ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA EN INFORMÁTICA

APLICACIÓN DE SEGUIMIENTO DE ENVÍOS DE UNA AGENCIA DE TRANSPORTE PARA ANDROID

Realizado por
JOAQUÍN VALONERO ZAERA

Dirigido por
ÁNGEL FRANCISCO JIMÉNEZ FERNÁNDEZ
MANUEL JESÚS DOMÍNGUEZ MORALES

Departamento
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Sevilla, Septiembre de 2013

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

1. Introducción.

Este documento constituye la memoria del Proyecto Fin de Carrera: "Aplicación de seguimiento de envíos de una agencia de transporte para Android". Proyecto que consiste en el desarrollo de una aplicación móvil, la cual permite a los empleados de una empresa de transporte y a sus clientes usar sus dispositivos para efectuar el control y seguimiento de sus envíos respectivamente.

La decisión de llevar a cabo y emprender este proyecto se debe principalmente a mi interés por aprender a desarrollar en dispositivos móviles así como a la actual aceptación e implantación en la sociedad de los mismos.

El mercado de la telefonía móvil es uno de los mercados más dinámicos dentro de la electrónica de consumo. Cuando hablamos de sistemas operativos móviles debemos mencionar en primera instancia a los grandes dominadores del mercado en la actualidad: iOS y Android. Entre ambos dominan la mayoría de estudios de cuota de mercado con cifras tan aplastantes como entre el 85% y el 90% de cuota de mercado conjunta que consiguieron en el último trimestre de 2012, según a que estudio hagamos caso. Este dominio del mercado de los sistemas operativos móviles no es fruto de una casualidad ya que son años los que ambos sistemas llevan batallando entre ellos. Fruto de esta batalla han conseguido despegarse con diferencia del resto de competidores. Android está mejor posicionado en la actualidad entre todos los sistemas operativos móviles y su dominio parece, de momento, incontestable.

Android es un sistema operativo Open Source (código abierto) con lo que no hay que pagar para desarrollar en él, aunque si se quiere publicar una aplicación en Google Play (tienda oficial de aplicaciones Android) si es necesario adquirir una licencia de desarrollador, cuyo coste es de 25 $ con una duración de por vida. Otro aspecto para mí decisivo a la hora de afrontar el proyecto, es el hecho de que las aplicaciones se pueden programar en Java, lenguaje que he estudiado durante mi formación en la Universidad. Por tanto, todos estos factores convierten a Android en el perfecto candidato para desarrollar la aplicación que nos atañe.

Para acabar con esta introducción, diré que llevar a cabo este proyecto me permitirá explorar las características técnicas de los dispositivos móviles actuales. Por supuesto, también me ayudará a la hora de implementar futuros desarrollos a empresas profesionales en la citada tecnología, elevando así mi experiencia como desarrollador.

1.1. Estructura del documento.

Obviando la pasada introducción, en este documento se incluyen los capítulos o apartados siguientes:

2. Objetivos:

Describiremos en este capítulo de forma general cuáles van a ser las metas a alcanzar por nuestra aplicación a desarrollar.

3. Planificación:

En este apartado incluimos la planificación establecida para la consecución del proyecto así como un análisis temporal de la misma.

4. Tecnología empleada:

En este capítulo se detallarán las herramientas, el software y la tecnología empleada para la realización del proyecto.

5. Desarrollo:

Se expone en este capítulo la descripción de cómo se ha llevado a cabo la implementación de nuestra aplicación.

6. Manual de usuario:

Incluimos en esta sección un sencillo manual que sirva como referencia para asistir a los usuarios de nuestra aplicación.

7. Conclusiones:

Este capítulo incluye un balance del trabajo desarrollado, los resultados obtenidos, conclusiones, posibles aplicaciones y objetivos futuros.

8. Apéndices:

Finalmente se incluye una recopilación de todo el código desarrollado.

9. Bibliografía:

Se citarán aquí las diferentes fuentes bibliográficas consultadas para la elaboración del proyecto.

2. Objetivos.

Como ya hemos comentado brevemente en la introducción este proyecto tiene como finalidad la realización de una aplicación para dispositivos móviles que permita efectuar un seguimiento de los envíos de una agencia de transporte. Para poder llegar a ese fin nuestra aplicación deberá alcanzar los siguientes objetivos:

- La aplicación a desarrollar deberá poder ser ejecutada y utilizada en terminales que incorporen sistemas operativos Android.
- Permitir emplear el dispositivo móvil a modo de señuelo para que éste envíe datos sobre su posición a una base de datos remota. Para ello, tendrá que hacer uso de las tecnologías del mismo dedicadas a tal efecto, como la conexión de datos y el GPS.
- Para que la aplicación sea totalmente efectiva, si está trabajando como señuelo deberá seguir enviando su posición a la base de datos aunque esté cerrada (sólo dejará de hacerlo cuando el usuario lo indique a través de la misma).
- Permitir realizar un seguimiento de los envíos indicando la posición y estado actual de los mismos sobre un mapa de Google. Para ello deberá acceder a la base de datos remota donde se guarda la información referente a los mismos.
- Por último, se entiende que los datos de ubicación recogidos por el dispositivo deberán ser todo lo fiables que permita el mismo.

Dicho esto, podemos pasar al siguiente apartado, donde como ya mencionamos al definir la estructura del documento, haremos una descripción de la planificación del proyecto y su análisis temporal.

3. Planificación.

Como es sabido, antes de comenzar el desarrollo de una aplicación a mediana o gran escala es casi de obligado cumplimiento el realizar una planificación previa del mismo. Para ello, debemos de tener en cuenta las tareas o hitos en los que podemos dividir nuestro proyecto. Debemos contemplar también los recursos a emplear por cada una de estas tareas, la dependencia entre las mismas y su tiempo estimado de realización. Para tal efecto, existen en la actualidad múltiples herramientas, en nuestro caso emplearemos un diagrama de Gantt. Una vez construido obtenemos una visión global del proyecto y la estimación del tiempo necesario para llevarlo a cabo.

A continuación se incluye la planificación del proyecto. Esta también se adjunta en formato digital junto con el cd del proyecto en un archivo de Microsoft Office Project de nombre: planPFC.mpp.

TRABAJO: PROYECTO FIN DE CARRERA	
Tarea	Aplicación de seguimiento de envíos de una agencia de transporte para Android.
Descripción	Desarrollar una aplicación para el sistema operativo Android que permita establecer un control y seguimiento de los envíos de una empresa de transporte.
Duración estimada	180 horas (18 créditos).
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Disponer de la propuesta inicial del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Productos	Aplicación Android.
Tareas predecesoras	Todas.
Tareas sucesoras	Ninguna.
Riesgos	No satisfacer la propuesta inicial del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Tarea 1	Estudio previo del problema (búsqueda de objetivos, requisitos, etc…).
Descripción	Búsqueda de ideas, del material a estudiar y la tecnología a emplear para llevar a cabo el proyecto.
Duración estimada	10 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Disponer de la propuesta inicial del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Productos	Objetivos del proyecto.

Tareas predecesoras	Ninguna.
Tareas sucesoras	Todas las demás.
Riesgos	No entender la propuesta del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla. Error en los objetivos.
Tarea 2	Estudio de la tecnología Android.
Descripción	Estudio del desarrollo de aplicaciones para el sistema operativo Android.
Duración estimada	20 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos previos del lenguaje de programación Java.
Productos	Solución de implementación para la aplicación Android.
Tareas predecesoras	Tareas 1 y 3.
Tareas sucesoras	Tareas 3, 4, 5, 6, 7, 8 y 9.
Riesgos	No adquirir el conocimiento adecuado. No encontrar una solución para la implementación.
Tarea 3	Estudio de la tecnología PHP y MySQL.
Descripción	Estudio del desarrollo de scripts en PHP, repaso de la programación en HTML y de las bases de datos.
Duración estimada	10 horas
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos previos de programación web y bases de datos.
Productos	Solución de implementación para las páginas web alojadas en el servidor.
Tareas predecesoras	Tareas 1 y 2.
Tareas sucesoras	Tareas 2, 3, 4, 5, 6, 7, 8 y 9.
Riesgos	No adquirir el conocimiento adecuado. No encontrar una solución para la implementación.
Tarea 4	Descarga, instalación y configuración de las herramientas de trabajo.
Descripción	Descarga e instalación del IDE Eclipse, del SDK de Android y los plugins necesarios para el desarrollo.
Duración estimada	3 horas
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos de las herramientas de trabajo.
Productos	Entorno de trabajo configurado para el desarrollo.
Tareas predecesoras	Tareas 1 y 2.
Tareas sucesoras	Tareas 3, 5, 6, 7, 8 y 9.

Riesgos	El hardware no cumpla los requisitos de instalación de las herramientas de trabajo o éstas no se instalen correctamente.
Tarea 5	Implementación de la aplicación para el sistema Android.
Descripción	Desarrollo del código de la aplicación Android que satisfaga la propuesta inicial del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Duración estimada	50 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos de programación de aplicaciones en Java para el sistema operativo Android.
Productos	Código de la aplicación Android.
Tareas predecesoras	Tareas 1, 2 y 4
Tareas sucesoras	Tareas 3, 6, 7, 8 y 9.
Riesgos	Errores de código. Implementación incorrecta de la solución del proyecto.
Tarea 6	Diseño y creación de la base de datos MySQL.
Descripción	Diseño y creación de la base de datos remota usada por el proyecto, mediante el uso del panel phpMyAdmin del servidor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Duración estimada	7 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos de bases de datos.
Productos	Base de datos del proyecto.
Tareas predecesoras	Tareas 1 y 3.
Tareas sucesoras	Tareas 2, 4, 5, 6, 7, 8 y 9.
Riesgos	No adoptar un diseño que guarde la información necesaria para satisfacer las exigencias del proyecto.
Tarea 7	Escritura de los scripts PHP para manipulación de base de datos MySQL.
Descripción	Creación del código de las páginas que van a estar alojadas en el servidor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Duración estimada	30 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos de programación en HTML, PHP y bases de datos.
Productos	Código de las páginas web alojadas en el servidor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Tareas predecesoras	Tareas 1, 3 y 6.

Tareas sucesoras	Tareas 2, 4, 5, 7, 8 y 9.
Riesgos	Errores de código. Implementación incorrecta de la solución del proyecto.
Tarea 8	Revisión del trabajo, pruebas y corrección de incidencias.
Descripción	Revisión del trabajo realizado, búsqueda de incidencias, corrección de las mismas y pruebas de los resultados obtenidos.
Duración estimada	10 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Código de la solución del proyecto.
Productos	Código de la solución del proyecto actualizado y carente de incidencias de alta relevancia. Desarrollo de la implementación del proyecto finalizado.
Tareas predecesoras	Tareas 1, 2, 3, 4, 5, 6 y 7.
Tareas sucesoras	Tarea 9.
Riesgos	No satisfacer la propuesta inicial del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla.
Tarea 9	Escritura del presente documento y creación de las diapositivas para la exposición del proyecto.
Descripción	Escritura de la documentación relativa al proyecto, preparación de la presentación del mismo y creación de las diapositivas para la presentación de éste, por medio de software ofimático.
Duración estimada	40 horas.
Recursos	Propios, PC, conexión a Internet, servidor.
Personal	Alumno y tutores del proyecto.
Requisitos	Conocimientos previos de Ofimática.
Productos	Documentación escrita del proyecto y diapositivas para la presentación del mismo.
Tareas predecesoras	Todas las anteriores.
Tareas sucesoras	Ninguna.
Riesgos	Errores en la metodología a seguir para la escritura de la documentación y la presentación del proyecto.

Ilustración 1. Planificación del proyecto.

4. Tecnología empleada.

Empecemos a describir la tecnología que hemos considerado necesaria y empleada para la consecución de nuestros objetivos. Dado el volumen de esta sección la dividiremos por apartados desglosando la tecnología empleada e indicando en qué consiste, sus características y modo de empleo.

4.1. Android.

Exponemos a través de los siguientes puntos una visión sintetizada de Android, tecnología base de nuestro proyecto. Lo describiremos y comentaremos su historia, evolución, características y arquitectura.

4.1.1. ¿Qué es Android?

Android es un paquete de software diseñado principalmente para dispositivos móviles táctiles, y que consiste en un sistema operativo basado en Linux, librerías middleware y un conjunto de aplicaciones.

