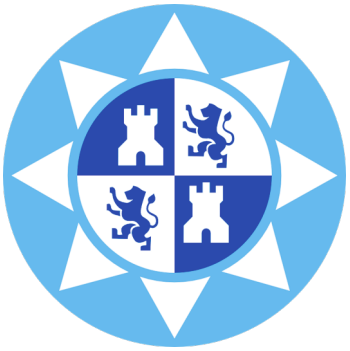
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Universidad Politécnica
De Cartagena

Proyecto Fin de Carrera

DESARROLLO DE UNA APLICACIÓN PARA ANDROID DE GEOLOCALIZACIÓN Y COMUNICACIÓN EN INTERIORES MEDIANTE BLUETOOTH

AUTOR: Alejandro Barceló Sánchez
DIRECTOR: Juan Carlos Sánchez Aarnoutse
Noviembre/13

Autor	Alejandro Barceló Sánchez
E-mail del autor	alexbarcelos90@gmail.com
Director	Juan Carlos Sánchez Aarnoutse
E-mail del director	JuanC.Sanchez@upct.es
Título del PFC	DESARROLLO DE UNA APLICACIÓN PARA ANDROID DE GEOLOCALIZACIÓN Y COMUNICACIÓN EN INTERIORES MEDIANTE BLUETOOTH

Resumen

El objetivo del proyecto es crear una aplicación para localización y comunicación en interior, de manera que no se necesite GPS, sino otras tecnologías como Bluetooth.

La aplicación comenzará a escanear dispositivos Bluetooth en la zona. Los dispositivos que nos interesan ya son conocidos con anterioridad, con lo cual, la aplicación indicará la posición de nuestro terminal en función del número de dispositivos encontrados y de la posición de éstos.

Una vez completa la fase de búsqueda, nos conectaremos con cada uno de los dispositivos encontrados para intercambiar información, la cual se almacenará en una base de datos.

Se debe destacar que cuando hablamos de dispositivos Bluetooth, se hace referencia a cualquier nodo que implemente dicha tecnología, como pueden ser impresoras, móviles, tablets, hardware industrial, etc.

Titulación	Ingeniero Técnico de Telecomunicaciones, Esp. Telemática
Departamento	Tecnologías de la información y las comunicaciones
Fecha de presentación	11/2013

AGRADECIMIENTOS

Quiero agradecer y dedicar este trabajo a mi familia, especialmente a mis padres y hermanos.

Merecen una mención destacada mis amigos y compañeros de clase, ya que me han apoyado en cada momento de la carrera, haciendo este duro camino mucho más fácil. También quiero mencionar a todos los profesores que en algún momento han contribuido a mi formación. En este punto quiero expresar mi agradecimiento a Juan Carlos Sánchez Aarnoutse por el trato recibido durante el desarrollo de ese Proyecto Fin de Carrera y por ser muy benevolente a pesar de retrasarnos más de lo que debimos.

Finalmente, quiero destacar a Antonio Hellín Hernández, compañero y por supuesto amigo, ya que ha sido un gran apoyo para la realización de este proyecto, consiguiendo ambos los objetivos deseados.

Muchas gracias a todos por vuestra ayuda, tiempo y dedicación.

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

Este proyecto nace con el objetivo de familiarizarse con las características del sistema operativo para móviles Android. Dicha plataforma se ha ido extendiendo rápidamente entre la sociedad dada su similitud al popular iPhone, pero con la ventaja que supone, tanto para usuarios como desarrolladores, el haber sido distribuido como código abierto, permitiendo un importante crecimiento en el abanico de aplicaciones disponibles. Este condicionante ha posibilitado el nacimiento de una extensa red de información al alcance de cualquiera, que ofrece el soporte necesario para desarrolladores nóveles.

Estos enormes fondos de información permiten obtener un apoyo indispensable para las tareas formativas orientadas a conocer el funcionamiento de la plataforma en su conjunto, absolutamente necesarias para el fin último del proyecto, el desarrollo de una nueva aplicación y su puesta a disposición de los usuarios potenciales.

Para conseguir el objeto final, el primer paso necesario es la investigación del funcionamiento del sistema operativo Android, las posibilidades de las características ofrecidas y su gestión por parte de las APIs disponibles y el manejo de aplicaciones.

Centrándonos en las tareas de desarrollo, es indispensable el conocimiento de la completa herramienta ofrecida para tal fin, así como su integración y uso en entornos de desarrollo ya existentes.

Finalmente, se llevará a cabo la implementación de la aplicación propiamente dicha, orientando a ser probado y testado sobre varios terminales. De forma complementaria, la aplicación se acompaña de la documentación necesaria para su correcta instalación y uso, así como su estudio para facilitar la compresión de sus componentes y la interconexión existente, al igual que posteriores aplicaciones por parte de terceros desarrolladores.

1.2. Objetivos

El principal problema a la hora de realizar aplicaciones para Smartphones que requieran geolocalización y no dispongan de señal GPS, es la imposibilidad de realizar una ubicación más o menos correcta de éste. Este hecho se da en aplicaciones *indoor*, aplicaciones para edificaciones interiores.

El objetivo de este proyecto es proporcionar al usuario un método de localización y conexión en interiores sin necesidad de una señal GPS. La tecnología que vamos a utilizar par llevar a cabo este proceso es *Bluetooth*.

Mediante la interfaz Bluetooth de los dispositivos empleados y utilizando el método de triangulación, podremos detectar aproximadamente nuestra situación en un plano interior.

Se puede decir que el proyecto podría dividirse en dos grandes partes, una centrada en la búsqueda de dispositivos Bluetooth, ya conocidos con anterioridad, y otra orientada a la conexión con estos, pudiendo así transferir cualquier información deseada. Estos datos recibidos son almacenados en el terminal en una base de datos, pudiendo acceder ellos posteriormente con la finalizad que se desee.

CAPÍTULO 2

TECNOLOGÍAS EMPLEADAS

2.1. Android

2.1.1 Introducción a Android

Android es una plataforma abierta de software para dispositivos móviles. Está desarrollada por la *Open Handset Alliance*, compuesta por más de 50 empresas y liderada por Google. Esta plataforma está compuesta por el Sistema Operativo, *Midleware* y aplicaciones claves del sistema. Está basado en un núcleo Linux 2.6 que hace de capa de abstracción entre el hardware y el resto del sistema, ofreciendo a la capa del software los diversos servicios existentes, como la seguridad, gestión de procesos y memoria, pila de red y modelo de drivers.

Por encima del núcleo se han diseñado una serie de capas que completan un entorno de desarrollo asequible para cualquier programador.

El *runtime* es un conjunto de bibliotecas que ofrece la mayoría de las funcionalidades disponibles en el núcleo de Java, lenguaje mediante el cual está programado el sistema.

Uno de sus principales componentes es la Máquina Virtual Dalvik. Android permite que un dispositivo pueda ejecutar varias máquinas virtuales, de modo que cada aplicación sea lanzada como un proceso independiente y con su propia instancia de la máquina virtual, a la vez que supone un bajo consumo de recursos.

El conjunto de bibliotecas, escritas en C/C++, son usadas para muchos de los programas de Android, y están puestas a disposición del programador mediante el Framework de aplicaciones. Sus competencias abarcan desde la propia gestión del sistema, como *System C*, hasta la manejo de bases de datos, con SQLite, pasando por diversas librerías multimedia con soporte para audio y video 2D y 3D, como *Surface Manager*, *Media Framework*, *OpenGL*, o *FreeType*, o la navegación web facilitada por el motor *LibWebCore*, entre otras funcionalidades.

El Framework de Android es el componente que da soporte a los desarrolladores para el uso de las características anteriormente expuestas, y es, además, el mismo sobre el que están diseñadas las aplicaciones propias incluidas en la plataforma Android.

Permite una sencilla reutilización de componentes y comunicación ente aplicaciones, siempre sujetas a ciertas medidas de seguridad, que facilita, por ejemplo, la actualización o sustitución de componentes por parte del usuario, resultando un sencillo y efectivo método para utilizar novedades o introducir mejoras en el software.

Los principales conjuntos de servicios ofrecidos son los siguientes:

- Un extenso y variado conjunto de vistas (*Views*) ofrecidas para el diseño de interfaces gráficas de usuario y su interactividad con el sistema, como los típicos botones, cuadros de texto o listas.

• Los proveedores de contenidos (*Contents Providers*) son el método de intercambio de información entre aplicaciones, ya sea compartiendo los datos propios o accediendo a los de otras aplicaciones.

• Los gestores de recursos (*Resources Manager*) permiten el acceso indexado a recursos como cadenas o gráficos, en un intento de modular aún más el diseño de las aplicaciones y el uso de sus recursos.

• El gestor de notificaciones (*Notification Manager*) dirige alertas personalizadas al software, que son mostradas en una barra de estado.

• Finalmente, el gestor de actividades (*Activities Manager*) es responsable ciclo de vida de las aplicaciones.

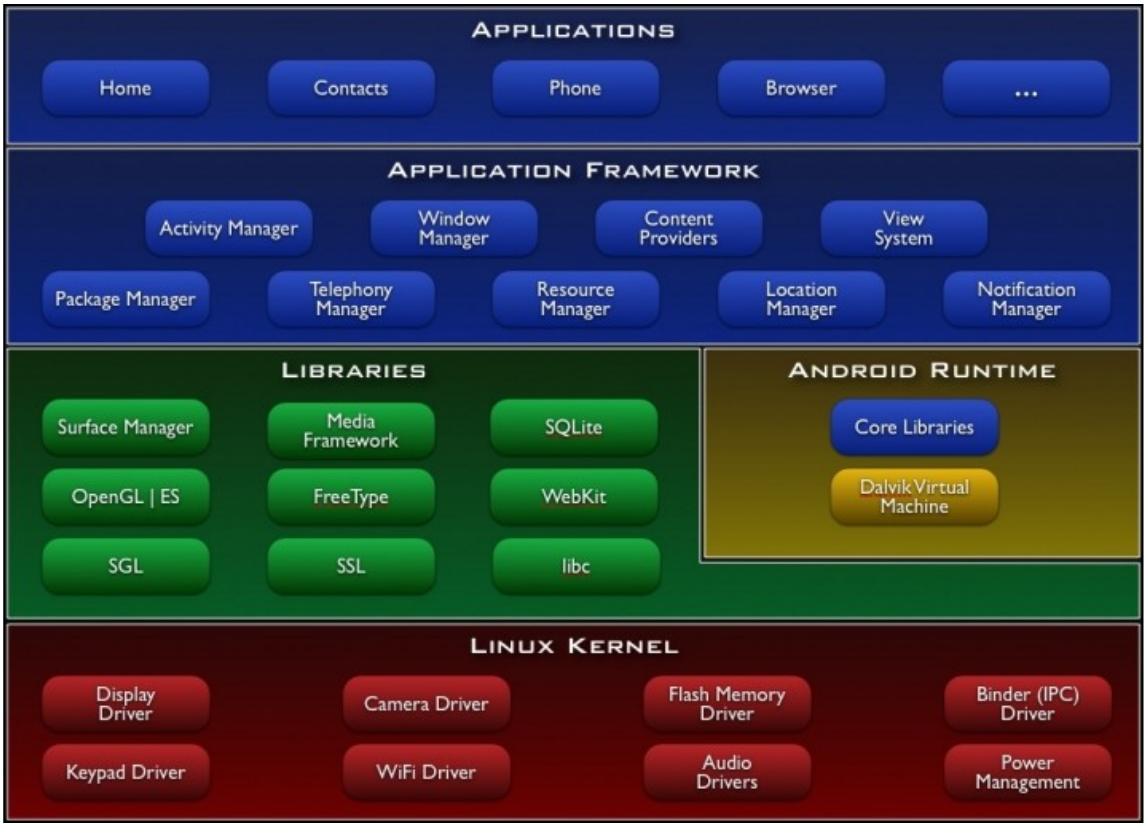


Figura 1: Arquitectura Android

Una vez proporcionados todos estos estratos, Android proporciona una serie de aplicaciones base, programadas en Java, que facilitan el manejo de las características del sistema, tales como un cliente de correo electrónico y SMS, calendario, mapas, navegador o gestor de contactos, entre otros. Sobre esta misma capa será donde se sustenten las aplicaciones de usuario diseñadas por terceros desarrolladores.

El resto de características del sistema Android podríamos agruparlas por conectividad (telefonía GSM, Bluetooth, EDGE, WIFI, 3G), formatos multimedia soportados (MPEG4, H.264, MP3, OGG, AAC, AMR, JPG, PNG, GIF) y otras características hardware, tales como GPS, etc.

En el ámbito de desarrollo, dispone de un complemento para el IDE Eclipse, así como herramientas de simulación y depuración. Las aplicaciones creadas pueden ponerse a disposición del usuario a través del Market Android, donde podrán ser descargadas e instaladas por los usuarios, sirviendo igualmente como fuente de reporte de fallos y errores en el funcionamiento.

