ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería Informática

Geolocalización de puntos de Interés en Android

Realizado por
Manuel Fernández Ruiz

Dirigido por
José Ramón Portillo

Departamento
Matemática Aplicada I

Sevilla, (12/2010)

Resumen

Esta documentación corresponde a la memoria de Proyecto Fin de Carrera basado en geolocalización de puntos de interés en un terminal móvil con sistema operativo Android, desarrollado para la titulación de Ingeniería Informática en la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla.

Se incluye, además de una introducción al sistema operativo Android, todos los pasos realizados para lograr el objetivo, desde el nacimiento de la idea hasta completar su desarrollo, incluida la publicación en el Market de Android y la puesta a disposición de todos los usuarios.

A esta documentación se adjunta el código fuente de la aplicación, asi como su ejecutable, convenientemente firmado en versión Release y listo para ser instalado en un terminal Android.

Agradecimientos

A Karol Unterharnscheidt, Maria Heredero Fernández, Caterina Monari, Gonçalo Bebiano Caetano y Jolien Luckx, por su ayuda en las traducciones; a María Dolores Ruiz Isac, por su opinión linguística; y a todos los que me han aguantado a mí y a mi ausencia durante el tiempo de dedicación a este proyecto.

Igualmente, sería injusto no recordar a todas aquellas personas que me han apoyado en mis años de estudios de esta titulación. Gracias a todos.

Índice general

Índice de cuadros

Índice de figuras

Índice de código

CAPÍTULO 1

Definición de objetivos

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 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 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 HTC WildFire, aunque gracias a la comunidad de usuarios, también se posibilita la documentación de experiencias en dispositivos de otros modelos y/o fabricantes.

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.

CAPÍTULO 2

Introducción a Android

Android es una plataforma abierta de software para dispositivos móviles. Está desarrollada por la Open Handset Alliance, compuesta por mas 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. Permite que un dispositivo pueda ejecutar varias maquinas 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

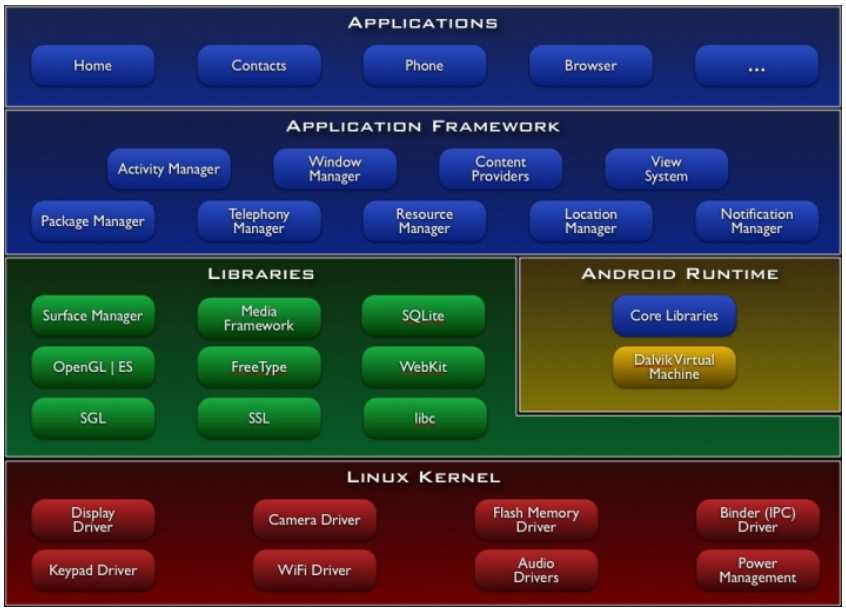


Figura 2.1: Arquitectura de Android

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 aun 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 del ciclo de vida de las aplicaciones.

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 podriamos agruparlas por conectividad (telefonía GSM, Bluetooth, EDGE, WIFI, 3G), formatos mutimedia soportados (MPEG4, H.264, MP3, OGG, AAC, AMR, JPG, PNG, GIF) y otras características hardware, tales como GPS, brújula, acelerómetro, cámara o pantalla táctil.

En el ámbito de desarrollo, dispone de un complemento para el IDE Eclipse, así como herramientas de simulación y depuración. Las aplicaciones creadas pueden ponense 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.

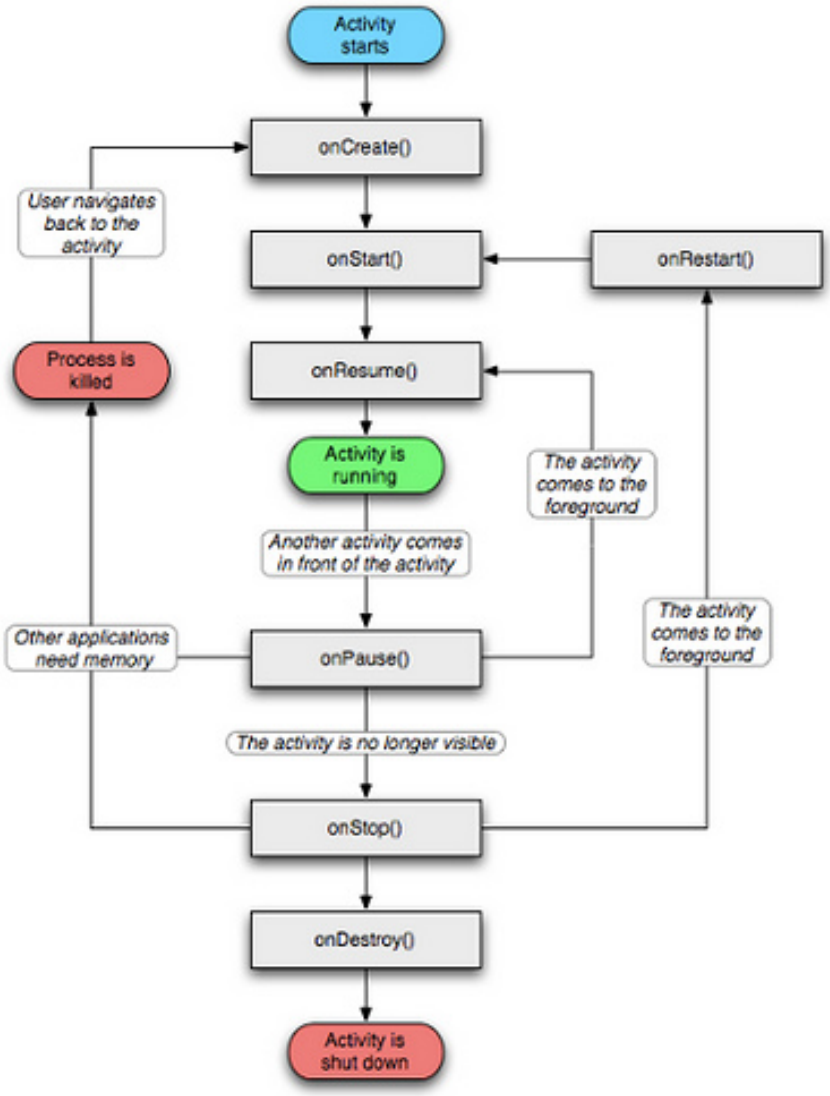


Figura 2.2: Ciclo de vida de una aplicación Android

CAPÍTULO 3

Análisis de antecedentes y aportación realizada

Año tras año, el mercado de la telefonía móvil ha ido evolucionando sin cesar, hasta llegar a un punto en que la telefonía en si no es mas que una de las múltiples características que puede llegar a ofrecer un terminal. Poco a poco, casi cualquier dispositivo portátil ha ido siendo integrado en uno solo que copa todas las necesidades, a las vez que se requería una arquitectura mas potente para darle soporte y un sistema operativo para gestionarlos.

Symbiam, Windows Mobile, iOs o Android han sido algunos de los que se han hecho con una posición significativa en el mercado, cada uno con características distintas, positivas y negativas, respaldados por diversas alianzas empresariales con filosofias distintas que han determinado el mayor o menor éxito de cada uno de ellos.

El último en llegar al mercado ha sido Android, apuesta de Google que ha supuesto su entrada en escena en este mercado tan amplio. Como principales características, además del propio apoyo de la gran compañía que tiene detrás, es el hecho de estar bajo software libre y código abierto, lo que ha propiciado una gran acogida por parte de desarrolladores independientes, dando lugar a una gran comunidad internacional que, en apoyo a la propia documentación oficial disponible, abre puertas a casi cualquier persona que quiera iniciarse en la programación de dispositivos móviles.

En la selección de esta plataforma, además de los motivos ya señalados, ha influido decisivamente el hecho de que el lenguaje de desarrollo sea Java, ampliamente estudiado en la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla. Además del propio SDK (Software Development Kit), Android dispone de un plugin que lo integra perfectamente en el IDE Eclipse, tanto en el ámbito de la programación como el de la simulación, incluyendo todas las funcionalidades de un ter-

minal real, lo que facilita las tareas de testeo y depuración del software.

La aplicación a desarrollar en este proyecto utiliza principalmente el dispositivo de geoposicionamiento y la transmisión de datos, ya sea vía WIFI o mediante la red de datos de la compañía de telefonía disponible, así como la API de desarrollo de la plataforma Google Maps.

La idea surge como un paso más allá de otra planteada anteriormente, en la cual el posicionamiento se llevaba a cabo mediante tags QR Code debido a las limitaciones de la plataforma Symbian, así como el precio de terminales con tecnología para geoposicionamiento. En cualquier caso, el fin último, mostrar información sobre puntos de interés turístico cercanos a nuestra posición, se mantiene intacto, al ser alcanzable mediante multitud de caminos.

Aunque ya existían aplicaciones de similares objetivos, al menos parcialmente, este desarrollo también supondrá el estudio independiente de cada módulo, su interconexión y la gestión de información global. Un todo que conformará la arquitectura de la nueva aplicación.

CAPÍTULO 4

Análisis temporal

Parte importante de todo proyecto recae sobre las características del producto desarrollado, pero éste no es el único punto de interés, pues los tiempos y costes de desarrollo podrían ser inabarcables y dar al traste con el conjunto del proyecto. Aunque en este caso el coste realmente no será un aspecto significativo, si lo será el tiempo empleado, que por otra parte nos puede dar una idea aproximada de los costes.

Hablando con propiedad, este proyecto comienza a desarrollarse a principios de Octubre de 2010, tras hacer una ligera especificación y acordar con el tutor el objetivo de concluir el desarrollo para Diciembre del mismo año. Aunque temporalmente es un periodo aparentemente corto, la disponibilidad de tiempo de dedicación es alta, casi exclusiva, debido a la escasa ocupación en otras tareas que puedan interferir y restar tiempo de trabajo al proyecto.

Sin embargo, la aproximación a la programación de dispositivos móviles, con vistas a un sistema similar a este objetivo, ya comenzó en torno a Marzo de 2010 como tareas de investigación para un intento de proyecto anterior, salvando ciertas distancias. La mayor diferencia de todas ellas es que se desarrollaba en una plataforma Symbian y, como consecuencia, escrito en C++, lo que también suponía el aprendizaje de dicho lenguaje de programación, apenas usado a lo largo de los años de formación en nuestra Escuela de Ingeniería Informática.

Finalmente, debido principalmente a rendimientos académicos no esperados inicialmente y al desarrollarlo forma simultanea a la realización de una beca Erasmus, con los inconvenientes que supone el seguimiento a distancia, el proyecto acabó desechándose tras la elicitación de los requisitos del sistema.

Así, aunque ambos sistemas, Symbian y Android, son completamente distintos e incompatibles, la familiarización con programación para dispo-

sitivos móviles y la propia elicitación de requisitos si han resultado útiles a la hora de embarcarse en este nuevo objetivo.

La primera parte del proyecto corresponde a los diez primeros dias de Octubre, donde se llevaron a cabo principalmente tareas de investigación y formación básica sobre la arquitectura hardware, paradigma del sitema operativo, y las necesidades para desarrollar un nuevo software. Este último aspecto es el más importante, ya que incluye el aprendizaje e instalación de las herramientas proporcionadas para culminar el proyecto.

Es en este momento cuando empieza a escribirse la documentación, o al menos a tomar notas preeliminares, pues la instalación del entorno de desarrollo conlleva detalles fáciles de olvidar, pues probablemente no se volverá a necesitar llevarlos a cabo.

Una vez disponibles, la siguiente fase fue familiarizarse con las herramientas mediante la realización de diversos programas sencillos, siempre orientados a las necesidades actuales. Este proceso se desarrolla aproximadamente durante la semana posterior a la fase formación.

En este punto llega la fase de desarrollo propiamente dicha, donde las aplicaciones que sirvieron de aprendizaje fueron incrementando su complejidad, mejorando su diseño e integrandose con otras unidades hasta llegar al producto final, casi a final de noviembre.

Antes de estar acabada, la aplicación ya habia sido probada parcialmente, pero no es hasta el momento de la finalización cuando se realizan test más específicos, a la vez que tareas de rediseño, mejoras y optimizaciones hasta la entrega definitiva del proyecto concluido.

CAPÍTULO 5

Requisitos del sistema

A partir de este momento vamos a tratar todo lo necesario para preparar nuestro entorno de trabajo y poder empezar con la fase de desarrollo. Para ello, el primer paso es revisar los requisitos. Dichos requisitos, según la propia documentación de Android, son los siguientes:

Sistema Operativo:

- Windows XP (32-bit), Vista (32 o 64-bit), o Windows 7 (32 o 64-bit)
- Mac OS X 10.5.8 o superior (solo x86)
- Linux (probado en Linux Ubuntu Hardy Heron)

Entorno de desarrollo:
Eclipse IDE

- Eclipse 3.4 (Ganymede) o 3.5 (Galileo)
- Eclipse JDT plugin
- JDK 5 or JDK 6
- Android Development Tools plugin (opcional)
- No compatible con Gnu Compiler para Java (gcj)

Otros entornos de desarrollo:

- Apache Ant

CAPÍTULO 6

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.

6.1– 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[1]. En la sección "Download", se elige la versión deseada, que en nuestro caso sera Eclipse IDE for Java Developers para Windows 32 bits, con un tamaño de 99MB.

Una vez descargado, basta descomprimir y ejecutar "eclipse.exe"para comenzar a usar la aplicación.

6.2– Instalación del SDK Android

Entre los numeros recursos disponibles en la web de Android[2] tenemos todas las versiones del SDK, en la sección"SDK". En nuestro caso elegimos la versión para Windows, con un tamaño de 23MB. 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

[1] http://www.eclipse.org
[2] http://developer.android.com

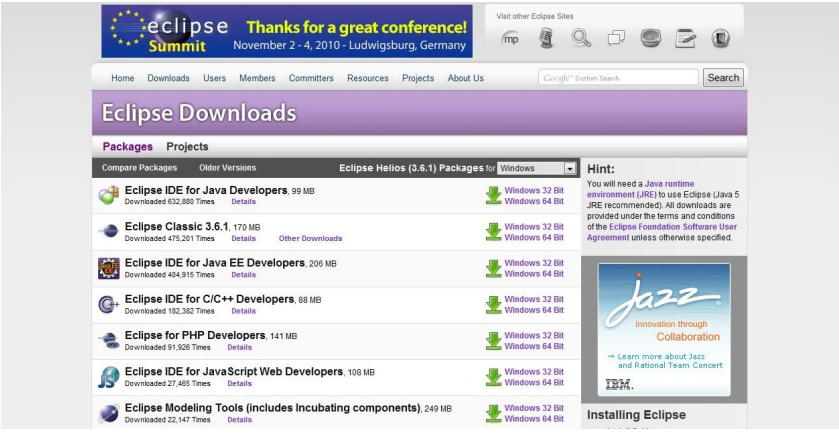


Figura 6.1: Descarga de Eclipse

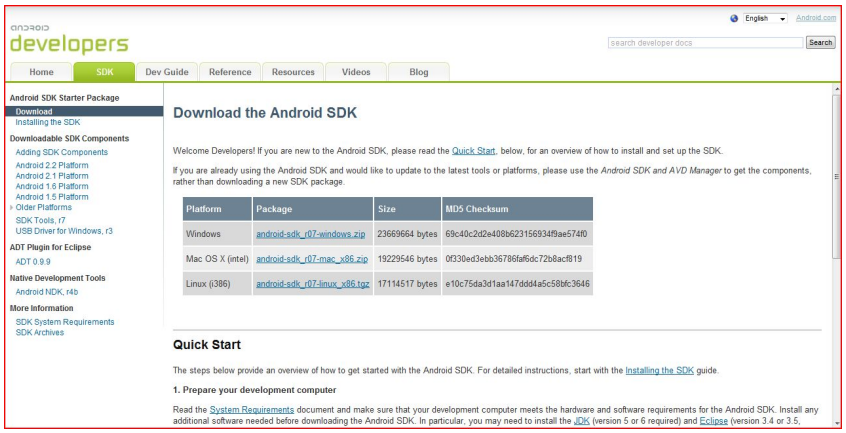


Figura 6.2: Descarga de SDK Android

hacemos clic con el boton 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 falicitará 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.

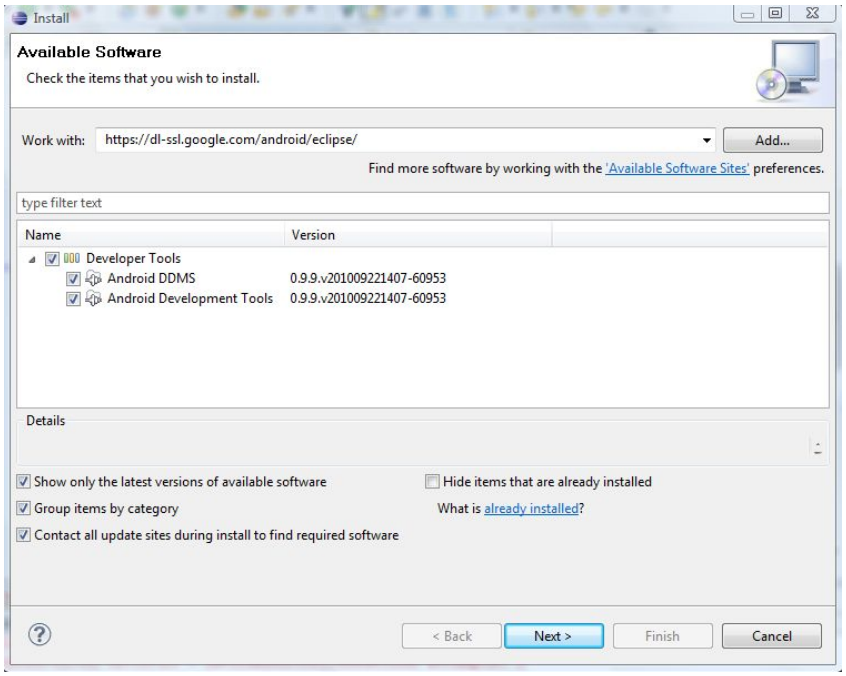


Figura 6.3: Instalación de Android Developer Tools

6.3– 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 [3] 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 arbol podremos ver que se compone de "Anodroid 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:\android-sdk-windows. Si todo el proceso se ha realizado correctamente, tras pulsar el boton "Apply " nos aparecerán todas las versiones de las APIs incluidas en nuestro SDK. Aceptamos los

[3] https://dl-ssl.google.com/android/eclipse

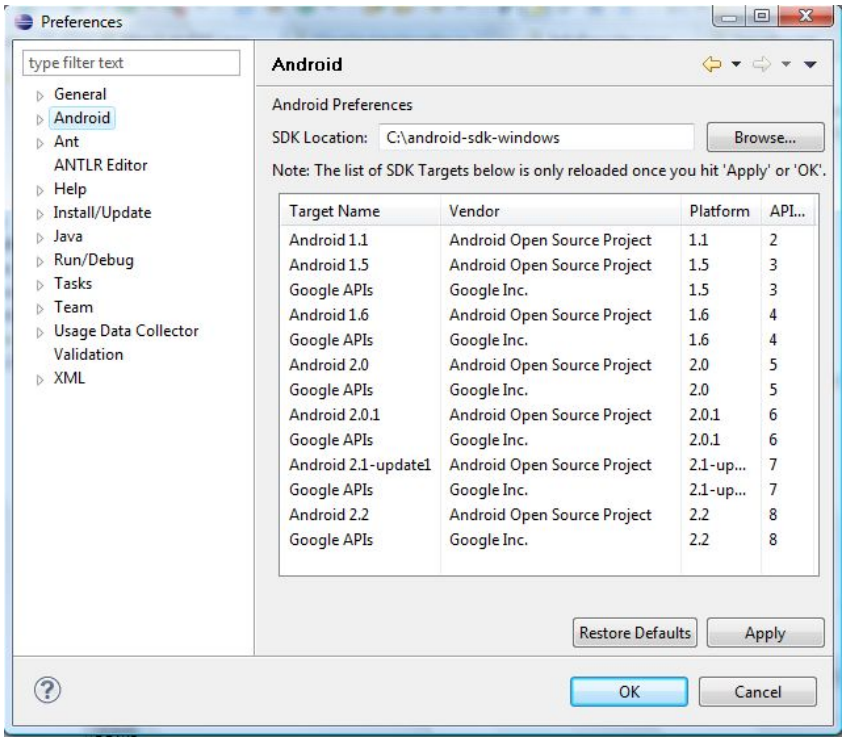


Figura 6.4: Especificar ubicación del SDK

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 mostrará 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.