Impulsado por la Open Handset Alliance, uno de cuyos principales socios es Google, se presenta como una alternativa de código abierto a los SSOO tradicionalmente cerrados para dispositivos móviles.

En la actualidad es el SO líder en móviles y su implantación en tabletas avanza rápidamente. Además su uso también se está extendiendo a otros ámbitos como e-book readers, televisiones, etc…

4.1.2. Historia y evolución de Android.

Los orígenes de Android se remontan a 2003, cuando varios profesionales relacionados con la industria de la telefonía móvil fundaron Android Inc. en Palo Alto, con el objetivo de desarrollar software para móviles más eficiente y avanzado.

En 2005 Google adquiere la compañía y mantiene en nómina a sus fundadores. A partir de ese momento se suceden los contactos secretos con fabricantes de hardware y operadores móviles para conseguir apoyos para su nuevo sistema operativo móvil, abierto y basado en Linux.

A finales de 2007 se anuncia la creación de la Open Handset Alliance, un consorcio de empresas formado por Google, HTC, Samsung, Sprint, T-Mobile, Qualcomm e Texas Instruments, entre otros, cuyo objetivo es impulsar la creación de estándares abiertos en la industria de la telefonía móvil. Al mismo tiempo se presenta el primer producto de la OHA, el sistema operativo Android.

En octubre de 2008 se pone a la venta el terminal HTC Dream, el primer dispositivo comercial que funciona con Android.

El HTC Dream se distribuyó con la versión de Android 1.5, que por tanto fue la primera en estar disponible para el público.

Desde entonces se han desarrollado varias iteraciones de Android y ha habido tres cambios en el número principal de versión:

• Android 2.x introduce mejoras sustanciales frente a su antecesor, como soporte para Bluetooth, gestión unificada de las cuentas de usuario, fondos de pantalla animados, gráficos más eficientes con OpenGL 2.0, etc…

• Android 3.x se desarrolla en exclusiva para tabletas, con el objetivo de optimizar la interfaz de Android en estos dispositivos, ya que la versión 2 estaba pensada principalmente para pantallas pequeñas. Se introducen mejoras como la System Bar, la Action Bar, soporte para procesadores multinúcleo, etc…

• Android 4.x unifica las interfaces para móviles y tabletas en una versión común para ambos tipos de dispositivos. Además introduces mejoras como soporte para múltiples usuarios, compartición de datos por NFC, interfaces RTL, etc…

En la siguiente tabla se muestra el historial de versiones Android, éstas reciben el nombre de postres en inglés. En cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético:

ÍNDICE ALFABÉTICO	NOMBRE	VERSIÓN	FECHA DE LANZAMIENTO
A	Apple Pie (tarta de manzana)	v1.0	23/09/2008
B	Banana Bread (pan de plátano)	v1.1	09/02/2009
C	Cupcake (magdalena glaseada)	v1.5	30/04/2009
D	Donut (rosquilla)	v1.6	15/09/2009
E	Éclair (pastel francés)	v2.0/v2.1	26/10/2009
F	Froyo (yogur helado)	v2.2.x	20/05/2010
G	Gingerbread (pan de jengibre)	v2.3.x	06/12/2010
H	Honeycomb (panal de miel)	v3.x (sólo tabletas)	22/02/2011

I	Ice Cream Sandwich (sándwich de helado)	v4.0.x	19/10/2011
J	Jelly Bean (gominola)	v4.1/v4.2/v4.3	27/06/2012
K	Key Lime Pie (tarta de limón)	v5.0	En desarrollo

Ilustración 2. Versiones de Android.

4.1.3. Características de Android.

En la siguiente ilustración se pueden observar las características y especificaciones de la última versión del sistema.

DISEÑO DE DISPOSITIVO	La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0 y diseño de teléfonos tradicionales.
ALMACENAMIENTO	SQLite, una base de datos liviana, que es usada para propósitos de almacenamiento de datos.
CONECTIVIDAD	Android soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+ y WiMAX.
MENSAJERÍA	SMS y MMS son formas de mensajería, incluyendo mensajería de texto y ahora la Android Cloud to Device Messaging Framework (C2DM) es parte del servicio de Push Messaging de Android.
NAVEGADOR WEB	El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome. El navegador por defecto de Ice Cream Sandwich obtiene una puntuación de 100/100 en el test Acid3.
SOPORTE DE JAVA	Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner.

SOPORTE MULTIMEDIA	Android soporta los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.
SOPORTE PARA STREAMING	Streaming RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5). Adobe Flash Streaming (RTMP) es soportado mediante el Adobe Flash Player. Se planea el soporte de Microsoft Smooth Streaming con el port de Silverlight a Android. Adobe Flash HTTP Dynamic Streaming estará disponible mediante una actualización de Adobe Flash Player.
SOPORTE PARA HARDWARE ADICIONAL	Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.
ENTORNO DE DESARROLLO	Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse (actualmente 3.4, 3.5 o 3.6) usando el plugin de Herramientas de Desarrollo de Android.
GOOGLE PLAY	Google Play es un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un PC.
MULTI-TÁCTIL	Android tiene soporte nativo para pantallas capacitivas con soporte multi-táctil que inicialmente hicieron su aparición en dispositivos como el HTC Hero. La funcionalidad fue originalmente desactivada a nivel de kernel (posiblemente para evitar infringir patentes de otras compañías). Más tarde, Google publicó una actualización para el Nexus One y el Motorola Droid que activa el soporte multi-táctil de forma nativa.
BLUETOOTH	El soporte para A2DF y AVRCP fue agregado en la versión 1.5; el envío de archivos (OPP) y la exploración del directorio telefónico fueron agregados en la versión 2.0; y el marcado por voz junto con el envío de contactos entre teléfonos lo fueron en la versión 2.2.
VIDEOLLAMADA	Android soporta videollamada a través de Google Talk desde su versión HoneyComb.
MULTITAREA	Multitarea real de aplicaciones está disponible, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj, a diferencia de otros sistemas de la competencia en la que la multitarea es congelada

	(Como por ejemplo iOS, en el que la multitarea se limita a servicios internos del sistema y no a aplicaciones externas)
CARACTERÍSTICAS BASADAS EN VOZ	La búsqueda en Google a través de voz está disponible como "Entrada de Búsqueda" desde la versión inicial del sistema.
TETHERING	Android soporta tethering, que permite al teléfono ser usado como un punto de acceso alámbrico o inalámbrico (todos los teléfonos desde la versión 2.2, no oficial en teléfonos con versión 1.6 o inferiores mediante aplicaciones disponibles en Google Play (por ejemplo PdaNet). Para permitir a un PC usar la conexión de datos del móvil Android se podría requerir la instalación de software adicional.

Ilustración 3. Características de Android.

4.1.4. Arquitectura de Android.

Los componentes principales de la arquitectura del sistema son:

- **Aplicaciones:** las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones:** los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- **Bibliotecas:** Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.
- **Runtime de Android:** Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases

compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx".

- **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

Finalmente en la ilustración inferior se puede observar una síntesis de cómo está estructurada la arquitectura del sistema operativo Android.

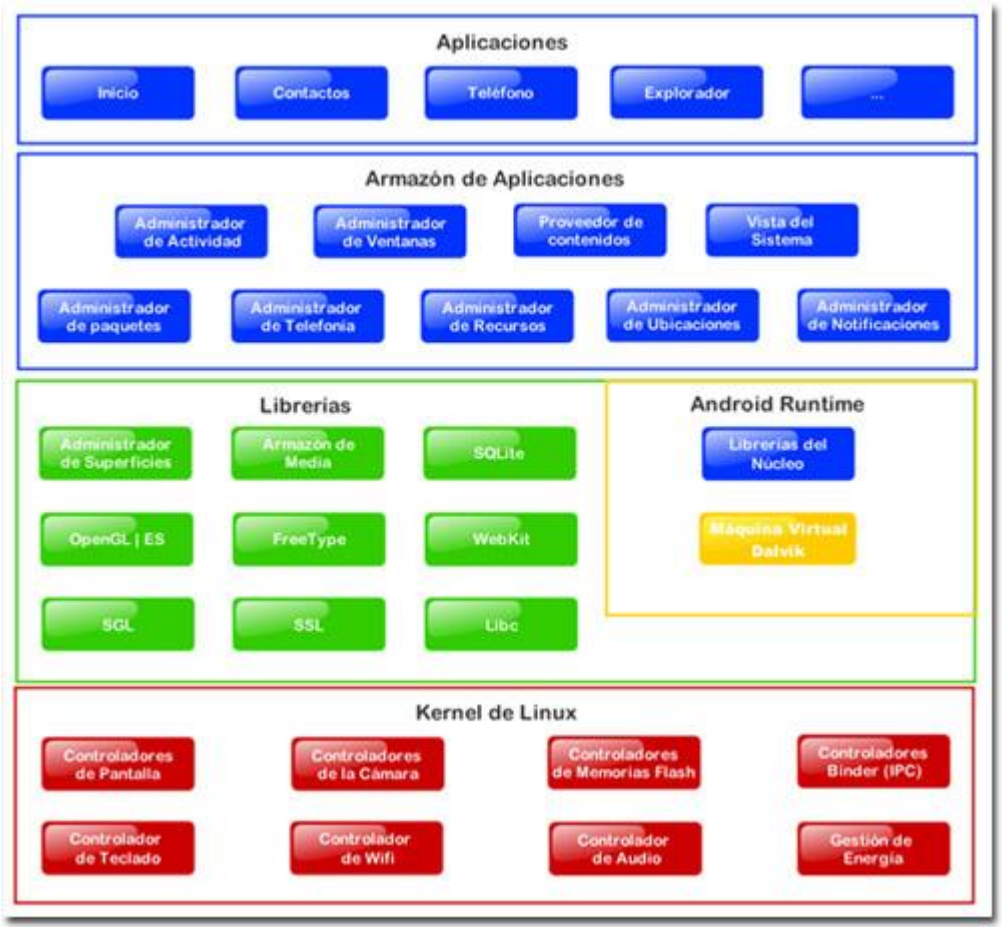


Ilustración 4. Arquitectura de Android.

4.2. Herramientas de desarrollo.

Existen diversas herramientas para el desarrollo en Android mencionamos aquí las más comunes:

- **Android SDK.**

El SDK (Software Development Kit) de Android, incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Max OS X 10.4.9 o posterior, y Windows XP o posterior. La plataforma integral de desarrollo (IDE, Integrated Development Environment) soportada oficialmente es Eclipse junto con el complemento ADT (Android Development Tools plugin), aunque también puede utilizarse un editor de texto para escribir ficheros Java y XML y utilizar comandos en un terminal (se necesitan los paquetes JDK, Java Development Kit y Apache Ant) para crear y depurar aplicaciones. Además, pueden controlarse dispositivos Android que estén conectados (ej. reiniciarlos, instalar aplicaciones en remoto).

Las Actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato .apk y guardada en el directorio /data/app del sistema operativo Android (este directorio necesita permisos de superusuario, root, por razones de seguridad). Un paquete APK incluye ficheros .dex (ejecutables Dalvik, un código intermedio compilado), recursos, etc.

- **Android NDK (desarrollo nativo).**

El NDK (Native Development Kit) permite instalar bibliotecas escritas en C y otros lenguajes, una vez compiladas para ARM o código x86 nativo. Los programas Java corriendo en la máquina virtual Dalvik (Dalvik VM) pueden llamar a clases nativas por medio de la función System.loadLibrary, que forma parte de las clases estándar Java en Android.

Se pueden compilar e instalar aplicaciones completas utilizando las herramientas de desarrollo habituales. El depurador ADB proporciona un Shell root en el Simulador de Android que permite cargar y ejecutar código nativo ARM o x86. Este código puede compilarse con GCC en un ordenador normal. La ejecución de código nativo es difícil porque Android utiliza una biblioteca de C propia (libc, llamada Bionic). Se accede al dispositivo gráfico como un framebuffer disponible en /dev/graphics/fb0. La biblioteca gráfica que utiliza Android para controlar el acceso a este dispositivo se llama Skia Graphics Library (SGL), disponible con licencia de código abierto. Skia tiene

implementaciones en win32 y Unix, permitiendo el desarrollo cruzado de aplicaciones, y es el motor de gráficos que soporta al navegador web Google Chrome.

