TRABAJO DE FINAL DE CARRERA

TÍTULO DEL TFC: Avisador médico basado en tecnología Bluetooth

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Miguel Sánchez Opazo

DIRECTOR: Juan Hernández Serrano

FECHA: 17 de Julio de 2009

Título: Avisador médico basado en tecnología Bluetooth

Autor: Miguel Sánchez Opazo

Director: Juan Hernández Serrano

Data: 17 de Julio de 2009

Resumen

Hoy en día, se están introduciendo mejoras tecnológicas en todos los ámbitos. Mejoras que ayudan y facilitan el trabajo de los profesionales en la mayoría de sectores. Es por eso, que hemos pensado en introducir una mejora tecnológica en el sector médico, más concretamente en el cuidado de personas con movilidad reducida y/o dificultades comunicativas.

Desde los hospitales se reporta la necesidad de un sistema de alarmas que facilite el trabajo de los auxiliares de enfermería, por ejemplo avisando cuando un paciente se ha efectuado sus necesidades. La solución trivial sería, en este caso, la instalación de sensores de humedad en los pañales que lanzasen una alerta a un dispositivo en posesión del personal médico cuando el pañal estuviese mojado. La misma solución podría aplicarse a multitud de situaciones tan sólo cambiando el tipo de sensor a, por ejemplo, sensores de la presión arterial, las pulsaciones, nivel de ácido úrico, etc.

El objetivo de este TFC, es precisamente desarrollar una aplicación que se ejecute en el dispositivo móvil y que permita que se comunique de manera inalámbrica con el sensor para la recepción de alertas. La aplicación debe, por tanto, ser capaz de registrar el sensor de forma segura, y de permanecer a la espera de posibles alertas. Además, al no disponer de un sensor comercial que reúna las características necesarias, se desarrolla también en este TFC un simulador del sensor médico que permita probar el funcionamiento de la aplicación.

La aplicación que se ejecuta en el dispositivo móvil del personal médico se ha implementado en JavaME haciendo uso de la tecnología bluetooth como medio de comunicación inalámbrico. El simulador del sensor que genera las alertas se ha implementado en un PC con interfaz bluetooth, sistema operativo Linux y scripts en Bash y Perl.

Title: A bluetooth based medical notifier

Author: Miguel Sánchez Opazo

Director: Juan Hernández Serrano

Date: July, 17th 2009

Overview

Today, new technological improvements appear in all areas. Such improvements help and facilitate a big amount of professional tasks. That is why we thought of introducing a technological improvement in the medical sector, specifically in the care of people with reduced mobility and other physical handicaps.

Nowadays, hospitals are reporting the need for an alarm system that facilitates the nurse tasks, for example notifying the nurse when a patient has pissed himself. A trivial solution, in this case, is the installation of humidity sensors that throw an alert to a device owned by the nurse whenever the diaper is wet. Similar solutions could apply to a great variety of situations just by changing the sensor type for another one in order to check the blood pressure, the heart beats, the level of uric acid, etc.

The target of this work is to develop an application that runs on a mobile device and let it communicate wirelessly with the sensor for receiving alerts. The application must therefore be able to register the sensor in a safe mode and be in alert for possible warnings. Moreover, because of the lack of a commercial sensor that meets the required characteristics, a sensor simulator is also developed. This simulator enables correct testing of the application.

Presented application would run on a mobile device owned by medical staff and has been developed in JavaME using bluetooth technology for the wireless link. The sensor simulator that generates the alerts has been implemented on a PC with bluetooth interface, Linux operating system and scripts in Bash and Perl.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

En nuestros días, los dispositivos móviles se han convertido en una herramienta imprescindible para nosotros. Todos los llevamos y utilizamos a diario, ya sea para comunicarnos con otras personas, para escuchar música o para jugar. Por este motivo, el desarrollo de aplicaciones para estos dispositivos es uno de los mercados mas explotados actualmente, y con mas perspectivas de futuro.

Sin embargo, esta tecnología no ha proporcionado solamente una nueva dimensión de ocio, sino que también, hasta ahora, ha facilitado y mejorado la vida diaria de las personas y continuará haciéndolo en el futuro con nuevos usos y aplicaciones. Por ejemplo, en el ámbito de los cuidados médicos, en muchas ocasiones, el médico o el auxiliar de enfermería, tiene dificultades para saber cuando necesita atención un paciente con movilidad reducida y/o dificultades comunicativas. Al no poder comunicarse ni moverse, dificulta mucho el trabajo del personal sanitario. Tomando este hecho como base, vamos a desarrollar una aplicación que ayudará al personal sanitario de un hospital a entender las necesidades de sus pacientes.

Más concretamente, vamos a solucionar un problema presente en todos los centros sanitarios: los pañales de las personas con movilidad reducida. En muchas ocasiones, los pacientes deben llevar pañales. Si a este hecho le añadimos que el enfermo puede tener dificultades comunicativas, resulta un problema para el personal sanitario saber cuando atender a dicho enfermo. Para solucionar este problema, proponemos utilizar sensores en los pañales. Estos sensores, deben ser sensibles a la humedad. En el momento en que detectan humedad, deben ser capaces de lanzar una alerta a un dispositivo receptor de manera inalámbrica.

Por todo lo anterior, el objetivo de este TFC es desarrollar un sistema de comunicación entre un dispositivo móvil y un sensor médico. Por lo tanto, la aplicación debe reunir los siguientes requisitos:

- Debe poder ejecutarse en un dispositivo móvil y recibir las alertas por un medio inalámbrico. Permitiendo que el personal médico conozca el estado del paciente sólo acercándose al entorno de éste.
- Debe consumir pocos recursos. Pensando en uso con dispositivos con una autonomía reducida, alimentados mediante una batería.
- Debe ser una aplicación segura. Solo el personal acreditado debe poder comunicarse con el sensor, por lo que deberá ser segura frente a ataques intencionados o no intencionados de otros dispositivos móviles.

La tecnología escogida para el medio inalámbrico ha sido bluetooh. Infrarrojos no nos sirve, ya que para poder comunicarse, los dos dispositivos deben estar uno al frente de otro y a muy poca distancia, con lo que no facilitaría ni una mínima movilidad al personal sanitario. WLAN 802.11 era otra alternativa interesante, pero se ha optado por la tecnología bluetooth principalmente por dos razones: la primera es que blueetooth se puede encontrar en la mayor parte de los móviles en la actualidad, y la segunda es que el consumo de

batería de blueetooh es mucho menor que el de WLAN, lo que es especialmente crítico si el sensor depende de una batería.

Una vez elegida la tecnología para comunicarnos con el sensor, debemos pensar en qué desarrollar nuestra aplicación. Hoy en día, existen diversos lenguajes de programación que funcionan en dispositivos móviles. Principalmente existen tres tecnologías muy potentes para la programación en móviles: C, Cocoa y Java. La programación en C, se usa para dispositivos con un sistema operativo Symbian. Cocoa, es un lenguaje de programación bastante reciente, utilizado en dispositivos con sistemas operativos Apple, como los actuales iPhone. Por último, la tecnología Java es un lenguaje multiplataforma que permite ejecutar el mismo código en hardwares diferentes y que está presente en la mayoría de dispositivos móviles. Por estos motivos, creemos que Java es el lenguaje ideal para desarrollar nuestra aplicación.

A parte de la aplicación para el dispositivo móvil, debemos probar que nuestra aplicación funcione correctamente. Para poder testear el correcto funcionamiento de la aplicación, debemos disponer de un sensor. Como no disponemos de un sensor comercial y el objetivo de este TFC no es desarrollar uno, vamos a simular las funciones de dicho dispositivo. Para simular el sensor, aprovecharemos las interfícies Bluetooth de Linux. Crearemos un script capaz de emular todas las funciones que deberá desempeñar el sensor. Por lo tanto, el desarrollo dentro de este TFC consta de dos partes diferenciadas: una primera, que es la aplicación Java que se ejecuta en el móvil; y una segunda, que simula el sensor con el que se tendrá que comunicar el móvil.

Para entender mejor el escenario de la práctica, vamos a fijarnos en la Fig. 0.1. Dispondremos de un dispositivo móvil, capaz de ejecutar una aplicación desarrollada en Java y un PC con interfície Bluetooth y sistema operativo Linux. El PC emulará el comportamiento del sensor comercial y el móvil deberá comunicarse con él y esperar las alertas producidas por el PC.

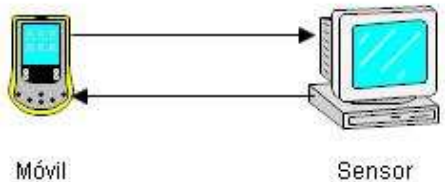


Fig. 0.1 Escenario

Este TFC se estructura de la siguiente forma. En el capítulo 1, realizamos un estudio del protocolo Bluetooth, centrándonos en su origen, sus principales características, su forma de comunicarse y sus métodos de seguridad. En el capítulo 2, presentamos las herramientas de trabajo: Java (JME) como lenguaje de programación para la aplicación móvil, y Bash y Perl como lenguajes de Scripting para emular el sensor. A continuación, en el capítulo 3, presentamos el desarrollo de la aplicación móvil y de la emulación del sensor. En el capítulo 4 detallamos una demo de uso de la aplicación desarrollada. Por último, en el capítulo 5, extraemos conclusiones y presentamos las posibles líneas futuras.

CAPITULO 1. BLUETOOTH

Bluetooth es un sistema de comunicaciones inalámbricas de corto alcance. Es un estándar global de comunicación inalámbrica establecido por la IEEE bajo la especificación 802.15.1, donde se pueden realizar conexiones de Red Inalámbricas teniendo la posibilidad de transmitir voz, datos, imagen, multimedia, etc. entre diferentes dispositivos utilizando la tecnología de radio frecuencia.

El tipo de redes en las que opera Bluetooth son las llamadas WPAN (Wireless Personal Area Network) o redes de área personal inalámbricas.

Actualmente, Bluetooth llega a velocidades de hasta 3 Mbps operando en la frecuencia destinada a fines instrumentales, científicos y médicos (*instrumental, scientific, and medical* "ISM", instrumental, científico y médico) en su respectiva banda de 2.4 GHz.

1.1 Historia del Bluetooth

El nombre de Bluetooth fue tomado de un rey Danés del siglo X llamado Harald Blatand (en lengua danesa significa "*de tez oscura*") cuya deformación lingüística haría que se convirtiera en Harold Bluetooth (en inglés *diente azul*). Este rey fue famoso por sus habilidades comunicativas ya que logro la unificación de las tribus noruegas, suecas y danesas. Estas cualidades comunicativas, sirvieron como ejemplo para la unificación en la comunicación que suponía la invención de la tecnología Bluetooth: Bluetooth debía conseguir unificar todos los medios de transmisión inalámbrica de corto alcance.

El logotipo de Bluetooth (Fig. 1.1) procede de la unión de los alfabetos rúnicos de sus iníciales (Harald) y (Blatand) y su color azul hace referencia a su nombre inglés Bluetooth (blue = azul).

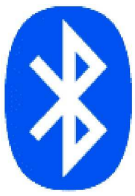


Fig. 1.1 Logotipo de Bluetooth

En 1994, la empresa sueca Ericsson empezó a investigar una interfaz vía radio de bajo costo y consumo para eliminar los cables de interconexionado entre teléfonos móviles y otros accesorios. Con este fin, se creó el proyecto *MC link*.
A comienzos de 1997, conforme el proyecto tomaba forma, Ericsson consiguió despertar el interés de otros fabricantes. A principios de 1998 se realizó un ISG (*Special Interest Group*, *Grupo de Interés Especial*) promovido por grandes empresas, entre ellas Ericsson, IBM, Intel, Nokia y Toshiba y se formaliza la

especificación del Bluetooth 1.0. En 2002, se crea el estándar IEEE 802.15.1. Finalmente, en julio de 2007 se define la versión actual de esta tecnología, la 2.1 que es compatible a las anteriores versiones, ofrece velocidades hasta los 3 Mbps, y mejoran el ahorro de energía y la seguridad de las comunicaciones.

1.2 Características

La tecnología Bluetooth, permite eliminar los cables entre dispositivos móviles, periféricos y PC. Transmite por Radio Frecuencia con un alcance de 10 metros (100 metros si aumentamos la potencia de transmisión y la sensibilidad de recepción). Es una interfície de bajo coste y compacta, lo que es ideal para utilizarla en dispositivos móviles. Es una tecnología segura, ya que permite autorización, autenticación de dispositivos y datos cifrados. Su interfície es de baja potencia (1mW) y bajo consumo, lo que lo hace ideal para dispositivos que se alimentan mediante una batería. Gracias a su bajo consumo, podremos ahorrar energía y conseguir que la batería de nuestros dispositivos dure más tiempo. Soporta las velocidades de transmisión de los dispositivos móviles a 3 Mbps (datos y voz).

1.3 Los perfiles Bluetooth

Todo dispositivo que se quiera comunicar con otro mediante la tecnología inalámbrica Bluetooth, debe saber interpretar los perfiles Bluetooth. Estos perfiles, describen las distintas aplicaciones posibles. Cada dispositivo, puede tener distintas aplicaciones que utilicen Bluetooth, por lo tanto, deberemos especificar qué servicio es el que nos interesa.

