ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Universidad
Politécnica
de Cartagena

DESARROLLO DE UNA APLICACIÓN PARA ANDROID DE GEOLOCALIZACIÓN Y COMUNICACIÓN EN EXTERIORES MEDIANTE WIFI

AUTOR: Antonio Hellín Hernández
DIRECTOR: Juan Carlos Sánchez Aarnoutse

Marzo / 2013

Autor	Antonio Hellín Hernández
E-mail del autor	antonio.hh7@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 Exteriores Mediante Wifi
Descriptores	

Resumen

El objetivo de este proyecto desarrollar una aplicación para Android de realidad aumentada que sea capaz de geolocalizar el dispositivo cuando cambia de ubicación en espacios interiores o sin cobertura GPS.

Para ello, la aplicación realizará un sondeo de los dispositivos wifi cercanos. La app debe contar con una BDD en la que está almacenada la posición geográfica de todos los dispositivos conocidos. Con toda esa información debe hacer una estimación del lugar en el que se encuentra. Finalmente, la aplicación mostrará en google-maps el lugar en que se encuentra. Además, dicho plano mostrará la situación de los dispositivos

Adicionalmente, se podrá hacer un intercambio de mensajes mediante el protocolo UDP.

Titulación	Ingeniero Técnico de Telecomunicaciones, Esp. Telemática
Departamento	Tecnologías de la información y las comunicaciones
Fecha de presentación	21/03/2013

AGRADECIMIENTOS

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

Merecen una mención destacada mis amigos y compañeros de clase, así como 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 este Proyecto Fin de Carrera y por ser muy benevolente a pesar de retrasarnos más de lo que debimos.

Y por supuesto, a Alejandro Barceló Sánchez. Apoyo incondicional en el desarrollo del proyecto porque siempre ha sabido ayudarme cuando más lo he necesitado. Ha cumplido con creces el significado de la palabra amigo y compañero.

A todos aquellos que me habéis ayudado a llegar hasta aquí, MUCHAS GRACIAS.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

Este proyecto nace con el objetivo de familiarizarse 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 funciona- miento 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 objetivo 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, orientado a ser probado y testado sobre el terminal Samsung Galaxy R con la ayuda de otros terminales como son el ZTE-Blade o el Motorola Delphi.

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 ampliaciones por parte de terceros desarrolladores.

1.2. Objetivos

El principal problema a la hora de realizar aplicaciones para *Smartphones* que requieran geolocalización es la imposibilidad de realizar una ubicación correcta debido a que el dispositivo no capta la señal de un número adecuado de satélites. Este hecho se da en interiores de edificios pero también se puede dar en exteriores, por ejemplo, en ciudades con una densidad de edificaciones que apantalle la señal GPS o en situaciones en las que simplemente el dispositivo no dispone de un receptor GPS o prefiere no emplearlo.

El objetivo del proyecto es proporcionar al usuario un método de localización en exterior de dispositivos Wifi. Como es probable que no se disponga de señal GPS, la localización la realizaremos mediante la triangulación de dispositivos Wifi encontrados. Las coordenadas de los dispositivos Wifi serán conocidas con anterioridad, con lo cual, la triangulación se realizará de manera trivial. El servicio deberá ser ejecutado en segundo plano para cuando el usuario se desplace por el mapa vaya localizando nuevos dispositivos.

La primera parte del proyecto se puede decir que se centra en el descubrimiento de terminales (mediante un servicio ejecutado en segundo plano) y el posicionamiento. Conforme se vayan localizando dispositivos Wifi nuevos, se irán almacenando en una base de datos. En segundo lugar, se encuentra la comunicación con ellos. La información recibida de cada dispositivo se almacena en la base de datos de manera que podrá ser utilizada posteriormente.

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.

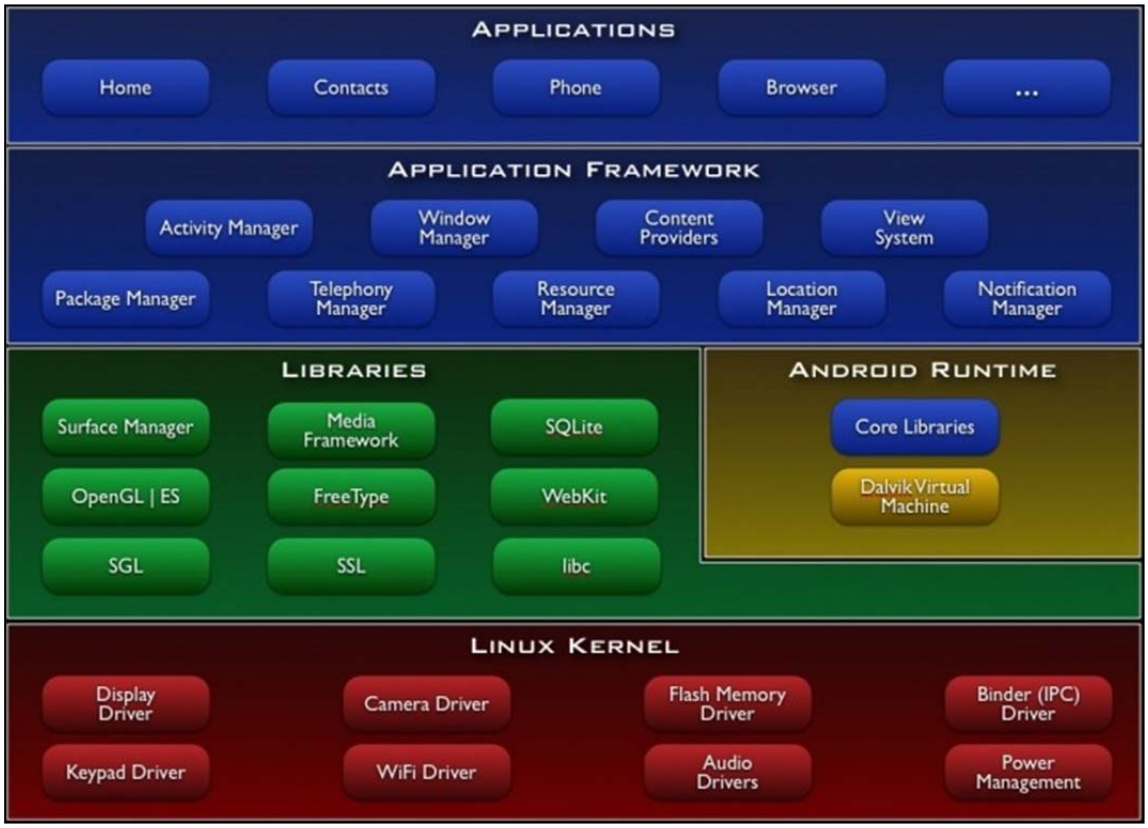

Figura 2.1: Arquitectura Android

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 aplicaciones, 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.

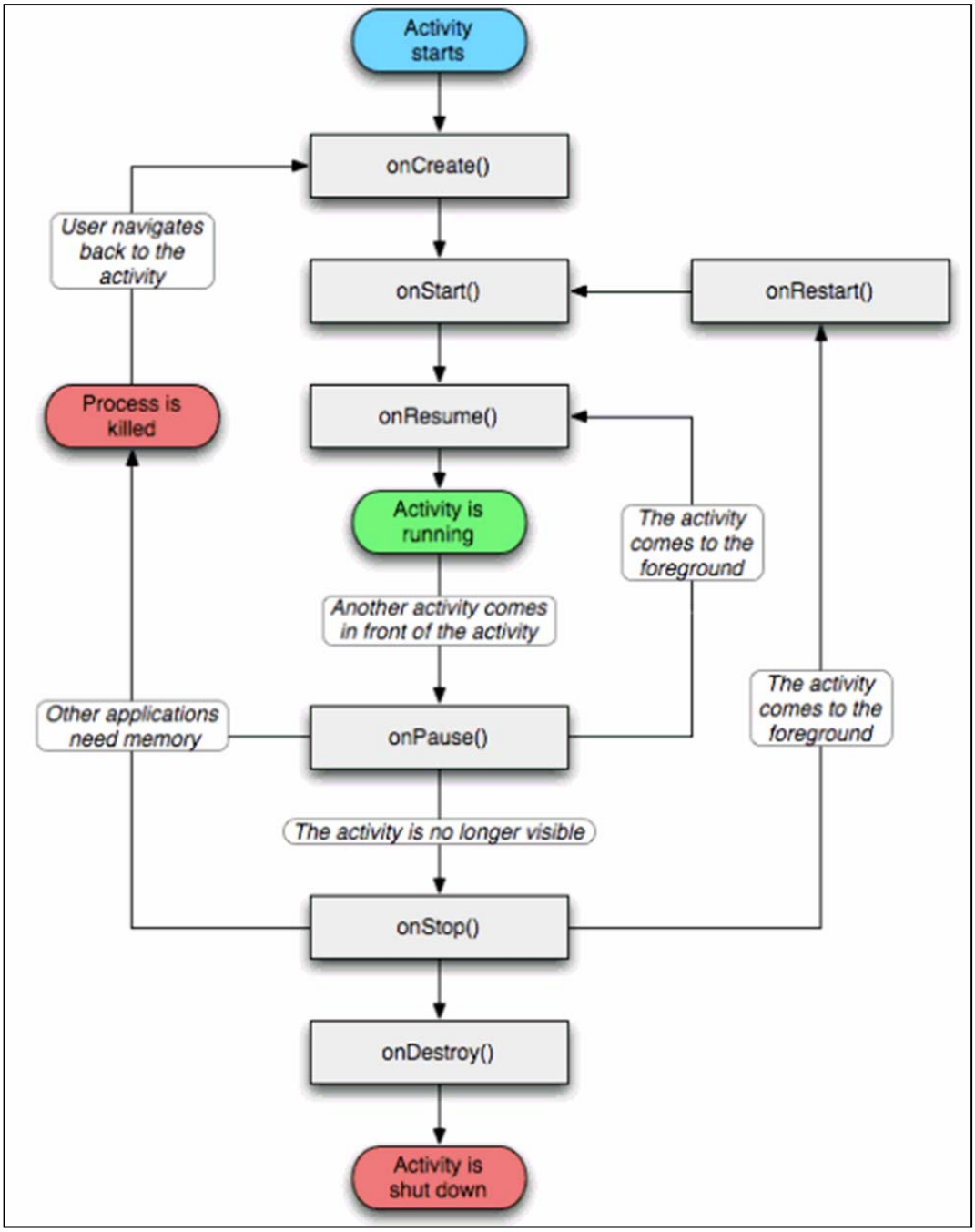

Figura 2.2: Ciclo de vida de una aplicación Android

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 Galileo 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 sera Eclipse *IDE for Java Developers* para Windows 32 bits. Una vez descargado, basta descomprimir y ejecutar eclipse.exe para comenzar a usar la aplicación.

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 "SDK". En nuestro caso elegimos la versión para Windows. Una vez completada la descarga, descomprimimos en la ubicación deseada.

Antes de proseguir, es recomendable almacenar la ruta de la subcarpeta "tools" en la variable de entorno "path" de nuestro sistema. Para ello hacemos clic con el botón derecho en "Equipo" para acceder a las Propiedades del sistema. En configuración avanzada podemos acceder a dichas variables. Basta buscar la variable "path" en la lista y hacer doble clic para editar. Al valor actual debemos añadir nuestra ruta, precedida por ";", para conservar los valores anteriormente almacenados.

Esto nos facilitará el acceso mediante símbolo del sistema a algunas configuraciones del SDK Android por comandos, no siendo necesario escribir la ruta completa de la ubicación de las utilidades.