- **Android Open Accessory Development Kit.**

La plataforma de Android 3.1 (portado también a Android 2.3.4) introduce soporte para Android Open Accesory, que permite interactuar a dispositivos USB externos (accesorios USB Android) interactuar con el dispositivo en un modo especial llamado "accessory". Cuando un dispositivo Android está en modo "accessory" el dispositivo externo actúa como hub USB (proporciona alimentación y enumera los dispositivos) y el dispositivo Android actúa como dispositivo USB. Los accesorios Android USB están diseñados específicamente para conectarse a dispositivos Android y utilizan un protocolo simple (Android Accessory Protocol) que les permite detectar dispositivos Android que soportan modo "accessory".

- **App Inventor para Android.**

Google anunció en julio de 2010 la disponibilidad de App Inventor para Android, que es un entorno de desarrollo visual Web, para programadores noveles, basado en la biblioteca Open Blocks Java, del MIT. Este entorno proporciona acceso a funciones GPS, acelerómetro y datos de orientación, funciones de teléfono, mensajes de texto, conversión habla a texto, datos de contacto, almacenamiento permanente, y servicios Web, incluyendo inicialmente Amazon y Twitter. Hal Abelson, director de proyecto en el MIT, dijo: "Sólo hemos podido hacerlo porque la arquitectura Android es tan abierta". Después de un año de desarrollo, la herramienta de edición de bloques se ha utilizado para enseñanza a principiantes en ciencias de computación en Harvard, MIT, Wellesley, y en la Universidad de San Francisco, donde el profesor David Wolber, desarrolló un curso de introducción a la ciencia de los ordenadores y un libro de enseñanza para estudiantes que no estudian computación, basado en App Inventor para Android.

- **HyperNext Android Creator.**

HyperNext Android Creator (HAC) es un sistema de desarrollo de programas dirigido a programadores que empiezan, permitiéndoles crear sus propias aplicaciones sin necesitar conocimientos de Java y del SDK de Android. Está basado en HyperCard, que gestiona el software como una pila de tarjetas en la que sólo una de ellas es visible en un momento dado y por tanto encaja bien en aplicaciones para teléfonos móviles, con una sola ventana disponible a la vez. El lenguaje principal de desarrollo se llama simplemente HyperNext y está relacionado con el lenguaje de HyperCards HyperTalk. HyperNext es un intérprete de un lenguaje similar al inglés y tiene muchas funciones para crear aplicaciones Android. Soporta un subconjunto creciente del SDK de Android incluyendo sus propias versiones de controles gráficos de interfaz de usuario (GUIs) y ejecuta automáticamente su propio servicio, de forma que las aplicaciones pueden continuar ejecutándose y procesando información, sin estar en el frontal del usuario.

- **El proyecto Simple.**

El objetivo de Simple es ser un lenguaje fácil de aprender para la plataforma Android. Simple es un dialecto de BASIC para Android. Sirve tanto para programadores profesionales como aficionados permitiendo escribir rápidamente aplicaciones que utilizan los componentes de Android.

Parecido a Visual Basic 6 de Microsoft, los programas Simple consisten en definiciones de formularios (que contienen componentes) y código (con la lógica del programa). La interacción entre ellos se hace por medio de eventos lanzados por los componentes. La lógica del programa consiste en gestores de eventos, que ejecutan código dependiendo del evento.

El proyecto Simple no tiene mucha actividad. La última actualización de código se realizó en agosto de 2009. Existe un producto comercial parecido llamado Basic4android, inspirado en Visual Basic 6 y Microsoft Visual Studio. Este proyecto si tiene actividad y hay una comunidad sólida de programadores.

El desarrollo de programas para Android se hace habitualmente con el lenguaje de programación Java y el conjunto de herramientas de desarrollo (SDK, Software Development Kit). Esta elección conlleva algunas ventajas e inconvenientes:

Ventajas:

- ✓ Los ejecutables son independientes de la plataforma, así que no hay que desarrollar una versión distinta para cada terminal.
- ✓ El lenguaje Java es uno de los más utilizados y con mayor base de programadores, por lo que existen gran cantidad de librerías disponibles y muchos recursos de ayuda.

Inconvenientes:

- × Java es un lenguaje interpretado, lo que implica un mayor consumo de recursos y menor velocidad de ejecución frente a otros lenguajes compilados.

Aunque la mayoría de las aplicaciones están escritas en Java, de forma nativa a través del SDK de Android, también podemos desarrollar aplicaciones web en HTML5, CSS3 y JavaScript o incluyendo código C o C++ a través del NDK.

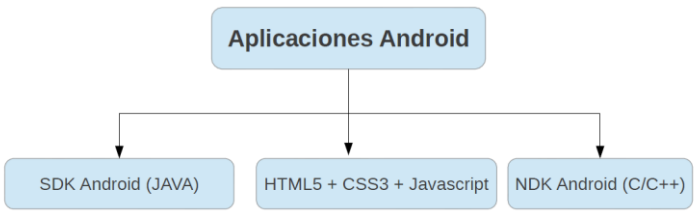


Ilustración 5. Aplicaciones Android.

Los programadores web con conocimientos HTML5, CSS3 y JavaScript también pueden desarrollar aplicaciones para Android sin conocimientos específicos en Java, como si lo hicieran de forma nativa. En la actualidad existen numerosos frameworks libres que vienen a cumplir con esta tarea y cada vez con más cuota en el número de desarrolladores. Algunos de ellos son:

- JqueryMobile (www.jquerymobile.com).
- PhoneGap (www.phonegap.com).
- SenchaTouch (http://www.sencha.com/products/touch).
- DojoMobile (http://dojotoolkit.org/features/mobile).
- Appcelerator Titanium (http://www.appcelerator.com).

A diferencia de otras plataformas, desarrollar con Android no supone costos en licencias y contamos con múltiples frameworks de desarrollo gratuitos y de código abierto.

El SDK de Android está a disposición de la comunidad en todas las plataformas existentes.

4.2.1. Ventajas e inconvenientes de aplicaciones nativas y aplicaciones web móviles.

Para acabar este punto y conociendo ahora el amplio abanico de herramientas y opciones de desarrollo que ofrece Android, veamos una comparación entre dos de las opciones más frecuentes.

APLICACIONES NATIVAS	
VENTAJAS	✓ El código es más rápido y se ejecuta directamente en el sistema operativo del dispositivo sin la necesidad de una aplicación de navegador intermediaria. Apuntar a distintos dispositivos requiere versiones completamente separadas de la misma aplicación debido a las diferencias en los lenguajes de programación y APIs en los diversos SDKs de plataformas. ✓ Pueden ser implementadas en una tienda de aplicaciones, ofreciendo un modelo de distribución simple y costeable. Implementar aplicaciones nativas mediante una tienda de aplicaciones normalmente requiere la aprobación de un tercero, lo que puede hacer considerablemente más lento el tiempo para llegar al mercado. ✓ Los SDKs nativos tienen incontables dispositivos específicos para el dispositivo, normalmente completos con documentación detallada y ejemplos. Las actualizaciones, incluso las pequeñas incrementales, requieren pasar por un proceso de aprobación, haciendo más difícil entregar actualizaciones críticas a los usuarios.

INCONVENIENTES	✖ Apuntar a distintos dispositivos requiere versiones completamente separadas de la misma aplicación debido a las diferencias en los lenguajes de programación y APIs en los diversos SDKs de plataformas. ✖ Implementar aplicaciones nativas mediante una tienda de aplicaciones normalmente requiere la aprobación de un tercero, lo que puede hacer considerablemente más lento el tiempo para llegar al mercado. ✖ Las actualizaciones, incluso las pequeñas incrementales, requieren pasar por un proceso de aprobación, haciendo más difícil entregar actualizaciones críticas a los usuarios.
APLICACIONES WEB MÓVILES	
VENTAJAS	✓ Escribir una aplicación para varias plataformas. No se requieren aplicaciones separadas para distintos dispositivos. Las aplicaciones distribuidas en la web para ser consumidas por los navegadores no tienen acceso a muchas funciones de dispositivo que las aplicaciones nativas sí tienen (cámara, micrófono, etc.), aunque esto puede cambiar en el futuro. ✓ Las aplicaciones pueden ser implementadas en la web sin una tienda de aplicaciones de terceros para su distribución. Necesita proporcionar la infraestructura para distribuir su aplicación, lo que puede ser más complicado y costoso que usar una tienda de aplicaciones. ✓ Usted mantiene todos los ingresos generados por sus aplicaciones (si cobra por ellas). Las aplicaciones nunca se ejecutarán tan rápido como sus contrapartes nativas, ya que son representadas en un navegador en lugar de ser ejecutadas por el sistema operativo mismo.
INCONVENIENTES	✖ Las aplicaciones distribuidas en la web para ser consumidas por los navegadores no tienen acceso a muchas funciones de dispositivo que las aplicaciones nativas sí tienen (cámara, micrófono, etc.), aunque esto puede cambiar en el futuro. ✖ Necesita proporcionar la infraestructura para distribuir su aplicación, lo que puede ser más complicado y costoso que usar una tienda de aplicaciones. ✖ Las aplicaciones nunca se ejecutarán tan rápido como sus contrapartes nativas, ya que son representadas en un navegador en lugar de ser ejecutadas por el sistema operativo mismo.

Ilustración 6. Ventajas e inconvenientes de aplicaciones nativas y aplicaciones web móviles.

4.3. Programación en Eclipse.

Por requerimientos de nuestro proyecto (uso del GPS entre otros) y sabiendo lo escrito en la sección anterior, desarrollaremos una aplicación nativa en Java empleando el SDK de Android con el plugin ADT para el IDE de Eclipse. Veremos a continuación, el procedimiento a llevar a cabo para poner en marcha y configurar nuestro entorno de trabajo.

4.3.1. Instalación de los programas necesarios.

Seguiremos el siguiente orden de instalación y configuración:

1. **Instalación del JDK (Java Development Kit).**

   Java Development Kit, o JDK, es un conjunto de herramientas que nos permiten compilar y depurar código escrito en Java.

   Las herramientas de Android utilizan las del JDK, por lo que el primer paso para configurar nuestro entorno siempre debe ser la instalación del último JDK disponible.

   Podemos obtenerlo gratuitamente en la página de Oracle: http://www.oracle.com/technetwork/java/javase/downloads

2. **Instalación del SDK de Android.**

   Android SDK es el conjunto de herramientas que nos va a permitir generar aplicaciones válidas para Android.

   Se puede obtener gratuitamente en la página de desarrolladores de Android:

   http://developer.android.com/sdk

   Además del SDK de Android, también debemos descargar los llamados SDK Targets de Android, que son librerías necesarias para desarrollar en cada una de las versiones concretas de Android.

   Por ejemplo, si queremos desarrollar para Android 2.2 Froyo, específicamente, tendremos que descargar los targets correspondientes a esta versión. Para ello accederemos al menú “Window / Android SDK Manager”, y en la sección “Available Packages” seleccionamos e instalamos los paquetes deseados.

   En nuestro caso hemos descargado los targets de la última API de Android ya que, queremos desarrollar para la última versión del sistema disponible. Esto como veremos más adelante en la sección de desarrollo no implica que nuestra

aplicación pueda ejecutarse perfectamente en versiones anteriores de Android siempre y cuando respetemos ciertas restricciones de compatibilidad.

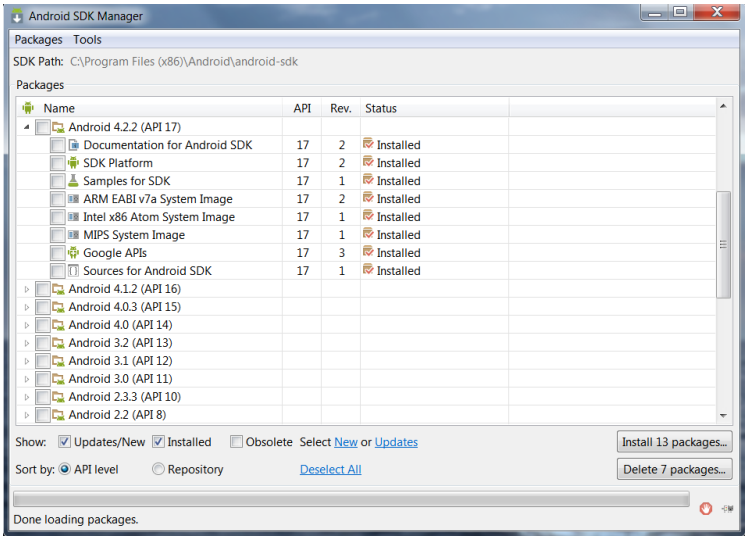


Ilustración 7. Android SDK Manager.

