UNIVERSIDAD CARLOS III DE MADRID

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES

SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

IMPLEMENTACIÓN DE UNA APLICACIÓN MOVIL DE TELEDOCENCIA EN J2ME CON TECNOLOGÍA BLUETOOTH

AUTOR: FRANCISCO JAVIER GARCÍA RUBIO

TUTOR: MARIO MUÑOZ ORGANERO

NOVIEMBRE 2009

AGRADECIMIENTOS

Sin duda, la mejor forma de comenzar este proyecto es agradeciendo todo el cariño y apoyo mostrado por aquellas personas que me quieren y aprecian.

En primer lugar, a mis padres y abuelos que siempre han confiado en mí y se han sacrificado tanto para que pudiese llegar hasta aquí. Sin vosotros nada de esto habría podido ser posible. Os quiero.

También quiero agradecerles a mis hermanos por haber tenido tanta paciencia conmigo, durante los años de universidad y sobre todo en los periodos de exámenes. Os quiero

A Elena, por ser tan compresiva y estar a mi lado durante todos estos años. Gracias por ser tan maravillosa, sin tu apoyo no habría podido terminar nunca la carrera. Te quiero.

A todos mis compañeros de la Universidad, y en especial a la AXSPI. Por toda la ayuda que me habéis prestado, y sobre todo porque sin vosotros la Universidad habría sido una experiencia muy diferente.

A mis compañeros de trabajo por animarme a terminar el Proyecto. Sé que no hay forma de agradeceros todo lo que he aprendido de vosotros, tanto profesional como personalmente, aun así, muchas Gracias.

Por último, pero no por eso menos importante, a mi tutor Mario, por toda la paciencia mostrada durante este tiempo, y por haber estado ahí siempre que he necesitado ayuda. Muchas Gracias.

RESUMEN

La tecnología de comunicación inalámbrica Bluetooth ha adquirido en estos últimos años una presencia importante en la conexión de dispositivos de uso cotidiano (móviles, impresoras, manos libres, cámaras de fotos,…). Al permitir su interconexión de una manera sencilla ha supuesto una revolución esencial que se convertirá en indispensable en pocos años

En este proyecto se ha implementado una aplicación en J2ME (Java 2 Micro Edition), que aprovecha las posibilidades que nos brinda la tecnología Bluetooth, y que está orientada a la docencia. A través de ella, el usuario podrá optar por el rol de alumno (cliente) o de tutor (servidor). De forma que, un usuario que haya elegido el rol de alumno podrá localizar a otros que hayan escogido el rol de tutor, basándose en unos criterios de búsqueda especificados por el alumno, y seleccionar aquel tutor que más le pueda interesar para establecer una conversación por chat con él.

Con la realización del proyecto se ha adquirido una visión en profundidad de J2ME, se ha estudiado la tecnología Bluetooth para comprender su funcionamiento, y posteriormente se han aplicado estos conocimientos en el análisis y desarrollo de la aplicación. También se ha realizado una batería de pruebas, con ayuda de la herramienta Sun Wireless Toolkit, para demostrar la robustez de la aplicación.

ABSTRACT

Wireless communication Bluetooth technology has acquired an important presence in daily use devices connection (mobiles, printers, free hands, cameras of photos,…) in the last years. It has supposed an essential revolution, that will become indispensably in a few years, allowing its interconnection in a simple way.

In this project an application has been implemented in J2ME (Java 2 Micro Edition), taking advantage of the possibilities that Bluetooth technology offers to us, and that is orientated to the teaching. Across it, the user will be able to choose for the pupil's role (client) or tutor´s role (servant). So that, a user who has chosen the pupil's role will be able to locate others who have chosen the tutor's role, being based on a few criteria of search specified by the pupil, and selecting that tutor who more could interest him to establish a chat conversation with him.

With the project accomplishment, it has been acquired a J2ME depth vision, there has been studied Bluetooth technology to understand its functioning, and later this knowledge has been applied in the application analysis and development. Also a tests battery has been realized, with the tool Sun Wireless Toolkit, to demonstrate the application hardiness.

Índice de Contenidos

Contenido

Índice de Figuras

Índice de Tablas

Capítulo I

Introducción

En este primer capítulo se ofrece una descripción detallada del proyecto. En primer lugar se explica cuáles han sido las razones que motivaron la realización de este proyecto y posteriormente se presentan los objetivos que se quieren alcanzar. Para finalizar este capítulo introductorio se expone la organización de la memoria.

1.1 Motivación

Hoy en día, gracias a internet, todo el mundo tiene acceso a prácticamente toda la información que pueda necesitar (enciclopedias libres, tutoriales, foros,…); algo que era impensable hace solo unos años. Sin embargo, se sigue empleando bastante tiempo en filtrar la información para hallar aquello que se busca; y si se trata de una cuestión muy técnica puede que no encontremos un resultado que abarque el tema completamente.

En el ámbito universitario, el acceso a la información es aún mayor (clases, profesores, seminarios,…). Sin embargo, en periodos de exámenes, no todos los alumnos son lo suficientemente previsores y organizados como para disponer de toda la información necesaria, por lo que acaban perdiendo bastante tiempo en recavarla antes de estudiarla. Y por si esto fuera poco, cuando la estudian, les surgen dudas, pero en la mayoría de los casos es demasiado tarde para asistir a tutorías.

Por tanto, la motivación principal que conduce a realizar este proyecto es crear una aplicación móvil basada en la tecnología Bluetooth, que permita a los estudiantes universitarios la posibilidad de establecer una conversación sobre un tema académico, con alguien que tiene conocimientos sobre el mismo y se encuentre dentro de un rango máximo de diez metros, de forma rápida y eficaz.

De esta forma, se pretende dar a los estudiantes otra herramienta más que les ayude a formarse, permitiéndoles preguntar o resolver dudas a otros alumnos que tienen conocimientos sobre el tema, bien por haberlo estudiado con anterioridad, o estar preparándose esa asignatura. Ahorrándoles así un tiempo valioso.

Por último solo mencionar que las características de la tecnología Bluetooth, tanto de alcance (ya que la aplicación está pensada para utilizarse dentro de la universidad, cafetería, biblioteca, aulas de estudio,…), transmisión (utiliza un enlace por radiofrecuencia seguro y globalmente libre, lo que no implica ningún gasto a costa del usuario) y de universalidad (prácticamente todos los teléfonos móviles de hoy en día llevan incorporada esta tecnología de serie) han sido las razones por las que se ha escogido para su desarrollo.

1.2 Objetivo

El objetivo del proyecto es diseñar e implementar una aplicación móvil, que mediante tecnología Bluetooth sea capaz de poner en contacto a los usuarios de la misma que tengan dudas acerca de una asignatura, con otros usuarios que posean conocimientos sobre esa materia, para que les ayuden a resolverlas.

Además la aplicación ha de tener las siguientes características:

- *Intuitiva* y *fácil de usar*: cualquier usuario podrá utilizarla sin tener conocimientos de J2ME o de tecnología Bluetooth, tampoco debe emplear menús complicados o que lleven a confusión.
- *Eficiente*: sobre todo en la búsqueda de usuarios que puedan realmente ayudar a resolver las dudas que les serán expuestas.
- *Flexible*: debe adaptarse con rapidez a los cambios del entorno, como los originados por usuarios que cierren la aplicación, o los usuarios con conocimientos que están ya ocupados ayudando a otros usuarios,…
- *Robusta*: ha de garantizar el servicio prestado, ya que el usuario la empleará la mayoría de las veces en momentos cercanos a los exámenes lo que implica cansancio, nerviosismo y ansiedad por lo que cualquier error podría provocar que la catalogara como inservible y no la volviese a utilizar.
- R*ápida*: ha de informar de aquellos usuarios que pueden ayudarle y establecer con ellos una comunicación que le permita resolver sus dudas en el menor tiempo posible; ya que si el usuario dispusiera de más tiempo seguramente no utilizaría esta aplicación sino que acudiría a una tutoría real o investigaría en la web sobre la duda que le ha surgido.
- *Gratuita*: la aplicación está dirigida a un sector de la población muy concreto, los estudiantes universitarios, que no dispone de mucha liquidez. Si además tenemos en cuenta que la aplicación será tanto más útil cuantas más personas la instalen en sus terminales móviles y hagan uso de ella. Considero que debe ser distribuida como software libre.

El fin que se pretende alcanzar es que los usuarios tengan una herramienta más que sea fácil de usar, eficiente, flexible y robusta que les ayude a resolver sus dudas rápidamente, sencillamente y sin ningún coste.

1.3 Organización de la memoria

La memoria ha sido organizada de tal forma que se ilustren los pasos que se han seguido para el desarrollo del presente proyecto.

En primer lugar, se tiene el proceso de documentación. Antes de comenzar a implementar es necesaria una labor de documentación acerca de los temas que se van a tratar en el proyecto, con el fin usar las técnicas más apropiadas en el mismo. En el capítulo denominado Estado del Arte se analizarán las tecnologías que se van a usar en el desarrollo de la aplicación.

Acto seguido, se abordará el desarrollo de la aplicación. En el capítulo de Especificación se expondrá qué es exactamente lo que se desea implementar indicando los requisitos que tiene que cumplir. A continuación, se incluye un capítulo de Diseño en el que se realiza un esquema de alto nivel de la aplicación con el objetivo de describir cómo se va a abordar el problema sin entrar en pequeños detalles. Será ya en el siguiente capítulo, Implementación, donde se entrará en detalle en el desarrollo de las distintas partes de la aplicación y donde se explican los diferentes problemas que han surgido y como se han solucionado, así como la batería de pruebas que se ha realizado a la misma.

Llegados a este punto se habrá adquirido un conocimiento teórico básico en cuanto a lo que a J2ME y Bluetooth se refiere; y se habrá desarrollado la aplicación y realizado una amplia batería de pruebas. Para completar la memoria, se incluye el capítulo Conclusiones y Trabajos futuros en el que se exponen las conclusiones obtenidas de la realización del proyecto y los posibles trabajos futuros relacionados con la herramienta implementada.

Después de las conclusiones y los posibles trabajos futuros, se expondrán los acrónimos y las referencias bibliográficas utilizadas. Por último, se ha añadido un anexo con diversa información que puede ser de utilidad para la comprensión de la memoria. Algunos temas importantes que se tratan son la instalación y configuración de la herramienta, y un manual del usuario básico.

Hay que señalar también que se adjunta un CD con el código realizado. En este CD se incluye una carpeta con la aplicación compilada, y otra con los ficheros fuente de la aplicación. Para hacerla funcionar solo será necesario instalar y configurar la herramienta tal y como se indica en el anexo.

Capítulo II

Estado del Arte

Previo a cualquier trabajo de desarrollo se requiere una labor de investigación para determinar qué tecnologías son las más adecuadas para la implementación de la herramienta. Por tanto, el objetivo de este capítulo es realizar una introducción a los diferentes temas que son necesarios para comprender el posterior desarrollo de la aplicación.

Dada la cantidad de conceptos relacionados con las aplicaciones móviles J2ME, no se va a profundizar en ellos sino que únicamente se va a hablar de aquello que se considere más importante para poder entender el resto de la memoria. El objetivo es tener una visión general de las tecnologías y herramientas usadas de tal forma que se pueda seguir la memoria con facilidad.

En primer lugar se hará una breve introducción al lenguaje J2ME, explicando aquellos conceptos que son básicos para comprender el proyecto como, por ejemplo, el perfil MIDP, el protocolo CLCD, un MIDlet, etc. Acto seguido, se pasará a explicar el protocolo de comunicación Bluetooth (arquitectura, protocolos, perfiles,...). Y por último, se comentarán los métodos del paquete javax.microedition.bluetooth pertenecientes a la especificación JSR-82, y que son necesarios para implementar en J2ME los protocolos Bluetooth en el cliente y el servidor de la aplicación.

2.1 Introducción a J2ME

El lenguaje de programación Java nació a mediados de los años 90 de manos de la empresa Sun Microsystems. Sus principales características de partida fueron una gran robustez e independencia de la plataforma donde se ejecutase el código. Lo que contribuyó a que desde sus comienzos se utilizara para diversos fines, como la creación de componentes interactivos integrados en páginas Web (applets) y programación de aplicaciones independientes entre otros.

Con el paso del tiempo, Java se ha ido adaptando a las necesidades de los usuarios y empresas, ofreciendo soluciones y servicios a ambos. Para ello, tuvo que desarrollar distintas versiones de Java que se ajustaran a las necesidades de cada uno:

- Java 2 Stantard Edition (J2SE): orientada al desarrollo de aplicaciones independientes y de applets.

- Java 2 Enterprise Edition (J2EE): enfocada al entorno empresarial, su cometido es ampliar la J2SE para dar soporte a los requisitos de las aplicaciones de empresa.

- Java 2 Micro Edition (J2ME): orientada a la programación de aplicaciones para dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs (Personal Digital Assistants), electrodomésticos inteligentes, etc…

Como podemos ver en la ilustración 1, J2ME es una versión simplificada de J2SE, de la misma forma que J2SE lo es también de J2EE. Sun separó estas versiones por razones de eficiencia, ya que J2ME estaba pensada para dispositivos con limitaciones de proceso y capacidad gráfica; mientras que J2EE exigía unas características muy pesadas o especializadas de E/S, trabajo en red, etc. Por esta razón, J2EE es un superconjunto de J2SE que contiene toda su funcionalidad y más características, así como J2ME es un subconjunto de J2SE excepto por el paquete javax.microedition **[3]**.

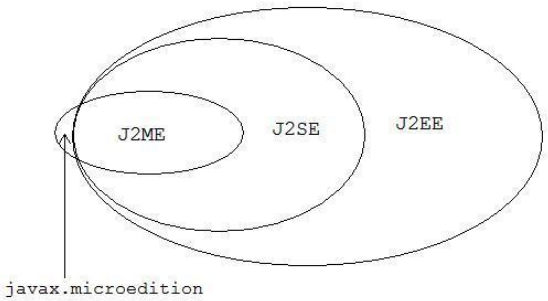


Ilustración 1: Relación entre las versiones de Java

De ahora en adelante, centraremos nuestro estudio en esta edición de J2ME, que fue presentada en 1999 por Sun Microsystems, con el propósito de habilitar aplicaciones Java para pequeños dispositivos limitados por una potencia de cálculo baja (velocidad de procesamiento lenta y cantidad de memoria escasa) e interfaces de usuario con capacidades restringidas, a través de una nueva Java Virtual Machine (JVM).

Por tanto, J2ME contiene tan solo una pequeña parte de las APIs de Java. Esto es debido a que la edición estándar de APIs de Java ocupa 20 MB, y los dispositivos pequeños disponen de una cantidad de memoria mucho más reducida. En concreto, J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util.

2.2 Arquitectura de J2ME

Un entorno de ejecución completo (Java Runtime Environment) de J2ME está compuesto por una máquina virtual, configuraciones, perfiles y un conjunto de paquetes adicionales u opcionales que dependen de la aplicación específica. J2ME utiliza las configuraciones y perfiles para personalizar el entorno de ejecución de Java (JRE), como podemos observar en la ilustración 2.

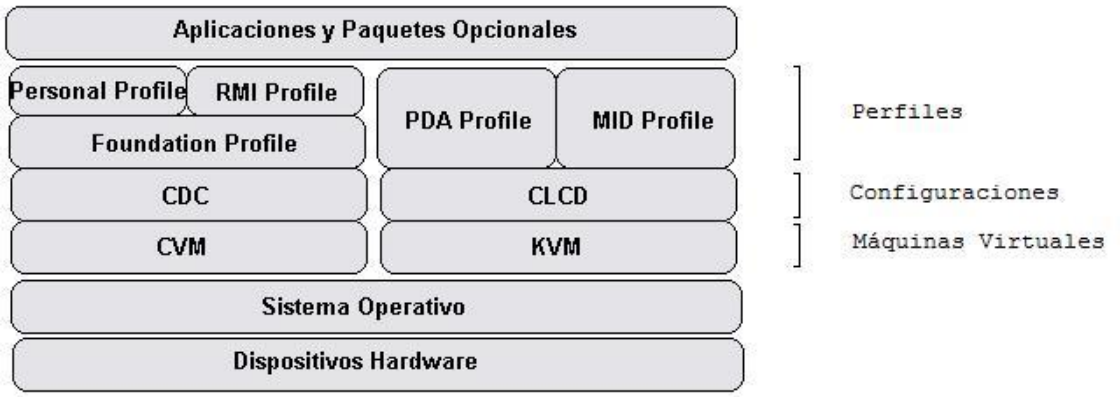


Ilustración 2: Arquitectura de J2ME

2.2.1 Máquinas Virtuales

La máquina virtual es un programa escrito en código nativo, que es el que entiende el hardware de la plataforma destino (teléfono móvil, PDA,…), y sus funciones son interpretar y ejecutar el código intermedio conocido como "bytecode" que resulta de compilar el código fuente, comunicarse con el sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java.

Esto significa que cualquier programa escrito en Java es independiente de la plataforma sobre la que vaya a ejecutarse; y por lo tanto el mismo programa puede funcionar sobre cualquier tipo de hardware, tal y como reza el axioma de Java "*write once, run everywhere*".

Existen 2 tipos de máquinas virtuales en J2ME: Kilo Virtual Machine (KVM) y Compact Virtual Machine (CVM), con requisitos diferentes para distintos dispositivos [1].

2.2.1.1 KVM (Kilobyte Virtual Machine)

Es la máquina virtual de referencia para la configuración CLDC (Connected Limited Device Configuration). Es una máquina virtual java altamente portable y compacta (ocupa en memoria entre 40Kb y los 80Kb), diseñada para funcionar en dispositivos con recursos limitados y con conexión a red. El objetivo era crear la máquina virtual de java más pequeña y completa, que mantuviese los aspectos más importantes del lenguaje de programación Java, y que funcionase en dispositivos con procesadores de 16 o 32 bits y capacidades de unos pocos cientos de KiloBytes (originalmente 128 KB de memoria).

Pero debido a su reducido tamaño en memoria, la KVM tiene ciertas limitaciones si la comparamos con la clásica JVM:

- No soporta tipos en coma flotante (double y float). Ya que los dispositivos carecen del hardware necesario para realizar estas operaciones.

- No existe soporte para JNI (Java Native Interface) debido a los recursos limitados de memoria, por lo que un programa escrito en Java no podrá interactuar con otros escritos en lenguajes distintos.

- No hay cargadores de clases (class loaders) definidos por el usuario, únicamente existen los predefinidos.

- No se permiten los grupos de hilos o hilos daemon. Cuándo se quiera utilizar grupos de hilos se deben utilizar los objetos Colección para almacenar cada hilo en el ámbito de la aplicación.

- No existe la finalización de instancias de clases, por lo que no existe el método Object.finalize().

- No hay referencias débiles.

- Limitada capacidad para el manejo de excepciones, ya que su manejo depende en gran medida de las Interfaces de Programación de Aplicaciones (APIs) de cada dispositivo; siendo éstos los que controlan la mayoría de las excepciones.

- El verificador de clases, encargado de rechazar las clases no válidas en tiempo de ejecución, es demasiado grande para la KVM, por lo que se decidió mover la mayor parte del trabajo de verificación de clases fuera del dispositivo (PC o servidor de descarga). A este proceso se le llama preverificación. De forma que los dispositivos son únicamente responsables

de ejecutar algunas pruebas en la clase preverificada para asegurarse de que aún es válida.

2.2.1.2 CVM (Compact Virtual Machine)

Es la máquina virtual de referencia para la configuración CDC (Connected Device Configuration). Soporta las mismas características que la máquina virtual de J2SE y está orientada a dispositivos electrónicos de gama alta con procesadores de 32 bits y con 2Mb o más de memoria RAM. Sus características son:

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Portabilidad.
- Rápida sincronización.
- Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).
- Soporte nativo de hilos.
- Baja ocupación en memoria de las clases.
- Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
- Conversión de hilos Java a hilos nativos.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

2.2.2 Configuraciones

Este nivel es menos visible a los usuarios, pero es muy importante para los desarrolladores de perfiles. Define el conjunto mínimo de características de la máquina virtual, las características del lenguaje de programación Java soportadas, así como las librerías de clases básicas y APIs disponibles en una categoría de dispositivos.

Así mismo, una configuración define el ambiente de ejecución básico, es decir un conjunto de clases principales y una máquina virtual específica, que están orientadas a conformar el corazón de las implementaciones para dispositivos con características específicas.

Actualmente existen dos configuraciones para J2ME, la CLDC, orientada a dispositivos con limitaciones tanto a nivel de procesamiento como de interfaz gráfica y la CDC, orientada a dispositivos con menos limitaciones.

2.2.2.1 Configuración para Dispositivos con Conexión (CDC)

Está orientada a dispositivos con cierta capacidad de cómputo y de memoria (decodificadores de televisión digital, televisores con internet, sistemas de navegación en automóviles, etc). Utiliza la CVM, que es similar en sus características a una máquina virtual de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. La CDC está enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits.
- Tener 2Mb o más de memoria total (incluyendo memoria RAM y ROM).
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

La CDC está basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones http y basadas en datagramas.

2.2.2.2 Configuración para Dispositivos Limitados con Conexión (CLDC)

Está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, procesamiento y memoria (teléfonos móviles, PDAs, etc). Utiliza la KVM, y la mayoría de las restricciones vienen dadas por su pequeño tamaño **[4]**. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

- Tener entre 160KB (128KB de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC + 32KB de memoria volátil para la Máquina Virtual en tiempo de ejecución) y 512KB de memoria total disponible.
- Procesador de 16 o 32 bits con al menos 25Mhz de velocidad.
- Bajo consumo, ya que estos dispositivos trabajan con suministro de energía limitado.
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600bps).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su máquina virtual (KVM).

- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

2.2.3 Perfiles

Los perfiles constituyen el nivel más visible a los usuarios y desarrolladores de aplicaciones, puesto que se desarrollan sobre un determinado perfil que a su vez está implementado sobre una determinada configuración. Un perfil define un conjunto de APIs y características comunes para un conjunto de dispositivos. Las clases de un perfil permiten el acceso a funcionalidades específicas de los dispositivos como la interfaz gráfica, funcionalidades de red, almacenamiento persistente, etc.

Las aplicaciones desarrolladas sobre un determinado perfil van a ser portables a cualquier dispositivo que soporte ese perfil; cabe destacar que un dispositivo puede soportar varios perfiles y que sobre una configuración pueden existir diversos perfiles.

Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Aquí se pueden encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles hasta los táctiles de los PDAs.

La principal diferencia entre configuración y perfil es que mientras que la configuración define las características de una familia de dispositivos, el perfil hace lo propio sobre un único dispositivo. Por esta razón, cuando se construye una aplicación hay que contar tanto con las APIs del perfil como de la configuración. Se debe tener en cuenta que un perfil siempre se construye sobre una configuración determinada, de este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica. En J2ME podemos escoger entre los siguientes perfiles:

2.2.3.1 Foundation Profile

Define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de J2SE, excluyendo totalmente los paquetes "java.awt" Abstract Windows Toolkit (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

2.2.3.2 Personal Profile

Diseñado sobre la CDC y es un subconjunto de la plataforma J2SE, que proporciona un entorno con un completo soporte gráfico AWT. El objetivo es que la configuración CDC disponga de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere una implementación del Foundation Profile.

2.2.3.3 RMI Profile

Este perfil también está construido sobre la CDC y requiere una implementación del Foundation Profile. El perfil RMI soporta un subconjunto de las APIs J2SE RMI (Remote Method Invocation), eliminándose algunas características de estas APIs debido a las limitaciones de cómputo y memoria de los dispositivos.

2.2.3.4 PDA Profile

El PDA Profile está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x 100 pixels) con un factor 2:1.

2.2.3.5 MID Profile (Mobile Information Device Profile)

En conjunto está diseñado para operar sobre la CLDC y es el más usado actualmente para el desarrollo de aplicaciones. Se encarga de la semántica y control de la aplicación, interfaz de usuario y trabajo en red. Permite la ejecución de aplicaciones java en dispositivos móviles (teléfonos móviles, PDAs...). Las aplicaciones desarrolladas con este perfil reciben el nombre de MIDlet.

En el desarrollo de este perfil participan empresas tales como Ericcson, Motorola, Nokia, NTT DoCoMo, Palm Computing, Sony, Siemens y Sun Microsystems.

Hasta el momento se han definido dos perfiles, el MIDP 1.0 **[5]** y el MIDP 2.0 **[6]**, cuyas especificaciones finales salieron en Septiembre de 2000 y Noviembre de 2002 respectivamente.

2.2.3.5.1 MIDP 1.0

Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma. Este perfil está orientado para dispositivos con las siguientes características **[5]**:

- Capacidad gráfica muy reducida (Display de al menos 96x54 píxeles).
- Entrada de datos alfa numérica reducida (por teclado o pantalla táctil).
- Reducida capacidad computacional y de memoria.
- 128Kb de memoria no volátil para componentes MIDP.

- 8Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32Kb de memoria volátil en tiempo de ejecución para la pila Java.
- Dos vías, acceso inalámbrico con ancho de banda limitado.

El MIDP 1.0 incluyó solo aquellas APIs que eran consideradas como requerimiento absoluto para alcanzar una amplia portabilidad, estas APIs están relacionadas con:

- Aplicación (Definición de la semántica y control de la aplicación MIDP).
- Interfaz de usuario (manejo de entradas y Despliegues).
- Almacenamiento persistente.
- Comunicaciones en red.
- Temporizadores (Timers).

2.2.3.5.2 MIDP 2.0

Desde su aparición, MIDP 1.0, se convirtió en la plataforma de programación universal más importante para los dispositivos móviles; incluyendo APIs para el manejo del ciclo de vida de las aplicaciones, conectividad con redes HTTP, interfaces de usuario y almacenamiento persistente. La versión 2.0 del MIDP incluye muchas mejoras y adiciones a la versión anterior, principalmente en los paquetes de interfaces gráficas de usuario, conectividad a redes y seguridad.

La evolución tecnológica que se ha producido en los dispositivos móviles modernos (aumento de memoria, pantallas a color, soporte multimedia (video, MP3), cámara fotográfica…) también se ve reflejada en la versión 2.0 del MIDP mejorando las capacidades de la plataforma de Java para estos dispositivos.

Para poder ejecutar aplicaciones MIDP 2.0, los dispositivos deberían tener las siguientes características mínimas **[6]**:

- Tamaño de Pantalla: 96x54 píxeles.
- Entrada por teclado o por pantalla táctil.
- 256KB de memoria no volátil para la aplicación MIDP.
- 8KB de memoria no volátil para datos persistentes de aplicaciones.
- 128KB de memoria volátil para el ambiente de ejecución Java.
- Dos vías, acceso inalámbrico con ancho de banda limitado.

- Capacidad para reproducir tonos, vía Hardware o software.

Al igual que MIDP 1.0, la versión 2.0 de este perfil solo incluyó las APIs consideradas como requerimiento indispensable para asegurar la portabilidad de las aplicaciones. El MIDP 2.0 proporciona APIs para:

- Manejo del ciclo de vida de las Aplicaciones (Definición de la semántica de una aplicación MIDP y cómo ésta es controlada).
- Firmado de aplicaciones y seguridad para dominios privilegiados.
- Transacciones seguras entre usuarios finales (https).
- Registro de aplicaciones push.
- Interconexión a redes.
- Almacenamiento persistente.
- Sonido.
- Temporizadores.
- Interfaces de usuario mejoradas (incluye despliegue, entradas y soporte para juegos).

2.2.3.5.3 Comparación entre MIDP 1.0 y MIDP 2.0

Hasta ahora hemos visto las características generales relacionadas con el hardware de los dispositivos y las capacidades de las APIs para ejecutar aplicaciones en las versiones 1.0 y 2.0 del MIDP. A continuación vamos a exponer las diferencias a nivel de software, tomando como punto de comparación los de paquetes que define cada perfil.

Tipo	Paquetes MIDP 1.0	Paquetes MIDP 2.0
Básicos o del Núcleo	java.lang	java.lang
	java.util	java.util
Ciclo de vida	javax.microedition.midlet	javax.microedition.midlet
Interconexión a Redes	javax.microedition.io	javax.microedition.io
Persistencia de datos	javax.microedition.rms	javax.microedition.rms
Interfaz de Usuario	javax.microedition.lcdui	javax.microedition.lcdui
Juegos	-	javax.microedition.lcdui.game
Seguridad	-	javax.microedition.pki
Sonido	-	javax.microedition.media
	-	javax.microedition.media.control

Tabla 1: Comparativa MIDP 1.0 y MIDP2.0 (Paquetes)

En la especificación del MIDP 2.0 tenemos 8 tipos de paquetes, incluyendo los paquetes del núcleo, mientras que MIDP 1.0 existen únicamente 5. En la tabla 1 se muestran los paquetes incluidos en cada uno de los perfiles.

A continuación veremos una breve descripción de estos paquetes **[10]**:

- **Paquetes principales:** Son dos:
  - *java.lang (CLDC):* clases del lenguaje, incluidas en el perfil, provenientes de J2SE.
  - *java.util (CLDC):* clases de utilidades incluidas en el perfil provenientes de J2SE.
- **Paquete de Ciclo de vida de las Aplicaciones (javax.microedition.midlet):** Este paquete permite a las aplicaciones MIDP (MIDlets) interactuar con el entorno, sobre el cual la aplicación se está ejecutando.
- **Paquete de Red (javax.microedition.io):** MIDP proporciona soporte de red basándose en CLDC.
- **Paquete de Persistencia (javax.microedition.rms):** MIDP proporciona este mecanismo para que los MIDlets guarden persistentemente datos y posteriormente puedan recuperarlos.
- **Paquete de Interfaz de Usuario (javax.microedition.lcdui):** Proporciona un conjunto de características para la implementación de interfaces en MIDP.
- **Paquete de Juegos (javax.microedition.lcdui.game):** Proporciona una serie de clases que permiten construir juegos ricos en contenidos para dispositivos móviles.
- **Paquete de Clave Pública (javax.microedition.pki):** Certificados usados para autentificar información proveniente de conexiones seguras.
- **Paquetes de Sonido:** son dos:
  - *javax.microedition.media:* El API Media de MIDP 2.0 es un bloque directamente compatible con la especificación Mobile Media API (MMAPI) que extiende la funcionalidad de J2ME proporcionando audio, video y otras características multimedia. Es un paquete opcional, simple y ligero, que también permite acceder a los servicios multimedia nativos de nuestro dispositivo móvil (como las cámaras de fotos de los móviles).

- *javax.microedition.media.control:* Este paquete define los tipos de control específicos que pueden ser usados en el reproductor (Player) de la API Media.

2.2.4 Aplicaciones MIDP: Midlets

Un MIDlet es una aplicación J2ME desarrollada sobre MIDP que podemos ejecutar sobre un amplio rango de dispositivos sin realizar modificación alguna. Para que esto sea posible la especificación MIDP **[6]** ha definido los siguientes requisitos:

- Todo dispositivo de información móvil ha de tener un módulo software denominado gestor de aplicaciones, encargado de la gestión de los MIDlets (cargarlos, ejecutarlos…)

- Todos los MIDlets deben ofrecer la misma interfaz a cualquier gestor de aplicaciones, para que, independientemente de la funcionalidad que implementen, los dispositivos puedan identificarlos y realizar acciones sobre ellos. Esto se consigue mediante el mecanismo de herencia, todos los MIDlets heredan de la misma clase javax.microedition.midlet.MIDlet.

2.2.4.1 Gestor de aplicaciones

Es el software encargado de gestionar los MIDlets. Reside en el dispositivo y es el que nos permite ejecutar, pausar o destruir nuestras aplicaciones J2ME. El gestor de aplicaciones recibe diversos acrónimos, entre ellos: AMS (Application Management System), JAM (Java Application Management) o GAJ (Gestor de Aplicaciones Java).

En la especificación **[6]**, no se indica que implementación concreta debe tener un gestor de aplicaciones. Sin embargo, se describen cuales son las principales funciones que debe realizar, por lo que deberán ser los fabricantes quienes se encarguen de desarrollar gestores de aplicaciones específicos para sus dispositivos. Las funciones del gestor de aplicaciones son:

2.2.4.1.1 Gestión de aplicaciones

El gestor de aplicaciones es el encargado de adaptar el funcionamiento de los MIDlets a los entornos concretos de ejecución de los dispositivos móviles. Las principales funciones que realiza un gestor de aplicaciones son:

- *Descubrimiento:* Los gestores de aplicaciones proporcionan mecanismos para obtener MIDlets de distintas fuentes (Internet, ficheros locales,…), de no ser así, solo tendríamos los MIDlets incluidos por el fabricante. Existen múltiples mecanismos de carga, entre los que destacan: el cable y la tecnología inalámbrica.

  Un dispositivo puede soportar varios métodos de carga; en cuyo caso, el proceso de carga deberá incluir una fase intermedia de selección, en donde el usuario pueda elegir el método de carga que desea emplear.

- *Instalación:* Una vez cargado el MIDlet, el gestor de aplicaciones será el encargado de su instalación. Este proceso varía en función del tipo de dispositivo, y suele incluir comprobaciones de seguridad y adaptaciones al modelo de almacenamiento concreto del dispositivo. Una vez instalado, el MIDlet queda almacenado en una zona de memoria persistente del dispositivo MID, por un tiempo indefinido, hasta que sea borrado.

- *Ejecución:* El gestor de aplicaciones es el encargado de iniciar la ejecución de los MIDlets. Además cuenta con facilidades para interrumpir y reanudar esta ejecución en función del estado del dispositivo.

- *Actualización:* El gestor de aplicaciones también proporciona facilidades de actualización y gestión de las versiones, de forma que podamos disfrutar de nuevas versiones de un MIDlet. Para ello deberá ser capaz de distinguir cuando un MIDlet es una actualización de un MIDlet ya instalado en el dispositivo, y darnos la opción de realizar esta actualización.

- *Borrado:* Por último, los gestores de aplicaciones permiten desinstalar los MIDlets, liberando los recursos reservados (memoria ocupada por el MIDlet y datos almacenados durante su utilización).

2.2.4.1.2 Gestión de estados del MIDlet

Pero los dispositivos se dedican a otras actividades (llamadas telefónicas, enviar mensajes de texto…) además de ejecutar aplicaciones. Por esta razón, un MIDlet tiene que poder ser interrumpido y lanzado repetidas veces sin que esto afecte a su comportamiento.

Para ello, el entorno de ejecución interactúa con los MIDlets a través de llamadas que realiza el gestor de aplicaciones a unas funciones predeterminadas comunes a todos los MIDlets. Estas llamadas permiten al MIDlet pasar de un estado de ejecución a otro de forma controlada, manteniendo la integridad de la información y de acuerdo a unas transiciones establecidas en el ciclo de vida de los MIDlets. El gestor de aplicaciones se encargará de almacenar el estado de los MIDlets en cada transición.