Opcional. El plugin de desarrollo crea de forma automática una carpeta llamada ".android" dentro de la carpeta "Mis documentos", donde se almacenan, entre otros ficheros de configuración, nuestros dispositivos virtuales creados. Si la configuración de esta carpeta es la definida por defecto en nuestro sistema operativo Windows no tendremos ningún problema, pero si hemos cambiado la ruta de acceso habrá que indicar a

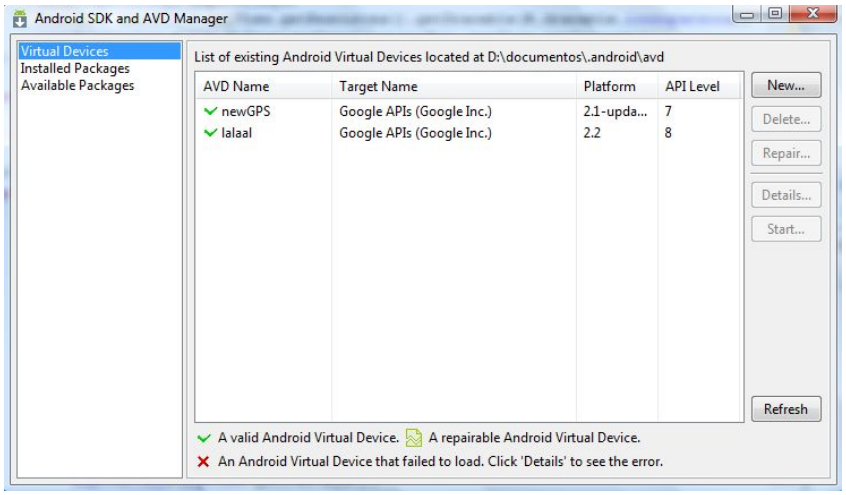


Figura 6.5: ADV Manager

Eclipse y al plugin de Android nuestra ubicación personalizada.

La forma mas sencilla es creando una nueva variable de entorno, procediendo de la misma forma que cuando añadimos a la variable "Path" la ruta de nuestro SDK Android. En este caso la diferencia será que crearemos una nueva variable en lugar de modificar una ya existente. La llamaremos "ANDROID_SDK_HOME", y su valor será la ruta de nuestra carpeta "Mis documentos", por ejemplo "D:\documentos".

Por último, señalar que en caso de que aparecieran nuevas versiones del plugin podremos acceder a ellas mediante la herramienta "Check for updates" del menú "Help".

Si lo que queremos es actualizar el SDK basta con descargar el nuevo paquete y cambiar, si fuera necesario, la ruta de instalación especificada en el plugin de Android.

CAPÍTULO 7

Análisis de requisitos, diseño e implementación

Una vez familiarizados con Android y teniendo preparado el entorno de trabajo, podemos comenzar con el desarrollo de la aplicación en sí.

Además del estudio del sistema Android, también se han desarrollado programas básicos con el objeto de obtener un mayor conocimiento de cada una de las tecnologías que vamos a necesitar, aprendiendo tanto el funcionamiento de las distintas funcionalidades del dispositivo (hardware) como la forma en que Android se comunica con ellos (software).

Aunque no se describirán detalladamente estos pasos previos, si lo serán las conclusiones y aprendizajes obtenidos, a lo que se añadirá información adicional si fuera necesario. En resumen, todo lo necesario para realizar el Análisis de Requisitos.

Teniendo los conocimientos necesarios sobre cada uno de los componentes utilizados, es momento de desarrollar el Diseño de la Aplicación. En él se analizará la estructura elegida para nuestro sistema, justificándola y explicándo la relación de cada clase con los módulos anteriormente expuestos.

Tras una fase teórica y otra teorico-práctica, pasaremos a la práctica, propiamente dicha: el Desarrollo de la Aplicación. Será entonces cuando se comentará con más detalle el código de la aplicación, ahondando en las caracteristicas propias de Android, pudiendo afianzar los conocimientos anteriormente expuestos.

7.1– Análisis de Requisitos

Antes de desarrollar cualquier sistema, ya sea hardware o software, es necesario llevar a cabo un estudio de nuestro objetivo, analizando las funcionalidades a desarrollar y atribuyendolas a cada uno de los modulos que se van a diseñar, encargados del funcionamiento de cada parte de nuestra

aplicación. De esta forma podremos tratar cada módulo por separado para integrarlos posteriormente y atacar la forma en que serán interconectados.

El fin último de nuestra aplicación es falicitar la visita a una ciudad o lugar turístico cualquiera, que, a priori, puede ser un lugar totalmente desconocido por el visitante, bien porque desconozca la existencia de lugares de interés o su ubicación en el callejero de la ciudad. Ambos aspectos quedarán resueltos con esta aplicación.

El primer paso necesario es la localización del usuario sobre el mapa. Esto se llevará a cabo gracias al dispositivo GPS disponible en los terminales Android. Este posicionamiento se realizará sobre un mapa porporcionado por la API Maps para Android de Google. A continuación se lleva a cabo la búsqueda de lugares de interés turístico cercanos a dicha posición. Este último aspecto sufrió varios contratiempos durante la fase de desarrollo, obligando a realizar cambios en el servicio de información inicialmente previsto que serán comentados más adelante.

Debido a que el dispositivo GPS suministra la ubicación mediante coordenadas geográficas, nuestra aplicaión usará como método de posicionamiento tanto las coordenadas como direcciones reales, según en uso que se le vaya a dar en cada caso. Esta transformación se realiza gracias a la API Geocoding.

Todos estos servicios utilizados necesitan conexión con una red dedatos, bien mediante WIFI o mediante la red de nuestro proveedor de telefonia, ya sea 3G o GPRS.

Cada uno de estos componentes comentados serán estudiados a continuación, incluyendo la conexión de datos, aunque de forma más superficial, ya que no será gestionada por nosotros y su uso se realiza de forma indirecta.

Sistema GPS (Global Position System). Es un sistema global de navegación por satélite (GNSS) que permite determinar en todo el mundo la posición de cualquier objeto o persona con un receptor disponible. Proporciona una precisión de unos pocos metros, aunque usando el sistema llamado diferencial podría llegarse incluso a centímetros. El sistema está operado por el Departamento de Defensa de los Estados Unidos.

Existen otros sistemas de posicionamiento, pero GPS es el más extendido. Los más importantes son: GLONAS (GLObal NAvigation Satellite System), desarrollado por Rusia para uso militar, Compass proyecto independiente desarrollado por China, o Galileo, actualmente en desarrollo por la Agencia Espacial Europea.

El sistema lo compone una red de 24 satélites distribuidos en 6 órbitas de 23.200km de altitud, con 4 satélites cada una, que cubren toda la superficie terrestre. De estos 24 satélites, se requiere la señal de, al menos, 3 de ellos. Dicha señal proporciona la posición del satélite y la distancia del receptor. Esta distancia se calcula midiendo retrasos en la propagación de dicha señal, gracias al reloj atómico instalado en los dispositivos.

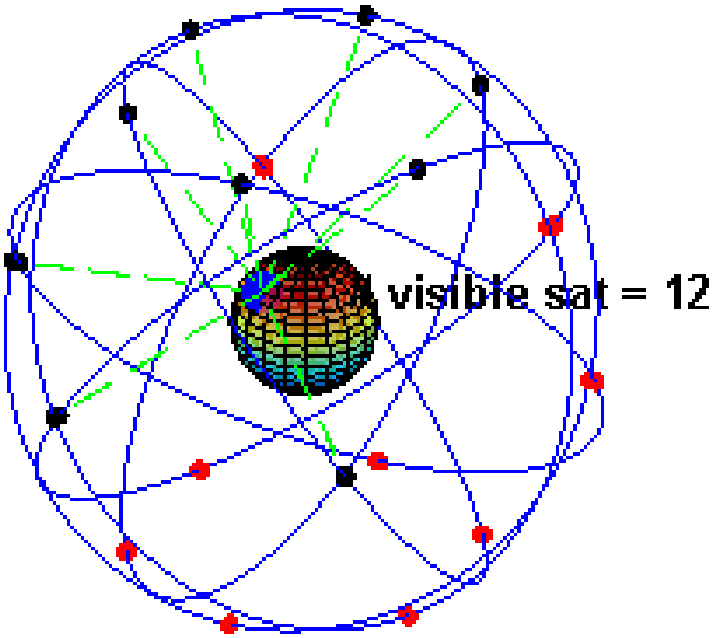


Figura 7.1: Representación de las constelaciones de satélites GPS

Con esos datos se trazan sendas esferas centradas en la posición de cada satélite y de radio la distancia al receptor. La intersección de esas esferas, de ahí la necesidad de 3 satélites, determina la posición del receptor. A priori, un cuarto satélite aportaría datos redundantes, sin embargo, la localización posee cierta incertidumbre debido a la no sincronización de los relojes, por lo que los datos de sucesivos satélites ayudaría a reducir el margen de error existente. Además, las distorsiones produicidas en la señal por las capas atmosféricas también podrían producir errores.

API Google Maps for Android. Porporciona un uso fácil y familiar del servicio Google Maps directamente desde nuestro terminal Android, gracias a la inclusión de la librería externa *com.google.android.maps*. Ofrece una funcionalidad similar al servicio Google Maps via web, permitiendo desplazamientos sobre el mapa, zoom, y otras opciones comunes.

La clase MapView, que hereda de la clase ViewGroup en la API de Android, es el punto de inicio para su inclusión en nuestra aplicación. Se encarga de mostrar el mapa, configurarlo y hacerlo usable. Sobre ella se desarrollará toda la funcionalidad adicional necesaria.

Hay otras clases suministradas por la misma librería que serán de especial interés en nuestra aplicación. Entre otras, GeoPoint nos permite construir un punto para localizarlo en el mapa, incluyendo sus coordenadas geográficas; MapController falicita todas las acciones sobre el mapa

descritas anteriormente; o el conjunto Overlay, que nos permite mostrar ventanas y mensajes en nuestro mapa, por ejemplo con nuestra ubicación actual.

Antes de comenzar a usar esta API se requiere una key para su correcto funcionamiento[1].

Una vez adquirida la posición geográfica y mostrada sobre un mapa, el siguiente paso es obtener los diversos lugares de intrés turístico cercanos a nuestra posición. Como ya se ha comentado, este servicio ha causado problemas y obligado a realizar cambios durante el desarrollo. Hasta tres alternativas han sido estudiadas, las dos primeras tuvieron que ser descartadas para decantarse finalmente por la herramienta GeoNames. A continuación se describen estos tres servicios y los motivos para ser descartados en base a los problemas surgidos.

API Google Places. Places es el servicio que almacena información sobre lugares basandose tanto en el nombre, sector y dirección. Aunque pueden ser obtenidos a través del buscador genérico de Google, su principal objetivo es ser posicionados sobre un mapa.

Esta API tiene un marcado carácter comercial, porque, aunque puede ser incluído todo tipo de lugares por cualquier persona, la información requerida está orientada al ámbito empresarial.

Desde el primer momento, Google Places parecía la herramienta más adecuada para el proyecto, sin embargo, al encontrarse su API aun en fase Lab (en pruebas) es necesario enviar una solicitud para la obtención de la API-Key. En dicha solicitud, además de incluir una descripción de la aplicación a desarrollar, también se necesita un idenficador de Google AdSense, el programa de publicidad de Google, lo que recalca aun más el propósito comercial de la plataforma.

Por desgracia, esta Key no ha llegado a ser concedida. Según Google, en principio solo están interesados en aplicaciones Check-In, por lo que la concesión de licencias de uso es limitada, quizás incluso destinada solo a empresas o desarrolladores profesionales. Por tanto, su uso fue descartado.

API Ajax. Es un conjunto de APIs con un objetivo que permite búsquedas de distinto tipo. Los dos productos más destacados son Web Search y Local Search. Esta última será la encargada de suministrarnos la información requerida para nuestra aplicación: dada una ubicación, extraer los puntos de interés turístico cercanos.

Aunque no es una API especifica para Android y el objetivo principal es usarla en servicios web mediante JavaScript, la documentación incluía soporte para utilizar el cagador de la librería en Java, posibilitando su aprovechamiento para nuestro sistema.

Esta API, dada una ubicación concreta y una palabra clave, nos permite extraer lugares situados en ese entorno, con un máximo de 32 resulta-

[1] Ver Apéndice A

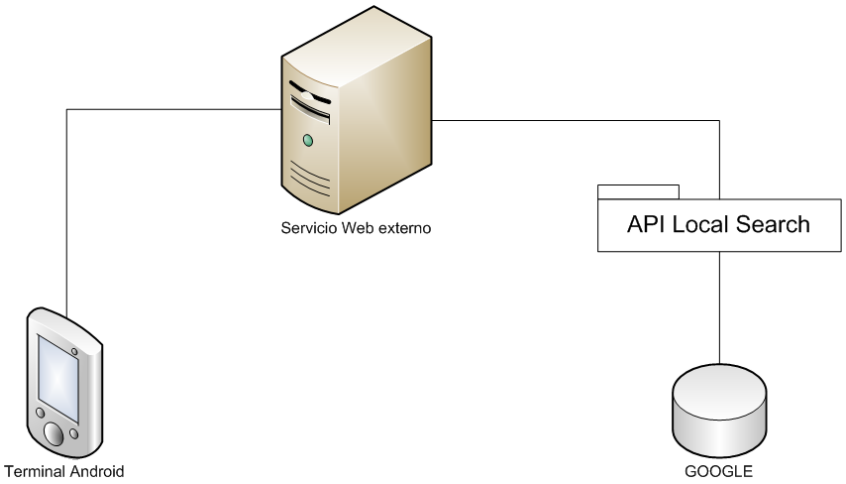


Figura 7.2: Estructura del Servicio Web

dos, los cuales pueden ser indexados mediante XML o JSON. La ubicación será una dirección real extraida de las coordenadas geográficas mediante la API GeoCoder, que estudiaremos más adelante.

Sin embargo, el 1 de Noviembre de 2010, Google realiza una serie de cambios en sus APIs y tanto Web Search como Local Search son declaradas obsoletas[2].

Aunque los servicios de estas APIs siguen estando disponibles, parte de la documentación es retirada de la web de desarrollo de Google. Entre la información descatalogada se encuentra el soporte al cargador Google desde Java, impidiéndo su uso de forma directa en nuestra aplicación.

La alternativa ofrecida por Google pasa por un nuevo servicio : Custom Search. Sin embargo, este servicio no falicita la extración de lugares y su posicionamiento, sino una búsqueda genérica.

Ante el actual panorama, la decisión tomada es seguir usando Local Search, pero no de forma directa desde la aplicación Android, ya que la documentación ha sido retirada, si no por medio se un servicio web auxiliar que haría las veces de intermediario.

Este servicio web consta de una simple aplicación ASP/Javascript que recibe nuestra ubicación mediante una petición HTTP enviada desde nuestro terminal Android, lanza la búsqueda y retorna los resultados a nuestra aplicación para ser procesados.

Una vez puesta en marcha, se comprobó que producía importantes latencias en nuestro sistema debido a las peticiones web necesarias y, quizas, al reducido ancho de banda proporcionado por el hosting gratuito usado

[2] http://googlecode.blogspot.com/2010/11/introducing-google-apis-console-and-our.html

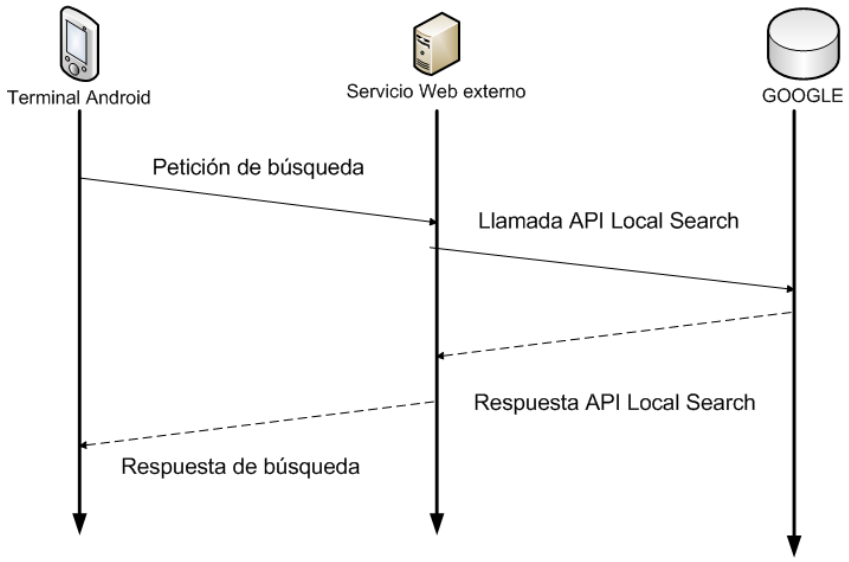


Figura 7.3: Latencia introducida por el Servicio Web

en pruebas: 7Host [3]. Además esta práctica resultó poco elegante y solo funcionaría de forma temporal, pues todo apunta a que finalmente la API dejará de estar operativa.

GeoNames La siguiente alternativa, que finalmente ha sido la adoptada, es GeoNames. Se trata de una base de datos de libre acceso bajo licencia Creative Commons, que nos ofrece igualmente la localización geográfica de lugares y su información asociada existente en la Wikipedia. Dicha base de datos es accesible mediante servicio web o haciendo uso de las APIs publicadas para diversos entornos de desarrollo, entre ellos Java. Sin embargo, dada la simplicidad requeridas con las consultas en comparación con el conjunto de funcionalidades ofrecidas, se decidió usar la API mediante servicio web, siendo necesario solo una sencilla petición HTTP y evitando así sobrecargar nuestro sistema con procesamientos innecesarios.

En Geonames, los propios usuarios pueden añadir nuevos contenidos, introducir puntos geograficos, categorizarlos y asociarles algún enlace informativo [4]. Estos puntos serán incorporados inicialmente a la capa Geonames y se sumarán a los ya existente en la capa Wikipedia (sobre la que el sistema realizará las consultas) mediante barridos periódicos.

Para mejorar los resultados de las búsquedas en el proyecto, al menos en Sevilla, han sido añadidas a GeoNames etiquetas para todos los articulos existentes en la capa Wikipedia de Google Maps, sin embargo éstos han no han sido incorporados a la base de datos sobre la que nuestro

[3] http://www.7host.com
[4] http://www.geonames.org/manual.html

proyecto realiza las consultas. En contacto con el fundador de GeoNames, Marc Wick, se ha podido saber que se pretende actualizar dicha base de datos antes de final de 2010, en funcnión de la disponibilidad del servidor de descarga de información de Wikipedia[5].

Geocoding API. Es una sencilla API con solo dos funcionalidades: dada unas coordenadas geográficas (latitud y longitud) devuelve una dirección real (calle, número, ciudad...) y viceversa. Su uso viene motivado por diversas razones:

- El dispositivo GPS proporciona la ubicación mediante latitud y longitud.
- Latitud y longitud son términos complicados para el usuario, es preferible mostrar direcciones reales.
- Las búsquedas son más efectivas proporcionando direcciones reales.
- El posicionamiento en el mapa es más exacto mediante latitud y longitud.