3. Instalación de Eclipse.

Eclipse es un entorno gráfico de desarrollo, es decir un IDE, que facilita la escritura de programas.

Para su uso con Android se recomienda instalar la *versión Eclipse IDE for Java EE Developers* o bien *Eclipse IDE for Java Developers*.

Se puede obtener gratuitamente en la página oficial de Eclipse: http://eclipse.org/downloads

La descarga consiste en un archivo comprimido en formato Zip. Para completar la instalación simplemente debemos descomprimirlo en una carpeta de nuestra elección. En nuestro caso hemos descargado la versión Eclipse Juno (v4.2.2).

4. Instalación del plugin ADT para Eclipse.

Google pone a disposición de los desarrolladores este plugin para Eclipse que permite integrarlo con las herramientas del SDK de Android, de modo que podremos acceder a ellas de forma muy cómoda.

Se recomienda instalar ADT utilizando el asistente de instalación de plugins de Eclipse. La dirección del repositorio es: https://dl-ssl.google.com/android/eclipse

Se debe seleccionar e instalar el paquete completo Developer Tools, formado por Android DDMS y Android Development Tools.

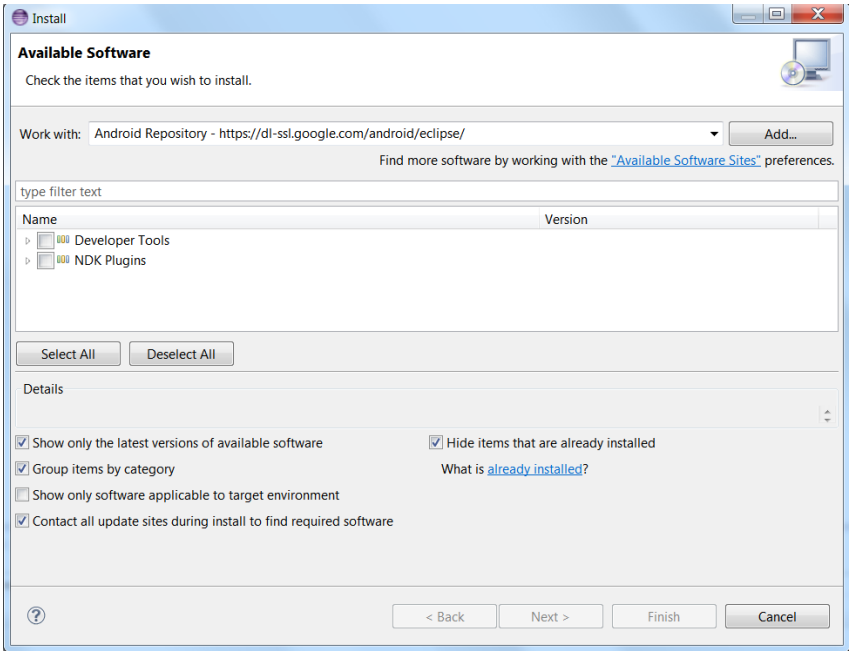


Ilustración 8. Instalación de plugins en Eclipse.

Una vez instalado, este plugin nos brinda las siguientes herramientas:

- **Android Project Wizard.- Genera archivos requeridos por el proyecto.**

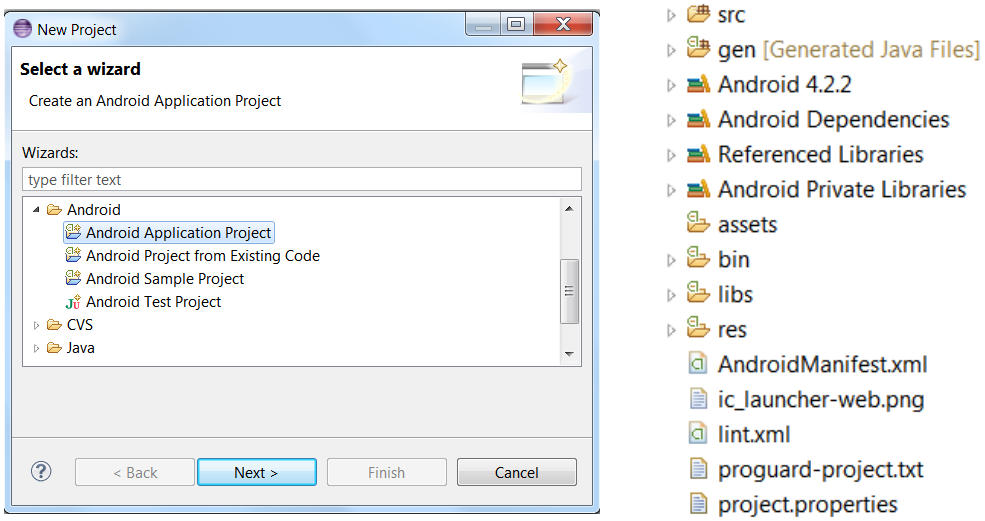


Ilustración 9. Android Project Wizard.

- **Editor de recursos.- Para diseñar interfaces de usuario.**

  Este visor de interfaces nos facilita la creación de las mismas, ya que permite ver los resultados sin necesidad de ejecutar la aplicación. Abajo se muestra una ilustración del mismo.

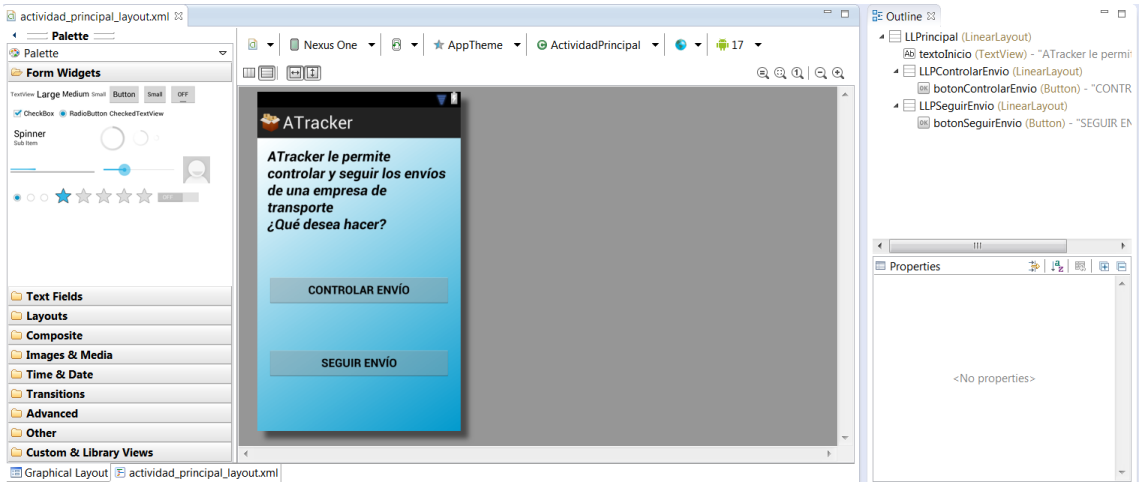
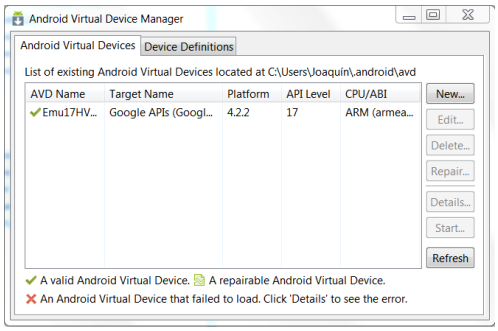


Ilustración 10. Editor de recursos de Android SDK.

- **Manager Android.- Para gestionar SDK y AVD.**

  Se añade la barra de herramientas a Eclipse que podemos ver en la ilustración de abajo.



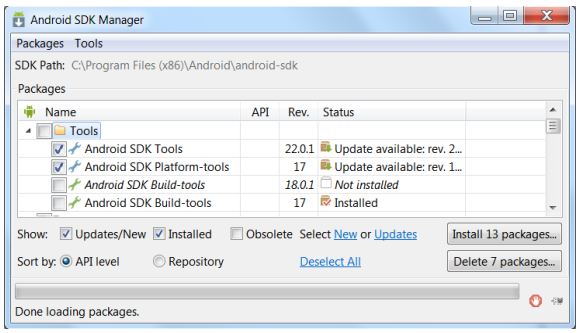


Ilustración 11. Manager Android.

- **Perspectiva DDMS.- Monitorizar y depurar aplicaciones Android.**

  Gracias al puente de depuración sobre Android que ofrecen las herramientas de desarrollo podemos disfrutar de la siguiente perspectiva, la cual nos facilita considerablemente el trabajo de depuración de nuestra aplicación.

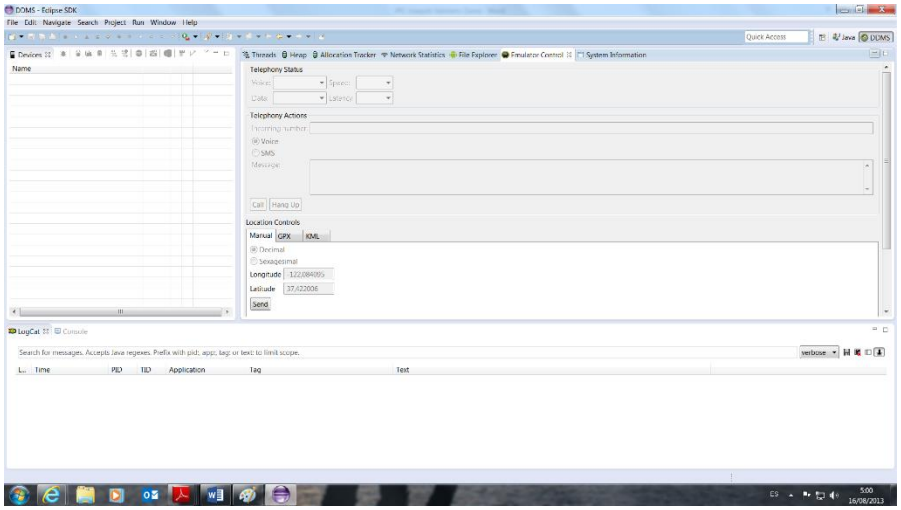


Ilustración 12. Perspectiva DDMS.

- **Construcción automatizada de aplicaciones para emuladores y dispositivos Android.**

  Los archivos se generan automáticamente en la carpeta /bin del proyecto.

- **Soporte ProGuard.- Para optimización de código y ofuscación.**

  Esta herramienta nos permite optimizar a la vez que ofuscar el código. En nuestro caso al ser un proyecto universitario no haremos uso de ella ya que deseamos que el código sea legible.

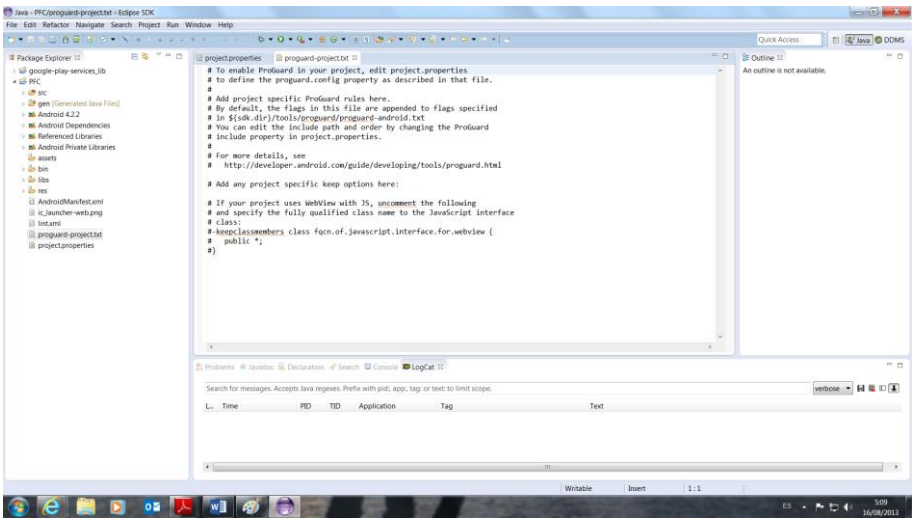


Ilustración 13. Soporte ProGuard.