Otra peculiaridad de Android es el ciclo de vida de las actividades que componen la apliación, ya que no se trata únicamente de abrir y cerrar a gusto del usuario, si no que éstas, una vez iniciadas, permanecen cargadas en memoria siempre que se disponga de recursos para ello. En caso contrario el propio sistema operativo se encargará de destruirlas definitivamente. Dicho ciclo de vida se rige por las llamadas a los métodos *onCreate*, *onStart*, *onResume*, *onPause*, *onStop*, *onStop*, *onDestroy* y *onRestart*. En el gráfico adjunto puede entenderse este flujo de forma más intuitiva.

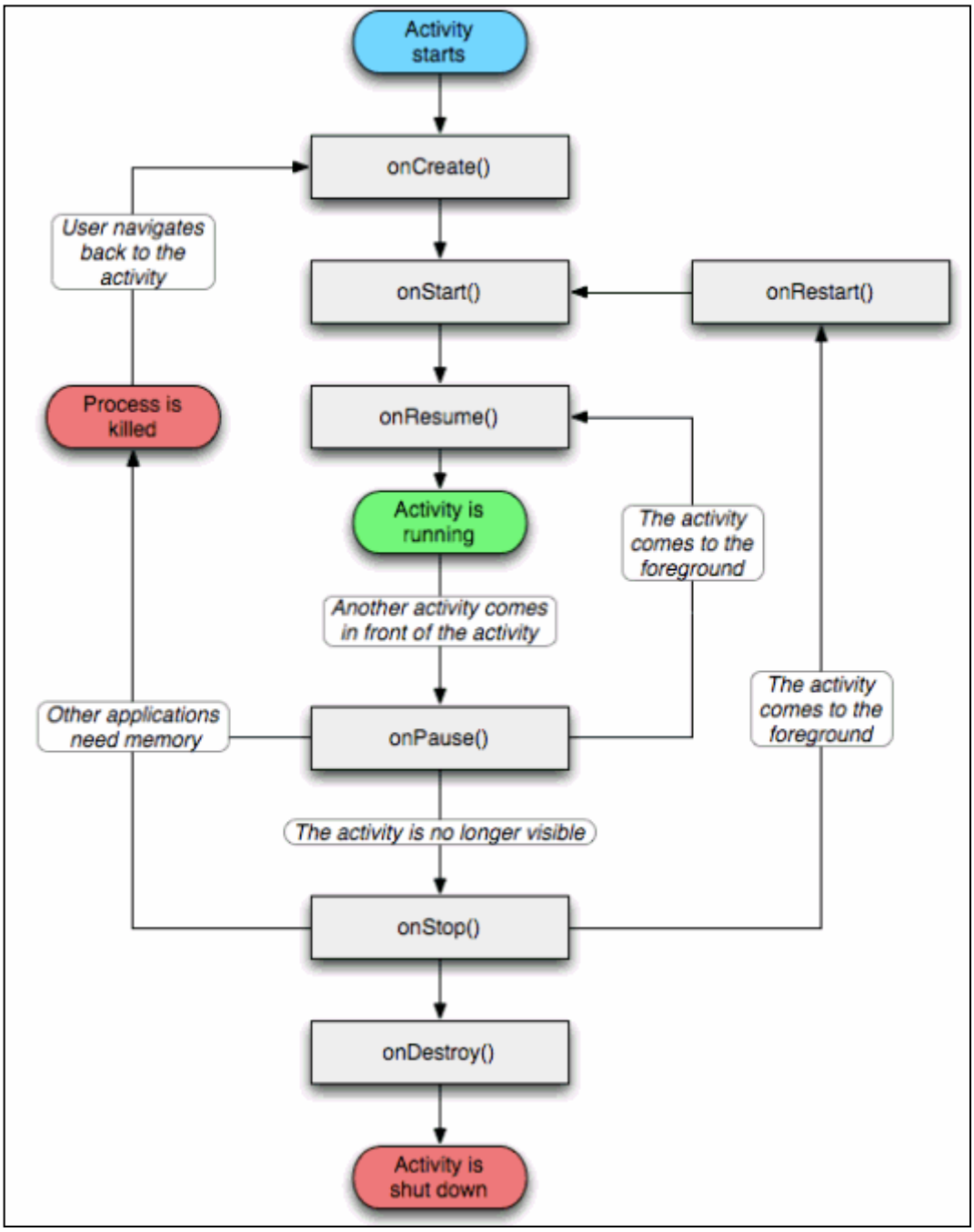


Figura 2: Ciclo de vida de una aplicación Android

2.1.2. Instalación del entorno de trabajo

En este manual se va a tratar la instalación paso a paso de las herramientas necesarias para comenzar a desarrollar nuestra aplicación sobre un sistema Android. Los principales componentes son el SDK de Android, el *plugin* ADT para eclipse, así como el propio IDE Eclipse. En este caso se trabaja con el *IDE Eclipse Índigo for Java Developers*, por lo este será el caso que se detalla para la instalación del SDK Android.

2.1.2.1. Instalación de Eclipse

Aunque la instalación del IDE no forma parte del proyecto, vamos a proceder a su instalación, que es fácilmente abordable desde su propia web (http://www.eclipse.org). En la sección *Download*, se elige la versión deseada, que en nuestro caso será *Eclipse IDE for Java Developers* para *Mac OS X 32 Bit*. Una vez descargado, debemos descomprimirlo y finalmente, basta con copiarlo en la carpeta de *Aplicaciones* de nuestro sistema operativo, *Mac OS X*, ya que no requiere instalación.

2.1.2.2. Instalación del SDK Android

Entre los numerosos recursos disponibles en la web de Android (http://developer.android.com) tenemos todas las versiones del SDK, en la sección *"Get the SDK"*. En nuestro caso nos ofrece la versión para *Mac OS X*.

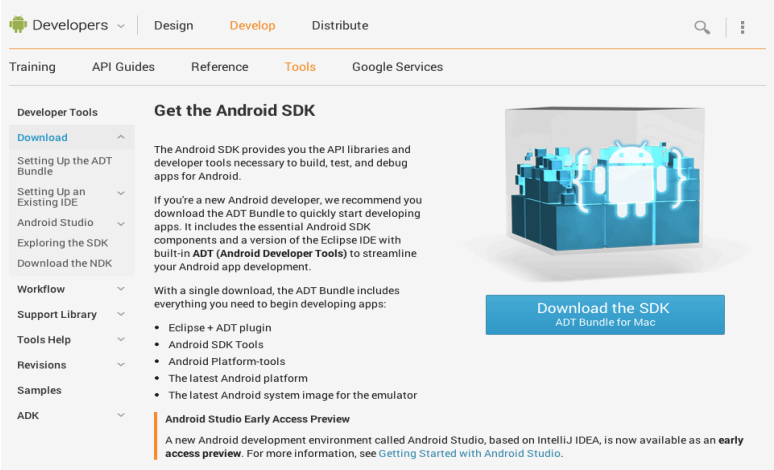
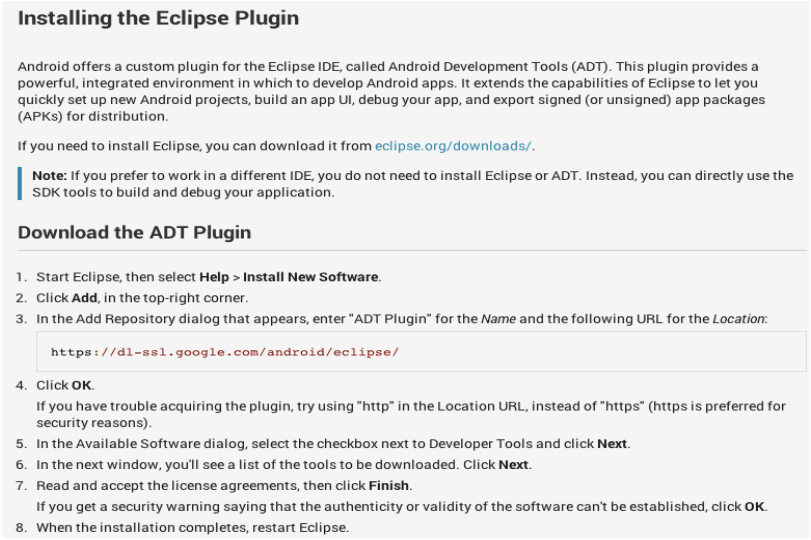


Figura 3: Descargar SDK Android

2.1.2.3. Instalación del plugin ADT de Eclipse

Usando Eclipse, este plugin nos permite utilizar el *sdk de Android*. Para instalarlo, regresar a la página de Android (http://developer.android.com) y escribimos en el buscador *"ADT Plugin"*, posteriormente vamos a "Installing the Eclipse Plugin", llegando a la página que se puede observar en la siguiente figura:



Installing the Eclipse Plugin

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT). This plugin provides a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to let you quickly set up new Android projects, build an app UI, debug your app, and export signed (or unsigned) app packages (APKs) for distribution.

If you need to install Eclipse, you can download it from eclipse.org/downloads/.

Note: If you prefer to work in a different IDE, you do not need to install Eclipse or ADT. Instead, you can directly use the SDK tools to build and debug your application.

Download the ADT Plugin

1. Start Eclipse, then select **Help** > **Install New Software**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*:

```
https://dl-ssl.google.com/android/eclipse/
```

4. Click **OK**.
   If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**.
   If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
8. When the installation completes, restart Eclipse.

Figura 4: Installing the Eclipse Plugin

Llegado a este punto, tenemos que copiar la URL que aparece en color rojo, para después copiarla en Eclipse.

A continuación debemos abrir Eclipse, y dirigirnos a la parte de *"ayuda"*, *"Install New Software…"*. Nos aparece la ventana que podemos ver en la siguiente figura:

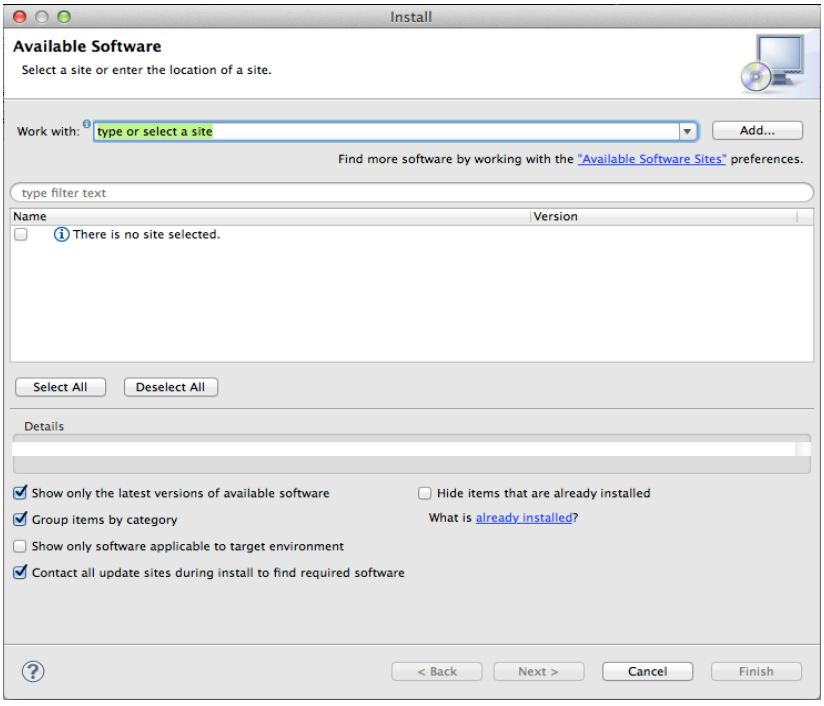


Figura 5: Install New Software

En dicha ventana, pulsamos en *"Add…"* y ahí es donde debemos poner:

- Name: *ADT Plugin*
- Location: *"La URL que hemos copiado anteriormente de la página de Android".*

En la siguiente figura aparece este paso, pero hay que indicar que aparece un icono en rojo con una cruz, indicando *"Duplicate location"*, ya que en mi ordenador ya está instalado. En el caso contrario, tendríamos que pulsar "OK" e instalar el paquete que nos saldría en la ventana de *"Name"*.