En este momento es cuando entran en funcionamiento los perfiles Bluetooth, que indican los procedimientos que deberán seguir los dispositivos para poder comunicarse entre sí.

Existen múltiples perfiles Bluetooth, especificando dichos procedimientos para los diferentes servicios que puede ofrecer un dispositivo mediante la tecnología Bluetooth. Cada perfil incluye, como mínimo, información sobre las siguientes cuestiones:

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

A continuación, en la Tabla 1.1, podemos ver algunos de los perfiles que ofrece esta Tecnología.

Tabla 1.1 Perfiles Bluetooth

Perfiles	
Perfil básico de impresión (BPP)	El perfil BPP permite enviar mensajes de texto, de correo electrónico, tarjetas de visita electrónicas e imágenes, entre otras cosas, a las impresoras disponibles dependiendo de las tareas de impresión. [11]
Perfil de transferencia de archivos (FTP)	El perfil FTP establece los procedimientos de exploración de carpetas y archivos de un servidor a través de un dispositivo cliente. [12]
Perfil manos libres (HFP)	El perfil HFP describe cómo un dispositivo que actúa como puerta de enlace puede utilizarse para realizar y recibir llamadas a través de un dispositivo manos libres. [13]
Perfil de aplicación de descubrimiento de servicio (SDAP)	El perfil SDAP detalla cómo una aplicación debe utilizar el perfil SDP para identificar los servicios de un dispositivo remoto. [14]
Perfil de servicio de puerto (SPP)	El perfil SPP describe cómo configurar puertos de serie y conectar dos dispositivos con tecnología *Bluetooth*. [15]

1.5 Los protocolos de comunicación

Una vez que sabemos el servicio al que nos queremos conectar, y sabemos las especificaciones que requiere dicho servicio mediante su perfil, debemos conocer el protocolo de comunicación.

Tabla 1.2 Protocolos Bluetooth

Protocolos	
Protocolo de intercambio de Objetos (OBEX)	OBEX es un protocolo de transferencia que define los objetos de datos y el protocolo de comunicaciones que deben utilizar dos dispositivos para intercambiar objetos. [16]
Protocolo de control de telefonía (TCP)	Este protocolo establece la señalización para el establecimiento de llamadas de voz y datos en dispositivos con tecnología *Bluetooth*. [17]
RFCOMM con TS 07.10	El protocolo RFCOMM emula los parámetros de un cable de serie y el estado de un puerto RS-232 para transmitir datos en serie. [18]

Existen distintos protocolos de comunicación Bluetooth. En la Tabla 1.2 podemos ver algunos protocolos y una breve descripción.

1.6 Estados del Bluetooth

Cuando ya conocemos el servicio al que nos queremos conectar y el protocolo que utilizaremos para la comunicación, deberemos buscar dispositivos con los que nos podamos comunicar.

Bluetooth es una tecnología que, como está adaptada a dispositivos de bajo consumo, implementa distintos procedimientos para el ahorro de energía. Por lo tanto, su estado normal será el de reposo o *standby*.

La conexión con un dispositivo, se hace mediante un mensaje *page*. Si la dirección es desconocida, antes del mensaje *page* se necesitara un mensaje *inquiry*.

El procedimiento de *inquiry* es asimétrico, es decir, por una parte el dispositivo Bluetooth que realiza la búsqueda envía activamente solicitudes de detección mientras que los dispositivos que están a la espera de ser detectados escuchan estos mensajes de búsqueda y los contestarán. Nuestros dispositivos, los podemos configurar para que no sean visibles. Esto hace que nuestro móvil no responda a mensajes *inquiry*, por lo que no seremos descubiertos por los dispositivos a nuestro alcance.

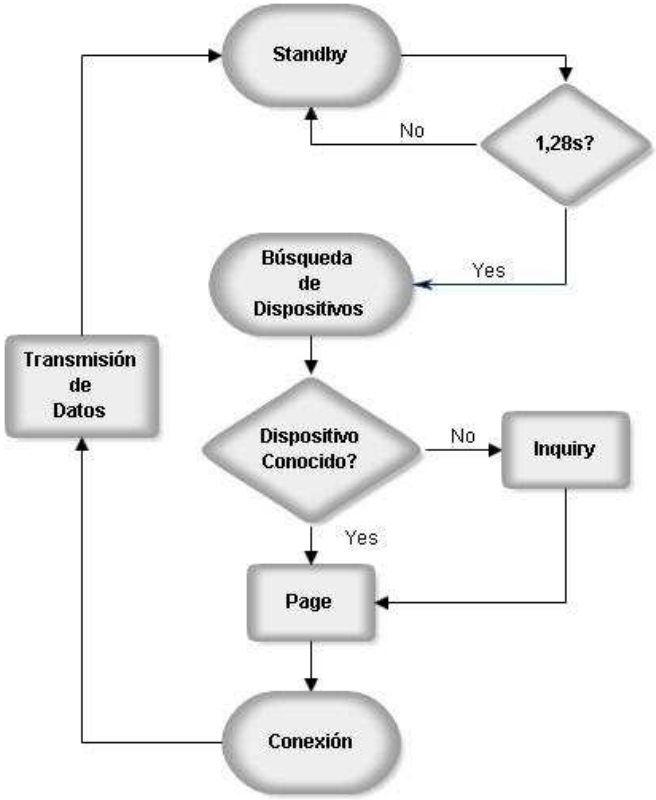


Fig. 1.2 Diagrama de Estados de conexión Bluetooth

El procedimiento de *page* también es asimétrico. Lo que significa que el master de nuestra red, que ya conoce al esclavo con el que se quiere conectar, enviará un mensaje *page* con el fin de crear una conexión entre los dos.

Antes de que se produzca ninguna conexión, se dice que todos los dispositivos están en modo *standby*.

Un dispositivo en modo *standby* se despierta cada 1.28 segundos para escuchar posibles mensajes *inquiry/page*. Cada vez que un dispositivo se despierta, escucha una de las 32 frecuencias de salto definidas. Un mensaje de

tipo *page*, será enviado en 32 frecuencias diferentes. Primero el mensaje es enviado en las primeras 16 frecuencias (128 veces), y si no se recibe respuesta, el maestro mandará el mensaje *page* en las 16 frecuencias restantes (128 veces). El tiempo máximo de intento de conexión es de 2.56 segundos.

Para dejar mas claro este proceso, vamos a mostrar un diagrama que describe todo el proceso de comunicación Bluetooth. Desde el modo reposo (*standby*) hasta la transmisión de datos. Fig. 1.2.

1.7 Seguridad en Bluetooth

Los medios inalámbricos, son susceptibles a ataques intencionados y no intencionados. Cualquier persona que esté a nuestro alcance, puede intentar establecer una comunicación con nosotros.

Para evitar robos de información, se han creado unos algoritmos de autenticación y encriptación.

Para conocer más a fondo los elementos de seguridad que ofrece Bluetooth, vamos a describir los pasos que siguen los dispositivos para poder comunicarse.

1.7.1 Emparejamiento/Pairing

El primer paso que deberá seguir un dispositivo Bluetooth para conectarse con otro es saber si ya lo conoce. Si no es así, deberán hacer un paso previo de emparejamiento, en el que se autenticarán y crearán una clave común que utilizarán en futuras comunicaciones.

1.7.1.1 Generación de la clave de inicialización

El primer paso del proceso de autenticación/encriptación es la generación de una clave de inicialización. Esta clave de inicialización se obtiene a partir de la dirección destino Bluetooth (BD_ADDR), un código PIN, y un número aleatorio IN_RAND. La salida obtenida es una clave llamada K_{init} de 128 bits y se utilizará para el intercambio de dos números aleatorios que servirán para generar la clave definitiva. Como se aprecia en la Fig. 1.3, el número aleatorio IN_RAND se pasa en claro al receptor. Esto no representa un fallo de seguridad ya que si un atacante intercepta este número, pero no conoce el PIN, no podrá generar K_{init}.

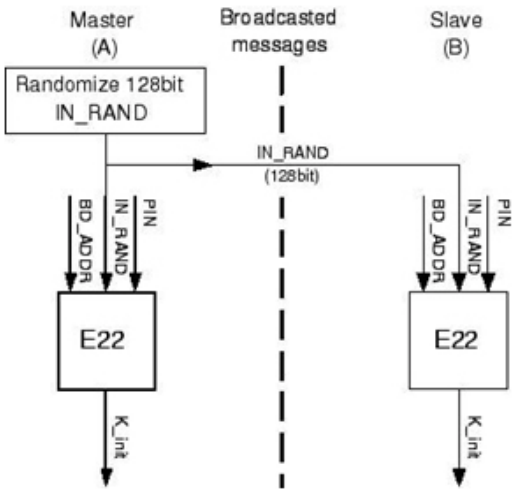


Fig. 1.3 Generación de la clave de inicialización

1.7.1.2 Generación de la clave de enlace

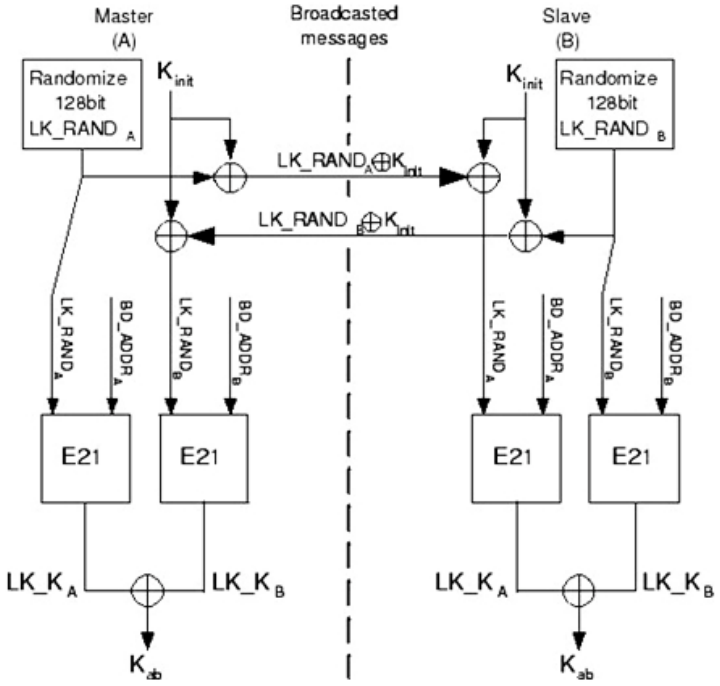


Fig. 1.4 Generación de la clave de enlace

Se genera la clave de enlace K_{ab} usando el algoritmo E21. Los dispositivos usan la clave de inicialización K_{init} para intercambiar dos nuevos números aleatorios, conocidos como LK_RAND A y LK_RAND B. Cada dispositivo genera un número aleatorio y se lo envía al otro dispositivo pasándolo previamente por una puerta XOR bit a bit con K_{init}. Dado que ambos dispositivos conocen K_{init}, cada dispositivo conoce ambas LK_RAND. A partir de la dirección BD_ADDR y LK_RAND, se genera la clave de enlace K_{ab}. Fig. 1.4

1.7.2 Autenticación

Ahora que los dispositivos están emparejados y conocen K_{ab}, ya pueden autenticarse cada vez que quieran realizar una comunicación. En este caso, se realiza una autenticación de dispositivo, no de usuario. Esto quiere decir, que garantizamos que la comunicación la realizamos con el dispositivo que conocemos, pero el usuario puede ser otra persona.

Los pasos a seguir en la autenticación son:

1) El dispositivo reclamante envía su dirección BD_ADDR al dispositivo verificador.

2) El verificador devuelve un desafío aleatorio de 128 bits al demandante.

3) El reclamante usa el algoritmo E1 para generar la respuesta de autenticación (SRES) de 32 bits, usando como parámetros de entrada la dirección BD_ADDR del reclamante, la clave de enlace Kab almacenada y el desafío. El verificador realiza la misma operación en paralelo.

4) El reclamante devuelve la respuesta SRES al verificador.

5) El verificador comprueba la respuesta SRES recibida por el reclamante con la respuesta SRES calculada por él.

6) Si los valores de SRES coinciden, el verificador y el reclamante intercambian los papeles y repiten el proceso entero. Fig. 1.5.

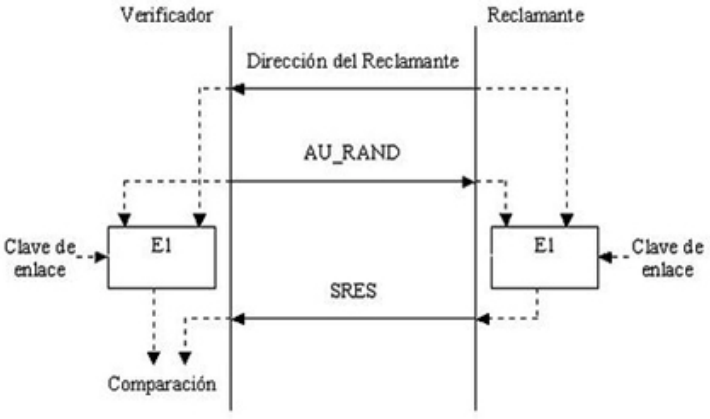


Fig. 1.5 Autenticación del dispositivo

En este proceso, se crea además un número de 96 bits llamado ACO (Authenticated Ciphering Offset) en ambos dispositivos, que será usada durante la creación de la Clave de Cifrado.