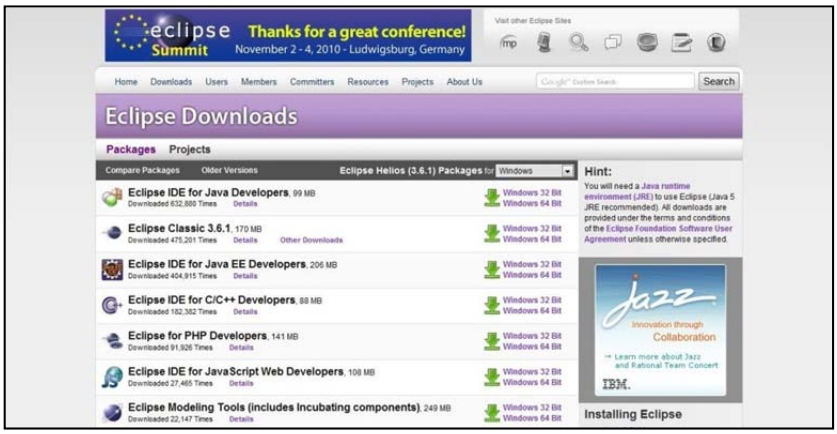

Figura 2.3: Descargar Eclipse

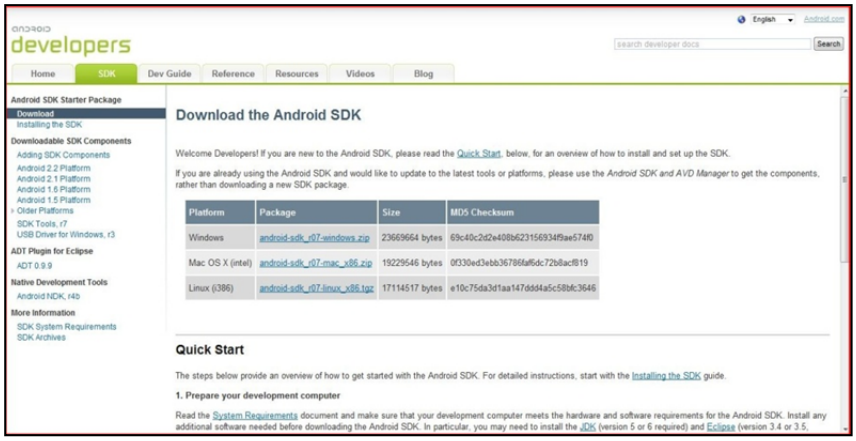

Figura 2.4: Descargar SDK Android

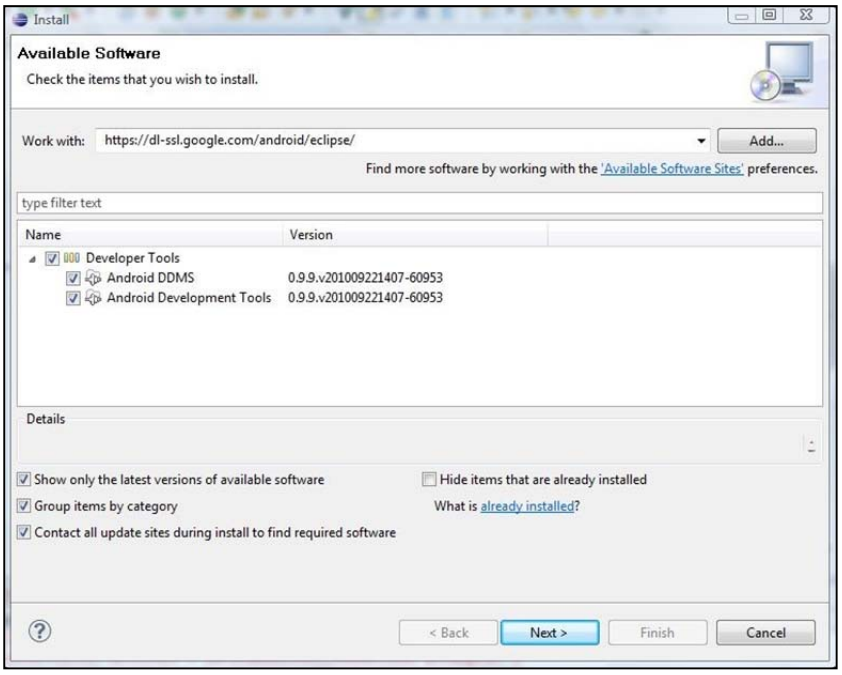

Figura 2.5: Instalación del *Android Developer Tools*

2.1.2.3. Instalación del plugin ADT de Eclipse

Usando Eclipse, este plugin nos facilitará tareas de programación, simulación y depuración de nuestras aplicaciones. Para descargarlo, basta acceder a *Install new software* del menú *Help* e introducir la dirección donde se ubica (https://dl-ssl.google.com/android/eclipse) como directorio de trabajo. Aunque por razones de seguridad es preferible usar el protocolo https, en caso de problemas también podría usarse http.

A continuación debe ofrecernos la instalación de *Developer Tools*, y desplegando el árbol podremos ver que se compone de *Android DDMS* y *Android Developer Tools*. Elegimos ambas herramientas y tras aceptar la correspondiente licencia en el siguiente paso, comienza la descarga del nuevo software.

Una vez descargado e instalado hay que configurar las nuevas utilidades para que Eclipse reconozca el SDK Android. Para ello, en el menú *Window/Preferences* elegimos la opción Android. En el cuadro de dialogo mostrado debemos introducir la ruta donde fue instalado el SDK en el paso anterior, por ejemplo C:\nandroid-sdk-windows. Si todo el proceso se ha realizado correctamente, tras pulsar el botón *Apply* nos aparecerán todas las versiones de las APIs incluidas en nuestro SDK.

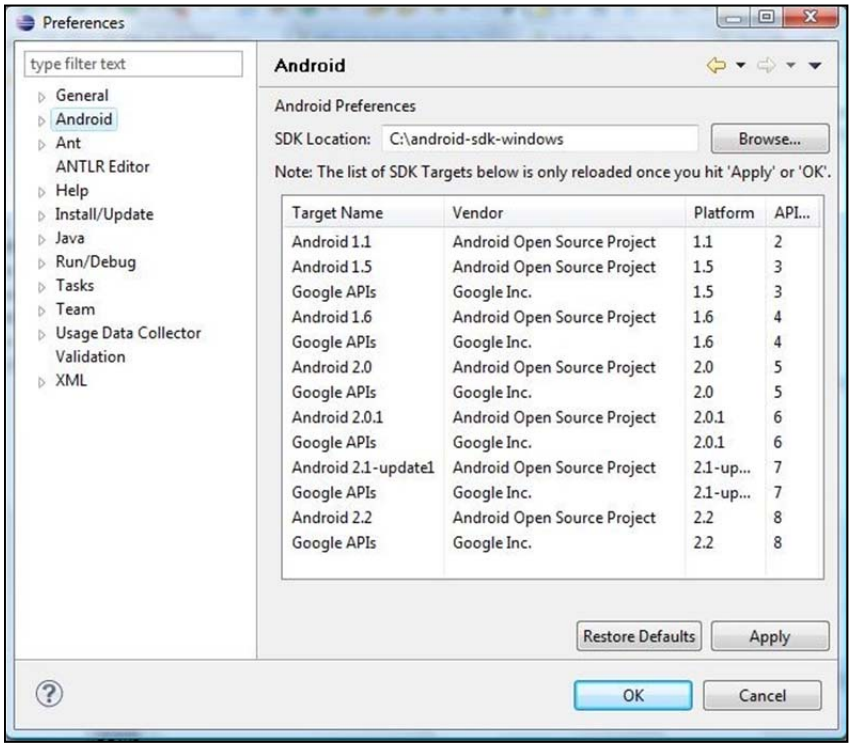

Figura 2.6: Ubicación del SDK

Aceptamos los cambios y ya podremos empezar a desarrollar para Android. Sin embargo, para completar la puesta en marcha de las utilidades que nos ofrece el plugin, todavía nos queda por crear nuestro simulador, o simuladores, para probar nuestras aplicaciones. Esto evitará tener que acceder a nuestro terminal para testear cada cambio realizado en el código fuente desarrollado.

Para ellos accedemos al menú *Window\Android SDK and ADV Manager*, que nos mostrara una ventana del mismo nombre. Ahí, seleccionando la opción *Virtual Devices* podremos crear fácilmente nuestros dispositivos haciendo clic en "new…". Basta elegir el nombre deseado, elegir la versión de la API deseada y añadir la compatibilidad hardware que vayamos a necesitar. Tras hacer clic en *Create ADV* nuestro dispositivo aparecerá en el listado.

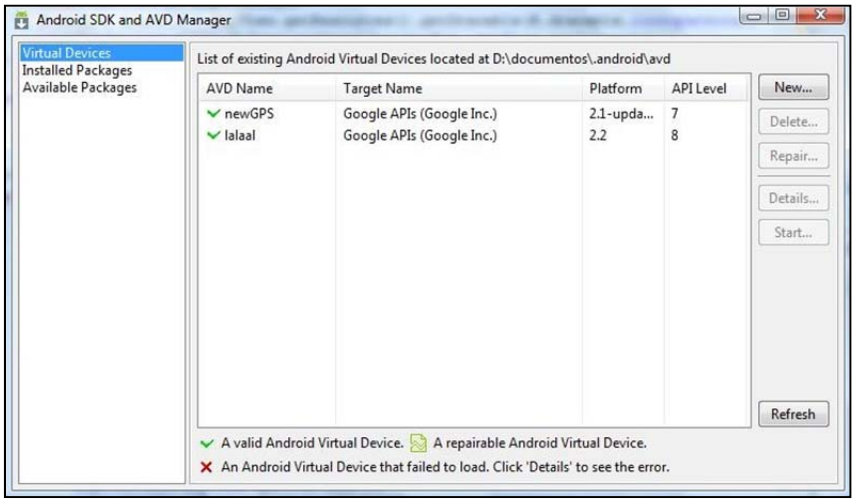

Figura 2.7: AVD *Manager*

2.1.3. Google Maps Android v1 API

En primer lugar debemos tener claro el concepto de API para un mejor entendimiento de la API de Google Maps. Una API (*Application Programming Interface*) es una *llave de acceso* a funciones que nos permiten hacer uso de un servicio web provisto por un tercero, dentro de una aplicación web propia, de manera segura.

Una vez comentado de forma generica el concepto de API es hora de profundizar un poco más en la API de Google Maps. Haciendo uso de la API de Google Maps podremos obtener una serie de funciones proporcionadas por Google que se encargaran de todo lo referido a mapas.

Antes de empezar a utilizar el servicio de mapas de Google es necesario realizar algunas tareas previas. En primer lugar, debemos asegurarnos de que tenemos instalado el paquete correspondiente a la versión de Android para la que desarrollamos enriquecido con las APIs de Google. Estos paquetes se llaman normalmente *Google APIs by Google, Android API x, revisión y*. Esto podemos comprobarlo y descargarlo si es necesario desde Eclipse accediendo al *Android SDK and AVD Manager*. Al crear nuestro proyecto de Eclipse tendremos que seleccionar como *target* que vamos a utilizar la API de Google. Con todo esto ya tendríamos creado nuestro proyecto de Eclipse y estaríamos preparados para poder ejecutar aplicaciones en el emulador sobre la versión correcta de Android y las APIs necesarias de Google. Por tanto, ya podemos centrarnos en la utilización de dichas APIs.