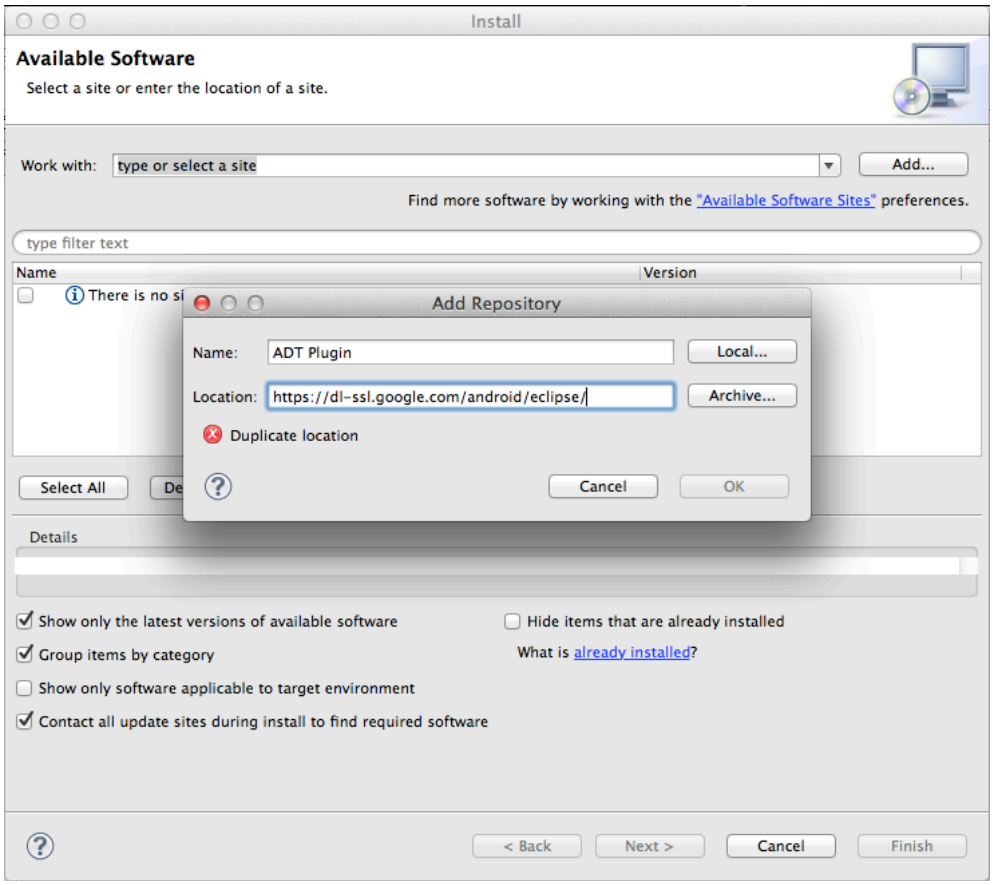


Figura 6: instalar ADT Plugin

Una vez instalado el *"ADT Plugin"*, nos pedirá reiniciar, y al abrir de nuevo necesitaremos introducir la ruta del *SDK de Android*, el cual ya lo hemos descargado anteriormente. Es conveniente guardar el *SDK* en un lugar separado del resto de descargas, donde tengamos pleno control de lo que pasa en esa carpeta, en concreto en *Mac OS X/Developer/*.

Posteriormente se nos pide la versión del *SDK* a instalar, la cual debemos descargar. Para ello vamos a la carpeta donde hemos situado el *SDK*, y nos dirigimos a */android-sdk-macosk/tolos/* para abrir el archivo *"android"*. Nos aparecerá la siguiente ventana:

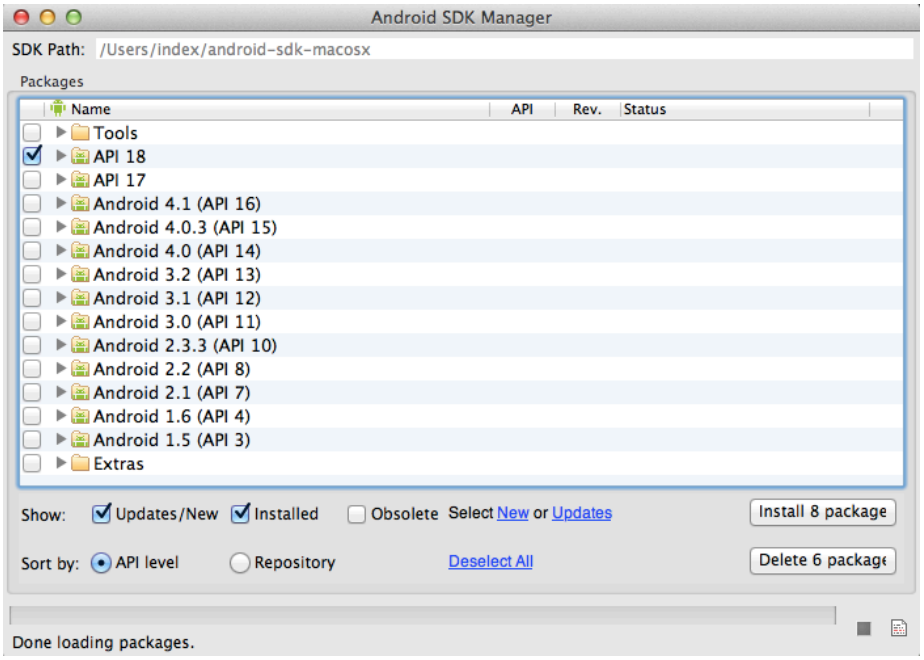


Figura 7: Android SDK Manager

Ahora sólo hay que elegir la versión deseada, esperar a que se instale y posteriormente es recomendable reiniciar Eclipse.

Finalmente ya podemos crear arrancar Eclipse y estará todo preparado para comenzar a crear nuestro proyecto y poder trabajar con el *SDK de Andorid*.

2.2. Tecnología Bluetooth

2.2.1. Introducción a las redes inalámbricas

Durante los últimos años han surgido con gran popularidad nuevas tecnologías inalámbricas como WIFI, WIMAX, Bluetooth, GSM, Infrarrojos, etc, siendo los dispositivos inalámbricos una de las grandes revoluciones tecnológicas de los últimos tiempos.

Las tecnologías inalámbricas, han conseguido esa popularidad gracias a la movilidad que permiten, llegando a cambiar la estructura y topología de las redes empresariales. Los dispositivos de almacenamiento de información que antes eran fijos ahora pueden ser portados y cambiar su conexión a distintas redes de una manera sencilla.

Es probable que en un futuro cercano todos los dispositivos que hoy utilizamos se unifiquen, pudiendo pasar a llamarse "Terminales Internet", en los que se reunirían funciones de teléfono, agenda, reproductor multimedia, ordenador personal, etc.

2.2.2. Historia Bluetooth

En 1994 la empresa sueca *Ericcson* inició un estudio para investigar la viabilidad du una interfaz vía radio, que presentara un bajo consumo y coste, con el fin de interconectar teléfonos móviles y otros accesorios sin necesidad de cables. El estudio partía de un largo proyecto que investigaba sobre unos multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance llamado **MClink**. Esto supuso un gran avance ya que su ventaja principal era el coste económico.

A principios de 1997, otros fabricantes despertaron su interés en dicho proyecto y pensaron que para que tuviera realmente éxito debían de incorporar esta tecnología en un gran número de dispositivos. Por este motivo a principios de 1998 surgió SIG (El grupo de interés especial de Bluetooth), compuesto por *Ericsson, IBM, Intel, Toshiba* y *Nokia*, los cuales formaron un consorcio y adoptaron Bluetooth como nombre para su especificación. Entre estas compañías se encontraban dos grupos líderes del mercado de las Telecomunicaciones, dos del mercado de los PCs portátiles y un líder de la fabricación de chips.

En 1999 Bluetooth lanzó su primera versión, Bluetooth v1.0, la cual ha ido evolucionando hasta la versión Bluetooth v4.0 actual:

- **Bluetooth v1.0 y v1.0b.** Versiones con muchos problemas, que mostraban la dirección del dispositivo en la transmisión (anonimato a nivel de protocolo imposible).

- **Bluetooth v1.1 (2002).** Se corrigieron muchos errores, se añadió soporte para canales no cifrados y un indicador de la señal recibida (RSSI).

- **Bluetooth v1.2 (2005).** Versión compatible con USB 1.1, nos encontramos mejoras como: una solución inalámbrica complementaria para coexistir, sin interferencias, la tecnología Bluetooth y Wi-Fi en el espectro de los 2.4 GHz. Utiliza la técnica *Adaptive Frequency Hopping (AFH)*, mediante la que se consigue una transmisión más eficiente y un cifrado más seguro. Además posee una mejora en la calidad de voz y una mayor rapidez de configuración de la comunicación con otros dispositivos Bluetooth dentro del rango de alcance, como pueden ser PDAs, ordenadores portátiles, ordenadores de escritorio, impresoras, etc.

- **Bluetooth v2.0 + EDR (2004).** Esta nueva actualización es compatible con la versión anterior. Su mejora principal es que presenta mayor velocidad de datos, *EDR "Enhanced Data Rate"*. La velocidad de transmisión es de 3Mbit/s, aunque en la práctica es de 2.1 Mbit/s. Puede proporcionar un menor consumo de energía a través de un ciclo de trabajo reducido.

- **Bluetooth v2.1 + EDR (2007).** Versión que también es compatible con la versión 1.2, fue adaptada por la SIG, ya mencionada anteriormente. Su principal incorporación fue *SSP*, *Secure Simple Pairing*, en la que se mejora la experiencia de emparejamiento de dispositivos Bluetooth. También se mejoró el consumo, siendo 5 veces menor y el filtrado de dispositivos antes de la conexión.

- **Bluetooth v3.0 + HS (2009).** Con esta evolución se soportan velocidades de transferencia de datos teórica de hasta 24 Mbit/s. La conexión Bluetooth nativa se utiliza para la negociación y el establecimiento, mientras que el tráfico de datos de alta velocidad se realiza mediante un enlace 802.11 (Wi-Fi). Su principal novedad es AMP *(Alternate MAC / PHY)* que permite el uso de alternativas MAC y PHY para el transporte de datos de perfil Bluetooth.

- **Bluetooth v4.0 (2010).** En esta versión se combina, Bluetooth clásico, Bluetooth de alta velocidad (se basa en *Wi-Fi*) y protocolos Bluetooth de bajo consumo (10 a 20 veces menos), ampliando la cantidad de aplicaciones, permitiendo la incorporación de

receptores y transmisores Bluetooth en dispositivos pequeños como relojes, reproductores portátiles, instrumental médico, etc. Se mejora la seguridad con AES-128. Bluetooth v4.0.

2.2.3. Arquitectura Hardware

En cuanto a la estructura hardware por la que está compuesta un dispositivo Bluetooth, tenemos que tener en cuenta dos partes: un *dispositivo de radio* y un *controlador digital*.

- **Dispositivo de radio:** encargado de modular y transmitir la señal.
- **Controlador digital**: compuesto por una CPU, un procesador de señales digitales y de las interfaces con el dispositivo anfitrión, llamado Link Controller.

El Link Controller, tiene varias funciones como, procesamiento de la banda base y del manejo de los protocolos *ARQ (Automatic Repeat reQuest)* y *FEC (Forward Error Correction)* de la capa física, funciones de transferencia (síncronas o asíncronas), el cifrado de datos y la codificación de audio.

La CPU se encarga de las instrucciones relacionadas con Bluetooth en el dispositivo anfitrión, simplificando así su operación. Esto se consigue mediante Link Manager, un software que corre sobre la CPU encargado de comunicarse con otros dispositivos por medio del protocolo *LMP (Link Manager Protocol)*.

2.2.4. Tecnología Bluetooth

2.2.4.1 Protocolos

En comunicaciones, un protocolo es un conjunto de reglas y normas que utilizan los equipos, para gestionar sus diálogos en los intercambios de información. Lo que implica que dos equipos distintos, de diferentes fabricantes, puedan comunicarse sin ningún problema siempre y cuando usen el mismo protocolo de comunicaciones.

Centrándonos en la tecnología que nos ocupa, Bluetooth, encontramos una pila de protocolos como la que se muestra en la siguiente imagen:

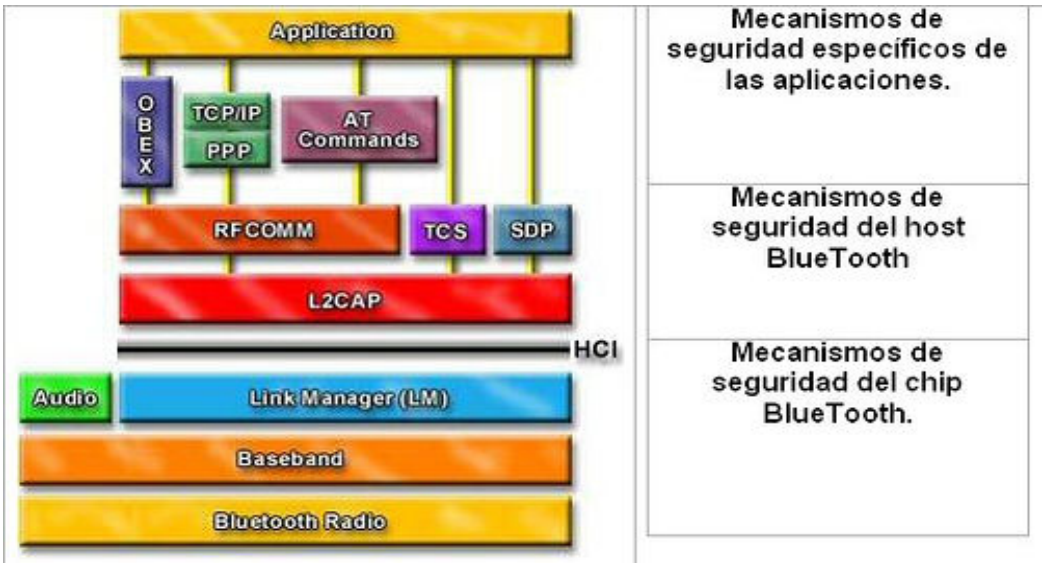


Figura 8: Pila de protocolos Bluetooth.