Aunque tampoco es una API destinada a Android, su funcionamiento se basa en una llamada HTTP con ciertos parámetros (la ubicación, ya sea mediante coordenadas o dirección real) y la devolución del resultado en un objeto XML o JSON.

Conexión de datos (WIFI/3G/GPRS). Por las diversas razones que ya hemos visto, nuestra aplicación necesitará disponer de conexión a la Red. Esta conexión se realiza mediante peticiones internas por parte de la API (por ejemplo: Maps) o bien por peticiones concretas (por ejemplo: búsquedas o Geocoding).

En cualquier caso, esas peticiones son dirigidas a nuestro dispositivo, que será el encargado de gestionar la conexión a la Red, realizar la petición y devolver los resultados.

JSON (JavaScript Object Notation). Es un formato ligero de intercambio de datos que se ha generalizado en AJAX como alternativa a XML gracias a su fácil analisis mediante JavaScript y a la existencia de este en la inmensa mayoria de los navegadores web. No obstante, progresivamente se ha ido ampliando el soporte a más lenguajes de programación, entre ellos Java, motivo por el cual ha sido elegido para manejar las respuestas de nuestras búsquedas.

En cuanto a requisitos no funcionales, además de los conocimientos requeridos por las herramientas de desarollo expuestas anteriormente, es necesario tener en cuenta otros aspectos importantes el proyecto debe desarrollarse con el concepto de modularidad necesarios, siguiendo las pautas de bajo acoplamiento y alta coherencia.

[5] http://download.wikipedia.org/

También es importante tener en cuenta el hecho de estar programando para un sistema empotrado, por lo que habrá que evitar consumir recursos que no sean estrictamente necesarios, que tendrían un alcance significativo más allá de nuestra aplicación, ya que podría provocar un drástico efecto sobre el consumo de bateria. El punto más crítico en este sentido será la recepción de la señal del GPS.

Sobre las posibilidades ofrecidas por la aplicación, dado que está orientado al turismo y a su uso en diversos países, deberá ofrecer la posibilidad de usar varios idiomas, tanto en la aplicación como en las búsquedas de información.

Finalmente, dadas las diferentes versiones del SDK de Android que han sido publicadas y las modificaciones que, aunque pequeñas, han ido realizando los distintos fabricantes, es importante que el sistema sea probado bajo la mayor variedad de versiones posibles, ya sea mendiante simulación o en terminales reales. Sobre este aspecto ahondaremos en la fase de Pruebas

7.2– Diseño

Tras el Análisis de Requisitos, ya podemos realizar el diseño de nuestro sistema, distribuyendo las funcionalidades previstas de la forma más óptima posible, proporcionando un entendimiento fácil e intuitivo, así como su mantenimiento.

Antes de ahondar en el diseño, es necesario explicar que una interfaz gráfica y la interactuación usuario-sistema se realiza, en la mayoria de los casos, por medio de Activities, mediante extensión de la clase Activity. Se podría decir que una Activity es una pantalla de la aplicación de la que el usuario dispone para interactuar con el sistema.

Empezaremos por distinguir los dos subsistemas del proyecto, con funcionalidades claramente diferenciadas: el mapa, donde de mostrará la ubicación del usuario y los lugares de interés, y el navegador web, donde se mostrará información tanto de dichos lugares como de la propia aplicación. Por tanto, la estructura elegida para el diseño de la interfaz gráfica ha sido un sistema de pestañas, constituyendo cada uno de estos dos subsistemas una pestaña independiente de la otra.

Como consecuencia, la aplicación debe incluir dos Activities: una para cada subsitema. Adicionalmente, una tercera actividad, un tanto peculiar, debe ser incluida para gestionar el sistema de pestañas. Esta Activity, *MainActivity.class*, será la encargada de la puesta en marcha de la aplicación, creando las otras dos Activities, *MapTab.class* e *InfoWeb.class*, y estableciendolas como hijas en sendas pestañas.

El núcleo principal del sistema está sustentado en *MapTab.class*, que realmente extiene una Activity especial: *MapActivity*. Tiene asignadas tres tareas básicas: construcción del mapa, designación de un localizador de

posición que interactue sobre él y la cofiguración de la aplicación. Las opciones configurables serán el idioma de la aplicación y el modo de vista en el mapa.

Hay que aclarar que esta configuración manual del idioma solo afectará en la búsqueda de lugares de interés, ofreciendo búsqueda en idioma automático (el idioma local del país donde nos encontremos o, en caso de no obtener resultados, inglés), o bien eligiendo manualmente el idioma deseado, en cuyo caso podriamos no encontrar la cantidad de resultados esperados. El modo de vistas nos ofrecerá la elección entre *MAP*, *Satellite* y *Traffic*, aunque esta última opción, por limitaciones propias de la API, no está disponible en todos los lugares.

Ambas configuraciones se realizan a partir de un menú, desde el cual se podrá obtener más información sobre la aplicación, ayuda o el acceso a la propia configuración. Los dos primeros casos serán atendidos por el navegador web de la aplicación, previa conexión con la web oficial de la aplicación. Por otro lado, la configuración se realiza lanzado una nueva Activity del tipo PreferenceActivity: *MyPreferences.class*. Este tipo de actividad nos proporciona un almacen de variables de uso común para la actividad que lo lanza.

Pero sin duda, la principal funcionlidad de esta clase es asignar el *listener* que se encargará de actualizar periódicamente la posición del usuario y realizar sobre el mapa las operaciones necesarias. Será, pues, la clase *GeoUpdateHandler.class*, que implementa *LocationListener*, la que centralice el funcionamiento de la aplicación.

Una vez captada la posición del usuario, las coordenadas geográficas serán convertidas en una dirección real haciendo uso de la clase *Geocoding.class*, que es la encargada de hacer la consulta adecuada a la API Geocode. La dirección obtenida será mostrada por pantalla gracias a la clase *UserOverlay.class*, que extiende a *ItemizedOverlay*. Esta clase situará en la posición correcta del mapa un icono que representará a usuario.

Conociendo las coordenadas, la siguiente consulta se realiza al proveedor de información, clase *InfoProvider.class*, un agregado de *GeoUpdateHandler*. Esta clase, dadas unas coordenadas geográficas hace una llamada a la API Geonames para obtener los lugares de interés proximos a la ubicación actual. Antes de lanzar esta consulta deberá decidir en que idioma se realizara la b‘úsqueda:

- Si en preferencias aparece marcada la opción de “Idioma automatico”, la búsqueda se realiza en el idioma que corresponde al país donde está ubicado. Si no encuentra ningún resultado la búsqueda volverá a lanzarse en inglés.

- Si en preferencias no está marcada dicha opción, la búsqueda se realizará en el idioma elegido, y en caso de no encontrar resultados se volverá a intentar, al igual que en el caso anterior, en inglés. Esta vez

no se posibilita la búsqueda en el idioma local, al haber sido decartado de antemano por el usuario, por tanto el objetivo de optimizar o no los resultados de la búsqueda queda bajo responsabilidad del usuario.

Estos dos supuestos y sus resultados en la práctica serán explicados posteriormente tanto en la fase de Implementación y de Pruebas.

Los resultados se obtienen en formato JSON e inmediatamente serán organizados en una lista de objetos más adecuados a nuestro objetivo: la clase *Place.class*. Esta clase se encarga únicamente de almacenar la información que será de nuestro interés para el tratamiento de los lugares de interés, includida una dirección URL que enlaza al correspondiente artículo en la Wikipedia. Esta lista de Place, será usada por *PlaceOverlay.class*, bajo petición de *GeoUpdateHandler*. El resultado será similar al obtenido mediante *UserOverlay*, pero en este caso se actua sobre una lista de elementos y se usa un icono diferente al anterior, por motivos de claridad en la información mostrada.

A partir de este instante es necesaria la interactuación del usuario. Es él quien debe navegar por el mapa y elegir el punto sobre el que desea mayor información. Esta decisión es tomada simplemente tocando el elemento por el que se haya decidido. Una instancia de *InfoDialog.class* será lanzada, y mostrará el nombre del lugar elegido, una breve descripción y una fotografia (si estuviera disponible). Junto a la información aparecen dos botones. Mediante *Cancelar* se cierra la ventana de información, volviendo nuevamente al mapa, donde poder consultar otro lugar.

Por contra, si se pulsa sobre *Mas Info*, la pagina web asociada será cargada en nuestro navegador, en la clase *Infoweb.class*, y dicha pestaña pasará a estar visible. Este navegador empotrado en nuestra aplicación dispone de una sencilla funcionalidad que cubre las necesidades de la aplicación: mostrar la web de información (haciendo uso de las operaciones de zoom y desplazamiento necesarias), navegar hacia atrás y hacia adelante (si fuera posible) de la misma forma que lo haría cualquier navegador, y recargar la página actual, en caso de haber obtenido algún error en un anterior intento.

A continuación se representa un diagrama de clases donde se reflejan las principales funcionalides de cada una, así como su interconexión.

El funcionamiento expuesto en esta sección, y cuya implementación será detallada a continuación, puede resumirse en el siguiente diagrama de actividad, que representa un hilo típico de la interactuación usuario-sistema y las consultas realizadas a los diversos servicios web.

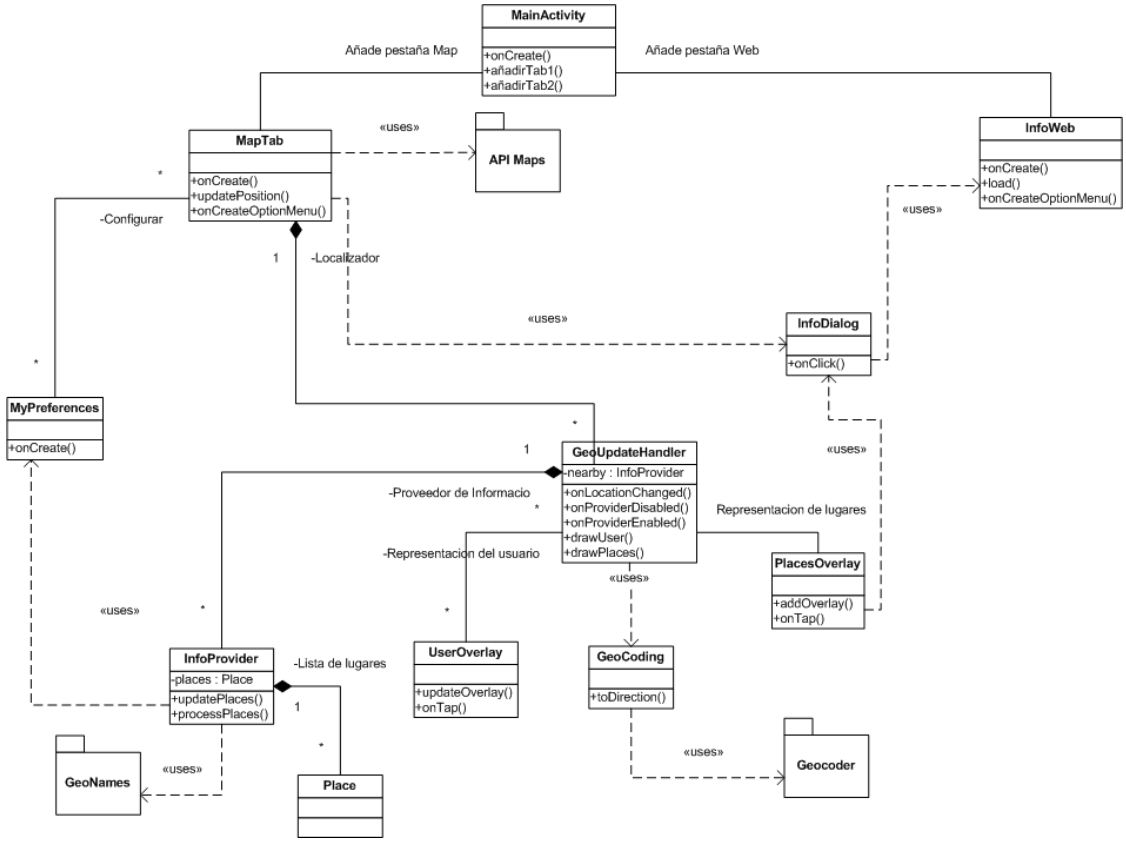


Figura 7.4: Diagrama de clases

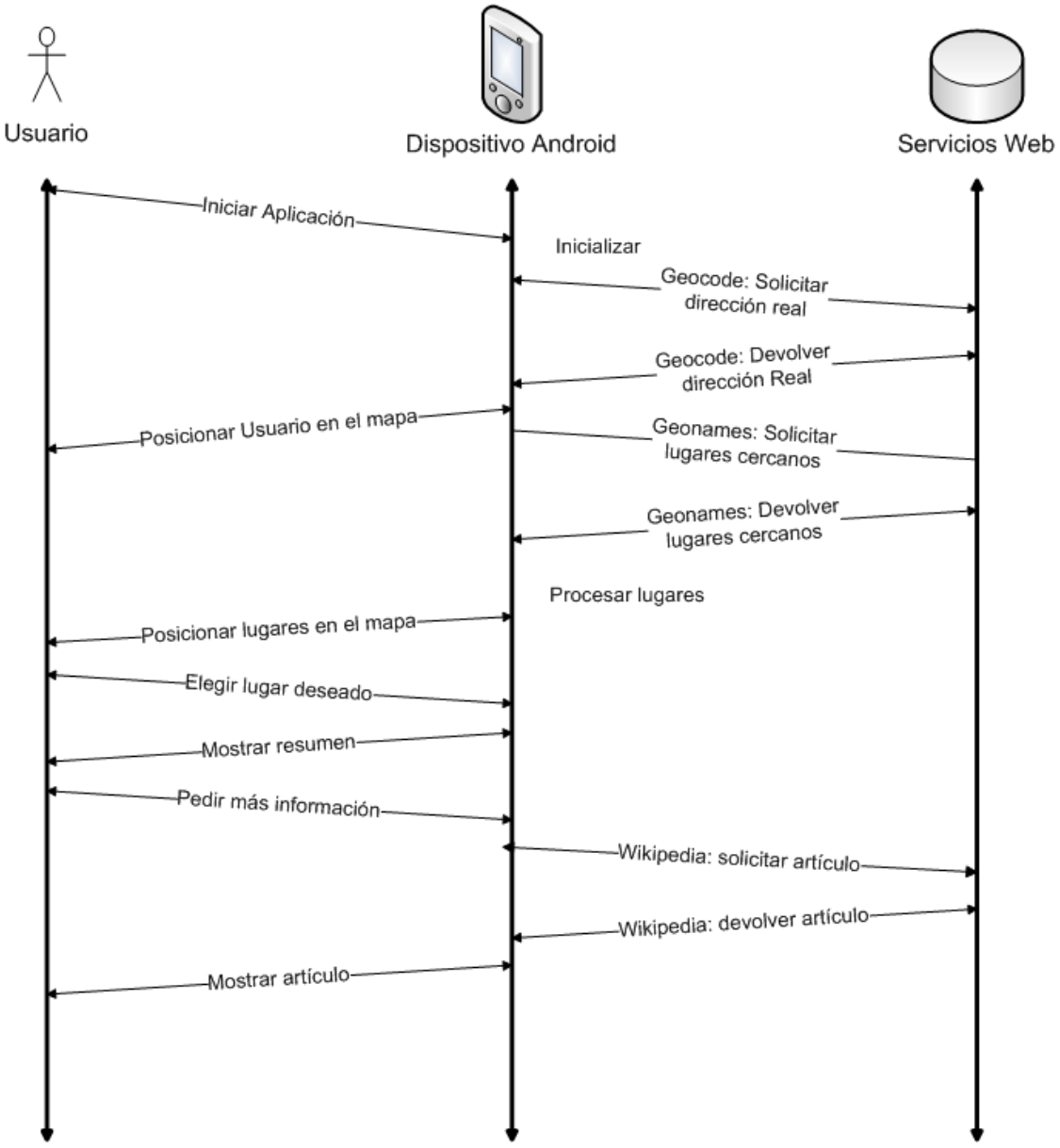


Figura 7.5: Interacción usuario-sistema

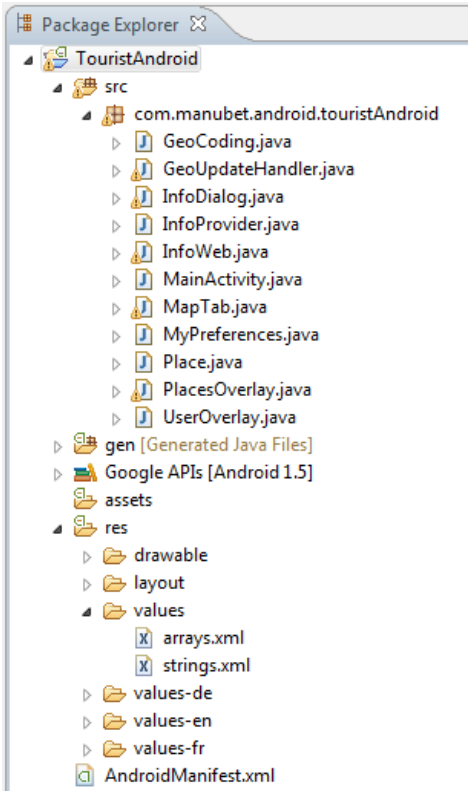


Figura 7.6: Estructura de un proyecto Android

7.3– Implementación

A continuación se detalla el nucleo de la aplicación y se comenta todo el codigo fuente desarrollado. El punto de inicio es la actividad principal, que se encarga de crear las otras dos actividades que actuaran a modo de pestañas. En cada actividad, además de documentar su clase principal, se referencian también cada una de las demás clases usadas, así como los ficheros layout necesarios.

Antes de empezar, es importante comentar la estructura de un proyecto Android. Su peculiaridad reside en la carpeta "res", de resources, donde a su vez se incluyen tres tipos de carpetas: "drawable", para imágenes, "layout", para los ficheros de configuración gráfica, y "values", donde se crean diccionarios de variables, por ejemplo Strings. Cada una de estas carpetas ofrece importantes funcionalidades, aunque en este proyecto solo se han usado las aportadas por "values", que se comentarán al final de la sección de Implementación.

Sobre esta estructura reside la modularidad definida en un proyecto Android, de forma que todos estos recursos quedan totalmente desaco-

plados del código de la aplicación. Su uso no es obligatorio, pero si muy recomendado, por directivas propias de *buenas prácticas* en el diseño del software.

Además existe la carpeta "Gen", autogenerada, que hace las funciones de índice para nuestros recursos, a través del fichero "R.java"contenido nuestro código fuente podrá acceder a los recursos comentados anteriormente.

7.3.1. Clase MainActivity.class

Es la clase principal de la aplicación, extiende a *TabActivity*, encargada de inicializar todo el proceso y crear las actividades hijas. Éstas serán añadidas a partir de mTabHost, mientras que en mResources se construirá el almacén de recursos de donde se extraerá toda la información necesaria a lo largo del proceso.

```
1 public class MainActivity extends TabActivity {
2     public static TabHost mTabHost;
3     private Resources mResources;
4 }
```

Código 7.1: Encabezado de la clase MainActovity.class

El método onCreate() supone el punto de entrada en el programa. En primer lugar, llama al método onCreate() de la clase padre, y con el parámetro Bundle se crea la interfaz de la aplicación. La variable window sirve para configurar la ventana, en este caso se realiza la asignación del icono de la aplicación. *setContentView* es un método esencial, cuya función es establecer el layout en la actividad, haciendo visibles los elementos definidos en él (se detallará más adelante). Tras iniciar las variables necesarias, se crean las dos pestañas de nuestra catividad.

```
1  public void onCreate(Bundle savedInstanceState) {
2     super.onCreate(savedInstanceState);
3     Window window = getWindow();
4     window.requestFeature(Window.FEATURE_LEFT_ICON);
5     setContentView(R.layout.tab);
6     mTabHost = getTabHost();
7     mResources = getResources();
8     window.setFeatureDrawableResource(Window.FEATURE_LEFT_ICON, R.
          drawable.logo);
9     anadirTab1();
10    anadirTab2();
11 }
```

Código 7.2: método onCreate() de la clase MainActovity.class

Como ya se ha comentado, cada pestaña constituye una actividad nueva. Cada nueva actividad se crea mediante un *Intent* que recibe el nombre

de la clase correspondiente. Dicho *Intent* se establece como contenido de la pestaña mediante el método *setContent*, y además se le asigna una etiqueta, un nombre y un icono. Todas estas operaciones se realizan sobre el elemento *TabSpec*, que finalmente se agrega a *mTabHost*.