2.2.4.2 Ciclo de vida de un MIDlet

Los MIDlets pasan por distintos estados a lo largo de su ejecución, desde su inicio hasta su finalización. El ciclo de vida de un MIDlet **[9]** describe todos sus estados, así como los eventos que disparan las transiciones entre ellos, como podemos apreciar en la ilustración 3.

Los tres estados por los que pasa un MIDlet durante un ciclo de vida son:

- **Estado de Espera:** Un MIDlet se encuentra en estado de espera 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.

- **Estado Activado:** Un MIDlet se encuentra activado durante su ejecución normal. Se puede llegar aquí tanto al inicio de una ejecución como después de una interrupción.

- **Estado Destruido:** Cuando un MIDlet termina su ejecución pasa al estado de destruido y se liberan todos los recursos ocupados por él.

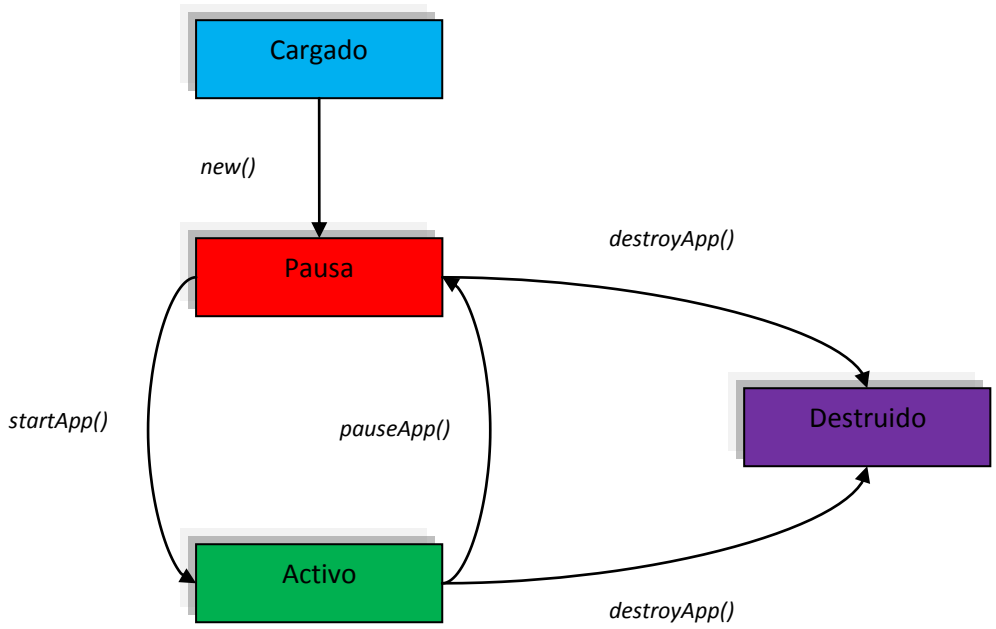


Ilustración 3: Ciclo de vida del MIDlet

Todos los cambios de estado de un MIDlet son realizados por el gestor de aplicaciones. Sin embargo, también los MIDlets pueden solicitar un cambio de estado al gestor de aplicaciones como consecuencia, por ejemplo, de una acción de usuario. Para facilitar estos cambios de estado los MIDlets cuentan con los siguientes métodos de la clase javax.microedition.midlet.MIDlet:

Estado	Llamada previa al cambio de Estado (Gestor de Operaciones -> MIDlet)	Solicitud de cambio de estado (MIDlet -> Gestor de Operaciones)
Activo	startApp()	resumeRequest()
Espera	pauseApp()	notifyPause()
Destruido	destroyApp()	notifyDestroy()

Tabla 2: Funciones de cambio de estado

El programador es responsable de incluir en estos métodos el código necesario para mantener la consistencia de la aplicación (borrar registros, inicializar variables, abrir o cerrar conexiones…).

Durante una ejecución típica, se dan los siguientes pasos en el ciclo de Vida de un MIDlet:

- **Paso 1:** se llama al constructor del MIDlet, el cual pasa a estado "Pausa" durante un corto período de tiempo. El gestor de aplicaciones crea una nueva instancia del MIDlet, e invoca al método startApp() para pasar a estado "Activo" cuándo el dispositivo está preparado para ejecutar el MIDlet.

- **Paso 2:** El MIDlet, en estado "Activo", se ejecuta normalmente ocupando todos los recursos necesarios. Durante este estado, el MIDlet puede pasar al estado de "Pausa", invocando al método pauseApp(), por acción de un evento externo o del propio MIDlet que obligue al gestor de aplicaciones a pararlo. En cuyo caso, se reducirá al máximo el uso de los recursos del dispositivo por parte del MIDlet.

- **Paso 3:** El gestor de aplicaciones decide continuar con la ejecución de la aplicación invocando al método startApp(), y cambia el estado del MIDlet a "Activo".

- **Paso 4:** Cuando el MIDlet finalice su ejecución, independientemente de que su estado sea "Activo" o "Pausa", el MIDlet libera sus recursos e informa al gestor de aplicaciones que desea pasar a estado "Destruido", mediante una llamada a notifyDestroyed(). En caso de que sea el Gestor de operaciones el que quiera finalizar la ejecución del MIDlet porque, por ejemplo, una aplicación prioritaria necesite ser ejecutada en memoria en su lugar, realizará una llamada al método destroyApp().

2.2.4.3 Persistencia de datos (RMS)

Durante la ejecución de un MIDlet, en la mayoría de los casos, se necesitará almacenar una serie de datos. La memoria RAM del dispositivo se utiliza para almacenar datos temporales, mientras que para salvar datos de forma persistente y que no se eliminen al finalizar de la ejecución del MIDlet, MIDP proporciona una sencilla base de datos orientada a registros llamada Sistema de almacenamiento de registros (RMS-Record Management System).

La integridad de los repositorios RMS dependerá del dispositivo, incluyendo reinicios y cambios de batería del mismo, al igual que la ubicación de los repositorios.

La unidad básica del RMS es el registro (record) que se almacenará dentro de una base de datos especial, llamada almacén de registros (record store), asignándole un identificador único (record id) **[2]**.

El API de MIDP incluye un paquete específico para el RMS, javax.microedition.rms. El primer paso para trabajar con RMS es abrir un almacén de registros, para ello se utilizará el método openRecordStore() de la clase RecordStore.

Para añadir un registro nuevo al almacén se usará el método addRecord() pasándole como argumento los datos que se desean almacenar convertidos en una

matriz de bytes. Este método devolverá el identificador del registro añadido que servirá para identificarlo en el almacén. Mientras que, para recuperar un registro del almacén de datos está el método getRecord(), al cual pasaremos como parámetro el identificador del registro, y nos devolverá una matriz de bytes. Otro método bastante utilizado es getNumRecords(), con el cual podremos obtener el número de registros almacenados en un Record Store.

Si se quiere eliminar un registro de manera definitiva de un almacén, se empleará el método deleteRecord() pasándole como argumento el identificador del registro. Por último es importante cerrar el almacén cuando se termina de trabajar con él mediante el método closeRecordStore() **[2]**.

2.3 Introducción a Bluetooth

Bluetooth es un protocolo de comunicaciones diseñado para dispositivos de bajo consumo, coste reducido y cobertura limitada; que utiliza señales de radiofrecuencia para establecer comunicación con el resto de dispositivos que lo implementan y se encuentran dentro de su alcance, sin necesidad de cables o de que los dispositivos estén alineados (infrarrojos). Los principales objetivos que se pretenden lograr con esta tecnología son:

- Facilitar las comunicaciones entre equipos fijos y móviles.
- Eliminar cables y conectores entre los quipos.
- Permitir que entre equipos personales se establezcan pequeñas redes inalámbricas, que faciliten la sincronización de datos entre ellos.

El nombre de tecnología proviene de un rey danés y noruego Harald Blåtand cuyo apellido se traduciría al inglés como Bluetooth, fue famoso por unificar a las tribus noruegas, danesas y suecas. Al igual que Harald unió esas tribus, se esperaba que la tecnología Bluetooth uniese los dispositivos móviles, los ordenadores y los periféricos.

La tecnología Bluetooth tiene su origen en el año 1994, cuando Ericsson empezó a investigar alternativas para comunicar móviles con ciertos accesorios. Posteriormente se decidió que para que esta tecnología tuviese éxito tendría que ser un estándar abierto; razón por la cual a principios de 1998 Ericsson se unió a Intel, IBM, Nokia y Toshiba para formar el Bluetooth Special Interest Group (SIG) con el fin de desarrollar esta tecnología conjuntamente. En los años sucesivos, otras compañías se añadieron a este grupo, y en julio de 1999 SIG publicó la versión 1.0 de la especificación Bluetooth.

Bluetooth opera en la banda libre de radio ISM a 2.4 Ghz, a una velocidad máxima de transmisión de datos de 1 Mbps. La banda ISM, al ser una banda libre, está abierta a cualquier usuario que la quiera utilizar, por lo que el sistema de radio Bluetooth deberá ser capaz de evitar las interferencias que puedan producirse. Para ello emplea la técnica de salto de frecuencia, a una alta velocidad (1600 saltos/segundo) y

corta longitud de paquetes (2871 bits máximo). Con este sistema se divide la banda de frecuencia en 79 canales separados 1MHz unos de otros, de forma que los transceptores durante la conexión cambiarán de un canal de salto a otro de manera pseudo-aleatoria garantizándose así seguridad y robustez [14].

Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo duplex), que divide el canal en intervalos de 625 µs llamados slots, y cada salto de frecuencia será ocupado por un único slot.

Cada dispositivo Bluetooth tiene asignada una dirección Bluetooth única de 48 bits, equivalente a la dirección hardware asignada a la tarjeta de interfaz de red (NIC) y que está basada en el estándar IEEE 802.11 para WLAN. Esta dirección no sólo se utiliza para la identificación, también para sincronizar el salto de frecuencia entre los dispositivos y la generación de claves en los procedimientos de seguridad.

Dos o más unidades Bluetooth pueden compartir el mismo canal dentro de una Piconet, como veremos más adelante. El salto de frecuencia del canal viene determinado por la secuencia de la señal, es decir, por el orden en que llegan los saltos y por la fase de esta secuencia. En Bluetooth, la secuencia está fijada por la dirección Bluetooth de la unidad maestra de la Piconet, y por su frecuencia de reloj.

La información que se intercambia entre dos unidades Bluetooth se realiza mediante un conjunto de slots que forman un paquete de datos. Cada paquete comienza con un código de acceso de 72 bits, que se deriva de la dirección Bluetooth del dispositivo maestro, seguido de un paquete de datos de cabecera de 54 bits que contiene importante información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits para la retransmisión automática de la pregunta, y chequeo de errores de campos de cabecera. Finalmente, el paquete que contiene la información, que puede seguir al de la cabecera, tiene una longitud de 0 a 2745 bits. En la ilustración 4 podemos ver la composición de un paquete de datos Bluettoth.

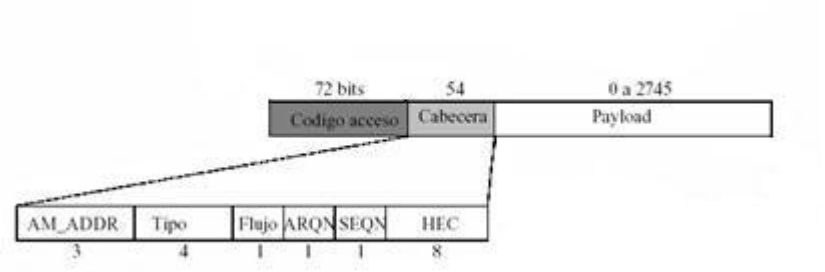


Ilustración 4: Composición del Paquete Bluetooth

En cualquier caso, cada paquete que se intercambia en el canal está precedido por el código de acceso. Los receptores de la Piconet comparan las señales que reciben con el código de acceso, si no coinciden, el paquete recibido no es considerado como válido en el canal y el resto de su contenido es ignorado.

Los paquetes de datos además están protegidos por un esquema ARQ (repetición automática de consulta), mediante el cual los paquetes perdidos son automáticamente retransmitidos.

Para conseguir un bajo consumo y coste reducido, se ideó una solución que se puede implementar en un chip utilizando circuitos CMOS (Complementary Metal Oxide Semiconductor) y que consume aproximadamente un 97% menos de energía que un teléfono móvil.

Bluetooth tiene un alcance óptimo de entre 10 y 100 metros dependiendo de la potencia empleada en la transmisión por el dispositivo y de si se emplean amplificadores. Como podemos ver en la tabla 3, los dispositivos se clasifican en tres clases en función de su potencia de transmisión, siendo totalmente compatibles los dispositivos de todas las clases entre sí.

Dispositivo	Potencia máxima permitida (mW)	Potencia máxima permitida (mW)	Alcance
Clase 1	100 mW	20 dBm	100 m
Clase 2	2,5 mW	4 dBm	10 m
Clase 3	1 mW	0 dBm	1 m

Tabla 3: Clasificación de los Dispositivos Bluetooth

La mayor parte de los dispositivos que usan Bluetooth son de clase 2, lo que permite un alcance de unos 10 metros en un ambiente libre de obstáculos. Aunque en la mayoría de los casos la cobertura efectiva de un dispositivo clase 2 aumenta cuando se conecta a un dispositivo de clase 1. Esto es debido a que el dispositivo clase 1 tiene mayor potencia de transmisión, lo que le permite enviar la señal con energía suficiente hasta el dispositivo de clase 2; y por otro lado al tener el dispositivo clase 1 una sensibilidad mayor, puede recibir del dispositivo de clase 2 la señal aun siendo esta más débil.

2.4 Arquitectura Bluetooth

La especificación Bluetooth tiene como objetivo que los dispositivos Bluetooth de diferentes fabricantes puedan trabajar unos con otros, por lo que no basta con especificar un sistema radio. Por esta razón, la especificación Bluetooth no sólo esboza un sistema de radio sino que define una pila de protocolos, así como varios perfiles con el fin de garantizar que los dispositivos Bluetooth puedan descubrirse entre sí, explorar los servicios que tiene cada uno, y hacer uso de ellos.

2.4.1 Arquitectura Hardware

En el hardware de un dispositivo Bluetooth podemos distinguir dos partes bien diferenciadas:

- Dispositivo de radio: su función es modular y transmitir la señal

- Controlador digital: compuesto por una CPU, un procesador de señales digitales (DSP-Digital Signal Processor) llamado Link Controller (Controlador de Enlace) y por los interfaces con el dispositivo anfitrión.

La CPU del dispositivo es la encargada de atender las instrucciones Bluetooth del dispositivo anfitrión. Para ello, sobre la CPU corre un software denominado Link Manager (LM) cuya función es comunicarse con otros dispositivos por medio del protocolo LMP.

El Link Controller (LC) se encarga de procesar la banda base, manejar los protocolos ARQ y FEC de capa física, de las funciones de transferencia (síncronas y asíncrona), de la codificación de Audio y del cifrado de datos.

2.4.2 Estructura de la Red Bluetooth

Cuando dos o más dispositivos Bluetooth dentro de un rango de alcance establecen una conexión, se forma un área de trabajo personal (PAN-Personal Area Network). Estas pueden ser Piconets o Scatternets **[14]**.

A una Piconet se pueden llegar a conectar 8 dispositivos, uno de ellos será el dispositivo maestro, y se encargará de controlar el tráfico de esta red; siendo este en principio el primer dispositivo que inicia la conexión Bluetooth. La unidad maestra será la encargada de controlar el tráfico de esta red. El resto de dispositivos serán unidades esclavas y sólo podrán transmitir datos cuando el dispositivo maestro les conceda tiempo de transmisión, tampoco podrán comunicarse directamente entre sí, toda comunicación debe estar orientada a través del maestro. Los Esclavos sincronizan su salto de frecuencia con el maestro utilizando el reloj de este y la dirección Bluetooth.

Las Piconets, por tanto, son redes con topología de estrella, con el maestro como nodo central, como se muestra en la ilustración 5. Un mismo dispositivo puede pertenecer a dos Piconets distintas. Entre las Piconets no se sincronizan los saltos de frecuencia que van a utilizar, de ahí que diferentes Piconets puedan encontrarse al azar en la misma frecuencia.

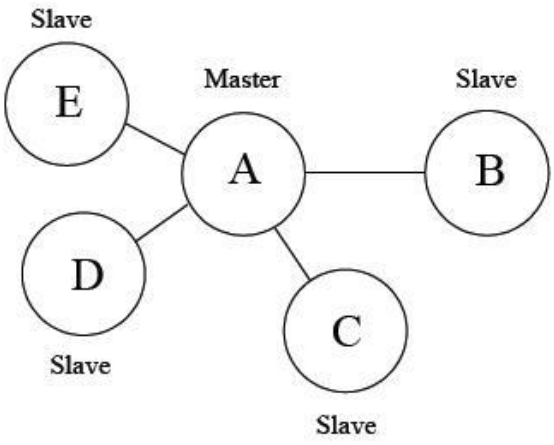


Ilustración 5: Estructura de una Piconet

Varias de estas Piconets pueden establecer comunicación entre ellas formando una Scatternet, de esta forma pueden comunicarse más dispositivos, y se aprovecha más el ancho de banda, ya que, de esta forma al tener diferentes canales cada Piconet, los dispositivos que necesiten comunicarse en mayor medida se ven menos limitados.

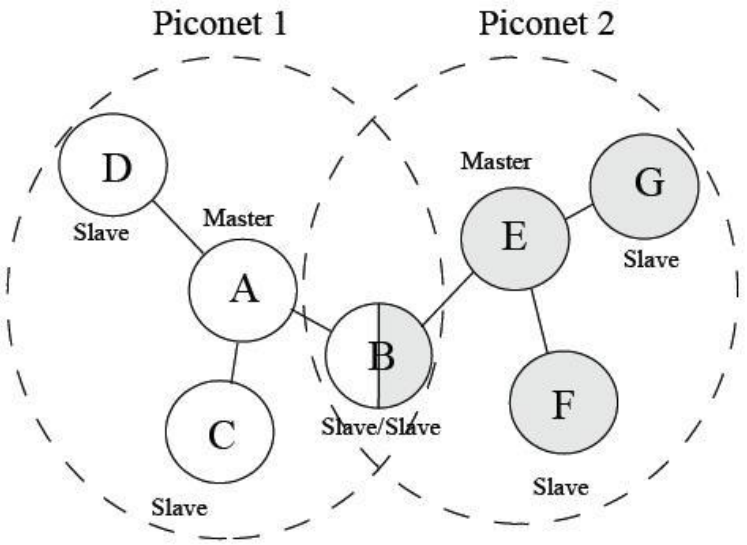


Ilustración 6: Scatternet compuesta por dos Piconets

La ilustración 6 muestra un ejemplo de Scatternet, con un nodo intermedio conectando dos Piconets. El nodo intermedio estará a tiempo compartido en ambas Piconets, lo que significa que debe seguir el salto de frecuencia de cada Piconet en instantes de tiempo distintos y que disminuirá la cantidad de slots de tiempo disponibles para la transferencia de datos entre el nodo intermedio y cualquiera de los maestros, reduciéndose la tasa de transferencia al menos a la mitad. También es importante señalar que ni la versión 1.1 ni la versión 1.2 de la especificación Bluetooth definen como se deben enrutar paquetes entre Piconets, por lo que las comunicaciones entre Piconets no son fiables.

Sin embargo, los dispositivos podrán intercambiar sus roles en una Piconet. Por ejemplo: tenemos dos dispositivos A y B. El dispositivo A se conecta al dispositivo B convirtiéndose en el dispositivo maestro de la Piconet formada por estos dos dispositivos tal y como podemos ver en la ilustración 7.

A continuación, un dispositivo C quiere unirse a la Piconet. El dispositivo C se conecta al dispositivo maestro, A y al iniciar la conexión se convertirá automáticamente en el maestro de la conexión entre el dispositivo C y el dispositivo A. Por lo que ahora tenemos dos maestros, y por lo tanto dos Piconets. El dispositivo A es el nodo intermedio entre estas Piconets, siendo el dispositivo maestro para el dispositivo B y el dispositivo esclavo para el dispositivo C, como se ve en la ilustración 8.

La ilustración 9 muestra que un cambio de roles entre el dispositivo A y el dispositivo C nos permitirá tener una única Piconet, donde A es el maestro y B y C son esclavos. Vemos que cuando un nuevo dispositivo quiere ser parte de una Piconet realmente es necesario un cambio de roles para que esto ocurra, de no ser así tendremos una Scatternet.

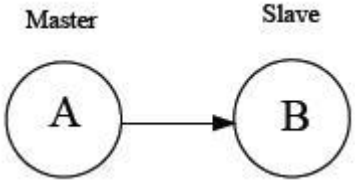


Ilustración 7: Intercambio de rol en una Piconet (1)

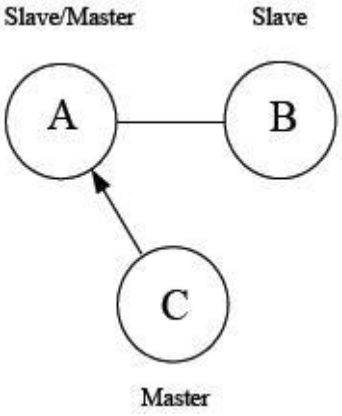


Ilustración 8: Intercambio de rol en una Piconet (2)

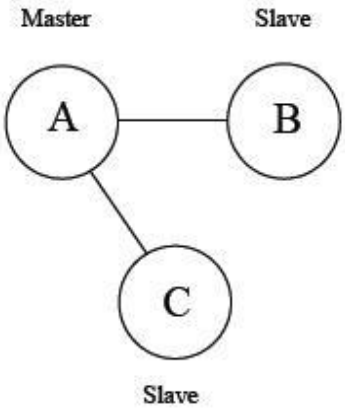


Ilustración 9: Intercambio de rol en una Piconet (3)

2.4.3 Protocolos Bluetooth

La pila de protocolos Bluetooth está basada en el modelo de referencia OSI (Open System Interconnect) de ISO (Internacional Standard Organization) para la 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.

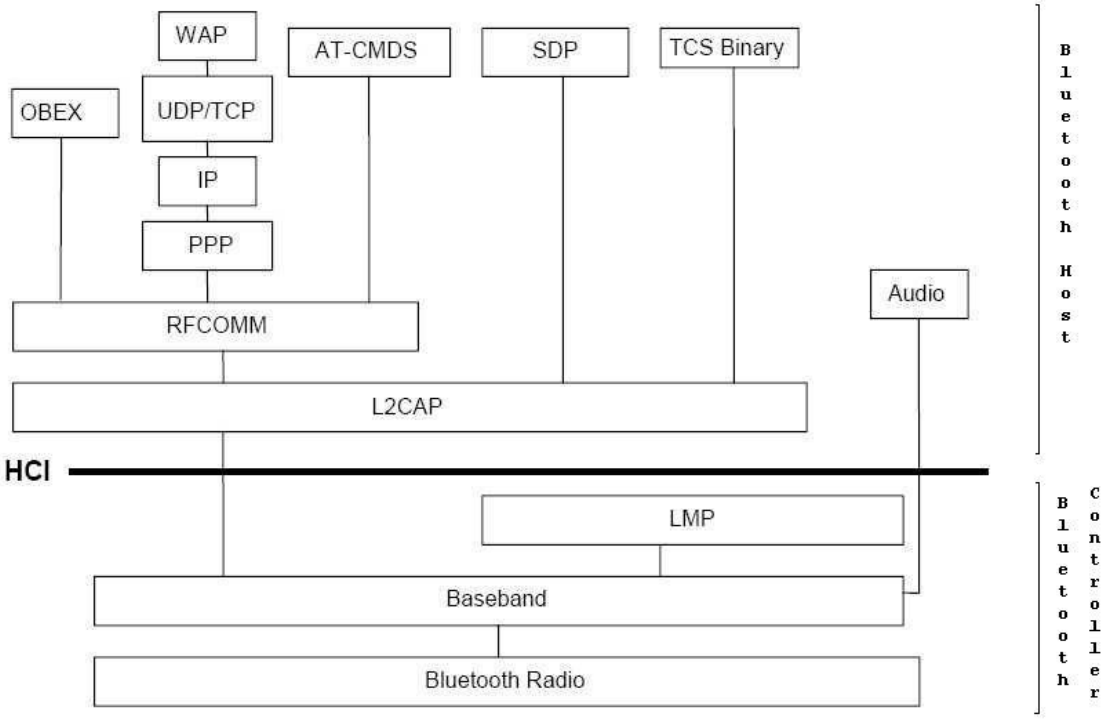


Ilustración 10: Pila de protocolos Bluetooth

Como se muestra en la ilustración 10, la pila de Bluetooth se puede dividir en dos componentes: el Host Bluetooth y el Controlador Bluetooth (o módulo de radio Bluetooth). La capa HCI (Host Controller Interface) provee una interfaz estandarizada para estos dos elementos. El HCI es por lo general la separación de la capa de software y de hardware, por lo que una parte está implementada en software y otra en hardware/firmware. Las capas por debajo de la HCI se suelen implementar en hardware y las capas por encima en software. Sin embargo, también podemos encontrar dispositivos de recursos limitados, tales como auriculares Bluetooth, que tengan toda su funcionalidad implementada en hardware/firmware.

La pila de protocolos se compone de protocolos específicos a la tecnología inalámbrica Bluetooth, como L2CAP y SDP; y de otros protocolos adoptados, como OBEX. La pila puede dividirse en cuatro capas de acuerdo a su propósito como vemos en la siguiente tabla:

Grupo de Protocolos	Protocolos en la pila
Protocolos del núcleo Bluetooth	Banda Base, Protocolo de Control de Enlace, L2CAP y SDP
Protocolo de Reemplazo de Cable	RFCOMM
Protocolo de Control de Telefonía	TCS Binario
Protocolos Adoptados	PPP, UDP/TCP/IP, OBEX, WAP

Tabla 4: Función de los Protocolos Bluetooth

La capa de banda base crea el enlace físico por radiofrecuencia entre las unidades Bluetooth estableciendo una conexión. El Protocolo de Control de Enlace (LMP-Link Manager Protocol) es responsable de la inicialización del enlace entre los dispositivos Bluetooth, de la seguridad, encriptación y autentificación. L2CAP adapta protocolos de la capa superior a la banda base y multiplexa entre varias conexiones lógicas y conexiones creadas por las capas superiores. Los datos de audio no atraviesan el L2CAP se envían desde y para la banda base. El SDP se usa para consultar información del dispositivo, servicios o características de estos. RFCOMM emula las señales de control y datos del estándar de comunicaciones RS-232 **[8]** sobre la banda base de Bluetooth, ofreciendo capacidades de transporte para servicios superiores que usan mecanismos de transporte de interfaz serie. TCS Binario define la señalización de llamada de control para el establecimiento de llamadas de voz y datos entre dispositivos Bluetooth. A continuación, describiremos un poco más en profundidad algunas de estas capas:

- Capa de banda base e interfaz de radio: su función principal es permitir el enlace físico por radiofrecuencia (RF) entre unidades Bluetooth dentro de una Piconet realizando tareas de modulación y demodulación de los datos en señales RF transmitidas. En este nivel distinguimos dos tipos de enlace físico:
    - Enlace asíncrono sin conexión (Asynchronous Connectionless-ACL):
        - Conexiones simétricas o asimétricas punto-multipunto entre maestro y esclavo.

        - Conexión utilizada para tráfico de datos.
        - Sin garantía de entrega, se retransmiten paquetes.
        - La máxima velocidad de envío es de 721 Kbps en una dirección 57.6 Kbps en la otra.

    - Enlace síncrono orientado a conexión (Synchronous Connection Oriented-SCO):
        - Conexiones simétricas punto a punto entre maestro y esclavo.
        - Conexión capaz de soportar voz en tiempo real y tráfico multimedia.
        - Velocidad de transmisión de 64 KB/s.

- Capa de Protocolo de Gestión de Enlace (LMP): es el responsable de la configuración y control de enlace entre dispositivos Bluetooth, incluyendo el control y negociación del tamaño de los paquetes de banda base.

- Host Controller Interface (HCI): es una capa de software que intercambia todos los datos entre un host (por ejemplo, un PC) y un controlador (un dispositivo Bluetooth USB). Datos y voz pasan a través del HCI.

  Una de las tareas más importantes del interfaz HCI es el descubrimiento de dispositivos Bluetooth que se encuentren dentro del radio de cobertura. Esta operación se denomina consulta o inquiry y funciona del siguiente modo:

    - Inicialmente, el dispositivo origen envía paquetes inquiry y se mantiene en espera de recibir respuestas de otros dispositivos presentes en su zona de cobertura.

    - Si los dispositivos destino están configurados en modo visible (discoverable) se encontrarán en estado inquiry_scan y en predisposición de atender estos paquetes. En este caso, al recibir un paquete inquiry cambiarán a estado inquiry_response y enviarán una respuesta al host origen con sus direcciones MAC y otros parámetros.

    - Los dispositivos que estén configurados en modo no visible (non discoverable) se encontrarán en modo inquiry_response y, por tanto, no responderán al host origen y permanecerán ocultos.

- Logical Link Control and Adaptation Protocol (L2CAP): Es la capa núcleo de la pila por la que todos los datos deben pasar. Incluye segmentación de paquetes y reordenación de datos, así como multiplexación de protocolos. Permite por lo tanto aceptar datos de SDP y RFCOMM, al estar en una capa superior. Sólo transporta datos, no voz.

- Service Discovery Protocol (SDP): permite a una aplicación cliente obtener información sobre servidores SDP disponibles en otros dispositivos Bluetooth cercanos, enumerar los servicios que ofrecen y las características de dichos servicios. Después de haber localizado los servicios disponibles en un dispositivo, el usuario puede elegir aquel de ellos que resulte más apropiado para el tipo de comunicación que desea establecer.

Toda la información relacionada con un servicio que mantiene un servidor SDP está contenida en un Service Record o registro individual de servicio. Un Service Record consiste en una lista de atributos que describen características de un servicio.

El resultado de la petición SDP devolverá al cliente que la originó una lista de servicios descubiertos acompañada por la definición de los mismos a través de sus Service Records.

- RFCOMM: emula las señales de control y datos RS-232 [8] sobre la banda base, proporcionando capacidades de transporte a los servicios de niveles superiores que utilizan el cable serie como mecanismo de transporte. Es un protocolo de transporte sencillo que soporta hasta 9 puertos serie RS-232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos Bluetooth.

- Telephony Control Protocol Specification (TCS, TCS-Binary, TCS-BIN): Se usa para enviar señales de control a dispositivos que quieren utilizar las capacidades de audio dentro de Bluetooth.

- Wireless Access Protocol (WAP): Es un protocolo adoptado en Bluetooth para cubrir sus necesidades. Requiere que PPP, IP y UDP estén presentes en la pila.

- Object Exchange (OBEX): Es un protocolo de comunicación inicialmente definido por la Asociación de Datos Infrarrojos (IrDA) Al igual que WAP, OBEX fue definido por otro grupo, pero fue adoptado por el SIG Bluetooth. Es útil para transferir objetos como archivos entre dispositivos Bluetooth. OBEX no requiere que TCP y IP estén presentes en la pula, pero el fabricante es libre de implementar OBEX sobre TCP/IP.

- Bluetooth Network Encapsulation Protocol (BNEP): Es una capa en la pila Bluetooth que permite a otros protocolos de red ser transmitidos sobre Bluetooth, por ejemplo Ethernet. Encapsula paquetes TCP/IP en paquetes L2CAP antes de transmitirlos por la capa L2CAP.

- Human Interface Device Protocol (HID): Es otro protocolo adoptado y proveniente de la especificación SUB. Lista las reglas y guías para transmitir información desde y hacia interfaces humanas como teclados o mandos de consolas.

El lector interesado podrá encontrar más información sobre las capas de la pila de protocolos Bluetooth en el libro Bluetooth application programming with de Java APIs [13] y en la especificación Bluetooth [7].

2.4.4 Perfiles Bluetooth

Un perfil Bluetooth define maneras estándar de usar protocolos y sus características para un uso en particular. Un dispositivo Bluetooth puede soportar uno o más de los 4 perfiles Bluetooth existentes **[10]**, que podemos apreciar en la ilustración 11.

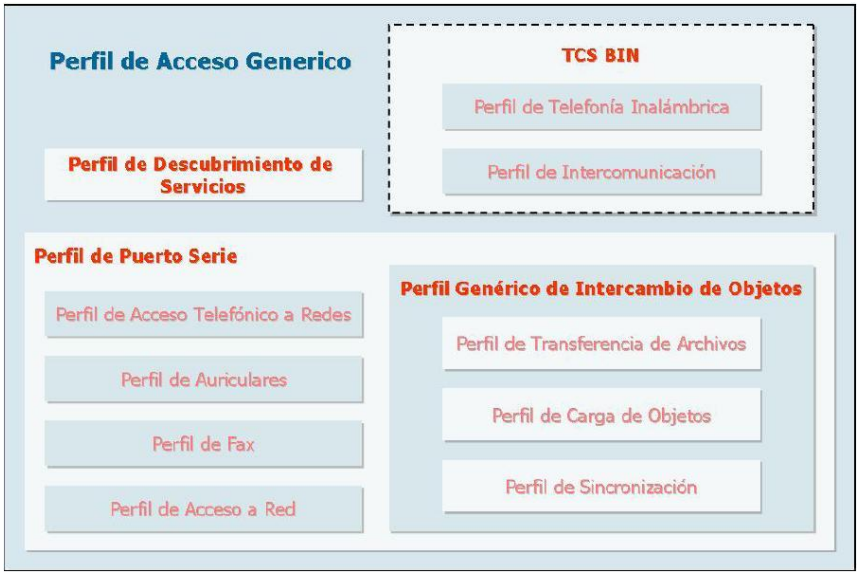


Ilustración 11: Perfiles Bluetooth

- Perfil de Acceso Genérico (Generic Access Profile-GAP): define cuales son los procedimientos básicos para descubrir dispositivos Bluetooth, establecer conexión entre ellos y de seguridad.

  Este perfil debe implementarse en todos los dispositivos Bluetooth para garantizar su interoperabilidad básica y coexistencia con otros dispositivos; y en caso de que el dispositivo soporte otro perfil con adaptaciones de los procedimientos genéricos, estos han de ser compatibles con el perfil GAP

- Perfil Puerto Serie (Serial Port Profile-SPP): este perfil se construye sobre el perfil GAP y define los requerimientos necesarios para que los dispositivos Bluetooth configuren conexiones emuladas de cable serie usando RFCOMM entre dos dispositivos.