Finalmente en la ventana de configuración de Eclipse, se debe acceder a la sección de configuración de Eclipse "Window / Preferences" e indicar la ruta en la que hemos descomprimido el SDK descargado en el paso 2.

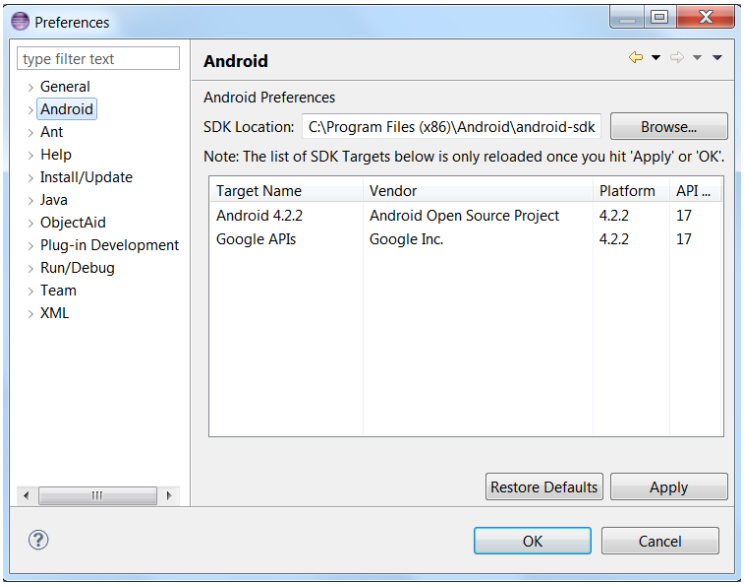


Ilustración 14. Preferencias proyecto Android.

5. Instalación simplificada con ADT Bundle y Android Studio.

Existe desde no mucho, a disposición de los desarrolladores de Android un método más sencillo para configurar el entorno de trabajo. Se trata del paquete ADT Bundle, que combina en un único instalador:

- Las herramientas de Android SDK.
- La última versión de Android disponible para compilar las aplicaciones.
- Una versión especial de Eclipse.
- El plugin ADT.

Únicamente tenemos que instalar de forma independiente el JDK, para lo que seguiremos el mismo proceso indicado en el punto 1.

Podemos obtener gratuitamente el instalador de ADT Bundle de la página de desarrolladores de Android: http://developer.android.com/sdk

La descarga de ADT Bundle consiste en un archivo comprimido en formato Zip. Para completar la instalación sólo tenemos que descomprimirlo en una carpeta de nuestra elección.

Por último, comentar también que recientemente desarrollado por parte de Google podemos encontrar en versión preview un IDE específico para Android basado en IntelliJ IDEA y conocido con el nombre de Android Studio.

4.3.2. Creación de un nuevo proyecto.

Una vez instalado todo lo necesario iniciamos eclipse indicando la ruta donde queremos que se aloje nuestro espacio de trabajo (workspace).

Con Eclipse arrancado crearemos un nuevo proyecto mediante el Android Project Wizard que anteriormente dijimos nos aporta el plugin ADT. Seleccionaremos por tanto, Android Application Project y elegiremos un nombre para el proyecto, el target deseado, el nombre de la aplicación, el paquete Java por defecto para nuestras clases y el nombre de la clase (activity) principal, opcionalmente también se puede elegir el icono que lanzará la aplicación.

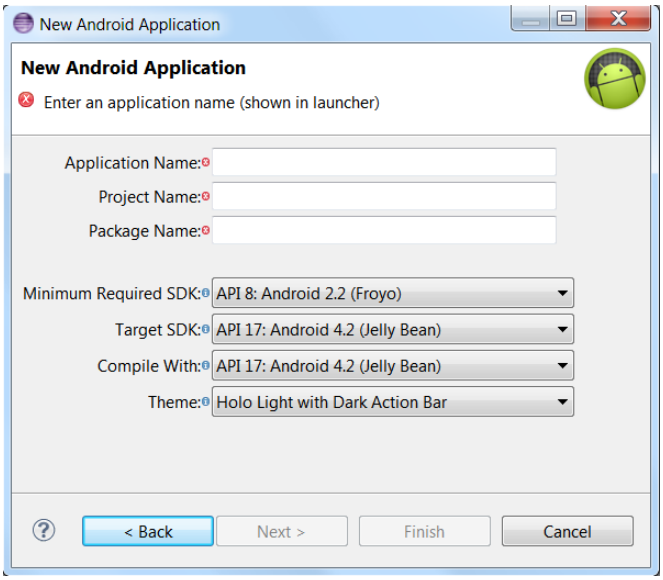


Ilustración 15. Nueva aplicación Android.

4.3.3. Creación de un emulador.

Para probar y depurar aplicaciones Android no tendremos que hacerlo necesariamente sobre un dispositivo físico, sino que podremos configurar un dispositivo virtual o emulador, para realizar estas tareas.

Para ello accedemos a “Window / AVD Manager”, y en la sección “Virtual devices” podremos añadir tantos dispositivos como deseemos, por ejemplo para distintas versiones de Android.

Seleccionaremos el target (versión) de Android que utilizará y sus características de hardware como resolución de pantalla, memoria SD, o la disponibilidad de funciones como el GPS.

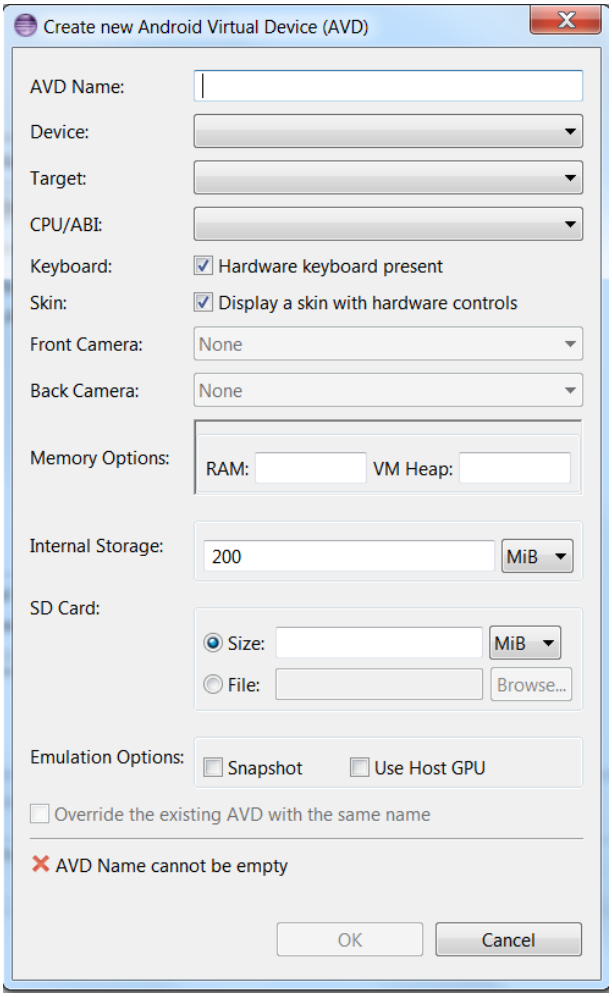


Ilustración 16. Nuevo AVD.

Evidentemente en función del tipo de aplicación que vayamos a crear necesitaremos de uno u otro emulador, aunque siempre es conveniente tener más de un emulador sobre todo si queremos que la aplicación funcione en varios dispositivos y que corra en versiones diferentes del sistema.

Aun así, hay veces en las que para nuestro desarrollo no basta con un emulador ya que por ejemplo, puede darse el caso de que nuestra aplicación tenga que hacer uso de sensores que se encuentren en nuestros dispositivos y que lógicamente no se pueden emular. Éste es el caso que nos atañe ya que nuestra aplicación usará el GPS, por eso y porque siempre es mejor la realidad que la ficción es muy recomendable disponer de dispositivos físicos reales para nuestro desarrollo. Veremos cómo conectarlos en el siguiente punto.

4.3.4. Instalación de un dispositivo móvil con fines de desarrollo.

Es muy recomendable por no decir imprescindible si se quiere realizar un desarrollo serio instalar un dispositivo móvil con fines de desarrollo. En este punto veremos cómo hacerlo puesto que la operación es bastante sencilla y carece de complicación alguna. Lo primero que necesitaremos para establecer la conexión de nuestro dispositivo es un driver (en Windows ya que en Linux no es necesario).

El driver ADB que es como se conoce, puede instalarse automáticamente en Windows al conectar el dispositivo o bien descargándolo directamente en la web del

correspondiente fabricante. También existen instaladores universales y Google provee un driver para los dispositivos Nexus a través del Android SDK Manager conocido como Google USB driver.

Finalmente antes de conectar el dispositivo, debemos activar el modo de depuración por USB en las opciones de desarrollo de su sistema Android. Con esto, ya podemos conectarlo y trabajar con él.

4.3.5. Estructura de un proyecto Android.

Cuando creamos un nuevo proyecto Android en Eclipse se genera automáticamente la estructura de carpetas necesaria para poder generar posteriormente la aplicación. Esta estructura será común para cualquier aplicación, independientemente de su complejidad.

Describiremos ahora que archivos se guardan en cada una de las diferentes carpetas del proyecto y cuál es la función de los mismos.

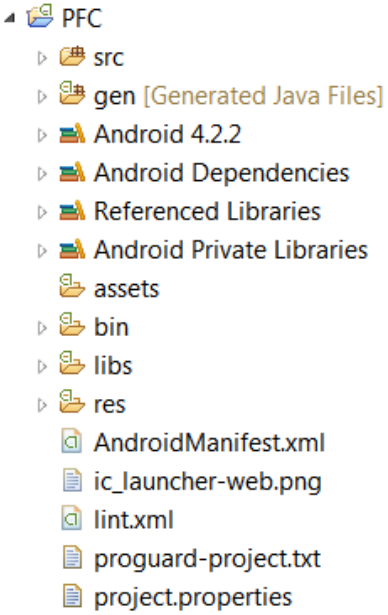


Ilustración 17. Estructura de un proyecto Android.

- **Carpeta de código fuente (/src).**

Contiene todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc… Inicialmente, Eclipse creará el código básico de la pantalla (Activity) principal de la aplicación, siempre bajo la estructura del paquete Java definido.

- **Carpeta de archivos Java auto generados (/gen).**

Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente en Java dirigidos al control de los recursos de la aplicación.

El más importante es el fichero R.java y la clase R. La clase R contendrá en todo momento una serie de constantes con los IDs de todos los recursos de la aplicación incluidos en la carpeta /res (que veremos más adelante), de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de

este dato. Por ejemplo, la constante R.drawable.ic_launcher contendrá el ID de la imagen "ic_launcher.png" contenida en la carpeta /res/drawable/.

- **Carpeta /assets.**

  Contiene los demás ficheros auxiliares necesarios para la aplicación (y que se incluirán en su propio paquete), por ejemplo, ficheros de configuración, de datos,…

  La diferencia con los ficheros incluidos en la carpeta /res/raw/ es que para éstos se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos.

  Para los contenidos en la carpeta assets no se generará ID y se podrá acceder a ellos mediante su ruta como a cualquier otro fichero del sistema.

  Usaremos uno u otro según las necesidades de nuestra aplicación.

- **Carpeta de binarios (/bin).**

  Como la carpeta /gen está carpeta se genera de forma automática al compilar el proyecto y contiene como su propio nombre indica los binarios de la aplicación incluyendo el paquete final de instalación .apk de la misma.

- **Carpeta de librerías (/libs).**

  Contiene librerías utilizadas por el proyecto, por defecto se añade la librería de soporte de Android (android-support-v4.jar) para facilitar la compatibilidad de las aplicaciones con versiones anteriores del sistema.

- **Carpeta de recursos (/res).**

  Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc.

  Estarán organizados por tipos, siendo la estructura de carpetas la siguiente:

  - Carpeta de animaciones (/res/anim).

    Contiene la definición de las animaciones utilizadas por la aplicación. Estas se suelen crear con ficheros XML.

  - Carpeta drawable (/res/drawable).