Esto último también requiere algunos pasos previos. Para poder utilizar la API de Google Maps se requiere la obtención previa de una clave de uso (*API Key*), que estará asociada al certificado con el que firmamos digitalmente nuestra aplicación. Esto quiere decir que si cambiamos el certificado con el que firmamos nuestra aplicación (algo que se hace normalmente como paso previo a la publicación de la aplicación en el Market) tendremos que cambiar también la clave de uso de la API. En el tema de los certificados no voy a entrar mucho puesto que lo trataremos en un artículo posterior, por ahora tan sólo diremos que durante la construcción y depuración de nuestras aplicaciones en el emulador de Android se utiliza automáticamente un certificado de depuración creado por defecto. Por tanto, para poder depurar aplicaciones en el emulador que hagan uso de Google Maps tendremos que solicitar una clave asociada a nuestro certificado de depuración.

Para ello, en primer lugar debemos localizar el fichero donde se almacenan los datos de nuestro certificado de depuración, llamado *debug.keystore*. Podemos saber la ruta de este fichero accediendo a las preferencias de Eclipse, sección *Android*, apartado *Build*, y copiar la ruta que aparece en el campo *Default Debug Keystore*. Una vez conocemos la localización del fichero *debug.keystore*, vamos a acceder a él mediante la herramienta keytool.exe de java para obtener el *hash* MD5 del certificado. Esto lo haremos desde la línea de comandos de Windows mediante el comando -> `keytool –list –alias androiddebugkey –keystore "Ruta" –storepass android –keypass android`

Copiaremos el dato que aparece identificado como Huella digital de certificado (MD5) y con éste accederemos a la web de Google (http://code.google.com/android/maps-api-signup.html) para solicitar una clave de uso de la API de Google Maps para depurar nuestras aplicaciones (Insisto, si posteriormente vamos a publicar nuestra aplicación en el Market deberemos solicitar otra clave asociada al certificado real que utilicemos). Dicha web nos solicitará la marca MD5 de nuestro certificado y nos proporcionará la clave de uso de la API.

Con esto, terminamos con todos los pasos previos para la utilización de los servicios de Google Maps dentro de nuestras aplicaciones Android. Una vez contamos con la clave de uso de la API, la inclusión de mapas en nuestra aplicación es una tarea relativamente sencilla y directa.

2.2. Redes Wi-Fi

2.2.1. Introducción a las redes inalámbricas

Durante los últimos años han surgido y se han hecho con gran popularidad nuevas tecnologías inalámbricas como WIFI, WIMAX, GSM, Bluetooth, Infrarrojos, etc, siendo

los dispositivos inalámbricos una de las grandes revoluciones tecnológicas de los últimos tiempos.

Las tecnologías inalámbricas, o *wireless*, 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

En 1999 los principales vendedores de soluciones inalámbricas (3com, Aironet, Intersil, Lucent Technologies, Nokia y Symbol Technologies) crearon una asociación conocida como WECA, con el fin de resolver el problema que suponía la existencia de diferentes estándares, lo que provocaba problemas de incompatibilidad. Por lo tanto esta asociación se propuso crear una marca que permitiese fomentar la tecnología inalámbrica y asegurar la compatibilidad de los dispositivos.

Es en el año 2000 cuando WECA con la norma IEEE 802.11b certifica la interoperabilidad de equipos bajo la marca Wi-Fi, con lo que se garantiza que todos los elementos con el sello Wi-Fi pueden trabajar juntos sin problemas independientemente del fabricante de cada uno de ellos.

En 2002 la asociación, formada ya por casi 150 miembros, anuncia la marca Wi-Fi utilizada para certificar equipos IEEE 802.11a de la banda de 5 Ghz, debido a que las velocidades máximas ofrecidas por la norma 802.11b (11 Mbps) había sido superada.

2.2.3. Protocolos

Desde 1997, cuando se certificó el primer estándar 802.11 con una velocidad de transferencia máxima de 2 Mbps, han ido surgiendo nuevos estándares que permiten velocidades cada vez mayores y con distintas bandas de frecuencias, alcanzando hoy en día hasta 300 Mbps.

A continuación se describen los diferentes protocolos para redes Wi-Fi que han sido certificados como estándares desde la aparición del IEEE 802.11

- **802.11** ***legacy:*** publicado en 1997, es la versión original del estándar IEEE 802.11. Permitía dos velocidades teóricas de transmisión, 1 y 2 Mbps, mediante señales infrarrojas en la banda ISM (*Industrial, Scientific and Medical*, de uso no

comercial) a 2,4 GHz. Este estándar definía el protocolo CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) como método de acceso. Una parte importante de la velocidad de transmisión se utiliza en las necesidades de esta codificación para mejorar la calidad de la transmisión bajo condiciones ambientales diversas, lo que produjo dificultades de interoperabilidad entre equipos de diferentes marcas y rechazo entre los consumidores. En la actualidad no se fabrican productos sobre este estándar.

- **802.11b:** certificado en 1999, corrige las principales debilidades del estándar original y es el primer protocolo de la familia en ser aceptado por los consumidores. Permite una velocidad máxima de transmisión de 11 Mbps trabajando en la misma banda de frecuencia de 2.4 GHz. También utiliza el método de acceso CSMA/CA lo que reduce en la práctica la velocidad máxima de transmisión a 5.9 Mbps sobre TCP y a 7.1 Mbps sobre UDP.

- **802.11ª:** creado en 1997, no fue aprobado hasta 1999, cuando lo hizo junto con el 802.11b, y apareció en el mercado en productos en el 2001. Este estándar utiliza el mismo protocolo de base que el estándar original, pero opera en la banda de 5 GHz y utiliza 52 subportadoras OFDM (*Orthogonal Frequency Division Multiplexing*) con una velocidad máxima de 54 Mbps, lo que hace que sea un estándar práctico para redes inalámbricas con velocidades reales de unos 20 Mbps. No puede interoperar con equipos del estándar 802.11b, a menos que dicho equipo implemente ambos estándares. Un punto a favor para este protocolo es que al utilizar la banda de frecuencias de 5 GHz se presentan muchas menos interferencias, debido a que la banda de 2.4 GHz es utilizada por una gran cantidad de aparatos domésticos. Como contrapartida esta banda restringe el uso de los equipos a puntos en línea de vista, lo que requiere una instalación de un mayor número de puntos de acceso y a una cobertura menor. Este protocolo conserva su velocidad máxima de 54 Mbps en un rango de 30 metros en el exterior y de 12 metros en el interior.

- **802.11h:** aparece en 2003 como una modificación del 802.11a con el fin de resolver los problemas derivados de la coexistencia de las redes Wi-Fi con sistemas de radares y satélites, debido a que la banda de 5 GHz era utilizada generalmente por sistemas militares. Este nuevo protocolo proporciona a las redes 802.11a la capacidad de gestionar dinámicamente tanto la frecuencia, como la potencia de transmisión.

- **802.11g:** aprobado en 2003, al igual que el 802.11b utiliza la banda de 2.4 GHz y es compatible con el mismo, pero opera con una velocidad teórica máxima de 54 Mbps (unos 24.7 Mbps de velocidad real), semejante a la del 802.11a. El

diseño del estándar se realizó pretendiendo hacerlo compatible con el 802.11b, pero en redes bajo el estándar g, la presencia de nodos bajo el estándar b reduce significativamente la velocidad de transmisión. En la actualidad se comercializan equipos con esta especificación que, con potencias de hasta medio vatio, permiten establecer comunicaciones de hasta 50 km de distancia mediante antenas parabólicas apropiadas.

- **802.11n**: se espera que este protocolo se implante en 2008, aunque existen dispositivos que ofrecen de forma no oficial este estándar. La velocidad real de transmisión podría llegar a 600 Mbps, lo que significa una velocidad teórica todavía mayor, y lo que supondría redes 10 veces más rápidas que las de estándar a y g, y casi 40 veces más rápidas que las de estándar b. El alcance que se espera sea alcanzado también se incrementará gracias a la tecnología MIMO (*Multiple Input – Multiple Output*), que permite utilizar varios canales a la vez para enviar y recibir datos gracias a la incorporación de varias antenas.

- **802.11e:** este estándar permite soportar tráfico en tiempo real en todo tipo de entornos y situaciones. El objetivo de este estándar es introducir nuevos mecanismos a nivel de enlace para soportar los servicios que requieren garantías de Calidad de Servicio (QoS).

- **802.11i:** este protocolo está dirigido a batir la vulnerabilidad actual en la seguridad para protocolos de autenticación y de codificación, abarcando los protocolos 802.1x, TKIP (Protocolo de Claves Integra – Seguras – Temporales), y AES (Estándar de Cifrado Avanzado).

- **802.11 Super G:** trabaja en la banda de 2.4 GHz y gracias al chipset Atheros alcanza velocidades de transferencia de 108 Mbps.

2.2.4. Tecnología Wi-Fi

Wi-Fi, es un conjunto de estándares para redes inalámbricas de área local (WLAN) basado en las especificaciones IEEE 802.11. Fue creada por la Wi-Fi Alliance (anteriormente WECA, *Wireless Ethernet Compability Alliance*), la organización comercial que prueba y certifica que los equipos cumplen los estándares 802.11. Se identifica con un símbolo con el estilo del ying-yang y su nombre no es un acrónimo de *Wireless Fidelity*, a pesar de que en sus comienzos se añadió junto con el nombre Wi-Fi la frase *The Standard for Wireless Fidelity* con el fin de dar significado al mismo, sino que fue creado como un juego de palabras relacionado con Hi-Fi (*High Fidelity*).

Resaltando sus principales características, se podría decir que las Redes Inalámbricas Wi-Fi son muy fáciles de adquirir, no tanto de configurar y muy difíciles de proteger. También cabe aclarar que la tecnología Wi-Fi no es compatible con otros tipos de conexiones wireless como Bluetooth, GPRS, UMTS, etc.

2.2.5. Dispositivos

2.2.5.1. Puntos de acceso Wireless

Los puntos de acceso en redes Wi-Fi son los elementos que interconectan los distintos dispositivos de comunicación inalámbrica para formar una red *wireless*. Habitualmente estos dispositivos pueden conectarse a redes cableadas, permitiendo intercambiar información entre dispositivos cableados y *wireless*. De la misma manera, distintos puntos de acceso pueden ser conectados permitiendo realizar *roaming*. Son los encargados de crear la red, permaneciendo a la espera de nuevos clientes a los que dar servicios. El punto de acceso recibe la información, la almacena y la transmite entre la WLAN y la LAN cableada. Estos dispositivos tienen direcciones IP asignadas, para poder ser configurados. Desde un único punto de acceso se puede dar soporte a un grupo de usuarios y trabajar en un rango desde unos 30 a varios cientos de metros, siempre en función de las antenas utilizadas. Los Puntos de Acceso pueden ser agrupados en dos categorías:

- **Puntos de Acceso Robustos:** son bastante inteligentes e incorporan funciones adicionales de gestión y seguridad, como *Firewall*, *Site Survey* o no emitir el ESSID (algo que se comentará más adelante). Además son más costosos y complicados de gestionar y suelen sobrecargar el tráfico. En algunos casos disponen de slots libres para futuras actualizaciones.
- **Puntos de Acceso Básicos:** son más económicos y sencillos de gestionar y configurar, además suele ser más sencillo compatibilizarlos con otras marcas.