- Perfil de Intercambio de Objetos Genéricos (Generic Object Exchange Profile-GOEP): perfil que define como deben soportar los dispositivos Bluetooth los modelos de uso de intercambio de objetos y se utiliza con OBEX.

- Perfil de Aplicación de descubrimiento de Servicios (Service Discovery Aplication Profile-SDAP): describe las operaciones necesarias para descubrir servicios registrados en otros dispositivos y obtener información de los mismos, mediante el protocolo SDP.

2.4.5 Establecimiento de la conexión

El primer paso para establecer una conexión es conocer la dirección del dispositivo con el que nos queremos conectar y que se obtiene, como veremos más adelante, realizando una Inquiry; para así poder enviarle un mensaje de petición de conexión llamado page. Antes de establecer la conexión, todos los dispositivos se encuentran en modo standby.

En standby, el dispositivo se despierta cada 1.28 segundos para escuchar, por una de las 32 frecuencias de salto definidas, posibles mensajes page/inquiry. Un mensaje page, se envía en todas las frecuencias posibles, primero se mandará por las 16 primeras frecuencias (128 veces), y si no se recibe respuesta, se intentará por las 16 frecuencias restantes (128 veces). Por lo que el tiempo máximo de intento de conexión es de 2.56 segundos.

Un dispositivo Bluetooth en estado conectado puede encontrarse en varios modos de operación:

- Active mode: En este modo, el dispositivo Bluetooth participa activamente en el canal.

- Sniff mode: En este modo, se reduce el tiempo de actividad durante el cual el dispositivo esclavo escucha. Esto significa que el maestro sólo puede iniciar una transmisión en unos slots de tiempo determinados.

- Hold mode: En el estado conectado, el enlace con el esclavo puede ponerse en espera. Mientras permanezca en este modo, el esclavo puede hacer otras cosas, como escanear en busca de otros dispositivos, atender otra Piconet, etc.

- Park mode: puede ocurrir que el esclavo no necesite participar en la Piconet, pero quiera seguir sincronizado con el canal, pero sin ser miembro de la Piconet. Esto es útil por si hay más de siete dispositivos que necesitan participar ocasionalmente en la Piconet.

2.4.6 Seguridad Bluetooth

Bluetooth es uno de los protocolos de comunicaciones más seguros y robustos frente a ataques y capturas de datos. Define mecanismos de seguridad en las siguientes capas de protocolos:

2.4.6.1 Seguridad a nivel de banda base

Durante el proceso de establecimiento de conexión de una Piconet, el dispositivo maestro genera una tabla pseudoaleatoria, basada en su reloj interno, con la secuencia o patrón de saltos de frecuencia, que deben utilizar los elementos pertenecientes a la Piconet durante las comunicaciones **[15]**. El maestro enviará esta secuencia a los esclavos por un canal determinado del espectro para que todos los dispositivos puedan acceder a ella.

Cuando se establece la Piconet, el dispositivo esclavo recibe un paquete FHS (Frequency Hop Synchronization) mediante el cual sincronizará su reloj interno con el del dispositivo maestro agregando un desplazamiento a su reloj interno. Posteriormente, durante la comunicación, se actualizarán estos desplazamientos puesto que el reloj de cada dispositivo funciona de forma independiente.

Una vez comenzada la comunicación, el intercambio de paquetes de datos se realiza de acuerdo con el patrón de saltos de frecuencia establecido y a una velocidad marcada por el reloj interno. Esto significa que en cada instante de tiempo cada dispositivo escribirá o escuchará durante su timeslot en un determinado canal del espectro.

Cualquier dispositivo ajeno que no pertenezca a la Piconet no podrá participar en la comunicación enviando paquetes o escuchando tráfico, ya que no dispone de la tabla con la secuencia de saltos utilizada en la misma y, además, la probabilidad de adivinar cuál de todos los canales puede ser empleado para la comunicación en cada instante de tiempo es mínima. En definitiva, la técnica de saltos de frecuencia empleada por Bluetooth garantiza, en principio, la participación exclusiva de dispositivos autorizados en una Piconet y una comunicación libre de escuchas por parte de usuarios ajenos a la misma.

2.4.6.2 Seguridad a nivel de enlace

Todas las funciones de seguridad de nivel de enlace están basadas en el concepto de clave de enlace, que es un número pseudoaleatorio de 128 bits almacenado individualmente por cada par de dispositivos Bluetooth. A nivel de enlace se definen tres mecanismos de seguridad:

- Autenticación: es el proceso por el cual un dispositivo Bluetooth verifica su identidad en otro dispositivo para poder acceder a los servicios que ofrece.

  La autenticación implica un esquema de desafío/respuesta entre cada par de dispositivos que emplea una clave de enlace secreta común. Se utiliza para autentificar dispositivos, no usuarios por lo que requiere de la intervención de estos. Una vez que los dispositivos emparejados disponen de la clave de enlace, la utilizan para autentificarse automáticamente en las sucesivas conexiones.

  La primera vez que dos dispositivos Bluetooth intentan comunicarse, se utiliza el proceso de emparejamiento (pairing) para crear una clave de enlace común de forma segura, para lo cual el usuario de cada dispositivo deberá introducir un código de seguridad de hasta 16 bytes de longitud que debe coincidir **[15]**.

  La especificación de Bluetooth establece que si se produce un fallo durante el proceso de autenticación, y para prevenir que un atacante pruebe claves de enlace aleatorias en un ataque de fuerza bruta, debe transcurrir cierto período de espera antes de que se pueda llevar a cabo un nuevo intento. Para

cada sucesivo intento fallido, el tiempo de espera aumenta exponencialmente.

- Autorización: es el procedimiento que determina los derechos que tiene un dispositivo Bluetooth para acceder a los servicios que ofrece un sistema. Se lleva a cabo mediante niveles de confianza que determinan la capacidad de acceso de un dispositivo a los servicios:

    - Confianza total: el dispositivo de confianza, mantiene una relación de emparejamiento y tiene acceso sin restricciones a todos los servicios.

    - Confianza parcial o restringida: el dispositivo mantiene una relación de emparejamiento pero tiene acceso restringido a uno o varios servicios.

    - Confianza nula: el dispositivo no confiable, no mantiene una relación de emparejamiento y no se le permite el acceso a ningún servicio.

  Cuando un dispositivo de confianza trata de acceder a un servicio autorizado lo hace de forma transparente, no se requiere ningún procedimiento de confirmación. Mientras que si un dispositivo de confianza restringida o no confiable intenta acceder a un servicio restringido, se requiere un procedimiento explícito de confirmación por parte del usuario para permitir o denegar el acceso a ese dispositivo.

  En la mayoría de dispositivos es posible marcar manualmente los dispositivos emparejados como dispositivos de confianza, para evitar la confirmación en cada intento de conexión.

- Cifrado de datos: protege la información que se transmite, garantizando la confidencialidad del mensaje transmitido, de forma que si el paquete es capturado por un usuario que no posea la clave de descifrado, el mensaje le resultará ininteligible.

  El cifrado es opcional, pero necesita que se haya producido anteriormente una autenticación. El maestro y el esclavo deben ponerse de acuerdo en utilizar cifrado o no. En caso afirmativo, deben determinar el tamaño de la clave de cifrado y si no llegan a un acuerdo, se les comunicará que no pueden utilizar cifrado en el enlace durante la comunicación.

  Tras esta negociación comienza el proceso de cifrado, el dispositivo maestro genera una clave de cifrado y transmite los datos cifrados deteniendo temporalmente el tráfico de datos de los niveles superiores para evitar la recepción de datos corruptos **[15]**.

2.5 API JSR-82

La especificación JSR 82 [7], estandariza el desarrollo de aplicaciones Bluetooth usando Java, ocultando la complejidad del protocolo Bluetooth detrás de unos APIs que permiten centrarse en el desarrollo en vez de los detalles de bajo nivel.

Por lo tanto, el objetivo de esta especificación es definir un API estándar abierto que se pueda utilizar en todos los dispositivos que implementen J2ME usando los APIs J2ME y el entorno de trabajo CLDC/MIDP.

Los APIs JSR 82 son muy flexibles, y permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth. El API intenta ofrecer las siguientes capacidades:

- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones RFCOMM, L2CAP y OBEX entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

Los APIs Java para Bluetooth definen dos paquetes que dependen del paquete CLDC javax.microedition.io: javax.bluetooth (que utilizaremos para implementar la aplicación y veremos en profundidad posteriormente) y javax.obex (que permite la comunicación mediante el protocolo OBEX).

2.5.1 Inicialización de la pila.

Los dispositivos Bluetooth que implementen este API pueden permitir que múltiples aplicaciones se ejecuten concurrentemente. El Bluetooth Control Center (BCC) es un conjunto de capacidades que permiten al usuario o al OEM (Original Equipment Manufacturer) resolver peticiones conflictivas de aplicaciones definiendo unos valores específicos para ciertos parámetros de la pila Bluetooth; previniendo así que una aplicación pueda perjudicar a otra que se ejecuta simultáneamente.

La pila Bluetooth es la responsable de controlar el dispositivo, por lo que en primer lugar será necesario inicializarla siguiendo un número de pasos que preparan el dispositivo para la comunicación inalámbrica. Sin embargo, la especificación no define la implementación del BCC, delegando esta tarea en los fabricantes, por lo que cada uno maneja la inicialización de la pila de una manera diferente. Un ejemplo de inicialización de la pila puede ser:

```
...
// Configuramos el puerto
BCC.setPortNumber("COM1");
// Configuramos la velocidad de la conexión
BCC.setBausRate(50000);
//Configuramos el modo conectable
BCC.setConnectable(true);
//Configuramos el modo discovery a LIAC
BCC.setDiscoverable(DiscoveryAgent.LIAC);
...
```

2.5.2 Paquete javax.bluetooth

Este paquete provee la funcionalidad para la realización de búsquedas de dispositivos, búsquedas de servicios y comunicación mediante flujos de datos o arrays de bytes. Es el paquete que utilizaremos para implementar el cliente y el servidor Bluetooth de la aplicación.

2.5.2.1 Cliente Bluetooth

Los dispositivos móviles Bluetooth, necesitan un mecanismo para encontrar, conectar y obtener información de las características de otros dispositivos. Un cliente Bluetooth debe realizar las siguientes operaciones para comunicarse con un servidor Bluetooth:

2.5.2.1.1 Descubrimiento de Dispositivos (Device discovery)

Es el primer paso que ha de realizar un cliente, para ello utiliza un objeto de la clase DiscoveryAgent, que nos permitirá realizar y cancelar búsquedas de dispositivos y de servicios. Para obtener el DiscoveryAgent necesitaremos el objeto LocalDevice, que representa al dispositivo en el que se está ejecutando la aplicación y permite obtener información del mismo (nombre del dispositivo, dirección Bluetooth, modo de conectividad,…).

Por lo tanto, el cliente podrá obtener una lista de los dispositivos que se encuentran dentro del rango de alcance del dispositivo, usando los siguientes métodos del DiscoveryAgent **[10]**:

- startInquiry() (no bloqueante): para poder utilizarla es necesario que la aplicación tenga un listener especificado, al que se notificará los nuevos dispositivos que se encuentren tras lanzar el proceso de búsqueda.

- retrieveDevices() (bloqueante): se utiliza cuando queremos omitir la búsqueda de dispositivos; pasando así directamente a la búsqueda de servicios sobre una lista de dispositivos favoritos, o que han sido encontrados en búsquedas anteriores. Es más rápido, ya que para confeccionar la lista no realiza la búsqueda de dispositivos, pero sin embargo no ofrece garantías técnicas de que los dispositivos estén a nuestro alcance.

Clases del Device Discovery:

- Interfaz javax.bluetooth.DiscoveryListener: Se usa para encontrar dispositivos, permite especificar un evento en el listener que reaccione ante eventos de búsqueda. Cada vez que se encuentra un dispositivo en un proceso de búsqueda se llama al método deviceDiscovered() pasándole un argumento de tipo RemoteDevice. Cuando el proceso de búsqueda se completa o se cancela, se llama al método inquiryCompleted(), que, en función de cómo finalice, recibe un argumento de tipo entero, que puede ser:

    - DiscoveryListener.INQUIRY_COMPLETED: si la búsqueda concluyó con normalidad.

    - DiscoveryListener.INQUIRY_ERROR: si se produjo un error durante el proceso de búsqueda.

    - DiscoveryListener.INQUIRY_TERMINATED: si la búsqueda se cancela manualmente.

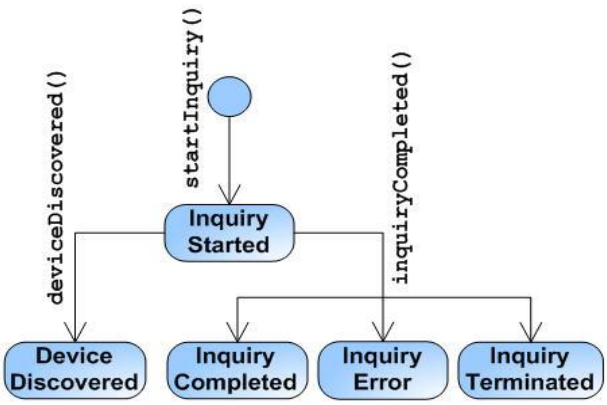


Ilustración 12: Diagrama de estado del Device Discovery

- Interfaz javax.bluetooth.DiscoveryAgent: compuesto por métodos muy útiles para descubrir dispositivos y servicios, como el método startInquiry() con el que el dispositivo notifica eventos a la aplicación cada vez que se descubre un dispositivo.

  El método retrieveDevices() devuelve un array de objetos RemoteDevice, cada uno de los cuales representa a un dispositivo remoto del que podremos obtener propiedades como su nombre o su dirección Bluetooth. La lista de dispositivos que obtendremos variará en función del argumento de tipo entero que se le pase al método:

    - DiscoveryAgent.PREKNOWN: para obtener una lista de dispositivos encontrados en búsquedas previas.

- DiscoveryAgent.CACHED: para obtener una lista de dispositivos pre-conocidos o favoritos, es decir, aquellos dispositivos definidos en el BCC con los que el dispositivo local contacta con frecuencia.

O el método cancelInquiry(), que permite cancelar una operación de búsqueda.

2.5.2.1.2 Descubrimiento de Servicios (Service Discovery)

El siguiente paso es descubrir los servicios disponibles en los dispositivos servidores encontrados, para saber si alguno de estos dispositivos ofrece el servicio en el que está interesado el cliente. Cada servicio tiene asignado un identificador numérico denominado UUID (Universal Unique IDentifier) así como varios atributos que lo describen y que también están identificados numéricamente. La clase DiscoveryAgent proporciona métodos para buscar servicios en un dispositivo servidor Bluetooth e iniciar transacciones entre el dispositivo y el servicio **[10]**.

Clases del Service Discovery:

- Clase javax.bluetooth.DiscoveryAgent: Esta clase provee métodos para descubrir servicios y dispositivos.

- Clase javax.bluetooth.UUID: Esta clase encapsula enteros sin signo que pueden ser de 16, 32 ó 128 bits de longitud. Estos enteros se usan como un identificador universal cuyo valor representa un atributo del servicio. Sólo los atributos de un servicio representados con UUID están representados en la capa de protocolos Bluetooth SDP.

- Clase javax.bluetooth.DataElement: Esta clase encapsula los tipos de datos en los que puede ser representado un atributo de servicio; y pueden ser: números enteros de diferente longitud con o sin signo, cadenas de texto, URLs, booleanos o secuencias de cualquiera de los tipos anteriores. Esta clase además presenta una interfaz que permite construir y recuperar valores de un atributo de servicio.

- Interfaz javax.bluetooth.ServiceRecord: Este interfaz define el Service Record de Bluetooth, que es como una tabla que relaciona los identificadores de los atributos (atributo ID) con sus valores. El atributo ID es un entero sin signo de 16 bits, y de tipo DataElement. Además, este interfaz tiene un método populateRecord() para recuperar los atributos de servicio deseados (pasando como parámetro al método el ID del atributo deseado).

- Interfaz javax.bluetooth.DiscoveryListener: Este interfaz permite a una aplicación especificar un listener que responda a un evento del tipo Service Discovery que obtendremos al realizar la búsqueda de servicios con el método searchServices(). Cuando un nuevo servicio es descubierto, se llama al método servicesDiscovered() y se nos pasará un array de objetos ServiceRecord que encapsulan los atributos del servicio que solicitamos y cuyo valor es, a su vez, objetos DataElement. Y cuando la transacción ha

sido completada o cancelada se llama a serviceSearchCompleted() que nos devolverá uno de estos argumentos de tipo entero:

- SERVICE_SEARCH_COMPLETED: La búsqueda de Servicios ha finalizado con normalidad.
- SERVICE_SEARCH_TERMINATED: La búsqueda de servicios ha sido cancelada manualmente.
- SERVICE_SEARCH_NO_RECORDS: no existe la información solicitada.
- SERVICE_SEARCH_ERROR: la búsqueda de servicios finalizó por un error.
- SERVICE_SEARCH_DEVICE_NOT_REACHABLE: el dispositivo no está a nuestro alcance.

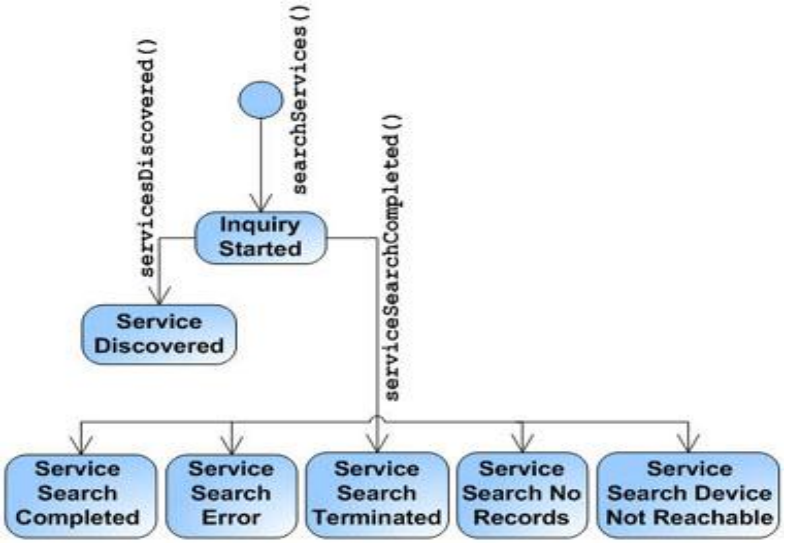


Ilustración 13: Diagrama de estado del Service Discovery

2.5.2.1.3 Establecimiento de la conexión y comunicación

Una vez se encuentra un servicio que nos interesa en alguno de los dispositivos, el siguiente paso es abrir la conexión. Para ello utilizaremos el método open() de la clase javax.microedition.Connector, que requiere como argumento una URL. A su vez, para obtener la URL que contendrá los datos necesarios para realizar la conexión emplearemos el método getConnectionURL() del objeto ServiceRecord, pasándole dos argumentos **[10]**. El primer argumento indica si se debe autenticar y/o cifrar la conexión y puede tomar estos valores:

- ServiceRecord.NOAUTHENTICATE_NOENCRYPT: no requiere autenticación ni cifrado.
- ServiceRecord.AUTHENTICATE_NOENCRYPT: se requiere autenticación pero no cifrado.

- ServiceRecord.AUTHENTICATE_ENCRYPT: se requiere autenticación y cifrado.

El segundo argumento es un booleano que especificará si nuestro dispositivo debe hacer de maestro o si no importa que haga de maestro o de esclavo.

Hay dos formas de comunicación posibles:

- Flujo de datos: utilizando el protocolo SPP, es la más sencilla.
- Array de bytes: a través del protocolo L2CAP.

En nuestro caso el servidor Bluetooth utiliza SPP por lo que el método open() nos devolverá un objeto de tipo javax.microedition.io.StreamConnection a partir del cual podremos obtener los objetos DataInputStream y DataOutputStream con los que establecer un flujo de lectura/escritura de datos.

2.5.2.2 Servidor Bluetooth

La implementación del Servidor Bluetooth es más sencilla que la del cliente puesto que no es necesario realizar ningún tipo de búsqueda. Los pasos que ha de realizar un servidor Bluetooth son:

2.5.2.2.1 Crear una conexión servidora

Para crear la conexión llamaremos al método connector.open() pasándole como argumento una URL con una sintaxis determinada **[12]**. En nuestro caso como nos vamos a comunicar mediante SPP la URL debe comenzar por "btspp://", a continuación se indicará el host y como queremos ser servidor este será "localhost", por último se concatenará el UUID del servicio que vamos a ofrecer y opcionalmente se podrán añadir otros parámetros como, por ejemplo, el requerimiento o no de autenticación. Este método nos devolverá un notifier que en nuestro caso al ser SPP, se tratará de un objeto de tipo javax.microedition.StreamConnectionNotifier, que escuchará las conexiones cliente.

2.5.2.2.2 Especificar atributos de servicio

Una vez creado el Notifier se tendrán que especificar los atributos del servicio, que estarán almacenados en un ServiceRecord y que podemos obtener a través del método getRecord() de la clase LocalDevice. Para establecer el atributo en el ServiceRecord utilizaremos el método setAttributeValue() al que se pasarán como argumentos un identificador numérico y un objeto DataElement que representa el valor del atributo de servicio. De forma que cada conexión servidora tendrá un ServiceRecord asociado.

2.5.2.2.3 Abrir conexiones cliente

Una vez establecidos los parámetros del servicio podemos escuchar y procesar las conexiones cliente usando el método acceptAndOpen() que en nuestro caso, al

tratarse de una conexión servidora SPP devolverá un objeto javax.microedition.StreamConnection. con el que podremos leer y escribir datos igual que lo hace el cliente **[10]**.

Un servicio run-before-connect (se ejecuta sin esperar a que haya una conexión establecida) invocará a este método y el registro del servicio es añadido a la base de datos de servicios descubiertos (SDDB-Service Discovery Database). En el caso de que se hagan modificaciones en el ServiceRecord para realizarlas también en la SDDB deberemos usar el método updateRecord(). Finalmente el ServiceRecord es eliminado de la SDDB cuando la aplicación servidora manda un close() al notifier.

Desde el punto de vista de conexión, un dispositivo debe hallarse en uno de estos dos modos, en función de cual escoja el usuario:

- Modo conectable: el dispositivo escucha periódicamente intentos de iniciar una conexión de un dispositivo remoto.

- Modo no-conectable: el dispositivo no escucha intentos de iniciar una conexión de un dispositivo remoto.

Para que una aplicación servidora funcione correctamente, es necesario que el dispositivo servidor esté en modo conectable. Por ello, el método acceptAndOpen() deberá asegurar que el dispositivo local esté en modo conectable realizando una petición al BCC. En caso de que no sea posible poner al dispositivo en modo conectable se lanzará una excepción de tipo BluetoothStateException. En caso de que se eliminen o deshabiliten todos los servicios en la SDDB, la aplicación puede, opcionalmente, pedir al dispositivo servidor que pase a modo no conectable.

Capítulo III

Especificación

En general, para el desarrollo de cualquier aplicación, es aconsejable seguir tres pasos: especificación de los requisitos, desarrollo de la aplicación y pruebas del sistema. El objetivo es conseguir realizar la aplicación de una forma más sencilla y correcta en el menor tiempo posible. En primer lugar es necesario especificar claramente los requisitos que debe cumplir la aplicación con el objetivo de tener bien claro lo que se quiere hacer.

Este capítulo se centra en el primer paso, especificar los requisitos que debe cumplir la aplicación. Además, se incluirá un apartado con las herramientas que se aconseja usar para el desarrollo.

3.1 Objetivo final

El objetivo final de este Proyecto es la creación de una aplicación móvil, que permita a un usuario que desee obtener información, al que a lo largo del proyecto llamaremos "alumno", realizar búsquedas de otros terminales que presten este mismo servicio, y cuyos propietarios estén dispuestos a entablar una conversación, a los que denominaremos como "tutores" en esta memoria.

De esta forma, si el usuario opta por el rol de alumno podrá, de una forma fácil y cómoda comprobar en todo momento y en cualquier lugar si hay algún tutor cerca que tenga conocimientos sobre una materia. De ser así podrá seleccionar un tutor con el que establecer una conexión, y tratar de resolver con su ayuda a través de un chat las dudas que tenga. O también puede optar por el rol de tutor y prestarse de esta forma a ayudar a otros usuarios.

3.2 Requisitos

Para mejorar la funcionalidad, y sacar el máximo partido a la aplicación se llevó a cabo un estudio en una primera fase de análisis, a través del cual se definieron una serie de especificaciones que deben cumplirse.

Cuando se utilice la aplicación por primera vez, será necesario rellenar un formulario, en donde el usuario (tanto el alumno como el tutor) tendrá que autoevaluar "sinceramente" sus conocimientos en las distintas asignaturas cursadas. Esto no le llevará al usuario más de cinco minutos, y la información se almacenará en el terminal y se utilizará, en el caso de elegir posteriormente el rol de alumno, para agilizar el proceso de búsqueda de un tutor; o en el caso de escoger el rol de tutor, para validarse o descartarse como posible tutor de los alumnos solicitantes.

Posteriormente cada vez que el usuario entre en la aplicación será informado de cuáles son las carreras para las que ha valorado su conocimiento en alguna de sus asignaturas, y se le permitirá realizar las siguientes acciones:

- Modificar las evaluaciones de las asignaturas que escoja, de cada una de las carreras que tenía registradas.

- Añadir una nueva carrera, para poder evaluar alguna asignatura que ha cursado de la misma.

- Borrar una carrera, en el caso de que considere no tener conocimientos sobre ninguna de sus asignaturas.

Por esta razón un usuario que no tenga registros de ninguna carrera almacenados (bien porque no haya querido evaluar sus conocimientos de ninguna asignatura de al menos una carrera, o porque los haya borrado) no podrá utilizar la aplicación. Hay que resaltar que cuanto mayor sea el grado de sinceridad con el que se autoevalúen los usuarios, mayor será la ayuda que les prestará la aplicación, tanto para encontrar un tutor que cumpla con unas condiciones, como para ser de ayuda a los alumnos que lo soliciten como tutor.

La aplicación, ha de preguntar al usuario que rol quiere desempeñar: el de alumno (o cliente), en el caso de que quiera encontrar un tutor que le ayude a resolver alguna duda; o el de tutor (o servidor) en el caso de que no le importe estar disponible para resolver dudas a otros alumnos. En ningún caso un alumno podrá ayudar a otro a resolver una duda.

Gracias a los datos almacenados en la aplicación (evaluaciones), el tutor podrá verificar si cumple las condiciones impuestas por el alumno para ayudarle a resolver sus dudas. Mientras que el alumno, podrá establecer unos criterios de filtrado en la búsqueda de tutores de una asignatura, que le ayuden a encontrar eficientemente y rápidamente una fuente de información capaz de responder a sus preguntas. Los criterios de filtrado que podrá utilizar un alumno en la búsqueda de tutores serán:

- Tutores con nota superior a un valor.
- Tutores con nota igual a un valor.
- Cualquier tutor.
- Tutores con nota superior a mi nota.

Una vez finalizada la búsqueda de tutores se mostrará al alumno lo que denominaremos como “lista de posibles tutores” compuesta por aquellos que se encuentran disponibles y cumplen las condiciones impuestas por el alumno. De entre ellos, el alumno seleccionará aquel con el que quiere establecer una conversación a través de lo que denominaremos de ahora en adelante como “chat”.

En caso de que algún otro alumno se le adelantase y estableciera con ese tutor un chat, la aplicación debe de ser lo suficientemente flexible y eficaz para advertir al alumno que dicho tutor está ocupado y darle a elegir de entre los tutores que aún están disponibles.

Si un tutor, decide abandonar su rol (bien porque quiera adoptar el rol de alumno, o sencillamente porque haya cerrado la aplicación) y formaba parte de la lista de posibles tutores de uno o varios alumnos, la aplicación debe gestionar esta baja y eliminar este tutor de las listas implicadas, manteniéndolas de esta forma actualizadas en todo momento.

Puede ocurrir, por tanto, que una vez confeccionada la lista de posibles tutores el número de estos decrezca, bien porque ya no estén disponibles por estar conversando con otro alumno; o porque haya abandonado su rol como tutor, por ejemplo cerrando la aplicación. En el caso de que no quedase ningún tutor en la lista, la aplicación debe

advertir al usuario que ya no hay ningún tutor disponible que cumpla las condiciones impuestas, y le ofrecerá la posibilidad de realizar una nueva búsqueda, ya que durante su confección es el único momento en el que el número de tutores disponibles puede incrementar.

Como máximo, se decidió que el alumno fuese capaz de establecer conexión con 100 tutores disponibles, ya que se consideró un número suficientemente grande para que el alumno pudiese encontrar un tutor apto que le ayudase entre ellos.

Una vez el alumno ha elegido un tutor de la lista y se ha establecido el chat entre ambos, el tutor pasará a un estado que denominaremos como "ocupado o "no disponible", lo cual no afectará en modo alguno a la conversación establecida. Pero para el resto de alumnos esto implicará, que no podrán localizar a este tutor en sus búsquedas a pesar de que cumpla las condiciones. Y en caso de que algún alumno ya lo hubiera localizado y formase parte de su lista de posibles tutores, desaparecerá de esta.

El tutor permanecerá en ese estado "ocupado" hasta que alguna de las dos partes implicadas de por finalizado el chat, y el tutor cierre la ventana de la conversación; momento en el que volverá a estado "disponible". Este requisito es importante, ya que aunque el objetivo de la aplicación es ayudar a los alumnos a resolver sus dudas, puede ocurrir que durante el proceso el tutor también solucione alguna cuestión que tuviera pendiente. Por lo que, en caso de que fuera el alumno quien diese por finalizada la comunicación cerrando el chat, no sería justo que otro alumno pudiese abrir un nuevo chat con este tutor antes de que le hubiese dado tiempo al tutor a tomar nota de la última conversación mantenida.

Por lo tanto, siempre que una de las dos partes implicadas en la comunicación (tutor y alumno) cierre el chat, la aplicación se encargará de gestionar el cierre de la conexión y de avisar al otro extremo de que la comunicación ha concluido, mostrándole un historial de la conversación que ha mantenido.

Por último, solo comentar que como las dudas suelen surgir a los alumnos durante el estudio, y no es conveniente el uso de móviles en la biblioteca, aulas de estudio,… Para tratar de molestar lo menos posible a las personas de alrededor se decidió que la mejor manera para avisar al usuario que ha recibido un mensaje mientras está conectado a través del chat es mediante la función de vibración del móvil.

3.3 Tecnologías y Herramientas

A continuación se proporciona una lista con las herramientas necesarias para llevar a cabo el desarrollo de la aplicación.

- Plataforma de desarrollo Java (J2SDK, por ejemplo) disponible en java.sun.com.

- Compilador CLDC (1.0, 1.1) y perfil MIDP, ambos disponibles en java.sun.com.

- Editor de textos (block de notas, edit, vi, emacs, etc.).

- Java Wireless Toolkit: Este paquete se puede descargar del sitio Java de Sun Microsystems, provee de un compilador (incluye CLDC y MIDP) y organizador de proyectos. Se crea en este programa un proyecto y automáticamente se generan las carpetas donde se han de guardar los códigos fuente, las clases, imágenes y otros accesorios que la aplicación utilice. Disponible para Windows y Linux.

Capítulo IV

Diseño

Una vez que se tienen claros los requisitos, el siguiente paso es el desarrollo de la aplicación. Al comenzar a desarrollar es conveniente realizar un diseño de alto nivel para tener una primera aproximación de cómo se va a solucionar el problema. Dada la importancia del diseño dentro de la parte de desarrollo, ya que es la base para comenzar la implementación, se ha decidido dedicar un capítulo propio a este apartado.

Este capítulo se dedica al diseño de la solución que se da al problema planteado. Destacar que es un diseño de alto nivel y que en el capítulo de implementación será donde se entre en detalles. En primer lugar se incluye un diseño de bloques en el que se determina qué partes componen la aplicación y qué función realiza cada una. A continuación se tiene un apartado en el que se comenta brevemente la idea de cómo se va a implementar cada operación. Por último se incluye un apartado en el que se da una breve introducción a las herramientas que se van a usar en la implementación.

4.1 Diseño de Bloques

Ahora que se tiene claro qué es lo que hay que hacer, el siguiente paso será realizar un diseño de alto nivel de lo que se va a implementar. Se va a realizar un esquema de bloques indicando los componentes de la aplicación y las relaciones entre ellos.

La idea principal es que el usuario configure la aplicación, introduciendo los datos necesarios para que esta, mediante tecnología Bluetooth, pueda realizar una búsqueda de terminales móviles que presten este servicio; y a bajo nivel establecer comunicación con ellos e intercambiar información. De esta forma la aplicación será capaz de seleccionar a aquellos tutores que puedan serle de utilidad al usuario y confeccionar con ellos una lista para que el usuario escoja a uno, con el cual se establecerá una comunicación en forma de chat a través del cual pueda resolver las dudas que tenga.

En la ilustración 14 se muestra el esquema con los distintos bloques funcionales que van a formar la aplicación. Este diagrama de bloques es a alto nivel, ya que no profundiza hasta la identificación de qué clases componen cada nivel. Cada bloque de los que aparecen en la figura tiene una función dentro de la lógica de la aplicación. A continuación se enumeran las partes funcionales de la aplicación describiendo brevemente la función que cumple cada una:

- **Bloque de interfaz con el usuario:** este bloque es el encargado de interaccionar con el usuario. Mostrará un formulario al usuario para que éste autoevalué sus conocimientos; le dejará introducir modificaciones en estos registros, crear o borrar registros de otras carreras; y también seleccionar un rol (alumno o tutor).

  En caso de escoger el rol de alumno, le permitirá establecer las condiciones para la búsqueda de tutores; escoger su tutor de una lista confeccionada con todos los tutores disponibles; y por último introducir mensajes que serán enviados a dicho tutor, así como visualizar los que reciba del mismo.

  Si por el contrario, el rol elegido es el de tutor, le mostrará los mensajes que reciba de aquel alumno que le haya elegido como tutor, y le permitirá enviar mensajes para ayudarle a aclarar sus dudas.

- **Bloque de registros:** su función es almacenar la información de interés en lo que denominaremos almacén de registros o repositorio; permitiéndonos acceder a estos datos siempre que lo necesitemos, así como realizar modificaciones, ampliarlos o borrarlos. De esta forma, cuando se cierre la aplicación no se perderán los datos introducidos quedando estos almacenados en la memoria interna del dispositivo y pudiéndose volver a recuperar posteriormente. Un ejemplo de la información que podremos

encontrar en el repositorio son las notas de las distintas asignaturas de las carreras con que el alumno se autoevalúa.