Se puede observar la forma en que son llamados tanto el nombre de la pestaña como el icono a través de la clase *R.java* comentada anteriormente.

```
1     //Pestana 1
2
3     private void anadirTab1() {
4         Intent intent = new Intent(this, MapTab.class);
5         TabSpec spec = mTabHost.newTabSpec("map");
6         spec.setIndicator(mResources.getString(R.string.Tab1),
              mResources.getDrawable(R.drawable.ic_map));
7         spec.setContent(intent);
8         mTabHost.addTab(spec);
9     }
10
11    //Pestana 2
12
13
14    private void anadirTab2() {
15
16        Intent intent = new Intent(this, InfoWeb.class);
17        TabSpec spec = mTabHost.newTabSpec("web");
18        spec.setIndicator(mResources.getString(R.string.Tab2),
              mResources.getDrawable(R.drawable.ic_lupa));
19        spec.setContent(intent);
20        mTabHost.addTab(spec);
21    }
```

Código 7.3: añadir pestañas a la clase MainActovity.class

En este caso, los recursos de tipo *drawable* especificados (*ic_map* e *ic_lupa*) no son archivos gráficos, si no un fichero XML que elige entre dos iconos, en color o en escala de grises, según la pestaña se encuentre activa o no.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <selector xmlns:android="http://schemas.android.com/apk/res/
       android">
3      <!-- selected-->
4      <item android:drawable="@drawable/lupa"
5            android:state_selected="true" />
6      <!-- not selected-->
7      <item android:drawable="@drawable/lupa_off" />
8  </selector>
```

Código 7.4: Selección de iconos según estado de la pestaña

Finalmente, se ha creado un método *swichtTab*, que recibe un entero como parámetro y establece dicha pestaña como visible. Está definido como *static* para poder acceder a él desde fuera de la clase.

```
1     public static void swichtTab(int i){
2         mTabHost.setCurrentTab(i);
3     }
```

Código 7.5: Método para cambiar la pestaña activa

Una vez mostrada la actividad, el contenido visible será el especificado en el layout establecido. El elemento padre de dicho layout es un *TabHost*, que será el gestor de las pestañas. *LinearLayout* establece la alineación de los contenidos, que en este caso son un *TabWidget*, la representación de las pestañas en sí, y *FrameLayout*, donde se muestra el contenido de cada una de ellas.

Para cada elemento se especifica su tamaño, *width* y *height*, y su margen , *padding*. Un atributo muy importante, aunque en esta clase no se ha usado, es el *id*, que sirve como identificador para controlar el elemento en tiempo de ejecución.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TabHost xmlns:android="http://schemas.android.com/apk/res/android
       "
3      android:id="@android:id/tabhost"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <LinearLayout
7          android:orientation="vertical"
8          android:layout_width="fill_parent"
9          android:layout_height="fill_parent">
10         <TabWidget android:id="@android:id/tabs"
11             android:layout_width="fill_parent"
12             android:layout_height="wrap_content"
13             android:paddingLeft="1dip"
14             android:paddingRight="1dip"
15             android:paddingTop="1dip"/>
16         <FrameLayout android:id="@android:id/tabcontent"
17             android:layout_width="fill_parent"
18             android:layout_height="0dip"
19             android:layout_weight="1"/>
20     </LinearLayout>
21 </TabHost>
```

Código 7.6: Layout de MainActivity.class

7.3.2. Clase MapTab.class

La primera de las pestañas de la aplicación es la correspondiente al mapa, donde se desarrollan los principales procesos y donde se crean las variables necesarias. *mapView* es la representación del mapa, *mapController* permite su control, *mapOverlays* es el conjunto de los elementos pintados sobre el mapa y *locationManager* es el escuchador del GPS y gestor de toda la actividad. Finalmente, *preferences* y *mode* son necesarios para gestionar el modo de vista del mapa.

```
1  public class MapTab extends MapActivity {
2     private MapController mapController;
3      private static MapView mapView;
4      private List<Overlay> mapOverlays;
5      private LocationManager locationManager;
6      private SharedPreferences preferences;
7      private String mode="MAP";
```

Código 7.7: Encabezado de la clase MapTab.class

Tras establecer el layout de la clase, el método *onCreate* inicializa y configura los elementos visibles, en este caso *mapView*. En primer lugar obtiene el objeto desde el layout mediante *findViewById* y posteriormente establece el controlador de zoom e inicializa las demás variables creadas anteriormente. Además, carga las preferecias por defecto.

```
1      public void onCreate(Bundle savedInstanceState) {
2          super.onCreate(savedInstanceState);
3          setContentView(R.layout.main);
4          mapView = (MapView) findViewById(R.id.mapview);
5          preferences = PreferenceManager.getDefaultSharedPreferences
               (this);
6          mapView.displayZoomControls(true);
7          mapView.computeScroll();
8          mapOverlays = mapView.getOverlays();
9          mapController = mapView.getController();
10         mapController.setZoom(17);
11     }
```

Código 7.8: Método onCreate de MapTab.class

Esta actividad es la que muestra el menú para la configuración de la aplcación. El método *onMenuCreateOptionMenu* se encarga de crearlo al presionar el boton "Menu"del dispositivo. Tras crear el objeto *MenuInflater*, solo es necesario asignarle los elementos definidos en su layout. Tanto el método como el objeto *MenuInflater* forman parte de la API Android.

```
1     public boolean onCreateOptionsMenu(Menu menu) {
2         MenuInflater inflater = getMenuInflater();
3         inflater.inflate(R.layout.menumap, menu);
4         return(super.onCreateOptionsMenu(menu));
5     }
```

Código 7.9: Creación del Menú en MapTab.class

Una vez creado el menú, sus gestión corre a cargo de *onOptionsItemSelected*. Mediante un switch se determina la opción elegida y se ejecuta la correspondiente acción. "About" lanza un *InfoDialog*, cuyo contenido será información acerca de la autoría. La opcción "Help" accede al navegador web para mostrar el manual de la aplicación, disponible en la web oficial.

Para realizar esta última operación, en primer lugar escribe la dirección en el navegador, en la variable *currentUrl* y posteriormente ejecuta el método de carga *load*. La ruta de dicho manual se construye concatenando la dirección del host y la de entrada de la web correspondiente a dicho manual, almacenadas en el diccionario de String,. Posteriormente se activa el navegador como pestaña activa.

Finalmente, la opción más interesante es "configure", pues lanza una nueva actividad, *MyPreferences.clas*, mediante la cual se accede a los parámetros configurables del mapa. Dicha actividad será tratada posteriormente.

```
1     public boolean onOptionsItemSelected(MenuItem item) {
2         // Handle item selection
3         switch (item.getItemId()) {
4         case R.id.configure:
5             startActivity(new Intent(this, MyPreferences.class));
6             return(true);
7         case R.id.about:{
8             InfoDialog aboutDialog = new InfoDialog(mapView.
                  getContext());
9             aboutDialog.show();
10        }
11            return true;
12        case R.id.help:{
13            try {
14            InfoWeb.currentUrl=getResources().getString(R.string.
                  homepage)+getResources().getString(R.string.manual);
15            InfoWeb.load();
16            }
17            catch  (NullPointerException e) {
18                // TODO Auto-generated catch block
19                e.printStackTrace();
20            }
21            MainActivity.swichtTab(1);
```

```
22              }
23              return true;
24          default:
25              return super.onOptionsItemSelected(item);
26          }
27      }
```

Código 7.10: Gestión del Menú en MapTab.class

El método *onResume*, que, al igual que el anterior, forma parte de la API Android, es llamado cuando la actividad pasa a estar en primer plano. Es decir, tanto al ser creada como, por ejemplo, al salir del menú de preferencias. Aquí podrán comprobarse variaciones en dichas preferencia, cuyo único cambio que tendrá impacto en este momento es modo de vista en el mapa.

Tras capturar el valor de dicha variable mediante el objeto *preferences*, se comprueba si ha variado respecto al actual, en cuyo caso se comprueba el nuevo valor y se utilizan los métodos *setTraffic* y *setSatellite* para ajustar el mapa a los cambios que se hayan podido realizar. Nótese que la vista "Traffic" no está disponible en todas las ubicaciones, por lo que en algunas ocasiones puede no verse reflejado ningún cambio en el mapa. La otra función de este método es la llamada a *updatePosition*.

```
1   public void onResume() {
2           super.onResume();
3           String newmodo=preferences.getString("vistas", "MAP");
4           if (!newmodo.equals(mode)) {
5               mode=newmodo;
6               if (newmodo.equals("TR")){
7                   mapView.setTraffic(true);
8                   mapView.setSatellite(false);
9                   }
10              else if (newmodo.equals("MAP")){
11                  mapView.setTraffic(false);
12                  mapView.setSatellite(false);
13                  }
14              else if (newmodo.equals("SAT")){
15                  mapView.setTraffic(false);
16                  mapView.setSatellite(true);
17                  }
18              }
19      }
```

Código 7.11: Gestión de cambios en el menú

Este método *updatePosition* se encarga de lanzar el funcionamiento propiamente dicho. En primer lugar crea el objeto myGPS, estudiado más adelante, y, tras comprobar la disponibilidad del dispositivo GPS, lo asigna como objeto responsable de escuchar la posición. Dicha operación se realiza con el objeto *locationManager*. Si el dispositivo está activo, se mostrará un mensaje informando al usuario del correcto funcionamiento, en caso contrario se mostrará una advertencia para que el usuario lo active y la aplicación sea reiniciada. Estos mensajes se muestran mediente *Toast*, una clase proporcionada por Android para tal fin.

Por motivos de economía en la batería, factor fundamental en los dispositivos móviles, la escucha del GPS se actualizará cada 5 segundos o 5 metros. Dado que la apliación está pensada para ser usada a pie, esta frecuencia de escucha no causarán ningun efecto negativo.

```
1     private void updatePosition() {
2         GeoUpdateHandler myGPS = new GeoUpdateHandler(this,
              mapController,mapView,mapOverlays); //crear el listener
3         locationManager = (LocationManager) getSystemService(
              Context.LOCATION_SERVICE);
4         if (locationManager.isProviderEnabled(LocationManager.GPS_
              PROVIDER)) {
5             locationManager.requestLocationUpdates(
                  LocationManager.GPS_PROVIDER, 5000, 5, myGPS);
6             Toast.makeText(getBaseContext(),
7                     getResources().getString(R.string.gps_signal)
                          ,
8                     Toast.LENGTH_LONG).show();
9             }
10        else {
11                Toast.makeText(getBaseContext(),
12                    getResources().getString(R.string.provider_not_
                          found),
13                    Toast.LENGTH_LONG).show();
14            }
15    }
```

Código 7.12: Método updatePosition de MapTab.class

Finalmente, el método *onPause* tiene la función contraria a *onResume*, ya que se ejecuta cuando la actividad pasa a estado inactivo. En ese momento la aplicación va a dejar de ser utilizada, por lo que es conveniente liberar recursos, sobre todo teniendo en cuenta el importante consumo del dispositivo GPS. Para ello, el objeto myGPS creado anteriormente es desacoplado como *listener*, y posteriormente se elimina. El GPS dejará de recibir señal y de consumir batería innecesariamente.

Esto no supone ningun problema, puesto que cuando la aplicación vuelva a estar activa se volverá a llamar a *onResume*, y se procederá como ya se ha explicado.

```
1     public void onPause(){
2         super.onPause();
3         locationManager.removeUpdates(myGPS);
4         myGPS=null;
5     }
```

Código 7.13: Método updatePosition de MapTab.class

En el layout del menú se representan los tres items que deben ser mostrados. A cada uno se le asigna un "id", usado en los métodos anteriores, un icono y un texto, tomados de "Drawable" y 'String" respectivamente.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android">
3      <item android:id="@+id/configure"
4            android:icon="@drawable/ic_conf"
5            android:title="@+string/configure"/>
6      <item android:id="@+id/about"
7            android:icon="@drawable/ic_about"
8            android:title="@+string/about" />
9      <item android:id= "@+id/help"
10           android:icon="@drawable/ic_help"
11           android:title="@+string/help"/>
12 </menu>
```

Código 7.14: Layout del menú

Anteriormente se ha comentado la necesidad de generar una key que permita usar la API Maps.Esa key se inserta en Layout de la actividad, donde se instancia el *MapView*, contenido en un *LinearLayout*, sobre el que se inserta el mapa.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
       android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <com.google.android.maps.MapView
7      xmlns:android="http://schemas.android.com/apk/res/android"
8      android:id="@+id/mapview"
9      android:layout_width="fill_parent"
10     android:layout_height="fill_parent"
11     android:clickable="true"
12     android:apiKey="0EOvUUT8BRMVfCdwI184xehnVKB2I05PknqDGKg"/>
13 </LinearLayout>
```

Código 7.15: Layout de TabMap.class

Clase GeoUpdateHandler.class

Como se ha dicho anteriormente, esta clase centraliza las principales acciones que se realizan en la aplicación, pues desde aquí se llama a los métodos de los distintos objetos que intervienen. Se ha elegido esta clase porque además es la que va a recibir la comunicación del dispositivo GPS (implementa *LocationListener*), por lo que será precisamente aquí donde se disponga de los datos de geoposicionamiento, esenciales para el funcionamiento del resto del sistema.

Esta clase recoge las variables que fueron pasadas en el constructor llamado en la clase *MapTab*. Entre los nuevas variables, *Position* recogerá la información de posicionamiento captada; *userItem* y *placeItem* son marcadores de usuario y lugar respectivamente, que serán representados en el mapa; *userPicOverlay* y *marketOverlay* son los iconos que los representarán; *nearby* es el obejeto usado para la búsqueda de información; y *dirección* almacenará la dirección real una vez haya sido realizada la conversión en base a sus coordenadas geográficas.

```
1  public class GeoUpdateHandler implements LocationListener {
2
3      private MapController mapController;
4      private GeoPoint position=new GeoPoint(0,0);
5      private MapView mapView;
6      private List<Overlay> mapOverlays;
7      private OverlayItem userItem,placeItem;
8      private UserOverlay userPicOverlay;
9      private PlacesOverlay markerOverlay;
10     private InfoProvider nearby;
11     static String direccion=null;
12     private Context mContext;
```

Código 7.16: Encabezado de la clase GeoUpdateHandler.class

Su constructor es sencillo, únicamente asigna los parámetros recibidos y crea el proveedor de información *InfoProvider*.

```
1      public GeoUpdateHandler(Context context, MapController
           mapController, MapView mapView, List<Overlay> mapOverlays){
2          this.mapController = mapController;
3          this.mapView=mapView;
4          this.mapOverlays=mapOverlays;
5          mContext= context;
6          nearby=new InfoProvider(mContext);
7
8
9      }
```

Código 7.17: Constructor de GeoUpdateHandler

Cada vez que se realiza una actualización de la posición, el método *onLocationChanged* es llamado y se lee la nueva posición, se reflejan los cambios en *position* y se borran los iconos representados en el mapa para realizar la actualización. Primero el usuario, mediante la llamada a *drawUser* y posteriormente los lugares, llamando a *drawPlaces*. Previamente, los lugares deben actualizarse. Para ello se llama al método *updatePlaces* del proveedor de información *nearby*. Por ultimo, el mapa se centra en la posición del usuario.

Los datos de posicionamiento son almacenados como "double" para no perder información en la consulta de la dirección, sin embargo es necesaria una conversión a "int" para el constructor de *GeoPoint*.

```
1     public void onLocationChanged(Location location) {
2             double lat = location.getLatitude();
3             double lng = location.getLongitude();
4             position = new GeoPoint((int)(lat*1E6),(int)(lng*1E6));
5
6             mapOverlays=mapView.getOverlays();
7             mapOverlays.clear();
8
9             drawUser(lat,lng);
10            nearby.updatePlaces(lat,lng);
11
12            drawPlaces(nearby.getPlaces());
13            mapController.animateTo(position);
14            mapView.setBuiltInZoomControls(true);
15    }
```

Código 7.18: Método onLocationChanged

Antes de representar al usuario se realiza la conversión de coordenadas geográficas a dirección real mediante la clase GeoCoding. Posteriormente se crea el icono, el item a representar (junto a la dirección anteriormente calculada) y se actualiza userPicOverlay para mostrarlo sobre el mapa.

```
1     private void drawUser(double lat, double lng) {
2        direccion=GeoCoding.toDirection(lat, lng);
3        Drawable userPic = mapView.getResources().getDrawable(R.
             drawable.iconopersona);
4       userPicOverlay = new UserOverlay(userPic, mapView.getContext
            ());
5        userItem = new OverlayItem(position, "",direccion);
6        userPicOverlay.updateOverlay(userItem);
7
8        mapOverlays=mapView.getOverlays();
9        mapOverlays.add(userPicOverlay);
10    }
```

Código 7.19: Método drawUser

La representación de lugares sigue un proceso similar, pero en este caso se itera sobre la lista de *Place* que los contiene, extrayendo de cada uno la información necesaria para la representación.

```
1     private void drawPlaces(LinkedList<Place> places){
2         Iterator<Place> it = places.listIterator();
3         Place aux;
4
5         Drawable markPic = mapView.getResources().getDrawable(R.
              drawable.marker);
6         markerOverlay = new PlacesOverlay(markPic, mapView.
              getContext());
7         mapOverlays=mapView.getOverlays();
8
9         while (it.hasNext()){
10            aux=it.next();
11            placeItem = new OverlayItem(new GeoPoint((int)(aux.
                  getLat()*1E6),(int)(aux.getLng()*1E6)),
12                                        "",aux.getNombre());
13            markerOverlay.addOverlay(placeItem);
14            mapOverlays.add(markerOverlay);
15            }
16    }
```

Código 7.20: Método drawPlaces

Finalmente se incluyen tres métodos que chequean el estado del GPS, los primeros chequean el cambio de activo a inactivo y viceversa por parte del usuario, informando de dicho cambio, el tercero se llama cuando cambia el estado interno, pero no se realiza ninguna actividad palpable por el usuario.

```
1     public void onProviderDisabled(String provider)
2     {
3         Toast.makeText( mContext,R.string.gpsDesactivado,Toast.
              LENGTH_SHORT ).show();
4     }
5     public void onProviderEnabled(String provider)
6     {
7         Toast.makeText( mContext,R.string.gpsActivo,Toast.LENGTH_
              SHORT ).show();
8     }
9     public void onStatusChanged(String provider, int status, Bundle
           extras){
10    }
```

Código 7.21: Otros métodos de GeoUpdateHandler

Clase GeoCoding.class

Esta clase realiza, consultando la API Geocode, la conversión entre coordenadas geográficas y dirección real. Posee un único método, estático, y varias variables auxiliares para la correcta ejecución del algoritmo.