1.7.3 Autorización

La autorización es el procedimiento que determina los derechos que tiene un dispositivo Bluetooth para acceder a los servicios que ofrece un sistema.

Por este motivo, este proceso se realiza mediante niveles de confianza. Estos niveles de confianza determinarán si un dispositivo tiene permiso para acceder a todos los servicios, a alguno o a ninguno.

Los dispositivos que estén en un nivel de confianza restringida o no confiable, deberán de ser aceptados por el receptor para poder realizar la comunicación.

1.7.4 Cifrado de los datos

El cifrado de los datos en Bluetooth es opcional, y se deberán poner de acuerdo las dos partes mediante una negociación. En el caso que no se pueda llegar a un acuerdo, no se realizará la comunicación. La Clave de Cifrado es generada aplicando al algoritmo E3 la Clave de Linkado, un número aleatorio de 128 bits y un Ciphering Offset (COF) basado en el valor de ACO del proceso de autenticación. Fig. 1.6

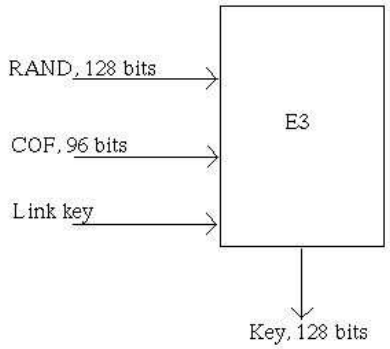


Fig. 1.6 Cifrado de los datos

CAPITULO 2. HERRAMIENTAS DE TRABAJO

En este capítulo, comentaremos las herramientas de trabajo utilizadas a lo largo de este TFC. Básicamente son tres: Java (JME) como lenguaje de programación y bash y perl como lenguaje de script para implementar el sensor.

2.1 JAVA (JME)

Java es un lenguaje de programación orientado a objetos muy extendido mundialmente. Gracias a sus propiedades multiplataforma, lo encontramos presentes en múltiples sistemas operativos y dispositivos diferentes.

El lenguaje Java se utiliza en diversos ámbitos. Tanto en programas de gestión, como en la Web, como en dispositivos móviles. Es en este último en el que haremos hincapié, y estudiaremos a fondo todas las posibilidades que posee el lenguaje Java para móviles.

2.1.1 Historia del lenguaje JAVA

Java fue diseñado en 1990 por James Gosling, de Sun Microsystems, como software para dispositivos electrónicos de consumo como calculadoras, microondas y la televisión interactiva.

En un principio, Java se denominó Oak. Este lenguaje, fue desarrollado por el equipo dirigido por Gosling llamado “Green Team”. Su objetivo era conseguir un lenguaje que se ejecutara independientemente del dispositivo, ya que creían ciegamente en la convergencia entre dispositivos electrónicos y los ordenadores. El proyecto culmina con la realización de un controlador de dispositivos de mano para uso doméstico destinado al sector de la televisión digital por cable. Por desgracia para el equipo, la idea resultó demasiado avanzada para el momento y no tuvo aceptación. No volvieron a tener protagonismo hasta mediados de los 90 gracias al boom de Internet y de la Web.

Patrick Naughton, miembro del “Green Team”, procedió a la construcción del lenguaje de programación Java que se accionaba con un browser prototipo. El 29 de septiembre de 1994 se termina el desarrollo del prototipo de *HotJava*.

Una de las características de *HotJava* fue su soporte para los "applets", que son las partes de Java que pueden ser ejecutadas localmente y así lograr soluciones Web dinámicas.

El 23 de mayo de 1995, en la conferencia SunWorld ‘95, John Gage, de Sun Microsystems, y Marc Andreessen, cofundador y vicepresidente de Netscape,

anunciaban la versión alpha de Java, que en ese momento solo corría en Solaris, y el hecho de que Java iba a ser incorporado en Netscape Navigator, el navegador mas utilizado de Internet.

Con la segunda alpha de Java en Julio, se añade el soporte para Windows NT y en la tercera, en Agosto, para Windows 95. En enero de 1995 aparece la versión 1.0 del JDK.

Con el fin de extender la tecnología Java rápidamente a nivel mundial, Java Soft otorgó permisos a otras compañías para acceder al código fuente y al mismo tiempo mejorar sus navegadores. También les permitía crear herramientas de desarrollo para programación Java y desarrollar máquinas virtuales Java (JVM).

Es curioso pensar que este lenguaje, diseñado con un fin tan específico, acabara convirtiéndose en uno de los lenguajes mas utilizados y que se pudiera ejecutar en tanta variedad de dispositivos.

2.1.2 Java en la actualidad. La versión 2

Sun, con el fin de llegar a todos los usuarios, decidió crear Java 2. Java 2, son una serie de versiones de Java que se adaptan a las necesidades del usuario. Por este motivo, se crean tres versiones que satisfacen las necesidades tanto de desarrolladores de aplicaciones independientes de la plataforma (J2SE), como de desarrolladores Web y entorno empresarial (J2EE), como las de los desarrolladores de aplicaciones para dispositivos móviles (J2ME).

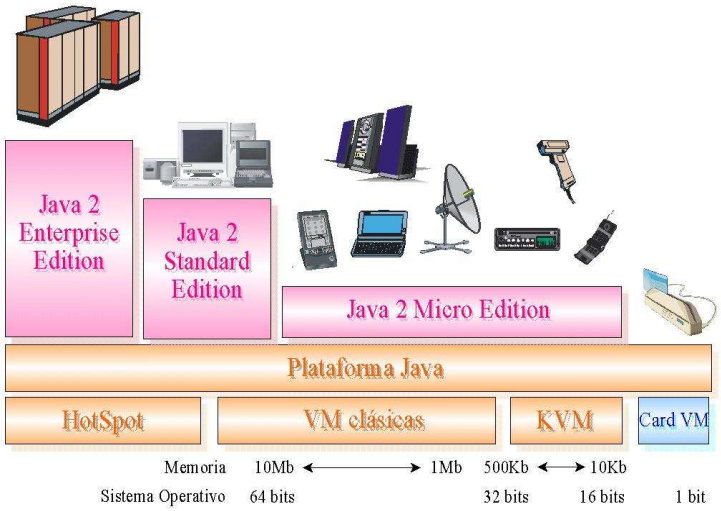


Fig. 2.1 Comparativa de las versiones JAVA

En la Fig. 2.1, se representan las distintas versiones de Java ordenadas de manera descendente según la memoria necesaria del sistema operativo. Apreciamos que J2EE, usada en entornos empresariales en redes extensas, necesita grandes recursos, mientras que J2ME (actual JME) no necesita tanta memoria, lo que lo hace ideal para dispositivos móviles y dependientes de baterías.

2.1.3 Java Micro Edition (JME)

Con la introducción de los dispositivos móviles en nuestras vidas, se han tenido que adaptar lenguajes óptimos a estas tecnologías.

Los dispositivos móviles, ofrecen potencias de cálculo bajas e interfícies de usuario pobres. Esto limita en gran parte las posibilidades del programador, ya que tiene unos límites bastante significativos.

2.1.3.1 Configuraciones JME

Las configuraciones son el conjunto mínimo de APIs (o librerías) que permiten desarrollar aplicaciones para dispositivos móviles. Estas APIs describen las características básicas, comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación Java.
- Características soportadas por la Máquina Virtual Java.
- Bibliotecas básicas de Java y APIs soportadas.

Existen dos configuraciones en J2ME, CLDC orientada a dispositivos con limitaciones computacionales y de memoria y CDC, orientada a dispositivos con no tantas limitaciones.

La configuración CDC, está orientada a dispositivos con cierta capacidad computacional, y que disponen de una fuente de alimentación continua (no baterías). Estos aparatos son, normalmente, decodificadores de televisión digital, sistemas de navegación de automóviles, etc. CDC utiliza una máquina virtual muy parecida a las versiones enterprise y standard de Java2 pero con algunas limitaciones gráficas y de memoria. Mas concretamente, los dispositivos deben disponer de un procesador de 32 bits, un mínimo de 2 Mb de memoria (tanto ROM como RAM), poseer una funcionalidad completa de la máquina virtual de Java2 y conectividad con algún tipo de red. A continuación, en la Tabla 2.1, mostramos las librerías que tiene disponibles esta configuración.

Tabla 2.1 Librerías CDC

Nombre de Paquete CDC	Descripción
java.io	Clases e interfaces estándar de E/S.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfaces de reflection.
java.math	Paquete de matemáticas.
java.net	Clases e interfaces de red.
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de seguridad.
java.text	Paquete de texto.
java.util	Clases de utilidades estándar.
java.util.jar	Clases y utilidades para archivos JAR.
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos.
javax.microedition.io	Clases e interfaces para conexión genérica CDC.

La configuración CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de estos dispositivos son las PDA, móviles, etc. En este caso, la máquina virtual llamada KVM, ofrece muchas restricciones. La configuración mínima de los dispositivos debe ser: Disponer entre 160 Kb y 512 Kb de memoria total disponible. Como mínimo se debe disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución; Procesador de 16 o 32 bits con al menos 25 Mhz de velocidad; Ofrecer bajo consumo, debido a que éstos dispositivos trabajan con suministro de energía limitado, normalmente baterías y por último, tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

Como podemos apreciar, las limitaciones son mucho mayores que en la anterior configuración. Esta configuración, nos aporta algunas de las librerías comunes de Java, con algunas restricciones. Estas librerías nos ofrecen E/S, acceso a redes y seguridad. En la Tabla 2.2, detallamos las librerías que disponemos en la configuración CLDC.

Tabla 2.2 Librerías de la configuración CLDC

Nombre de paquete CLDC	Descripción
java.io	Clases y paquetes estándar de E/S. Subconjunto de J2SE.
java.lang	Clases e interfaces de la Máquina Virtual. Subconj. de J2SE.
java.util	Clases, interfaces y utilidades estándar. Subconj. de J2SE.
javax.microedition.io	Clases e interfaces de conexión genérica CLDC

2.1.3.1 Perfiles JME

Los perfiles, son los encargados del mantenimiento del ciclo de vida de la aplicación, interfaces de usuario y el manejo de los eventos. Más concretamente, el perfil es un grupo más específico de APIs, desde el punto de vista del dispositivo. Por lo tanto, si la configuración define una familia de dispositivos, los perfiles identifican dispositivos concretos dentro de esa familia. A continuación, mostramos una gráfica que muestra la posición jerárquica de los perfiles y las configuraciones. Fig. 2.2.

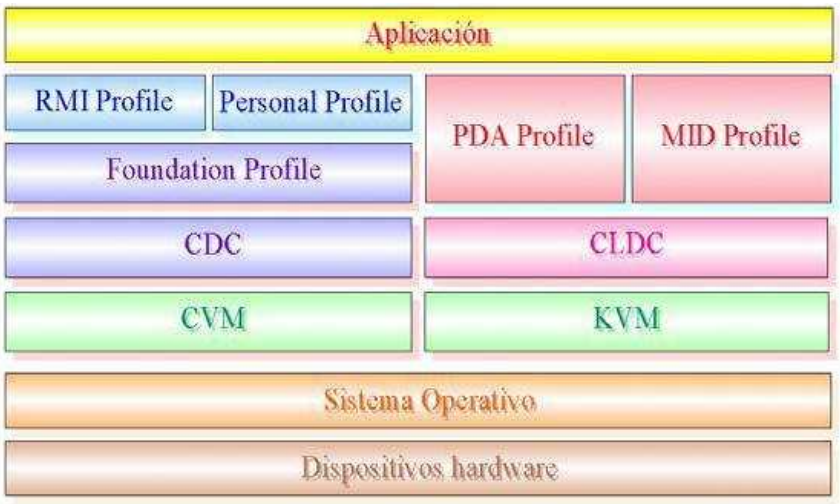


Fig. 2.2 Perfiles JME

Como nuestro entorno de desarrollo se centra en dispositivos de bajo rendimiento y entorno gráfico reducido (CLDC) nos vamos a fijar en los perfiles PDAP y MIDP.

Los dos perfiles son muy parecidos, solo que el perfil de PDA aporta mejoras en la capacidad de proceso y mejoras en la interfície ya que disponen de pantalla mas grande y táctil. El perfil MIDP, está preparado para dispositivos con las siguientes características:

- Reducida capacidad computacional y de memoria.
- Conectividad limitada (en torno a 9600 bps).
- Capacidad gráfica muy reducida (mínimo un display de 96x54 pixels monocromo).
- Entrada de datos alfanumérica reducida.
- 128 Kb de memoria no volátil para componentes MIDP.
- 8 Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

A continuación, en la Tabla 2.3, mostraremos las librerías específicas del perfil MIDP. Como podemos apreciar, son muy escasas y no nos dan demasiada libertad a la hora de programar.