Los protocolos que nos encontramos, empezando de forma ascendente, son:

- **Bluetooth Radio:** cuya tarea principal es el funcionamiento de radio Bluetooth, 2.4GHz.

- **Link Manager Protocol (LMP):** este protocolo, ya mencionado anteriormente, es el encargado de la conectividad entre los dos dispositivos Bluetooth, calidad de servicio, etc. Cabe destacar de este protocolo, que es el encargado de que los dispositivos se comuniquen entre las 79 frecuencias diferentes posibles. Cuando dos dispositivos quieren comunicarse por radio, primero se comunican a través de una frecuencia determinada *(control cannel)*, mediante la cual se ponen de acuerdo en qué orden van a utilizar esas 79 frecuencias. La secuencia que se utiliza se genera de forma aleatoria. Esta técnica se conoce como *FHSS (Frequency-hopping spread spectrum)*. De esta forma se obtiene una mayor seguridad frente ataques externos que pretendan acceder a la conexión.

- **L2CAP:** protocolo que se encarga de los puertos que hay que utilizar para mandar y recibir datos. Dispone de 32767 puertos.

- **RFCOMM:** se encarga de emular las conexiones por puerto series, dispone de 30 puertos y sólo puede gestionar 6 conexiones a la vez.

- **Host Controller Interface (HCI):** es la forma en la que el ordenador o el móvil se conectan al dispositivo Bluetooth, en concreto a aquellos que se conectan por USB a los computadores para poder disponer de esta tecnología. Estos dispositivos se encargan del Link Manager, pudiendo

conseguir que dicho ordenador que no dispone de Bluetooth incorporado, pueda utilizar esta tecnología inalámbrica.

El resto de mecanismos de seguridad son específicos de cada aplicación, con lo cual no los trataremos.

2.2.4.2. Usos

Esta tecnología ha ido cada vez ampliando el rango de productos que la incorporan, debido a su bajo consumo, independencia de conexiones físicas por cables, innecesaria colocación lineal entre ambos dispositivos, correcto funcionamiento con obstáculos intermedios, etc. Estos productos pueden ser tales como impresoras, teléfonos, módems y auriculares.

El avance tecnológico ha logrado mejorar la calidad de vida del usuario, consiguiendo acciones de manera más fácil, como auriculares sin cables, conexiones de dispositivos en distintas habitaciones, y en otras ocasiones desde un punto de vista más seguro para el individuo, por ejemplo dispositivos Bluetooth para el coche.

CAPÍTULO 3

DESARROLLO E IMPLEMENTACIÓN

En este capítulo vamos a ver los aspectos más relevantes de lo que ha sido el desarrollo de este proyecto. En primer lugar, daremos una visión más detallada de nuestro entorno de trabajo utilizado (Eclipse) para poder detallar las partes más importantes de la implementación de nuestra aplicación.

3.1. Creación de un proyecto en Eclipse

Una vez instalado Eclipse, el SDK y el ADT, ya tenemos todo lo necesario para poder desarrollar aplicaciones en Android. A continuación procedemos a abrir Eclipse y una vez abierto pulsaremos en "*File, New, Android Applicaiton Project*". A continuación nos aparecerá la siguiente ventana:

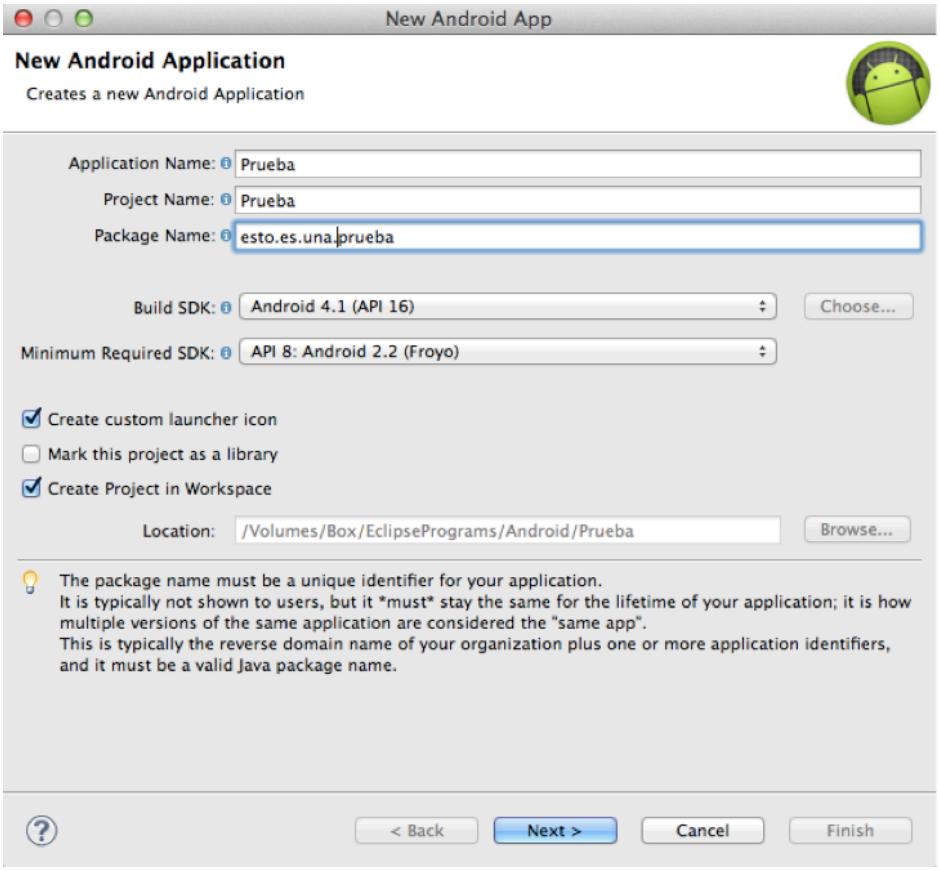


Figura 9: creando un nuevo proyecto Android.

A continuación se detallarán los campos que aparecen en la ventana emergente que se nos presenta:

- *Application Name:* nombre de la aplicación.
- *Project Name:* nombre del proyecto Android.
- *Package Name:* nombre del paquete donde estará ubicado el proyecto.
- *Build SDK:* versión en la que se creará nuestro proyecto Android.

- *Minimum Required SDK:* versión mínima de Android que soportará nuestra aplicación.
- *Location*: directorio en el que queremos que se guarde el proyecto.

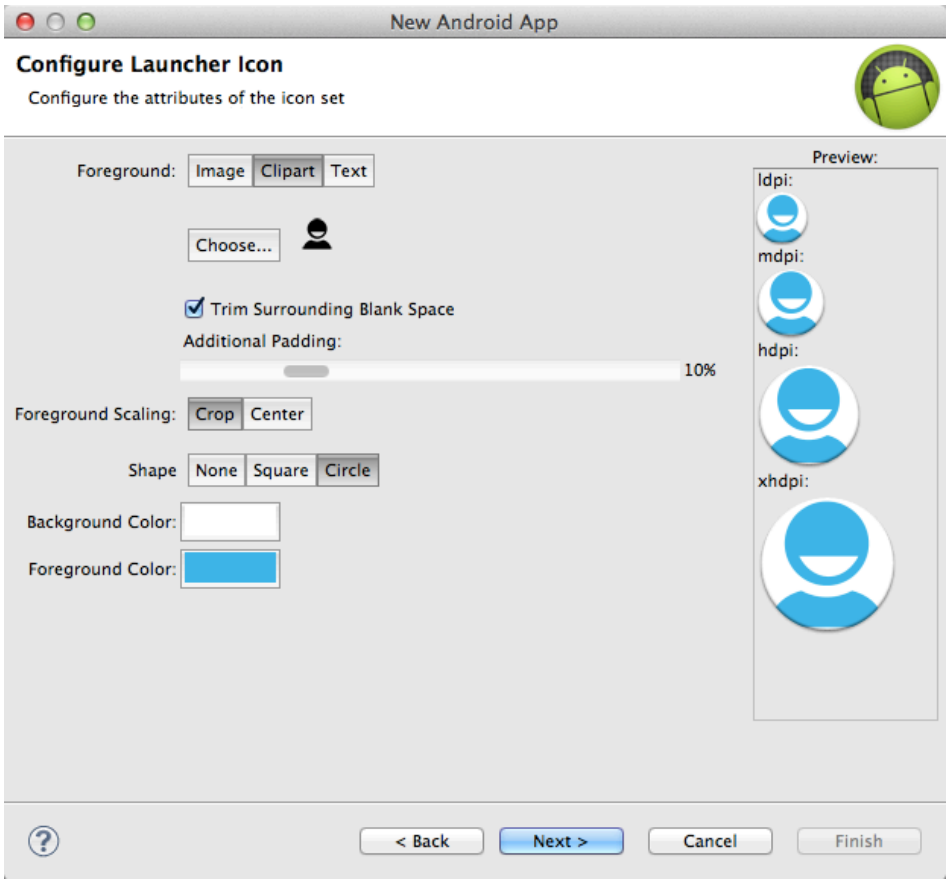


Figura 10: creando un nuevo proyecto Android.

En esta figura, se nos da la opción de elegir el icono para la aplicación: colores, forma, etc.

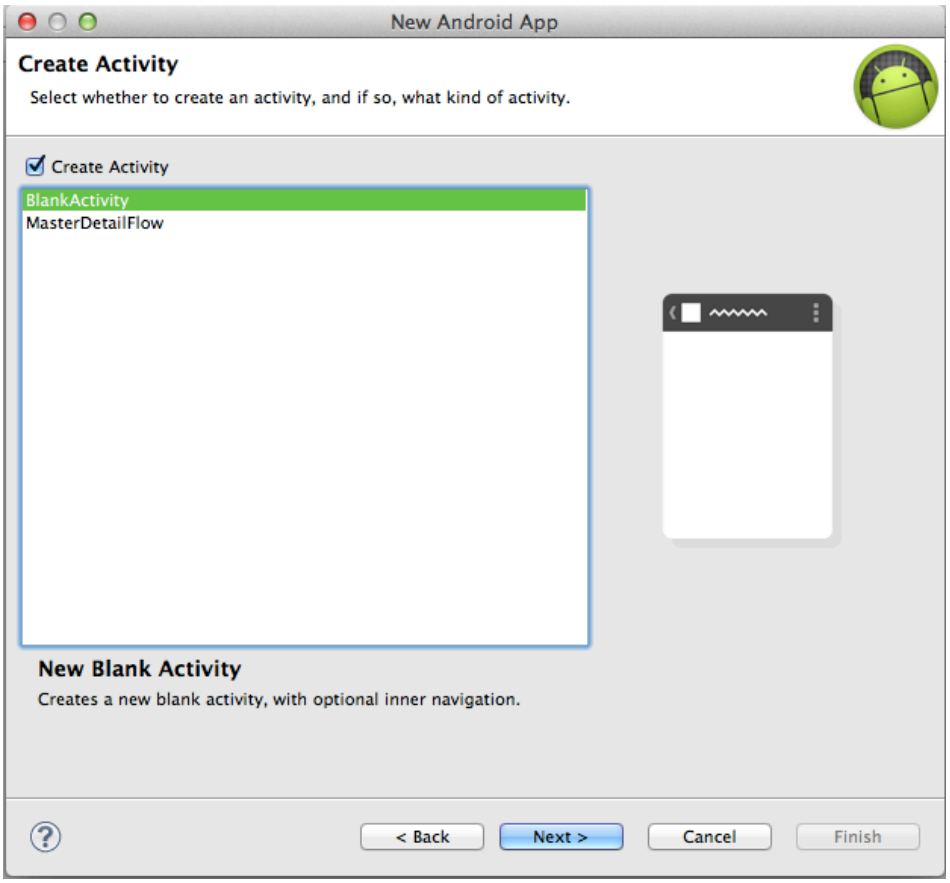


Figura 12: creando un nuevo proyecto Android.

En esta nueva ventana, seleccionamos el *"Blank Activity"*, teniendo dos opciones disponibles.