2.2.5.2. Routers

Los routers reciben la señal de la línea que ofrezca el operador de telefonía y se encargan de todos los problemas relacionados a la recepción de la señal, como el control de errores y la extracción de la información, para que los diferentes niveles de red puedan trabajar. Los routers trabajan de manera conjunta con los puntos de acceso *wireless*, funcionando estos últimos a modo de emisor remoto, es decir, en lugares donde la señal wifi del router no tenga suficiente radio.

Otros dispositivos como *hubs* o *switches* también pueden encargarse de la distribución de la señal, pero no pueden encargarse de las tareas de recepción. Hoy en día la mayoría de los Puntos de Acceso incluyen las funcionalidades de los Routers, por lo que estos se están viendo sustituidos.

2.2.5.3. Tarjetas de red Wi-Fi

Son los dispositivos encargados de la recepción de información en estaciones de redes Wi-Fi. Una de las principales características que las diferencian entre sí es el tipo de interfaz que utilizan, es decir, el puerto de conexión de la tarjeta. A continuación se describen los diferentes tipos de tarjetas en función de la interfaz que utilizan:

- **PCI:** *Peripherial Component Interconnect*, bus de interconexión de componentes periféricos, que conecta directamente a la placa base de la computadora dispositivos periféricos (bus local). Permite configurar el dispositivo de manera dinámica y suele ser utilizado en ordenadores de sobremesa. Habitualmente suele disponer de conectores para antenas.
- **Mini PCI:** Este tipo de tarjetas posee características semejantes a las PCI, pero su tamaño es mucho menor, se utiliza en portátiles y no dispone de conectores para antenas.
- **PCMCIA:** *Personal Computer Memory Card Internacional Association*, es un dispositivo utilizado habitualmente en portátiles, la mayoría de estas tarjetas solo son capaces de llegar a la tecnología 802.11b, no permitiendo disfrutar de una velocidad de transmisión demasiado elevada. Las tarjetas PCMCIA también reciben el nombre de PC Card si son de 16 bits o CARD BUS si son de 32 de bits.
- **CENTRINO:** *Centrino Mobile Technology* es una iniciativa de Intel para promocionar una combinación prestablecida de CPU, chipset de la placa base e interfaz de red inalámbrica en el diseño de ordenadores personales portátiles. La interfaz de red es del tipo Intel PRO/*Wireless* 2100 (802.11b) o PRO/*Wireless* 2200 (802.11g). Todo lo mencionado correspondía simplemente al tipo de interfaz que utiliza la tarjeta Wi-Fi, sin embargo a la hora de considerar una tarjeta *wireless*, sobre todo para la tarea de auditorías, es conocer el chipset y los drivers que utiliza, que no tienen por qué ser desarrollados por la compañía que ha fabricado la tarjeta Wi-Fi.
- **USB***: Universal Serial Bus*, provee un estándar de serie para conectar dispositivos a un PC.

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 -> Project. Dentro encontraremos una carpeta de nombre *Android*, dentro de esta seleccionaremos la opción *Android Application Project*. A continuación nos aparecerá la siguiente ventana:

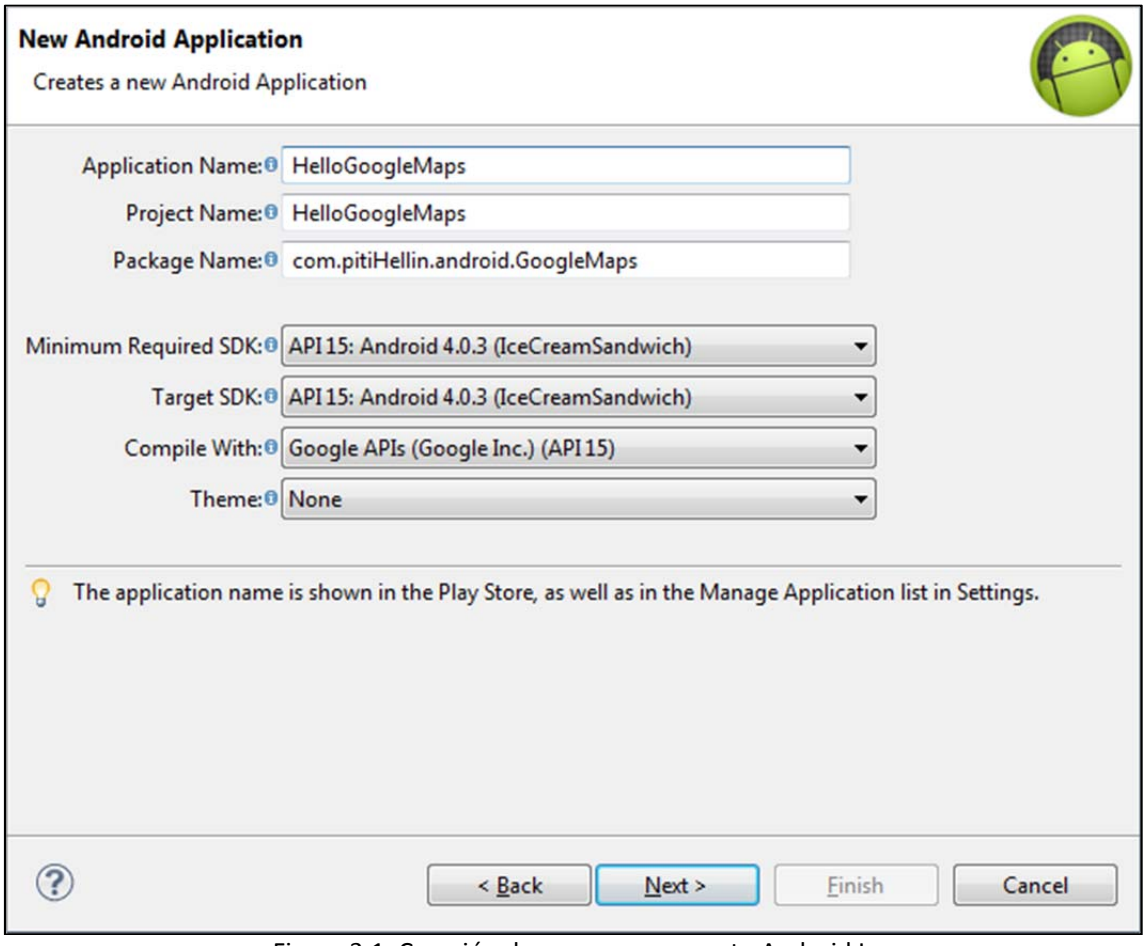

Figura 3.1: Creación de un nuevo proyecto Android I

A continuación se detallaran los campos que aparecen en la ventana emergente que nos acaba de aparecer:

- *Application Name*: nombre de la aplicación.
- *ProjectName:* nombre del proyecto Android.
- *Package Name*: nombre del paquete donde estará ubicado el proyecto.
- *Minimum Required SDK*: versión mínima de Android que soportara nuestra aplicación.
- *Target SDK*: versión máxima de Android que soportara nuestra aplicación
- *Theme*: tema que tendrá por defecto la aplicación.

Una vez que se han rellenado todos los campos, pulsamos *Next* y aparecerá un menú donde deberemos de indicar donde se creara el proyecto Android, entre otras opciones de menos índole.

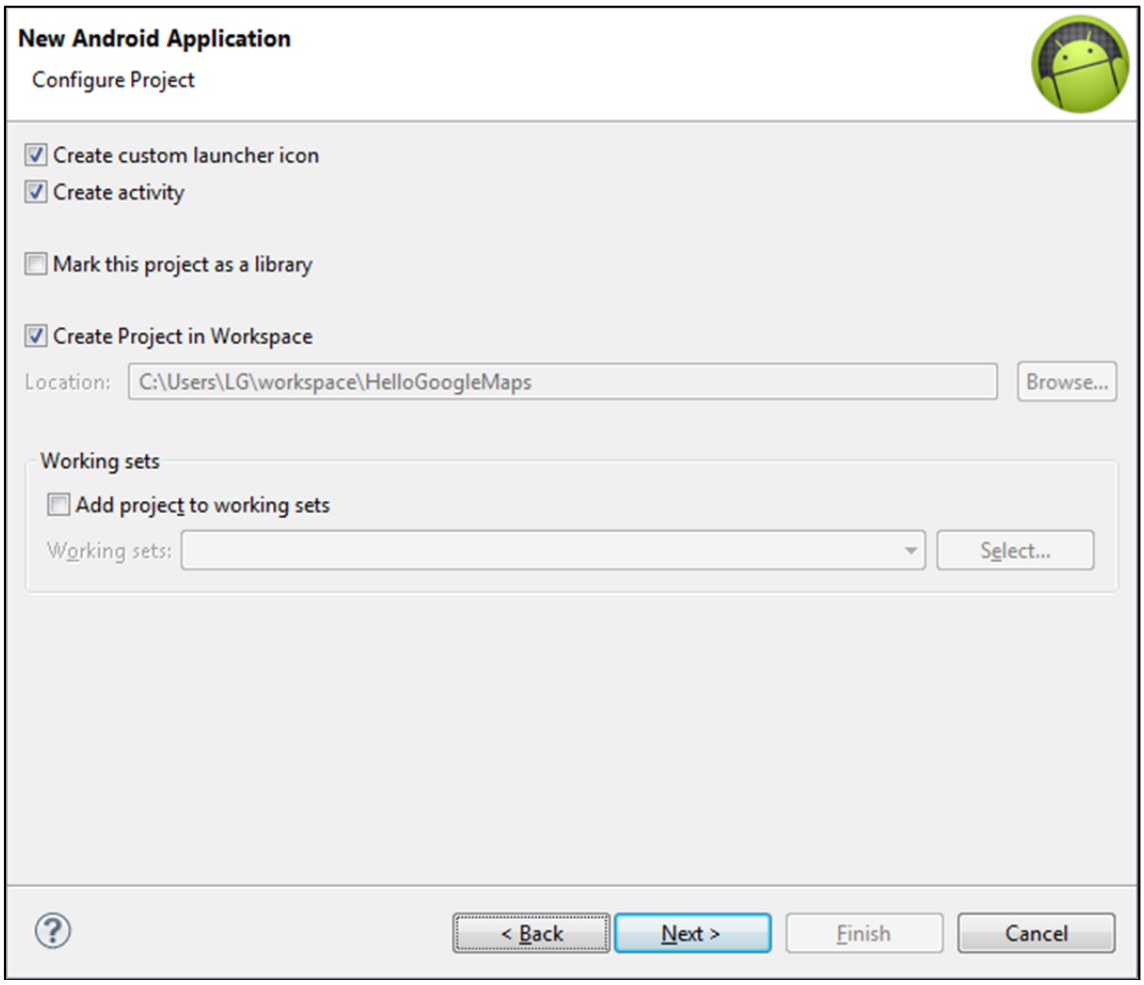

Figura 3.2: Creación de un nuevo proyecto Android II

En la siguiente pantalla tendremos la posibilidad de elegir el icono que tendrá nuestra aplicación una vez instalada en un terminal. También se podrá elegir algunos iconos por defecto.