Tabla 2.3 Perfiles JME

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

2.2 Bash

Bash es una shell, o intérprete de órdenes de Unix, escrito para el proyecto GNU. Su nombre es un acrónimo de bourne-again shell (renacimiento sobre el Bourne shell “sh”), que fue uno de los primeros intérpretes importantes de Unix.

Hacia 1978 el intérprete Bourne era el intérprete distribuido con el Unix Versión 7. Stephen Bourne, por entonces investigador de los Laboratorios Bell, escribió el intérprete Bourne original. Brian Fox escribió el intérprete bash en 1987. En 1990, Chet Ramey se convirtió en su principal desarrollador. Bash es el intérprete predeterminado en la mayoría de sistemas GNU/Linux, además de Mac OS X Tiger, y puede ejecutarse en la mayoría de los sistemas operativos tipo Unix. También podemos ejecutar bash en sistemas operativos Windows con el emulador Cygwin.

Mediante este intérprete de comandos, podemos “lanzar” las aplicaciones Bluetooth que dispone el sistema operativo Linux y hacer pequeñas “operaciones” con el resultado.

El lenguaje es muy sencillo e intuitivo. Nos permite lanzar aplicaciones, hacer operaciones aritméticas sencillas, y hacer operaciones lógicas típicas en la programación, como bucles, ifs, etc.

Los Scripts en Bash deben ser iniciados con `#!/bin/bash`. A partir de aquí, todo lo que escribamos es equivalente a lo que ejecutaríamos si lo escribiéramos directamente en la shell “bash”.

A continuación mostramos parte de la sintaxis que utilizaremos.

```
# Esto es un ejemplo de if

if [ $VAR -eq 0 ]; then
   echo "La variable VAR es igual a 0"
fi

# Esto es un ejemplo de bucle

while [ -z $VAR ]; do
```

```
echo “Esto se ira escribiendo mientras no exista VAR”
done
```

2.3 Perl

Perl significa Practical Extraction and Report Language, algo así como lenguaje práctico de extracción y de informes. Es un lenguaje de programación basado en el lenguaje C, en el lenguaje de Shell, AWK, y muchos otros lenguajes de programación.
Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK. Sus características principales son la gran capacidad en el procesado de texto y que no tiene ninguna de las limitaciones de los otros lenguajes de Script.

Este lenguaje, nos será muy útil para parsear grandes cadenas de caracteres, como las salidas de los programas de shell para controlar Bluetooth. Como posee muchas similitudes con el lenguaje de Shell, veremos que es muy parecido al lenguaje de Bash, anteriormente descrito. La forma de declarar variables y los tipos primarios son similares. Además la interacción entre los dos es sencilla, ya que al ser lenguajes de Shell, podemos hacer llamadas desde bash a perl y viceversa.

Al igual que el lenguaje anterior, al principio del script, debemos llamar al compilador del sistema operativo. En este caso, llamamos así al compilador perl en Linux. `#!/usr/bin/perl`

A continuación vamos a ver un ejemplo de código perl en el que se solicita al usuario que escriba un valor que se captura por el teclado, y a continuación imprime el valor de su mitad por pantalla.

```
#!/usr/bin/perl

print "Escribe un número? "; # Printa por pantalla
$valor1 = <STDIN>; # Recoge un valor por teclado
$valor2 = $valor1 * 0.5; # Calcula la mitad
print "Valor mitad = $valor2\n"; # Printa el cálculo
```

CAPÍTULO 3. DESARROLLO DEL PROGRAMA

Una vez presentadas las herramientas, vamos centrarnos en el desarrollo del trabajo. Como ya hemos introducido, el trabajo consta de dos bloques diferenciados.

La primera parte, la forma la aplicación Java (JME) para el dispositivo móvil. Esta aplicación, debe poder comunicarse mediante la tecnología Bluetooth con un sensor. El programa tendrá que ser capaz de registrar y recibir alertas provenientes de dicho sensor. A parte, debe tener una interfaz agradable al usuario y que le permita configurar preferencias varias, como el tono de las alertas, el volumen, si desea vibración, etc.

La segunda parte la forma el sensor, que simularemos mediante las herramientas Bluetooth que nos ofrecen los sistemas operativos Linux. Más concretamente usaremos Kubuntu, que está basado en Debian. Este sensor, deberá ser capaz de guardar la dirección de un dispositivo que se intente comunicar con él, y además deberá lanzar alertas al dispositivo previamente registrado.

3.1 Aplicación Bluetooth

Esta es la primera parte del montaje. Nuestro objetivo es crear una aplicación capaz de comunicarse por Bluetooth con otro dispositivo y además sea amigable y sencilla de usar para el usuario. Aquí entran dos temas. Deberemos estudiar como funciona la API que tiene Java para Bluetooth y además, examinaremos a fondo las posibilidades que nos ofrece JME. Fig. 3.1.

Fig. 3.1 Logotipo JME

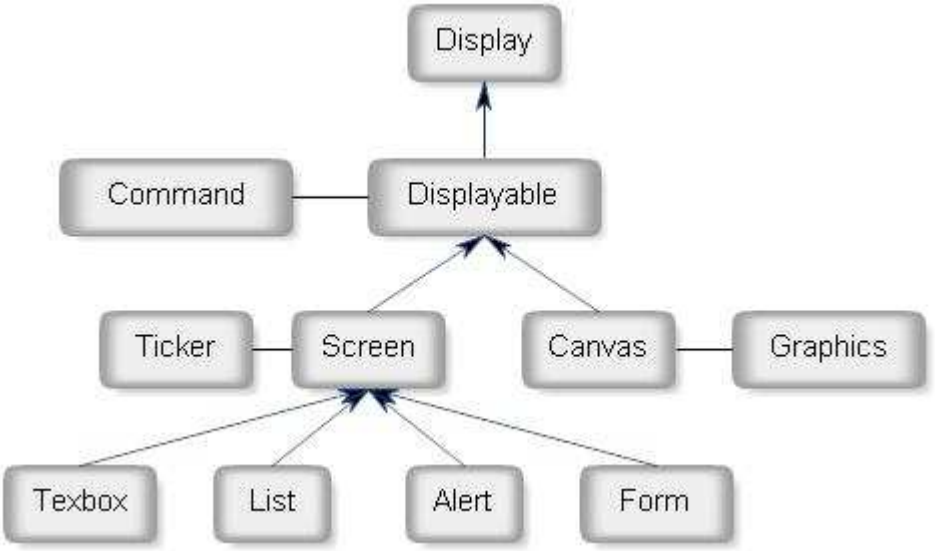
3.1.1 JME. Entorno gráfico

Uno de los requerimientos de este proyecto, es que la aplicación sea amigable y fácil de usar. Para ello, exprimiremos las posibilidades que nos ofrece Java para móviles, estudiando sus clases más importantes y posteriormente implementando nuestra aplicación.

Para empezar, debemos conocer las clases de la API de JME para entorno gráfico. No es muy extensa, ya que las posibilidades que nos brindan los dispositivos móviles son muy limitadas.

Para nuestra aplicación, escogeremos un perfil MIDP, descrito anteriormente. Este perfil es el encargado de definir las peculiaridades de un tipo de dispositivos, en nuestro caso, teléfonos móviles. Este perfil se ocupa de definir aspectos tales como interfaces de usuario, sonidos, almacenamiento en bases de datos, etc.

Nos vamos a centrar en el paquete *javax.microedition.lcdui* (Interfaz de usuario con pantalla LCD), que es donde se encuentran los elementos gráficos que vamos a utilizar en nuestras aplicaciones.



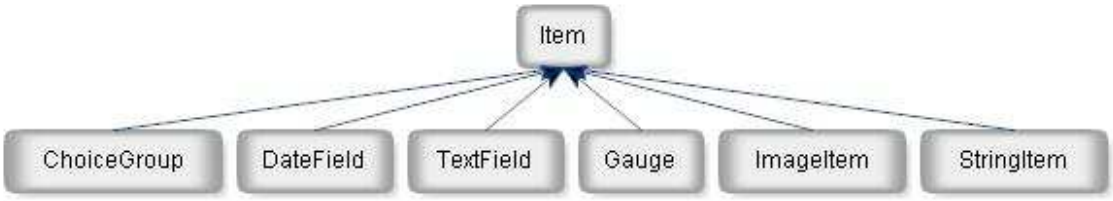
Esquema 3.1 Organización de las clases de alto y bajo nivel de javax.microedition.lcdui

Antes de entrar en profundidad en las clases que nos proporciona el perfil MIDP, debemos hacer una clasificación inicial. Esta clasificación nos diferencia entre las interfaces de usuario de alto nivel y las de bajo nivel. Las de alto nivel, usan componentes tales como botones, cajas de texto, formularios, etc. Estos elementos son implementados por cada dispositivo y la finalidad de usar estas APIs es su portabilidad. Son muy sencillas y frecuentemente son usadas para construir aplicaciones de negocios, en las que no es imprescindible tener control total sobre la apariencia de la aplicación y no es necesario que el usuario tenga el control total sobre ella. Por otro lado, las interfaces de bajo nivel, nos ofrecen un control total sobre la apariencia y el control de la aplicación. Nos permite definir los elementos que se van a mostrar y controlar los elementos de bajo nivel, como el rastreo de pulsaciones de teclas. Estas interfaces, son muy utilizadas en la creación de juegos para móviles. El principal inconveniente, a diferencia de la anterior descrita, es que la

portabilidad es más limitada. Corremos el riesgo que un dispositivo que soporte MIDP, no soporte algún componente de bajo nivel de nuestra aplicación.

En nuestro caso, al no ser un juego ni requerir un control completo de las acciones de usuario, utilizaremos mayormente las interfaces de alto nivel. Esto no quita que para hacer más vistosa la aplicación y añadir algún gráfico, usemos alguna clase de bajo nivel, como canvas.

A continuación mostraremos la organización de las clases de alto y bajo nivel que incluye el paquete *javax.microedition.lcdui*. Esquema 3.1 y Esquema 3.2.



Esquema 3.2 Jerarquía de las clases derivadas de Display e Item

3.1.1.1 La clase Display y Displayable

La clase *Display* es el manejador de la pantalla y de los dispositivos de entrada. Todo Midlet debe tener al menos un *Display*. Esta clase, nos da información sobre las características de la pantalla del dispositivo, además de mostrar los objetos que componen nuestras interfaces. Con esta información, podríamos hacer varias interfaces de usuario, dependiendo de las características del dispositivo. Si es un móvil muy sencillo en blanco y negro, no le podemos poner imágenes en color. Si antes de mostrarlas, averiguamos si las soporta o no, podremos elegir una interfaz u otra dependiendo del dispositivo.

La clase *displayable* gestiona las pantallas de nuestra aplicación. Esto nos permitirá navegar por diferentes pantallas jugando con los métodos `getCurrent()` y `setCurrent()`, con los que iremos mostrando o recuperando pantallas. A parte de estos métodos, hay otros muy útiles en la API.

3.1.1.2 Las clases Command y CommandListener

Un objeto de la clase *Command* contiene información sobre un evento. Es decir, todos los eventos que se generen en la aplicación, como pulsar un botón o hacer cualquier acción, los generará la clase *Command*. Si hacemos la analogía con un interruptor, un *Command* se asemejaría al interruptor en sí.

Pero al pulsar el interruptor, esperamos una reacción. Esta reacción es que se encienda la luz. Para recoger esta acción, deberemos implementar la interfaz *CommandListener*. En la implementación de esta interfaz, deberemos indicar cual será la acción que se desarrollará al generar un evento con un *Command*.

3.1.1.3 La clase Screen. La superclase de alto nivel

La clase *Screen* agrupa todas las clases que conforman la interfaz de usuario de alto nivel. Esta superclase, agrupa a 4 clases que utilizaremos en el proyecto:

- La clase *Alert*: Representa una pantalla de aviso. Es útil para mostrar al usuario situaciones especiales, como las de error o información. Las alertas pueden ser:
    - Modales: La pantalla de aviso permanece un tiempo indeterminado hasta que es cancelada por el usuario
    - No Modales: La pantalla permanece un tiempo definida por el programador
- La clase *List*: Nos permitirá construir pantallas que posean una lista de opciones, es decir, menús de usuario. Hay tres tipos de listas:
    - Implícitas: Al seleccionar un elemento provocamos una acción. Es el clásico menú. La acción la implementaremos con el método `commandAction(Command c, Displayable d)`
    - Exclusivas: Solo podemos seleccionar un elemento de la lista y los demás elementos serán deseleccionados.
    - Múltiples: Podemos seleccionar varios elementos.

  Podemos ver un ejemplo en la Fig. 3.2.

Fig. 3.2 Listas: implicita, explusiva y múltiple

- La clase *TextBox*: Nos permite editar texto en una pantalla.
- La clase *Form*: Es un componente que actúa como contenedor de objetos. Los objetos que puede contener derivan de la clase Item.