- **Bloque Core:** esta parte es el cerebro de la aplicación. Lleva la gestión de toda la aplicación y se puede subdividir en tres bloques:

    - *Bloque común:* se encarga de la gestión de la aplicación hasta que el usuario elije su rol. Deberá interaccionar con el Bloque de registros para ver si el usuario tiene algún registro almacenado. En caso de que no exista ninguno pedirá al usuario que cree uno con sus autoevaluaciones a través del interfaz con el usuario. Por el contrario si ya existe alguno le dará opción de modificarlo, crear otro o borrarlo también a través de estos dos interfaces. Por último, pedirá al usuario que escoja un rol para utilizar la aplicación mediante el interfaz con el usuario.

    - *Bloque cliente:* es el encargado de realizar todas las operaciones necesarias en el caso de que el usuario escoge el rol de alumno. Entre sus funciones destacan: acceder al repositorio de información mediante el interfaz de registros, interactuar con el bloque de comunicaciones para realizar la búsqueda de dispositivos Bluetooth que presten este servicio (tengan instalada esta aplicación) y hayan escogido el rol de tutor, enviarles las condiciones que ha de cumplir un dispositivo para ser tutor del cliente, confeccionar una lista con todos los tutores disponibles que cumplen estas condiciones impuestas por el alumno y mantenerla actualiza en todo momento para que el usuario escoja a uno como tutor mediante el bloque de interfaz con el usuario, y por último establecer una comunicación con ese dispositivo en forma de chat a través del bloque de comunicaciones de nuevo.

    - *Bloque servidor:* cuando el usuario elije el rol de tutor, será este bloque el que gestione todas las operaciones necesarias, como pueden ser: especificar los atributos del servicio que se va a ofrecer, crear una conexión servidora, abrir las conexiones cliente, responder a las condiciones recibidas de los alumnos todo ello mediante el bloque de comunicaciones, la gestión de la disponibilidad del tutor en todo momento, el acceso al repositorio, a través del bloque de registro, para comprobar si el tutor cumple con las condiciones recibidas del cliente, y en caso de resultar ser el tutor elegido por alguno de los clientes establecer con él una comunicación chat mediante el bloque de comunicaciones.

- **Bloque de comunicaciones:** este bloque será el responsable de establecer contacto con otros dispositivos Bluetooth, establecer una comunicación con ellos a bajo nivel y abrir una comunicación mediante un chat entre el alumno y el seleccionado por este. En este bloque distinguiremos dos partes claramente diferenciadas:

    - *Bloque de control:* se encargará de realizar aquellas comunicaciones invisibles al usuario que son necesarias entre los dispositivos como

son: las comunicaciones propias a la implementación del protocolo Bluetooth (búsquedas de dispositivos y servicios, registro de los atributos del servicio, establecimiento de conexión y de comunicación), el envío de las condiciones impuestas por el cliente para aceptar un servidor como tutor y la respuesta por parte del servidor a estas condiciones evaluándolas y clasificándose como un tutor apto o no apto para ese cliente.

- *Bloque de tráfico:* será el responsable de la comunicación, a través de un chat, que se establece entre el alumno y el tutor elegido, para ayudarle a resolver sus dudas. Será el bloque Core el que le dará entrada, una vez se han realizado todas las operaciones previas, invisibles al usuario, cerrando las conexiones mantenidas con el resto de tutores y pasándole los parámetros necesarios para la comunicación con el tutor escogido.

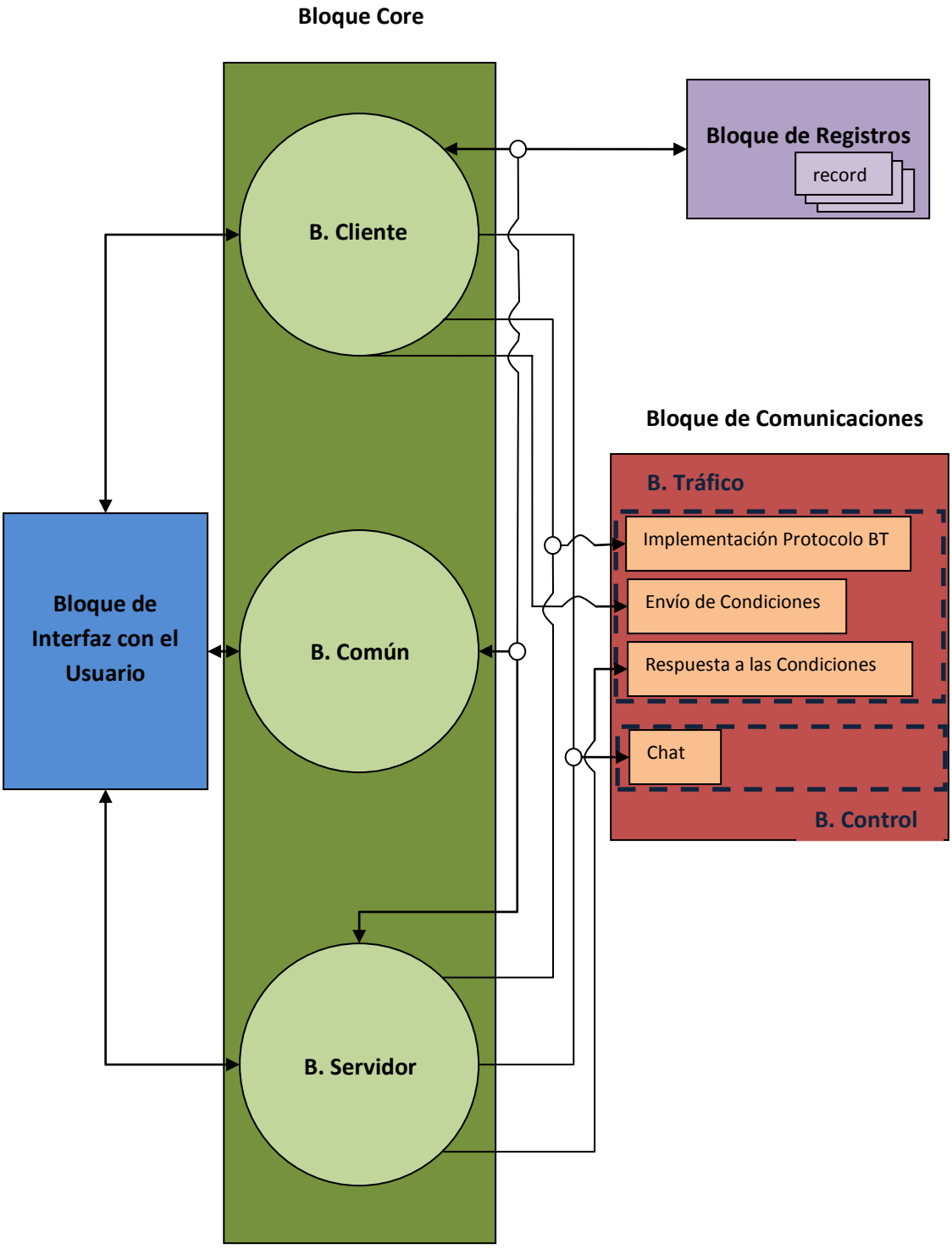


Ilustración 14: Esquema Aplicación

4.2 Diseño Lógico

En el apartado anterior se han explicado los distintos bloques funcionales que tendrá la aplicación. En este punto se va describir brevemente qué lógica se va a aplicar en la implementación de las operaciones que va a realizar la aplicación

Siguiendo los puntos que se piden en la especificación de requisitos, se pueden distinguir los siguientes apartados:

4.2.1 Autoevaluación

El usuario ha de evaluar sus conocimientos antes de empezar a utilizar la aplicación, posteriormente podrá acceder a los datos de sus evaluaciones y realizar diversas operaciones sobre estos datos.

4.2.1.1 Dar de alta un registro

La única operación obligatoria de la aplicación será que el usuario, la primera vez que acceda, seleccione al menos una carrera y evalúe, de 0 a 9, los conocimientos que posee en las diferentes asignaturas de la misma. El objetivo de esta operación es crear lo que denominaremos de ahora en adelante como un "registro", que la aplicación utilizará para ayudar al alumno a escoger un tutor apto, es decir, que cumpla las condiciones de búsqueda impuestas por el alumno.

Sin embargo, el usuario podrá pedir tutoría de una asignatura que no ha sido evaluada por él anteriormente, ya que se entiende que será con asignaturas nuevas con las que pueda necesitar más ayuda un alumno.

Cuando se termina el formulario oportuno, el usuario podrá almacenar los datos en el repositorio, o añadir un nuevo registro de otra carrera, lo que le permitirá evaluar sus conocimientos en las asignaturas de la misma de haber cursado alguna de ellas como, por ejemplo, asignaturas de libre elección.

No se guardara un registro de una carrera en el que no se haya evaluado al menos una asignatura de la misma con una nota distinta de 0 o null. Y en el caso de no tener al menos un registro almacenado no se podrá hacer uso de la aplicación.

4.2.1.2 Modificar un registro

Esta operación permitirá al alumno, cada vez que acceda a la aplicación, realizar modificaciones sobre los registros anteriormente almacenados, lo que implica una acción de lectura/escritura del repositorio. De esta forma el usuario podrá cambiar su autoevaluación o añadir valoraciones a asignaturas que anteriormente no había cursado, manteniendo así actualizados los registros. Lo ideal sería que el usuario realizará un examen cuatrimestral de autoevaluación para modificar con sinceridad sus conocimientos.

Una vez el usuario ha introducido los cambios deseados, la aplicación le permitirá guardar el registro y continuar con la aplicación (siempre y cuando no intente guardarlo con los valores de todas sus asignaturas a 0 o a null), o en el caso de que tuviera almacenado más de un registro le dará la opción de modificar cualquiera de ellos salvando automáticamente la aplicación los cambios realizados en el registro actual.

4.2.1.3 Añadir otra carrera

El objetivo de esta operación es dar al usuario la posibilidad de añadir cada vez que acceda a la aplicación nuevos registros de otras carreras, bien porque se haya matriculado en ellas, o porque quiera evaluar asignaturas optativas o de libre elección.

En el repositorio únicamente se podrá almacenar por cada carrera un registro, por lo que la aplicación deberá verificar que registros hay ya guardados, y dar a elegir al usuario las carreras que puede añadir. Por lo tanto también será necesario realizar acciones de lectura/escritura del repositorio.

Cuando el usuario evalúe las asignaturas de la nueva carrera que ha añadido, la aplicación le permitirá guardar el registro de esta nueva carrera o continuar añadiendo nuevas carreras, salvando los cambios realizados automáticamente.

4.2.1.4 Eliminar una carrera

La aplicación ofrece también al usuario la posibilidad de borrar registros almacenados anteriormente en el repositorio a través de esta operación. Esta operación está pensada para usuarios que cambien de carrera, o que consideren que ya no tiene sentido tener registros de otras carreras (evaluaciones de asignaturas de libre elección u optativas) por haber olvidado conceptos con el paso del tiempo.

Por ello, la aplicación mostrará al usuario aquellas carreras que tienen almacenados registros, y le permitirá borrar aquellas que desee; pudiendo llegar incluso a borrar todas antes de continuar. En caso de que el usuario borrase todos los registros la aplicación le recordará que para poder hacer uso de ella es necesario tener al menos un registro, y le remitirá al formulario para que elija su carrera y autoevalué sus conocimientos.

4.2.2 Selección del rol

Una vez tenemos almacenado al menos un registro, la aplicación nos solicitará que escojamos un rol, de una lista, para poder continuar. Hay dos roles posibles:

- Rol de tutor: el usuario debe elegir este rol, cuando desee servir de ayuda a otros usuarios que tengan dudas sobre materias de las que él dispone conocimientos. En caso de escoger este rol, la aplicación a ojos del usuario permanecerá a la espera, mientras que de forma invisible para el usuario, recibirá peticiones de conexión de otros dispositivos que hayan escogido el rol de alumno, intercambiará datos con ellos para ver si cumple las condiciones impuestas por el alumno. Y en caso de ser el tutor seleccionado

por alguno de estos dispositivos para ayudarle a solventar sus dudas, establecerá con él un chat de forma transparente al usuario para que puedan intercambiar información.

- Rol de alumno: Lo primero que debe hacer la aplicación en el caso de que el usuario escoja este rol, será gestionar que ningún otro alumno pueda localizar nuestro terminal. Ya que ahora será el usuario quien quiere utilizar la aplicación para encontrar un tutor que cumpla con unas condiciones impuestas por él mismo, de forma que le garantice en un porcentaje elevado que va a servirle de ayuda. Una vez localizados estos tutores, la aplicación establecerá una conexión con cada tutor para enviarles las condiciones, que estos validen si las cumplen, y le transmitan el resultado para poder confeccionar una lista con aquellos que así lo hagan. De forma que el usuario pueda seleccionar a cual prefiere de tutor para establecer con él una comunicación a través de un chat.

4.2.3 Condiciones de búsqueda

En el caso de que el usuario elija el rol de alumno, para que la aplicación le pueda ayudar a encontrar un tutor le pedirá una serie de parámetros:

- Carrera: en primer lugar, la aplicación mostrará una lista con todas las carreras del área de Telecomunicaciones impartidas en el campus politécnico, de Leganés, de la Universidad Carlos III de Madrid. Y el usuario tendrá que escoger aquella a la que pertenece la asignatura que busca. Ya que puede ocurrir que la misma asignatura tenga el mismo nombre en varias carreras y, sin embargo, estas no tengan el mismo temario.

- Asignatura: a continuación la aplicación mostrará una lista con el nombre de todas las asignaturas pertenecientes a esa carrera y el usuario deberá elegir aquella en la que necesita ayuda.

- Condición del Tutor: Por último, se le dará al usuario la posibilidad de imponer al tutor que cumpla una condición de entre las siguientes:

    - Tutores con nota superior a un valor: el usuario elegirá un valor y el tutor deberá haberse evaluado con una nota superior a ella. En caso de no introducir ningún valor se tomará por defecto el 0.

    - Tutores con nota igual a un valor: esta condición únicamente estará disponible en el caso de que el usuario tenga almacenado en el Record Store el registro de la carrera para la que precisa ayuda. Para que el tutor sea apto deberá tener una nota igual a la elegida por el alumno, y en caso de que este realice la búsqueda sin meter ninguna nota, se tomará por defecto 0.

    - Cualquier tutor: esta opción es equivalente a no hacer ningún filtrado en función de la nota del tutor. Mostrando al usuario todos los tutores disponibles que han evaluado (con un valor mayor a 0) esa asignatura.

- Tutores con nota Superior a mi nota: con esta opción, la condición necesaria para que un tutor sea apto será que se haya evaluado en esa asignatura con una nota superior a la evaluación que tiene el alumno almacenada.

Estos parámetros serán almacenados temporalmente para posteriormente enviárselos a los posibles tutores localizados y que estos valoren si sirven o no como tutores del alumno.

4.2.4 Búsqueda de dispositivos y servicios

Una vez fijados los parámetros para la búsqueda de un tutor, la aplicación empleará la tecnología Bluetooth del terminal móvil para realizar una búsqueda de otros terminales o dispositivos que también estén equipados con dicha tecnología.

Pero, como hemos explicado en el apartado *4.2.2 Selección del rol*, solo encontraremos a los dispositivos que han escogido el rol de tutor, puesto que la aplicación se encargará de gestionar que sea imposible localizar a los usuarios que hayan escogido el rol de alumno.

Podía haberse permitido esta opción, pero en el diseño de la aplicación se decidió como requisito que los alumnos únicamente puedan recibir formación de los tutores y no de otros alumnos. La razón fue que se consideró que cuando un usuario escoja el rol de alumno, lo hará motivado por resolver unas dudas en particular; y seguramente le molestaría que en ese momento otros usuarios le preguntasen sobre otras cuestiones diferentes, impidiéndole alcanzar su objetivo. Por ello, se tomó la decisión de que únicamente los tutores pudiesen resolver dudas a los alumnos.

Después de haber localizado a todos los tutores disponibles, la aplicación buscará en ellos, uno por uno, si tienen instalada esta aplicación o servicio, descartando a aquellos que no lo tengan; y establecerá conexión con el resto, hasta un máximo de 100 conexiones. De forma que pueda establecerse una comunicación entre el terminal del alumno y el de todos los posibles tutores.

El número de conexiones máximas permitidas a un alumno, se impuso de forma arbitrará en 100 por considerarlo un número lo suficientemente grande para que entre ese número de tutores disponibles el alumno pudiese encontrar al menos uno apto; y no supusiese una carga computacional demasiado elevada al dispositivo móvil.

Por otro lado hay que resaltar que un mismo tutor puede ser objetivo de varias búsquedas de dispositivos mientras esté disponible, por lo que debe de ser capaz de establecer múltiples conexiones simultaneas. Creará una conexión nueva por cada alumno que realice una búsqueda de dispositivos, para que así todos los alumnos que están buscando un tutor encuentren a todos los tutores.

4.2.5 Filtrado de Tutores

El siguiente paso será identificar de entre todos los tutores disponibles a aquellos que cumplen las condiciones impuestas por el tutor. Para ello, la aplicación instalada en el terminal del alumno enviara las condiciones de búsqueda a los terminales de los tutores a través de la conexión establecida en la búsqueda de dispositivos.

Una vez la aplicación instalada en el terminal del tutor reciba las condiciones de búsqueda, deberá comprobar si tiene algún registro almacenado de la misma carrera De ser así, comprobará con que nota tiene el tutor evaluada la asignatura que le interesa al alumno, para a continuación verificar si cumple con la condición impuesta por el mismo.

De esta forma, el tutor sabrá si cumple o no con las condiciones de búsqueda impuestas por el alumno, y podrá comunicarle a este el resultado. En caso de cumplir con las condiciones será un tutor "apto" y mantendrá la conexión con el terminal del alumno a la espera de saber si resulta finalmente elegido por este como tutor. Y en caso de resultar no apto, también se lo comunicará al alumno y después ambos cerrarán la conexión establecida.

4.2.6 Selección de tutor

Con toda la información recibida de los tutores, la aplicación del terminal del alumno, creará una lista, que denominaremos como "lista de posibles tutores"; que contendrá el "friendly name" de los terminales de los tutores aptos así como la nota con que sus usuarios evaluaron la asignatura de la carrera implicada.

El friendly name es el nombre que da un usuario a su terminal, para que otros lo identifiquen en una conexión Bluetooth.

La aplicación se encargará de mantener actualizada esta lista con la información que reciba de los tutores aptos disponibles, de forma que si un tutor abandona su rol, cierra la aplicación, es seleccionado por otro alumno,… desaparecerá automáticamente de la lista impidiendo así que el alumno pueda escogerlo como tutor.

En caso de que la lista se quedara sin ningún tutor apto disponible; la aplicación se lo comunicará al alumno para que este pueda repetir la búsqueda, o realizar una nueva cambiando las condiciones impuestas.

4.2.7 Comunicación a través del Chat

Cuando el usuario con el rol de alumno seleccione un tutor de la lista de posibles tutores, la aplicación avisará al resto de posibles tutores con quienes mantenía una conexión de que no han sido seleccionados y se cerrará la conexión por ambos lados. Y al tutor seleccionado se le comunicará la decisión y se establecerá con él otra comunicación a modo de chat, para que el alumno y el tutor puedan comunicarse directamente y resolver sus dudas.

Durante el chat ambos usuarios podrán intercambiar un número ilimitado de mensajes. En caso de que alguno intente enviar un mensaje en blanco (sin nada escrito) al otro, la aplicación no enviará nada.

Posteriormente, cualquiera de los dos usuarios podrá cerrar el chat finalizándose así también las comunicaciones que hay por debajo. La aplicación se encargará de avisar al otro extremo de lo sucedido, sin embargo, continuará mostrando al usuario que no cerró la comunicación la conversación mantenida; para que pueda verla y obtener la información que de otra forma se perdería involuntariamente.

Por lo tanto, en caso de que, el usuario que cierre el chat sea el tutor, este inmediatamente volverá a estar disponible; mientras que el alumno podrá acceder a la conversación mantenida hasta que la cierre voluntariamente.

En caso contrario (cierra el chat el alumno), este podrá volver a realizar una búsqueda inmediatamente, mientras que el tutor podrá visualizar la conversación mantenida. Sin embargo, no volverá a estar disponible hasta que no cierre dicha conversación, ya que quizás también le haya aportado algo y desee repasarla; por lo que podría molestarle que otro alumno abriese con él un nuevo chat perdiéndose esta información.

4.3 Herramientas de Desarrollo

En este apartado se explicará brevemente qué herramientas se han usado para el desarrollo del proyecto. Se comenzará realizando algunas anotaciones respecto a la versión del lenguaje de programación utilizado y se continuará hablando del entorno de desarrollo.

4.3.1 Java 2 Enterprise Edition 5.01

El lenguaje de programación que se ha usado en este proyecto es Java 2 Micro Edition (J2ME). Las plataformas Java relacionadas con J2ME son Java 2 edición estándar (J2SE) y Java 2 edición empresa (J2EE); ya que en realidad J2ME es una versión reducida de ellas orientada a dispositivos móviles. En nuestro caso en concreto, se ha instalado la versión 5.01 de J2EE, para dar soporte a la aplicación Sun Wireless Toolkit.

4.3.2 UltraEdit

Es un Editor de textos fácil de manejar, que permite trabajar con proyectos Java de forma sencilla. Se escogió UltraEdit, y no un entorno de desarrollo de código como Eclipse, porque al incluir la aplicación Sun Wireless Toolkit un compilador únicamente necesitábamos un editor. Y de entre todos los editores posibles se seleccionó este porque, al igual que el Editplus, entiende la sintaxis Java haciendo el sintax highlight de palabras clave de este lenguaje; y además cuenta con algunas prestaciones muy útiles, como por ejemplo:

- La posibilidad de abrir en ventanas diferentes múltiples archivos de tamaño ilimitado. Pudiendo así, analizar partes del código de diferentes clases simultáneamente, y realizar búsquedas y sustituciones de un fragmento de texto en todos los archivos.

- La opción de comparar archivos a través del UltraCompare, especialmente útil cuando hay que buscar diferencias entre dos versiones distintas de un mismo fichero; para detectar errores e introducir mejoras.

- El modo de edición columna (muy útil a la hora de editar listas).

En la ilustración 15 se muestra una captura de pantalla en la que se observa el aspecto de UltraEdit. Puede observarse la existencia de ventanas en las que se muestran las distintas clases en código Java utilizadas en este proyecto.

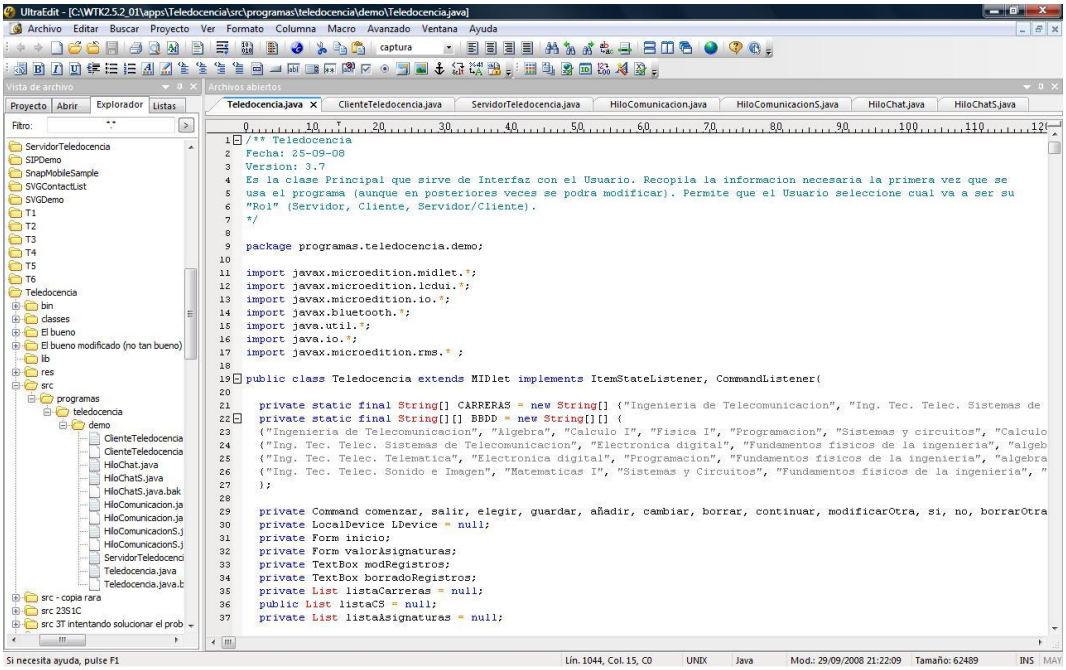


Ilustración 15: Aspecto de UltraEdit

4.3.3 Sun Wireless Toolkit

Se trata de un conjunto de herramientas que permite crear aplicaciones para teléfonos móviles y otros dispositivos inalámbricos de conexión limitada (CLDC). Aunque se basa en el perfil para información de dispositivos móvil (MIDP) 2.1, también soporta muchos paquetes opcionales, lo que la convierte en una herramienta de desarrollo muy completa y versátil.

Con Sun Wireless Toolkit crearemos un proyecto para nuestra aplicación Java (MIDlet), generándose así una estructura de directorios dentro de la cual se almacenarán las clases (archivos de texto con el código fuente de la aplicación, que han sido editados con el UltraEdit, y tienen extensión .java), necesarias para la aplicación.

También compilaremos nuestra aplicación con esta herramienta, con lo que se crearán en la ruta de directorios los archivos compilados (.class) y ejecutables (.jad) que contienen todos los detalles del MIDlet. A partir de estos últimos podremos generar los archivos java.(.jar) que son los que deberemos introducir en nuestro terminal móvil para poder utilizar la aplicación.

Además Sun Wireless Toolkit es un emulador que nos permite probar los MIDlets, una vez han sido compilados, antes de pasarlos al teléfono móvil. Mostrando en su consola los mensajes generados por la aplicación, y los errores que se producen durante su funcionamiento a modo de Log.

En la ilustración 16 podemos ver cuál es el aspecto de la herramienta Sun Wireless Toolkit. En la captura de pantalla podemos observar dos emuladores de terminales móviles con la aplicación en funcionamiento; y los logs que deja la aplicación durante su funcionamiento en la consola principal.

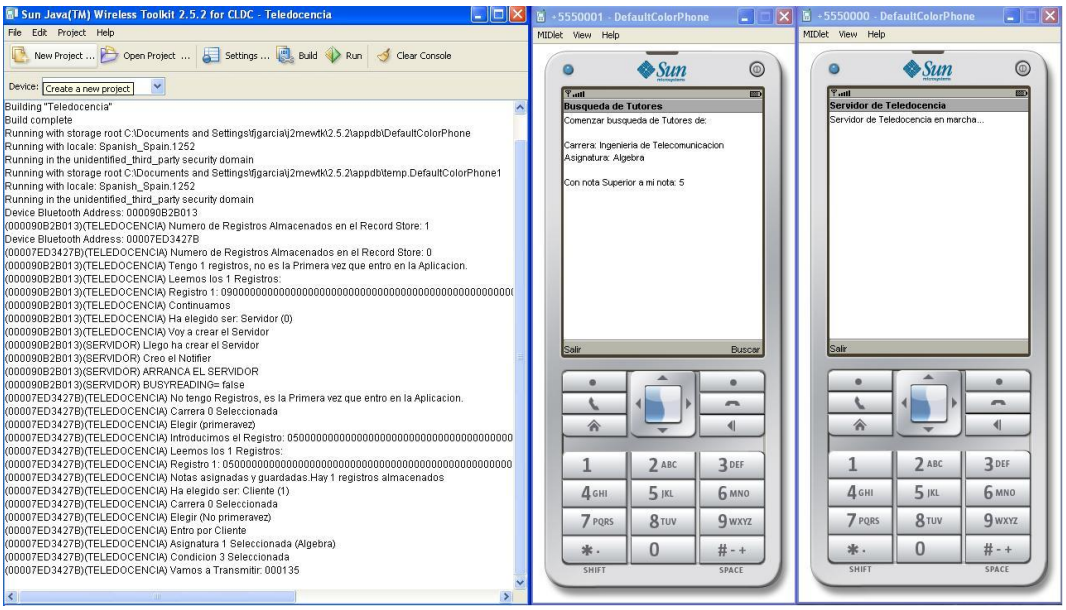


Ilustración 16: Aspecto de Sun Wireless Toolkit

Para finalizar, indicar que en el Anexo 1 se ha incluido un manual básico de Sun Wireless Toolkit, en el cual se explica en mayor profundidad su manejo.

Capítulo V

Implementación

En este capítulo se van a detallar todos los pasos que se han seguido para la realización de la aplicación. Una vez que en el capítulo anterior se realizó un diseño a alto nivel de la aplicación, en este punto se puede comenzar con la implementación de todos los componentes.

En primer lugar se detallará cada operación realizada por la aplicación desde que el usuario inicia la aplicación hasta que selecciona un rol. A continuación, se especificarán los pasos seguidos en función del rol escogido, para ello se añadirán distintos apartados dedicados a puntos importantes que se han tenido en cuenta en la implementación. Una vez completada la fase de implementación, se ha creído conveniente incluir un apartado dedicado a las pruebas de la aplicación.

5.1 Implementación de las Operaciones de la Aplicación

En este apartado se definirán en mayor grado de detalle, las operaciones implementadas en la aplicación.

5.1.1 Autoevaluación

Para que el usuario pueda evaluar sus conocimientos, es necesario que la aplicación disponga de un listado de todas las carreras y asignaturas que se pueden cursar. Por lo que en la clase Teledocencia, que implementa la mayor parte del bloque Core Común y del bloque Interfaz con el Usuario, se definirá una matriz de strings bidimensional, denominada "BBDD", de forma que cada fila de esta matriz se corresponderá con una carrera, y cada columna con una asignatura de esa carrera (a excepción de la primera columna que contendrá el nombre de la carrera).

De esta forma cada vez que la aplicación necesite mostrar por pantalla un listado con las asignaturas de una carrera, bien porque quiera que el usuario evalúe sus conocimientos, darle la opción de modificar evaluaciones pasadas,… la aplicación confeccionará la lista en J2ME utilizando esta matriz.

En primer lugar, cuando el usuario arranca la aplicación de Teledocencia y pulsa el botón "Comenzar" en la pantalla de presentación, esta abrirá el almacén de registros (Record store) y comprobará el número de registros que tiene almacenados:

```
rs = RecordStore.openRecordStore("AlmacenTeledocencia",true);
numRegistros = rs.getNumRecords();
```

En caso de que no haya ninguno, se obligará al usuario a dar de alta uno haciendo que escoja una carrera y autoevalúe los conocimientos que posee en sus asignaturas. Una vez tiene, al menos un registro almacenado, la aplicación permitirá al usuario realizar otro tipo de operaciones, como: modificar un registro ya existente, añadir un nuevo registro o borrar uno de los que ya están introducidos. Al finalizar cualquiera de estas operaciones, la aplicación dará la opción al usuario de repetir esa misma operación sobre otro registro, también de realizar alguna de las otras operaciones antes de continuar y escoger el rol.

5.1.1.1 Dar de alta un registro

Este operación únicamente se puede realizar cuando el usuario no tiene almacenado ningún registro; bien porque sea la primera vez que se utiliza, porque no se haya dado de alta ninguno antes, o porque el usuario haya borrado todos los registros existentes. Para utilizar la aplicación es obligatorio que el usuario escoja una carrera de una lista con todas las carreras posibles, y evalúe sus conocimientos en las asignaturas de esta. Para ello, cuando el usuario escoge una carrera de la lista, la aplicación crea un Form [1] que contendrá el nombre de las asignaturas pertenecientes a la carrera

seleccionada, y un TextField **[1]** al lado de cada uno de ellos, para recoger el valor numérico entre "0" y "9" que puede dar el usuario para evaluar su conocimiento.

Consideraremos que un usuario ha evaluado correctamente sus conocimientos en una carrera cuando al menos ha introducido un valor distinto de "0" o "null" en alguno de estos TextFields. De no ser así cuando el usuario intente continuar, bien pulsando el botón "Guardar" o "Añadir otra carrera" le saldrá un error por pantalla; ya que al no tener registros almacenados no se le dejará continuar hasta que dé, al menos, un registro de alta correctamente.

Una vez la aplicación muestra por pantalla el Form al usuario para que este valore su conocimiento, la aplicación permitirá al usuario realizar 3 acciones:

- Salir: siempre que el usuario escoja esta acción se cerrará el Record Store, sin guardar el registro con las notas que estaba introduciendo, y a continuación se cerrará la aplicación.

  rs.closeRecordStore();
  notifyDestroyed();

- Guardar: con esta acción la aplicación recogerá todos los valores introducidos por el usuario en los TextFields, valorará si el usuario ha evaluado correctamente sus conocimientos de las materias y creará con ellos y el identificador de la carrera un registro.

  Si el usuario evaluó correctamente sus conocimientos, se almacenará el registro en el Record Store. Después, se mostrará por pantalla un TextBox **[1]** con el nombre de las carreras cuyos registros tenemos almacenados y que obtendremos con el método "leerRegistros()", el cuál accede al Record Store y devuelve una matriz de números enteros con todos los registros almacenados, en donde cada fila será un registro y cada columna el valor correspondiente a una asignatura, exceptuando la primera columna que es el identificador del registro. Y por último permitirá al usuario realizar las siguientes operaciones, que explicaremos más adelante: "Modificar Conocimientos", "Añadir otra Carrera" (siempre que no haya ya un registro para cada carrera) o "Eliminar Carrera"; o bien le permitirá "Continuar" con la aplicación eligiendo su Rol.