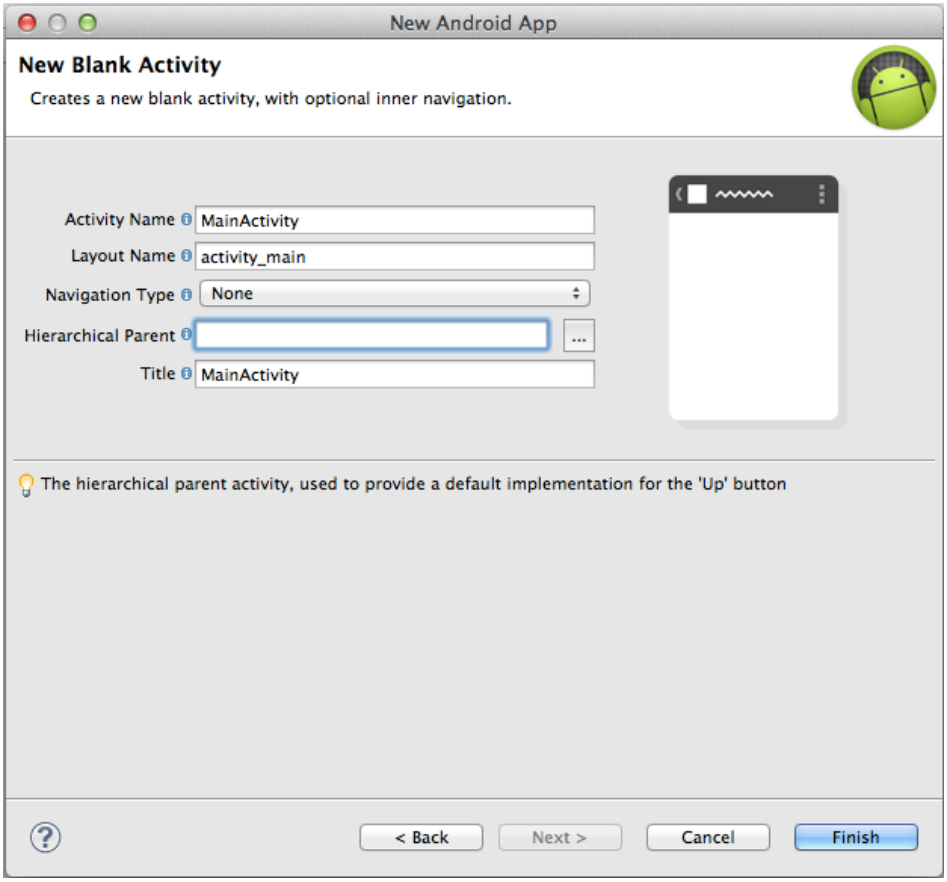


Figura 13: creando un nuevo proyecto Android.

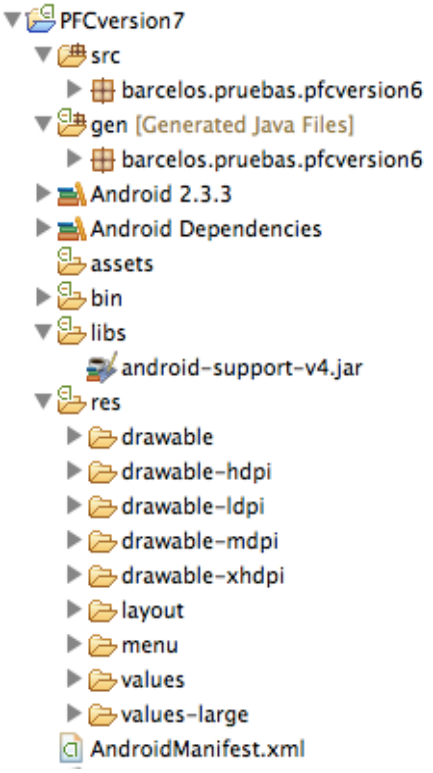
Y por último, deberemos de darle un nombre a la actividad principal de la aplicación y al fichero XML, el cual va a contener la vista de nuestra primera pantalla de la aplicación.

3.2. Carpetas y Archivos generales

Una vez que creamos un proyecto Android, aparecen una lista de carpetas y archivos comunes, cada uno de estos con una finalidad propia y específica. A continuación explicaremos brevemente sus funciones y utilidades:

- **Carpeta "src":** esta carpeta contendrá todos los paquetes que pueda contener nuestra aplicación y en cada paquete estarán los diferentes archivos JAVA implementados.

- **Carpeta "gen":** esta carpeta contendrá los archivos R.java y BuildConfig.java. Estos ficheros son generados automáticamente por Eclipse y tienen una gran importancia dentro de una aplicación Android. En el archivo R.java se establece la relación entre la interfaz gráfica y su correspondiente implementación en java a través de referencias. Ninguno de estos dos archivos puede ser modificado por el programador.



- **Carpeta "assets":** este directorio está vacío por defecto. En él se podrá incluir cualquier tipo de fichero externo para que sea utilizado en cualquier momento de la aplicación.

- **Carpeta "bin":** Esta carpeta se genera automáticamente, y la utiliza el compilador para preparar los archivos para el empaquetado final en forma de APK.

- **Carpeta "res":** esta carpeta contiene ocho directorios. Cinco de ellos llamados *"**drawable**", "**drawable-hdpi**", "**drawable-ldpi**", "**drawable-mdpi**"* y "***drawable-xhdpi***", que contendrán las imágenes usadas en nuestra aplicación. En la carpeta "**layout**" y "**menu**" se encuentran los archivos *".xml"* que definen las interfaces gráficas y menús de cada pantalla. Y por último, la carpeta "**values**" que contiene el archivo string.xml, dimens.xml o styles.xml donde se podrán definir constantes dependiendo de qué fichero se trate.

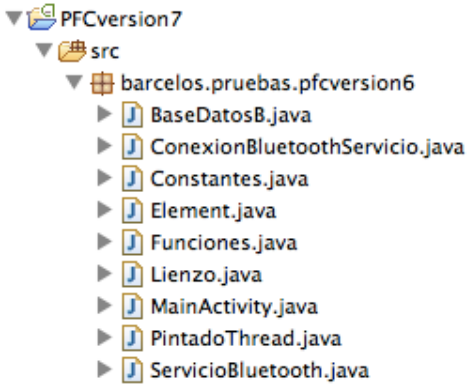
- **Archivo "AndroidManifest.xml":** Este archivo es muy importante e imprescindible en cualquier aplicación Android. Está escrito en XML y describe los componentes de los que constara nuestra aplicación. Esta descripción contiene aspectos como las clases que los implementan, sus requisitos, los datos que pueden manejar o cuando deben ser ejecutados. También se definen los permisos que el desarrollador solicita para utilizar en su aplicación, como pueden ser acceso a internet, utilizar Bluetooth, tarjeta SD, etc.

- **Archivo "project.properties":** contiene configuraciones básicas como la versión de la plataforma destino. Se configura automáticamente y no debe ser modificado por el programador.

3.3. Carpetas y archivos propios de la aplicación presente

En el apartado anterior, se ha explicado los tipos de carpetas y archivos que siempre aparecen al crear, en Eclipse, un proyecto Android. Ahora nos centraremos más en esos directorios pero orientados a la aplicación del proyecto. Para ello iremos viendo en cada carpeta, qué clases y archivos se han añadido.

- **Carpeta "src":** En esta carpeta se han añadido las siguientes clases JAVA:
  - ***Constantes.java***
  - ***Element.java***
  - ***BaseDatosB.java***
  - ***Funciones.java***
  - ***MainActivity.java***
  - ***ServicioBluetooth.java***
  - ***ConexionBluetoothServicio.java***
  - ***Lienzo.java***
  - ***PintadoThread.java***



En el siguiente apartado explicaremos el funcionamiento de cada una de ellas.

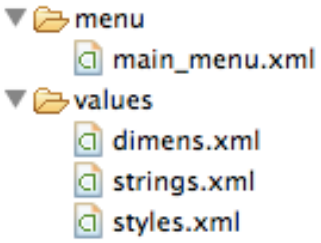
- **Carpeta "menu":** En esta carpeta encontramos el código que se ha implementado para la opción menú, pudiendo así empezar a buscar dispositivos Bluetooth o parar la búsqueda. Para ello, se ha creado el siguiente código XML:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/menu_inicioBlueth"
          android:icon="@android:drawable/ic_menu_search"
          android:title="@string/menu_iniciarBlueth" />
    <item android:id="@+id/menu_stopBlueth"
          android:icon="@android:drawable/ic_menu_mylocation"
                  android:title="@string/menu_stopBlueth" />
</menu>
```

Código 1: archivo para el menú.

En este código se han incorporado dos *"ítem"* que hacen referencia a la necesidad de dos botones, uno para empezar el escaneo y otro para terminar la búsqueda. De esta forma conseguimos tener pleno control de la interfaz Bluetooth propia del terminal, ya que podemos controlar su estado. Dichos botones tienen un icono relacionado con la acción que llevarán a cabo.

En este punto tenemos que nombrar la carpeta *"values"*, ya que es utilizada en la parte del código anterior, por ejemplo en:
"`android:title="@string/menu_iniciarBlueth"`".



Aquí se hace referencia a una constante declarada en el archivo "string.xml", con el id *"menú_iniciarBlueth"*. La cual es declarada mediante el código:

```
<resources>
    <string name="app_name">Cliente</string>
    <string name="title_activity_main">Cliente</string>
    <string name="menu_iniciarBlueth">Scan Bluetooth</string>
    <string name="menu_stopBlueth">STOP</string>
</resources>
```

Código 2: string.xml

Cabe destacar que Android y Eclipse nos permiten crear este tipo de constantes de dos maneras distintas: mediante código XML, como el mostrado en el recuadro anterior, o mediante una interfaz gráfica, mostrada a continuación:

Figura 14: interfaz gráfica que nos ofrece Eclipse para crear archivos, en este caso es el archivo String.xml, donde se almacenan las cadenas de caracteres utilizadas en la aplicación.

- **Archivo AndroidManifest.xml:** En este archivo de gran importancia, ya explicado anteriormente, se incorpora:

- ***<manifest>:*** tag raíz. En él se declara el número de versión de desarrollo, el número de versión de nuestra aplicación y el paquete raíz que la contiene.

- ***<uses - sdk>:*** tag utilizado para determinar las distintas versiones Android en las que nuestra aplicación podrá funcionar. Mediante el atributo android:minSdkVersion establecemos a partir de qué versión de Android nuestra aplicación podrá correr.

- ***<uses-permissions>:*** mediante este tag especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse. Permisos que se requieren: utilización del Bluetooth y de la tarjeta SD. Con estos permisos, al instalar la aplicación en el terminal, se le informa al usuario que se van a utilizar los recursos indicados, dándole la libertad de poder aprobar o denegar las condiciones.

- ***<application>:*** *tag* que contiene todas las *Activities, Services, Providers, Receivers* y las bibliotecas que se usan en nuestra aplicación. Servicios en segundo plano que implementa: "*ServicioBluetooth*". En esta parte tenemos que introducir todos los servicios que se llevarán a cabo en nuestra aplicación. En caso contrario la aplicación crearía una excepción.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="barcelos.pruebas.pfcversion6"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-feature android:name="android.hardware.bluetooth" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait"
            android:label="@string/title_activity_main" >
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:enabled="true"
android:name=".ServicioBluetooth"/>
    </application>

</manifest>
```

Código 3: AndroidManifest.xml

3.4. Implementación y Funcionamiento de Clases:

En este apartado, se tratará de explicar los detalles más importantes de las clases utilizadas para la creación de la aplicación. No se copiará el código completo de la éstas, ya que se considera innecesario, simplemente aquellos fragmentos que se crean convenientes. En el anexo se a reflejado el Diagrama de Clases UML para su mayor comprensión. Las clases implementadas son las ya citadas en puntos anteriores:

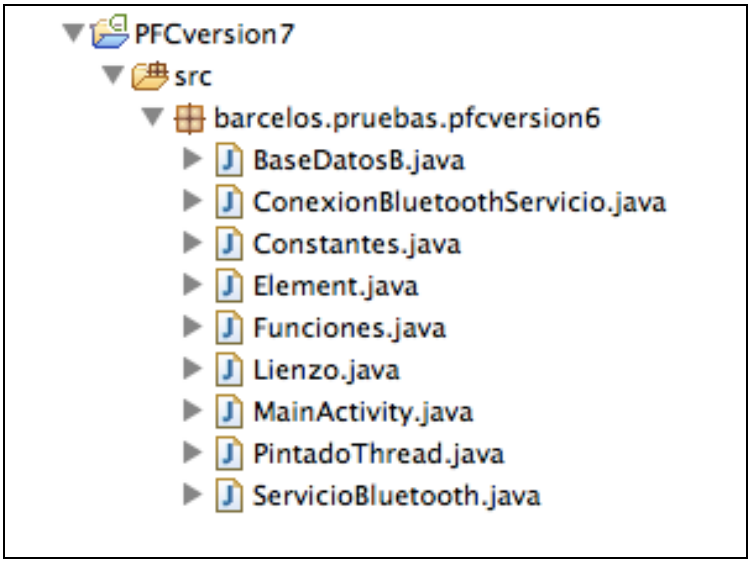


Figura 15: Clases implementadas

- ***Constantes.java***

Esta clase se ha utilizado para contener las constantes utilizadas en la creación de la Base de Datos. Simplemente almacena el nombre de la tabla donde se almacenan todos los datos, "dbB", y las columnas que ésta posee: *SSID, MAC, ESTADO, RSSI, X, Y, DATOS* y *ULTIMOS_D*.