3.1.1.4 La clase Item

Como hemos dicho en el apartado anterior, los objetos de la clase *item* son los contenientes de la clase *Form*. Como hemos visto en el
Esquema 3.2, posee varias clases. Nos centraremos solo en la clase *Gauge* que sí que nos será muy útil para hacer un control de volumen.

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

- Interactivo: El usuario puede modificar el valor.
- No interactivo: Solo el programa actualiza el valor del objeto

En nuestro caso, si queremos implementar un control de volumen, deberá ser interactivo para que el usuario elija el nivel de volumen que desea. Podemos ver un ejemplo en la Fig. 3.3.

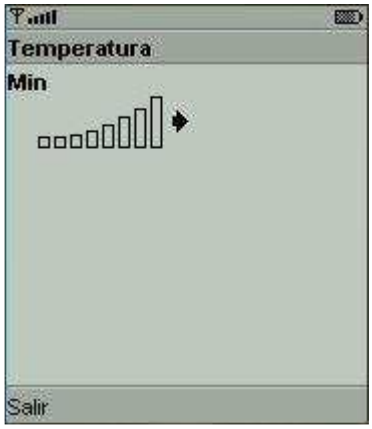


Fig. 3.3 Ejemplo de Gauge

3.1.1.5 La clase Canvas

La clase *Canvas* es la superclase de todas las clases de bajo nivel. Es muy útil en la creación de juegos, ya que nos da un control total de la pantalla.
No será muy útil en nuestro proyecto, pero sí que nos puede ofrecer vistosidad en alguna parte de la aplicación. Nos permitirá añadir imágenes de fondo e indicar en que posición queremos que se muestre la imagen.

3.1.2 Comunicaciones en JME. Bluetooth

Ahora que ya conocemos las posibilidades gráficas que nos ofrece JME, vamos a estudiar a fondo las opciones que tenemos para comunicarnos con otros dispositivos. Las librerías *input/output* de JME, nos permiten la comunicación mediante sockets, http, etc. pero lo que mas nos interesa es estudiar una API que nos proporcione una interfície Bluetooth.

Anteriormente, hemos estudiado el protocolo Bluetooth desde un punto de vista teórico. Sabemos como funciona el protocolo, pero ahora hay que programarlo. Gracias a la API *javax.bluetooth*, disponemos de diferentes clases y métodos que nos ayudarán en el proceso de búsqueda, conexión y transmisión de datos. Esta API, ofrece soporte tanto para RFCOMM, para L2CAP como para OBEX. En nuestro caso, como lo que queremos es transmitir datos o simplemente establecer conexión con los dispositivos, nos centraremos en el protocolo RFCOMM y en el perfil que provee este protocolo, el SPP (Serial Port Profile). RFCOMM, es un protocolo orientado a conexión, por lo que en todo momento tanto cliente como servidor sabrán si están conectados y llegan los mensajes. Si puede establecer conexión es que el dispositivo se encuentra en el radio de acción del sensor, y dicho sensor podrá enviar la alerta.

En este tipo de comunicaciones, un dispositivo se tiene que declarar como servidor, y los demás como clientes. Como ya hemos introducido anteriormente, en nuestro proyecto el rol de servidor y de cliente se irá alternando dependiendo del estado en que nos encontremos, modo registro o estado normal de recepción de alertas. Cuando estamos en modo registro, nuestro dispositivo móvil deberá ser capaz de actuar como cliente. Actuar como cliente implica que debe ser capaz de hacer una búsqueda de dispositivos, de sus servicios y de establecer comunicación con uno de los dispositivos y uno de sus servicios. Si estamos en el modo de funcionamiento normal, de recepción de alertas, el dispositivo móvil actuará como servidor. En el modo servidor, el dispositivo estará escuchando el canal a la espera de conexiones entrantes.

Como podemos ver, deberemos programar el dispositivo tanto para actuar como cliente, como para actuar de servidor. A continuación explicaremos detalladamente algunos de los métodos que nos ofrecen la API Bluetooth tanto para el modo servidor como para el modo cliente.

3.1.2.1 Servidor SPP

Como ya hemos comentado, el protocolo que utilizaremos es RFCOMM que simula el puerto serie. El perfil encargado de gestionar estas conexiones es el SPP (*Serial Profile Port, Perfil de Puerto Serie*).

Las responsabilidades de una aplicación servidora de Bluetooth son:

- Crear un Service Record que describa el servicio ofrecido por la aplicación.
- Añadir el Service Record para avisar a los clientes potenciales de este servicio.
- Registrar las medidas de seguridad Bluetooth asociadas a un servicio.
- Aceptar conexiones de clientes que requieran el servicio ofrecido por la aplicación.
- Actualizar el Service Record del servidor si las características del servicio cambian.
- Quitar o deshabilitar el Service Record del servidor cuando el servicio no está disponible.

Como estamos creando un servidor, debemos indicarle a nuestro dispositivo que debe ser visible por los demás dispositivos Bluetooth. Para ello, crearemos un objeto de la clase *LocalDevice* para recoger la información de nuestro dispositivo local. A continuación, le daremos visibilidad llamando al método `setDiscoverable()` y pasándole un entero que especifica el modo. Normalmente le pasaremos el valor *DiscoveryAgent.GIAC* (General/Unlimited Inquiry Access Code).

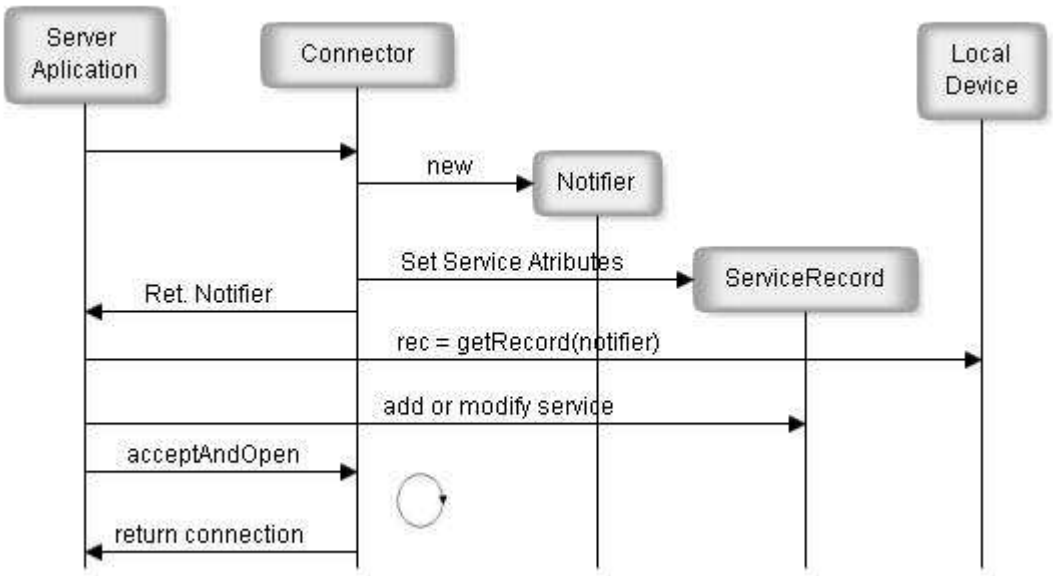
Para crear un conector que esté escuchando el canal, deberemos llamar al método `Connector.open()` y pasarle un String con la URL. La URL debe contener el protocolo, la dirección, el UUID (o número del protocolo a utilizar), el nombre del servicio y las políticas de seguridad. Este método, devuelve un objeto de la clase *StreamConnectionNotifier*.

Una vez tenemos este notificador, debemos registrar un servicio (Record). Para ello, llamaremos al método `getRecord(StreamConnectionNotifier notifier)` de la clase *LocalDevice*, al que pasaremos el *StreamConnectionNotifier* instanciado anteriormente.

En este momento, ya tenemos un servidor arrancado, con un servicio activo y esperando conexiones entrantes. En el momento en que un cliente quiera establecer una conexión, se llamará al método `acceptAndOpen()` de la clase *StreamConnectionNotifier*. Una vez creado el *StreamConnectionNotifier*, podemos obtener toda la información necesaria del dispositivo remoto, creando un objeto de la clase *RemoteDevice*

Cuando ya hemos creado la conexión, podemos proceder a instanciar los input y output streams para la recepción y envío de información.

A continuación, en el Esquema 3.3, mostramos un gráfico que resume todo el proceso.



Esquema 3.3 Servidor Bluetooth

3.1.2.2 Cliente SPP

Una vez tenemos levantado un servidor, debemos disponer de un cliente para poder realizar la comunicación. Un cliente Bluetooth deberá realizar las siguientes operaciones para comunicarse con un servidor Bluetooth:

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

Para empezar, partimos de la clase *LocalDevice*, usada anteriormente para declarar el dispositivo conectable en el modo servidor. A partir del objeto de la clase *LocalDevice*, podemos recoger un objeto único *DiscoveryAgent*. El objeto *DiscoveryAgent* nos va a permitir realizar y cancelar búsquedas de dispositivos y de servicios.

Una vez creado el objeto *DiscoveryAgent*, ya podemos empezar la búsqueda de dispositivos. Esto se hace llamando al método `startInquiry()`. La búsqueda de dispositivos, la controlaremos implementando la interfaz *DiscoveryListener*. Esta interfaz dispone de cuatro métodos que implementaremos según nuestros intereses. Estos métodos son:

- `deviceDiscovered()`: Se llama cada vez que se encuentra un dispositivo.
- `inquiryCompleted()`: Se llama cuando se finaliza la búsqueda de dispositivos.

- `servicesDiscovered()`: Se llama cada vez que se encuentra un servicio.
- `serviceSearchCompleted()`: Se llama cuando se finaliza la búsqueda de servicios.

Los dispositivos encontrados los iremos metiendo en un vector para su posterior análisis. Para conectarnos con un dispositivo, al igual que en el servidor, crearemos un objeto *StreamConnection* al que le pasaremos la dirección del servicio al que nos queremos conectar.

3.2 Sensor Bluetooth

Esta es la segunda parte del montaje. Como no disponemos de un sensor comercial para hacer las pruebas, lo simularemos con el fin de comprobar el funcionamiento de la aplicación móvil.

Para simular el sensor, utilizaremos un PC con sistema operativo Kubuntu. Kubuntu es un sistema operativo basado en Debian, con escritorio KDE. Este sistema operativo, tiene soporte para dispositivos Bluetooth. Más concretamente, usaremos `hcitool` y `sdptool` y el protocolo RFCOMM para enviar las alertas.

Para hacer más amigable la parte del sensor, utilizaremos una aplicación que nos permite de manera muy sencilla, dotar de parte gráfica a un Script. De esta forma, los paneles informativos y los interrogativos serán más sencillos y vistosos. El programa que utilizaremos es propio de los sistemas operativos con escritorio GNOME, aunque también lo podemos incorporar a los KDE, llamado zenity. Con una simple línea de código, podemos mostrar un panel gráfico. A continuación, en la Fig. 3.4 y en la Fig. 3.5, mostraremos algunos ejemplos de paneles gráficos realizados con la aplicación zenity.

```
zenity --info \
          --text="Combinación completa. Actualizado 3 de 10
archivos."
```

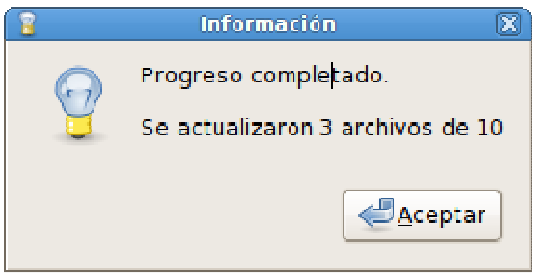


Fig. 3.4 Ejemplo zenity. Panel de información

```
zenity --question \
          --text="¿Está seguro de que desea continuar?"
```

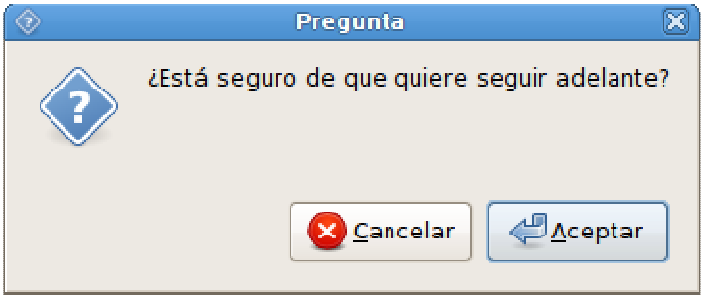


Fig. 3.5 Ejemplo zenity. Panel interrogativo

3.2.1 Modo registro

El sensor en modo registro, debe ser capaz de esperar una conexión y al recibirla, recoger la dirección MAC del dispositivo que intenta establecer conexión y guardarla en un fichero de texto.

En la Fig. 3.6, podemos ver un diagrama de flujo que explica el funcionamiento del programa sensor en modo registro.