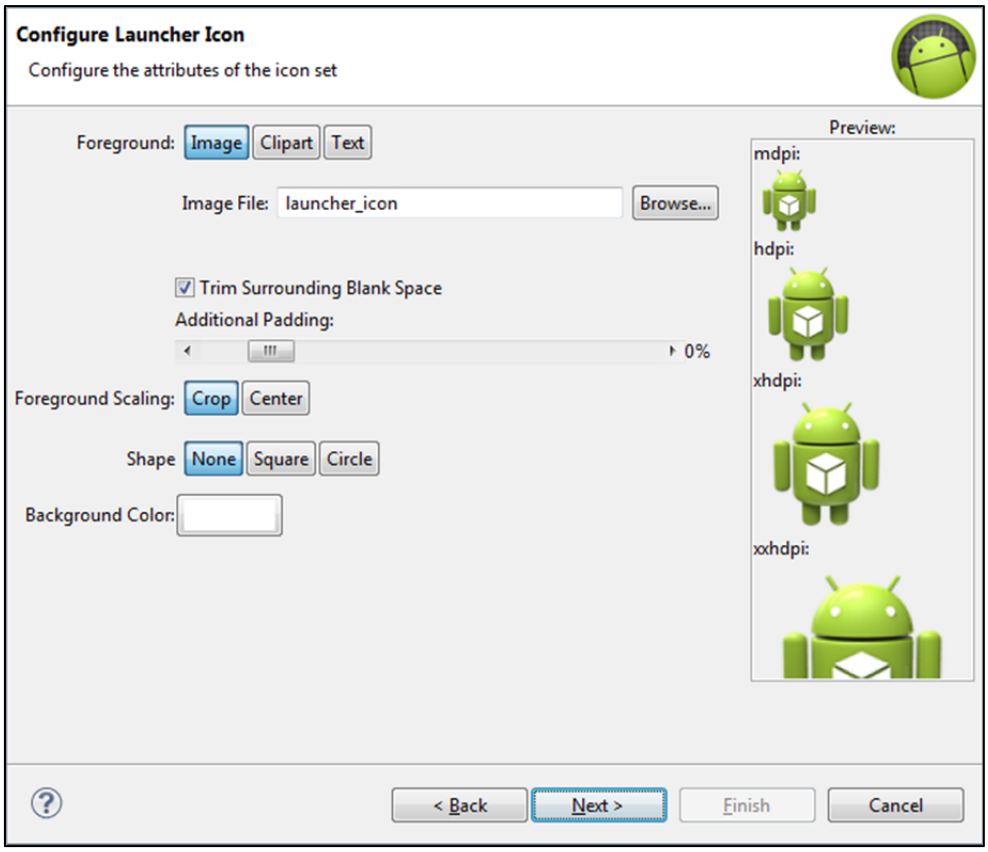

Figura 3.3: Creación de un nuevo proyecto Android III

En la siguiente ventana seleccionamos el *BankActivity*.

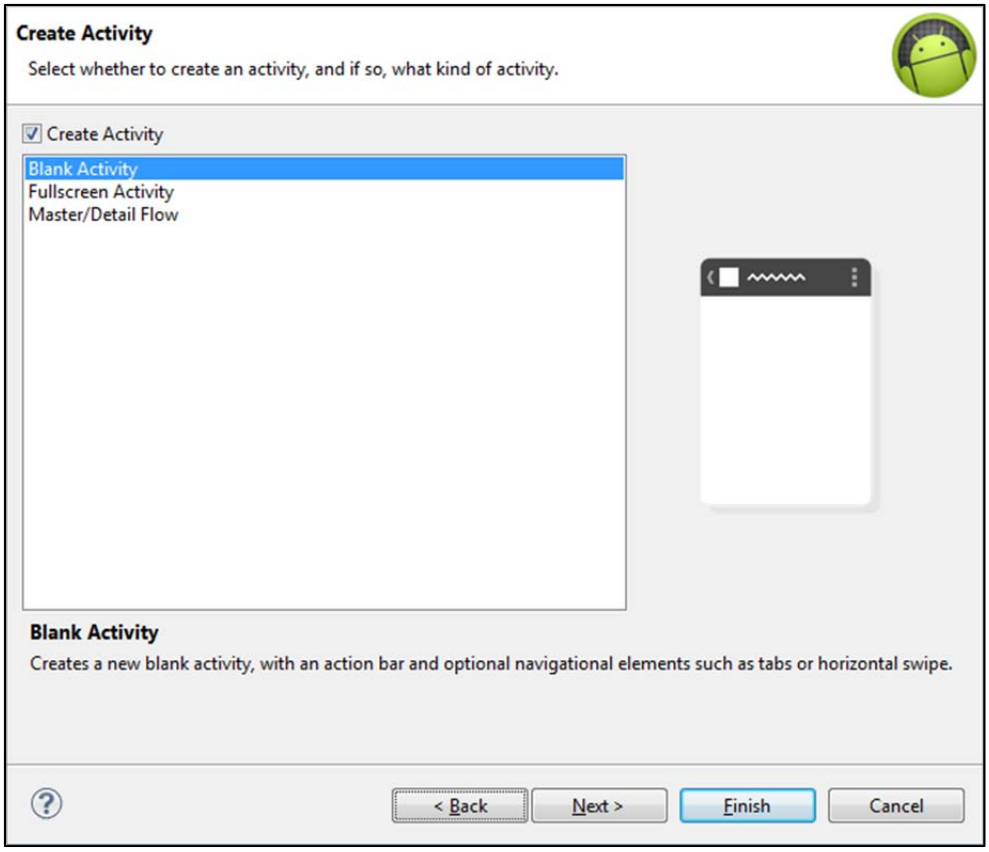

Figura 3.4: Creación de un nuevo proyecto Android IV

Y por último, deberemos de darle un nombre a actividad principal de la aplicación y al fichero XML que va a contener la vista de nuestra primera pantalla de la aplicación, es decir, lo que aparecerá cuando iniciemos nuestra aplicación desde el menú de cualquier terminal.

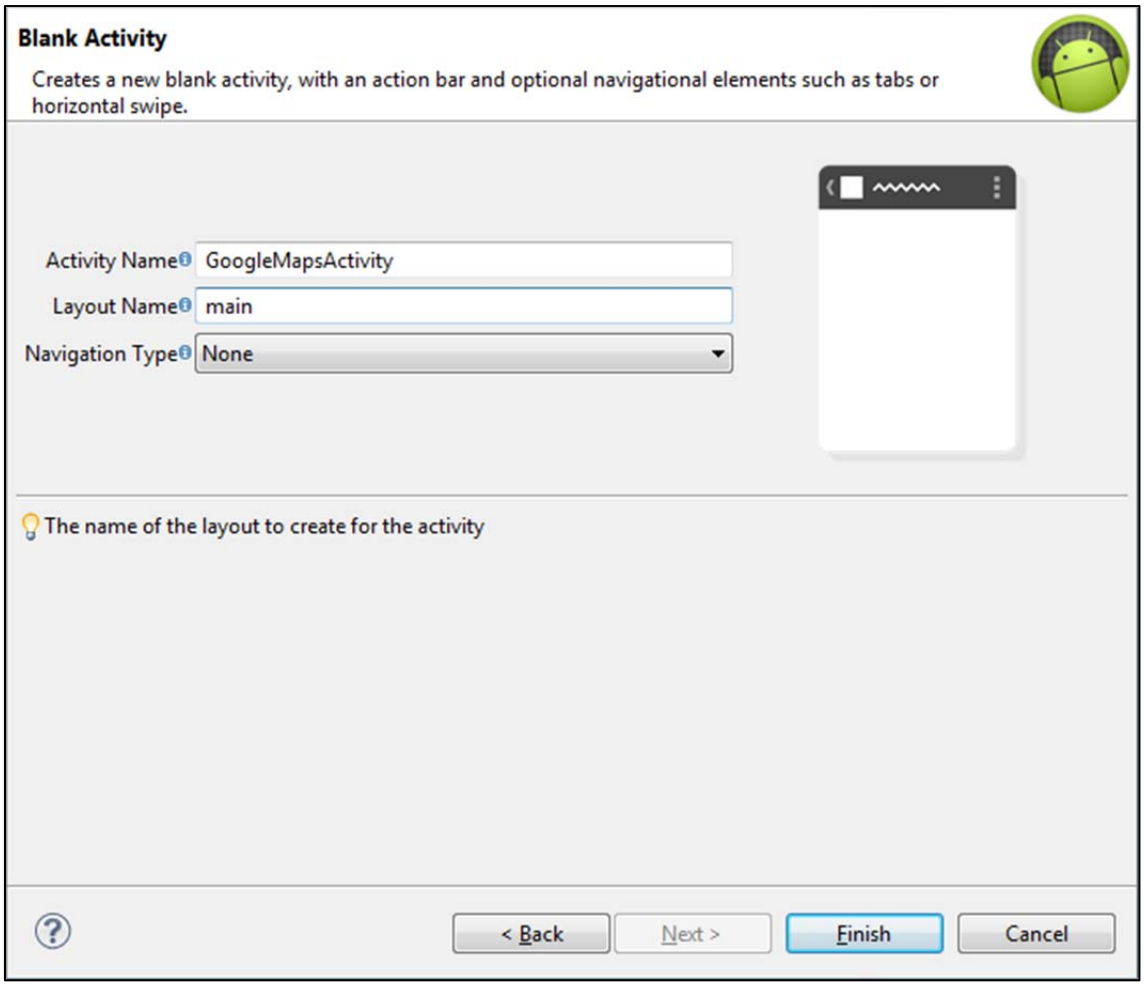

Figura 3.5: Creación de un nuevo proyecto Android V

Una vez creado nuestro proyecto se procederá a explicar detalladamente las carpetas y ficheros más característicos que se han generado.

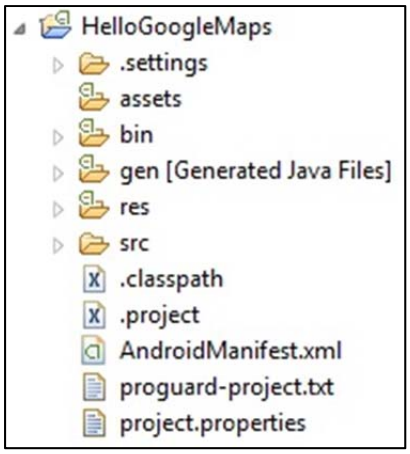

Figura 3.6: Estructura de carpetas de un proyecto Android

- Carpeta *src:* esta carpeta contendrá todos los paquetes que pueda contener nuestra aplicación y en cada paquete habrán 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 *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.
- *AndroidManifiest*: 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 puede ser localización GPS, acceso a internet o utilizar Bluetooth.
- *project.properties*: contiene configuraciones básicas como la versión de la plataforma destino. Se configura automáticamente y no debe ser modificado por el programador.

3.2. Archivos

Para desarrollar una aplicación Android correctamente, será de vital importancia tener una excelente base en programación *java* y *xml*. A lo largo del desarrollo de una aplicación Android, nos encontraremos con números archivos xml (interfaz gráfica) y archivos java (funcionalidad de la aplicación) y, por eso, será conveniente hacer un repaso general de estos dos tipos de archivos.

3.2.1. Archivos XML

Para desarrollar la interfaz gráfica de una actividad se puede hacer de manera más intuitiva y sencilla mediante archivos XML. Cada archivo XML está formado por un árbol de elementos que especifican la manera en la que los elementos de la UI y contenedores se acomodaran para definir la parte visual de un objeto *View.* Para la creación de estos archivos se nos ofrecen dos posibilidades en Eclipse, una de forma visual (*Graphical Layout*) y otra en modo texto (*archivo.xml*).