    Contiene las imágenes de la aplicación. Para utilizar diferentes recursos dependiendo de la resolución del

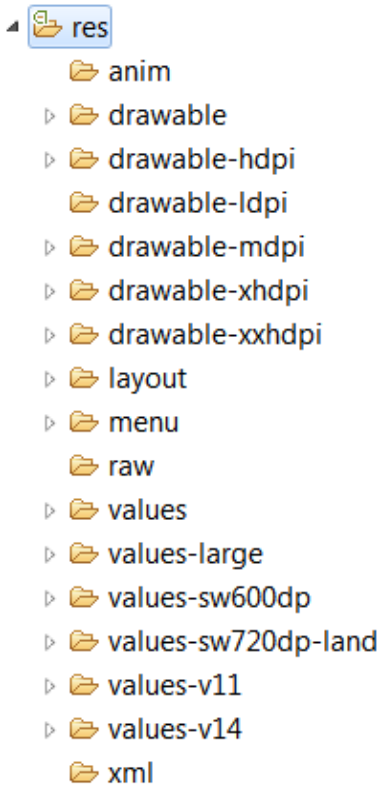


Ilustración 18. Carpeta de recursos de un proyecto Android.

dispositivo, se suele dividir en varias subcarpetas. Por ejemplo:

➔ /drawable-ldpi: baja resolución.

➔ /drawable-mdpi: resolución media.

➔ /drawable-hdpi: resolución alta.

- Carpeta layout (/res/layout).

Contiene los ficheros XML de definición de las diferentes pantallas de la interfaz gráfica.

Para definir distintos layouts dependiendo de la orientación del dispositivo se puede dividir en dos subcarpetas:

➔ /layout: orientación vertical.

➔ /layout-land: orientación horizontal.

- Carpeta de menús (/res/menu).

Contiene la definición de los menús de la aplicación.

- Carpeta raw (/res/raw).

Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de la carpeta de recursos. Se puede acceder a ellos en código mediante la función Resources.openRawResource().

- Carpeta values (/res/values).

Contiene otros recursos de la aplicación como, por ejemplo, cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc...

- Carpeta XML (/res/xml).

Contiene los ficheros XML utilizados por la aplicación. Se accede a ellos con la función Resources.getXML().

Como se puede apreciar en la imagen algunas carpetas tienen un modificador o cualificador. Estos modificadores se utilizan para distinguir los recursos en función de ciertas características de los dispositivos.

Puede encontrarse una tabla completa de los mismos y su función en la web de desarrolladores de Android:

http://developer.android.com/guide/topics/resources/providing-resources.html.

- **Manifiesto de la aplicación (AndroidManifest.xml).**

  Contiene la definición en XML de los aspectos principales de la aplicación, como su identificación (nombre, versión, icono...), sus componentes (pantallas, mensajes…), y permisos necesarios para su ejecución. Este fichero no puede faltar en ninguna aplicación Android ya que aporta información de la aplicación al sistema cuando se ejecuta en éste.

  El plugin ADT contiene un editor para este fichero llamado AndroidManifest Editor.

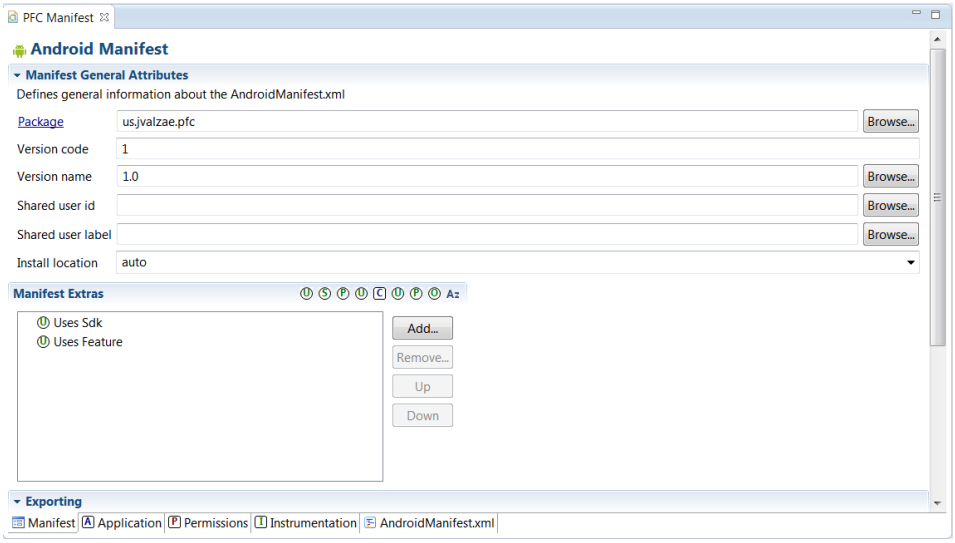


Ilustración 19. AndroidManifest Editor.

4.4. Componentes de una aplicación en Android.

Las aplicaciones Android están construidas siguiendo un modelo basado en componentes. Esto significa que cada aplicación contiene uno o varios bloques independientes que realizan tareas específicas y pueden comunicarse entre sí.

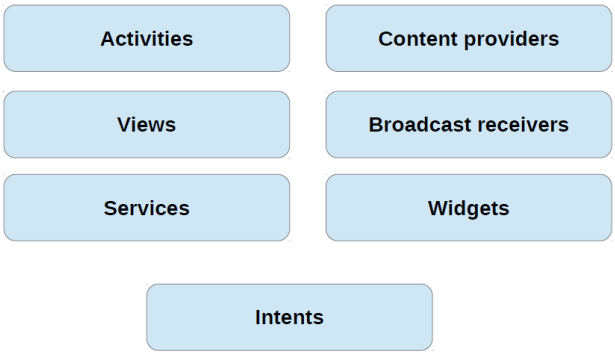


Ilustración 20. Componentes de una aplicación Android.

En otros entornos como Java o .NET estamos acostumbrados a manejar conceptos como ventana, control, eventos o servicios como los elementos

básicos para construir una aplicación. En Android vamos a disponer de esos mismos elementos pero con un pequeño cambio en la terminología y el enfoque.

Veamos cuáles son los componentes fundamentales de una aplicación Android.

4.4.1. Actividades (Activities).

Representan el componente principal de la interfaz gráfica de una aplicación Android. Serían como una ventana en cualquier otro tipo de lenguaje visual. Son implementadas por la clase *Activity*. Hablaremos de ellas con más detalle más adelante.

4.4.2. Vistas (Views).

Componentes básicos con los que se construye la interfaz gráfica de la aplicación. Serían los controles en Java o .NET. Android incorpora una gran cantidad de controles básicos, aunque podemos extender la funcionalidad de estos o crearlos personalizados:

- Cuadros de texto.
- Botones.
- Listas desplegables.
- Imágenes …

Son implementadas por la clase *View*.

4.4.3. Servicios (Services).

Componentes sin interfaz gráfica, se ejecutan en segundo plano, al igual que los de otro sistema operativo.

Pueden realizar cualquier tipo de acciones como actualizar datos, lanzar notificaciones, o mostrar elementos visuales (activities), si se necesita en algún momento la intervención del usuario.

Los implementa la clase *Service*. También hablaremos de ellos en líneas posteriores.

4.4.4. Proveedores de contenido (Content providers).

Un proveedor de contenido es el mecanismo que utiliza Android para compartir datos entre aplicaciones. Podemos compartir datos de nuestras aplicaciones sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. Nuestra aplicación podrá acceder a los datos de otra a través de los proveedores de contenido definidos. Una aplicación que acceda a los contactos de nuestra agenda utilizará un proveedor de contenidos, por ejemplo.

Los datos de los proveedores de contenido pueden estar guardados en bases de dato tipo SQLite, en los directorios del móvil, o en cualquier otro tipo de formato. Para implementarlos se utiliza la clase ContentProvider.

4.4.5. Receptores de avisos (Broadcast receivers).

Componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema, como por ejemplo “Batería baja”, “SMS recibido”, “tarjeta SD insertada”,… o por otras aplicaciones. Cualquier aplicación puede generar mensajes (intents) broadcast, es decir, que no van dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlos. Los implementa la clase *BroadcastReceiver*.

4.4.6. Widgets.

Elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) y recibir actualizaciones periódicas. Para implementarlos usaremos la clase *AppWidgetProvider*.

4.4.7. Intenciones (Intents).

Elemento básico de comunicación entre los distintos componentes Android descritos. En definitiva, mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones.

Cada *intent r*epresenta la voluntad del componente de llevar a cabo una tarea determinada, por ejemplo enviar un correo, realizar una llamada telefónica, mostrar una actividad en pantalla, etc…

Los intents se modelan mediante la clase *Intent*, y se clasifican en dos tipos:

- **Explícitos.** Son aquellos que especifican explícitamente qué componente debe llevar a cabo la tarea. Por ejemplo lanzar un servicio o mostrar una actividad determinados.
- **Implícitos.** Son aquellos que especifican la tarea pero no qué componente debe llevarla a cabo. En estos casos el sistema decide cual es el componente más adecuado haciendo uso de un mecanismo conocido como *Intent Filters*. Algunos ejemplos son enviar un correo o mostrar una URL.

4.5. Actividades en Android.

Como ya sabemos, las actividades son uno de los tipos de componentes de Android y se encargan de mostrar la interfaz en la pantalla y de recoger las interacciones con el usuario.

Para crear una actividad debemos implementar una clase Java que derive de la clase *Activity* y declararla en el manifiesto mediante el elemento *<activity>*.

Típicamente cada actividad encapsula una tarea sencilla y muy concreta, por ejemplo visualizar un mapa, mostrar una lista, etc… Las aplicaciones pueden combinar varias actividades para llevar a cabo tareas más complejas.

Las actividades se combinan usando una pila de actividades, que en la jerga de Android se conoce como *Task*. La actividad de la base de la pila se denomina actividad principal y es la primera que se muestra cuando la aplicación es lanzada.

Las demás actividades se van apilando y desapilando del Task según van siendo necesarias y la aplicación finaliza cuando se desapila la actividad principal.

4.5.1. Estados y ciclo de vida de una actividad.

Una actividad se puede considerar como una aplicación que a su vez está dentro de la aplicación que estamos desarrollando. Éstas representan el proceso en ejecución que el usuario del dispositivo ve en un momento dado. Así que una aplicación sólo puede estar ejecutando una actividad en un momento concreto.

Como consecuencia de esto las actividades podrán pasar por una serie de estados que a su vez conformarán lo que se conoce como el ciclo de vida de la misma. Es decir, el conjunto de todas las posibles transiciones entre los diferentes estados en los que ésta puede existir.

Desde el momento en que se crea una actividad hasta que deja de ser necesaria y se destruye, ésta sólo puede estar en alguno de los siguientes estados:

- **Activa:** es el estado en que se encuentra la actividad situada en la cima del Task, y que por tanto es visible y tiene el foco de interacción con el usuario.
- **Pausada:** la actividad es visible pero no tiene el foco de interacción con el usuario, ya que hay otra actividad «encima» de ella en el Task. El sistema puede destruir estas actividades en condiciones extremas de escasez de memoria.
- **Detenida:** La actividad no es visible y tampoco tiene el foco de interacción con el usuario. El sistema puede eliminar estas actividades cuando estime oportuno.

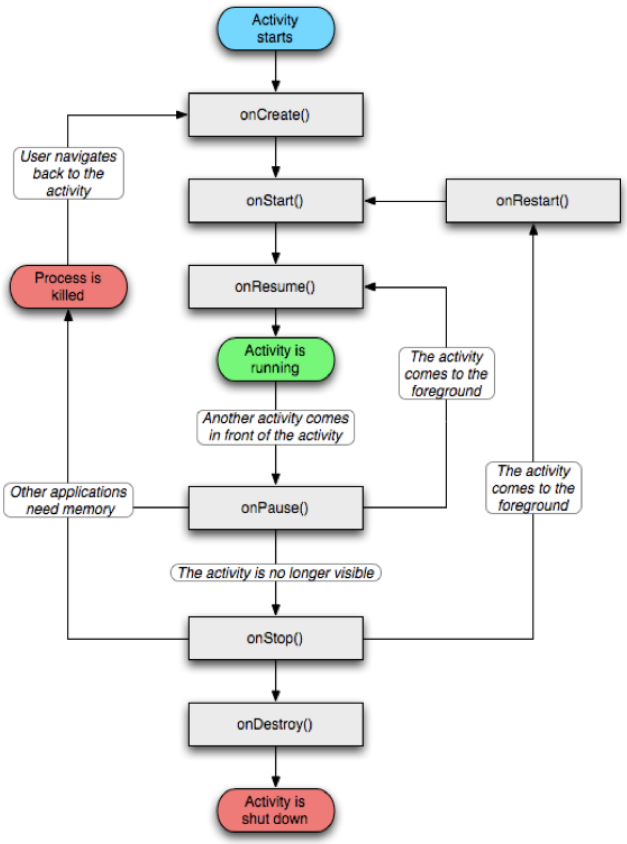


Ilustración 21. Ciclo de vida de una actividad.