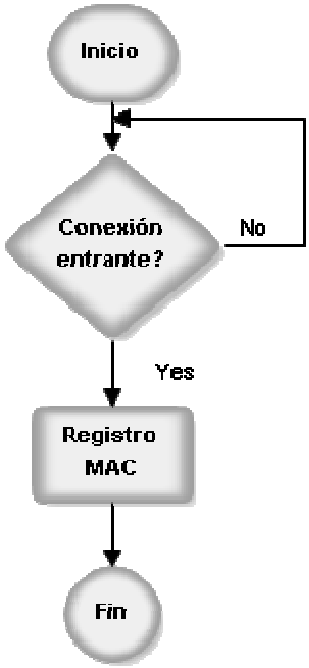


Fig. 3.6 Estados del modo registro

Para realizar la operación de registro, utilizaremos el programa de linux `rfcomm`, que como su nombre indica nos permitirá comunicarnos con un dispositivo Bluetooth mediante el protocolo RFCOMM.

Como queremos escuchar el canal, deberemos utilizar el parámetro *listen.* Además, le deberemos indicar el nombre de la interfície Bluetooth y el canal en el que escucharemos.

```
sudo rfcomm listen hci0 5
```

Con esto, obtendremos la siguiente salida:

```
Waiting for connection on channel 5
```

En estos momentos, el programa se quedará bloqueado a la espera de conexiones entrantes. En el momento en que reciba comunicación de algún dispositivo, mostrará la siguiente información por pantalla:

```
Waiting for connection on channel 5
Connection from 00:16:4E:26:61:81 to /dev/rfcomm0
Press CTRL-C for hangup
Disconnected
```

Como vemos, la primera línea es la misma que la que hemos mostrado anteriormente, pero ahora se ha añadido mas información. De toda esta información, necesitamos la dirección MAC de la segunda línea. Para obtenerla, deberemos *parsear* toda esta información.

Para realizar esta acción, utilizaremos el lenguaje Bash. Este lenguaje nos permitirá cortar la segunda línea y desechar todo lo que no sea la dirección MAC.

```
sudo rfcomm listen hci0 5 | tail -n 3 | head -1 | awk '{
print $3 }'
```

Con `tail` y `head` conseguimos seleccionar solo la segunda línea y finalmente con `awk` imprimimos la tercera palabra, la MAC.

Una vez obtenida, la guardamos en un fichero de texto para utilizarla posteriormente para lanzar alertas.

3.2.2 Modo Sensor

Una vez registrado, tendremos la posibilidad de lanzar alertas. De esta manera, simularemos el momento en el que el sensor se activa.

Este paso es un poco más complejo que el anterior, ya que hay que hacer varias operaciones. La primera a realizar es descubrir el canal que está utilizando el servicio de nuestra aplicación. El programa del sistema operativo para descubrir servicios Bluetooth es `sdptool`. El problema es que la salida de este programa es muy extensa, y dispone de mucha información innecesaria para nuestro propósito. Con el fin de depurar todo esa información innecesaria y obtener solo el canal en el que opera nuestro servicio, realizaremos un pequeño script en perl.

`Sdptool`, al igual que `hcitool`, dispone de múltiples opciones y modos de funcionamiento. Mas concretamente, `sdptool` nos permite buscar servicios, mostrar todos los servicios de un dispositivo, añadir o borrar servicios locales… A nosotros nos interesa que nos muestre todos los servicios de un dispositivo. Pero la lista es muy extensa, ya que los móviles disponen de múltiples servicios Bluetooth. Además, el mismo tipo de servicio lo pueden utilizar varias aplicaciones, con lo que la lista que obtenemos es muy extensa. La orden que utilizaremos para buscar servicios en un dispositivo es la siguiente.

```
sdptool browse --uuid 0x003 XX:XX:XX:XX:XX:XX
```

Este comando, nos mostrará por pantalla todos los servicios que usen el protocolo con UUID 0x0003 (RFCOMM) del dispositivo con esa MAC. El problema es que de toda esa lista, debemos quedarnos solo con la información relacionada con nuestra aplicación. Por lo tanto deberemos buscar el "*Service Name*" (nombre elegido por nosotros al programar la aplicación), y descartar todos los otros. A continuación mostramos la información relacionada con un *Service Name*.

```
Service Name: Headset Audio Gateway
Service RecHandle: 0x10014
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 2
Language Base Attr List:
  code_ISO639: 0x454e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Headset" (0x1108)
    Version: 0x0100
```

Como podemos ver en este fragmento de la salida de `sdptool`, tenemos un servicio de manos libre llamado Headset Audio Gateway. Utiliza entre otros, el protocolo RFCOMM que es el que nosotros buscábamos. Pero la información que de verdad necesitamos está en la novena línea, el canal. En este caso, el canal es el número 2.

En este momento tenemos dos problemas. Debemos rechazar todos los bloques de información de los diferentes servicios del dispositivo y una vez eliminados los bloques innecesarios, debemos quedarnos solo con el número de canal y rechazar toda la información extra. Para ello, escribiremos un programa en Perl que buscará el número de canal. Al programa deberemos pasarle tres parámetros externos, la MAC, el UUID del protocolo que queremos buscar y el nombre del servicio ("*Service Name*"). El programa lanzará `sdptool` con la MAC y el UUID que le hemos pasado como parámetro. El programa irá leyendo hasta que encuentre una cadena igual al *Service Name*. En este momento se activa una variable switch, y buscará la palabra channel. Cuando la encuentra, recoge el número de canal y lo mete en una variable. El programa en Perl finaliza y devuelve el canal. Si no encuentra el canal, el programa se dormirá durante 2 segundos y volverá a buscar hasta que encuentre.

Ahora que ya tenemos el canal, podemos lanzar RFCOMM. El protocolo RFCOMM emula los parámetros de un cable de serie y el estado de un puerto RS-232 para transmitir datos en serie. La sintaxis de este programa es la siguiente.

```
rfcomm connect hci0 XX:XX:XX:XX:XX:XX CHANNEL
```

Con esta orden, nos conectaremos mediante la inferfície hci0 a la MAC i al canal que le indiquemos. Para conocer el nombre de la interfície Bluetooth podemos utilizar el comando `hciconfig`.

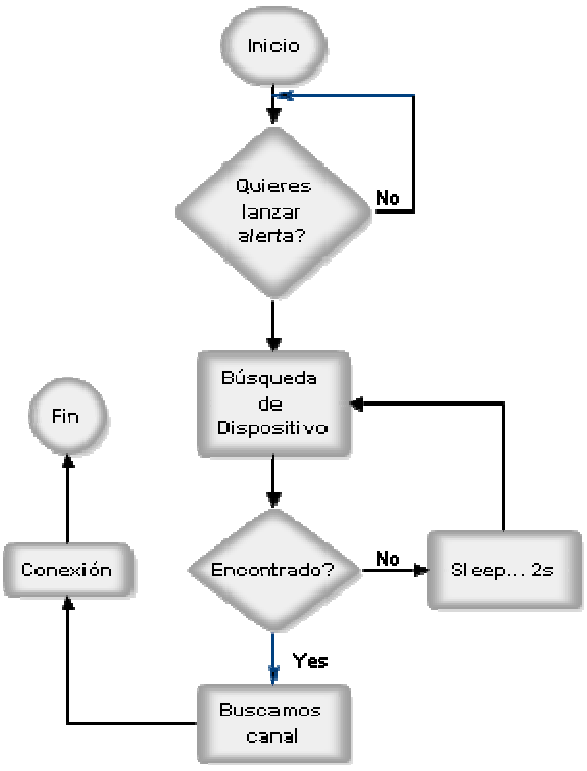


Fig. 3.7 Estados del modo sensor

Para aclarar este proceso, podemos ver el diagrama de flujo de la Fig. 3.7.

3.3 Clases de la Aplicación Java

En este apartado, vamos a fijarnos en las diferentes clases que tiene nuestra aplicación JME.

La aplicación está estructurada en cuatro package diferentes. Cada package gestionará una parte de la ésta. En la Fig. 3.8, vemos un diagrama UML que representa los diferentes packages que forman nuestra aplicación.

En primer lugar, tenemos el package *tfc.BTsensor.UI*. Este package es el que tiene el Midlet principal y la gran parte de la interfaz gráfica. El package *tfc.BTsensor.net* se encarga de gestionar las conexiones bluetooth en el modo de funcionamiento normal (modo sensor). El package *tfc.BTsensor.registro* controla los accesos a la *RecordStore* (base de datos del móvil), tanto para guardar la dirección del sensor como para guardar las preferencias de usuario. Además, es el encargado de buscar posibles dispositivos y realizar el registro. Por último, tenemos el package *tfc.BTsensor.config* que gestiona las preferencias de usuario (tono, volumen y vibración).

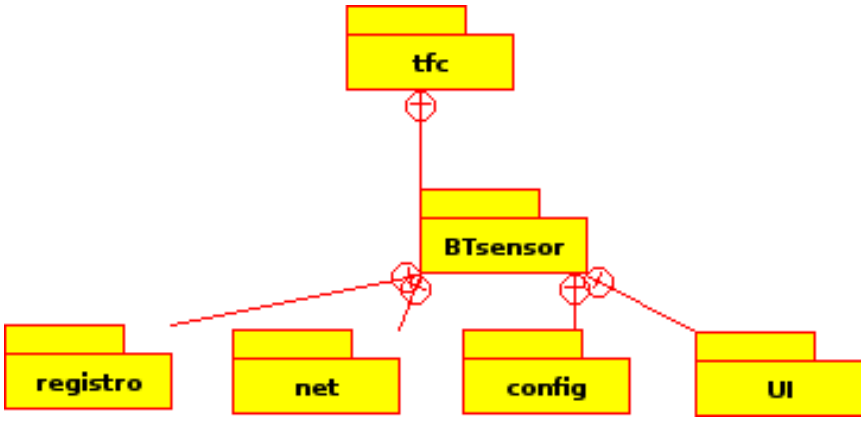


Fig. 3.8 Packages de la aplicación Java

A continuación vamos a detallar en profundidad las clases que forman los packages descritos anteriormente.

3.3.1 El package tfc.BTsensor.UI

Este package, controla la interfaz gráfica de usuario. Para analizar más concretamente que clases tiene, vamos a ver el siguiente diagrama UML. Fig. 3.9.

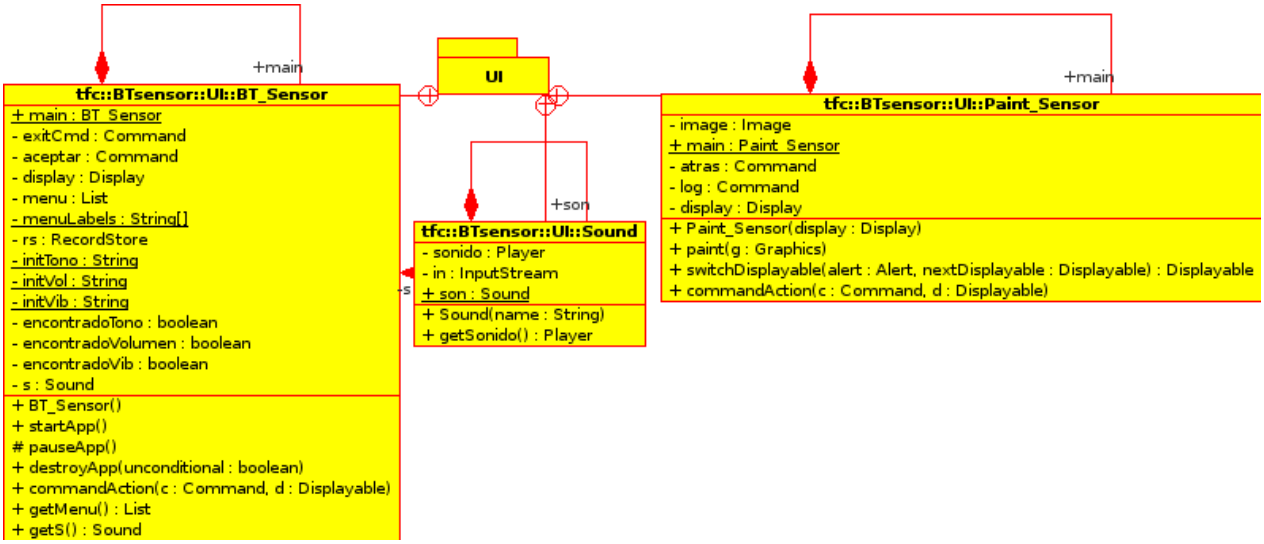


Fig. 3.9 Package tfc.BT_sensor.UI

Podemos observar que este package tiene tres clases. En primer lugar, tenemos la clase **BT_Sensor**, que es la clase principal de nuestra aplicación. Como podemos ver, tiene los métodos de Midlet tales como `startApp()`, `pauseApp()` y `destroyApp(bolean unconditional)`. A parte, inicia todas las *RecordStore* si es la primera vez que se arranca la aplicación en el móvil y les da un valor por defecto a las preferencias de usuario. Este Midlet dispone de un menú que permite seleccionar el modo de ejecución y seleccionar el menú de configuración.

A continuación tenemos la clase **Sound**, que controla la reproducción de sonidos de la aplicación. Su constructor recibe un parámetro con el nombre del sonido a reproducir (dependiendo de las preferencias de usuario) y crea un reproductor para reproducir el sonido. El método `getSonido()` devuelve el reproductor para que desde cualquier clase podamos reproducir un sonido.