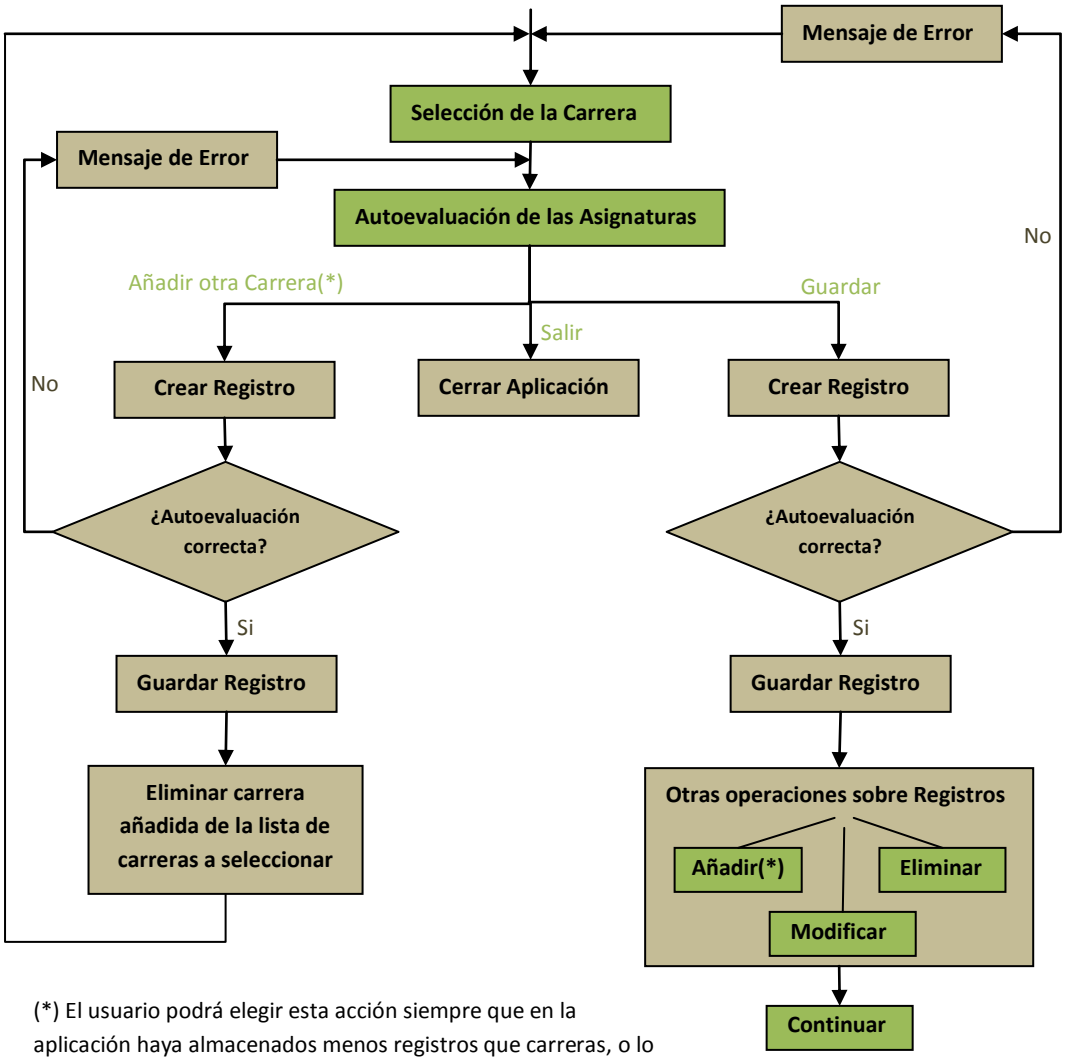
  En caso contrario, la aplicación no almacenará el registro en el Record Store, además devolverá un error por pantalla explicándole el problema, y a continuación le volverá a mostrar la lista de carreras completa para que seleccione una y evalúe sus conocimientos.

- Añadir otra carrera: esta acción es muy parecida a la de "Guardar" salvando algunas diferencias.

  En este caso, si el usuario evalúa correctamente sus conocimientos, se guardará el registro en el Record Store, y se mostrará por pantalla de nuevo la lista de carreras, con la salvedad de que no aparecerán todas las carreras, sino solo aquellas para las que no se tenga algún registro almacenado.

Por lo que el usuario podrá escoger una de estas carreras, evaluar sus asignaturas y escoger de nuevo alguna de estas operaciones:“Salir”, “Guardar” o “Añadir otra Asignatura”. Excepto si en el Record Store hay almacenados N registros, y hay M carreras en total, siendo N=M-1, lo cual nos indicaría que la carrera que estamos evaluando es la única para la que no tenemos aún un registro introducido, en cuyo caso la aplicación únicamente permitirá “Guardar” o “Salir”.

Otra diferencia es, que en el caso de que el usuario no evalúe sus conocimientos adecuadamente, después de sacar un error por pantalla no se le volverá a mostrar la lista de las carreras que aún no han sido evaluadas por considerarse reiterativo al haber elegido el usuario esta carrera en particular en el paso anterior, por lo que se le debe volver a mostrar el Form con las asignaturas de la misma para que las evalúe.



(*) El usuario podrá elegir esta acción siempre que en la aplicación haya almacenados menos registros que carreras, o lo que es lo mismo, no se hayan evaluado todas las carreras.

Ilustración 17: Diagrama de Flujo “Dar de alta un Registro”/“Añadir otra Carrera”

En la ilustración 17 podemos ver el diagrama de flujo correspondiente a la operación "Dar de Alta" un registro. Los procesos, decisiones y acciones en color verde, son las que realiza el usuario; mientras que las de color gris, son las que realiza la aplicación. Esta convención de colores la emplearemos también en los sucesivos diagramas de flujos.

Posteriormente, cuando el usuario arranque la aplicación ya no tendrá que dar de alta un registro, si ya lo hizo anteriormente. Mostrándole la aplicación al pulsar el botón "Comenzar" un TextBox con el nombre de las carreras evaluadas y además se le dará la opción de "Modificar Conocimientos", "Añadir otra Carrera" (siempre que no haya ya un registro para cada carrera), "Eliminar Carrera"; o "Continuar" con la aplicación y elegir el rol que va a desempeñar.

5.1.1.2 Modificar un registro

El usuario podrá realizar esta operación una vez haya dado de alta al menos un registro. Con esta operación el usuario podrá recuperar las evaluaciones de las asignaturas pertenecientes a una carrera ya almacenada y modificarlas.

Para ello, el usuario elegirá una carrera de una lista de carreras creada por la aplicación con el nombre de todas las carreras para las que se tiene un registro almacenado. Una vez el usuario seleccione la carrera que desea modificar, a los TextFields del Form, que se crea con el nombre de todas las asignaturas de la carrera, se les inicializa con el valor del registro almacenado con que se evaluaron anteriormente; de forma que el usuario pueda ver que evaluaciones tiene guardadas y modificar las que crea conveniente. Llegados a este punto el usuario podrá realizar una de estas 3 acciones:

- Salir (comentada anteriormente)

- Guardar: la aplicación creará un registro con las evaluaciones almacenadas en los TextFields, y valorará si las evaluaciones asignadas son válidas para ser almacenadas como registro, en función de las condiciones anteriormente comentadas (exista alguna evaluación distinta de "0" o "null").

  Si se evaluaron correctamente las asignaturas, la aplicación accederá al Record Store y borrará el registro que tenía almacenado hasta ahora de dicha carrera utilizando el método implementado "borrarRegistro()"; que realizará una búsqueda en el Record Store hasta encontrar aquel registro que tenga en la primera posición el mismo identificar de carrera que la carrera que se quiere modificar, y lo eliminará usando el identificador del registro encontrado.

  rs.deleteRecord(idRegistro);

  A continuación, almacenará el nuevo registro creado con las modificaciones realizadas empleando el método implementado "escribirRegistro()", al que le pasará como parámetro de entrada el registro que se desea almacenar en formato string, para que realice la conversión a binario y lo introduzca en el Record Store.

rs.addRecord(registro,0,registro.length);

Finalmente, la aplicación mostrará por pantalla un TextBox con el nombre de las carreras cuyos registros tenemos almacenados obtenidos con el método "leerRegistros()" (anteriormente mencionado) y permitirá al usuario realizar las siguientes operaciones: "Modificar Conocimientos", "Añadir otra Carrera" (siempre que no haya ya un registro para cada carrera) o "Eliminar Carrera"; o bien le permitirá "Continuar" con la aplicación eligiendo su Rol.

En caso que el usuario no evaluará correctamente las asignaturas al modificarlas, la aplicación devolverá un error por pantalla explicándole el problema, y a continuación le volverá a mostrar el Form con el nombre de las asignaturas y los TextFields con las modificaciones realizadas por el usuario y que no son válidas para guardar dicho registro, para que las corrija.

- Modificar otra Carrera: esta acción es prácticamente idéntica a "Guadar", por lo que vamos a explicar cuáles son sus diferencias.

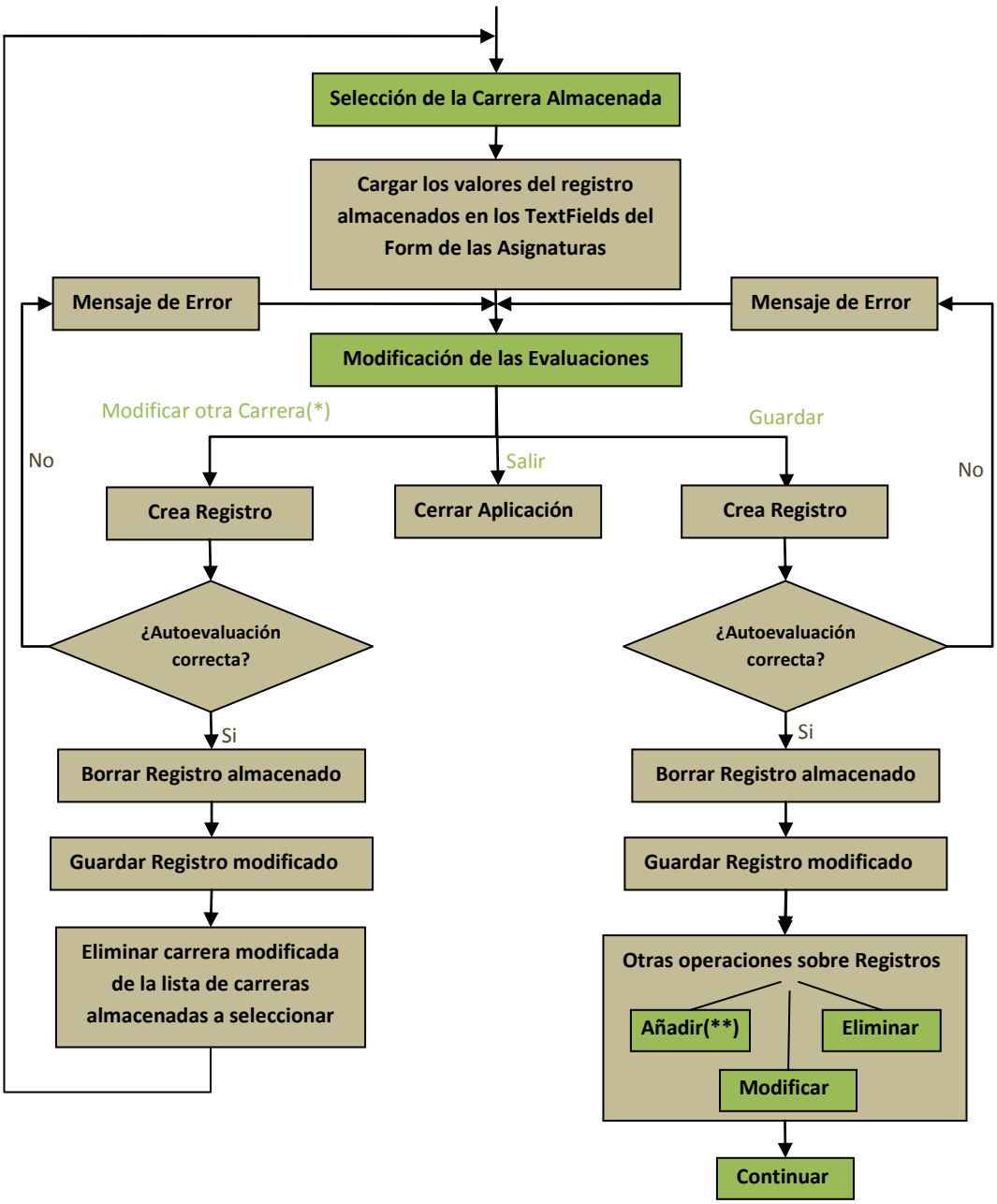
  Al seleccionar esta acción, la aplicación seguirá los mismos pasos que con la acción "Guardar", hasta que almacene el registro modificado en el Record Store. Una vez almacenado el registro, eliminará de la lista de carreras a modificar disponibles (cuyos registros tenemos almacenados) el nombre de la carrera modificada, para que el usuario escoja otra carrera si lo desea y modifique sus conocimientos.

  De esta forma, con cada registro modificado por el usuario irá disminuyendo el número de carreras disponibles a modificar. En caso de que el usuario emplease esta acción para actualizar todos los registros almacenados, cuando esté modificando el último de ellos la aplicación no le permitirá modificar más registros teniendo como únicas acciones disponibles: "Guardar" o "Salir".

  El objetivo de que una vez modificado un registro no se permita en esa operación ("Modificar Registros") volver a modificarlo, es que el usuario sepa en todo momento con claridad en que carreras ha modificado sus conocimientos, ya que contando las asignaturas de libre elección y optativas el número de registros puede ser elevado.

  Esto no implica que el usuario únicamente pueda modificar sus conocimientos en una carrera determinada una única vez, ya que si el usuario escoge la acción de Guardar podrá, como hemos visto anteriormente, volver a escoger la opción "Modicar Conocimientos" y volver a modificar de nuevo las evaluaciones de todas las carreras registradas.

En la ilustración 18 se muestra el diagrama de Flujo de la operación "Modificar un Registro".



(*) El usuario podrá elegir esta acción siempre tenga registros almacenados que no haya modificado.
(**) El usuario podrá elegir esta acción siempre que en la aplicación haya almacenados menos registros que carreras, o lo que es lo mismo, no se hayan evaluado todas las carreras.

Ilustración 18: Diagrama de Flujo "Modificar un Registro"

5.1.1.3 Añadir otra carrera

Esta operación únicamente se podrá realizar cuando el usuario tenga almacenado en el Record Store un registro. Su función será permitir al usuario añadir nuevos registros de carreras cuyas asignaturas no había cursado anteriormente, cada vez que acceda a la aplicación.

Por lo que su implementación será idéntica a la de la operación "Dar de alta un Registro", con la diferencia de que la lista de carreras que mostrará para seleccionar la carrera, cuyas asignaturas van a ser evaluadas, no contendrá el nombre de todas las carreras existentes en la BBDD sino únicamente de aquellas que aún no tienen un registro almacenado en el Record Store. La ilustración 17, por lo tanto, también muestra el Diagrama de Flujo de la operación "Añadir otra Carrera".

5.1.1.4 Eliminar una carrera

Para poder realizar esta operación es necesario que el usuario haya dado de alta al menos un registro. Con esta operación el usuario podrá eliminar del Record Store aquellos registros almacenados que ya no le sean necesarios. Es posible eliminar todos los registros almacenados en el Record Store, pero en ese caso no será posible continuar usando la aplicación, a menos que se dé de alta como mínimo un registro.

Con este fin, la aplicación confeccionará una lista con el nombre de todas las carreras para las que hay almacenado un registro en el Record Store, y el usuario escogerá aquella que quiere borrar definitivamente. Una vez el usuario seleccione la carrera que desea eliminar, la aplicación realizará una última comprobación de seguridad, que consiste en mostrar al usuario en un TextBox un mensaje, a través del cual, se le preguntará si está seguro de eliminar dicha carrera; y en el que el usuario únicamente podrá escoger entre 2 acciones: "Si" o "No".

Si el usuario finalmente se retracta de eliminar el registro correspondiente a la carrera seleccionada, escogiendo la acción "No", automáticamente la aplicación le mostrará de nuevo el TextBox compuesto por el nombre de todas las carreras evaluadas y le dará de nuevo la oportunidad de "Modificar Conocimientos", "Añadir otra Carrera" (siempre que no haya ya un registro para cada carrera), "Eliminar Carrera" o "Continuar" usando la aplicación y elegir un rol.

Por otro lado, si el usuario confirma la eliminación del registro a través de la acción "Si", la aplicación procederá a borrarlo empleando el método anteriormente mencionado "borrarRegistro()", informará al usuario por pantalla de que se ha eliminado dicho registro correctamente y le permitirá realizar una de estas 2 acciones:

- Borrar otro Registro: Esta acción únicamente estará disponible cuando después de eliminar el registro seleccionado por el usuario, aún quede al menos un registro más almacenado en el Record Store.

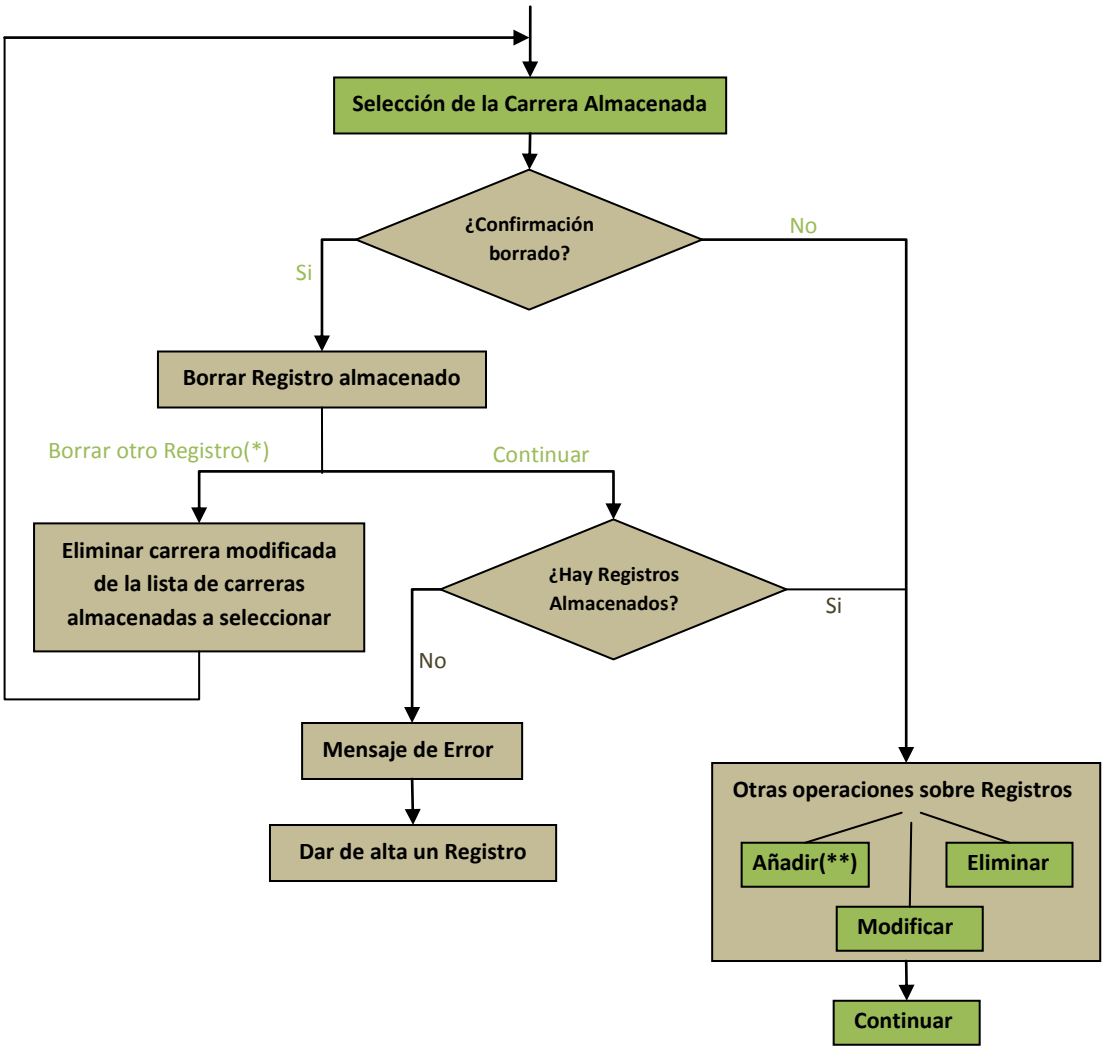
  Se encargará de eliminar el nombre de la carrera cuyo registro ha sido eliminado, de la lista que contiene los nombres de las carreras que aún tienen registros almacenados en el Record Store; para a continuación mostrarle esta lista al usuario y que este escoja el nombre de otra carrera cuyo registro desea eliminar.

- Continuar: cuando el usuario haya terminado de eliminar registros del Record Store, deberá escoger esta acción, y la aplicación deberá comprobar si tras las operaciones realizadas queda algún registro almacenado en el Record Store.

Si se han eliminado todos los registros, la aplicación mostrará un error en pantalla, en el que explicará al usuario que para poder continuar utilizando la aplicación es imprescindible dar de alta al menos un registro. A continuación, le mostrará una lista con el nombre de todas las carreras disponibles en la BBDD, para que escoja una y evalúe sus conocimientos.

Si por el contrario, aún quedan registros almacenados en el Record Store, la aplicación le mostrará al usuario por pantalla un TextBox en el que aparecerán los nombres de todas la carreras registradas, sobre las que el usuario podrá realizar las operaciones: "Modificar Conocimientos", "Añadir otra Carrera" (siempre que no haya ya un registro para cada carrera), "Eliminar Carrera" o "Continuar" con la aplicación y elegir un rol.

En la ilustración 19 se muestra el diagrama de Flujo de la operación "Eliminar un Registro".



(*) El usuario podrá elegir esta acción siempre tenga algún registro almacenado.
(**) El usuario podrá elegir esta acción siempre que en la aplicación haya almacenados menos registros que carreras, o lo que es lo mismo, no se hayan evaluado todas las carreras.

Ilustración 19: Diagrama de Flujo "Eliminar un Registro"

5.1.2 Selección del rol

Una vez la aplicación cuenta con al menos un registro en el Record Store, el usuario ha realizado todas las operaciones sobre los registros ("Modificar", "Añadir" o "Eliminar") que ha considerado oportunas; es hora de que continúe y escoja el rol que va a desempeñar.

Por lo tanto, cuando la aplicación muestre al Usuario el TextBox compuesto por los nombres de las carreras para las que tiene un registro almacenado, y le dé la opción de realizar sobre ellas alguna operación de las anteriormente comentadas, o de "Continuar", el usuario deberá escoger esta última operación para proseguir con la aplicación y poder elegir un rol: "Alumno" o "Tutor".

En este punto la aplicación realiza una comprobación de seguridad extra para garantizar que hay al menos un registro almacenado en el Record Store y aportar así robustez a la misma. De no ser así, mostrará un mensaje de error por pantalla informando del problema al usuario y le indicará que debe dar de alta un registro para poder continuar. En caso de que todo este correcto y al menos exista un registro almacenado, le mostrará al usuario por pantalla una lista con los dos posible roles que puede tomar, dándole la opción de "Elegir" cualquiera de ellos o de "Salir" de la aplicación.

5.1.2.1 Rol de Tutor (o servidor)

En caso de que el usuario escoja el rol de "Tutor" la aplicación arrancará un hilo destinado a realizar las funciones de un servidor Bluetooth, que además implemente las necesidades de nuestra aplicación, al que denominaremos "hilo servidor".

5.1.2.1.1 Hilo servidor

El objetivo del "hilo servidor" es crear una conexión, abrirla y esperar a recibir las condiciones (que deberá cumplir para poder ser su tutor) enviadas por un alumno que necesite un tutor (a través de un "hilo cliente"). Si por la conexión recibe las condiciones de algún cliente, la aplicación arrancará un "hilo de comunicación servidora" al que pasará como parámetros la conexión y los datos que ha recibido (condiciones), para a continuación volver a crear una nueva conexión y permanecer a la escucha de nuevas condiciones enviadas por otros clientes; y así sucesivamente hasta que algún cliente le verifique como su tutor.

Para ello, cuando la aplicación arranque el "hilo servidor", este se encargará de hacer visible por Bluetooth nuestro dispositivo a todos los dispositivos que realicen búsquedas de dispositivos mediante esta tecnología:

LDevice.setDiscoverable(DiscoveryAgent.GIAC);

Únicamente existirá un "hilo servidor", que se estará ejecutando continuamente, mientras el usuario no abandone su rol. Por esta razón, lo primero que deberá hacer será comprobar si el servidor, en la anterior ejecución, ha mantenido una conversación por chat con algún usuario, y verificar si el tutor está aún leyendo un Form con dicha conversación. Esto solamente ocurrirá, si fue el cliente quién cerró el chat; ya que de

haberlo cerrado el tutor habría dado por concluida la comunicación y no necesitaría ningún dato de la misma. Por lo que la aplicación se encargará de volver a nuestro dispositivo "no localizable" a otros clientes hasta que el tutor cierre este Form, puesto que de no hacerlo podrían establecerse chats con otros clientes y perder la conversación mantenida, que no ha sido cerrada por él y que puede tener datos de su interés.

Si en la anterior ejecución el tutor no hubiese mantenido una conversación con un alumno que hubiera sido finalizada por el cliente, o si el Form que contiene dicha conversación ha sido cerrado por el tutor; el siguiente paso de la aplicación consistirá en crear una conexión servidora, tal y como vimos en el apartado 2.2.5.2.1 de esta memoria:

```
url =new StringBuffer("btspp://localhost:");
url.append(SERVICIO_TELEDOCENCIA.toString());
url.append(";name=ServicioTeledocencia;authorize=false");
notifier = (StreamConnectionNotifier)Connector.open(url.toString());
```

De esta forma obtendremos un notifier, que es un objeto que nos permitirá escuchar las conexiones entrantes de los clientes. Con la opción "authorize=false", indicamos al notifier que cada vez que reciba una petición de conexión por parte de un cliente al lanzar este una búsqueda de dispositivos (startInquiri()), la acepte automáticamente. De esta forma, la aplicación no tendrá que preguntar continuamente al usuario si acepta una conexión Bluetooth no autorizada.

Al crear el notifier, el módulo de seguridad de J2ME pedirá confirmación al usuario para abrir una conexión Bluetooth servidora, lo que implicará que otros dispositivos Bluetooth podrán conectarse al terminal del usuario. Esto es justo lo que la aplicación pretende, por lo que, la primera vez que esto ocurra, el usuario deberá dar su permiso para abrir dicha conexión **[12]**, en caso de no hacerlo la aplicación le mostrará una alerta **[1]** informándole que no tiene sentido ejercer de servidor si ningún dispositivo puede conectarse con él; y volverá a mostrarle la lista de selección de rol, para que escoja uno.

El siguiente paso que debe realizar el "hilo servidor" será abrir las conexiones mediante el método acceptAndOpen(), a través del cual la aplicación guardará el servicio Teledocencia en la BSDB y lo publicará para que pueda ser visto por los clientes que buscan dispositivos que ofrezcan dicho servicio.

```
con = notifier.acceptAndOpen();
```

Cuando el notifier lanza el acceptAndOpen(), lo hace con el objetivo de aceptar una nueva conexión de algún cliente. Por esta razón cuando se ejecuta este método, el notifier tomará el control del "hilo servidor" permaneciendo a la espera de recibir, por parte de algún cliente, una búsqueda de dispositivos (startInquiry()); y no devolverá el control al hilo a menos que reciba una, o bien se cierre el notifier.

Si el dispositivo servidor recibe una señal de búsqueda de dispositivos por parte de algún cliente, el notifier, tal y como se explicó en el apartado 2.2.5.2.3 de esta memoria, como resultado de invocar este método obtendrá un objeto javax.microedition.StreamConnection, a través del cual podremos leer y escribir datos:

in = con.openDataInputStream();
out = con.openDataOutputStream();

A continuación, el "hilo servidor" quedará a la espera de que este cliente le envíe las condiciones necesarias que debe cumplir un servidor para poder actuar como su tutor. En el momento que el "hilo servidor" recibe las condiciones impuestas por un alumno, automáticamente creará un "hilo de comunicación servidora" al que pasará como parámetros los datos de la conexión (con, in, out) y las condiciones recibidas del cliente. E inmediatamente después abrirá una nueva conexión y volverá a permanecer a la espera de ser localizado por algún otro cliente, puesto que el que pueda servir como tutor a un alumno no le garantiza que este finalmente le escoja de entre todos los tutores que hayan cumplido sus condiciones.

En todo momento, el "hilo servidor" tendrá un registro con todos los "hilos de comunicaciones servidoras" abiertos. Y volverá a ejecutarse (cerrando el notifier que espera conexiones de nuevos clientes), volviendo a realizar todos los pasos que explicados anteriormente, siempre que el usuario no abandone el rol de tutor y además:

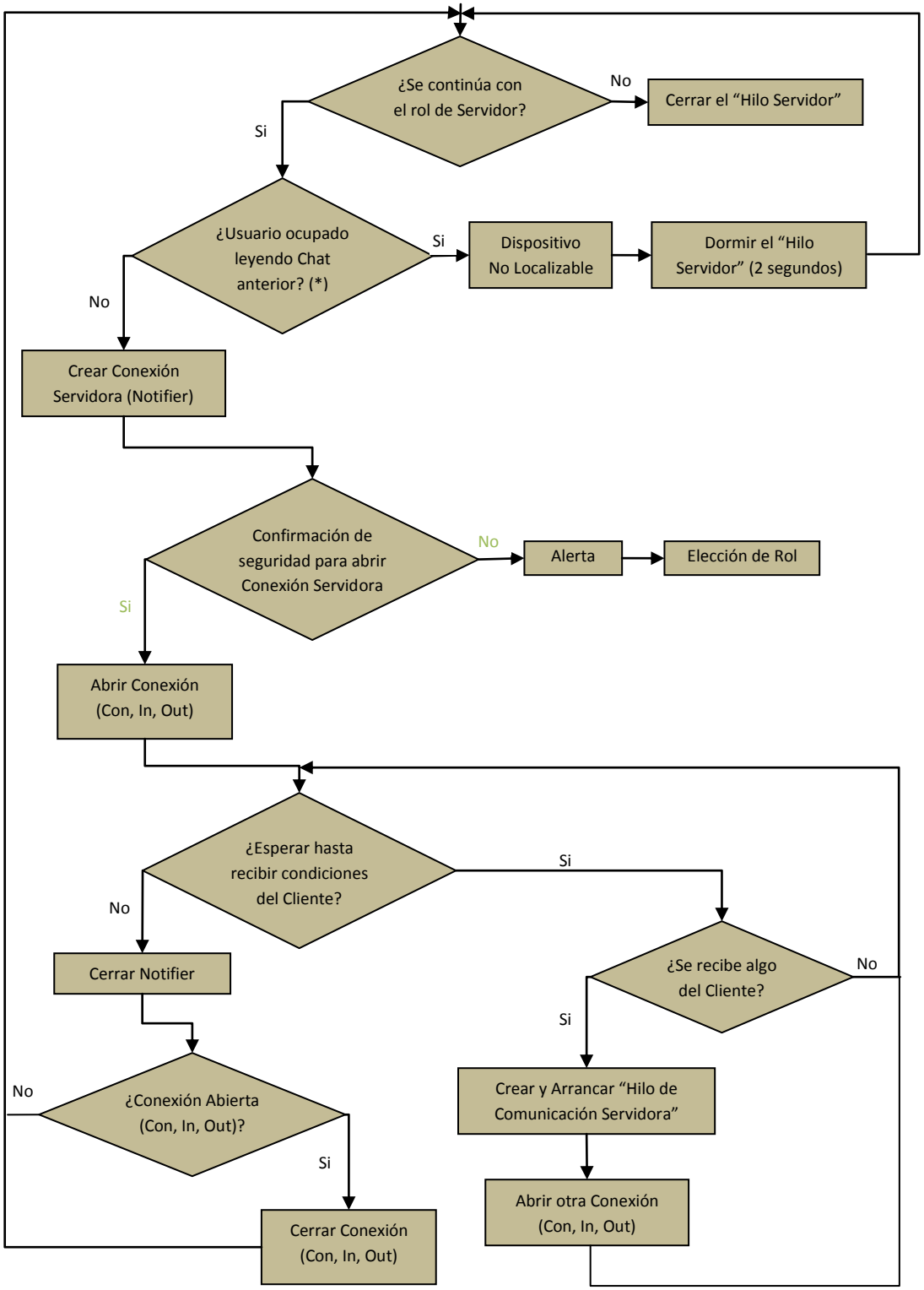
- Cuando un "hilo de comunicación servidora" le informe que el dispositivo no tiene almacenado en el Record Store ningún registro de la carrera para la que necesita ayuda el cliente.

- Cuando un "hilo de comunicación servidora" le comunique que el dispositivo no cumple las condiciones impuestas por el cliente para ser su tutor.

- Cuando un "hilo de comunicación servidora" le indique que no ha resultado elegido como tutor del cliente (a pesar de cumplir las condiciones impuestas), o bien que ha resultado elegido para ser su tutor, pero el dispositivo es tutor de otro cliente que se la ha adelantado. Y además, el "hilo servidor", no tenga otros "hilos de comunicaciones servidoras".

Todos estos pasos, salvo el de confirmación de apertura de una conexión Bluetooth, se realizan de forma invisible al usuario, que desde que escoge el rol de tutor, verá en el display **[1]** de su dispositivo un Form con el mensaje "Servidor de Teledocencia en marcha..." y una única acción disponible: "Salir".

Si el usuario selecciona la acción "Salir" la aplicación deberá informar a través de los "hilos de comunicaciones servidoras" que tiene abiertos a todos los "hilos clientes", que el usuario de este dispositivo ha decidido abandonar el rol de tutor. A continuación cerrará el Notifier, la conexión por la que esperaba recibir nuevas condiciones de un nuevo cliente, así como todos los "hilos de comunicaciones servidoras" asociados a este "hilo servidor", también cerrará este hilo y finalmente mostrará al usuario la lista de selección de rol. Por lo tanto, los dispositivos clientes que tenían el nombre del dispositivo insertado en una lista como un posible tutor lo eliminarán de la misma y cerrarán el "hilo de comunicación cliente" que tenían abierto con este dispositivo, y en caso de que este fuese el único tutor disponible volverán a mostrar al usuario la lista de selección de tutor. La ilustración 20 muestra el diagrama de Flujo correspondiente al "hilo servidor".



(*) Esta opción únicamente podrá ocurrir cuando, en la anterior ejecución, el servidor haya mantenido una conversación por Chat (con intercambio de información) con un cliente, y el cliente cerrase el Chat.
Nota: El usuario podrá abandonar el rol de tutor en cualquier momento con la opción "Salir".

Ilustración 20: Diagrama de Flujo "Hilo Servidor"

5.1.2.1.2 Hilo de comunicación servidora

Este hilo recibirá como parámetros una conexión (con, in, out), y las condiciones que envía un cliente al servidor. Se encargará de verificar si el tutor cumple esas condiciones, le contestará al cliente si es un tutor válido para ayudarle a resolver sus dudas y esperará una respuesta por parte del mismo en la cual le indique si ha sido seleccionado o no como su tutor. En caso de resultar finalmente seleccionado por el cliente como su tutor, abrirá un "hilo chat servidor" para poder comunicarse con él.

El primer paso de este hilo, será comprobar si existe algún registro en el Record Store de la carrera para la que necesita ayuda el alumno. A continuación, comprobará si la calificación con que se autoevaluó el tutor en la asignatura en cuestión, cumple con las condiciones impuestas por el alumno, para ser considerado como un posible tutor.