Para formada la cadena de la URL de la consulta combina el patrón de conexión y la posición capturada por el GPS. Como se comentó anteriormente, los datos de posición de usan con formato "double" para evitar pérdidas de precisión al obtener la dirección real. Durante la conexión con el sitio web es importante controlar que no se produzcan errores, en cuyo caso, lamentablemente, se devuelve la dirección en forma de coordenadas (primer bloque *try-catch*). Además, se fija como "EN" el parametro country de *InfoProvider* para evitar errores mayores, como se verá posteriormente.

```
1     public static String toDirection(double lat, double lon) {
2         URL url=null;
3         StringBuilder builder;
4         String pais=null;
5         boolean paisEnc=false;
6         int jsonIndex=3;
7         JSONObject primero=null;
8         try {
9             url = new URL("http://maps.google.com/maps/api/geocode/
                  json?latlng="+lat+","+lon+"&sensor=true");
10        } catch (MalformedURLException e) {
11            InfoProvider.country="EN";
12            return lat+","+lon;
13        }
```

Código 7.22: Método toDirection de GeoCoding.class: conexión

Si la conexión se realizó con éxito dentro del *timeout* fijado, se extrae la parte del resultado necesaria para construir la dirección del usuario. La respuesta obtenida es un objeto JSON serializado, por lo que una vez recibido se realiza la conversión a este formato para ser tratado[6]. Podría darse el caso de que el formato no fuera el esperado y no pudiera procesarse adecuadamente. Dicho error sería tratado de igual forma que el caso anterior (representado en los cuatro bloques *try-cacht* sucesivos).

```
1         URLConnection connection;
2         try {
3             connection = url.openConnection();
4             connection.setConnectTimeout(000);
5             connection.setReadTimeout(1000);
6             connection.addRequestProperty("Referer","http://
                  touristdroid.blogspot.com/");
```

[6]Ver Apéndice C

```
7             String line;
8             builder = new StringBuilder();
9             BufferedReader reader = new BufferedReader(new
                  InputStreamReader(connection.getInputStream()));
10            while((line = reader.readLine()) != null) {
11                builder.append(line);
12            }
13        }catch (IOException e) {
14            InfoProvider.country="EN";
15            return lat+","+lon;
16        }
17
18        JSONObject json;
19        JSONArray results = null;
20        try {
21            json = new JSONObject(builder.toString());
22            results= json.getJSONArray("results");
23        } catch (JSONException e1) {
24            InfoProvider.country="EN";
25            return lat+","+lon;
26        }
27
28        try {
29         primero = results.getJSONObject(0);
30        }
31        catch (JSONException e){
32            InfoProvider.country="EN";
33            return lat+","+lon;
34        }
35        JSONArray add_comp;
36        try {
37            add_comp = primero.getJSONArray("address_components");
38        } catch (JSONException e) {
39            InfoProvider.country="EN";
40            return lat+","+lon;
41        }
```

Código 7.23: Método toDirection de GeoCoding.class: extracción del resultado

A continuación se realizan sucesivas consultas sobre “calle”, “número”y “ciudad”. Si alguna de las consultas fallara ese campo quedaría en blanco. Esto podría ser debido a que Geocode no dispusiera de información suficientemente detallada de la posición.

Por último, se consulta el país. Esta consulta también puede provocar errores, pero no se podrá dejar en blanco, ya que su valor va a proporcionar mejores o peores resultados en la posterior búsqueda de lugares. Por esa razon, el algoritmo diseñado intenta buscar más concienzudmente en los diferentes campos de la estructura JSON tratando de encontrar

el país (bloque *while*). Si finalmente no pudiera encontrarse, se asumiría "EN"como el país para la búsqueda.

El resultado devuelto es la concatenación correctamente formateada de todos los parámetros anteriores.

```
String street_number;
        try {
            street_number = add_comp.getJSONObject(0).getString("
                long_name");
        } catch (JSONException e) {
            street_number="";
        }
        String street;
        try {
            street = add_comp.getJSONObject(1).getString("long_name
                ");
        } catch (JSONException e) {
            street="";
        }
        String city;
        try {
            city = add_comp.getJSONObject(2).getString("long_name")
                ;
        } catch (JSONException e) {
            city="";
        }
        while(jsonIndex<add_comp.length()&!paisEnc){ //pais
            JSONObject aux;
            try {
                aux = add_comp.getJSONObject(jsonIndex);
                JSONArray aux1=aux.getJSONArray("types");
                if(aux1.getString(0).equals("country")){
                    pais=aux.getString("short_name");
                    paisEnc=true;
            }
            } catch (JSONException e) {
                continue;
            }
            jsonIndex++;
        }
        if(pais==null){
            pais="EN";
        }
        InfoProvider.country=pais;
        return street+","+street_number+","+city;
    }
```

Código 7.24: Método toDirection de GeoCoding.class: construcción del resultado

Este proceso que, a priori, parecería tan sencillo ha tenido que ser rediseñado sucesivamente debido a errores producidos por los diversos formatos encontrados, tras haber observado que la respuesta no es estandar para todos los países. Solo se ha considerado incidir en la búsqueda del país, ya que tratar de obtener todos los datos, alguno quizás inexistente, podría provocar costosos procesamientos que causarían impacto tanto en el rendimiento de la aplicación como en la batería del telefono móvil.

Clase InfoProvider.class

Esta clase será usada para consultar la infofmación acerca de los lugares de interés. Su variable más característica es precisamente la lista de *Place* generada y las preferencias compartidas, inicializadas por el constructor. El resto son variables auxiliares necesarias para el algoritmo.

```
1  public class InfoProvider {
2      private static LinkedList<Place> places = new LinkedList<Place
           >();
3      private JSONObject json=null;;
4      URL url=null;
5      static String country;
6      Context mContext;
7      private SharedPreferences preferences;
8      private String autoIdi="AUTO";
9
10     public InfoProvider(Context context){
11         mContext=context;
12         preferences = PreferenceManager.getDefaultSharedPreferences
               (context);
13         }
```

Código 7.25: Declaración de InfoProvider.class

La búsqueda de lugares sigue un esquema parecido a GeoCoding, incluyendo los posibles errores a tener en cuenta, solo que en este caso, en caso de producirse algún error, se van a realizar hasta tres intentos. El motivo es que en caso anterior podríamos mostrar la posición geográfica en caso de no disponer de la dirección, pero si falla la búsqueda de lugares no se dispondrá de ninguna información alternativa, por lo que la aplicación quedaría inservible.

Antes de iniciar el proceso hay que decidir el idioma en que se va a realizar. Normalmente el mayor número de resultados se producen buscando en el idioma local, de ahí la importancia de resolver esa información al buscar la dirección del usuario. Sin embargo, en el menú de configuración, que se analizará más adelante, se da la posibilidad de hacer búsquedas en idioma automático (este caso) o bien especificar un idioma.

Esa es la comprobación que se hace al iniciar la ejecución de ese método consultando el valor de la variable “idi” en *Preferences*. A continuación

se contruye la URL necesaria para la consulta, del mismo modo que se vió anteriormente, y se lanza el bucle *while* que controlará la búsqueda.

Los dos primeros errores a controlar son los mismos que los ya vistos, a diferencia de que esta vez de informará al usuario y se incrementará la variable que controla el número de fallos. Para proceder al siguiente intento se usa la sentencia *continue*, que salta al inicio del bucle para realizar la próxima iteración, es decir, el próximo intento de búsqueda.

```
1     public boolean updatePlaces(double lat, double lng) {
2         boolean correcto=false;
3         int cont=0;
4         JSONArray resultado = new JSONArray();
5         //dada una direccion, hago una consulta
6         autoIdi=preferences.getString("idi", "AUTO");
7         if(!autoIdi.equals("AUTO")){
8             country=preferences.getString("idi", country);
9         }
10        StringBuilder responseBuilder = new StringBuilder();
11        while(!correcto & cont<3) {
12            try {
13                url = new URL("http://ws.geonames.org/
                      findNearbyWikipediaJSON?lat="+lat+"&lng="+lng+"&
                      lang="+country.toLowerCase()+"&maxRows=20&radius
                      =20");
14            } catch (MalformedURLException e) {
15                cont++;
16                Toast.makeText(mContext, R.string.error_proveedor,
                      Toast.LENGTH_SHORT).show();
17                continue;
18            }
19            URLConnection connection;
20            try {
21                connection = url.openConnection();
22                connection.setConnectTimeout(2000);
23            } catch (IOException e1) {
24                cont++;
25                Toast.makeText(mContext, R.string.error_conexion,
                      Toast.LENGTH_SHORT).show();
26                continue;
27            }
```

Código 7.26: Método updatePlaces, fragmento 1

El siguiente fragmento vuelve a ser equivalente a la consulta anterior. Los chequeos corresponden a la correcta recepción de la respuesta y que efectivamente corresponda a la serialziación de un objeto JSON[7].

[7] Ver Apéndice C

```
1         String line;
2        BufferedReader in;
3         try {
4             in = new BufferedReader(new InputStreamReader (url.
                  openStream()));
5            while ((line = in.readLine()) != null) {
6                responseBuilder.append(line);
7             }
8             in.close();
9         } catch (IOException e1) {
10            cont++;
11            Toast.makeText(mContext, R.string.error_recepcion,
                  Toast.LENGTH_SHORT).show();
12            continue;
13        }
14        try {
15         json = new JSONObject(responseBuilder.toString());
16       } catch (JSONException e) {
17          cont++;
18         continue;
19       }
```

Código 7.27: Método updatePlaces, fragmento 2

Por último, se toma la parte de la respuesta necesaria para procesarla. En caso de que esa operación sea errónea, se vuelve a intentar de la forma ya explicada. Si el resultado es positivo, se procesará si, y solo si, existe más de un resultado válido, es decir, si el tamaño del array de resultados es mayor que cero. En ese caso, la variable *correcto* toma valor *true* para no volver a ejecutar el bucle y ya se puede llamar al método *processPlaces*. En caso negativo se fija el inglés (valor "EN") como idioma de búsqueda y se vuelve a intentar.

El motivo para cambiar el idioma antes de volver a intentarlo es que, tras haberse producido ningún error en el proceso, no se han encontrado resultados en ese idioma, sea cual sea. Normalmente, tras el idioma local, en segundo lugar suelen estar los resultados en inglés, lo que justifica el cambio realizado.

```
1        try {
2            resultado=json.getJSONArray("geonames");
3         } catch (JSONException e) {
4            cont++;
5            continue;
6         }
7         if(resultado.length()>0 ){
8            correcto=true;
9            this.processPlaces(resultado,lat,lng);
10           }
```

```
11          else{
12              country="EN";
13              cont++;
14            responseBuilder = new StringBuilder();
15            url=null;
16            json=null;
17
18          }
19          }
20          return correcto;
21      }
```

Código 7.28: Método updatePlaces, fragmento 3

Una vez obtenidos los resultados deben ser procesados, eliminando la información no necesaria y convirtiéndolos en un objeto fácil de manejar: la clase *Place*, que será detallada en el siguiente epígrafe.

Además de declarar las variables auxiliares, el procesamiento comienza borrando el contenido ya existente en la lista "places"para almacenar los nuevos resultados. El proceso de almacenamiento se realiza mediante un bucle *for* que recorre los elementos del JSONArray obtenido en el método anterior, en cuya primera operación se toma cada elemento. Si en la obtención de dicho objeto se produce algun error será automáticamente descatado con la sentencia "continue".

A continuación se toman los valores "thumb "(imagen en miniatura del lugar) y "sumary"(resumen de la información asociada). En cado de no existir alguno de ellos se le daría un valor vacío y se continua el procedimiento para consultar los demás valores útiles: título, enlace URL a wikipedia, latitud y longitud. Si alguno de estos valores produce un fallo el lugar será descartado, puesto que no tendría lugar su existencia sin alguno de dichos valores, ya que o no podría representarse sobre el mapa (caso de no obtener sus coordenadas geográficas) o no podrá identificar el lugar (caso de no obtener titulo o articulo de Wikipedia).

Los lugares procesados adecuadamente son agregados a la lista de *Place*.

```
1   private void processPlaces(JSONArray result, double lat, double
        lng) {
2           JSONObject aux=null;
3           String thumb, summary;
4           getPlaces().clear();
5
6           for (int i=0;i<result.length();i++){
7               try {
8                   aux=result.getJSONObject(i);
9               } catch (JSONException e) {
10                  continue;
11              }
```

```
12              try {
13                  thumb=aux.getString("thumbnailImg");}
14              catch (JSONException j){
15                  thumb="";
16              }
17              try {
18                  summary = aux.getString("summary");
19              } catch (JSONException e1) {
20                  summary="";
21              }
22              try {
23                  getPlaces().add(new Place( aux.getString("title"),
24                            Double.valueOf(aux.getString("lat")).
                              doubleValue(),
25                            Double.valueOf(aux.getString("lng")).
                              doubleValue(),
26                            thumb,
27                            aux.getString("wikipediaUrl"),
28                            summary
29                            ));
30              } catch (NumberFormatException e) {
31                  continue;
32              } catch (JSONException e) {
33                  continue;
34              }
35          }
36      }
```

Código 7.29: Método processPlaces

Finalmente, la clase contiene los métodos *get* y *set* para obtener y modificar, respectivamente, el campo *places*.

```
1       public static void setPlaces(LinkedList<Place> places) {
2           InfoProvider.places = places;
3       }
4       public static LinkedList<Place> getPlaces() {
5           return places;
6       }
7               } catch (NumberFormatException e) {
8                   continue;
9               } catch (JSONException e) {
10                  continue;
11              }
12          }
13      }
```

Código 7.30: Métodos get y set de InfoProvider.class

Clase Place.class

La clase *Place* representa un lugar de una forma manejable para la aplicación, que incluye únicamente la información necesaria y que será mostrada al usuario. Su constuctor recibe el valor de esos parámetros y desde ese momento el lugar queda completamente definido. Aunque es necesario capturar las excepciones, éstas no deben producirse nunca, puesto que ya han sido capturadas en la el método *processPlaces* de *InfoProvider*, único lugar donde se llama a dicho constructor.

```
1  public class Place {
2
3  private String nombre;
4  private double lat;
5  private double lng;
6  private String summary;
7  private URL imagen;
8  private URL url;
9
10 public Place(String nom, double lat, double lng,String img,String
       web,String summary){
11     setNombre(nom);
12     this.setLat(lat);
13     this.setLng(lng);
14     this.setSummary(summary);
15     try {
16         setImagen(new URL(img));
17     } catch (MalformedURLException e) {
18         e.printStackTrace();
19             }
20
21     try {
22         setUrl(new URL("http://"+web));
23     } catch (MalformedURLException e) {
24         e.printStackTrace();
25     }
26
27 }
```

Código 7.31: Declaración y constructor de Places.class

El resto de los métodos de la clase son los getters y setters para cada uno de sus campos.

```
1  public void setNombre(String nombre) {
2      this.nombre = nombre;
3  }
4
5  public String getNombre() {
6      return nombre;
7  }
8
9  public void setLat(double lat) {
10     this.lat = lat;
11 }
12
13 public double getLat() {
14     return lat;
15 }
16
17 public void setLng(double lng) {
18     this.lng = lng;
19 }
20
21 public double getLng() {
22     return lng;
23 }
24
25 public void setImagen(URL imagen) {
26     this.imagen = imagen;
27 }
28
29 public URL getImagen() {
30     return imagen;
31 }
32
33 public void setUrl(URL url) {
34     this.url = url;
35 }
36
37 public URL getUrl() {
38     return url;
39 }
40
41 public void setSummary(String summary) {
42     this.summary = summary;
43 }
44
45 public String getSummary() {
46     return summary;
47 }
```

Código 7.32: getters y setters de Places.class

Clase UserOverlay.class

UserOverlay, que extiende a *ItemizedOverlay*, es la clase que permite situar al usuario sobre el mapa. El principal elemento de la clase es *mOverlays*, que será donde se guarde el elemento a posicionar. Los métodos *updateOverlay* y *addOverlay* son los que nos permitiran modificar esa lista, actualizando así con cada nueva posición. La sentencia *populate* es la responsable de pintar los puntos sobre el mapa.

```
1  public class UserOverlay extends ItemizedOverlay<OverlayItem> {
2
3      private ArrayList<OverlayItem> mOverlays = new ArrayList<
           OverlayItem>();
4      private Context context;
5
6      public UserOverlay(Drawable defaultMarker, Context context) {
7          super(boundCenterBottom(defaultMarker));
8          this.context = context;
9      }
10
11     public void updateOverlay(OverlayItem overlay) {
12         mOverlays.clear();
13         mOverlays.add(overlay);
14         populate();
15     }
16     public void addOverlay(OverlayItem overlay) {
17         mOverlays.clear();
18         mOverlays.add(overlay);
19         populate();
20     }
21
22     @Override
23     protected OverlayItem createItem(int i) {
24         return (OverlayItem) mOverlays.get(i);
25     }
```

Código 7.33: UserOverlay.class

Por otra parte, el método *onTap*, que se ejecuta cuando el usuario toca sobre la posición del *Overlay* en el mapa, mostrará un Toast informando de la dirección actual o, en caso de haber obtenido algún error, la coordenadas geográficas de la posición.

Los otros dos métodos permiten crear un nuevo Item, obtener el tamaño del Array de items y obtener y modificar el contexto.

```
@Override
    protected OverlayItem createItem(int i) {
        return (OverlayItem) mOverlays.get(i);
    }
    @Override
    public int size() {
        return mOverlays.size();
    }
    @Override
    protected boolean onTap(int i) {
        OverlayItem overlayItem = (OverlayItem) mOverlays.get(i);
        Toast.makeText(context, overlayItem.getSnippet(), Toast.
            LENGTH_SHORT).show();
        return true;
    }
    public Context getContext() {
        return context;
    }
    public void setContext(Context context) {
        this.context = context;
    }
```

Código 7.34: UserOverlay.class

Clase PlacesOverlay.class

PlacesOverlay.class es muy similar a *UserOverlay*, al igual que su objetivo. La diferencia es que ésta se usará sobre *Place* y además serán una lista de ellos, en lugar de un único elemento como en el caso de *UserOverlay*. Esta diferencia viene marcada a la hora de agregar los elementos en los métodos *drawUser* y *drawPlaces* de *GeoUpdateHandler*, ya que en el primero se usa *updateOverlay* y en le segundo *addOverlay*.

La novedad de esta clase reside en el método *onTap*, que de hecho es la única variación en el código. En este caso, en lugar de lanzarse un *Toast* se llama a la clase *InfoDialog.class*, que se muestra la información del *Place* al ser tocado sobre el mapa.

```
        @Override
        protected boolean onTap(int i) {
            //OverlayItem overlayItem = (OverlayItem) mOverlays.get
                (i);
            InfoDialog customizeDialog = new InfoDialog(context,
                InfoProvider.getPlaces().get(i));
            customizeDialog.show();
            return true;
        }
```

Código 7.35: Método onTap de PlacesOverlay.class

Clase InfoDialog.class

Esta clase, que extiende a *Dialog* e implementa *onClickListener*, permitirá, principalmente, mostrar la información resumida sobre el lugar elegido, aunque también será el elemento que mostrará la información sobre la aplicación al elegir la correspondiente opción en el menú de la aplicación. Los objetos que intervienen en ambos casos son similares:

```
1  public class InfoDialog extends Dialog implements OnClickListener
       {
2
3      private Button infoButton, cancelar;
4      private TextView text,mas,fecha;
5      private ImageView imagenview;
6      private Place place;
7      private String url;
```

Código 7.36: Declaración de InfoDialog.class

La diferencia viene dada por el constructor usado en cada caso, el más simple es la información de la aplicación. Tras fijar *place* como *null*, carga el layout correspondiente, recoge cada elemento por su "id", definido en el layout, establece el texto a mostrar por cada uno y asigna el *listener* a los botones. Además fija un color de fondo al elemento *text* para mejorar el efecto visual.

```
1  public InfoDialog(Context context) {
2      super(context);
3      place=null;
4      setContentView(R.layout.dialog);
5      this.setTitle(R.string.about);
6      infoButton = (Button) findViewById(R.id.info);
7      infoButton.setOnClickListener(this);
8      cancelar = (Button) findViewById(R.id.cancel);
9      cancelar.setOnClickListener(this);
10     cancelar.setText("OK");
11     text=(TextView)findViewById(R.id.summary);
12     text.setText(R.string.about1);
13     text.setBackgroundColor(Color.DKGRAY);
14     imagenview=(ImageView)findViewById(R.id.thb);
15     url=context.getResources().getString(R.string.homepage);
16     imagenview.setImageResource(R.drawable.logo);
17     mas=(TextView)findViewById(R.id.mas);
18     mas.setText(R.string.about2);
19     mas.setLinksClickable(true);
20     fecha=(TextView)findViewById(R.id.fecha);
21     fecha.setText(R.string.aboutFecha);
22    }
```

Código 7.37: Constructor para información de la aplicación

En el caso de informar sobre un *Place*, la información a mostrar se toma del propio objeto y hay que controlar el caso de que no exista imagen asociada a él.

```
1  public InfoDialog(Context context,Place elemento) {
2      super(context);
3      place=elemento;
4      setContentView(R.layout.dialog);
5      this.setTitle(place.getNombre());
6      infoButton = (Button) findViewById(R.id.info);
7      infoButton.setOnClickListener(this);
8      cancelar = (Button) findViewById(R.id.cancel);
9      cancelar.setOnClickListener(this);
10     text=(TextView)findViewById(R.id.summary);
11     text.setText(place.getSummary());
12     text.setBackgroundColor(Color.DKGRAY);
13     mas=(TextView)findViewById(R.id.mas);
14     mas.setText("");
15     fecha=(TextView)findViewById(R.id.fecha);
16     fecha.setText("");
17     imagenview=(ImageView)findViewById(R.id.thb);
18     url=place.getUrl().toString();
19
20     try {
21         if(place.getImagen()!=null){
22         HttpURLConnection img = (HttpURLConnection) place.
               getImagen().openConnection();
23         img.connect();
24         Bitmap imagen = BitmapFactory.decodeStream(img.
               getInputStream());
25         imagenview.setImageBitmap(imagen);
26         }
27     } catch (IOException e) {
28         e.printStackTrace();
29     }
30 }
```

Código 7.38: Constructor para información de un lugar

En ambos botones el *listener* será el mismo. *infoButton* cargará en el navegador la web asociada al cuadro de dialogo y lo establece como pestaña activa, *cancelar* simplemente cierra el marco de información.

```
1      @Override
2      public void onClick(View arg0) {
3          if (arg0==infoButton){
4              try {
5              InfoWeb.currentUrl=url;
6              InfoWeb.load();
7              }
8              catch  (NullPointerException e) {
```

```
9                 e.printStackTrace();
10            }
11            MainActivity.swichtTab(1);
12            }
13        this.dismiss();
14    }
```

Código 7.39: onClick listener de InfoDialog.class

En el layout de *InfoDialog* aparecen todos los elementos que se mostrarán al cargar la ventana de información. En primer lugar, los dos botones, especificando su tamaño, alineación, contenido e id. A continuación el cuadro de texto principal, contenido en un *TableLayout* para conseguir el efecto marco y, a su vez, contenido en un *ScrollView* para permitir hacer scroll en caso de que el texto sea mayor que el espacio disponible en el *TextView*.