- ***Element.java***

Esta clase trata de hacer referencia a cada dispositivo Bluetooth externo, es decir, todos aquellos terminales que se encuentren mediante la interfaz Bluetooth, se almacenaran como objetos de tipo "*Element*" en las arrays convenientes, para así, trabajar con ellos posteriormente. Las variables de instancia que presenta esta clase son:

```
private String name;
private String mac;
private boolean encontrado;
private int X;
private int Y;
```

Código 4: Variables de instancia clase Element.

Las variables corresponden con:

- ***name:*** contendrá el SSID del dispositivo encontrado.
- ***mac:*** almacenará la dirección MAC del terminal detectado.
- ***encontrado:*** si es "false" nos indica que aún no se a encontrado el dispositivo, por el contrario, si es "true" sabemos que sí se a encontrado y con ello que está a nuestro alcance.
- ***X e Y:*** nos indica la coordenada X e Y en el plano, estos valores se conocen con anterioridad.

Esta clase, implementa los respectivos métodos para la modificación y obtención del valor de las distintas variables (métodos ***get*** y ***set***).

- ***BaseDatosB.java***

Esta clase se ha creado con el propósito de almacenar la distinta información de los dispositivos que encontramos al escanear con la interfaz Bluetooth, siempre y cuando sean los que deseemos. Junto a la información de cada dispositivo remoto, se almacenan los datos que éstos nos envían, pudiendo acceder a ellos aunque la aplicación se cierre. Los campos que componen la tabla *"dbB"* de la base de datos *"DB_Bluetooth_PFC"* son:

- *SSID:* nombre del dispositivo remoto.
- *MAC:* dirección MAC del dispositivo remoto.
- *ESTADO*: estado (pendiente, encontrado) del terminal remoto.

- *X, Y:* coordenadas en el plano en el que se encuentra.
- *DATOS:* datos que nos manda dicho terminal.
- *ULTIMOS_D:* último paquete de datos que hemos recibido.

- ***Funciones.java***

  En esta clase se han definido he implementado los métodos o funciones que más veces son utilizadas en la aplicación, consiguiendo así tenerlo todo más concentrado y sin necesidad de implementar el código de cada método por separado. Las funciones que presenta son:

  - **insertarBD**(ContentValues valores, SQLiteDatabase db): este método se encarga de almacenar en la base de datos, los dispositivos que se crean necesarios. Se debe destacar la función de un objeto tipo *"Cursor"* (código 5), encargado de situarse en cada una de las filas de la tabla de la base de datos, pudiendo así acceder a cada uno de los campos. Para almacenar el resultado se utilizan los métodos, "*insertOrThrow"* y *"update"* (código 8), mediante los cuales podemos insertar o actualizar los valores que se deseen. Para ello se utilizan diversos bucles que comparan las direcciones MAC del dispositivo almacenado y del que se quiere guardar, si son iguales, se actualiza y en caso contrario, se inserta.

```
Cursor cursor = db.query(NOMBRE_TABLA, new String[] {_ID,
MAC}, null, null, null, null, null);
```

Código 5: obtener una referencia para poder recorrer la tabla de la base de datos, pudiendo acceder los campos del identificador y de la MAC.

```
cursor.moveToFirst()
```

Código 6: nos situamos en la primera fila de la tabla, si existe nos devuelve "true", en caso contrario "false".

```
cursor.moveToNext()
```

Código 7: nos situamos en la siguiente fila de la tabla, si existe nos devuelve "true", en caso contrario "false".

Código 8: insertamos o actualizamos en la tabla la información almacenada en "valores".

```
db.insertOrThrow(NOMBRE_TABLA, null, valores)
db.update(NOMBRE_TABLA,valores, _ID + "=" +
cursor.getString(0), null)
```

- **guardarDatosBD**(String datos_almacenar, String MAC_remota, SQLiteDatabase db): mediante este método, almacenas en la tabla de la Base de Datos, la información que hemos recibido de los dispositivos externos. Con el fin de poder acceder a dichos datos posteriormente con el propósito que se desee. El código implementado es parecido al del método *"insertarBD"*, pero la diferencia reside en que el objeto "Cursor", ya mencionado, debe contener referencias a los "DATOS" y "ULTIMOS_D", no como en el caso anterior que bastaba con el identificado de la fila y la MAC del dispositivo.

```
Cursor cursor = db.query(NOMBRE_TABLA, new String[] { _ID,
MAC, DATOS, ULTIMOS_D}, null, null, null, null, null);
```

Código 9: cursor para recorrer la tabla de la base de datos, pudiendo acceder a los campos del identificador, la MAC, los datos totales y los últimos datos recibidos.

- **leerBDexterna**(SQLiteDatabase db): este método es el primero que se ejecuta. Es el encargado de copiar, en la base de datos interna de la aplicación, los dispositivos con los que vamos a trabajar, los cuales están almacenados en una base de datos externa, situada en la micro SD del terminal.
  El código a destacar es el siguiente:

```
        File rutaSD = Environment.getExternalStorageDirectory();
dbExterna=SQLiteDatabase.openDatabase(rutaSD.getAbsolutePath()+"
/basedatos/bluetoothBD", null, SQLiteDatabase.OPEN_READWRITE);
```

Código 10: de esta forma creamos una referencia a la base de datos almacenada en la microSD, para posteriormente trabajar con ella y almacenar su información en la base de datos propia de la aplicación.

- **buscarMACarray**(ArrayList<Element> array, String MAC): este método es muy simple, ya que sólo se dedica a buscar una dirección MAC, en una *array*, devolviendo *"true"*, en caso de que exista, o *"false"* en caso contrario.

- **getPosicion**(String MACfind, SQLiteDatabase db): este método es utilizado para obtener las coordenadas de un dispositivo que esté almacenado en la base de datos. También hace uso de una referencia *"Cursor"*, ya que recorre todas la filas comparando la dirección MAC a buscar, para posteriormente devolver dichas coordenadas.

- ***MainActivity.java***

Es la clase inicial, con la que arranca la aplicación, puesto que así se ha indicado en el archivo "AndroidManifest.xml", ya explicado anteriormente. En ella:

- se configura la interfaz para no mostrar el nombre de la aplicación en la barra superior.
- Se elimina la visualización de la barra de notificaciones.
- Se configura para que la pantalla no se apague hasta que no se salga de la aplicación.
- Se carga el mapa *indoor*.
- Se configura el menú, el cual constará de dos botones: iniciar escaneo y parar escaneo de dispositivos. Si pulsamos el primero, arranca el servicio encargado de realizar la búsqueda de terminales remotos, y si pulsamos el segundo, finaliza este servicio, lo que conlleva a eliminar las conexiones con cualquier dispositivo.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case R.id.menu_inicioBlueth:
        if(ServicioBluetooth.getActivo() != true){
           getApplicationContext().startService(new
Intent(getApplicationContext(), ServicioBluetooth.class));
                ServicioBluetooth.setActivo(true);
        }
        return true;

       case R.id.menu_stopBlueth:
          if(ServicioBluetooth.getActivo()){
      ServicioBluetooth.setActivo(false);
      while(ConexionBluetoothServicio.getState()   != 0){
       Toast.makeText(this, "IMPORTANT: wait closing
socket...", Toast.LENGTH_SHORT).show();
           ConexionBluetoothServicio.stop();
          }
         getApplicationContext().stopService(new
Intent(getApplicationContext(), ServicioBluetooth.class));
        }
```

```
        return true;
      }
       return false;
}
```

Código 11: implementación para mostrar los botones de menú, iniciar el escaneo o pararlo.

- Se configura la pantalla, mediante un método "onTouchEvent", para detectar si es pulsada, ya que cada vez que se detecte dicha pulsación, comienza el escaneo de dispositivos y si se presiona de nuevo, se detiene. Tiene la misma finalidad que los botones del menú.

```
public boolean onTouchEvent(MotionEvent event){
   switch (event.getAction()){
      case MotionEvent.ACTION_DOWN:
          if(ServicioBluetooth.getActivo() != true){
            startService(new Intent(getApplicationContext(),
   ServicioBluetooth.class));
            ServicioBluetooth.setActivo(true);
          }else{
            ServicioBluetooth.setActivo(false);
            while(ConexionBluetoothServicio.getState() != 0){
           Toast.makeText(this, "IMPORTANT: wait closing
   socket...", Toast.LENGTH_SHORT).show();
            ConexionBluetoothServicio.stop();
          }
           getApplicationContext().stopService(new
   Intent(getApplicationContext(), ServicioBluetooth.class));
         }
      break;
       }
return true;
}
```

Código 12: implementación para programar la pulsación en pantalla, cada vez que sea presionada, comenzará el escaneo o finalizará.

- ***ServicioBluetooth.java***

Esta clase, es la encargada de crear el servicio para la búsqueda de terminales remotos mediante la interfaz Bluetooth, además de iniciar las conexiones con éstos.

Pero ¿qué es un servicio en Android?. Consiste en añadir un nuevo componente a tu aplicación para llevar a cabo algún tipo de acción que se ejecute en segundo plano, es decir, que no requiera una interacción directa con el usuario, pero que queramos que permanezca activo aunque el usuario cambie de actividad.

Partes en las que está formada la clase *SercicioBluetooth.jaca*:

- *Búsqueda de dispositivos:* lo primero que se realizará será un escaneo radio intentando detectar cualquier dispositivo Bluetooth. Cada vez que se detecta un dispositivo se comprueba si es un dispositivo que esté ya emparejado, y si es uno de los terminales que tenemos almacenados en nuestra Base de Datos externa. Si cumple ambas condiciones, lo almacenamos en la base de datos interna y en una *array* especial, "aEncontrados", la cual pertenece a la calse "Lienzo". Esta *array*, almacenará todos los dispositivos que estén a nuestro alcance, es decir, aquellos que debemos mostrar en el mapa como encontrados y a los que posteriormente debemos conectarnos.
  Todo el procedimiento de detección de dispositivos se implementa mediante un "IntentFilter" y un "BroadcastReceiver", cuya función es poder enviar mensajes entre actividades y capturar eventos, tales como: detectar dispositivos, notificaciones, etc. En nuestro caso se configura para que detecte dos acciones, detectar dispositivos Bluetooth y finalizar el escaneo radio (código 13).

```
          IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(mReceiver, filter);
```

```
     filter = new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        registerReceiver(mReceiver, filter);
```

Código 13: configuración del objeto "BroadcastReceiver", para que pueda detectar cuando se ha encontrado un dispositvo Bluetooth y cuando se termina de descubrir dispositivos (fin del escaneo).

```
final BroadcastReceiver mReceiver = new
BroadcastReceiver() { //resto de código }
```

Código 14: aquí se declara el objeto encargado de realizar las dos acciones anteriormente mencionadas (detectar dispositivos y detectar fin del escaneo).

- *Conexión con los dispositivos detectados:* una vez que hemos terminado de detectar terminales y de almacenarlos en la *array* de dispositivos encontrados y en la base de datos, procederemos a conectarnos con ellos.

  Nota: en este punto cabe destacar una pequeña limitación impuesta por la interfaz Bluetooth, ya que no es posible realizar de manera paralela el escaneo de dispositivos y la conexión con éstos. Es decir, debe de realizarse por separado, de lo contrario no se garantiza el correcto funcionamiento. A pesar de ello, se han realizado en versiones anteriores de este proyecto, diversas pruebas para ver la gravedad de los errores que se pueden producir. Tras varias comprobaciones, se observó que no existía una conexión fiable entre los dispositivos, obteniendo datos erróneos e incompletos en recepción. Por este motivo hay que dividir el servicio en estas dos partes: búsqueda y conexión.

  La conexión con los dispositivos se ha realizado por medio de un hilo *(Thread)*. Éste es el encargado de recorrer la *array* "encontrados", que es una copia de la *array* "aEncontrados" de la clase "Lienzo", y de realizar la conexión con cada uno de los dispositivos. Para ello se hace uso de la clase *ConexiónBluetoothServicio.java*, ya que se trabaja con un objeto de ese tipo.

- ***ConexionBluetoothServicio.java***

  Llegados a este punto se explicará qué es un hilo o *thread*. Cuando utilizamos un *thread* estamos realizando una tarea en segundo plano, con lo cual, lo que se realice en éste deja de formar parte del hilo principal de la aplicación. Para poder transferir información de los hilos secundarios al principal no se puede realizar directamente, necesitamos utilizar un *Handler*, el cual hace de "puente" entre ambos hilos.

Figura 16: esquema funcionamiento de los *"Threads"*.

Por otra parte, esta clase se encarga exclusivamente de la conexión con el dispositivo externo. Consta de dos hilos *(Threads)*, que hacen que nuestro terminal trabaje como un cliente en la comunicación, intentando acceder a los servidores que son los que están a espera de la conexión. Un hilo se encarga de la conexión con el terminal remoto, y el otro del intercambio de mensajes una vez que se encuentra conectado.

- Thread ConnectThread: mediante este hilo creamos el socket para comunicarnos con el otro terminal. Para ello debemos crear una referencia *"BluetoothSocket"* y crear una comunicación según una cadena de caracteres, llamada *"UUID"*. Dicha cadena es como un identificador, el cual debe ser conocido por el servidor al que quiere ser conectado .