En caso de no existir, para esas carrera, ningún registro almacenado en el Record Store, o de no cumplir el tutor las condiciones impuestas por el alumno; se reiniciará el "hilo servidor" (lo que implica que se cerrará el notifier, con lo que el "hilo servidor" dejará de esperar conexiones de otros clientes, tampoco esperará recibir condiciones del cliente cerrándose la conexión (con, in, out) en caso de que se hubiesen abierto; para volver a ejecutarse desde el comienzo), se cerrará la comunicación (con, in, out) del "hilo de comunicación servidora", para a continuación cerrarse también el propio "hilo de comunicación servidora".

Sin embargo, si el tutor tiene almacenado en el Record Store un registro de la carrera para la que solicita ayuda el cliente, el "hilo de comunicación servidora" comprobará si la nota con que se autoevaluó el usuario cumple con las condiciones impuestas por el cliente para ser su tutor, contestará al cliente (a través del "hilo de comunicación cliente") si cumple sus condiciones, y esperará a que el cliente le comunique si elige a este usuario como su tutor o no.

Si finalmente el cliente no elige a este usuario como su tutor; o lo elige, pero ya no está disponible porque se le ha adelantado otro cliente seleccionándolo como tutor un instante antes. En este caso, la aplicación deberá comprobar si el "hilo servidor" tiene otros "hilos de comunicaciones servidoras" abiertos o, lo que es lo mismo, si existe algún otro cliente que pueda elegir al servidor como su tutor:

- De ser este "hilo de comunicación servidora" el único del "hilo servidor", se reiniciará el "hilo servidor" como se ha explicado anteriormente, se cerrará la comunicación (con, in, out) del "hilo de comunicación servidora" y por último se cerrará este hilo.

- Mientras que, si por el contrario, hay otros "hilos de comunicaciones servidoras" activos en el "hilo servidor", únicamente se cerrará la comunicación de este "hilo de comunicación servidora" y a continuación se cerrará el hilo. Puesto que si ha resultado "no elegido" puede que alguno de los clientes con los que esta comunicándose a través de estos hilos lo seleccione como su tutor. Y en el caso de que se le haya adelantado otro alumno seleccionando a este servidor como su tutor, como veremos más adelante, será el propio "hilo de comunicación servidora" mantenido con ese cliente el encargado de avisar al resto de clientes que el tutor ya ha sido

seleccionado, y de cerrar el resto de “hilos de comunicaciones servidoras” de este “hilo servidor”.

Por otro lado, si el servidor resulta elegido por el alumno como su tutor estando disponible (no ha sido seleccionado por ningún otro alumno, y se está comunicando con él en este instante), la aplicación se encargará de hacer que el dispositivo sea invisible al resto de dispositivos, de forma que cuando un cliente realice una búsqueda de dispositivos no lo localizará.

LDevice.setDiscoverable(DiscoveryAgent.NOT_DISCOVERABLE);

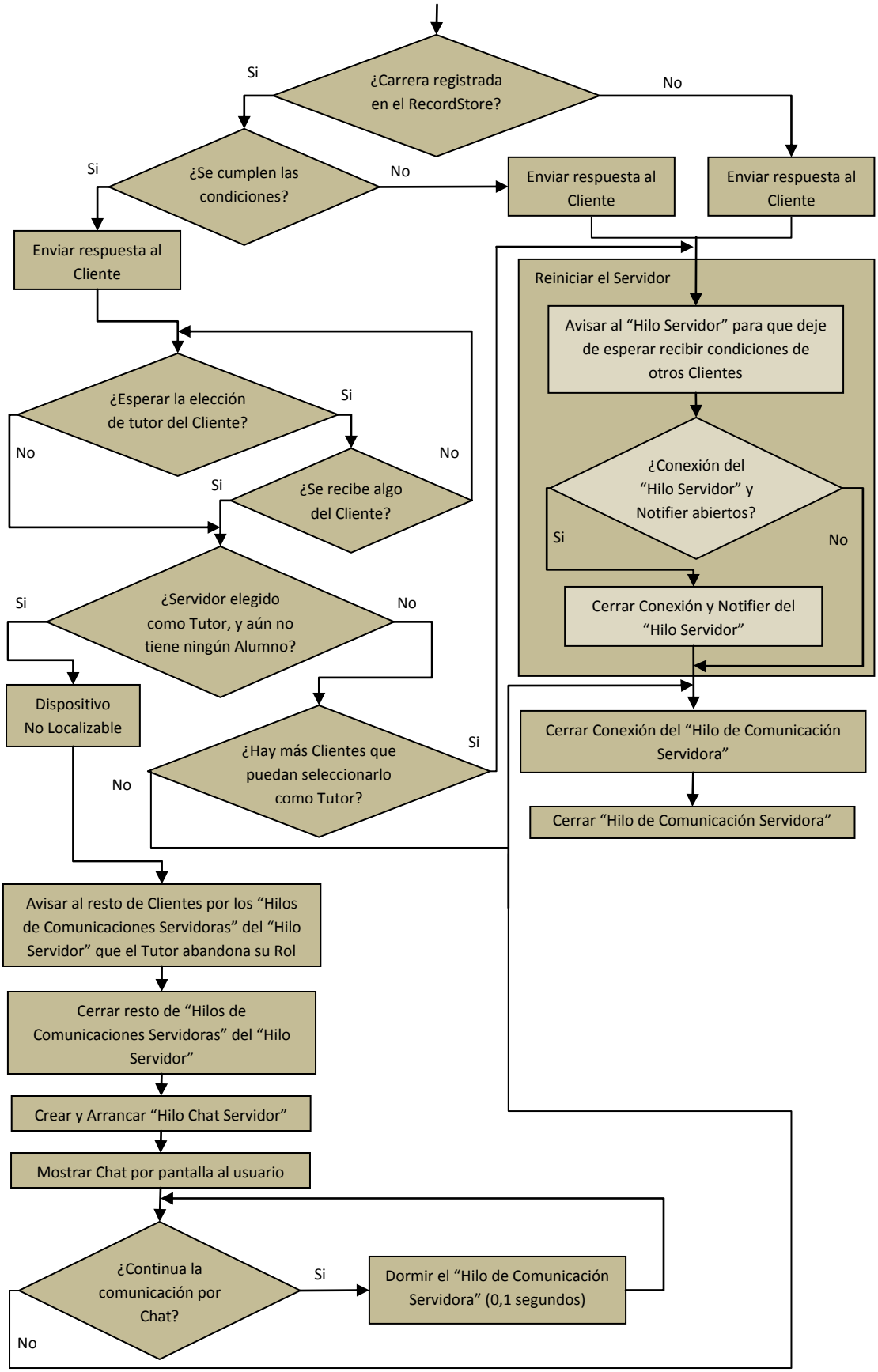
A continuación, avisará a todos los clientes cuyas condiciones estaban siendo valoradas o para los que estaba esperando ser seleccionado como su tutor, a través del resto de “hilos de comunicaciones servidoras” del “hilo servidor”, que el usuario de este dispositivo abandona el rol de tutor, para que, como veremos más adelante, el cliente no continúe esperando una valoración de sus condiciones, o en el caso de que el cliente tuviese en la lista de posibles tutores a seleccionar a este dispositivo lo elimine de ella, para evitar errores a la hora de establecer el chat.

El siguiente paso será cerrar todos los demás “hilos de comunicación servidoras” del “hilo servidor”, para seguidamente crear un “hilo chat servidor”, arrancarlo y mostrarlo por pantalla al usuario para que le sirva de interfaz de comunicación con el alumno que lo ha escogido como tutor.

Por último el “hilo de comunicación servidora” se dormirá a la espera de que se dé por finalizada esta comunicación por alguna de ambas partes (alumno o tutor), o lo que es lo mismo a que se cierre el “hilo chat servidor”, momento en el cual se cerrará la comunicación servidora (con, in, out) del “hilo de comunicación servidora” para a continuación cerrarse también el propio hilo.

Al igual que ocurría con el “hilo servidor”, toda la operativa de este hilo se ejecuta de forma invisible al usuario, que una vez elige el rol de tutor, únicamente verá un Form con el mensaje “Servidor de Teledocencia en marcha...” en el display de su móvil, y la única acción que tendrá disponible será “Salir” (explicada en el apartado anterior) hasta que sea elegido como tutor por algún alumno y la aplicación le muestre un chat para comunicarse con él.

La ilustración 21 muestra el diagrama de Flujo correspondiente al “hilo de comunicación servidora”.



Nota: El usuario podrá abandonar el rol de tutor en cualquier momento con la opción "Salir".

Ilustración 21: Diagrama de Flujo "Hilo de Comunicación Servidora"

5.1.2.1.3 Hilo chat servidor

Es el encargado de establecer con el cliente, que escoge al servidor como tutor, una conversación a modo de chat. Para ello utilizará la comunicación establecida en el "hilo de comunicación servidora", por la cual escuchará permanentemente y en caso de recibir algún mensaje proveniente del alumno se lo mostrará por pantalla al tutor. Y en caso de que el tutor escriba algún mensaje se encargará de enviárselo al cliente.

Por lo que, durante la ejecución de este hilo el usuario visualizará por pantalla un chat creado a partir de un Form, que en principio únicamente contendrá un Textfield pensado para que el usuario pueda introducir los mansajes que desee enviar al alumno, y dos acciones: "Enviar" para mandar el mensaje que hay escrito en el TextField, y "Salir del Chat" para que el usuario cierre el chat en el momento que desee. Posteriormente, a medida que el usuario reciba mensajes del cliente o los envíe, se irán añadiendo también al Form dichos mensajes, uno tras otro, secuencialmente.

El "hilo chat servidor", se ejecutará de forma continua, siempre que el alumno no cierre el chat desde su lado de la comunicación, ni el tutor abandone el mismo con la acción "Salir del Chat".

El primer paso será comprobar, por la conexión establecida con el cliente (con, in, out) a través del "hilo de comunicación servidora" (padre del "hilo Chat servidor"), si el cliente nos envía algún mensaje (por el "hilo chat cliente"). Si no recibimos nada, continuaremos comprobándolo hasta que o el alumno o el tutor cierren el chat.

En caso de recibir algún mensaje del cliente, el hilo deberá comprobar en primer lugar si se trata de una cadena vacía, en cuyo caso lo ignorará y continuará escuchando más mensajes. El objetivo de obviar las cadenas de caracteres vacías, es evitar que el chat se llene de mensajes que no tienen ningún valor para los usuarios, al no contener ninguna información y disponer de un display de tamaño reducido para mostrarlos.

En caso de que el mensaje recibido no fuese un string vacio, el hilo comprobará si se trata del mensaje que el cliente tiene reservado para comunicar al servidor que el alumno abandona el chat. De no ser este tampoco el mensaje recibido, podemos asegurar que el mensaje recibido se trata de un mensaje escrito por el alumno en el dispositivo cliente y enviado por el "hilo chat cliente" a través de la conexión del "hilo de comunicación cliente" como veremos en el siguiente apartado. Por lo que el hilo añadirá dicho mensaje al Form que se muestra al usuario por pantalla a modo de chat, avisará al usuario que se ha recibido un mensaje por parte del cliente mediante un tono de vibración, y volverá a ejecutarse (esperando recibir algo del cliente).

Llegados a este punto, hay que explicar el concepto de "hilo de comunicación servidora especial", como un "hilo de comunicación servidora" que se crea cuando un cliente realiza una búsqueda de dispositivos y localiza a un servidor un instante antes de que este sea seleccionado por otro alumno como tutor. De forma que, aunque el "hilo de comunicación servidora", establecido con el alumno que lo ha elegido como tutor, comunique al resto de "hilos de comunicaciones servidoras" del "hilo servidor", que el usuario abandona el rol de tutor mientras haya comunicación por chat. Puede darse el caso especial (y altamente improbable) de que el "hilo de comunicación servidora" establecido con el cliente (al que denominaremos como "cliente especial") que justo ha

localizado a este servidor un instante antes de ser seleccionado como tutor por el otro dispositivo, no tuviese aún registrado dicho hilo en el "hilo servidor", por lo que no recibirá ningún mensaje advirtiéndole de que el tutor ha abandonado su rol momentáneamente, permaneciendo por lo tanto a la espera de recibir si es o no seleccionado por el servidor como tutor. Esto que supondría un problema ya que se estarían dejando hilos abiertos en la aplicación.

Para resolver el problema de la posible existencia de "hilos de comunicaciones servidoras especiales" que se quedarán abiertos permanentemente, se decidió que cuando alguna de ambas partes diese por finalizado el chat, la aplicación se encargase de avisar a los clientes especiales por todos los "hilos de comunicación servidoras especiales" que el tutor abandona su rol, para que se cierren los "hilos de comunicación clientes especiales", y a continuación se cerrarán todos los "hilos de comunicación servidoras" (el correspondiente al "hilo chat servidor" y los "hilos de comunicaciones servidoras especiales" en caso de existir).

De esta forma, en caso de se hubiese producido un caso especial, y hubiese algún cliente especial que tuviera en su lista de posible tutores a un servidor que ya no estuviese disponible por estar ocupado ayudando a otro alumno. Al cerrarse el chat que mantenía ocupado al servidor, por alguna de ambas partes, se avisará a este alumno, que tiene su lista de posibles tutores incorrecta, para que elimine al tutor de la misma y cierre la comunicación establecida con él; y en caso de que fuese el único tutor disponible en ella, le pedirá que vuelva a introducir las condiciones de búsqueda para localizar a un tutor.

Si el cliente especial seleccionase a este tutor, que ya está ocupado ayudando a otro usuario, antes de que se cierre el chat, no se le permitirá contactar con él mostrándole un mensaje indicándole la situación del tutor seleccionado, y acto seguido le volverá a mostrar la pantalla de selección de rol.

Por lo tanto, si el mensaje recibido por el "hilo chat servidor" coincide con el mensaje reservado para que el cliente comunique que el alumno ha abandonado el chat, el "hilo chat servidor" deberá avisar a todos los clientes a través de todos los "hilos de comunicaciones servidoras" que el tutor abandona su rol. Después, volverá a hacer que el dispositivo sea visible por todos los dispositivos:

LDevice.setDiscoverable(DiscoveryAgent.GIAC);

El siguiente paso consistirá en avisar al cliente con el que tenía abierto el chat por el "hilo de comunicación servidora", y al resto de clientes especiales por los "hilos de comunicaciones servidoras especiales", que la comunicación ha terminado para que dejen de esperar y se cierren. Acto seguido, se reiniciará el "hilo servidor".

Por último, antes de cerrarse el "hilo chat servidor", este comprobará si se llegó a intercambiar algún mensaje, entre el alumno y el tutor, por el chat. De no haberse transmitido nada, el "hilo chat servidor" volverá a mostrar por pantalla el Form con el mensaje "Servidor de Teledocencia en marcha..." y se cerrará. En caso contrario, el "hilo chat servidor" mostrará un mensaje de alerta al usuario indicándole que el alumno cerró el chat, a continuación mostrará la conversación mantenida para que el tutor pueda extraer aquella información que le interese, y por último se cerrará. El usuario podrá

cerrar la ventana con la conversación mantenida cuando desee empleando la acción "Volver" y de nuevo visualizará por el display el Form con el mensaje "Servidor de Teledocencia en marcha...".

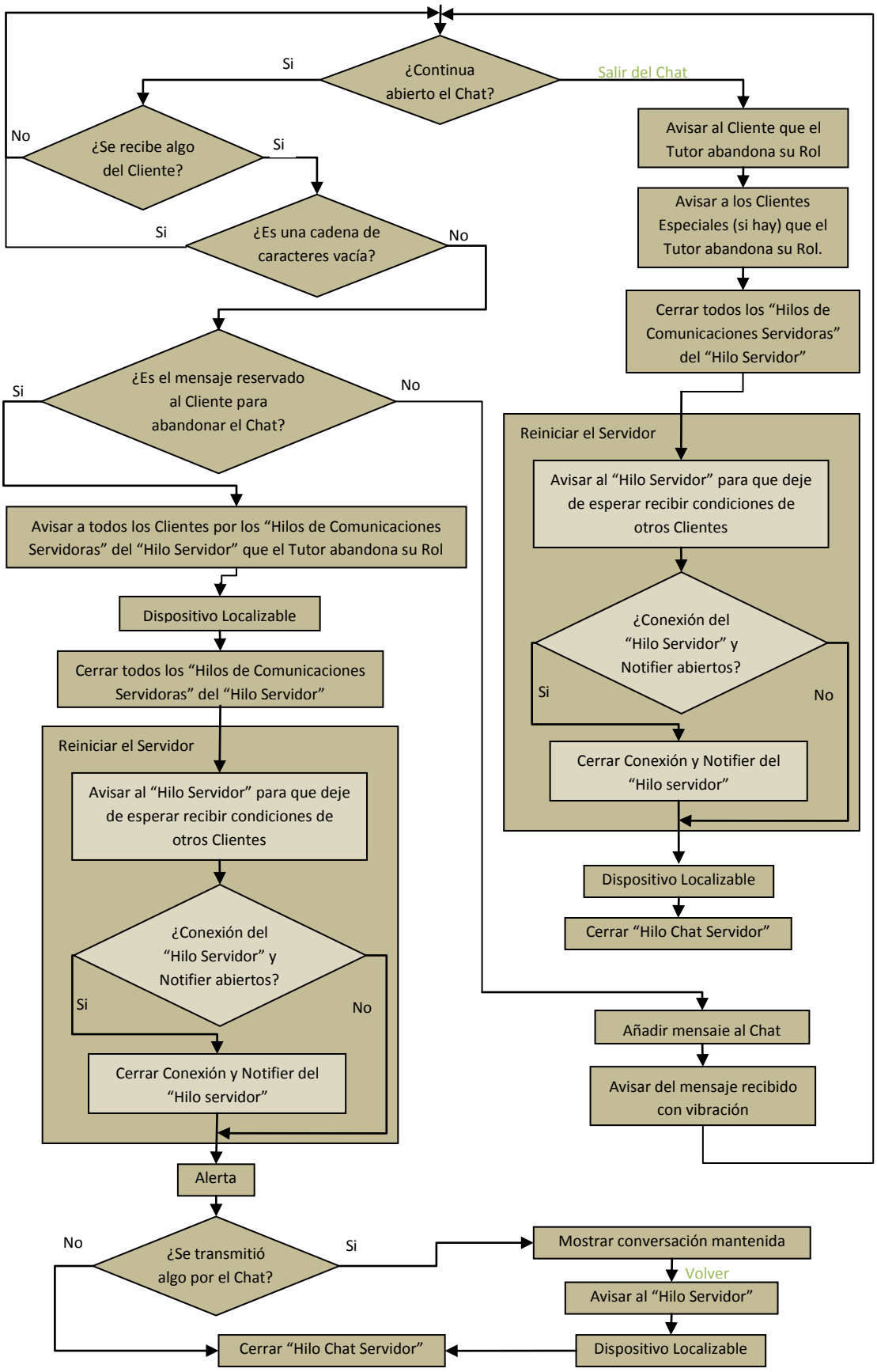
Como hemos visto, durante la ejecución del "hilo chat servidor" el usuario podrá realizar tres acciones, que ahora vamos a explicar más detenidamente:

- Enviar: con esta acción el usuario enviará al alumno con quien mantiene una comunicación un mensaje con aquello que ha escrito en el TextField del chat. Después, insertará dicho mensaje en el Form que compone el chat precedido por la cabecera "Tutor:" para que sea más fácil seguir la conversación. Los mensajes se insertarán en el form secuencialmente, lo que significa que el último mensaje enviado o recibido se encontrará en la última posición del Form (sin tener en cuenta el TextField). Por último la aplicación vaciará el TextField, para que el usuario no tenga que borrar manualmente el último mensaje enviado, y mostrará por pantalla al usuario el chat modificado con el mensaje enviado.

- Salir del Chat: En caso que el propio usuario sea quien desea poner fin a la comunicación por chat, lo hará a través de esta acción. De ser así, la aplicación deberá avisar al alumno con el que mantenía comunicación que el tutor abandona el chat, para que en el cliente también se cierre la comunicación. El siguiente paso será comunicar a los clientes especiales, en caso de que existan, que el tutor abandona su rol para que se cierren las comunicaciones también en el lado del cliente. Acto seguido, se cerrarán todos los "hilos de comunicaciones servidoras" del "hilo servidor" (el correspondiente a la comunicación mantenida y los de los casos especiales, en caso de existir) y finalmente se reiniciará el servidor. Por último, se hará de nuevo visible al dispositivo para que otros usuarios puedan localizarlo como tutor, se cerrará el "hilo chat servidor", y se mostrará al usuario por pantalla el Form con el mensaje "Servidor de Teledocencia en marcha.

- Volver: esta acción únicamente estará disponible cuando después de haberse mantenido un chat entre el tutor y un alumno, en el cual haya existido un intercambio de información, sea el cliente quien dé por finalizado el chat. En este caso, se mostrará al tutor (que puede estar interesado por algún dato de la conversación mantenida) por pantalla, un Form con la conversación mantenida y esta única acción. Cuando el usuario haya recuperado la información que le interesaba deberá pulsar en esta acción; en ese momento la aplicación avisará al "hilo servidor" que estaba a la espera y mantenía al dispositivo como no localizable para que continúe, a continuación hará visible al dispositivo servidor para que otros puedan localizarle, y volverá a mostrar al usuario por pantalla el Form con el mensaje "Servidor de Teledocencia" en marcha.

La ilustración 22 muestra el diagrama de Flujo correspondiente al "hilo chat servidor".



Nota: El usuario podrá enviar también mensajes en todo momento con la opción "Enviar".

Ilustración 22: Diagrama de Flujo "Hilo Chat Servidor"

5.1.2.2 Rol de Alumno (o cliente)

En caso de que el usuario escoja el rol de "Alumno" la aplicación (a través de la clase Teledocencia) mostrará al usuario una alerta con las instrucciones de uso, en la cual le pedirá que escoja la carrera sobre la que necesita tutoría. A continuación le mostrará una lista con todas las posibles carreras y una única opción "Elegir".

Una vez el usuario haya escogido la carrera, la aplicación le mostrará una nueva lista compuesta por todas las asignaturas de dicha carrera, para que el usuario escoja aquella en la que necesita ayuda con la opción "Elegir".

En el siguiente paso, la aplicación volverá a mostrar al usuario una alerta con las instrucciones de uso, en la que le pedirá que introduzca la condición de búsqueda del tutor. A continuación, le mostrará una lista con las condiciones de búsqueda posibles:

- Tutores con nota Superior a un valor: si el usuario selecciona esta opción, la aplicación le mostrará por pantalla un Form con el resumen de las opciones de búsqueda escogidas por el tutor (carrera, asignatura, condición de búsqueda), además de un TextField para que introduzca el valor umbral para el que se buscarán tutores con una nota superior, y una única opción "Buscar" para iniciar la búsqueda. Si el usuario comienza la búsqueda de tutores sin introducir ningún valor en el TextField, la búsqueda tomará como valor por defecto el "0".

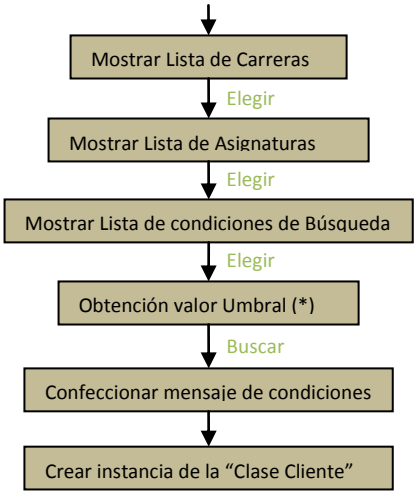
- Tutores con nota Igual a un valor: esta opción es idéntica a la anterior, con la única diferencia que ahora no se bucarán tutores con una nota superior a la introducida por el alumno sino igual a esta.

- Cualquier Tutor: en este caso, la aplicación mostrará un Form al usuario en el que le informara que va a realizar la búsqueda de todos los tutores disponibles (con notas entre "0" y "9") para la asignatura elegida de la carrera selecciona; y le dará como única opción "Buscar".

- Tutores con nota Superior a mi nota: esta condición de búsqueda únicamente estará disponible en el caso de que el usuario tenga almacenado en el Record Store el registro de la carrera para la que precisa ayuda. La aplicación tendrá que acceder a dicho registro y obtener el valor con el que el usuario evaluó la asignatura escogida. A continuación, la aplicación mostrará al usuario por pantalla un Form con el resumen de las opciones de búsqueda escogidas por el usuario (carrera, asignatura, condición de búsqueda) y la nota de su autoevaluación. Al igual que en los casos anteriores, la única opción que se le dará al usuario será la de comenzar la búsqueda con la acción "Buscar".

Finalmente, cuando el usuario escoja la acción "Buscar", la aplicación confeccionará un mensaje que incluya las condiciones de búsqueda impuestas por el cliente, así como su Friendly name y se lo pasará como parámetro a una instancia de la "clase cliente".

En la ilustración 23, se muestra el diagrama de Flujo correspondiente a la creación de la instancia de la "clase cliente".



(*) La obtención del valor umbral puede realizarse accediendo al Record Store, leyendo el valor introducido por el usuario, o tomando uno por defecto.

Ilustración 23: Diagrama de Flujo "Elección rol de Alumno"

5.1.2.2.1 Clase cliente

Se creará una instancia de esta clase en el hilo principal de la aplicación, cuando el usuario escoge el rol de alumno (cliente); y recibirá como parámetro las condiciones que debe cumplir un servidor para poder ser un posible tutor de este alumno. Será la encargada de iniciar la "búsqueda de dispositivos", y la "búsqueda de servicios" en cada uno de los dispositivos localizados. Además, creará un "hilo de comunicación cliente" por cada servidor localizado que tenga el servicio correspondiente a nuestra aplicación (si el usuario lo autoriza), y a través de este hilo les enviará las condiciones, impuestas por el usuario. Esperará las contestaciones de los servidores y creará una lista con aquellos que cumplen con estas condiciones para que el usuario seleccione uno como tutor. Por último, comunicará a todos los posibles tutores la elección del alumno mediante los "hilos de comunicación cliente", cerrándose los hilos establecidos con los posibles tutores no elegidos, mientras que el "hilo de comunicación cliente" establecido con el tutor elegido abrirá un "hilo chat cliente" para poder comunicarse con él.

En primer lugar, al escoger la aplicación el rol de alumno, la aplicación deberá hacer que el dispositivo no sea localizable por el resto de terminales cuando estos realicen una búsqueda de dispositivos; dado que va a desempeñar la función de cliente y no de servidor en este caso.

El siguiente paso, consistirá en mostrar al usuario por pantalla un Form que le indique que se está realizando la búsqueda de tutores con las condiciones introducidas. En este Form, que denominaremos de búsqueda, se incluye un objeto Gauge [1] que representa a Duke (mascota de java) en movimiento, para que en el caso de que la búsqueda tardase más de lo previsto el usuario no piense que la aplicación se ha quedado colgada; así como un botón con la opción "Cancelar" que le permitirá al usuario abandonar la búsqueda de tutores en todo momento.

En caso de que el usuario desee finalizar la búsqueda de tutores, antes de que esta encuentra algún posible tutor que cumpla con sus requisitos, mediante la acción "Cancelar"; la aplicación deberá encargarse de cerrar correctamente la "búsqueda de dispositivos" así como las "búsquedas de servicios" que hayan comenzado en los distintos dispositivos localizados, además deberá cerrar los posibles hilos que se hayan abierto en el cliente durante estas búsquedas y comunicarle al servidor dicha acción para que también cierre los hilos abiertos por su causa, por último mostrará al cliente por pantalla la lista de selección de rol y se cerrará el cliente.

Por lo tanto, el Form de búsqueda, será visible desde que arranque la instancia cliente hasta que el usuario cancele las búsqueda de tutores con la acción "Cancelar", o bien, hasta que esta finalice y se haya localizado al menos un tutor que cumple los requisitos y puede ser elegido como tutor por el cliente (momento en el que se confeccionará una lista con todos los posible tutores como veremos más adelante).

De esta forma, mientras el usuario visualiza el Form de búsqueda, el cliente comenzará la "búsqueda de dispositivos":

DAgent.startInquiry(DiscoveryAgent.GIAC, this);

Como se explicó en el apartado *2.4.6.1.1 Descubrimiento de dispositivos* de esta memoria, por cada dispositivo localizado la aplicación llamará al método "deviceDiscovered()" pasándole como parámetro un objeto de tipo RemoteDevice que representará al dispositivo localizado. Este método será el encargado de obtener el friendlyname (en caso de que lo tenga), y la dirección Bluetooth del dispositivo localizado (muy útil para escribir las trazas del programa entre otras cosas); y además comenzará la búsqueda de servicios en ese dispositivo, para comprobar si tiene cargada nuestra aplicación.

Una vez finaliza la "búsqueda de dispositivos" (ocurre automáticamente transcurrido un tiempo desde que se inició) o se cancela, la aplicación llamará al método "inquiryCompleted()", que se encargará de comprobar si la búsqueda ha finalizado correctamente, y de si se ha localizado algún dispositivo. De ser así, informará (a nivel de traza o log) del número de dispositivos localizados, así como de que la búsqueda finalizó correctamente. En caso contrario, además de informar a nivel de traza de esta situación y del motivo de finalización de la "búsqueda de dispositivos", avisará al usuario con una alerta informativa, le mostrará de nuevo por pantalla la lista de selección de rol y por último se cerrará el cliente.

Por lo tanto, por cada dispositivo localizado, arrancará una "búsqueda de servicios". Como se explicó en el apartado *2.5.1.2.2 Descubrimiento de Servicios*, en caso de localizar el servicio correspondiente a nuestra aplicación en ese dispositivo, la aplicación llamará al método "servicesDiscovered()" pasándole como atributo un array de objetos ServiceRecord que encapsulan los atributos del servicio localizado.

A partir del ServiceRecord la aplicación obtendrá la URL del servidor localizado que tiene el servicio deseado, y creará un "hilo de comunicación cliente" al que le pasará como parámetro esta URL. A su vez, el "hilo de comunicación cliente" en su instanciación intentará crear una conexión (con, in, out) con este servidor, pero al

tratarse de una conexión no segura, el módulo de seguridad de J2ME pedirá confirmación al usuario para abrir una conexión Bluetooth cliente.

En nuestro caso particular, se ha configurado el módulo de seguridad J2ME del Wireless Toolkit (ver Anexo 1) para que únicamente solicite esta autorización la primera vez que se abre un "hilo de comunicación cliente", de forma que guardará la opción escogida por el usuario (de autorizar o de no autorizar), para hacer lo mismo cada vez que se abra un nuevo "hilo de comunicación cliente". Se escogió esta opción porque puede resultar tedioso tener que autorizar a cada "hilo de comunicación a abrir una conexión cada vez que se abre. Aunque también hay que tener en cuenta que en el caso de que el usuario escogiese no autorizar a abrir la conexión Bluetooth cliente, la aplicación ya no le permitirá abrir ninguna después (abría que cerrar dicho cliente y volverle a abrir).

Si el usuario acepta establecer dicha conexión (con, in, out) con el servidor la aplicación arrancará el "hilo de comunicación cliente", y con esto se dará por finalizada la "búsqueda de servicios" en este dispositivo. Mientras que, en el caso de que el usuario no autorizase a abrir esta conexión, el "hilo de comunicación cliente" avisará a la "clase cliente" de la situación y finalizará así también la "búsqueda de servicios" en este dispositivo.

Una diferencia importante entre la "búsqueda de servicios" y la "búsqueda de dispositivos" es que en la "búsqueda de dispositivos" únicamente se llama al método "inquiryCompleted()" cuando ya no se van a localizar más dispositivos. Mientras que, el método "serviceSearchCompleted()" se llamará cada vez que finalice la "búsqueda de servicios" realizada en cada uno de los dispositivos localizados mediante la "búsqueda de dispositivos".

Por lo tanto, cuando finalice o se cancele la "búsqueda de servicios" en un dispositivo, la aplicación llamará al método "serviceSearchCompleted()" que en primer lugar deberá comprobar si han finalizado todas las "búsquedas de servicios" realizadas. De no ser este el caso, la aplicación informará (a nivel de traza o log) del motivo de finalización de la búsqueda. Sin embargo, si la "búsqueda de servicios" que ha terminado es la única que quedaba por finalizar; entonces, además de informar (a nivel de traza o log) del motivo de finalización de la búsqueda, la aplicación deberá comprobar si alguno de los dispositivos localizados tiene el servicio requerido.

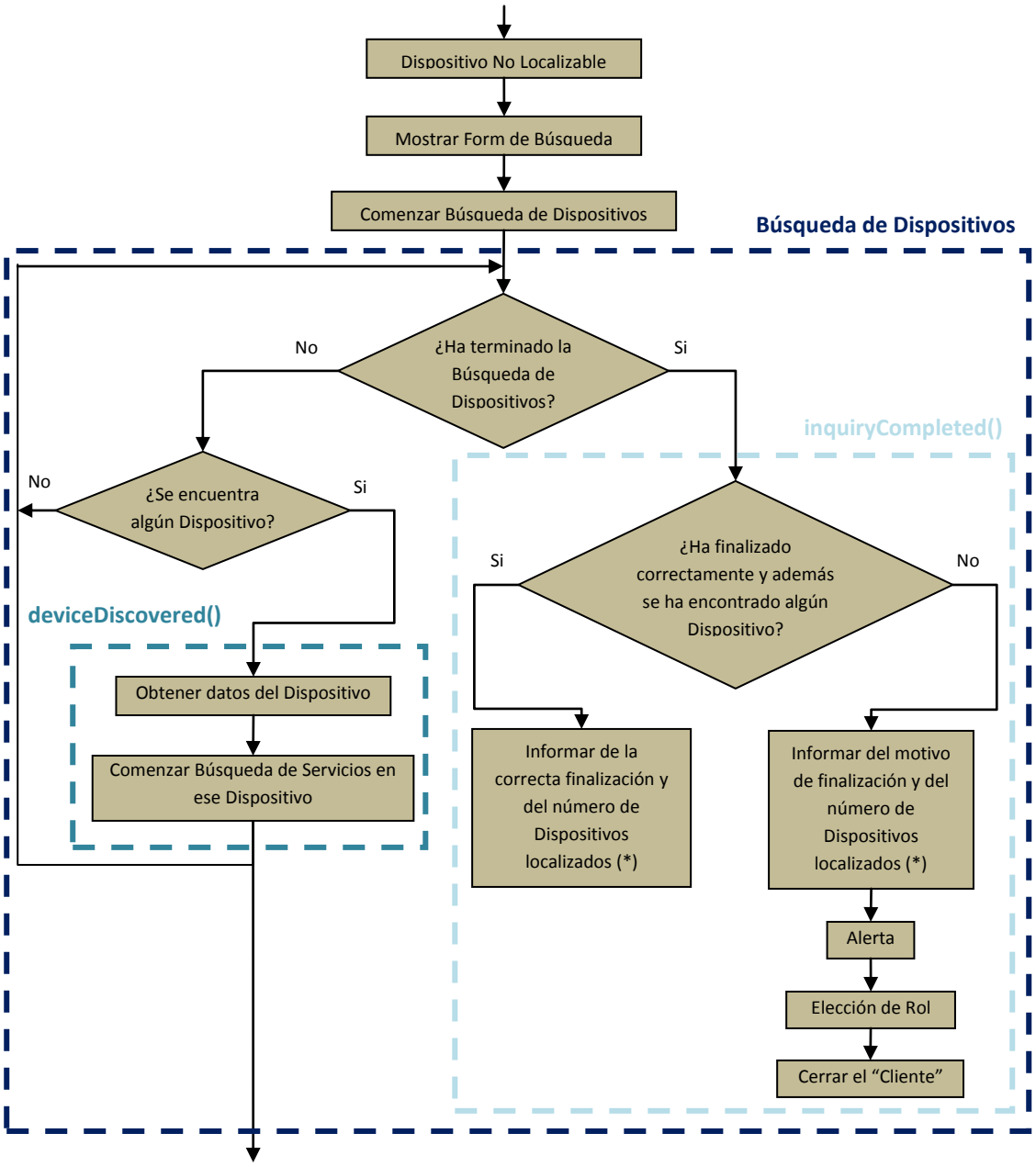
En caso de que ninguno de los dispositivos localizados (servidores) tuviese el servicio deseado, se informará de ello al usuario, se le mostrará por pantalla de nuevo la lista de selección de cliente, y a continuación se cerrará el cliente. Si por el contrario, existiese algún dispositivo que prestase este servicio, la clase cliente deberá verificar si alguno de los tutores con los que se ha puesto en contacto mediante el "hilo de comunicación cliente" cumple con las condiciones impuestas por el usuario.