Finalmente, aparecen otros dos *TextView*, que, como ya se ha visto, solo se usará para mostrar información sobre la aplicación.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout
3  android:id="@+id/widget32"
4  android:layout_width="fill_parent"
5  android:layout_height="fill_parent"
6  xmlns:android="http://schemas.android.com/apk/res/android">
7  <Button
8  android:id="@+id/cancel"
9  android:layout_width="85px"
10 android:layout_height="40px"
11 android:text="@string/cancelar"
12 android:textSize="12sp"
13 android:layout_below="@+id/info"
14 android:layout_alignParentLeft="true"/>
15 <Button
16 android:id="@+id/info"
17 android:layout_width="85px"
18 android:layout_height="40px"
19 android:text="@string/informacion"
20 android:textSize="12sp"
21 android:layout_below="@+id/thb"
22 android:layout_alignParentLeft="true"/>
23 <ImageView
24 android:id="@+id/thb"
25 android:layout_width="103px"
26 android:layout_height="71px"
27 android:layout_alignParentTop="true"
28 android:layout_alignParentLeft="true"/>
29 <ScrollView
30 android:id="@+id/scroll"
31 android:layout_height="110px"
```

```
32 android:layout_width="wrap_content"
33 android:layout_alignParentTop="true"
34 android:layout_alignParentRight="true"
35 android:scrollbars="vertical">
36     <TableLayout
37     android:id="@+id/TableLayout01"
38     android:layout_width="200px"
39     android:layout_height="130px"
40     android:background="#010101">
41         <TextView
42         android:id="@+id/summary"
43         android:layout_width="200px"
44         android:layout_height="130px"
45         android:text="TextView"
46         android:textSize="12sp"
47         android:layout_alignParentTop="true"
48         android:layout_alignParentRight="true"
49         android:layout_marginLeft="1px"
50         android:layout_marginBottom="1px"
51         android:layout_marginRight="1px"
52         android:layout_marginTop="1px"
53         android:padding="2px">
54         </TextView> </TableLayout> </ScrollView>
55 <TextView
56 android:id="@+id/mas"
57 android:layout_width="220px"
58 android:layout_height="40px"
59 android:text="TextView"
60 android:textSize="12sp"
61 android:layout_below="@+id/scroll"
62 android:layout_alignParentRight="true"
63 android:gravity="center"/>
64 <TextView
65 android:id="@+id/fecha"
66 android:layout_width="200px"
67 android:layout_height="20px"
68 android:text="TextView"
69 android:textSize="12sp"
70 android:layout_below="@+id/mas"
71 android:layout_alignParentRight="true"
72 android:gravity="center"/>
73 </RelativeLayout>
```

Código 7.40: Layout de InfoDialog.class

7.3.3. Clase InfoWeb.class

La clase *InfoWeb* representa la segunda actividad de la aplicación, que será visible en la segunda pestaña creada. El método *onCreate*, tras cargar el layout, establece las propiedades de zoom, escala y visualización adecuadas para que la web y el zoom tengan un correcto funcionamiento, finalmente carga la web inicial: la web oficial de la aplicación.

Además asigna un cliente web propio, mediente *setWebViewClient*, ya que de lo contrario las webs se abrirán en un navegador nuevo, el definido por defecto en el terminal, de forma que se desacoplaría de la ventana de la aplicación. A dicho cliente web se le añade el *onPageFinished*, el cual será responsable de cerrar la ventana mostrada mientra una web es cargada por completo.

```
1  public class InfoWeb extends Activity {
2
3     static String currentUrl="http://touristdroid.blogspot.com/";
4     static WebView web;
5     boolean back,forward;
6     private static ProgressDialog progressBar;
7     private static Context context;
8
9      @Override
10     public void onCreate(Bundle savedInstanceState)
11     {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.web);
14          web = (WebView) findViewById(R.id.webview);
15         context=this;
16
17         web.getSettings().setSupportZoom(true);
18         web.getSettings().setBuiltInZoomControls(true);
19         web.getSettings().setUseWideViewPort(true);
20
21         web.invokeZoomPicker();
22         web.setInitialScale(30);
23         this.load();
24
25         web.setWebViewClient(new WebViewClient()
26        {
27             public void onPageFinished(WebView view, String url) {
28             if (progressBar.isShowing()) {
29                 progressBar.dismiss();
30             }
31         }});
32     }
```

Código 7.41: Declaración y método onCreate de Infoweb.class

Dicha ventana de carga es mostrada por el método *load*.

Esta actividad contiene un menú independiente del ya visto en el mapa, y ofrece nuevas funciones al navegador, como ir atrás, adelante y recargar la página actual. El método que lo invoca es *onCreateOptionMenu*, ya visto anteriormente, pero en este caso el método *onPrepareOptionsMenu* se ejecuta para ajustar las propiedades de dicho menú.

Este método comprobara si las funciones "Atras"y "Adelante" están disponibles, según el historial almacenado en el navegador, y en función del resultado asignara como botones los correspondientes iconos activos o inactivos.

```
1  @Override
2  public boolean onPrepareOptionsMenu (Menu menu){
3        back=web.canGoBack();
4        forward=web.canGoForward();
5        if(!back){
6            menu.getItem(0).setIcon(R.drawable.icon_back_off);
7        }
8        else {
9            menu.getItem(0).setIcon(R.drawable.icon_back);
10       }
11       if(!forward){
12           menu.getItem(2).setIcon(R.drawable.icon_forward_off);
13       }
14       else {
15           menu.getItem(2).setIcon(R.drawable.icon_forward);
16       }
17    return true;}
```

Código 7.42: Ajustar propiedades de navegación

Las acciones consecuentes de las pulsaciones del menú se invocan en el *onManuOptionSelected*, pero basará su decisión en la posibilidad o no de realizar una navegación hacia detrás o hacia adelante. Si el usuario pulsa un botón no permitido no realizará dicha navegación y lanzara un Toast de advertencia.

El layout de esta actividad es sencillo, ya que solo contiene un *WebView* contenido en un *LinerLayout*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
<WebView xmlns:android="http://schemas.android.com/apk/res/android
    "
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"/>
</LinearLayout>
```

Código 7.43: Layout de InfoWeb.class

El layout del menú es similar al de la pestaña principal, variando solo los "id"y los iconos asignados incialmente, que serán los correspondientes a "activos".

7.3.4. Clase MyPreferences.class

La última actividad de la aplicación corresponde a la configuración de las preferencias. La clase que la implementa es muy sencilla, ya que unicamente agrega las propiedades especificadas en el layout.

```
public class MyPreferences extends PreferenceActivity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.layout.preferences);
        }
}
```

Código 7.44: Clase MyPreferences.class

Por tanto, lo realmene interesante en esta actividad es precisamente su layout. Contiene dos *ListPrefences* contenidos en sendos *PreferencesCategory*. Cada uno tiene especificado la *key* que deberá ser utilizada para extraer su valor, además de dos arrays, uno que contiene las cadenas que serán mostradas y otro con los valores que toma dicha *key* al estar seleccionada cada una de las opciones. Además tienen asignado un valor por defecto y la propiedad *persistent*, que permite que el valor sea almacenado y recuperado la próxima vez que la aplicación sea ejecutada.

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="@string/idioma">
    <ListPreference
            android:key="idi"
            android:title="@string/appIdioma"
            android:summary="@string/appIdiomaDesc"
            android:entries="@array/idiomas"
            android:entryValues="@array/idiomasCodes"
            android:persistent="true"
            android:defaultValue="ES"
            android:shouldDisableView="false" />
      </PreferenceCategory>
<PreferenceCategory android:title="vistasCat">
    <ListPreference
        android:key="vistas"
        android:title="@string/vistas"
        android:summary="@string/vista_summary"
        android:entries="@array/vistas"
        android:entryValues="@array/vistasCodes"
        android:persistent="true"
        android:defaultValue="MAP"
        android:shouldDisableView="false"
        />
</PreferenceCategory>
</PreferenceScreen>
```

Código 7.45: Layout de MyPreferences.class

7.3.5. AndroidManifest.xml

El Manifiesto es un archivo imprescindible en toda aplicación Android, debe tener exactamente este nombre y estar situado en el directorio raiz. De este fichero se extrae información esencial para una correcta ejecución. En este caso, la información que contiene es la siguiente:

- pckage: com.manubet.android.touristAndroid. Sirve de identificador único para la aplicación.
- VersionCode:1. Es la primera versión de la aplicación.
- VersionName:0.9. Nombre de la versión.
- Permisos: INTERNET y FINE_LOCATION
- minSdkVersion:3. Versión mínima del SDK requerida, pues hasta esa versión se ha probado su correcto funcionamiento.

- Logo de la aplicación.
- Librerías: uso de "com.google.android.maps".
- Actividades: contiene cuatro actividades, con sus correspondientes etiquetas. La princial (MAIN) y arracable (LAUNCHER) es MainActivity
-

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
      package="com.manubet.android.touristAndroid"
      android:versionCode="1"
      android:versionName="0.9">
      <uses-permission android:name="android.permission.INTERNET"
          />
      <uses-permission android:name="android.permission.ACCESS_FINE
          _LOCATION" />
      <uses-sdk android:minSdkVersion="3"/>
      <application android:icon="@drawable/logo" android:label="
          @string/app_name">
        <uses-library android:name="com.google.android.maps" />
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.
                    LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MapTab" android:label="@string/Tab
            1"></activity>
        <activity android:name=".InfoWeb" android:label="@string/
            Tab2"></activity>
        <activity
            android:name=".MyPreferences"
            android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

Código 7.46: Manifiesto de la aplicación

7.3.6. Especificación del idioma

Durante la implementación de la aplicación unicamente se ha hablado del idioma en las búsquedas, nunca del idioma de la aplicación. El motivo

es que la configuración del idioma se realiza gracias a la estructura de ficheros existente en el proyecto, en concreto los ficheros de la categoría *values*.

Dicha carpeta contiene los ficheros de cadena por defecto, sin embargo, existe la posibilidad de crear nuevas carpetas llamadas *values-xx*, donde xx representan las iniciales de un idioma concreto. La elección de una capeta fuente viene dada por la configuración del idioma del telefóno. Asi, si estuviera configurado en inglés se cargará *values-en*, francés *values.fr* y así sucesivamente. La carpeta sin ninguna terminación de idioma será el idioma por defecto, y se cargará cuando no se encuentre una carpeta adecuada a la configuración del terminal.

Podría llegarse más lejos incluso, definiendo variedades idiomaticas por países. Así, la carpeta *values-en-US* sería usada en caso de inglés de US.

En TouristDroid existen las carpertas *values*, en español, que será el idioma por defecto, y además *values-en*, *values-fr*, *values-po* y *values-de*, que definen el inglés, francés, portugués y alemán, respectivamente. Cada carpeta contiene 2 ficheros: *arrays.xml* y *strings.xml*, donde se incluyen los valores de cada tipo de variables en los respectivos idiomas, Aunque realmente, en el caso de los arrays, el contenido es el mismo en todos los idiomas. A continuación de incluye un ejemplo de dichos archivos

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3 <string name="error_proveedor">Informationsanbieter wurde nicht
      gefunden. Es wird erneut eine Verbindung hergestellt</string>
4 </resources>
```