Figura 3.7: Fichero XML (modo gráfico)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="3"
        android:orientation="horizontal" >
        <com.google.android.maps.MapView
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/map_view"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:enabled="true"
            android:clickable="true"
            android:apiKey="0ji7BrDnf-V5JGktevAs385AQ_wAy0aNfJJrPAA"/>
    </LinearLayout>
</LinearLayout>
```

Figura 3.8: Fichero XML (modo texto)

A continuación, vamos a describir algunos de los elementos que componen una UI, en concreto algunos de los usados en el proyecto. Vamos a analizar el contenido del archivo *main.xml,* que será el archivo donde haremos uso de la API de Google Maps para poder visualizar mapas cada vez que se desee.

En esta parte del capítulo sobre mapas solamente me voy a limitar a mostrar el mapa en la pantalla inicial de la aplicación, dando unas breves pinceladas de las partes de las que consta el archivo XML que lo contiene.

Para incluir un mapa de Google Maps en una ventana de nuestra aplicación utilizaremos el control *MapView*. Estos controles se pueden añadir al *layout* de la ventana (en mi caso, *main.xml*) como cualquier otro control visto hasta el momento, tan sólo teniendo en cuenta que tendremos que indicar la clave de uso de Google Maps en su propiedad `android:apiKey` como se muestra en la figura anterior.

Si observamos la figura anterior, vamos a diferenciar entre dos tipos de elementos: *layouts* y componentes. Los *layouts* describen una superficie específica de la pantalla, configurada por el desarrollador, en la que se incluirán los distintos componentes. En nuestro archivo tenemos dos contenedores del tipo *LinearLayout,* donde el segundo de ellos contiene el *MapView*. Para uno de estos elementos se han definido una serie de propiedades entre las que vamos a destacar `android:id`. Esta propiedad nos permite definir un nombre para dicho elemento por medio del cual esté será accesible desde el código. Posteriormente veremos todo esto en ejemplos con código Java. También se pueden apreciar varias propiedades como `android:layout_width` o `android:layout_height` donde se indicara el ancho y largo de interfaz de la aplicación. Si ponemos a true la propiedad `android:clickable` podremos interactuar con el mapa mediante gestos con los dedos.

3.2.1.1. AndroidManifiest

Este es un archivo XML que estará en todas las aplicaciones Android. En él se declaran todas las especificaciones de nuestra aplicación. Cuando hablamos de especificaciones nos referimos a las *Activity, Service, Intents*, bibliotecas, el nombre de la aplicación, el hardware que se necesitará, los permisos de la aplicación, etc…

Ahora vamos a ver los principales *tags* del manifiesto:

- *<manifiest>: 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.

- *<application>: tag* que contiene todas las *Activities, Services, Providers, Receivers* y las bibliotecas que se usan en nuestra aplicación.

- *<supports-screens>: tag* utilizado para describir las pantallas soportadas por nuestra aplicación.

- *<uses-permissions>:* mediante este *tag* especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse, además son los que deberá aceptar el usuario antes de instalarla. Por ejemplo, si se desea utilizar funcionalidades con Internet o el vibrador del teléfono, hay que indicar que nuestra aplicación requiere esos permisos.

- *<uses-sdk>: tag* utilizado para determinar las distintas versiones Android que va a utilizar nuestra aplicación, tanto sobre qué versiones va a correr como qué versión fue utilizada para realizar nuestras pruebas. Mediante el atributo `android:minSdkVersion` establecemos a partir de qué versión de Android nuestra aplicación podrá correr.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.pitiHellin.android.GoogleMaps"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:allowBackup="true">
        <uses-library android:name="com.google.android.maps" />

        <activity
            android:name=".GoogleMapsActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name= ".BroadcastChatThread"></activity>
        <service android:enabled="true" android:name=".BroadcastChatService"/>
    </application>
</manifest>
```

Figura 3.9: *AndroidManifiest.xml*

3.3. Funcionamiento e implementación de clases

En este apartado se verán las clases que componen la aplicación y los aspectos más relevantes en la implementación java.

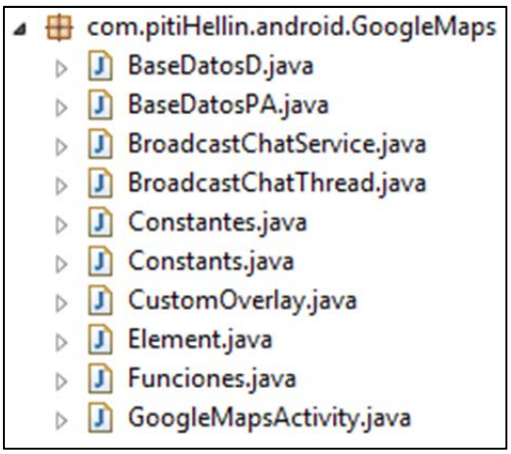

Figura 3.10: Estructura de clases

- *GoogleMapsActivity.java*

  Esta clase será la encargada de visualizar el mapa (haciendo uso de la API de Google Maps), los dispositivos WiFi detectados, los puntos de acceso o mi propio dispositivo. Para poder dibujar en la misma vista el mapa, los dispositivos WiFi, los puntos de acceso o mi dispositivo, se podrá realizar añadiendo capas con la clase *CustomOverlay.java*. Como las coordenadas de los dispositivos WiFi detectados o de los puntos de acceso se conocen de antemano, representarlos en el mapa se realizará de forma trivial llamando a la función `addOverlay()`.

```
for(int i = 0; i<Constants.GEO_POINTS.length;i++){
    itemizedOverlay = new CustomOverlay(this.getResources().getDrawable(R.drawable.red_pin),
            this.getBaseContext());
    itemizedOverlay.addOverlay(new OverlayItem(Constants.GEO_POINTS[i], "", ""));
    mapOverlays.add(itemizedOverlay);
    mapView.invalidate();
}
```

Figura 3.11: Uso de la función *addOverlay()*

La representación en el mapa del dispositivo que recibirá los datos se realizará por triangulación con los dispositivos WiFi o puntos de acceso detectados. Se implementó un hilo (solo se detendrá cuando se pare la aplicación) que comprueba cada cinco segundos si ha detectado algún dispositivo WiFi o punto de acceso nuevo para así poder variar mi posición o no.

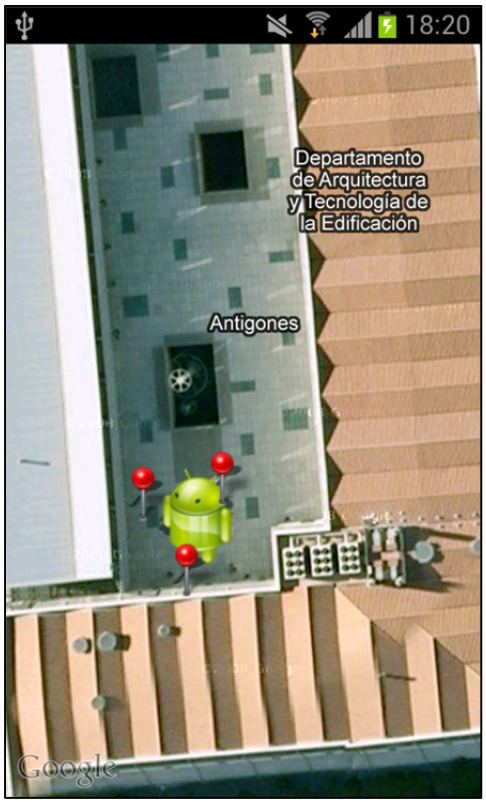

Figura 3.12: Visualización de la pantalla principal de la aplicación

En esta clase también se ha implementado un menú que estará compuesto de tres botones con una funcionalidad vital en nuestra aplicación. Uno de ellos se encargará de activar el servicio para poder recibir datos, otro parará el servicio para que no pueda recibir nada y, el último botón, se encargará de mandar un mensaje broadcast a todos los dispositivos conectados en la misma red que el dispositivo.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.startRx:
            if(BroadcastChatService.getActivo()!=true){
                mStart = new Start();
                mStart.start();
            }
            return true;

        case R.id.stopRx:
            if(BroadcastChatService.getActivo()){
                mStop = new Stop();
                mStop.start();
            }
            return true;