En la ilustración anterior podemos ver el ciclo de vida completo de una actividad donde observamos los distintos estados por los que ésta puede pasar y a su vez los métodos que implementan dichas transiciones.

- **onCreate():** se llama a éste método cuando se crea la actividad por primera vez. Aquí es donde el programador tiene que configurar todas las vistas de la interfaz, valores de los objetos y variables que vaya a emplear.
- **onRestart():** como vemos en el gráfico este método se ejecuta justo antes de que una actividad comience de nuevo y después de haber sido detenida.
- **onStart():** se ejecuta antes de que la actividad sea visible al usuario.
- **onResume():** la actividad es visible al usuario pudiendo éste interactuar con la interfaz de la misma.
- **onPause():** se ejecuta cuando otra actividad pasa al primer plano o se va a cerrar la actual.
- **onStop():** cuando una actividad deja de ser visible al usuario dejando de estar en primer plano o antes de que vaya a ser destruida.
- **onDestroy():** método que se ejecuta cuando la actividad está siendo destruida. Después de eso ésta desaparecerá del Task. Una actividad puede ser destruida cuando el usuario sale de ésta o por el sistema en caso de necesidad de recursos y si no se encuentra en primer plano.

4.5.2. El contexto de una actividad.

En una aplicación normal de Android, por lo general existen dos tipos de contexto (clase *Context*), el de la actividad y el de la propia aplicación. Por lo general, el primer tipo es el que se pasa como parámetro a las clases y métodos que requieren un contexto.

En Android, un contexto se utiliza para muchas operaciones, pero sobre todo para cargar y acceder a los recursos. Por eso, por ejemplo todos los widgets reciben un parámetro de contexto en su constructor.

El contexto es una forma de referenciar a la actividad en la que queremos que se ejecute un proceso concreto o a la actividad misma, para desde otras clases acceder a componentes y métodos de la misma. Al contexto de una actividad puede accederse con el método *getContext()* si la clase lo implementa o también suele poder recogerse con la llamada *<clase>.this*.

4.5.3. Fragmentos.

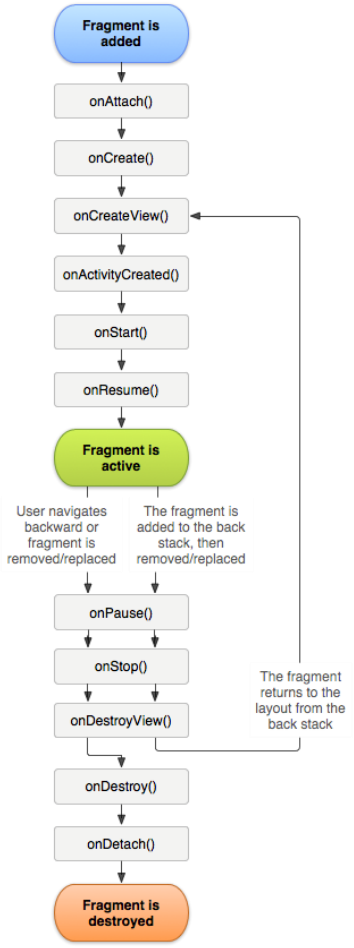


Ilustración 22. Ciclo de vida de un fragmento.

Los fragmentos se introducen a partir de la versión 3.0 de Android (API level 11). Cuando empezaron a aparecer dispositivos de gran tamaño tipo Tablet, el equipo de Android tuvo que solucionar el problema de la adaptación de la interfaz gráfica de las aplicaciones a ese nuevo tipo de pantallas. Una interfaz de usuario diseñada para un teléfono móvil no se adaptaba fácilmente a una pantalla mayor de 4 o 5 pulgadas. La solución a esto vino en forma de un nuevo tipo de componente llamado *Fragment*.

Un *Fragment* no puede considerarse ni un *control* ni un *contenedor*, aunque se parecería más a lo segundo. Un *Fragment* podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades. Esto, aunque en principio puede parecer algo trivial, nos va a permitir poder dividir nuestra interfaz en varias porciones de forma que podamos diseñar diversas configuraciones de pantalla, dependiendo de su tamaño y orientación, sin tener que duplicar código en ningún momento, sino tan sólo utilizando o no los distintos fragmentos para cada una de las posibles configuraciones.

Un fragmento siempre debe estar integrado en una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad principal.

Para insertar un fragmento en una actividad se puede declarar el fragmento en el archivo de diseño de la actividad (layout), como elemento *<fragment>*, o desde el código de la aplicación mediante la adición a un *ViewGroup* (los veremos a continuación) existente. Sin embargo, un fragmento no está obligado a ser parte de la disposición de la actividad, también se puede usar un fragmento sin su propia interfaz de usuario como trabajador invisible para la actividad.

En nuestra aplicación como veremos más adelante, usaremos un fragmento para dibujar nuestro mapa.

4.6. Interfaces gráficas en Android.

La interfaz de usuario es la parte de nuestra aplicación que se muestra en pantalla y que se encarga de presentar información y recibir feedback del usuario.

En Android, las actividades son los únicos componentes que pueden tener una interfaz de usuario asociada, a la que también se conoce como layout.

Dichos layouts podemos definirlos mediante código Java, o más comúnmente mediante archivos XML externos. Estos últimos tienen la ventaja de reforzar la separación entre la vista y la lógica de nuestras aplicaciones.

En Android cualquier elemento que tiene la capacidad de ser presentado en pantalla se conoce como vista y deriva de la clase *View*.

Para poder crear interfaces más ricas y complejas las vistas pueden organizarse jerárquicamente en árboles, de modo que algunas vistas pueden contener varias vistas hijas, y estas a su vez otras vistas hijas, etc…

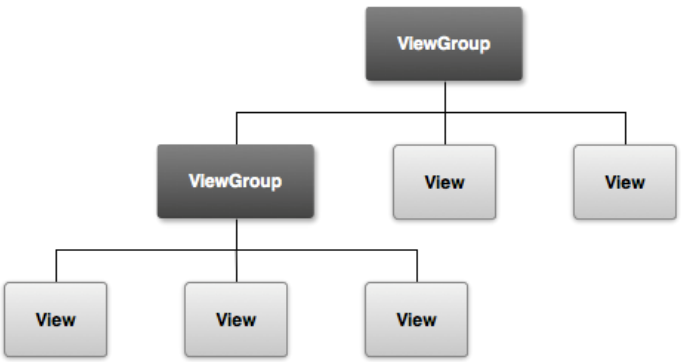


Ilustración 23. Clases de las vistas en Android.

Las vistas que pueden contener hijos son especiales y todas derivan de la clase *ViewGroup*, que a su vez deriva de *View*. Las clases derivadas de *ViewGroup* también se conocen como contenedores o layouts.

Aquellas vistas que derivan de *View* pero no de *ViewGroup* no pueden contener vistas hijas y también se conocen como controles o widgets.

Como los contenedores también derivan de *View*, a su vez pueden ser contenidos en otros contenedores. El resultado de este modelo es una jerarquía en forma de árbol

con un único nodo raíz, que será el que reciba la actividad que se encargue de presentar nuestra interfaz.

4.6.1. Widgets.

A pesar de la gran variedad de widgets predefinidos en Android, en ocasiones se necesita una funcionalidad o un aspecto visual específicos que no pueden lograrse utilizando los controles preexistentes. En tales casos, podemos crear nuestros propios controles personalizados utilizando tres técnicas básicas:

- **Widget compuesto:** utilizaremos una combinación de widgets y layouts ya existentes para crear un nuevo widget que se comportará como una unidad.
- **Widget refinado:** crearemos una nueva clase derivando de un widget ya existente e implementaremos el nuevo comportamiento o aspecto deseados. Su ventaja es que podemos reutilizar las funcionalidades preexistentes en el widget que nos sirve de base.
- **Widget personalizado:** crearemos una nueva clase derivando de *View* e implementaremos el comportamiento y el aspecto deseados partiendo desde cero. Este método es el más complejo pero también el más flexible.

4.6.2. Layouts.

En Android las clases derivadas de la clase *ViewGroup* se conocen como contenedores o layouts. Los layouts se utilizan para contener vistas hijas, ya sean widgets u otros layouts, y dotan a estos de una organización espacial específica. Existen varios tipos de layouts ya predefinidos en Android y cada uno se especializa en una jerarquía espacial concreta:

- **LinearLayout:** este contenedor presenta a sus hijos unos detrás de otros, en sentido horizontal o vertical.
- **FrameLayout:** es un layout que permite apilar a sus hijos unos delante de otros, en capas de profundidad.
- **TableLayout:** este contenedor presenta a sus hijos organizados en forma de rejilla con un número de filas y columnas configurable.
- **RelativeLayout:** es un layout que permite posicionar a sus hijos de forma relativa al layout en sí o a la posición de otros hijos del mismo layout.

4.6.3. AdapterViews y Adapters.

Existe un tipo especial de contenedores que descienden de la clase *AdapterView*, que a su vez deriva de *ViewGroup*. Su característica principal es que obtienen sus vistas hijas usando unos objetos llamados adaptadores. Android proporciona de serie varios AdapterViews listos para utilizar en nuestras interfaces. Entre ellos los más importantes son:

- **ListView**, que implementa una lista vertical de elementos con scroll.
- **GridView**, que implementa una cuadrícula de elementos con scroll.
- **Spinner**, que implementa una lista desplegable de elementos.

Un adaptador es un objeto que implementa la interfaz *Adapter*, por lo que permite acceder a un conjunto de datos asociados a él. Android implementa de serie muchos tipos de adaptadores en función de la procedencia de sus datos, pero todos ellos operan con la misma interfaz. Esta abstracción permite acceder a los datos de manera unificada independientemente de donde estén almacenados, por ejemplo en una base de datos local o en un servidor remoto. Los adaptadores más importantes que están predefinidos en Android son:

- **ArrayAdapter**, que permite acceder a un array local de elementos.
- **SimpleAdapter**, que permite acceder a una lista local de elementos y representar cada uno en una vista personalizada.
- **SimpleCursorAdapter**, que permite acceder a elementos almacenados en una base de datos local.

Ambos tipos de objetos, AdapterViews y Adapters, interactúan de la siguiente forma:

1. Se le asigna un Adapter al AdapterView usando su método setAdapter.
2. El AdapterView interroga al Adapter sobre el número de elementos que contiene.
3. El AdapterView va solicitando al Adapter uno a uno los elementos que necesita representar en pantalla.
4. Para cada elemento solicitado el Adapter crea una vista que contiene los datos de dicho elemento y se la devuelve al AdapterView.

Mediante este sistema es posible visualizar colecciones de datos muy extensas utilizando sólo los recursos imprescindibles. Es decir, se solicitan y se representan sólo los datos que están visibles en pantalla en cada momento.

4.6.4. Menús y ActionBar.

Un menú es un elemento de la interfaz cuya función es presentar al usuario una lista de acciones disponibles. En Android existen tres tipos de menús:

- **Menús de opciones:** son menús asociados a una actividad concreta y permiten ejecutar las tareas disponibles dentro de la misma. Se muestran cuando el usuario pulsa el botón Menú del dispositivo.
- **Menús contextuales:** son aquellos asociados a un AdapterView concreto y permiten operar sobre un elemento específico de su contenido. Se muestran cuando el usuario hace una pulsación larga sobre el AdapterView.
- **Menús popup:** son menús asociados a una vista concreta y permiten llevar a cabo tareas relacionadas con ella. Se muestran programáticamente, normalmente como respuesta a una pulsación sobre la vista asociada. Están disponibles a partir de la versión 3.0 de Android.

En todos los tipos de menús de Android cualquiera de sus opciones puede conducir a un submenú, pero las opciones de este último no pueden desplegar nuevos submenús. Es decir, los menús están limitados a un máximo de dos niveles.

Las opciones de los menús se componen de un título, un icono, una caja de selección, o un botón de selección en el caso de opciones agrupadas. Dependiendo del tipo de menú y del nivel dentro del mismo, la opción de menú se representa con todos o sólo algunos de sus componentes.