Código 7.47: Ejemplo de diccionario de Strings

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3 <string-array name="vistas">
4             <item>Traffic</item>
5             <item>Map</item>
6             <item>Satellite</item>
7     </string-array>
8 </resources>
```

Código 7.48: Ejemplo de diccionario de Arrays

CAPÍTULO 8

Pruebas y conclusiones

Una vez completado el desarrollo de una aplicación es necesario llevar a cabo una fase de pruebas que, aunque no garanticen totalmente el correcto funcionamiento de la aplicación, sí podremos decir que, al menos, en las condiciones que han sido supervisadas el resultado ha sido positivo.

El motivo es que es imposible desarrollar una batería de pruebas para experimentar absolutamente todas las posibilidades de funcionamiento y todas las condiciones del sistema. Además, el periodo de pruebas también supone un coste, por lo que habrá que decidir la fiabilidad que deseamos ofrecer, basándose en un equilibrio coste-garantía. Estas pruebas no comprueban únicamente el funcionamiento de la aplicación final, lo que se conoce como pruebas de integración, sino también el funcionamiento parcial de de cada funcionalidad que se va incorporando al sistema, pruebas unitarias.

De lo anterior se desprende que las pruebas no se llevan a cabo únicamente al finalizar el desarrollo, sino que es una operación necesaria a lo largo de todo el proceso. Inicialmente serán casos más parecidos a pruebas unitarias y progresivamente irán acercándose a las pruebas finales de integración.

Distinguiremos los dos tipos de pruebas en la documentación de éstas.

8.1– Pruebas unitarias

Para la realización de pruebas, el SDK de Android ofrece la herramienta de gestión de AVDs (Android Virtual Devices), de la que ya hablamos en el capítulo dedicado a la instalación del entorno de desarrollo. Este ha sido el método principal elegido en estas pruebas de unitarias, aunque también se han realizado algunas, más reducidas, en un terminal HTC WildFire.

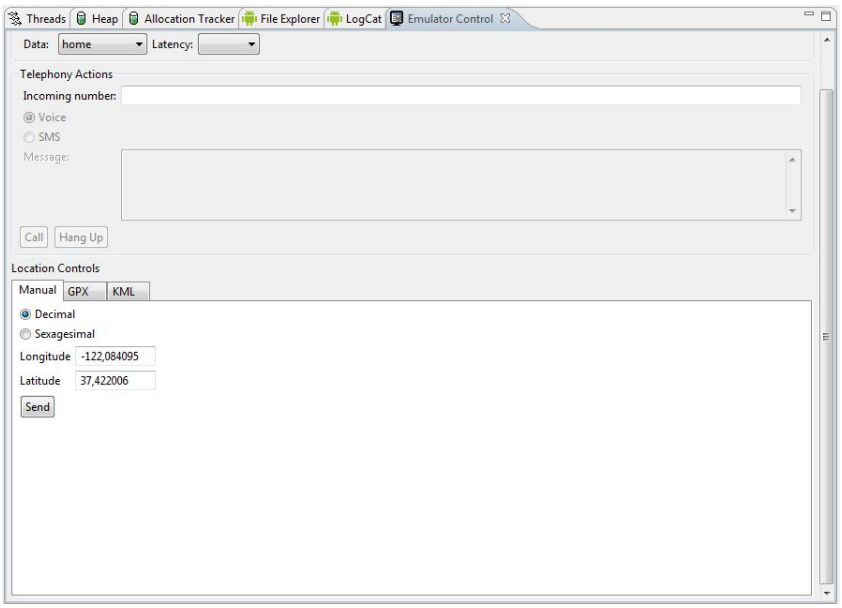


Figura 8.1: Perspectiva DDMS en Eclipse

El AVD elegido estaba basado en la versión Google APIs 2.1 Update1, que era la más avanzada en el momento de iniciar le desarrollo y además coincide con la versión instalada en el terminal HTC. Este terminal vitual cuenta con una pantalla tipo HVGA y lleva instalado el dispositivo GPS. La simulación del GPS se realiza obligatoriamente enviando coordenadas al receptor, ya sea mediante Eclipse o mediante la linea de comandos. El funcionamiento de la primera opción es trivial, activando la perspectiva DDMS de Eclipse. En el segundo caso hay que conectar usando telnet con el AVD mediante el comando "telnet localhost X", donde X es el ID asignado por el Sistema Operativo al AVD, normalmente 5554. Una vez conectado, el comando necesario es "geo fix [latitud] [longitud]".

En esta fase de pruebas unitarias se han simulado cada una de las funcionalidades que han sido desarrolladas, corrigiendo aquellas donde aparecia algún error o fallo, posibilitando incluso la implementación de mejoras que han surgido al descubrir un funcionamiento diferente al esperado inicialmente. Debido a su desarrollo de forma conjunta a la implementación, el tiempo utilizado no ha sido contabilizado en "Pruebas", si no en "Desarrollo"

Para estas pruebas unitarias también se ha usado, aunque en menor medida, el terminal HTC, debido mayormente a que el comportamiento del simulador no es fidedigno a un terminal real, sobre todo en tiempos de procesamiento y de respuesta, resultando en unos casos más rápido y en otros más lento. Incluso se han dado casos en los que el simulador pierde la

	Longitud	Latitud
Sevilla	-5.99	37.3862
París	2.35	48.86
Berlín 1	13.4083	52.5186
Berlín 2	13.40	52.51
Rio de Janeiro	-43.2330	-22.9
Helsinki	24.9333	60.1666
New York	74	40.71
Las Vegas	-115.1363	36.175
Pekin	116.3916	39.9138
Tokyo	139.715	35.7
Moscu	37.6177	55.7516
Sydney	151.2111	-33.8599
Abu Dhabi	54.3666	24.4666

Cuadro 8.1: Coordenadas usadas en las pruebas

conexión con la linea de comandos o con Eclipse, imposibilitando el envío de señales de geoposicionamiento o el acceso a Internet.

8.2– Pruebas de integración

Las pruebas de integración son las que determinan la calidad y fiabilidad final del sistema desarrollado, pues supone garantizar el correcto funcionamiento, de forma global, bajo las condiciones probadas. En primer lugar se ha probado el funcionamiento en el simulador bajo todas las versiones de la API. Realizando un flujo de operaciones normal: localización de un punto en el mapa, selección de un lugar de interés y petición de información completa de dicho lugar. Además se comprobará el correcto funciomamiento de, al menos, dos modos de vistas para el mapa. Sobre idiomas, se ha probado el funcionamiento en, al menos dos idiomas distintos, previo cambio de idioma predeterminado en el terminal HTC, así como búsquedas en el idioma local del país donde se realiza y en otro idioma distinto. En este punto se ha llegado a dar el caso de no encontrar ningún resultado, lo que ha motivado cambios en el algoritmo de búsqueda.

A continuación se comentan los resultados para las diferentes versiones, empezando por la más avanzada. De esta forma podremos concluir enunciando las compatibilidades de la aplicación.

En el cuadro anterior se muestran las coordenadas usadas en las pruebas.

- Google APIs 2.2 (API Level 8): Tras arrancar la aplicación con el terminal en inglés, se observa un funcionamiento normal. Se localiza

Figura 8.2: Simulación en API Level 8

el punto de Sevilla, devolviendo "Calle Cuesta de Rosario, 8, Seville" y se obtienen, en español, los puntos esperados, su información resumida en el cuadro de dialogo informativo y la información completa en el navegador.

A continuación se elige la vista "MAP" y se introducen las coordenadas correspondientes a Paris seleccionando búsquedas en inglés. La dirección obtenida "Rue de la Verrerie,42,4th arrondissement of Paris"y los resultados obtenidos se muestran en inglés.

El resultado es satisfactorio.

- Google APIs 2.1-update1 (API Level 7): Ha sido la versión utilizada para las pruebas unitarias, por lo que aunque en este apartado solo se comentan las pruebas de integración, su compatibilidad esta ampliamente testada.

  Trás iniciar la aplicación con el terminal en inglés, se introducen las coordenadas de Berlín, devolviendo la dirección "Sophienstraße,14,Berlin" y se obtienen los resultados esperados, en alemán, así como su correcta visualización.

  A continuación se cambia el idioma del terminal y se inicia la aplicación, que en este caso se muestra en español, se elige la vista satélite, español como idioma de búsqueda y se localiza un segundo punto en

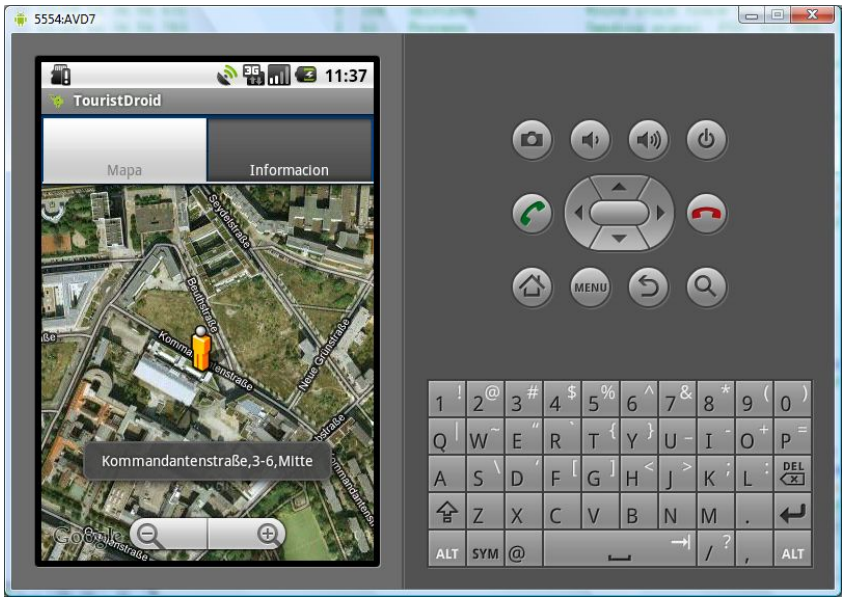


Figura 8.3: Simulación en API Level 7

Berlín. Ahora se posiciona sobre " Kommandantenstraße,3-6,Mitte" y los resultados se muestran en español.

La ejecución ha sido la correcta

- Google APIs 2.0.1-update1 (API Level 6): Se inicia el terminal en inglés y se introducen las coordenadas de la ciudad de Rio de Janeiro. El puntero se posiciona en "Av. Brasil,1818-1968,Caju" y lo resultados se muestran en inglés, a pesar de estar marcada la opción de idioma por defecto para búsquedas. Es decir, el sistema extrae BR como iniciales del país, pero al usar esas iniciales como idioma para lugares no recibe resultados, pues el idioma debería ser PO (Portugués), por tanto, el algoritmo de búsqueda decide usar inglés.

  Ahora se cambia el idioma del telefono a español, y tras cambiar la vista del mapa a modo Satélite y elegir inglés como idioma, se introducen las coordenadas de la ciudad de Helsinki, situandose en "Itäinen Vaihdekuja,4,Helsinki" y mostrandose los resultados en inglés.

  Los resultados son los esperados.

- Google APIs 2.0 (API Level 5): Se inicia la aplicación configurando alemán como el idioma en el temrinal y se introduce la localización de Nueva York. El usuario se posiciona en "Exchange Pl,72-88,Manhattan" y se muestran los resultados adecuados en inglés.

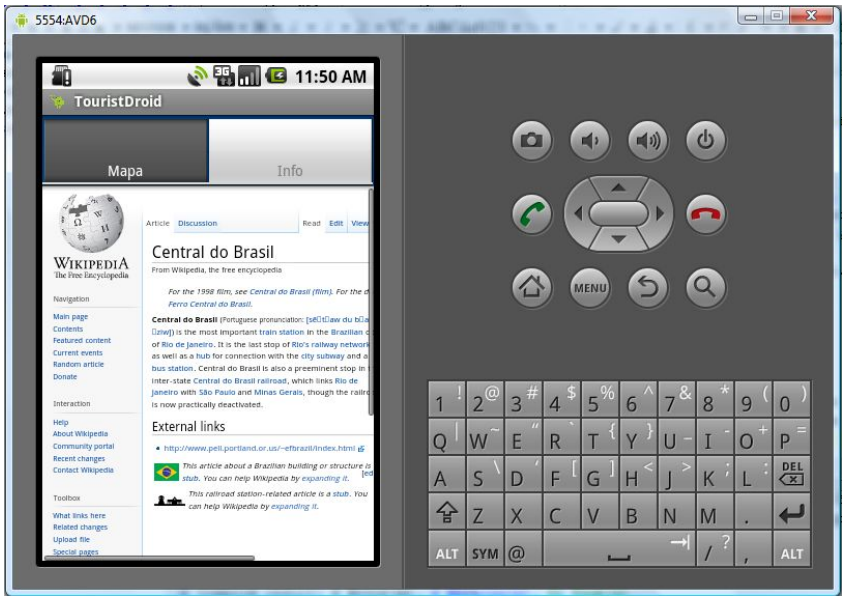


Figura 8.4: Simulación en API Level 6

Sin embargo, en un primer momento se muestra una advertencia que informa de la inexistencia de resultados. Esto es debido a que, al realizar la búsqueda de la dirección, se obtiene ÜSçomo país, y al buscar lugares adjuntando "US" como idioma se obtiene una respuesta vacia, ya que US es un país, no un idioma. Tras la advertencia, la búsqueda se repite, esta vez en inglés, y los resultados son los mostrados por pantalla.

Ahora se introducen las coordenadas geográficas de la ciudad de Las Vegas, previa selección de alemán como idioma de búsqueda y la visualización del mapa en modo Tráfico, soportada en esta ciudad. La dirección devuelta es "Las Vegas,Veterans Memorial Hwy,Las Vegas" y los resultados se muestran correctamente en alemán.

Por tanto, la aplicación también es compatible con esta versión de la API Android

- Google APIs 1.6 (API Level 4): Arrancamos la aplicación habiendo fijado español como idioma por defecto del terminal e introducimos las coordenadas de Tokyo. La dirección obtenida es ilegible, al igual que en debuger de Eclipse. Sin embargo los datos devueltos por la API Geocoder de Google si son resultados válidos, por lo que el defecto puede deberse al propio terminal HTC. Los resultados, a pesar de estar seleccionado el idioma de búsqueda por defecto, se muestran en inglés. El motivo es similar al caso de Nueva York:

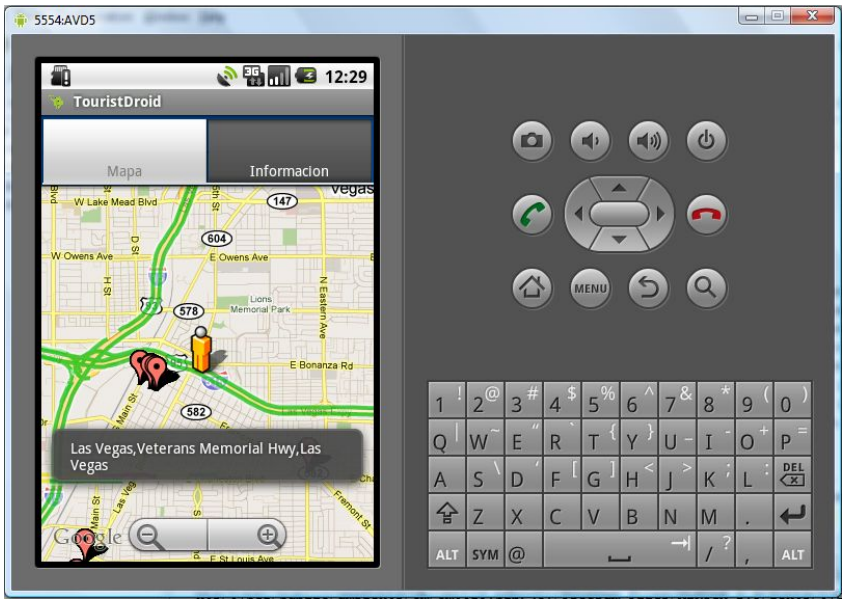


Figura 8.5: Simulación en API Level 5

Geocoder devuelve JP como codigo de país y Geonames reconoce JA como código del idioma, por lo que no puede obtener ningún resultado en el idioma nativo.

Ahora cambiamos el modo de vusualización y localizamos el mapa en Moscú y también español como idioma de búsqueda. El resultado se situa en"Tretyakovsky Proyezd,19-21/1,Tversko" y la información de puntos de interés es mostrada correctamente en español.

Se puede decir que la aplicación también es compatible con esta versión.

- Google APIs 1.5 (API Level 3): finalmente iniciamos la aplicación simulando la versión más antigua disponible en el simulador, configurando alemán como idioma por defecto. La posición introducida corresponde a la ciudad de Sydney, y el resultado es similar al caso de Nueva York: se obtienen las iniciales del país (AU) no corresponden a su idioma, por lo que, al no obtener resultados, la búsqueda se realiza en inglés. En este caso no se ha devuelto dirección alguna, unicamente informa de las coordenadas, dado que la localización introducida coincide con una zona de mar en la bahía de Sydney.

  Por último, elegimos francés como idioma de búsqueda y cambiamos a la vista Satélite. Al introducir coordenadas correspondientes a Abu Dahbi se localiza correctamente en "Al Rowdah, Al khaleej Al Arabi

Figura 8.6: Simulación en API Level 4

st, Abu Dahbi" y se muestran los resultados, aunque solo dos, debido al idioma elegido.

Nuevamente, la aplicación a resultado compatible con esta versión de la API Android.

También se han realizado varias pruebas reales sobre el terminal HTC WildFire de Vodafone y la versión 2.1 de la API instalada. Dichas pruebas consistieron en recorrer la Avenida de la Constitución de Sevilla, obteniendo durante el recorrido información cercana sobre los Reales Alcázares, la Catedral, la Giralda, la Real Maestranza o la Torre del Oro. El posicionamiento fue absolutamente correcto, al igual que la navegación por la información detallada de los lugares.

Adicionalmente, la aplicación ha sido instalada en diversos terminales, donde su comportamiento ha sido correcto, sin embargo, al no haber controlado las pruebas personalmente no se puede comentar exhaustivamente las operaciones realizadas. Estos terminales son los siguientes:

- HTC Desire, API 2.2.
- HTC Magic, API 1.6.

Nótese que también ha sido testado por parte de los usuarios que han obtenido la aplicación desde el Market de Android (se detallará más adelante), que hasta el momento de la publicación de este documento se han podido verificar 134 descargas.

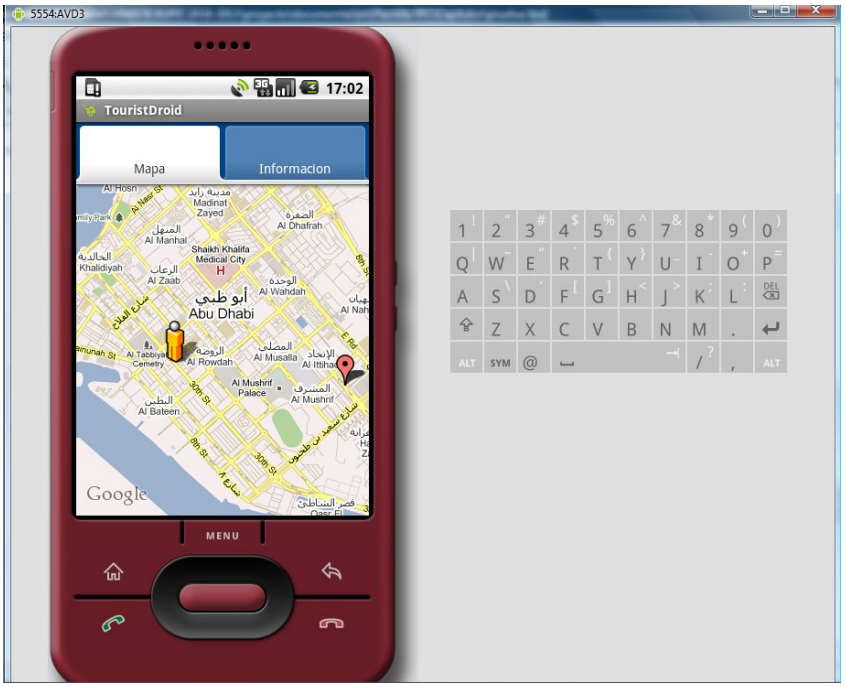


Figura 8.7: Simulación en API Level 3

8.3– Pruebas de visualización

Por úlitmo, se han realizado pruebas de visualización en todos los tipos de pantallas disponibles en la herramienta de simulación.

Android proporciona cuatro pautas recomedables para que la compatibilidad de la aplicaciones sea mayor. En primer lugar, se recomienda usar "fill_parent"y "wrap_content" a la hora de fijar el tamaño de las vistas, ya que de esta forma se permite un tamaño variable a nuestra ventana, pudiéndose acoplar adecuadamente a la pantalla. Por el mismo motivo, se recomienda no usar AbsoluteLayout. Además, este tipo de Layout fue declarado obsoleto a partir de API level 3.

	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*
Small screen	QVGA (240x320)		
Normal screen	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854)
Large screen		WVGA800* (480x800) WVGA854* (480x854)	

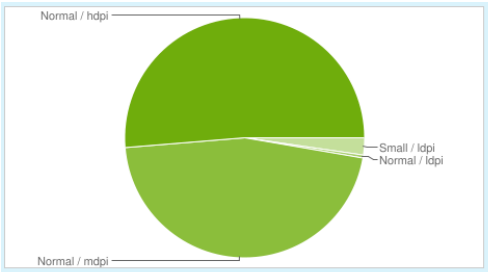
Figura 8.8: Tamaños de pantalla y densidades de skins disponibles en Android SDK

Figura 8.9: Visualización en la pantalla WQVGA432

Las otras dos recomendaciones, no usar medidas en pixels y usar recursos direfentes para las diferentes densidades de pantalla, no han sido seguidos totalmente en el diseño de la aplicación, aún asi, no han sido fuentes de errores. El primer caso, solo influiría en la visualición del Dialog de información, pero se observa que su funcionamiento es correcto. El segundo influye en la calidad de los recursos gráficos, por tanto, dado que los iconos son los únicos recursos usados y la nitidez en la visualización no es un parámetro importante para definir la calidad de la apliación, no se ha considerado necesario tenerlo en cuenta. De hecho, la diferencia en la visualización ha pasado desapercibida en las pruebas realizadas.

Los resultados han sido satisfactorios. Sin embargo, al igual que en las pruebas anteriores, esto no garantiza el correcto funcionamiento en todos los terminales, ya que es posible que algún fabricante lance pantallas con caracter'sticas que se salgan del estandar.

Si se han notado aspectos mejorables en 3 casos. En el primero de ellos, en WQVGA432 (240x432), el resultado aparece una pequeña franja negra en la parte inferior de la aplicación, no llegando ésta a ocupar toda la superficie de la pantalla. Aun así, la aplicación si se muestra correctamente. Según Android, este tipo de pantallas (normal size, low density) representan el 0.4 % del total del mercado, por lo que no supone un defecto significativo.



	Low Density	Medium Density	High Density
Small	2.3%		
Normal	0.4%	45.9%	51.2%
Large			

Data collected during two weeks ending on August 2, 2010

Figura 8.10: Distribución de tipos de pantalla en los terminales existentes.

Igualmente, en las pantallas Large Size, la visualización no ocupa toda la pantalla, pero es este caso, el número de dispositivos ni siquiera aparace en las estadísticas de Android.

Además, estos defectos no supondrán ningún inconveniente para el usuario, dado que según la API usada, versión 1.5, por defecto no se incluye compatibilidad para estos tamaños, por lo que el Market no permitira su descarga a esos usuarios. Dado el bajo impacto de los defectos podría haberse forzado la compatibilidad editando el Manifiesto, pero, debido al pequeño aumento de terminales compatibles, se ha decidido dar mayor importancia a la visualización, a priori, 100 % correcta que a llegar a un pequeño número de usuarios más.

8.4– Conclusiones

En función de los resultados obtenidos, se puede concluir que la aplicación desarollada es compatible con todas las versiones de la API de Android. Sin embargo, esto no quiere decir que funcione correctamente sobre todos los terminales, pues cabe la posibilidad de algún fabricante introduzca alguna modificación fuera del estandar y de al traste con la compatibilidad.

También podemos concluir que, en teoría, el 100 % de los usuarios que pueda descargar la aplicación obtendrá una correcta visualización, aunque añadiendo la misma incertidumbre del caso anterior.

CAPÍTULO 9

Manual

Una vez finalizada la implementación de la aplicación y habiendo probado su correcto funcionamiento, es necesario redactar un manual de uso, de forma que, aun teniendo un uso intuitivo, cualquier usuario sea capaz de familiarizarse fácilmente con la interfaz y los procedimientos.

Este manual, además será publicado en la web de la aplicación, hacia donde se dirigirán las peticiones de ayuda desde el menú de la aplicación.

9.1– Requisitos

Tourist Android es una sencilla aplicación destinada a la localización del turista mediante GPS, montrando su posición sobre un mapa y aconsejándole lugares interesantes para visitar, mostrando información asociada a ellos.

La localización se realiza mediante el dispositivo GPS integrado en el terminal móvil, por lo que hay que asegurarse de su disponibilidad antes de usar la aplicación. En caso de no estar disponible se mostrará un mensaje indicando la imposibilidad de localizar al usuario.

Además, es necesaria una conexión a Internet para obtener la información deseada, ya sea mediante una red WIFI o a través de la conexión de datos ofrecida por el operador de telefonía móvil. Si no dispone de conexión no se mostrará ninguna advertencia, pero el funcionamiento no será el correcto. En primer lugar, será imposible mostrar el mapa de situación, por lo que el usuario será situado en una pantalla gris sin ningún tipo de detalles. Además, no podrá obtenerse ningún lugar de interés turístico, y por tanto tampoco será posible navegar a través de su página de información.

NOTA: La señal del GPS solo estará disponible en lugares abiertos, por lo que en edificios o cualquier espacio que represente obstáculos para la recepción no se conseguirá un correcto funcionamiento.

9.2– Funcionamiento

Una vez iniciada la aplicación correctamente, el dispositivo GPS captará la posición del usuario, que será mostrado sobre un mapa. Junto al icono que representa al usuario aparecerán una serie de puntos que corresponden a lugares de interés que tienen asociados algún tipo de información. Haciendo click sobre ellos se mostrará un resumen sobre una nueva ventana.

Esta ventana incluye nombre, sumario y foto, si estuviera disponible. Ahora se puede elegir entre obtener más información o volver al mapa en busca de otro lugar (Botones “Mas Info” o “Cancelar”)

Una vez elegido el lugar deseado, la información se mostrará sobre el navegador de la aplicación, situado en la otra pestaña.

El mapa soporta la funciones de zoom y desplazamiento, que permite además un movimiento libre al usuario.

Igualmente, el navegador soporta las mismas funcionalidades de zoom y desplazamiento, que por otra parte son necesarias para una cómoda lectura de la información. Además, la web mostrada es clickable, es decir, el usuario puede seguir los hiperenlaces mostrados. Sin embargo,el poder hacerlo no garantiza su correcto funcionamiento y visualización.

Además, pulsando la tecla “Menu”se mostrarán las funciones “Atras” y “Adelante”(si estuvieran disponibles) y “recargar”.

9.3– Personalización de la aplicación

Desde la pestaña inicial (Mapa) se puede acceder los dos parámetros configurables de la aplicación pulsando la tecla “Menu” y posteriormente “Ajustes”.

El primero de ellos es el idioma. Hay que reseñar que este cambio únicamente efectuará a las búsquedas y que no está recomendado cambiarlo, ya que se obtendrán mejores resultando optanto por el idioma automático, es decir, la lengua oficial del país donde estemos o, en su defecto, inglés, si el algoritmo de búsqueda lo considera adecuado.

En caso de no entender el idioma local, se recomienda, una vez mostrada la información detalla del lugar elegido, seleccionar un idioma de entre los disponibles en la columna de la izquierda, en la web mostrada.

Las otras dos opciones accesibles en el Menu son informativas. La “Ayuda” se dirigirá a la web de la aplicación, al articulo de ayuda Ayuda. El otro mostrará los créditos de la aplicación y si se desea, se dirigirá a la Homepage, desde donde se podrá acceder a las novedades publicadas.

9.4– Otras consideraciones sobre el idioma

TouristDroid está disponible hasta en seis idiomas (Español, Inglés, Francés, Italiano, Alemán y Portugués), pero el idioma no es configurable por el usuario. Al cargar la aplicación se mostrará automáticamente en el idioma definido en el terminal o, en caso de no estar disponible, en Español.

CAPÍTULO 10

Publicación de una aplicación

Tras el desarrollo de una aplicación, tanto en Android como en cualquier otro entorno, el siguiente paso es su publicación.

La plataforma Android ofrece dos posibilidades: dentro o fuera de Android Market. Android Market es la herramienta que Google pone a disposición de los desarrolladores para la publicación de sus aplicaciones, pero su uso no es obligatorio. Veremos ambas posibilidades.

10.1– Publicar fuera del Market

Se trata de la forma convencional de dar a conocer tu aplicación, y además es fácil y barato. De esta forma depende del propio desarrollador la distribución de la aplicación, incluyendo tanto ponerla a disposición del usuario, publicitarla y adoptar un canal de comunicación con el usuario final, de forma que exista un intercambio de información sobre fallos, consejos, ayuda y actualizaciones, entre otras cosas.

Consiste principalmente en compilar la aplicación y subir el archivo .apk resultante a alguna web para que pueda ser descargada. La promoción puede llevarse a cabo en redes sociales, foros, o mediante una página especifica donde, además, se puede proporcionar información acerca de novedades sobre la aplicación.

Esta compilación debe realizarse en Modo Release. Para conseguirlo, basta seleccionar “Export Unsigned Aplication Package” dentro del menú “Android Tools”, pulsando con el botón derecho sobre la carpeta principal de nuestro proyecto en Eclipse. El archivo generado es que el debe ponerse a disposición del usuario para su descarga.

Sin embargo, el usuario debe configurar su terminal para aceptar aplicaciones que no hayan sido descargadas del Market, por lo que puede suponer un inconveniente.

Otros inconvenientes vienen del lado del control de la aplicación, ya que no se podrá vender, al menos no tan fácil como en el Market, ni se podrá insertar publicidad como fuente de beneficios. Además, no será tan inmediato saber el número de descargas ni notificar al usuario sobre actualizaciones.

10.2– Publicar en Android Market

La otra forma de publicar una aplicación es hacerlo en el Market dispuesto para tal fin. Supone la forma más natural, útil y, a la larga, más fácil y beneficiosa. Será por tanto la forma elegida para la publicación de este proyecto.

Este método, a pesar de tener un coste de 25$, permitirá vender nuestra aplicación, insertarle publicidad, y tener acceso a una red de distribución mucho más extensa de lo que seguramente podríamos lograr mediante la autopromoción, dado que es una herramienta existente en todos los terminales Android.

El primer paso sera crear una cuenta en el Market accediendo a su web[1] con una cuenta Google e introducir la información requerida, incluyendo los datos de pago. Una vez confirmado podemos empezar a publicar aplicaciones.

Hay una pequeña diferencia con el caso anterior: en el Market las aplicaciones deben ser firmadas, por lo que al crear el .apk habrá que elegir "Export Signed Aplication Package"[2].

En este proyecto, además, es necesario generar una nueva key para la API Maps, de la misma forma que se hizo al inicio, pero esta vez usando la firma generada en este paso. De lo contrario los mapas dejarán de funcionar.

Una vez completados estos pasos, la aplicación ya puede ser publicada en Android Market.

Para posteriores actualizaciones, es importante hacer dos cambios en el Manifiest de la aplicación, de lo contrario el Market no aceptará la subida. Los parametros son los siguiente:

- android:versionCode="1". Hay que incrementar una unidad cada vez que se realice una actualización.

- android:versionName="1.0". Realmente no es obligatorio, pero se recomienda cambiar la numeración, de forma que sea orientativo para el usuario.

[1] http://market.android.com/
[2] Ver Apendice B

Este ha sido el método usado para publicar TouristDroid. Aunque al mismo tiempo, como se ha comentado, se ha creado una web oficial[3] que da soporte informativo, como por ejemplo, el ya comentado manual de la aplicación.

Gracias a una de las ventajas comentadas sobre el Market, se puede saber que, a pesar de que la actualización de los datos no se realice de forma periódica, TouristDroid ha alcanzado 134 descargas en las primeras 48 horas tras la publicación.

[3] http://touristdroid.blogspot.com

CAPÍTULO 11

Oportunidades de negocio

Más allá del Proyecto Fin de Carrera, la aplicación desarrollada pretende ser el modo de iniciarse en la programación del mundo Android, tanto como afición como, si fuera posible, como pequeño negocio dentro de este extenso mercado, donde las cifras son abrumadoras.

11.1– El mercado actual

El avance de los Smartphones es una realidad en los últimos meses. Según un reciente estudio de la consultora Gartner[1], las ventas de móviles en el tercer trimestre de 2010 han crecido un 35 % respecto al mismo periodo de 2009. En particular, los smartphones han incrementado sus ventas en un 96 %, suponiendo el 19.3 % del total de las ventas.

Dentro de este mercado, las ventas por sistema operativo han experimentado un importante cambio de tendencia. Aunque sigue estando liderado por Symbian, que mantiene a Nokia como pricipal fabricante, el avance de Android es notable. Su cuota de mercado ha pasado de un residual 3.5 % a un 25.5 % que lo situa en la segunda posición, pasando a liderar el influyente mercado estadounidense. Lo sigue iOS (del fabriacante Apple), que sufre un estancamiento, y RIM (de BlackBerry), que incluso retrocede. También destaca Microsoft, cuya estrategia parece no funcionar. Su reducida cuota del 7 % ha caido a la tercera parte, con un testimonial 2.8 %.

A las cifras de terminales hay que añadir el mercado propio de las aplicaciones. Android Market acumula más de 100.000 aplicaciones, aunque App Store va mucho más allá, acercandose a las 300.000. Según Screen Digest[2] , en 2013 podría llegar a un volumen de negocio de 100.000 millones

[1] http://www.gartner.com//it//page.jsp?id=1466313
[2] www.greenfieldscommunications.com

Worldwide Smartphone Sales to End Users by Operating System in 3Q10 (Thousands of Units)

Company	3Q10 Units	3Q10 Market Share (%)	3Q09 Units	3Q09 Market Share (%)
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
iOS	13,484.4	16.7	7,040.4	17.1
Research In Motion	11,908.3	14.8	8,522.7	20.7
Microsoft Windows Mobile	2,247.9	2.8	3,259.9	7.9
Linux	1,697.1	2.1	1,918.5	4.7
Other OS	1,214.8	1.5	612.5	1.5
Total	**80,532.6**	**100.0**	**41,093.3**	**100.0**

Figura 11.1: Ventas de smartphones por sistema operativo a usuarios finales en el tercer trimestre de 2010. En miles de unidades.

de euros.

Hablando en términos más cercanos, las principales opciones para intentar obtener beneficios por parte de los pequeños desarrolladores son las siguientes:

- Version inicial gratuita. Consiste en distribuir una versión inicial, de reducida funcionalidad o bien de tiempo de uso limitado, para conseguir aumentar el número de usuarios de la aplicaón y posteriormente, alcanzada cierta popularidad, liberar la versión completa, ya de pago.
- De pago desde el principio. En lugar de esperar un periodo inicial, la aplicación sería puesta a la venta directamente y con todas sus funcionalidades disponibles.
- Versión gratuita y versión de pago. Es una combinación de las dos opciones anteriores. La versión gratuita debe ser ser lo suficientemente util para gustar al usuario, y, al mismo tiempo, ofrecer una interesantes funcionalidades extras en la versión de pago.
- Donaciones. La aplicación se distribuye de forma gratuita, sin ninguna contraprestación, a la espera de que el interés de los usuarios sea tal que se decidan a hacer donaciones.
- Publicidad. Finalmente, la publicidad también es siempre una oportunidad, que incluso puede barajarse la opción de adoptarla de forma conjunta a alguno de los modelos anteriores.

11.2– El mercado de TouristDroid

Son muchas las aplicaciones que podemos encontrar dedicadas al sector turístico, más aún teniendo en cuenta su apertura a un público cada vez más numeroso, un sector del que Android no se mantiene al margen. Con sus pros y sus contras, aquí se analizan algunas aplicaciones similares.

- Places Directory. Es la más popular de todas. Se basa en una localización del usuario para mostrar lugares cercanos. La diferencia es que está más dedicada a necogios, tales como restaurantes, cafeterias o tiendas. Según su pagina web, soporta inglés y chino. Sin embargo, no ha sido posible probarlo, ya que no se ha podido encontrar en el Market.

- Wikitude. Esta aplicación es muy parecida a la desarrollada, y además permite búsquedas sobre multitud de bases de datos, tanto de interés cultural, como la propia Wikipedia, o comerciales, como Booking.com. Haciendo pruebas sobre la Wikipedia, en Sevilla se ha obtenido un menor número de resultados que, además, muestra la información detallada en el navegador por defecto instalado en el terminal, en lugar de hacerlo dentro de la propia aplicación.

- Layar. Al igual que Wikitude, también muy similar a TouristDroid, pero incluye acceso a más bases de datos que la anterior que, además, se encuentran mejor clasificadas por numerosas categorias.

- TravelDroyd. Es una aplicación con un concepto diferente, pues por consiste en una guia de viajes offline. De esta forma, podemos descargar previamente la guia correspondiente al lugar de destino para poder consultarla libremente, ahorrando el gasto que pueda suponer la conexión de datos.

En definitiva, además de éstas, existe gran cantidad de aplicaciones orientadas al sector, cada una con sus puntos clave y sus puntos débiles, que al igual de dar una idea de la competencia existente, también deja claro la gran demanda existente sobre el posicionamiento de lugares de interés, ya sean de una categoría en concreto, orientados hacia una ciudad o bien alguna combinación de varias.

Bibliografía

- Web oficial de Android:
  http://developer.android.com
- Web oficial de API Maps for Android:
  http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/index.html
- Web oficial de API AJAX Local Search:
  http://code.google.com/intl/es-ES/apis/maps/documentation/localsearch/devguide.html
- Web oficial de API AJAX Local Search:
  http://code.google.com/intl/es-ES/apis/maps/documentation/places/
- Uso de JSON en Java:
  http://www.json.org/java/
- Uso de la capa Wikipedia de GeoNames:
  http://www.geonames.org/export/wikipedia-webservice.html
- Googlegroups Desarrolladores Android:
  http://groups.google.com/group/desarrolladores-android
- Googlegroups Android Developers:
  http://groups.google.com/group/android-developers
- Googlegroups Android Developers:
  http://groups.google.com/group/android-developers
- Googlegroups Google Maps:
  http://groups.google.com/group/google-maps-api
- Googlegroups Google AJAX Search:
  http://groups.google.com/group/google-ajax-search-api

- Googlegroups GeoNames:
  http://groups.google.com/group/geonames
- StackOverFlow, foro de consulta para desarroladores en general:
  http://stackoverflow.com/
- Android-Spa, foro de desarrolladores Android:
  http://www.android-spa.com
- Wikipedia, la enciclopedia libre: http://www.wikipedia.org/
- Jaume Pomés Olesti, "Introducción a Android y sus aplicaciones". Proyecto de fin de carrera de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla. Tutor: José Ramón Portillo Fernández, departamento de Matemática Aplicada I. Sevilla, Septiembre de 2009.
- Fernando Antonio Caro Vega y Francisco Javier Delgado Villegas, "Clase LATEX para Proyecto Fin de Carrera". Proyecto de fin de carrera de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla. Tutor: José Ramón Portillo Fernández, departamento de Matemática Aplicada I. Sevilla, Septiembre de 2008.
- Roberto Calvo Palomino, "Desarrollo en Android v1.0". GSyC/LibreSoft, 17 de Junio de 2009.

Apéndices

A.- Obtención de una clave de API Maps for Android

Para el uso de esta API se requiere obtener una API-key, asociada con la firma de nuestra aplicación. Por decirlo de alguna forma, es un control de acceso a la API. Para obtenerla es necesario introducir el certificado MD5 de la firma electrónica de nuestra aplicación en un formulario dispuesto para tal fin[3].

Dicho certificado se obtiene mediante la herramienta keytool, en la linea de comandos:

keytool -list -keystore ruta\firma

donde "ruta" es la ubicación y firma es la el fichero keystore de la firma cuyo certificado se quiera obtener. A continuación se pedirá una clave, que por defecto es "Android".

Cabe distinguir entre dos tipos de API-key, que serán generadas según la firma utilizada por nuestra aplicación: Debug o Released.

B.- Firmas para una aplicación Android

La firma de una aplicación es un proceso esencial para el desarrollo, pues identifica inequívocamente al autor de ésta. Hay dos tipos de firmas: Debug y Release.

Al iniciar un proyecto, la firma utilizada es en modo Debug, y además el proceso es automático, por lo que el programador no debe preocuparse en ese sentido.El modo Debug sirve para el proceso de desarrollo de la aplicación, aunque también puede ser usada posteriormente bajo esta firma.

El uso de la firma en modo Release llega a la hora de publicar en el Market, como se ha comentado anteriormente. Para ello, hay que elegir la opción "Export Signed Package" en el menu "Android Tools".

Tras elegir el proyecto deseado, hay que decidir si deseamos utilizar una clave existente o crear una nueva. En caso de disponer ya de una firma el proceso es trivial. Si se quiere crear una nueva, habrá que determinar

[3]: http://code.google.com/intl/es-ES/android/add-ons/google-apis/maps-api-signup.html

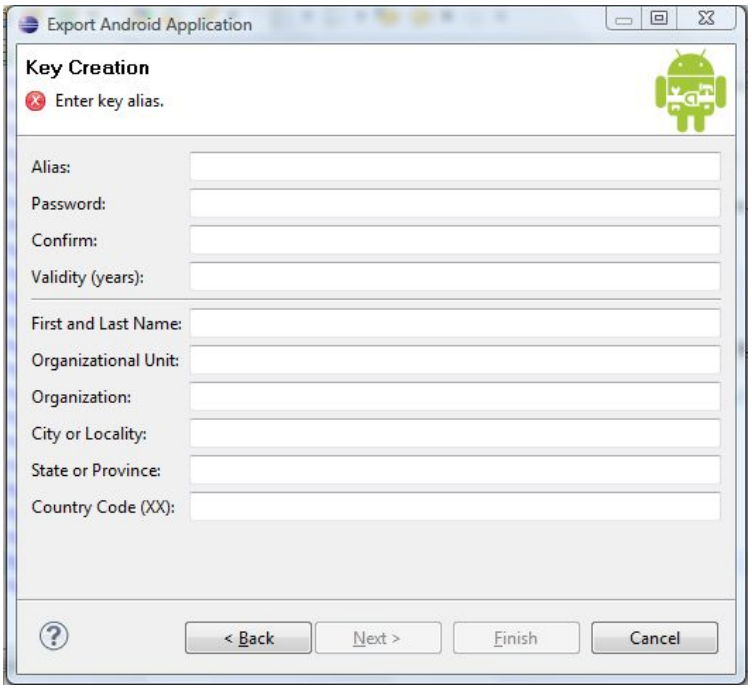


Figura 11.2: Creación de una firma en modo Release

una ubicación, contraseña, alias y una serie de datos personales. También habrá que especificar una validez, con un máximo de 50 años. Es importante guardar esta firma con seguridad, pues debe ser usada SIEMPRE para firmar las aplicaciones.

C.- Patrones JSON procesados: ejemplo

Como ya se ha explicado anteriormente, los resultados de las consultas realizadas son devueltos en formato JSON, que son procesados por la aplicación para extraer la información útil. A continuación se muestran dos ejemplos de los resultados devueltos, que pueden servir para entender visualmente el proceso realizado.

En primer lugar, la salida de una consulta a la API Geocoder para obtener la dirección de la posición 37.3862N 5.99W, cuya cadena HTTP corresponde a:

http://maps.google.com/maps/api/geocode/json?latlng=37.38950046666666,-5.991677666666668&sensor=true

Su respuesta será un JSONObject, cuyo segundo campo es un JSONArray que contiene la dirección buscada en varios formatos. Se muestra solo el primero, pues ha sido el utilizado para ser procesado.