```
public static final UUID MY_UUID =
UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");

//resto de código
BluetoothSocket tmp =
device.createInsecureRfcommSocketToServiceRecord(MY_UUID);
//resto de código

mmSocket = tmp;
mmSocket.connect();
```

Código 15: parte del código del hilo ConnectThread; identificadior UUID, creación del socket con dicho número y método connect().

Si la conexión con el dispositivo no se lleva a cabo, es porque el dispositivo no se encuentra a nuestro alance. En este punto salta una excepción, la cual es tratada para poder seguir el curso de la aplicación; se cierra el socket creado y se continúa con la conexión del resto de dispositivos. Además eliminamos de la *array* de dispositivos encontrados el terminal que a hecho saltar la excepción, ya que no está a nuestro alcance.

```
try {
     mmSocket.connect();
} catch (IOException e1) {
     Log.e("ConexionBluetoothSercicio", "  - Connect no: " +
mmSocket.getRemoteDevice().getName(), e1);
     setState(STATE_NONE);
     //borramos el dispositivo de la array de "encontrados"
     Lienzo.deleteDevice(mmDevice.getAddress(), 0);
     cancel();
     //pasamos a conectarnos con el siguiente
     if(ServicioBluetooth.getActivo()){
        ServicioBluetooth.nextConnect();
     }else{
        Lienzo.clearArrayE();
     }
     return;
}
```

```
public void cancel() {
    try {
   mmSocket.close();
   } catch (IOException e) {
   Log.e("ConexionBluetoothSercicio", "close() of connect,
socket failed", e);
   }
}
```

Código 16: parte del código del hilo ConnectThread; se intenta conectar con el dispositivo remoto, en caso de no ser posible, se trata dicho terminal como no encontrado.

- Thread ConnectedThread: una vez que ya estamos conectados con el dispositivo Bluetooth remoto, comienza a ejecutarse este hilo, cuya función es utilizar una referencia *"InputStream"* (Código 17) , para leer continuamente los datos del socket de comunicación Bluetooth *(BluetoothSocket),* creado en el hilo anterior *(ConnectThread)*.

```
private final InputStream mmInStream;
while (continuar) {
       try {
           bytes = mmInStream.read(buffer);
           String readMessage = new String(buffer, 0, bytes);
           ServicioBluetooth.saveRead(readMessage);

           if(readMessage.indexOf("/r")!=-1){
                   continuar =false;
           }
        } catch (IOException e) {
           Log.e("ConexionBluetoothSercicio", "ConnectedThread:
disconnected", e);
            if(ServicioBluetooth.getActivo()){
                    ServicioBluetooth.nextConnect();
            }
            synchronized (ConexionBluetoothServicio.this) {
                txrx= null;
            }
           break;
       }
}
```

Código 17: parte del código encargado de la lectura de los datos que el dispositivo remoto manda. Estos datos se almacenan en un *String* y se llama a la función *"saveRead"*, implementada en el servicio *"ServicioBluetooth",* para almacenar los datos en la Base de Datos.

> **Nota:** cabe destacar que para realizar las pruebas se propuso terminar la conexión con el dispositivo remoto, cuando éste mandara la cadena *"/r".*

Las siguientes dos Clases, están orientadas a la representación del mapa en pantalla y a la posición en éste de nuestra localización, mediante la técnica de triangulación. En este proyecto, como se podrá observar, no se han implementado códigos ".xml" para la representación de la interfaz gráfica, ya que resulta una tarea costosa, debido a que se producen continuas variaciones en pantalla. De esta forma se optó por utilizar la técnica que se lleva a cabo en juegos, la subclase *SurfaceView*.

SurfaceView es una subclase de la clase *View* optimizada para el rendimiento de operaciones gráficas. Una instancia de esta clase debe ser manejada desde un hilo de ejecución secundario (en este caso PintadoThread), de modo que las operaciones que efectúan no tengan que esperar al proceso de la jerarquía de *Views* que realiza el sistema periódicamente.

- ***Lienzo.java***

Esta clase extiende de *SurfaceView*, ya mencionada. Se encarga de mostrar por pantalla cada *Bitmap* necesario, estos son:

- Mapa *indoor*
- Dispositivos pendientes de encontrar
- Dispositivos encontrados
- Posición actual

Cada *Bitmap* es un conjunto de bits que representan una imagen a mostrar. Esta clase posee dos *arrays* , una que almacena los terminales que aún no se han encontrado (aPendientes) y otra para los terminales ya encontrados (aEncontrados). De esta forma, para mostrar por pantalla cada icono que hace referencia a cada dispositivo, sólo hay que recorrer continuamente estas dos *arrays,* las cuales siempre estarán actualizadas. Esto es posible porque cada vez que se detecta un dispositivo encontrado, se actualiza la *array* correspondiente, y cada vez que se sale del alcance de algún terminal, también se actualiza la *array* necesaria.

```
@Override
protected void onDraw(Canvas canvas) {
	super.onDraw(canvas);
	canvas.save();
	canvas.drawBitmap(plano, 0, 0, null);
	if(aPendientes.size() > 0 ){
		for(int i = 0 ; i < aPendientes.size() ; i++){
			x = aPendientes.get(i).getX();
			y = aPendientes.get(i).getY();
			canvas.drawBitmap(bmpP, x, y, null);
		}
	}
	if(aEncontrados.size() > 0){
		for(int i = 0 ; i < aEncontrados.size() ; i++){
			x = aEncontrados.get(i).getX();
			y = aEncontrados.get(i).getY();
			canvas.drawBitmap(bmpE, x, y, null);
		}
	}
	puntoMedio();
	x = array[0];
	y = array[1];
	canvas.drawBitmap(bmpL, x, y, null);
	canvas.restore();
}
```

Código 18: Método encargado de dibujar. Representar cada Bitmap en pantalla (dispositivos remotos, posición actual y mapa *indoor*).

Para poder representar nuestra localización en el mapa *indoor*, se propuso utilizar la técnica de triangulación. Con lo cual al aumentar el número de dispositivos remotos, mayor precisión se obtendrá en la medida. La clase *Lienzo.java* incorpora un método encargado de ello.