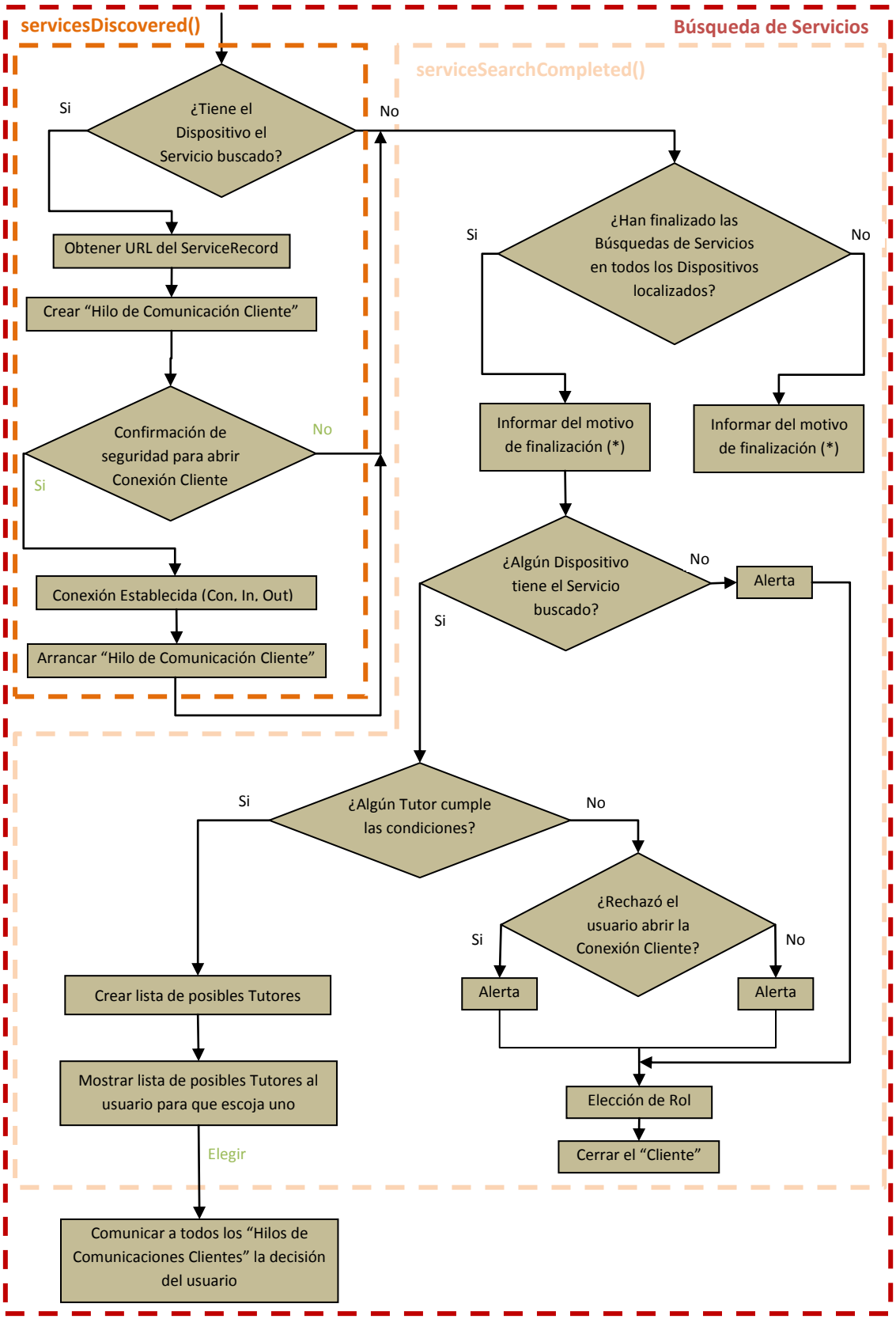
De no cumplir ningún tutor con estas condiciones, la aplicación deberá en primer lugar comprobar si esto se debe a que el propio cliente no permitió abrir una conexión Bluetooth cliente, en cuyo caso mostrará una alerta en la que le explicará que no podrá contactar con ningún tutor a menos que se permita establecer dicha conexión; y acto seguido le mostrará la pantalla de selección de tutor y se cerrará el cliente. Si este no es el motivo, se informará al usuario mediante una alerta que ningún tutor cumple los

requisitos establecidos y a continuación se mostrará la pantalla de selección de rol justo antes de finalizar el cliente.

Por último, en caso de que si existiese al menos un tutor que cumpliese con las condiciones impuestas por el cliente, la aplicación creará con ayuda de los "hilos de comunicaciones cliente" una lista con todos los tutores que cumplen con estos requisitos y a los que denominaremos como "posibles tutores" ya que de esta lista el cliente deberá "Elegir" a uno para establecer el chat. Mediante la acción "Elegir" el cliente comunicará a todos los "hilos de comunicaciones cliente" la elección realizada por el cliente; de forma que únicamente continuará en ejecución el "hilo de comunicación cliente" establecido con el dispositivo elegido como tutor, cerrándose el resto como veremos en el siguiente apartado de la memoria.

En la ilustración 24, se muestra el diagrama de Flujo correspondiente a la "clase cliente".





(*) Esta información se realizará a nivel de traza (o log), de forma que, será invisible al usuario.

Nota: El usuario podrá abandonar la búsqueda de tutores en cualquier momento con la opción "Cancelar".

Ilustración 24: Diagrama de Flujo de la "Clase Cliente

5.1.2.2.2 Hilo de comunicación cliente

Como hemos visto en el apartado anterior, la "clase cliente" establece un "hilo de comunicación cliente" para cada servidor detectado. Cada uno de estos hilos se encargará de transmitir las condiciones impuestas por el cliente a cada uno de estos dispositivos y esperar su respuesta. En caso de que su servidor cumpla los requisitos, deberá comunicárselo a la "clase cliente", para que genere una lista de forma que el cliente pueda seleccionar uno, y esperarán hasta que este haga su elección mientras que el resto se cerrarán directamente. Finalmente cuando el cliente elija a un tutor, la "clase cliente" informará a todos los "hilos de comunicaciones clientes" aún activos de la decisión, para que se la comuniquen a los servidores, y posteriormente se cierren todos a excepción del mantenido con el tutor elegido que abrirá un "hilo chat cliente" para comunicarse con él.

En primer lugar, hay que tener en cuenta que es durante la instanciación de este hilo cuando se intenta establecer una conexión con el servidor cuya URL hemos recibido como parámetro. En caso de que el cliente no autorice realizar dicha conexión la "clase cliente" será la encargada de tratar esta situación sin llegar a arrancar el hilo, como se ha explicado en el apartado anterior.

El primer paso que realizará el hilo cuando sea arrancado, consistirá en enviar al servidor, a través de la conexión establecida, las condiciones impuestas por el cliente que debe cumplir un servidor para poder ser considerado como un posible tutor.

A continuación, el "hilo comunicación cliente" quedará a la espera de que el servidor evalúe las condiciones recibidas y le responda indicándole si es o no un posible tutor para este cliente. Si durante la espera el cliente cancelase la búsqueda de tutores, se cerrará el "hilo de comunicación cliente" como ya hemos explicado.

Sin embargo, si el usuario no cancela la búsqueda de tutores, es posible que cuando el "hilo comunicación cliente" reciba el mensaje que estaba esperando procedente del servidor, este no sea la evaluación de las condiciones, sino un mensaje reservado de la aplicación que sirve para indicar al cliente que el servidor ha decidido abandonar su rol en ese momento, y por lo tanto no se le debe tener en cuenta como tutor. Por lo tanto cuando el "hilo de comunicación cliente" reciba un mensaje del servidor deberá en primer lugar verificar si se trata de la respuesta a las condiciones o de dicho mensaje reservado.

Si la repuesta del servidor es el mensaje reservado por la aplicación para indicar que el tutor abandona su rol, el "hilo de comunicación cliente" deberá cerrar la conexión establecida en caso de que estuviese abierta y a continuación finalizará su ejecución. En caso contrario, el mensaje enviado por el servidor será la evaluación que ha hecho este sobre las condiciones impuestas por el cliente, indicando por lo tanto si el servidor es un posible tutor o no.

En caso de que el mensaje recibido nos indique que el servidor no cumple los requisitos impuestos por el cliente para ser su tutor, el "hilo de comunicación cliente" cerrará la conexión establecida con este servidor, y por último se cerrará el propio "hilo de comunicación cliente".

Por otro lado, si el mensaje nos indica que el servidor es un posible tutor, porque cumple las condiciones, el "hilo de comunicación cliente" deberá añadir a la lista de posibles tutores de la "clase cliente", un elemento compuesto por el FriendlyName y la nota con que se evaluó el tutor (que también recibe en este mensaje), para que la "clase cliente" le muestre dicha lista al usuario y este escoja un tutor, como ya se ha comentado en el apartado anterior. Una vez hecho esto, el "hilo de comunicación cliente" quedará a la espera de que el usuario elija a un tutor de la lista, momento en el cual, la "clase cliente" comunicará la decisión tomada por el usuario a todos los "hilos de comunicaciones clientes" establecidos con los posibles tutores.

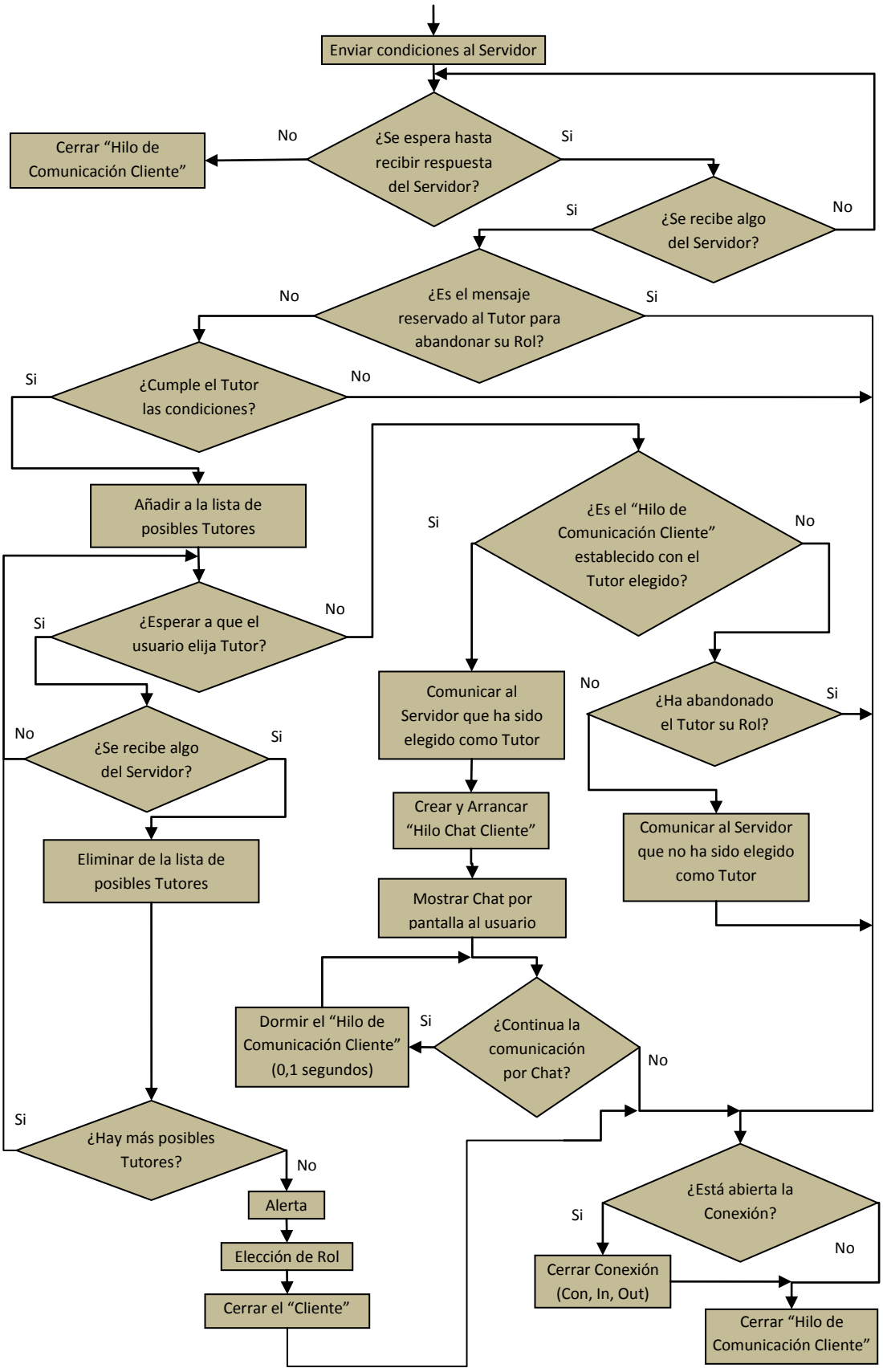
Durante esta espera, es posible que el "hilo de comunicación cliente" reciba por la conexión establecida con el servidor, el mensaje reservado para indicar que el usuario con el rol de tutor ha decidido abandonar este rol. En este caso, la aplicación se ocupará de eliminar el elemento correspondiente a este tutor de la lista de posibles tutores de la "clase cliente" para que el usuario no seleccione un tutor que ya no existe. Si este era el único elemento de la lista de posibles tutores, se mostrará una alerta al usuario indicándole que el único tutor que cumplía las condiciones ha abandonado su rol, le mostrará por pantalla la lista de selección de rol, se cerrará la instancia de la "clase cliente", y por último se cerrará la conexión del "hilo de comunicación cliente" en caso de estar abierta, así como el propio "hilo de comunicación cliente". Sin embargo, si hay más elementos en la lista de posibles tutores, el "hilo de comunicación cliente" continuará esperando a que el usuario elija un tutor de entre los que quedan.

Cuando el usuario elige finalmente a un tutor para que le ayude a resolver sus dudas, de la lista de posibles tutores, la "clase cliente" se encargará de comunicar a todos los "hilos de comunicaciones clientes", que están a la espera, cual ha sido la decisión, es decir, si el servidor con el que mantiene conexión ha sido seleccionado como tutor o no.

En caso que el servidor con el que ha establecido conexión el "hilo de comunicación cliente" no haya resultado elegido como tutor, la aplicación deberá comprobar en primer lugar si dicho tutor había abandonado su rol, ya que de no haberlo hecho el "hilo de comunicación cliente" deberá comunicar al servidor que no ha resultado elegido como tutor, para que este también cierre el "hilo de comunicación servidor" que mantenía con nuestro cliente. Por último, el "hilo de comunicación cliente" deberá cerrar la conexión que mantenía con el servidor (en caso de que permanezca aún abierta) y finalizará su ejecución.

Para terminar, si el servidor con el cual el "hilo de comunicación cliente" tiene abierta una conexión resulta elegido como tutor por el usuario, la aplicación comunicará a dicho servidor que ha resultado elegido como tutor de este cliente (para que el "hilo servidor" cierre todos los "hilos de comunicaciones servidoras" abiertos a excepción de este), creará y arrancará un "hilo chat cliente" para comunicarse con él, mostrará al usuario por pantalla el Form chat y esperará a que finalice la comunicación por chat entre ambos usuarios momento en el que cerrar las conexiones abiertas así como el "hilo de comunicación cliente".

En la ilustración 25, se muestra el diagrama de Flujo correspondiente al "hilo de comunicación cliente".



Nota: El usuario podrá abandonar la búsqueda de tutores en cualquier momento con la opción "Cancelar".

Ilustración 25: Diagrama de Flujo "Hilo de Comunicación Cliente"

5.1.2.2.3 Hilo chat cliente

Es el encargado de establecer con el servidor elegido por el usuario como tutor, una conversación a modo de chat. Con esta finalidad utilizará la comunicación establecida en el "hilo de comunicación cliente", mediante la cual escuchará permanentemente y en caso de recibir algún mensaje proveniente del tutor se lo mostrará por pantalla al alumno. Y en caso de que el alumno escriba algún mensaje se encargará de enviárselo al servidor.

Este hilo es muy parecido al "hilo chat servidor". Al igual que en el "hilo chat servidor", durante la ejecución de este hilo el usuario verá un chat creado a partir de un Form, que será idéntico y tendrá las mismas opciones que el utilizado en el "hilo chat servidor". Además, el "hilo chat cliente" también se ejecutará de forma continua hasta que alguno de los usuarios implicados decida cerrar el chat.

En primer lugar, el "hilo chat cliente" deberá comprobar, mediante la conexión establecida con el servidor (con, in, out) a través del "hilo de comunicación cliente", si este nos envía algún mensaje (por el "hilo chat servidor"). En caso de no recibir nada, continuará comprobándolo hasta que alguno de los dos cierre el chat.

Si el "hilo chat cliente" recibiese algún mensaje del servidor, lo primero que deberá hacer será comprobar si se trata de una cadena vacía, ignorándolo de ser así y quedando a la espera de recibir nuevos mensajes al igual que sucedía en el "hilo chat servidor". Sin embargo, si el mensaje recibido no es un string vacío, el "hilo chat cliente" comprobará si se trata de uno de los tres mensajes reservados al servidor para indicar que abandona su rol, cierra el Chat, o que no está disponible; o bien de un mensaje escrito por el tutor para el alumno en el chat.

Si el mensaje recibido, es uno de los mensajes reservados para indicar que el tutor cierra el chat o que el tutor abandona su rol, la aplicación deberá ocuparse de cerrar el "hilo de comunicación cliente" establecido con ese tutor, informar al usuario mediante una alerta de la situación y comprobar si se llegó a intercambiar algún mensaje, entre el alumno y el tutor, por el chat. De no haberse transmitido nada, el "hilo chat cliente" se cerrará la instancia de la "clase cliente" y a continuación se cerrará el propio "hilo chat cliente". En caso contrario, el "hilo chat cliente" mostrará al cliente la conversación mantenida para que pueda extraer aquella información que le interese. El usuario podrá cerrar la ventana con la conversación mantenida cuando desee empleando la acción "Volver", visualizándose por el display la lista de selección de rol, y cerrándose también la instancia de la "clase cliente" y el "hilo chat cliente".

En caso de que el mensaje recibido, sea el mensaje reservado al tutor para indicar que ya no está disponible, la aplicación deberá cerrar el "hilo de comunicación cliente", mostrar al usuario una alerta explicándole lo sucedido, y finalmente cerrar la instancia de la "clase cliente" y del "hilo chat cliente". Este mensaje reservado, únicamente se recibirá en el caso de que el cliente sea en realidad un "cliente especial" (ver aparatado *5.1.2.1.3* Hilo chat servidor), y escoja de la lista de posible tutores a un "tutor virtual" (visible en la lista de posible tutores de nuestro cliente especial a pesar de ya no estar disponible) antes de que este cierre el chat que tiene abierto con otro cliente.

Por último, si el mensaje recibido se trata de un mensaje escrito por el tutor en el dispositivo servidor y enviado por el "hilo chat servidor" a través de la conexión del "hilo de comunicación servidor" como ya hemos visto. En este caso el hilo añadirá dicho mensaje al Form que se muestra al usuario por pantalla a modo de chat, le avisará de que se ha recibido un mensaje por parte del tutor mediante un tono de vibración, y volverá a ejecutarse (esperando recibir algo del servidor).

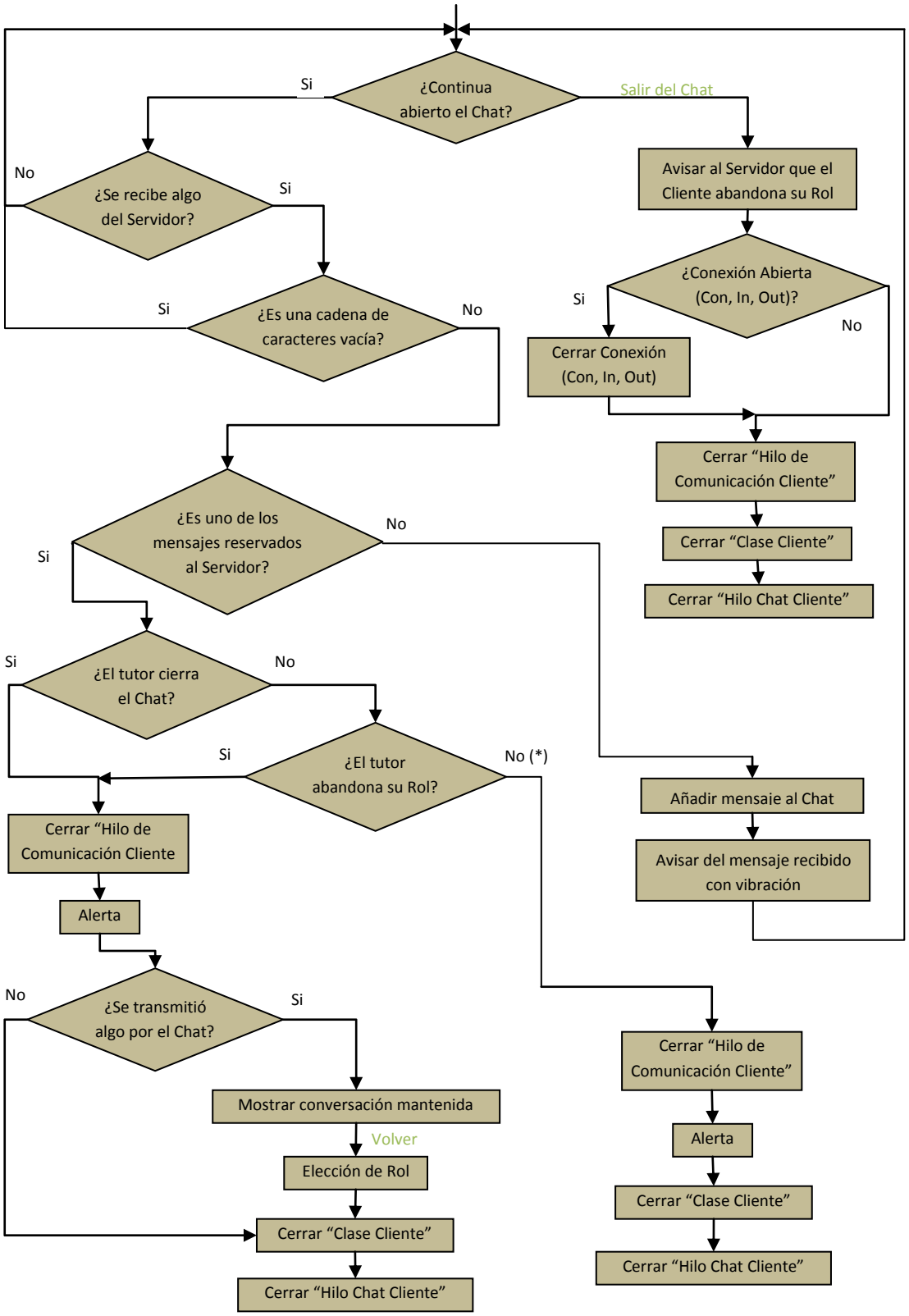
Como hemos visto, durante la ejecución del "hilo chat cliente" el usuario podrá realizar tres acciones, que ahora vamos a explicar más detenidamente:

- Enviar: acción a través de la cual el usuario enviará un mensaje que ha escrito en el TextField del chat, al tutor con quien mantiene una comunicación. Una vez enviado, la aplicación insertará dicho mensaje precedido por la cabecera "Alumno:" en el Form que compone el chat. Al igual que ocurría en el "hilo de comunicación servidora" los mensajes se insertarán en el Form secuencialmente, lo que significa que el último mensaje enviado o recibido se encontrará en la última posición del Form (sin tener en cuenta el TextField). Finalmente, la aplicación vaciará el TextField, y mostrará por pantalla al usuario el chat modificado con el mensaje enviado.

- Salir del Chat: el usuario usará esta acción para finalizar la comunicación por chat. De ser así, la aplicación deberá avisar al tutor con el que mantenía comunicación para que en el servidor también se cierre la comunicación. El siguiente paso será cerrar la conexión del "hilo de comunicación cliente", así como el propio "hilo de comunicación cliente". Por último, se mostrará al usuario por pantalla la lista de selección de rol, se cerrará la instancia de la "clase cliente", y se cerrará el propio "hilo chat cliente".

- Volver: esta acción únicamente estará disponible cuando después de haberse mantenido un chat entre el tutor y un alumno, en el cual haya existido un intercambio de información, sea el tutor quien dé por finalizado el chat. En este caso, se mostrará al alumno por pantalla, un Form con la conversación mantenida y esta única acción. Una vez recuperada la información, el usuario deberá pulsar en esta acción; en ese momento la aplicación mostrará al usuario por pantalla la lista de selección de rol, y a continuación se cerrará la instancia de la "clase cliente".

La ilustración 26 muestra el diagrama de Flujo correspondiente al "hilo chat cliente".



(*) Este es el mensaje reservado que recibiremos cuando el tutor seleccionado por el cliente no esté disponible, por haberlo localizado un instante antes de que otro cliente lo seleccionase como tutor, y aún se están comunicando a través del Chat

Nota: El usuario podrá enviar también mensajes en todo momento con la opción "Enviar".

Ilustración 26: Diagrama de Flujo "Hilo Chat Cliente"

5.2 Pruebas

Una vez desarrollada la aplicación, comienza la no menos importante fase de pruebas, que nos ayudará a garantizar el cumplimiento de los requisitos impuestos y a encontrar posibles errores en la aplicación.

Sería conveniente tener una batería de pruebas automáticas que se pudieran pasar en cualquier momento y que dieran como resultado si la aplicación responde como debe o no. Sin embargo, en este caso particular, la aplicación no tiene demasiadas operaciones e interacciona con el usuario, por lo que realizar una batería de pruebas automática que simulara un usuario nos llevaría demasiado tiempo y no compensa en relación al número de casos posibles de uso que se pueden determinar. Es por ello que se deben realizar las pruebas manualmente.

La fase de pruebas se puede dividir en dos partes bien diferenciadas. En primer lugar definiremos una batería de pruebas para comprobar el correcto funcionamiento del proceso de autoevaluación, por el que deberán pasar todos los usuarios al menos una vez. A continuación, se definirán las pruebas de conexión realizadas para diferente número de usuarios con distintos roles.

5.2.1 Pruebas de Autoevaluación

A medida que se han implementado las distintas operaciones, se han determinado los posibles casos de uso y se ha validado que funcionan correctamente. En la tabla 5 se registran todas las pruebas relacionadas con la inserción de registros (por primera vez y en sucesivas ocasiones), la modificación de registros y el borrado de los mismos.

CASO DE USO	COMPORTAMIENTO ESPERADO	CHK
Alta de registro	Inserción de un registro por primera vez	OK
Alta de Registro Errónea (sin evaluar ninguna asignatura o evaluándolas a "0")	La aplicación no permite guardar el registro en el Record Store	OK
Añadir otra carrera	Acción disponible hasta que se ha añadido un registro por cada carrera al Record Store	OK
Modificar Conocimientos	Modificación de las evaluaciones de un registro almacenado en el Record Store	OK
Modificar otra carrera	Acción disponible hasta que se han modificado todos los registros del Record Store	OK
Eliminar Carrera	Elimina el registro de la carrera seleccionada del Record Store	OK

Eliminar otra carrera	Acción disponible hasta que se han borrado todos los registros del Record Store	OK
Eliminar todos las carreras evaluadas	La aplicación no permite continuar y seleccionar un rol	OK
Continuar	Permite al usuario elegir su Rol, siempre que tenga al menos un registro almacenado en el Record Store	OK
Salir o colgar	Cierre de la aplicación	OK

Tabla 5: Pruebas de Autoevaluación

5.2.2 Pruebas de Conexión

Se han realizado múltiples pruebas de conexión, clasificadas en función del número de terminales y los roles empleados en cada una de ellas. Estas pruebas se registran en las siguientes tablas:

	1 TERMINAL		
	CASO DE USO	**COMPORTAMIENTO ESPERAD**	**CHK**
TUTOR	El usuario acepta abrir una conexión Bluetooth Servidora	La aplicación queda a la espera de recibir condiciones de algún cliente	OK
	El usuario rechaza abrir una conexión Bluetooth Servidora	La aplicación muestra un alarma explicando que no se puede conectar con ningún cliente y vuelve a la pantalla de selección de Rol	OK
	Salir	Finaliza el rol de tutor, y vuelve a la pantalla de selección de Rol	OK
	Colgar	Cierre de la aplicación	OK
ALUMNO	El usuario elige para la tutoría una carrera para la que no tiene un registro almacenado	La aplicación solo permite realizar tres condiciones de búsqueda	OK
	El usuario elige para la tutoría una carrera para la que tiene un registro almacenado	La aplicación le permite realizar cuatro condiciones de búsqueda	OK
	El usuario no introduce ninguna nota para realizar la búsqueda de tutores	La aplicación toma por defecto el "0"	OK

ALUMNO	El usuario realiza una búsqueda de tutores y no hay ningún servidor activo.	A una alerta, y le mostrará la pantalla de selección de rol	OK
	El usuario cancela la búsqueda de Tutores	La aplicación le mostrará la pantalla de selección de rol	OK
	Colgar	Cierre de la aplicación	OK

Tabla 6: Pruebas de Conexión con 1 Terminal

	2 TERMINALES		
	CASO DE USO	**COMPORTAMIENTO ESPERADO**	**CHK**
2 TUTORES	Los dos usuarios eligen el rol de Tutor	La aplicación queda a la espera de recibir condiciones de algún cliente en ambos casos	OK
2 ALUMNOS	Los dos usuarios eligen el rol de Alumno	Al realizar la búsqueda de tutores, ninguno de ellos localizará a ningún tutor	OK
1 ALUMNO/1 TUTOR	El cliente realiza una búsqueda de tutores y la cancela	Se informa al cliente de que la búsqueda ha sido cancelada y se le muestra la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores y el tutor no cumple las condiciones impuestas	Se informa al cliente que el tutor no cumple las condiciones y se le muestra la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores y el tutor cumple las condiciones impuestas	La aplicación le muestra el tutor en la lista de posibles tutores al usuario	OK
	El tutor abandona su rol o cierra la aplicación mientras el cliente realiza la búsqueda de tutores	El cliente no lo localiza	OK
	El tutor abandona su rol o cierra la aplicación cuando ya estaba en la lista de posibles tutores del cliente	El tutor desaparece de la lista de posibles tutores del cliente. Al ser el único se informará al usuario, y se le mostrará la lista de selección de rol	OK

1 ALUMNO/1 TUTOR	El cliente elige al tutor	Se establece el chat entre ambos	OK
	El cliente\tutor cierra el chat establecido, sin haberse transmitido ningún mensaje por él	Se avisa al tutor\cliente con quien mantenía la comunicación. El dispositivo con el rol de tutor continuará a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, cerrará dicho rol y le mostrará al usuario la pantalla de selección de rol	OK
	El cliente cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al tutor del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida. El dispositivo con el rol de tutor continuará ilocalizable hasta que cierre el Form con la conversación mantenida volviendo a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, al cerrar el Chat le mostrará al usuario la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores mientras el tutor está viendo el Form de la conversación mantenida con el cliente hasta que el mismo cliente cerró el chat.	El cliente no localizará al tutor por estar este aún recuperando información de una conversación anterior.	OK
	El tutor cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al cliente del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida para que pueda obtener los datos que le interesen. El dispositivo con el rol de tutor que ha cerrado el chat volverá a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, le mostrará al usuario la pantalla de selección de rol cuando cierre el Form con la conversación mantenida	OK

Tabla 7: Pruebas de Conexión con 2 Terminales

	3 TERMINALES		
	CASO DE USO	**COMPORTAMIENTO ESPERADO**	**CHK**
3 TUTORES	Los tres usuarios eligen el rol de Tutor	La aplicación queda a la espera de recibir condiciones de algún cliente en los tres casos	OK
3 ALUMNOS	Los tres usuarios eligen el rol de Alumno	Al realizar la búsqueda de tutores, ninguno de ellos localizará a ningún tutor	OK
1 ALUMNO/2 TUTORES	El cliente realiza una búsqueda de tutores y la cancela	Se informa al cliente de que la búsqueda ha sido cancelada y se le muestra la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores y ninguno de los tutores cumple las condiciones impuestas	Se informa al cliente que ningún tutor cumple las condiciones y se le muestra la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores y solo un tutor cumple las condiciones impuestas	La aplicación le muestra dicho tutor en la lista de posibles tutores al usuario	OK
	El cliente realiza una búsqueda de tutores y ambos cumplen las condiciones impuestas	La aplicación mostrará ambos al usuario en la lista de posibles tutores	OK
	Cualquiera de los tutores abandona su rol o cierra la aplicación mientras el cliente realiza la búsqueda de tutores	El cliente no lo localizará, de hacerlo los dos no localizará a ninguno	OK
	Cualquiera de los tutores abandona su rol o cierra la aplicación cuando ya estaba en la lista de posibles tutores del cliente	El tutor desaparece de la lista de posibles tutores del cliente, quedando únicamente el otro tutor. En caso de que lo hicieran ambos, se informará al usuario, y se le mostrará la lista de selección de rol	OK
	El cliente elige a uno de los tutores	Se establece el chat entre ambos	OK