```
{
  "status": "OK",
  "results": [ {
    "types": [ "street_address" ],
    "formatted_address": "Calle Cuesta de Rosario, 8, 41004 Sevilla
        , Espana",
    "address_components": [ {
      "long_name": "8",
      "short_name": "8",
      "types": [ "street_number" ]},
        {"long_name": "Calle Cuesta de Rosario",
      "short_name": "Calle Cuesta de Rosario",
      "types": [ "route" ]},
        {"long_name": "Sevilla",
      "short_name": "Sevilla",
      "types": [ "locality", "political" ]},
        {"long_name": "Sevilla",
      "short_name": "Sevilla",
      "types": [ "administrative_area_level_2", "political" ],
        {"long_name": "Andalucia",
      "short_name": "AL",
      "types": [ "administrative_area_level_1", "political" ]},
        {"long_name": "Espana",
      "short_name": "ES",
      "types": [ "country", "political" ]},
        {"long_name": "41004",
      "short_name": "41004",
      "types": [ "postal_code" ]} ],
    "geometry": {
      "location": {
        "lat": 37.3897519,
        "lng": -5.9916263},
      "location_type": "ROOFTOP",
      "viewport": {
        "southwest": {
          "lat": 37.3866043,
          "lng": -5.9947739},
        "northeast": {
          "lat": 37.3928995,
          "lng": -5.9884787} } }
  },...]}
```

Código 11.1: Ejemplo de respuesta de Geocoder, en formato JSON

A continuación se muestra un fragmento del otro tipo de objeto JSON procesado, resultante de la consulta sobre sitios cercanos a Geaonames. En este caso, el JSONObject es un JSONArray cuyos elementos contienen la información sobre un lugar. En este ejemplo se muestra únicamente el primer lugar correspondiente a la consulta partiendo de la mis-

ma posición del ejemplo anterior. La cadena HTTP de dicha petición es: http://ws.geonames.org/findNearbyWikipediaJSON?lat=37.38950046666666&lng=-5.991677666666668&lang=es&maxRows=20&radius=20

La respuesta obtenida es la siguiente:

```
1  {"geonames":[{"summary":", la catedral, la mas grande de Espana y
       verdadera joya   del arte almohade. Excepto el cuerpo superior
       , de construccion renacentista, la Giralda es el alminar de
       finales del siglo XII que pertenecio la antigua mezquita de
       Sevilla, emplazada en el lugar donde actualmente se encuentra
       la Catedral. A esta torre almohade se le anadio en epoca
       cristiana un remate para albergar las campanas (...)" ,
2  "distance":"0.3792" ,
3  "title":"Giralda",
4  "wikipediaUrl":"es.wikipedia.org/wiki/Giralda" ,
5  "elevation":0,
6  "countryCode":"ES",
7  "lng":-5.992480555555556,
8  "feature":"landmark", "thumbnailImg":"http://www.geonames.org/img/
       wikipedia/60000/thumb-59778-100.jpg",
9  "lang":"es",
10 "lat":37.38615,
11 "population":0} ...] }
```

Código 11.2: Ejemplo de respuesta de Geocoder, en formato JSON