Por último, la clase **Paint_Sensor**, nos sirve para “pintar” un elemento en uno de los menús del móvil. Esta clase extiende de la librería canvas, anteriormente explicada. Como podemos ver, tiene un método `paint(Graphics g)` que se encarga de añadir una imagen en la posición que nosotros le indiquemos. En nuestra aplicación, no es imprescindible utilizar estas librerías, pero si que se utilizan en el desarrollo de juegos para móviles.

3.3.2 El package tfc.BT_sensor.net

Este package se encarga de la gestión de las comunicaciones Bluetooth en el modo de funcionamiento normal. Para analizar las clases que dispone, vamos a ver la Fig. 3.10.

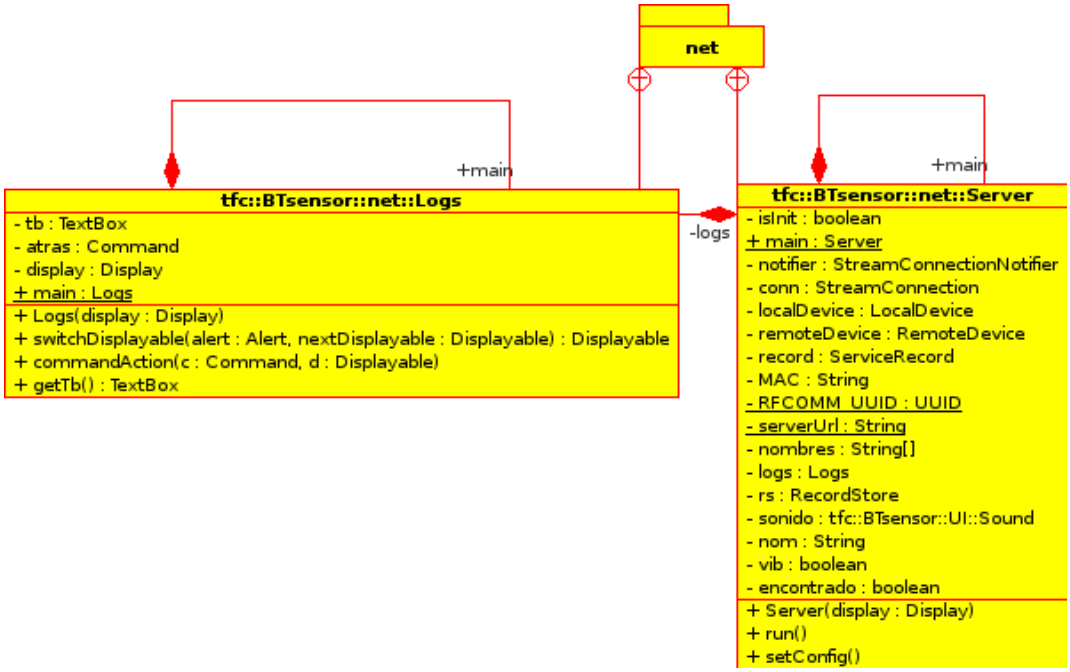


Fig. 3.10 Package tfc.BT_sensor.net

Como apreciamos en el UML, este package tiene dos clases, la clase **Server** y la clase **Logs**.

La clase **Server**, es la que dispone de los métodos necesarios para ejecutar un servidor Bluetooth. El constructor, llama a los métodos de la clase Paint_Sensor para crear la interfaz gráfica de la pantalla. A la vez, arranca un Thread para que escuche las peticiones Bluetooth. En cuanto le llega una petición, si la dirección de esta coincide con la que tiene registrada previamente, lanzará una alerta por pantalla y reproducirá un sonido. Si el usuario lo desea, vibrará durante unos segundos.

Por otro lado, tenemos la clase **Logs**. Esta clase es útil durante el desarrollo de la aplicación porque nos irá mostrando por pantalla todos los pasos que sigue la ejecución del programa. De esta forma, podemos ir depurando la aplicación durante la fase de desarrollo.

3.3.3 El package tfc.BT_sensor.registro

Este package, contiene las clases necesarias para buscar un dispositivo Bluetooth, realizar una conexión con el y acceder a la *RecordStore* de nuestro móvil para guardar una dirección MAC. A continuación, en la Fig. 3.11, vamos a ver en detalle las clases que contiene este package.

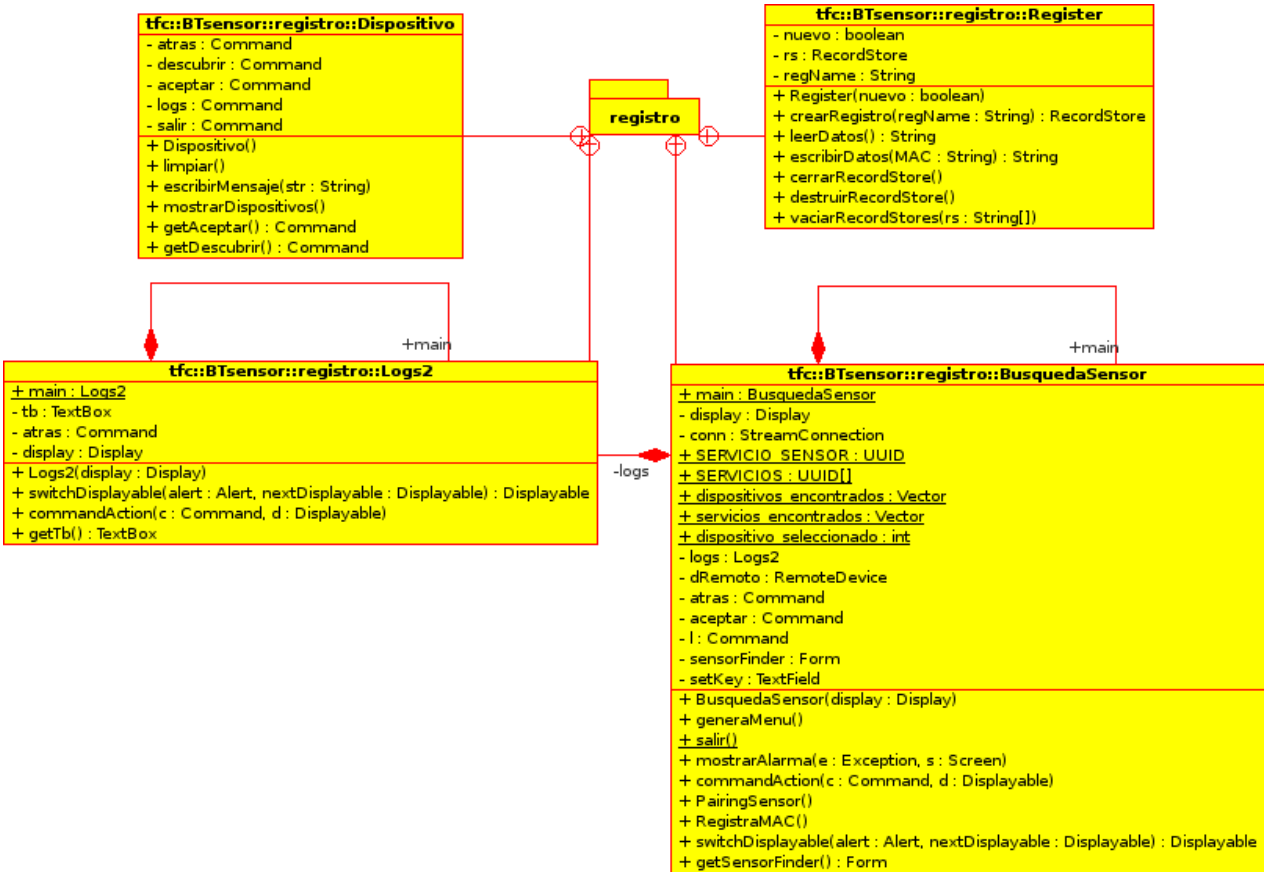


Fig. 3.11 Package tfc.BT_sensor.registro

Como podemos ver en el UML, este package contiene cuatro clases. Tres de ellas, controlan la búsqueda del dispositivo Bluetooth y lo muestra por pantalla. La otra clase, es la encargada del acceso a *RecordStore*.

La clase **Client** se encarga de recoger un *String* con la dirección MAC del dispositivo sensor, que será entregada con la compra del mismo. A partir de esta dirección, realizaremos la conexión con el sensor que deberá estar escuchando para poder registrar el móvil.

Este procedimiento nos aporta seguridad al hacer el registro del sensor. El sensor Bluetooth, podrá estar en estado “no visible”, con lo que solo podrá realizar la conexión con él, alguien que sepa su dirección. El fabricante del

sensor, nos la proporcionará en el momento de la compra y de esta forma, se asegura que sólo la persona que ha realizado la compra de dicho dispositivo pueda registrar su móvil.

La clase **Register**, se encarga del acceso a la base de datos del móvil (*RecordStore*). Contiene los métodos `crearRegistro( String regName )`, que genera una nueva *RecordStore*. `leerDatos()` devuelve todos los datos que contiene una *RecordStore*. `escribirDatos( String MAC )` permite introducir datos en una *RecordStore*. `cerrarRecordStore()` permite cerrar una *RecordStore*. `destruirRecordStore()` elimina una *RecordStore* en concreto. Y finalmente `vaciarRecordStores( String[] rs )` permite eliminar todas las *RecordStore* de un dispositivo.

3.3.4 El package tfc.BT_sensor.config

Finalmente, el package config controla las preferencias de usuario tales como el tono de alarma, el volumen y la vibración. Para detallar las clases vamos a ver la Fig. 3.12.

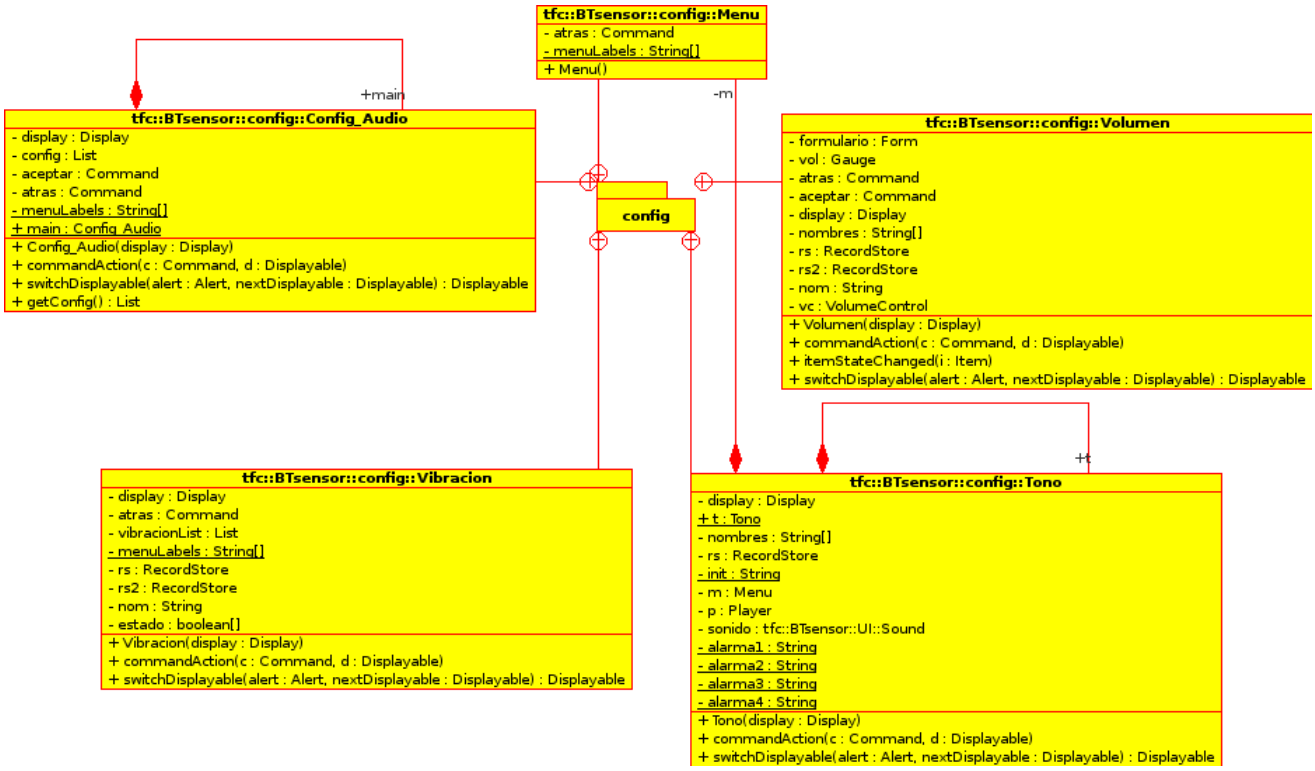


Fig. 3.12 Package tfc.BT_sensor.config

Como podemos ver, este package dispone de cinco clases: **Menu**, **Config_Audio**, **Tono**, **Volumen** y **Vibracion**.