1 ALUMNO/2 TUTORES	El cliente\tutor cierra el chat establecido, sin haberse transmitido ningún mensaje por él	Se avisa al tutor\cliente con quien mantenía la comunicación. El dispositivo con el rol de tutor continuará a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, cerrará dicho rol y le mostrará al usuario la pantalla de selección de rol	OK
	El cliente cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al tutor del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida. El dispositivo con el rol de tutor continuará ilocalizable hasta que cierre el Form con la conversación mantenida volviendo a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, al cerrar el Chat le mostrará al usuario la pantalla de selección de rol	OK
	El cliente realiza una búsqueda de tutores mientras uno de los tutores está viendo el Form de la conversación mantenida con el cliente hasta que el mismo cliente cerró el chat.	El cliente no localizará dicho tutor por estar este aún recuperando información de una conversación anterior.	OK
	El tutor cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al cliente del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida para que pueda obtener los datos que le interesen. El dispositivo con el rol de tutor que ha cerrado el chat volverá a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, le mostrará al usuario la pantalla de selección de rol cuando cierre el Form con la conversación mantenida	OK

2 ALUMNO/1 TUTOR	Cualquiera de los clientes realiza una búsqueda de tutores y la cancela	Se informa al cliente de que la búsqueda ha sido cancelada y se le muestra la pantalla de selección de rol	OK
	El tutor no cumple las condiciones impuestas por los clientes en sus búsquedas de tutores	Se informa a los cliente que el tutor no cumple las condiciones, y se les mostrará la pantalla de selección de rol	OK
	El tutor solo cumple las condiciones impuestas por uno de los clientes	La aplicación le muestra dicho tutor en la lista de posibles tutores a ese usuario	OK
	El tutor cumple las condiciones impuestas por los dos clientes	La aplicación mostrará el tutor a ambos usuarios en la lista de posibles tutores	OK
	El tutor abandona su rol o cierra la aplicación mientras los clientes realizan la búsqueda de tutores	Ningún cliente localizará al tutor	OK
	El tutor abandona su rol o cierra la aplicación cuando ya estaba en la lista de posibles tutores de alguno de los clientes	El tutor desaparece de la lista de posibles tutores de todos los clientes para los que cumplía sus condiciones. Al ser el único, se informará a los usuario, y se le mostrará la lista de selección de rol	OK
	Uno de los clientes elige al tutor para establecer un chat	Se establece el chat entre ambos, y en caso de ser un posible tutor del otro cliente desaparecerá de la lista de posibles tutores de este	OK
	El cliente\tutor cierra el chat establecido, sin haberse transmitido ningún mensaje por él	Se avisa al tutor\cliente con quien mantenía la comunicación. El dispositivo con el rol de tutor continuará a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, cerrará dicho rol y le mostrará al usuario la pantalla de selección de rol	OK

2 ALUMNO/1 TUTOR	El cliente cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al tutor del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida. El dispositivo con el rol de tutor continuará ilocalizable hasta que cierre el Form con la conversación mantenida volviendo a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, al cerrar el Chat le mostrará al usuario la pantalla de selección de rol	OK
	Uno de los clientes realiza una búsqueda de tutores mientras el tutor está viendo el Form de la conversación mantenida anteriormente.	El cliente no localizará dicho tutor por estar este aún recuperando información de una conversación anterior.	OK
	El tutor cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al cliente del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida para que pueda obtener los datos que le interesen. El dispositivo con el rol de tutor que ha cerrado el chat volverá a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, le mostrará al usuario la pantalla de selección de rol cuando cierre el Form con la conversación mantenida	OK
	Caso Especial: Un cliente (CL1) realiza una búsqueda de tutores, cuyas condiciones va a cumplir el tutor, un instante antes de que el otro cliente (CL2) seleccione al tutor para establecer un chat	Se establecerá un chat entre el CL2 y el tutor, pero el CL1 añadirá dicho tutor a su lista de posibles tutores. Si el CL2 o el tutor cierran el chat, el tutor desparecerá de la lista de posibles tutores del CL1, en este caso al ser el único se mostrará al usuario la pantalla de selección de rol. Sin embargo, si el CL1 selecciona este tutor antes, se le informará que no puede comunicarse con él y se mostrará al usuario la pantalla de selección de rol	OK

Tabla 8: Pruebas de Conexión con 3 Terminales

	4 TERMINALES		
	CASO DE USO	**COMPORTAMIENTO ESPERADO**	**CHK**
4 TUTORES	Los cuatro usuarios eligen el rol de Tutor	La aplicación queda a la espera de recibir condiciones de algún cliente en los tres casos	OK
4 ALUMNOS	Los cuatro usuarios eligen el rol de Alumno	Al realizar la búsqueda de tutores, ninguno de ellos localizará a ningún tutor	OK
1 ALUMNO/3 TUTORES	Estas Pruebas son idénticas a las realizadas con 3 terminales para 1 Alumno/2 Tutores, con la diferencia de que en este caso va ha haber un tutor más		OK
3 ALUMNO/1 TUTOR	Estas Pruebas son idénticas a las realizadas con 3 terminales para 2 Alumno/1 Tutores, con la diferencia de que en este caso va ha haber un alumno más		OK
2 ALUMNO/2 TUTORES	El/Los cliente/s realiza/n una búsqueda de tutores y la cancela/n	Se informa a el/los cliente/s de que la búsqueda ha sido cancelada y se le/s mostrará la pantalla de selección de rol	OK
	El/Los cliente/s realiza/n una búsqueda de tutores y ninguno de los tutores cumplen las condiciones impuestas	Se informa a el/los usuario/s de el/los cliente/s que ningún tutor cumple las condiciones y se le/s mostrará la pantalla de selección de rol	OK
	El/Los cliente/s realiza/n una búsqueda de tutores y solo un tutor cumple las condiciones impuestas	La aplicación muestra dicho tutor en la lista de posibles tutores a el/los usuario/s de el/los cliente/s	OK
	El/los cliente/s realiza/n una búsqueda de tutores y ambos cumplen las condiciones impuestas	La aplicación mostrará ambos tutores a el/los usuario/s de el/los cliente/s, en la lista de posibles tutores	OK

2 ALUMNO/2 TUTORES	El/Los tutor/es abandona/n su rol o cierra/n la aplicación mientras el/los cliente/s realiza/n la búsqueda de tutores	El/Los cliente/s no localizará/n a el/los tutor/es	OK
	El/Los tutor/es abandona/n su rol o cierra/n la aplicación cuando ya estaba/n en la lista de posibles tutores de el/los cliente/s	El/Los tutor/es desaparece/n de la lista de posibles tutores de el/los cliente/s. En caso de que lo hicieran ambos tutores, se informará a el/los usuario/s de el/los cliente/s, y se le/s mostrará la lista de selección de rol	OK
	Un cliente elige a uno de los tutores	Se establece el chat entre ambos, y en caso de ser un posible tutor del otro cliente desaparecerá de la lista de posibles tutores de este	OK
	El cliente\tutor cierra el chat establecido, sin haberse transmitido ningún mensaje por él	Se avisa al tutor\cliente con quien mantenía la comunicación. El dispositivo con el rol de tutor continuará a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, cerrará dicho rol y le mostrará al usuario la pantalla de selección de rol	OK
	El cliente cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al tutor del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida. El dispositivo con el rol de tutor continuará ilocalizable hasta que cierre el Form con la conversación mantenida volviendo a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, al cerrar el Chat le mostrará al usuario la pantalla de selección de rol	OK
	Uno de los clientes realiza una búsqueda de tutores mientras el tutor está viendo el Form de la conversación mantenida anteriormente.	El cliente no localizará dicho tutor por estar este aún recuperando información de una conversación anterior.	OK

2 ALUMNO/2 TUTORES	El tutor cierra el chat establecido, después de haberse utilizado para transmitir algún mensaje	Se avisa al cliente del cierre de la comunicación, y se le mostrará por pantalla la conversación mantenida para que pueda obtener los datos que le interesen. El dispositivo con el rol de tutor que ha cerrado el chat volverá a quedar a la espera de ser seleccionado por algún cliente. Mientras que el dispositivo con rol de cliente, le mostrará al usuario la pantalla de selección de rol cuando cierre el Form con la conversación mantenida	OK
	Caso Especial: Un cliente (CL1) realiza una búsqueda de tutores, cuyas condiciones va a cumplir ambos tutores (T1 y T2), un instante antes de que el otro cliente (CL2) seleccione a uno de los tutores (T1) para establecer un chat	Se establecerá un chat entre el CL2 y el T1, pero el CL1 también añadirá al T1 a su lista de posibles tutores. Si el CL2 o el T1 cierran el chat, el T1 desparecerá de la lista de posibles tutores del CL1, quedando únicamente el T2. Sin embargo, si el CL1 selecciona al T1 antes de que esto ocurra, se le informará que no puede comunicarse con él y se mostrará al usuario la pantalla de selección de rol. En caso de elegir el Cl1 al T2 que está libre se comunicará con el sin problemas.	OK

Tabla 9: Pruebas de Conexión con 4 Terminales

Capítulo VI

Conclusiones y Trabajos Futuros

Una vez concluidos tanto la introducción teórica como el desarrollo y la implementación, en este último capítulo se incluyen las conclusiones obtenidas del desarrollo del proyecto y una enumeración de posibles trabajos futuros que pueden realizarse.

6.1 Conclusiones

En primer lugar, hay que indicar que se ha alcanzado el objetivo principal que se planteó al comienzo del proyecto, al implementar una aplicación que cumple todos los requisitos y funcionalidades impuestos en la especificación. Se deseaba que cualquier usuario fuera capaz usar la herramienta sin necesidad de tener conocimientos en J2ME o en tecnología Bluetooth y, efectivamente, se ha cumplido este objetivo; ya que el usuario, en función del rol que elija, únicamente deberá centrarse en ayudar a otros usuarios a resolver dudas, o en encontrar algún usuario que le ayude a resolver sus propias dudas.

En un principio se realizó un estudio de la tecnología Bluetooth y del lenguaje J2ME con el fin de aplicar el conocimiento en el posterior desarrollo. Este análisis ha aportado valor a la herramienta ya que, por ejemplo, la aplicación es capaz de generar un tono de vibración para avisar al usuario de la recepción de un mensaje, gracias a que en la investigación de J2ME se descubrió esta funcionalidad.

Una vez realizado el estudio dio comienzo la etapa de diseño, en la que se determinó una solución válida para el problema planteado. En este punto se tomaron importantes decisiones de diseño como, por ejemplo, acotar la aplicación a las carreras del área de Telecomunicaciones impartidas en el campus politécnico de la universidad Carlos III de Madrid, o limitar a 100 el número máximo de conexiones simultaneas establecidas por la aplicación, por considerarse suficientes para elegir un tutor. Además, se determinaron los distintos componentes que formarían parte de la herramienta, así como su relación y sus operaciones principales.

Definido el diseño, se comenzó la etapa de desarrollo. En general, se siguieron los pasos planteados en el diseño previo, aunque en algún apartado particular surgió alguna complicación que hizo que se tuviera que volver a plantear el problema desde otro punto de vista. La complicaciones más importantes fueron: la que surgió al intentar establecer conexión con varios dispositivos simultáneamente (independientemente de su rol), por lo que fue necesario introducir en el diseño el manejo de hilos; y el tratamiento de los denominados clientes especiales, cuyo comportamiento no había sido previsto y exigió un análisis posterior. A la vez que se iban desarrollando las diferentes partes de la herramienta, se fueron realizando pruebas básicas para validar el correcto funcionamiento de la aplicación.

En definitiva, indicar que se han cubierto las expectativas del proyecto y que los resultados obtenidos son satisfactorios ya que se ha implementado con éxito una aplicación móvil, basada en la tecnología Bluetooth, apta para ayudar a los estudiantes a resolver sus dudas en una materia, localizando y estableciendo una conversación con otros estudiantes que también tienen conocimientos sobre el tema.

6.2 Trabajos Futuros

La aplicación diseñada es una primera versión, por lo que con el paso del tiempo y el manejo de la misma se detectarán posibles puntos de mejora, que habrá que tener en cuenta en futuras versiones. Algunas de esas mejoras podrían ser las siguientes:

- Realizar pruebas con dispositivos reales. Instalar la aplicación en dispositivos de distintos fabricantes y testear su funcionamiento.

- Ampliación y mantenimiento del catálogo de carreras. Ampliar el catálogo añadiendo el todas las carreras de todos los campus de la universidad Carlos III de Madrid, y realizar un seguimiento de las asignaturas impartidas en cada una de ellas al comienzo de cada año académico para mantenerlas actualizadas.

- Implementar un rol cliente/servidor. Desarrollar un rol que permita al usuario actuar como alumno y como tutor simultáneamente; dando preferencia a las búsquedas de tutores y al establecimiento de conexión por chat como alumno, sobre el establecimiento de conexión por chat como tutor con otro usuario, pero permitiendo establecer ambas conexiones al mismo tiempo y pasar de una a otra según le interese.

- Crear una BBDD de favoritos/excluidos. Almacenar en un registro aquellos usuarios con los que se ha mantenido una conversación y permitir al usuario de la aplicación configurar dos listas: una primera lista compuesta por aquellos usuarios que le han sido de ayuda, de forma que pueda intentar localizarlos de nuevo posteriormente; una segunda lista compuesta por aquellos usuarios con los que el usuario no quiere volver a establecer comunicación (a modo de lista negra).

- Implementar el envío de ficheros. Permitir al usuario que durante un chat pueda realizar el envío de ficheros almacenados en la memoria de su dispositivo móvil, dado que cada vez más estos se usan como dispositivos de almacenamiento externo.

La mayoría de las mejoras están relacionadas con añadir más funcionalidad a la aplicación, de forma que esta sea más útil al usuario. Anteriormente se han descrito algunas pero es posible imaginar más con el objetivo de hacer más cómodo el uso de la aplicación de cara al usuario.

Todas estas mejoras podrían realizarse en posteriores versiones de la herramienta con el objetivo de añadir más valor a la misma.

ACRÓNIMOS

ACL	Asynchronous Connectionless
AMS	Application Management System
API	Application Program Interface
ARQ	Automatic Repeat Request
AWT	Abstract Windows Toolkit
BCC	Bluetooth Control Center
BNEP	Bluetooth Network Encapsulation Protocol
BSDB	Base Structure Data Base
CDC	Connected Device Configuration
CLCD	Connected Limited Device Configuration
CMOS	Complementary Metal Oxide Semiconductor
CVM	Compact Virtual Machine
DSP	Digital Signal Processor
FEC	Forward Error Correction
FH/TDD	Frequency Hop/Time-Division Duplex
FHS	Frequency Hop Syncronization
GAJ	Gestor De Aplicaciones Java
GAP	Generis Access Profile
GOEP	Generis Object Exchange Profile
GUI	Graphical User Interface
HCI	Host Controller Interface
HID	Human Interface Device
HTTP	Hypertext Transfer Protocol
IEEE	Institute Of Electrical And Electronics Engineers
IrDA	Infrared Data Association
ISM	Industrial, Science And Medical (Band)
ISO	Internacional Standard Organization
JAD	Java Description
JAR	Java Archive
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SDK	Java 2 Software Development Kit
J2SE	Java 2 Standard Edition
JAM	Java Application Management

JNI	Java Native Interface
JRE	Java Runtime Enviroment=Jvm
JVM	Java Virtual Machine
JVMDI	Interfaz De Depuracion De La Maquina Virtual
KVM	Kilo Virtual Machine
L2CAP	Logical Link Control And Adaptation Protocol
LC	Link Controller
LM	Link Manager
LMP	Link Manager Protocol
MIDP	Mobile Information Device Profile
MMAPI	Mobile Media Api
MSA	Mobile Service Architecture
NIC	Network Interface Card
OBEX	Object Exchange
OEM	Original Equipment Manufacturer
OSI	Open System Interconected
PAN	Personal Area Network
PDA	Personal Digital Asisstant
PPP	Point To Point Protocol
RF	Radiofrrecuencia
RFCOMM	Radio Frequency Communication
RMI	Remote Method Invocation
RMS	Record Management System
SCO	Synchronous Connection Oriented
SDAP	Service Discovery Aplication Profile
SDDB	Service Discovery Databasa
SDP	Service Discovery Protocol
SIG	Special Interest Group (Blluetooth)
SPP	Serial Port Profile
TCP/IP	Transmission Control Protocol/Internet Protocol
TCS	Telephony Control Protocol Specification
UDP	User Datagram Protocol
URL	Universal Resource Locator
UUID	Universal Unique Identifier
WAP	Wireless Acces Protocol
WLAN	Wireless Local Area Network
WTK	Sun Wireless Toolkit

BIBLIOGRAFÍA

- J2ME

1. Java a tope: J2ME (JAVA 2 MICRO EDITION)
   Sergio Gálvez Rojas y Lucas Ortega Díaz
   ISBN: 84-688-4704-6

2. Pagina Web de la UC3M dedicada a tutoriales de Java 2 Micro Edition
   http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/

3. Tutorial de programación de juegos con J2ME
   http://www.mailxmail.com/curso-programacion-juegos-moviles-j2me

- Especificaciones

4. Página web de Java Community Process con la especificación CLDC (JSR 30)
   http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html

5. Página web de Java Community Process con la especificación MIDP 1.0 (JSR 37)
   http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html

6. Página web de Java Community Process con la especificación MIDP 2.0 (JSR 118)
   http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

7. Página web de Sun con la especificación APIs Java para Bluetooth (JSR 82)
   http://java.sun.com/javame/reference/apis/jsr082/

8. Página web del estándar de comunicaciones RS-232
   http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html

- MIDP

9. Tutorial web de la Universidad de Granda sobre MIDP
   http://leo.ugr.es/J2ME/MIDP/index2.htm

10. Wireless game development in Java with MIDP 2.0
Ralph Barbagallo
ISBN: 1556229984

11. Tutorial web sobre MIDP
http://grasia.fdi.ucm.es/j2me/_J2METech/MIDP.html

12. Página web de Sun sobre la arquitectura de seguridad en MIDP 2.0
http://developers.sun.com/mobility/midp/articles/permissions/

- Bluetooth

13. Bluetooth application programming with the Java APIs
C. Bala Kumar, Paul J. Kline y Timothy J. Thompson
ISBN: 1558609342

14. Tutorial web sobre Bluetooth
http://www.todosymbian.com/secartprint43.html

15. Tutorial web sobre la seguridad en Bluetooth
http://www.seguridadmobile.com/bluetooth/seguridad-bluetooth/elementos-de-seguridad.html

ANEXO 1: MANUAL BÁSICO SUN WIRELESS TOOLKIT

Este anexo tiene como objetivo aclarar los procesos de instalación, configuración y manejo de la herramienta Sun Wireless Toolkit. Para ello, en primer lugar, se comentarán los requisitos necesarios que ha de cumplir el sistema para poder realizar la instalación y la estructura de directorios que creará la misma. A continuación, se explicará la configuración y el manejo de la herramienta; y por último, se expondrá el archivo "emulator.properties" que configura las propiedades de la aplicación.

1 Instalación de la aplicación

La aplicación Sun Java Wireless Toolkit (WTK) es compatible con cualquier versión de los sistemas operativos Windows y Linux, y para su funcionamiento requiere de un entorno de programación J2SE o J2EE (JDK). Podremos descargar la última versión del entorno de programación y de la aplicación (WTK) en la URL http://java.sun.com/. Por otro lado, los requisitos de hardware mínimos que ha de tener nuestro equipo son:

100 MB de espacio libre en el disco duro
128 MB de memoria RAM
CPU Pentium III a 800 MHz

La instalación del WTK se realizará por defecto en la siguiente ruta, que denominaremos como directorio de instalación:

Windows: *C:\WTK2.5.2*
Linux: */opt/WTK2.5.2*

Durante la instalación del WTK, en este directorio, se creará un subdirectorio "app" con varias aplicaciones de demostración. Cada una de las cuales tendrá en dicha ruta un directorio de proyecto propio con su código fuente, y al que podrán acceder todos los usuarios. Además, se creará un directorio de trabajo para cada usuario, que contendrá los archivos personales de cada uno de ellos en:

Windows: *C:\Documents and Settings\Usuario\j2mewtk\2.5.2*
donde: "Usuario" es la cuenta del usuario
Linux: *~/j2mewtk/2.5.2*
donde: "~" es el home del usuario

Dentro del directorio de trabajo de cada usuario encontraremos la siguiente estructura de directorios [2]:

- appdb: directorio en el que se almacena las bases de datos (archivos RMS) de todos los proyectos del usuario.
- apps: directorio en el que se almacenan todos los proyectos del usuario. Cada proyecto tendrá un directorio propio, con su nombre y estructurado de la siguiente forma:

    - bin: contiene el fichero JAR [11] (archivo comprimido que contiene las clases generadas en la compilación del programa, y los recursos asociados a este), el fichero JAD (archivo con la información necesaria para la instalación de los MIDlets contenidos en el archivo JAR) y el manifiesto (archivo que contiene información sobre las clases del archivo JAR).
    - classes: contiene los ficheros de las clases compiladas
    - lib: contiene librerías externas.
    - res: contiene recursos asociados con el MIDlet (imágenes, sonidos,...)
    - src: contiene los ficheros fuente
    - tmpclasses: directorio temporal para uso del WTK
    - tmplib: directorio temporal para uso del WTK

- wtklib: en este directorio encontraremos el "emulator.properties" con todas las propiedades con que se ha configurado WTK.
- .settings: en este directorio encontraremos el "security.properties".
- sessions: este directorio está destinado a los resultados obtenidos con las herramientas de monitorización (inicialmente estará vacio).

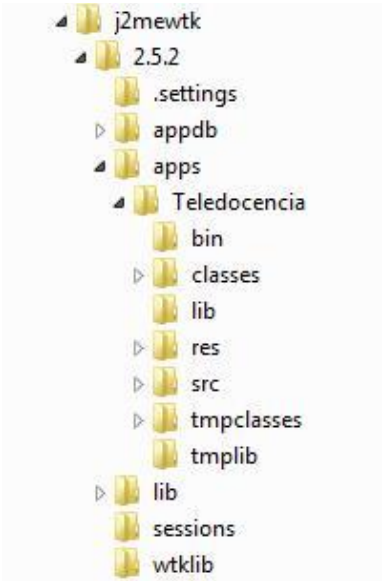


Ilustración 27: Estructura directorios de Sun Wireless Toolkit

2 Configuración y manejo de la aplicación

Una vez se conoce la estructura de directorios creada por el WTK durante su instalación en el directorio de trabajo del usuario, y en la creación de proyectos por parte del usuario; es el momento de explicar la configuración y el manejo de la herramienta. Para abrir el WTK se deberán seguir los siguientes pasos:

Windows: *Inicio > Programas > Sun Java Wireless Toolkit 2.5.2 for CLDC > Wireless Toolkit 2.5.2*
Linux: Ejecutar en el directorio de instalación/bin: *./ktoolbar*

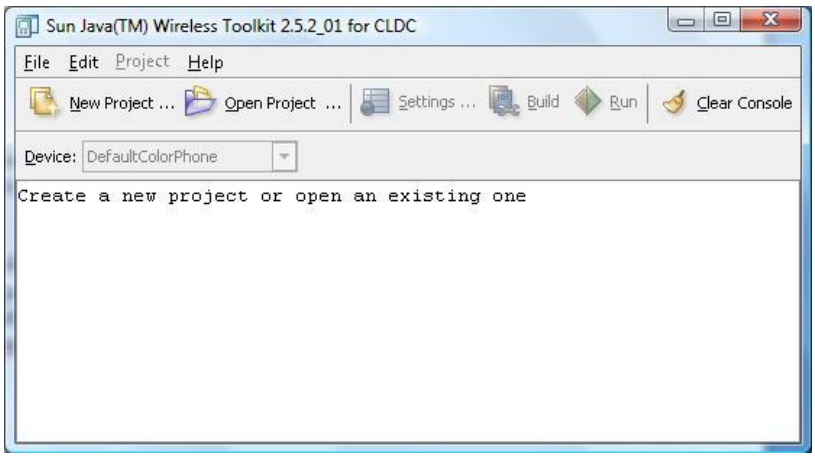


Ilustración 28: Sun Wireless Toolkit

2.1 Configuración de la herramienta

Para configurar la herramienta el usuario deberá pulsar *Edit > Prefences*, de esta forma obtendrá una ventana emergente con múltiples opciones de configuración. En este anexo se van a comentar aquellas opciones de configuración que han sido de utilidad para realizar el proyecto:

- Bluetooth/Obex: en esta categoría el usuario podrá fijar en la pestaña "Internal properties" el "Device discovery timeout", que es el tiempo máximo que la función "Device discovery" empleará para localizar dispositivos Bluetooth. Por defecto este valor es de 10s (10.000 ms) y en este proyecto se ha utilizado este valor. En la pestaña "BCC Properties" el usuario puede especificar el "Friendly name" del siguiente dispositivo que va a emular, por defecto tendrá el valor "WirelessToolkit".

- Security: en esta categoría el usuario podrá elegir que política de seguridad, definida en JSR 185(Java Technology for the Wireless Industry) o en JSR 248 (Mobile Service Architecture or MSA), deberá utilizar la aplicación. En este proyecto, se ha elegido como plataforma "MSA" en la configuración del MIDlet (explicada más adelante), por lo que el usuario puede elegir entre las siguiente políticas de seguridad:

  - unidentified_third_party: Garantiza un elevado nivel de seguridad para aplicaciones cuyo origen y autenticidad son desconocidos. Se consultará

al usuario constantemente, cada vez que la aplicación realice una operación delicada.

- identified_third_party: Destinados a MIDlets cuyos orígenes se determinaron utilizando los certificados de cifrado. Los permisos no se conceden automáticamente, es el usuario quien los concede, pero se le consultará con menor frecuencia que en el dominio unidentified_third_party. Esta política de seguridad será la escogida en este proyecto.
- manufacturer: destinado a MIDlets que tienen certificado del fabricante.
- minimum: Todos los permisos son denegados a cualquier MIDlet.
- maximum: Todos los permisos son aceptados a cualquier MIDlet.

2.2 Crear un proyecto

Para crear un nuevo proyecto, se deberá pulsar el botón "New Project", y a continuación, rellenar una ventana emergente con el nombre del proyecto y de la clase principal del mismo [2].

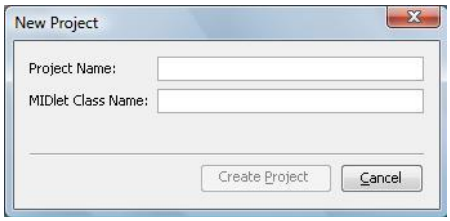


Ilustración 29: Ventana emergente de Creación de Proyecto

El siguiente paso será establecer la configuración del MIDlet. Por cada proyecto creado, WTK ofrece por defecto una configuración que el usuario podrá personalizar de acuerdo a sus necesidades. Algunas de las opciones más interesantes, que el usuario deberá tener en cuenta son:

- Posibilidad de añadir al MIDlet una descripción: el objetivo es poder localizarlo posteriormente con mayor facilidad a la hora de abrir un proyecto. Para ello en la pestaña "Optional", de la ventana de configuración, podremos rellenar el campo "MIDlet-Description".

- Posibilidad de añadir y modificar los MIDlets que componen nuestro MIDlet suite Project: un MIDlet suite Project es un MIDlet que está compuesto por varios MIDlets. Puede que al usuario no le interese almacenar todos los ficheros fuente (.java) de todos los MIDlets del MIDlet suite Project en el mismo directorio (scr). Para ello, en la pestaña "MIDlets", de la ventana de configuración, el usuario podrá editar el campo "Class" de cada MIDlet, para que en lugar de almacenar el nombre de la clase principal del MIDlet (por ejemplo, "Teledocencia"), almacene también el nombre del subdirectorio en el que se almacenarán los ficheros fuente de este MIDlet (por ejemplo, "PFC.Teledocencia").

Al establecer la configuración del MIDlet, se generará la estructura de directorios del proyecto. El usuario siempre podrá volver a acceder a la configuración del MIDlet y modificarla, abriendo el proyecto y pulsando el botón "Settings …".

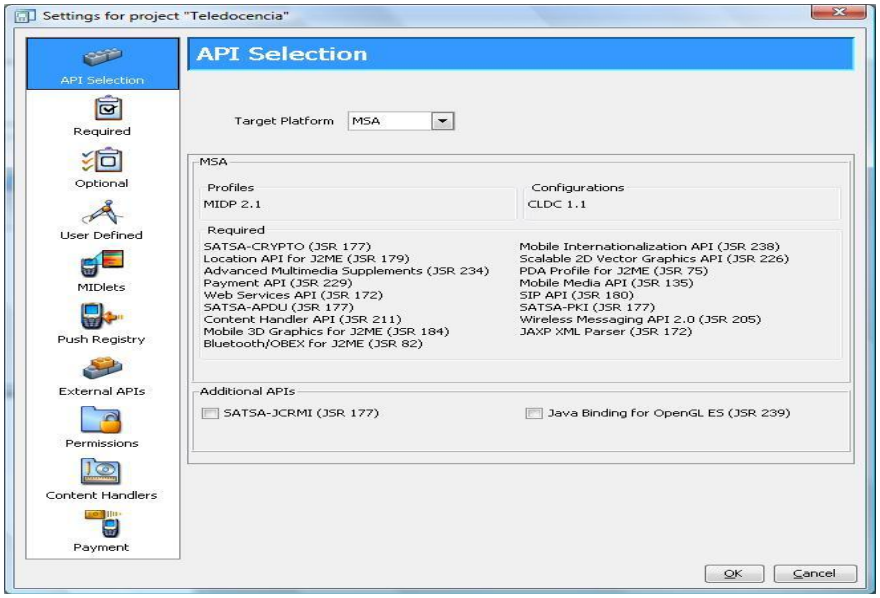


Ilustración 30: Ventana emergente de Configuración del MIDlet

2.3 Abrir un proyecto

Para abrir un proyecto ya existente, el usuario deberá pulsar el botón "Open Project", y seleccionar de una lista, en una ventana emergente, el proyecto que desea abrir. Esta lista mostrará al usuario el nombre de todos los proyectos que ha creado anteriormente (en negrita), así como el nombre de todos los proyectos de demostración (en cursiva); y las descripciones de todos ellos (en caso de haberla añadido durante la configuración del MIDlet). Si el usuario únicamente desea ver en la lista los proyectos que ha creado, deberá desactivar el check box “Show available demos” de esta ventana.

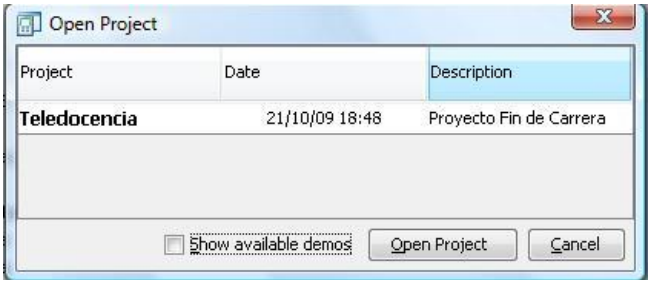


Ilustración 31: Ventana emergente Abrir Proyecto

2.4 Compilación y preverificación de un proyecto

La compilación y preverificación del proyecto se llevan a cabo con el botón "Build". Cuando se compila un MIDlet, se genera un archivo con la extensión “.class” por cada clase de este, que contiene el código intermedio que es capaz de ejecutar la máquina virtual de Java. Estos archivos son almacenados en el directorio “clases”. El siguiente paso es preverificar las clases generadas, para garantizar que no existe ningún tipo de código malintencionado que pueda dañar o crear un comportamiento extraño en nuestro dispositivo o en la máquina virtual **[10]**. Sin esta preverificación no será posible ejecutar el MIDlet.

2.5 Ejecutar un Proyecto

Una vez se ha compilado y preverificado el MIDlet ya puede ser ejecutado, para ello el usuario deberá pulsar el botón "Run". Al hacerlo, aparecerá en una ventana emergente un emulador con forma de teléfono móvil. La pantalla del móvil mostrará un menú con el nombre del MIDlet, y para ejecutarlo el usuario deberá pulsar en el emulador la opción "Launch".

3 Fichero de configuración de aplicación: emulator.properties

```
bluetooth.connected.devices.max: 7
bluetooth.device.authentication: on
bluetooth.device.authorization: on
bluetooth.device.discovery.enable: true
bluetooth.device.discovery.timeout: 10000
bluetooth.device.encryption: on
bluetooth.device.friendlyName: WirelessToolkit
bluetooth.device.next.address: 0000000decaf
bluetooth.enable: true
bluetooth.l2cap.receiveMTU.max: 512
bluetooth.sd.attr.retrievable.max: 10
bluetooth.sd.trans.max: 5
file.extension: jad
http.version: HTTP/1.1
irdaobex.discoveryTimeout: 10000
irdaobex.packetLength: 4096
kvem.device: DefaultColorPhone
kvem.netmon.autoclose: false
kvem.netmon.enable: false
kvem.netmon.filter_file_name: netmon_filter.dat
kvem.netmon.fixed_font_name: Courier New
kvem.netmon.fixed_font_size: 12
kvem.netmon.variable_font_name: Arial
kvem.netmon.variable_font_size: 14
kvem.otherVM: true
lapi.addressInfoRequired: false
lapi.altitudeRequired: false
lapi.costAllowed: true
lapi.horizontalAccuracy: 0
lapi.powerConsumption: None
lapi.responseTime: 0
lapi.speedRequired: false
lapi.verticalAccuracy: 0
microedition.locale: en-US
microedition.sip.outbound_proxy:
microedition.sip.registrar:
mm.control.capture: true
```

```
mm.control.midi: true
mm.control.mixing: true
mm.control.record: true
mm.control.volume: true
mm.format.midi: true
mm.format.video: true
mm.format.wav: true
open.show.demos: false
satsa.sim.present: true
satsa.simulator.type: cref
satsa.slot.0.port: 9025
satsa.slot.1.port: 9026
security.domain: identified_third_party
security.policy: MSA
sip.address:
sip.display.name:
sip.port:
uc.delay: 10080
uc.enable: true
uc.url: http://cds.cmsg.sun.com/update/update.properties?uid=%s&ver=%s
vmspeed.range: 100,1000
wma.server.firstAssignedPhoneNumber: +5550000
wma.smsc.phoneNumber: +1234567890
```