        case R.id.smsBroadcast:
            sms = new String("Mensaje Broadcast");
            try {
                sendMessage(sms);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return true;
    }
    return false;
}
```

Figura 3.13: Implementación del menú

- *CustomOverlay.java*

  CustomOverlay, que extiende a *ItemizedOverlay*, es la clase que permite situar al dispositivo que recibe datos, al dispositivo WiFi o al punto de acceso sobre el mapa. El principal elemento de la clase es `mOverlays`, que será donde se guarde el elemento a posicionar. El método `addOverlay` es el que permitiré modificar esa lista, actualizando así con cada nueva posición. La sentencia `populate()` es la responsable de pintar los puntos sobre el mapa.

- *Constants.java*

  Esta clase contendrá las coordenadas de los dispositivos WiFi que ya conocíamos previamente. Se usaran los *GeoPoint* porque nos permitirán construir un punto para localizarlo en el mapa fácilmente, es decir, incluye las coordenadas geográficas (latitud y longitud).

- *BroadcastChatService.java*
  Esta clase será un servicio ejecutado en segundo plano el cual se encarga de crear las bases de datos que necesita la aplicación, crear un objeto de la clase *BroadcastChatThread()* e inicializarlo. También contendrá el método *insertarBD()* al cual tendremos que llamar cada vez que sea necesario insertar algún dispositivo en la base de datos.

- *BroadcastChatThread.java*
  Esta clase contendrá el hilo más importante de la aplicación si hablamos de la comunicación con los dispositivos WiFi. Una vez que es creado el constructor de esta clase, creamos el hilo `ComThread()` donde, haciendo uso de las funciones `getBroadcastAddress()` y `getLocalAddress()`, se obtendrá la dirección broadcast de la red que nos proporciona servicio de red y la dirección IP Local respectivamente. Se hace uso de estas funciones para poder recibir datos solamente de los dispositivos que están conectados a la misma red que el terminal. Otra utilidad muy importante de la función `getBroadcastAddress()` es, enviando un mensaje broadcast a todos los dispositivos de la red, comprobar cuál de ellos está disponible. Los dispositivos que reciban dicho mensaje broadcast estarán preparados para enviar los datos oportunos al terminal. En la inicialización del hilo `ComThread()`, abriremos un *socket* UDP para poder transmitir paquetes desde un dispositivo WiFi detectado hasta el terminal. Se escogió el protocolo UDP porque no era necesario establecer conexión entre el terminal y el dispositivo WiFi y también se pueden enviar mensajes a varios receptores a la vez, aunque yo no hice uso de esta funcionalidad.
  Una vez que sea iniciado el hilo `ComThread()`, el método se mantendrá bloqueado hasta que no reciba ningún paquete. La función empleada para recibir paquetes mediante el protocolo UDP será `mSocket.receive(packet)`. Esta clase no parará de recibir paquetes hasta que no se parece el servicio y mientras se sigan recibiendo paquetes, se harán uso de una función capaz de obtener la dirección IP del dispositivo que mandó los datos y otra función para poder obtener los datos. Para poder insertar la IP del dispositivo y los datos en la base de datos, se hará uso de los *ContentValues* que se encargarán de retener los valores (IP y Datos) en un solo registro para que a la hora de insertar en la base de datos sea mucho más sencillo.

- *Funciones.java*
  Esta clase contiene las funciones *insertarBD(), insertarBDPA(), guardarDatosBD(), leerBDexterna()* las cuales detallare a continuación.
  La función *insertarBD()* tendrá como funcionalidad exclusiva almacenar los dispositivos en la base de datos creada para ello. Obviamente, si el dispositivo no existe en la base de datos, lo creara. Análogamente, la función *insertarBDPA()* se encargará de almacenar los puntos de acceso en una base de datos creada para ello.
  La función *guardarDatosBD()* ira guardando los datos recibidos de los distintos terminales en la base de datos creada para ello (*BaseDatosD.java*). Realizará

una breve comprobación de la IP del terminal que manda los datos para saber en qué fila habrá que almacenar los datos.
Con la función *leerBDexterna(),* leeremos de una base de datos externa las direcciones MAC de los puntos de acceso que ya conocemos previamente. Para crear la base de datos externa use un software para PC llamado *SQLite Database Browser*. Una vez creada, se depositará en cualquier parte de la memoria del terminal para, posteriormente, poder leerla desde la función *leerBDexterna()* haciendo uso de otra función (`openDatabase()`) facilitada por la librería de *SQLiteDatabase*. A la función `openDatabase()` se le pasara como parámetro la ruta donde hemos guardado la base de datos creada por el PC.

- *Element.java*
  Esta clase ha sido creada para poder acceder a la información de cada punto de acceso ya que han sido creadas dos variables para guardar el nombre del punto de acceso y su dirección MAC.

- *BaseDatosD.java*
  Esta clase ha sido creada exclusivamente para almacenar los dispositivos WiFi detectados y que se han comunicado con el terminal.

- *BaseDatosPA.java*
  Esta clase ha sido creada para almacenar las puertas de acceso a las cuales se va conectando el terminal conforme se va desplazando por el mapa.

- *Constantes.java*
  En eta clase se identificarán las columnas de las dos bases de datos con las que trabajará la aplicación (*BaseDatosD.java* y *BaseDatosPA.java*).

3.4. Descripción de las tablas de la BBDD

En primer lugar, la aplicación dispondrá de una base de datos donde se almacenarán todos los dispositivos que han transmitido cualquier tipo de datos. Esta tabla contendrá el campo ID, IP y Datos.

- ➔ El campo ID es obligatorio para poder asignarle un identificador único a cada dispositivo.
- ➔ El campo IP es bastante importante porque nos aportará la información necesaria para saber que dispositivo nos mando los datos.
- ➔ El campo Datos ha sido creado para almacenar el mensaje enviado por el dispositivo WiFi para su posterior uso.

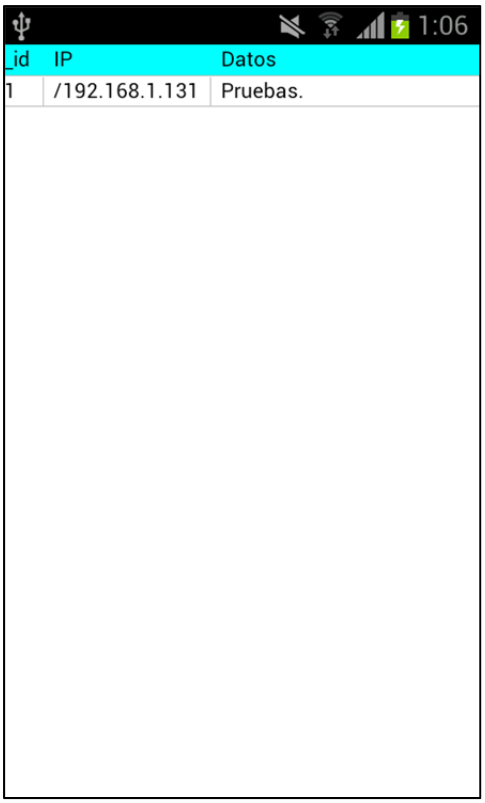

Figura 3.14: Tabla de los dispositivos WiFi

En segundo lugar, la aplicación dispondrá de una base de datos donde se almacenarán todos los puntos de acceso el terminal se irá conectando conforme se vaya desplazando por el mapa. Esta tabla contendrá el campo ID, SSID y MAC.

- ➔ El campo ID es obligatorio para poder asignarle un identificador único a cada dispositivo.
- ➔ El campo SSID será el nombre de la red que proporcionará el servicio de red.
- ➔ El campo MAC se almacenará la dirección MAC de cada punto de acceso.

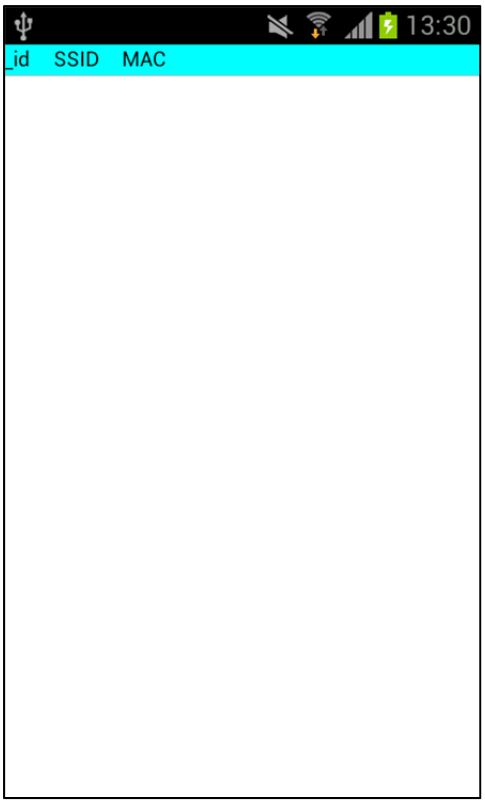

Figura 3.15: Tabla de los puntos de acceso

CAPÍTULO 4

MANUAL DE USUARIO

4.1. Instalación de la aplicación

Disponemos de tres opciones para instalar la aplicación:

- *Android Market*
  Todas las aplicaciones que se desarrollan en Android pueden alojadas en el Market (tienda virtual de Android). Si la aplicación *HelloGoogleMaps* ha sido subida al Market, simplemente hay que buscar la aplicación, seleccionarla y aceptar la instalación. Se instalara 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 Android Market.