La clase **Menu** muestra las diferentes opciones a configurar, audio, tono o vibración. La clase **Config_Audio** muestra los diferentes tonos que podemos escoger. Por último, las clases **Tono**, **Volumen** y **Vibración** son las que se encargan de recibir la opción elegida por el usuario y guardarlas en una *RecordStore*. De esta manera, la próxima vez que ejecutemos la aplicación conservaremos el perfil de usuario.

CAPÍTULO 4. EJEMPLO DE USO DE LA APLICACIÓN

4.1 Ejemplo de uso de la aplicación

En este apartado, vamos a describir el uso de la aplicación desarrollada. Supongamos que compramos el sensor Bluetooth y la aplicación para el dispositivo móvil. Fig. 4.1.

Fig. 4.1 Pantalla principal

En principio, si no hemos usado nunca el producto, deberemos registrarlo. Para ello, habrá que poner el sensor en "*modo registro*" y abrir el menú "*Registrar Sensor*". En nuestro caso, en el entorno simulado, ejecutaremos el script de Linux poniendo el siguiente comando:

```
sudo ./sensor
```

Fig. 4.2 Bienvenida del simulador

En este momento, se nos abrirá una pantalla como la de la **Fig. 4.2**.

A continuación, aceptaremos y se abrirá un nuevo formulario como el de la Fig. 4.3 preguntando si queremos registrar un dispositivo.

Fig. 4.3 Menú de petición de registro del simulador

Si aceptamos, el simulador del sensor se quedará a la espera de una conexión entrante. Es el momento de abrir el menú "*Registrar Sensor*", que vemos en la Fig. 4.4, e introducir la clave de producto que nos habrá proporcionado el fabricante del sensor.

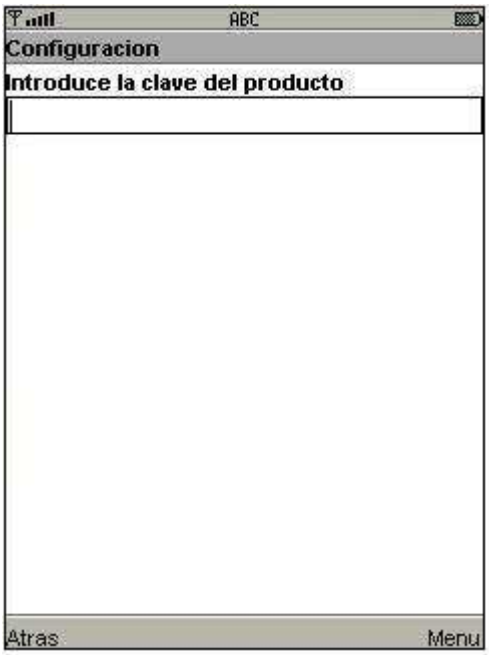


Fig. 4.4 Pantalla de registro de la aplicación

Aceptamos, y veremos como se conecta con el simulador del sensor y el móvil vuelve al menú principal. A continuación, el simulador del sensor nos preguntará si queremos lanzar alerta. Fig. 4.5.

Fig. 4.5 Petición de lanzar alerta del simulador

Para poder lanzar alertas, deberemos abrir el menú “Empezar” de la aplicación móvil. A continuación se nos aparecerá la Fig. 4.6. Esta pantalla permanecerá activa esperando alertas.

Fig. 4.6 Pantalla de recepción de alertas

En el momento que aceptemos “enviar alerta” en el simulador, escucharemos la alarma en el móvil.

Se pueden lanzar tantas alertas como se quiera, ya que una vez recibida, el simulador muestra otra vez el menú “enviar alerta”.

Por otro lado, si queremos ajustar preferencias de usuario, disponemos del menú “Configuración” de la pantalla principal. Si lo abrimos, veremos que podemos ajustar el volumen, la vibración y escoger un tono de alerta. Fig. 4.7.

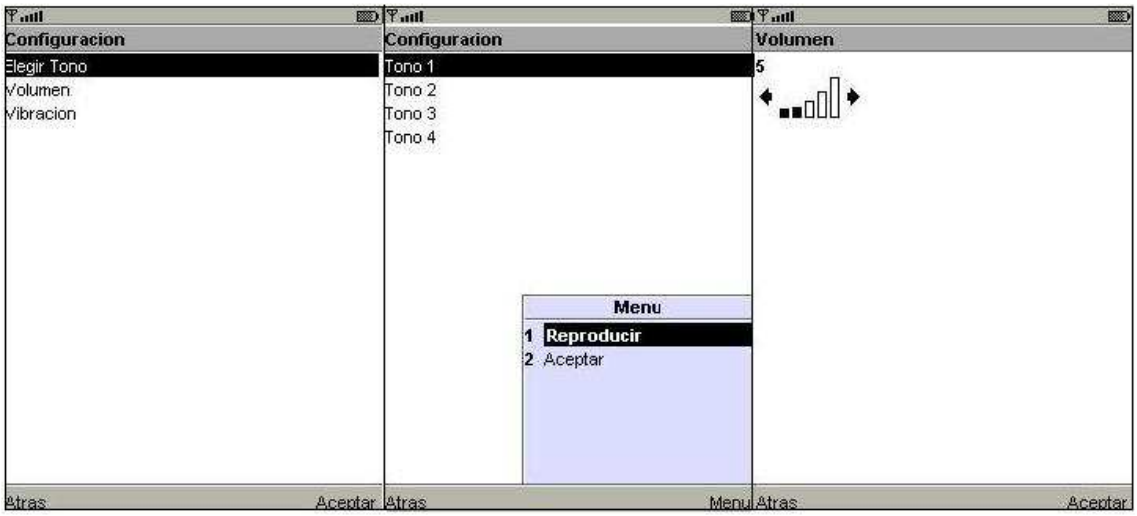


Fig. 4.7 Menús de configuración

CAPÍTULO 5. CONCLUSIONES Y LÍNEAS FUTURAS

Finalmente, vamos a cerrar este TFC extrayendo algunas conclusiones sobre el resultado obtenido, apuntaremos posibles líneas futuras para continuar este trabajo y realizaremos un pequeño estudio sobre los efectos de nuestro proyecto al medio ambiente.

5.1 Conclusiones

En este TFC, hemos desarrollado una aplicación para un dispositivo móvil que permitirá ayudar al cuidado de los pacientes con movilidad reducida y/o dificultades comunicativas.

Partiendo de unas especificaciones iniciales, hemos definido qué tecnologías y dispositivos son los más adecuados para este fin.

Como lenguaje de programación, hemos escogido el lenguaje Java. Esto es debido a su amplia implementación en dispositivos móviles y sus propiedades de multiplataforma, que nos permitirá usar el mismo código en cualquier dispositivo que disponga de la máquina virtual de Java Micro Edition.

También hemos analizado las diferentes tecnologías de comunicación inalámbrica. Al final nos hemos decantado por la tecnología Bluetooth por su amplia presencia en la mayoría de dispositivos móviles, porque JME nos ofrece librerías para la transmisión de datos mediante esta tecnología y, sobre todo, porque es la que, cubriendo nuestras necesidades, tiene un menor consumo de recursos (especialmente de batería).

Por último, para simular el sensor, hemos decidido utilizar la interfície Bluetooth de Linux y los lenguajes Bash y Perl para parsear los datos de los programas del Sistema Operativo.

Durante la realización de este TFC, nos hemos encontrado con algunas dificultades. Trabajar con el protocolo Bluetooth, es muy diferente a utilizar sockets TCP y UDP. A priori, como Bluetooth es orientado a conexión, parece similar a TCP. Pero al empezar a estudiar esta tecnología, nos dimos cuenta que es un poco diferente. Los sockets TCP, se identifican mediante una dirección IP y un puerto. En cambio, en Bluetooth las conexiones se forman a partir de una dirección MAC (ya que es un protocolo a nivel de enlace y no de transporte), un servicio y un número de canal de comunicación. Por este motivo, antes de poder realizar la comunicación, deberemos saber la dirección MAC del receptor, el servicio al que nos queremos conectar y el canal que utiliza dicho servicio.

Otra dificultad que nos hemos encontrado, ha sido configurar correctamente el Bluetooth de Linux. Sobre este tema hay muy poca documentación y surgieron muchos problemas, pero finalmente encontramos la forma de resolver el

emparejamiento del móvil con el simulador del sensor. Para incrementar la seguridad, no anunciamos los servicios del sensor y para realizar el emparejamiento se debe introducir a mano en la aplicación del móvil la MAC del sensor, que utilizamos como clave de conexión. Esto nos permite tener oculto el sensor y solo quien conozca la dirección podrá conectarse y nos ofrece seguridad ante ataques contra nuestro sensor, ya que no podrá ser encontrado.

A pesar de todos los problemas, en este TFC hemos conseguido implementar todo el sistema de envío y recepción de alertas, de un sensor en un paciente a un móvil en un personal médico, mediante conexión Bluetooth. La aplicación del móvil, por otra parte, ofrece una interfaz gráfica amigable y fácilmente configurable para sus posibles usuarios.

5.2 Líneas Futuras

Una vez finalizado este TFC, podemos pensar en futuras ampliaciones de la aplicación. En estos momentos, el programa es capaz de registrar y recibir alertas. Pero si quisiéramos ir más allá, podríamos implementar una aplicación en Java que gestionara las alertas que llegan a la aplicación móvil. Podríamos utilizar las librerías Bluetooth para JavaSE para recibir periódicamente el historial de alertas que recibe un dispositivo móvil. El móvil, enviaría una vez al día el número de alertas que ha recibido y las horas. Un servidor central, recogería este informe diario y lo procesaría. Mediante los conectores JDBC de acceso a bases de datos, podríamos guardar el histórico del paciente para posteriores consultas.

Otra posible ampliación del proyecto, puede ser centrarse más en las librerías de bajo nivel, para darle un aspecto mucho más vistoso y añadirle más opciones de usuario, como el envío y recepción de tonos.

5.3 Estudio de ambientalización

El único punto negro que podemos encontrar en este proyecto, referente a contaminación medioambiental, es la contaminación electromagnética producida por la transmisión Bluetooth. Pero lo cierto es que los dispositivos móviles con que hemos trabajado son dispositivos Bluetooth de clase 2 con muy baja potencia de transmisión, entre los 0,25 y 2,50 mW. Por lo tanto no podemos considerar que su radiación sea dañina o peligrosa, especialmente comparándola con otras radiaciones que solemos encontrar en cualquier espacio público, donde convivimos con ondas de radio, telefonía móvil, telefonía inalámbrica domestica, Wifi, televisión, radio, etc.

Es decir, podemos asegurar que estamos ante una tecnología de comunicación inalámbrica poco dañina en términos de radiación y que, por tanto, no refleja ninguna problemática destacada en el campo medioambiental real.

BIBLIOGRAFÍA

[1] Gálvez Rojas, S. y Ortega Díaz, L., “Java a tope: J2ME (JAVA 2 MICRO EDITION)”, Edición Electrónica, Málaga, 2003.

[2] Prieto, M.J., “INTRODUCCIÓN A J2ME (Java 2 MicroEdition)”, Edición Electrónica, España, 2002.

[3] Garcia, C., “El cableado del siglo XXI”.
Web http://www.it.uc3m.es/cgarcia/articulos/tutorials/bluetooth-cgarcia.pdf

[4] Página oficial de NetBeans. Documentación sobre NetBeans.
Web http://www.netbeans.org/kb/index.html

[5] Página oficial de Bluetooth. Documentación sobre Bluetooth.
Web http://spanish.bluetooth.com/Bluetooth/Technology/

[6] Página oficial de Java2 Micro Edition. Documentación sobre J2ME.
Web http://java.sun.com/javame/reference/index.jsp

[7] Guía avanzada de Bash-Scripting.
Web http://tldp.org/LDP/abs/html/

[8] Manuales de Perl
Web http://www.perl.org/

[9] Historia del lenguaje Java
Web http://www.cad.com.mx/historia_del_lenguaje_java.htm

[10] Seguridad en Bluetooth. Generación de claves.
Web
http://www.robotiker.com/revista/articulo.do%3Bjsessionid=22EB3357A3AA02E1BA5CB3E6B0ABF9CB?method=detalle&id=38

[11] Descripción completa del perfil: Perfil básico de impresión (BPP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/BPP.htm

[12] Descripción completa del perfil: Perfil de transferencia de archivos (FTP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/FTP.htm

[13] Descripción completa del perfil: Perfil manos libres (HFP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/HFP.htm

[14] Descripción completa del perfil: Perfil de aplicación de descubrimiento de servicio (SDAP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/SDAP.htm

[15] Descripción completa del perfil: Perfil de servicio de puerto (SPP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/SPP.htm

[16] Descripción completa del perfil: Protocolo de intercambio de Objetos (OBEX)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/OBEX.htm

[17] Descripción completa del perfil: Protocolo de control de telefonía (TCP)
Web http://spanish.bluetooth.com/Bluetooth/Technology/Works/RFCOMM.htm

[18] Descripción completa del perfil: RFCOMM con TS 07.10
Web
http://spanish.bluetooth.com/Bluetooth/Technology/Works/RFCOMM_1.htm