```
    private void puntoMedio(){
      x=y=0;
      if(aEncontrados.size()>0){
         for(int i = 0 ; i < aEncontrados.size() ; i++){
               x = x + aEncontrados.get(i).getX();
               y = y + aEncontrados.get(i).getY();
         }
         array[0] = x / aEncontrados.size();
         array[1] = y / aEncontrados.size();
      }
      else{
         array[0] = 0;
         array[1] = 0;
      }
    }
```

Código 19: implementación del método para obtener el punto medio y así una posición aproximada de nuestra situación.

- ***PintadoThread.java***

Esta clase es la encargada de llamar, de manera continua, al método *onDraw* de la clase *Lienzo*, para poder representar los elementos en pantalla. Simplemente consiste en realizar un bucle y en cada vuelta comprobar si una variable es *true* o *false*. Con lo cual mientras sea *true* pintará, por lo contrario no. Para realizar esta tarea, se extiende de *Thread* para que se comporte como un hilo, implementando el método común en todos los hilos, *run()*, el cual se ejecuta cuando iniciamos el hilo mediante *nombreHilo.start().*

CAPÍTULO 4

Manual de Usuario

4.1. Instalación de la Aplicación

Disponemos de tres opciones para instalar la aplicación:

- *Google Play (anteriormente Market)*
  Todas las aplicaciones que se desarrollan en Android pueden ser alojadas en el *Google Play* (tienda virtual de Android). Si la aplicación ha sido subida a dicha tienda de software, simplemente hay que buscarla, seleccionarla y aceptar la instalación. Se instalará automáticamente.
- *Instalación desde un archivo APK*
  Para crear el archivo ejecutable *apk* de un proyecto Android se podrá hacer seleccionando la opción de Eclipse *Import* o simplemente ejecutando el proyecto en un dispositivo virtual Android *(AVD Manager).* Una vez generado el archivo *apk,* guardaremos dicho archivo en cualquier parte de nuestro terminal. Usaremos un explorador de archivos, como por ejemplo *Astro,* para poder ejecutarlo y así poder instalarlo. Será necesario tener activado en el teléfono la opción *Origenes desconocidos,* que nos permitirá instalar aplicaciones que no hayan pasado por el *Google Play*.
- *Instalación desde Eclipse*
  Para instalar la aplicación desde Eclipse será necesario conectar el teléfono al PC mediante su cable USB. Una vez conectado, pondremos el terminal en modo depuración (*debug*), y al igual que si se tratará de un dispositivo virtual, nos aparecerá en el *AVD Manager*. Sólo tendremos que pinchar en la opción de Eclipse *Run as* y seleccionar nuestro dispositivo. Esta opción es la más usada para la creación de la aplicación, porque permite a los desarrolladores analizar todas las partes de la ejecución de la aplicación, pudiendo así detectar rápidamente cualquier error en la misma. También deberemos tener activado en el teléfono la opción *Orígenes desconocidos*.

4.2. Uso de la Aplicación

Como ya se ha mencionado anteriormente, el proyecto consta de una aplicación cliente encargada de buscar dispositivos Bluetooth, posicionarse en el plano *indoor* en función a los terminales encontrados, establecer conexión con ellos y recibir la información que se le proporcione para guardarla en una base de datos. Esa información recibida podrá ser tratada posteriormente con el fin deseado. Por lo tanto si esta aplicación ejerce de cliente, los otros terminales deben de tener una aplicación servidor escuchando, esperando la conexión para mandar los datos. En este proyecto fin de carrera propuesto, sólo se plantea la aplicación cliente, el programa que corra en el resto de dispositivos se sale del ámbito de este proyecto.

A la hora de probar la aplicación era necesario ese programa servidor en el terminal remoto, pero a pesar de no formar parte de este trabajo, se ha creado una pequeña aplicación, muy sencilla, para realizar las pruebas.

4.2.1. Pequeña aplicación servidor

Esta aplicación se ha creado con el fin de poder comprobar el correcto funcionamiento de la aplicación cliente. La actividad consiste en una pantalla principal, en la que aparece un *checkBox*, si es pulsado, el servidor se queda a la espera de la conexión con el cliente, y si se desactiva, finaliza la conexión y la escucha del canal radio. Una vez que se conecta con el cliente, simplemente manda un mensaje "ENVIO X", donde "X" es un contador que varía desde "0" hasta "N". El fin de este contador, era la comprobación correcta de los mensajes en el cliente.

En las siguientes imágenes se muestra esta pequeña aplicación. El envío de mensajes se hace mediante un servicio de manera invisible al usuario, de ahí que no se muestre ninguna imagen relacionada con ello.

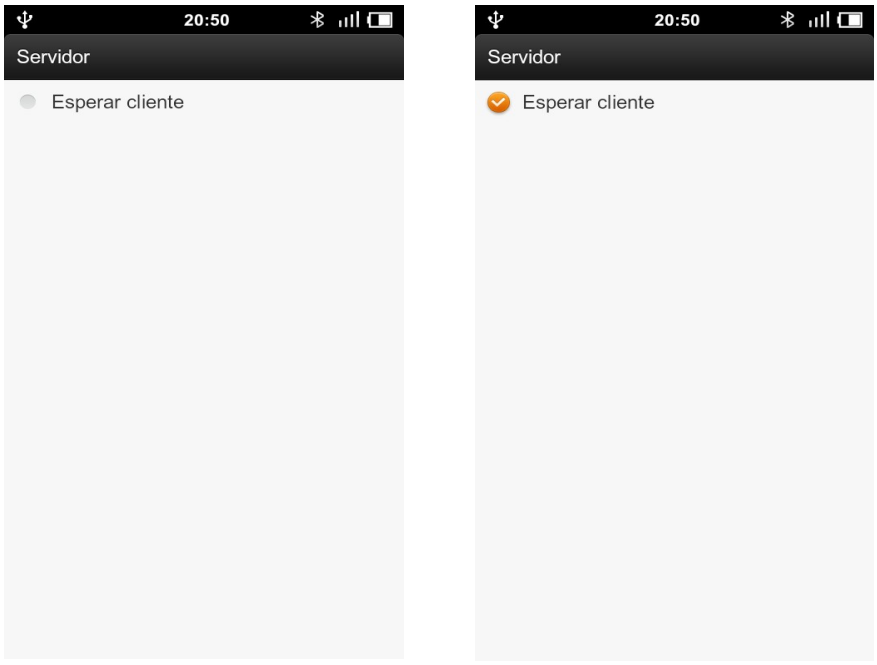
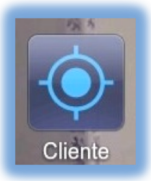


Figura 17: Aplicación servidor, *checkBox inactivo*, no se hace nada, *checkBox activo,* estamos a la espera del cliente.

4.2.1. Aplicación cliente.



La aplicación cliente, es la que forma la parte importante del proyecto presente. Antes de explicar su funcionamiento, se citarán algunos requisitos a tener en cuenta, los cuales se impusieron antes del desarrollo del trabajo:

- Cuando en este proyecto se habla de dispositivos o terminales Bluetooth, se engloban todos aquellos dispositivos que utilicen dicha tecnología, ya sean móviles, impresoras, marcos fotográficos, gafas 3D, máquinas industriales, etc.
- Se tiene el mapa en formato de imagen del local interior, y las direcciones MAC de los dispositivos Bluetooth deseados.
- Se conocen las posiciones de los terminales Bluetooth a los que queremos conectarnos, los cuales permanecerán fijos. El resto de dispositivos Bluetooth que encontremos y que no sean los deseados, se omiten y no se procederá a trabajar con ellos.
- El método utilizado para la geolocalización, es el conocido posicionamiento mediante triangulación. En líneas futuras se hablará de las posibles mejoras.
- La Base de Datos obtenida tras el proceso de búsqueda y recepción de datos, podrá ser utilizada mediante cualquier programa que lo permita. En este caso se visualizarán los resultados con la aplicación, *SQLite Database Browser 2.0*, para *MAC OS X* y *SQLite Editor* para *Android*.

Para comenzar, una vez instalada la aplicación, procederemos a buscarla en la lista de programas y a ejecutarla. Una vez iniciada, si no tenemos el Bluetooth activo en nuestro terminal, aparecerá la siguiente imagen:

Figura 18: Inicio de aplicación si se tiene la interfaz Bluetooth desactivada.

Como se puede apreciar en la figura 18, nos ofrece la opción de activar nuestra interfaz Bluetooth para poder comenzar nuestra aplicación. En caso de estar habilitada anteriormente, no aparecerá esta solicitud.

A continuación, si se a elegido la opción de activar nuestra interfaz Bluetooth, se nos muestra el mapa interior de nuestro edificio. Hay que tener en cuenta que aún no se ha empezado a buscar dispositivos, de ahí que solamente aparece una icono azul, en la parte superior izquierda, lo que nos indica que aún estamos sin realizar ninguna operación.

Figura 19: captura de pantalla inicial de la aplicación. Aún sin realizar la búsqueda de dispositivos.

Posteriormente para comenzar a escanear el canal radio en busca de los terminales Bluetooth, pulsaremos el menú y aparecerán dos opciones: *"Scan Bluetooth" y "STOP"*, como bien se puede apreciar en la siguiente imagen:

Figura 20: opciones para empezar o parar la búsqueda.

- Si elegimos la opción "Scan Bluetooth", comenzaremos a buscar dispositivos Bluetooth, mientras, nos aparecen en gris los dispositivos fijos con los que se quiere realizar las operaciones de localización y comunicación. Una vez que se localiza un terminal deseado, se muestra en color rojo, mientras que los que aún no se detecten permanecerán con el icono gris.

- Por otra parte, según se encuentren dichos terminales, el icono de localización (azul), variará, según las coordenadas obtenidas al realizar el mecanismo de triangulación. Aunque se citará en el siguiente capítulo, titulado *Líneas Futuras*, se ha de destacar que la elección de este método para localizar la posición, es simplemente un punto de partida, ya que el objetivo sería mejorarlo en versiones posteriores.

Cabe destacar que esta misma función de activar o desactivar el escaneo de dispositivos, puede realizarse pulsando la pantalla. Cuando se empieza a buscar dispositivos, se muestra un mensaje en pantalla para informar al usuario, al igual que cuando se finaliza de escanear. Estos dos mensajes ("Toast" en programación Andoird), pueden verse en las siguientes imágenes:

Figura 21: en la imagen de la izquierda se muestra el mensaje que nos informa del inicio de la búsqueda de dispositivos, mientras que en la imagen de la derecha la conclusión de dicha búsqueda.

Las siguientes dos imágenes muestran lo dos procesos de búsqueda y detección de los terminales, al igual que la localización aproximada. En caso de que los dispositivos no se encuentren en nuestro alcance, se muestran en color gris, mientras que si han sido detectados, se muestran en color rojo.

Figura 22: búsqueda de dispositivos. Muestra en el mapa los que se han detectado *(rojo)* y los que aún no se localizan *(gris)*.

- Una vez que hemos decidido terminar con la ejecución de la aplicación, se procederá a observar los resultados captados. Para ello deberemos detener la aplicación pulsando el botón *"STOP"* ya mostrado en la captura anterior, o pulsando la pantalla. De esta manera, todos los recursos utilizados en la aplicación se liberarán correctamente, al igual que se finalizará el curso de las actividades y servicios utilizados por la aplicación.

Para analizar los resultados obtenidos, se utilizará la aplicación en Andorid, *SQLite Editor*, como ya se ha mencionado al inicio de este apartado. Si se desea, esto se puede realizar con cualquier otra aplicación que permita la lectura de bases de datos.

A continuación se muestran dos imágenes para acceder a la tabla de la base de datos creada por la aplicación:

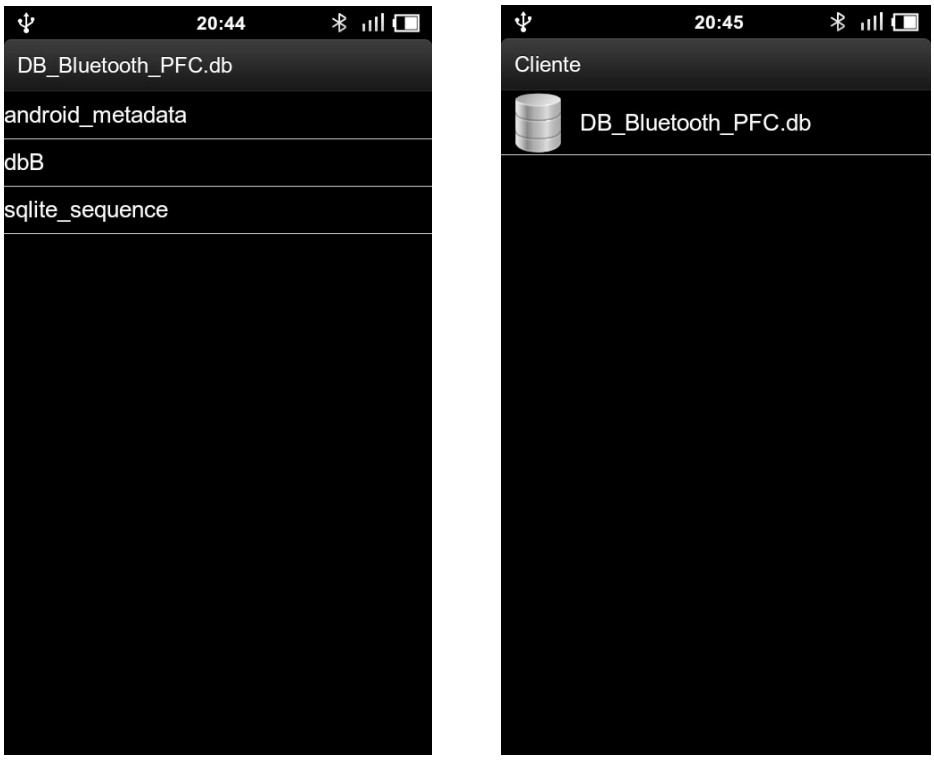


Figura 23: menú aplicación *SQLite Editor* para acceder a la información obtenida almacenada en la Base de Datos.

Y finalmente se muestra una figura con la tabla que contiene los siguientes campos:

- *SSID:* nombre del dispositivo remoto.
- *MAC:* dirección MAC del dispositivo remoto.
- *ESTADO*: estado (pendiente, encontrado) del terminal remoto.
- *X, Y:* coordenadas en el plano en el que se encuentra.
- *DATOS:* datos que nos manda dicho terminal.
- *ULTIMOS_D:* últimos datos recibidos por el dispositivo remoto.

_id	ssid	mac	estado
1	tablet Fnac 7	00:1A:13:31:C1:AB	encontrac
2		C8:64:C7:A4:CC:6F	pendiente
3		00:0B:13:65:A1:06	pendiente
4		00:22:A5:A7:1C:3B	pendiente
5	HTC Tattoo	00:22:A5:E1:F8:83	encontrac

Figura 24: campos _id (identificador), mac (dirección MAC).

estado	X	Y	DATOS	UL
encontrado	50	100	ENVIO 0 /r ENVIO 1 /r	EN
pendiente	100	600		
pendiente	380	300		
pendiente	200	450		
encontrado	230	300	ENVIO 0 /r ENVIO 1 /r	EN

Figura 25: estado (pendiente: no se han encontrado, encontrado: se han detectado), X e Y (coordenadas).

X	Y	DATOS	ULTIMOS_D
50	100	ENVIO 0 /r ENVIO 1 /r	ENVIO 1 /r
100	600		
380	300		
200	450		
230	300	ENVIO 0 /r ENVIO 1 /r	ENVIO 1 /r

Figura 26: DATOS (todos los datos recibidos), ULTIMOS_D (últimos datos recibidos).

CAPÍTULO 5

CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Conclusiones

Como conclusión general, mencionar que este proyecto ha conseguido alcanzar todos los objeticos propuestos inicialmente.

En primer lugar, se consiguió desarrollar la aplicación haciendo uso de las herramientas ofrecidas por la plataforma Android. Se logró obtener una versión de la aplicación donde se encuentran operáticas todas las funcionalidades que se exigieron al inicio.

Adicionalmente, se obtuvieron grandes conocimientos en el manejo de las tecnología relacionadas con el desarrollo del proyecto. Concretamente se aprendieron conceptos importantes para el desarrollo de software para terminales Android, manejo de diferentes APIs y de la tecnología Bluetooth. Junto a esto, se ha conseguido un aumento de conocimientos avanzados en el lenguaje de programación y en el funcionamiento de bases de datos.

Se comprendieron los elementos más importantes a tener en cuenta a la hora de diseñar e implementar una GUI que permita a los usuarios navegar a través de la aplicación de una manera fluida e intuitiva.

Finalmente, destacar que a medida que la creación de la aplicación avanzaba, se implementaron funciones adicionales al diseño original. Esto sucedió ya que se notaron deficiencias y posibles mejoras. De esta forma se logró llegar al diseño final, que cumple con los objetivos iniciales del proyecto.

5.1. Líneas futuras

Este proyecto deja una gran variedad de posibilidades de ampliación, pudiendo abrir varias líneas de investigación y desarrollo, ya sea desde un ámbito más específico, para una empresa o proyecto particular, o un ámbito más general, desde el punto de mejora en localización interior.

A medida que aparezcan nuevos SDK con funcionalidades mejoradas y nuevos avances en la tecnología Bluetooth, las posibilidades de mejora se irán ampliando, pudiendo desarrollar aplicaciones bastante más completas y con ello, más complejas. Algunas e estas líneas futuras pueden ser las siguientes:

- Mejorar la realizada aumentada. La aplicación podría representar en el plano, además del icono del dispositivos, una información adicional sobre ellos.
- Aumentar el número de tecnologías empleadas. Actualmente se ha realizado este proyecto con Bluetooth, sin embargo, se realizó en paralelo una versión similar pero mediante tecnología Wifi. Si ambas se fusionaran en una sola aplicación se podría conseguir mejoras significativas.
- Visitas guiadas. Una versión de esta aplicación podría ser creada para optimizar su funcionamiento como aplicación de visita guiada, pudiendo ir conociendo el lugar por el que se encuentra dentro de un edificio, y a la vez visualizando información relevante del lugar. Por ejemplo en museos, centros comerciales, parques de atracciones, etc.
- Mejorar la sensibilidad de localización. La aplicación, como ya se ha comentado, utiliza la fórmula matemática basada en la triangulación para mostrar la situación actual del dispositivo, ya que así se planteó al inicio del proyecto. Una posible mejora sería adaptar una nueva versión pero utilizando la potencia de la señal de los dispositivos encontrados. Esto es posible debido a que se puede obtener dicha información de manera relativamente fácil, ya que existe un método específico que nos devuelve la potencia del dispositivo encontrado. De esta forma se podrían obtener mejoras importantes en la localización del terminal, pero hay que destacar que no solo valdría con obtener la potencia del terminal remoto, ya que habría que tener en cuenta otros factores como las paredes, materiales, etc. Es por ello por lo que se planteó desde el principio que en esta aplicación inicial no se implementara esta mejora, dejándolo indicado para proyectos posteriores a este.

CAPÍTULO 6

BIBLIOGRAFÍA

[1] Android. Guía para desarrolladores. W. Frank Ableson, Robi Sen y Chris King.

[2] Web oficial para desarrolladores Android.

http://developer.android.com/index.html

[3] Comunidad de desarrolladores I.

http://stackoverflow.com/

[4] Comunidad de desarrolladores II.

http://rooibo.wordpress.com/

[5] Comunidad de desarrolladores III.

http://www.chuidiang.com/

[5] Video tutoriales Android de la Universidad Politécnica de Valencia.

http://politube.upv.es

[6] Video tutoriales Android en Youtube.

http://www.youtube.com/user/0utKast/videos?view=0

ANEXO / Diagrama de Clases UML