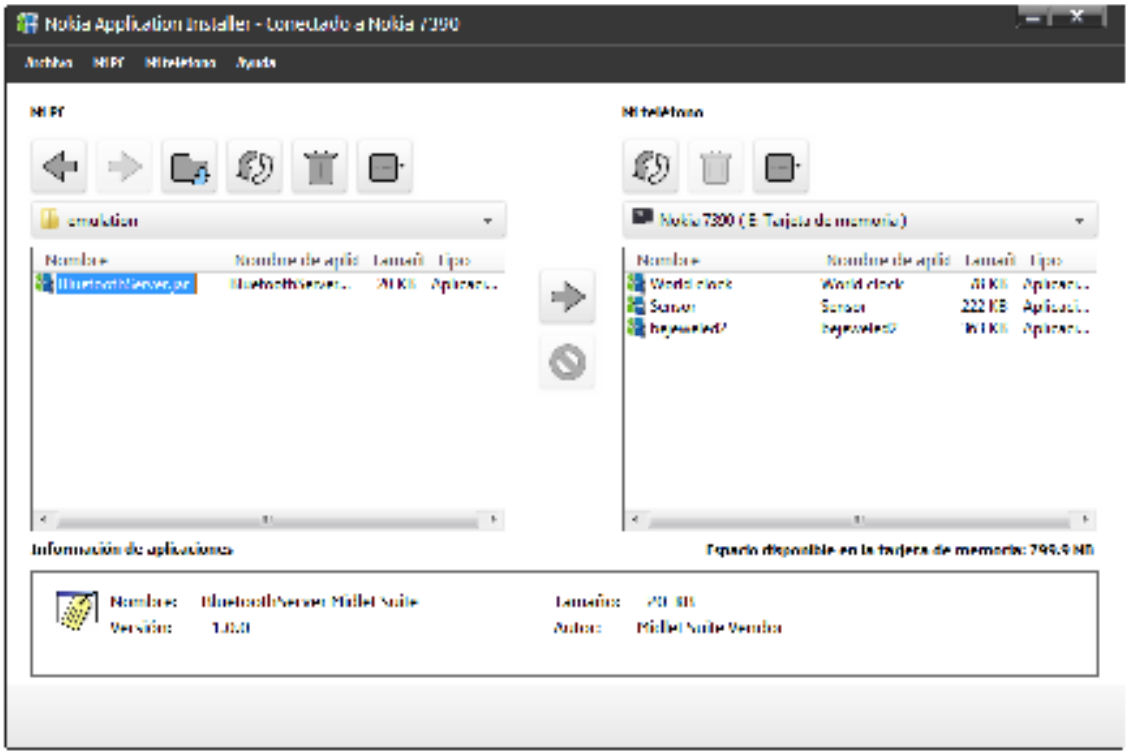


Ilustración 66. Nokia Application Installer - selección del archivo a instalar

Pinchamos sobre la flecha que va del cuadro del ordenador al cuadro del móvil, y el ejecutable se copiará al directorio seleccionado en el móvil:

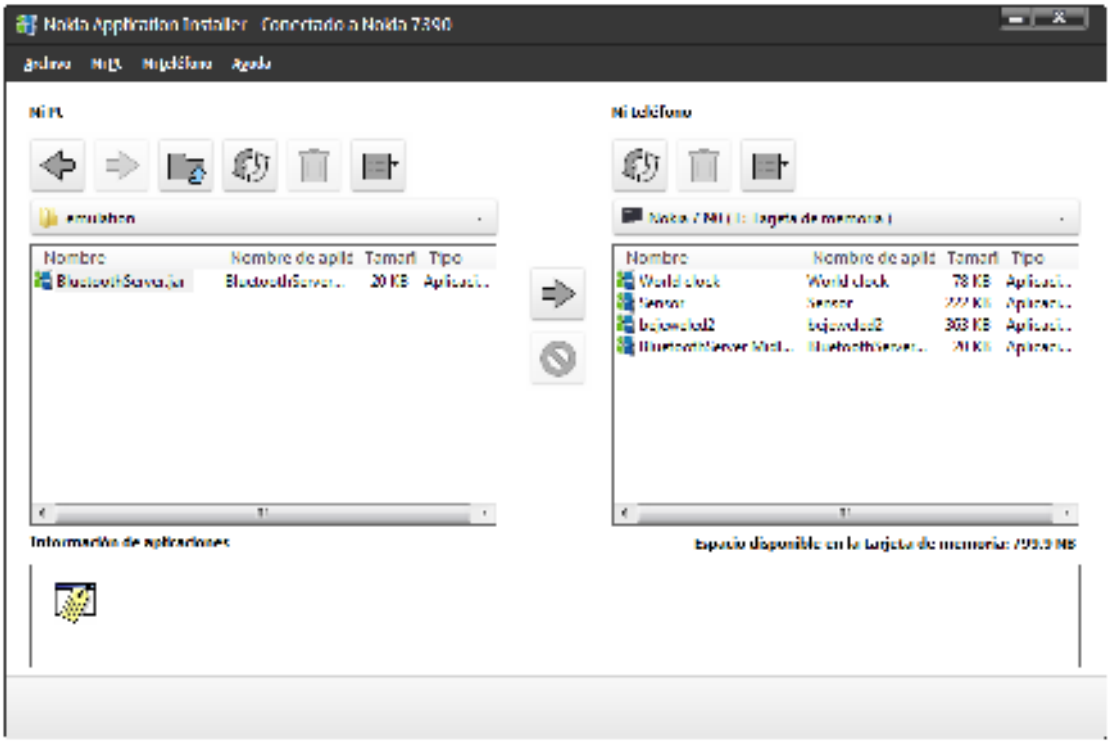


Ilustración 67. Nokia Application Installer - copia del archivo *.jar al teléfono móvil

Llegados a este punto la aplicación ya estaría instalada en el teléfono móvil.

Se cierra el *Nokia Application Instaler* y se sale correctamente del Nokia PC Suite:

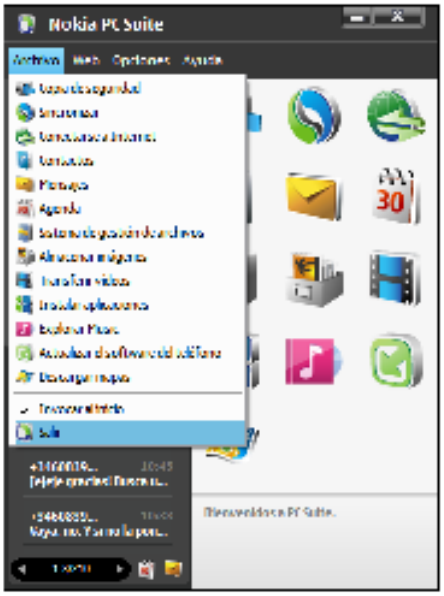


Ilustración 68. Salir de Nokia PC Suite.