- 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*. Solo tendremos que pinchar en la opción de Eclipse *Run as* y seleccionar nuestro dispositivo. Esta opción es la más usada porque permite a los desarrolladores analizar todas las partes de la ejecución de la aplicación para 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

Una vez instalada la aplicación, se procederá a ejecutarla para ver todas las funcionalidades en un caso real. Para ello haremos uso de un terminal con la aplicación *HelloGoogleMaps* instalada y otros dos terminales (dispositivos WiFi) en los se ha

tenido que implementar otra aplicación Android muy sencilla que se dedicará exclusivamente a recibir mensajes broadcast y, una vez recibido ese mensaje broadcast, podremos enviar datos al terminal geolocalizado.

Habrá que buscar entre las aplicaciones instaladas en el terminal y ejecutar la aplicación *HelloGoogleMaps*, que tendrá un icono tal que así:

Figura 4.1: Icono de la aplicación *HelloGoogleMaps*

Una vez ejecutada la aplicación, se localizarán en el mapa tres dispositivos WiFi de los cuales solo transmitirán datos dos de ellos. No habría ningún problema si fuera necesario realizar pruebas con más dispositivos, solamente deberíamos añadir sus coordenadas en la clase *Constants.java*.

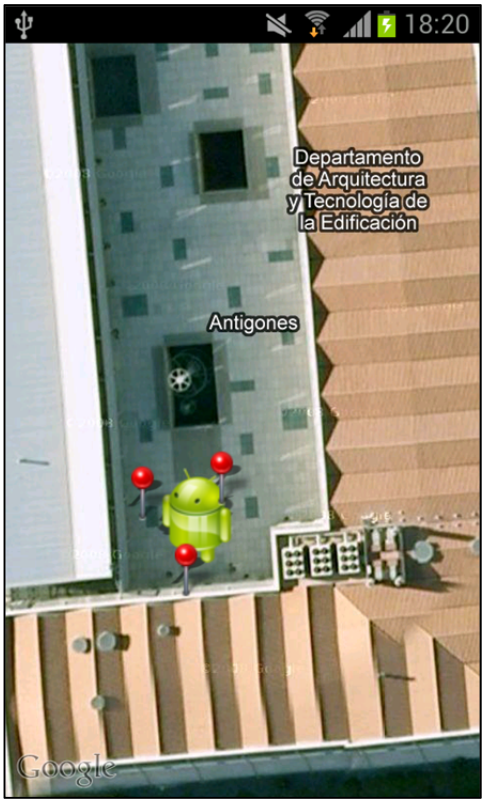

Figura 4.2: Visualización de la pantalla principal de la aplicación

En primer lugar, se activará el servicio ejecutado en segundo plano que mantendrá a la espera el terminal hasta que reciba algún dato de cualquier dispositivo WiFi localizado. Para activar el servicio habrá que desplegar el menú que se implementó en

la aplicación y seleccionar el botón “Recibir”. Si no activamos el servicio, el terminal nunca recibirá nada.

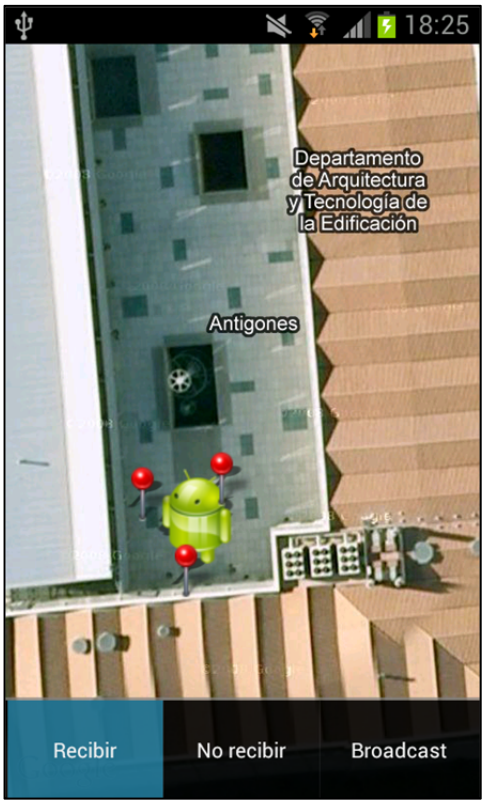

Figura 4.3: Iniciar el servicio

Para comprobar cuál de esos dispositivos está activo y está preparado para mandar datos al terminal, mandaré un mensaje broadcast. Para ello habrá que desplegar el menú que se implementó en la aplicación y seleccionar el botón “Broadcast”. Al pulsar ese botón mandará una cadena de texto a todos los dispositivos WiFi conectados a la misma red que el terminal.

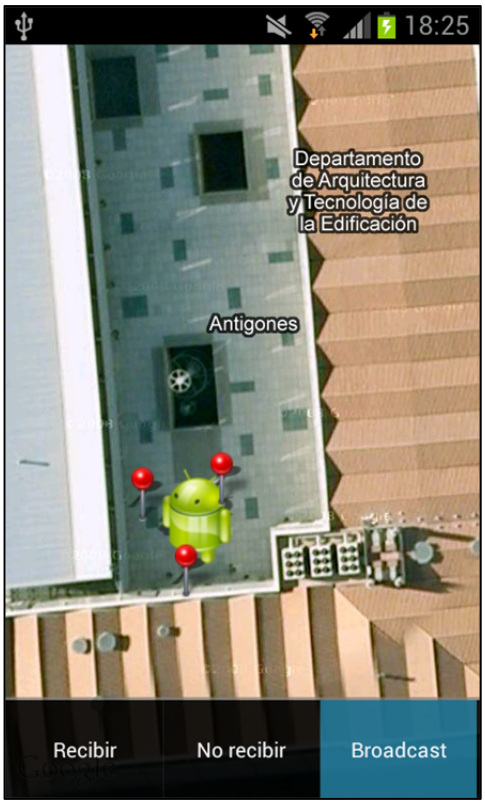

Figura 4.4: Mandar mensaje broadcast

Para que los dispositivos WiFi estén activos y puedan recibir el mensaje broadcast, deberán tener ejecutada la aplicación implementada exclusivamente para hacer pruebas que se comentó al comienzo del capítulo.

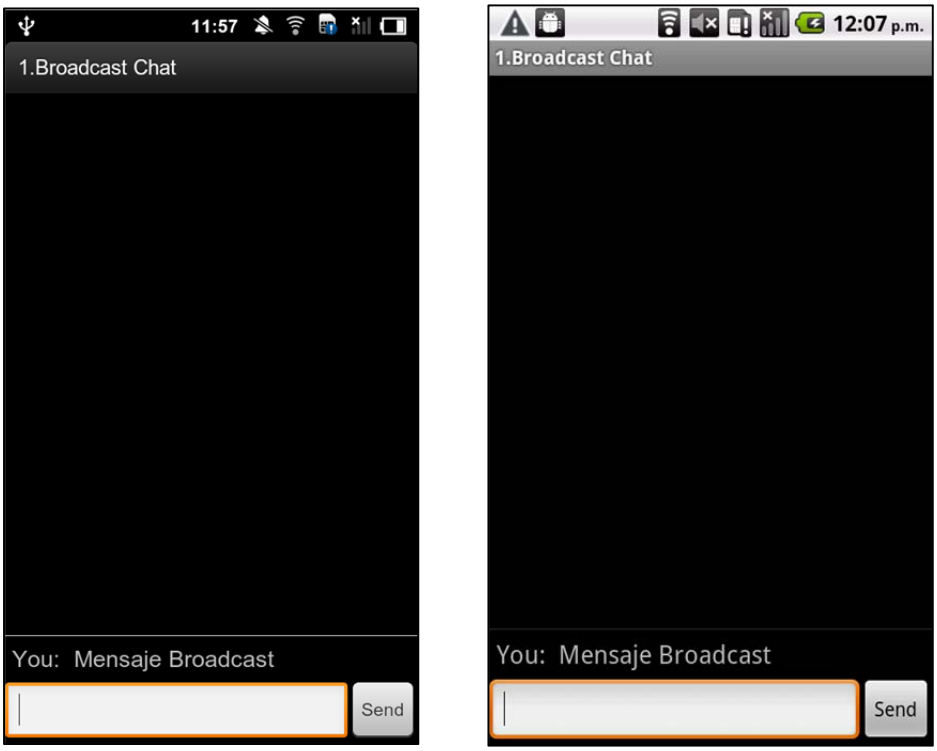

Figura 4.5: Mensaje broadcast recibido en el terminal Motorola y ZTE

Una vez que los dispositivos activos han recibido el mensaje broadcast, se dispondrán a mandar un paquete al terminal. El propio terminal se encargará de procesar ese paquete para sacar la IP y los datos del dispositivo que se lo envió. Como estamos usando el protocolo UDP, podrían mandar paquetes todos los dispositivos a la vez que no habría ningún problema para que el terminal los recibiera correctamente.

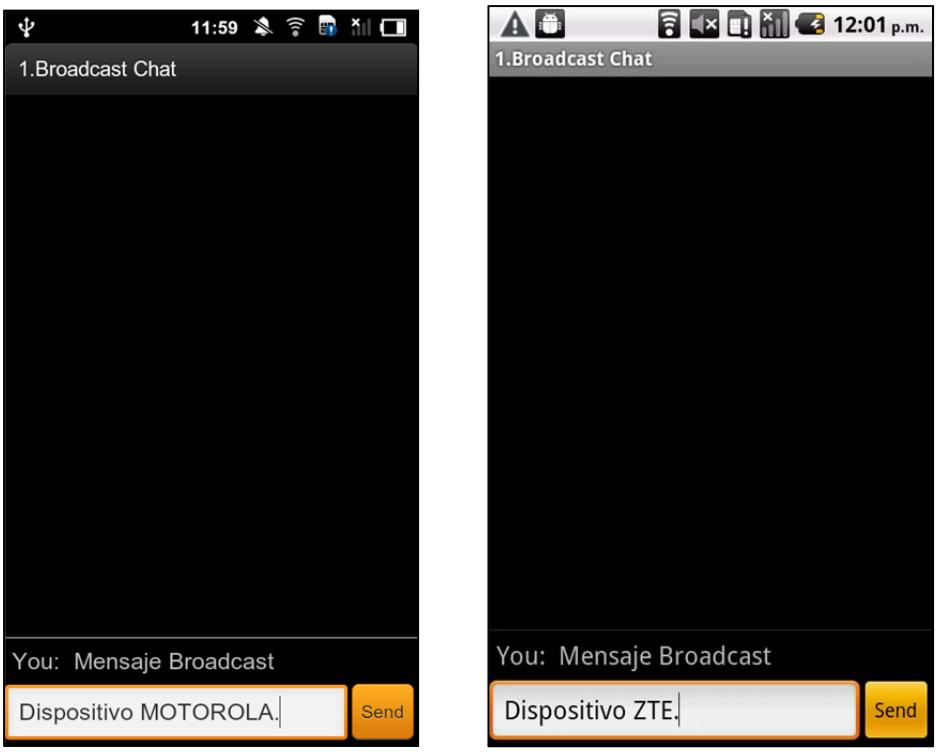

Figura 4.6: Mandar datos desde los terminales Motorola y ZTE

Una vez que los dispositivos WiFi han mandado todos los paquetes que tenían que mandar, es muy importante parar el servicio ejecutado en segundo plano porque puede suponer un consumo de recursos en el terminal bastante considerable. Para ello habrá que desplegar el menú que se implementó en la aplicación y seleccionar el botón “No Recibir”. Una vez hecho esto, el terminal no podrá recibir absolutamente nada de los dispositivos WiFi.

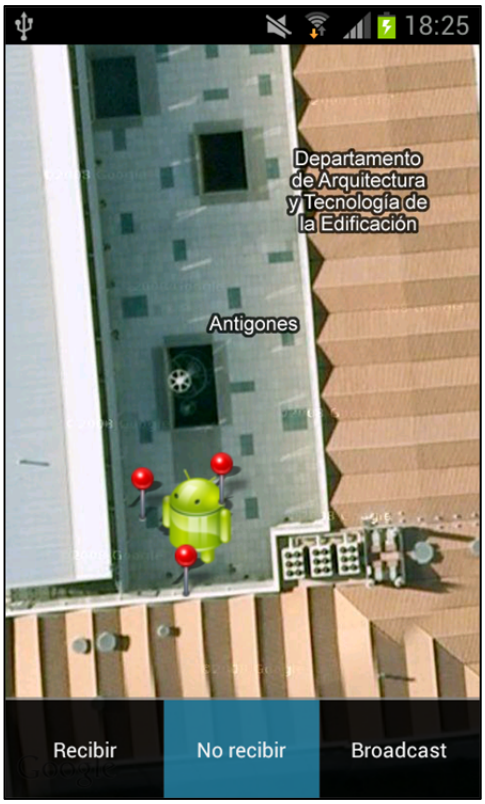

Figura 4.7: Parar el servicio

Para comprobar que se ha recibido y almacenado correctamente el paquete en la base de datos, se hará uso de una aplicación Android para poder visualizar las bases de datos creadas en mi terminal. A parte de poder visualizar las bases de datos creadas en nuestro terminal, también tendremos la posibilidad de modificarlas siempre y cuando tengamos permisos para ello. Esta aplicación recibe el nombre de *SQLite Editor*.

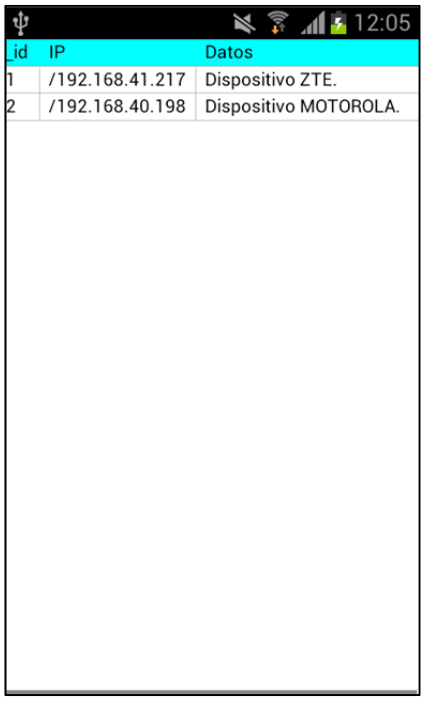

Figura 4.8: *BaseDatosD.java*

CAPÍTULO 5

CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Conclusiones

Como conclusión general, mencionar que este proyecto ha conseguido alcanzar todos los objetivos que se propusieron inicialmente.

En primer lugar, se consiguió satisfactoriamente desarrollar la aplicación haciendo uso de las herramientas ofrecidas por la plataforma Android. Se logró obtener una versión de la aplicación donde se encuentran operativas sus funcionalidades más importantes y muestra la potencia de esta plataforma.

Adicionalmente, se obtuvieron grandes conocimientos en el manejo de las tecnologías relacionadas con un proyecto de este tipo. Se aprendieron conceptos importantes para el desarrollo de software dentro de las limitaciones típicas de un dispositivo móvil. También se obtuvieron conocimientos avanzados en el uso del lenguaje de programación y un buen manejo de las diferentes APIs que hacen posible desarrollar aplicaciones bajo la plataforma Android.

Se comprendieron los elementos más importantes a tener en cuenta al momento de diseñar e implementar una GUI que permita a los usuarios navegar a través de la aplicación de una manera fluida e intuitiva. De hecho, es ésta una de las principales virtudes de esta versión de esta aplicación; una interfaz gráfica muy atractiva.

Por otra parte, a medida que la aplicación fue tomando forma, se implementaron funciones adicionales al diseño original. Esto sucedió ya que se notaron algunas deficiencias y posibles mejoras que no representaban grandes inconvenientes desde el punto de vista de implementación, pero otorgaban considerables mejoras en la usabilidad general de la aplicación.

5.2. Líneas futuras

Este proyecto deja una gran variedad de posibilidades de ampliación y se pueden abrir varias líneas de investigación y desarrollo, desde la mejora de la aplicación presentada como la implementación en dispositivos reales. A medida que aparezcan nuevos SDK con funcionalidades mejoradas y nuevas, las posibilidades se irán ampliando, pudiendo desarrollar aplicaciones bastante más completas pero a la vez bastante más complejas. Algunas de estas líneas futuras pueden ser:

- Mejorar la realidad aumentada. La aplicación podría representar en el plano la posición de los dispositivos localizados, quizás mostrando cierta información sobre ellos.
- Mejora de la realidad aumentada II. También podría mostrar en el plano los dispositivos no visibles desde su posición.
- Aumentar el número de tecnologías empleadas. Actualmente se ha realizado este PFC con wifi. Sin embargo, se está realizando en paralelo una versión similar en Bluetooth. Sería interesante que la misma aplicación emplease todas las tecnologías posibles: Bluetooth, WiFi, RFID, NFC, etc.
- Vistas guidas. Una evolución de esta aplicación puede ser emplear los conceptos de ésta para crear una aplicación que geolocalice el dispositivo dentro de un edificio y le ofrezca una visita guiada al mismo.

CAPÍTULO 6

BIBLIOGRAFÍA

[1] Web oficial para desarrolladores Android

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

[2] Comunidad de desarrolladores I (proyectos *open source*)

http://www.codeproject.com/

[3] Comunidad de desarrolladores II

http://stackoverflow.com/

[4] Curso de programación Android

http://www.sgoliver.net/

[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

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