Todos los tipos de menús de Android pueden definirse de forma programática o mediante un archivo XML. Se recomienda esta última opción para facilitar el mantenimiento de la aplicación.

4.6.5. Toast, diálogos y notificaciones.

En Android podemos encontrar los siguientes tipos de cuadros de diálogo y notificaciones:

- **Los Toast**: también conocidos como Toast Notifications, son elementos que de forma muy sencilla, permiten informar al usuario sobre las operaciones que está llevando a cabo la aplicación.

Se muestran como mensajes de texto flotante que aparecen superpuestos al resto de elementos de la interfaz y transcurrido un breve periodo de tiempo desaparecen automáticamente.

Los Toast Notifications sólo permiten mostrar texto y no pueden recibir ningún tipo de respuesta por parte del usuario. Por ello suelen emplearse para mostrar mensajes de confirmación de tareas completadas, satisfactoriamente o no, pero se desaconsejan para informar sobre errores críticos.

Los Toast pueden utilizarse a partir de la versión 2.0 de Android.

- **Los diálogos:** son ventanas flotantes que permiten mostrar interfaces totalmente personalizadas al usuario e interactuar con él. Se utilizan por tanto en aquellas ocasiones en las que la información a presentar es demasiado compleja para un Toast o bien se requiere una respuesta para continuar.

  Los diálogos pueden ser modales o no modales, o en la terminología de Android cancelables o no cancelables. Los diálogos modales (no cancelables) obligan al usuario dar una respuesta para poder seguir trabajando con la aplicación. Se utilizan cuando es importante obtener una respuesta explícita, como en el caso de los diálogos de confirmación de guardado de documentos.

  A partir de la versión 3.0 de Android las funciones originales para gestionar diálogos quedan obsoletas y se recomienda utilizar en su lugar la nueva clase *DialogFragment*. Gracias a ella es posible presentar diálogos flotantes, así como diálogos integrados en un panel de la interfaz principal.
- **Notificaciones:** en Android a las notificaciones también se las conoce como Status Bar Notifications, y consisten en un mecanismo mediante el cual se puede informar al usuario de forma no intrusiva de alguna circunstancia relacionada con la aplicación.

  Las notificaciones no son intrusivas porque se muestran en la barra de estado. Por ello son ideales para que un proceso en segundo plano, generalmente un servicio, pueda comunicarse con el usuario sin interferir con la aplicación que en ese momento esté en primer plano.

  Además las notificaciones pueden interactuar con el usuario y enviar una respuesta de éste a la aplicación que las originó. Para ello cada notificación contiene una lista de posibles acciones, que a su vez están asociadas a un Intent. Cuando el usuario selecciona una de las acciones se lanza el Intent correspondiente.

4.7. Soporte para distintos tamaños de pantalla y dispositivos.

Dado que existen gran variedad de dispositivos Android. En muchas ocasiones, es necesario poder distinguir entre los mismos. Si queremos desarrollar una aplicación que sea compatible con varios dispositivos no podemos pasar por alto estos aspectos.

Para referirnos formalmente al tamaño de una pantalla debemos distinguir entre varios conceptos:

- **Tamaño:** es la medida física de la pantalla, normalmente expresada en base a las pulgadas de su diagonal.
- **Densidad:** es la cantidad de píxeles que tiene la pantalla en una pulgada.
- **Orientación:** se valora desde el punto de vista del usuario. Si el ancho de la pantalla es mayor que su alto se dice que es una pantalla *landscape* y en caso contrario que es una pantalla *portrait*.
- **Resolución:** es el número total de píxeles de ancho y alto de la pantalla. Android define además una unidad de medida llamada píxel independiente de la densidad, o *dpi*, que permite especificar medidas que serán iguales en cualquier dispositivo, independientemente de la densidad de su pantalla. Un *dpi* equivale a un pixel en una pantalla de 160 píxeles por pulgada.

4.7.1. Grupos de pantallas soportadas.

Android clasifica las pantallas que pueden incorporar los dispositivos atendiendo a dos criterios:

- **Tamaño:** en base a este criterio Android distingue cuatro grupos de pantallas:
  - *Small (≥ 426dp x 320dp).*
  - *Normal (≥ 470dp x 320dp).*
  - *Large (≥ 640dp x 480dp).*
  - *Xlarge (≥ 960dp x 720dp).*
- **Densidad:** en base a este criterio, actualmente Android distingue otros cinco grupos:
  - *Low (< 160 dpi).*
  - *Medium (~ 160 dpi).*
  - *High (~ 240 dpi)*
  - *Xhigh (~ 320 dpi).*
  - *XXhigh (~ 480 dpi).*

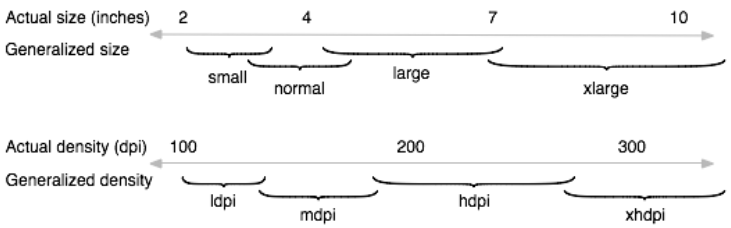


Ilustración 24. Clasificación de tamaños y densidades de pantalla en Android.

Una pantalla puede presentar cualquier combinación de un valor del primer grupo y otro del segundo, es decir mayor tamaño no implica mayor densidad y viceversa.

4.7.2. Interfaces independientes de la densidad.

Es importante que los elementos de la interfaz de las aplicaciones tengan siempre un tamaño físico adecuado independientemente de la densidad de pantalla del dispositivo donde se ejecuten.

Si se especifica el tamaño de estos elementos en píxeles se corre el riesgo de que al cambiar a otra pantalla de distinta densidad la aplicación deje de ser usable. Por ejemplo, un botón de 160 píxeles de ancho medirá 1 pulgada en una pantalla de 160 ppi (pixels per inch), mientras que medirá sólo ½ pulgada en una pantalla de 320 ppi. Esta diferencia de tamaño puede hacer que el usuario encuentre difícil o incluso imposible pulsar dicho botón.

Para solucionar este problemática en Android se siguen dos estrategias:

- Especificar todas las dimensiones, bien en *dpi* o bien usando *wrap_content*.
- Android escala automáticamente las imágenes para ajustarse a la densidad del dispositivo. Sin embargo, esto puede hacer que se muestren difuminadas. Para ello podemos suministrar una versión distinta de cada imagen para cada grupo de densidad de pantalla.

4.7.3. Cualificadores de recursos y su aplicación a interfaces.

Todos las subcarpetas de la carpeta res de un proyecto Android están sujetas al control del gestor de recursos. Esto quiere decir que en lugar de leer sus contenidos directamente, las aplicaciones deben solicitárselos al gestor de recursos.

Cada carpeta de recursos puede especificar una serie de atributos, llamados cualificadores, para informar al gestor de recursos para que tipo de dispositivo están optimizados los recursos que contiene. Así podemos tener varias versiones de un mismo recurso cualificadas según el tamaño de pantalla, la densidad de pantalla, el idioma del dispositivo, la versión de Android que ejecuta el dispositivo, etc…

En tiempo de ejecución el gestor de recursos evaluará cual es la versión más adecuada para el dispositivo y será dicha versión la que cargue cuando la aplicación se lo solicite.

Si no se encuentra una versión específica para el dispositivo el gestor carga la versión por defecto, que es la que contiene la carpeta de recursos que no especifica ningún cualificador. Por esta razón siempre se recomienda suministrar una versión por defecto de todos los recursos.

Uno de los usos más comunes de la cualificación de recursos es adaptar la apariencia de la interfaz de la aplicación a las características de la pantalla del dispositivo en el que se está ejecutando.

Para ello se sigue una estrategia basada en tres principios:

1. **Cualificar los layouts según tamaño de pantalla.** De esta forma el usuario percibe que la interfaz ha sido diseñada exclusivamente para su dispositivo, y no sólo escalada para ajustarse a él. Por ejemplo, en dispositivos con pantallas pequeñas se suele prescindir de elementos de la interfaz no esenciales, mientras que en pantallas grandes se incluyen elementos adicionales para dar uso al espacio extra disponible.
2. **Cualificar los drawables por densidad de pantalla**. Proporcionando versiones de los drawables para cada tipo de densidad se consigue que se muestren siempre con una buena definición. Se evita así que el sistema tenga que escalar los elementos gráficos en proporciones muy elevadas, con los problemas de calidad que ello conlleva.
3. **Cualificar los layouts según la orientación de la pantalla.** En las pantallas portrait los elementos de la interfaz suelen tener un desarrollo vertical. Sin embargo, al pasar a una pantalla landscape resulta más adecuado que los elementos inferiores de la interfaz se desplacen a la derecha de la pantalla.

4.7.4. Cualificación basada en el ancho de pantalla.

A partir de la versión 3.2 de Android se introduce una nueva forma de cualificar los recursos según el tamaño disponible en la pantalla.

Esta forma de cualificación es más precisa, ya que en lugar de utilizar los grupos tradicionales de tamaño (small, normal, large, xlarge), permite especificar un tamaño mínimo que debe tener la pantalla para que se utilice el recurso. Existen tres cualificadores de este tipo:

- **Ancho más pequeño:** es el ancho mínimo que debe tener la pantalla independientemente de la orientación. Es decir, este valor no cambia cuando la pantalla rota.

- **Ancho disponible:** es el ancho mínimo que debe tener la pantalla en la orientación actual. Este valor cambia cuando la pantalla rota.
- **Alto disponible:** es el alto mínimo que debe tener la pantalla en la orientación actual. Este valor cambia cuando la pantalla rota.

Todos los valores se refieren al tamaño disponible para la aplicación. Es decir, no se tienen en cuenta la status bar, la system bar y otros elementos del sistema. Sin embargo si se incluye en dicho tamaño la action bar, ya que se considera un elemento perteneciente a la aplicación.

Los tres cualificadores anteriores especifican los tamaños en dips. Se utiliza esta unidad porque para seleccionar la interfaz el criterio más adecuado es el tamaño físico disponible en la pantalla, y no los píxeles disponibles.

4.7.5. Declarar las pantallas soportadas en el manifiesto.

Después de adecuar la aplicación a los distintos tamaños y densidades de pantalla que esta soportará es importante indicar dicha información en el manifiesto.

Para ello se hace uso del elemento *<supports-screens>* que permite informar al sistema de los grupos de tamaños de pantalla para los que la aplicación está preparada. A partir de Android 3.2 se puede especificar en este elemento el ancho mínimo que la aplicación necesita, pero nunca deben mezclarse los dos métodos. Es decir no se deben especificar grupos de tamaño y ancho mínimo simultáneamente.

Android utiliza la información del elemento *<supports-screens>* para activar el modo de compatibilidad en el caso de dispositivos con pantallas mayores a las soportadas por la aplicación.

Además Google Play y otras tiendas de aplicaciones filtran las búsquedas por este criterio. De este modo los usuarios de dispositivos no soportados por la aplicación no podrán instalarla, lo que es deseable, ya que así se evitan reclamaciones y comentarios negativos.

4.8. Servicios en Android.

En muchos casos, será necesario añadir un nuevo componente a una aplicación para ejecutar algún tipo de acción que se ejecute en segundo plano, es decir, que no

requiera una interacción directa con el usuario; pero que queramos que permanezca activa aunque el usuario cambie de actividad. Este es el momento de crear un servicio.

Un servicio es un componente de aplicación que puede realizar operaciones de larga duración en segundo plano y no proporciona una interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y éste continuará funcionando en segundo plano, incluso si el usuario cambia a otra aplicación. Además, un componente puede conectarse a un servicio para interactuar con él e incluso realizar comunicación entre procesos (IPC). Por ejemplo, un servicio puede manejar las transacciones de red, reproducir música, realizar operaciones con archivos de E/S, o interactuar con un proveedor de contenidos, todo desde el background.

En Android los servicios tienen una doble función:

- La primera función permite indicar al sistema que el elemento que estamos creando ha de ejecutarse en segundo plano, normalmente durante un largo período de tiempo. Este tipo de servicios son iniciados mediante el método *startService()*, que indica al sistema que lo ejecute de forma indefinida hasta que alguien le indique lo contrario.
- La segunda función permite que nuestra aplicación se comunique con otras aplicaciones, para lo cual ofreceremos ciertas funciones que podrán ser llamada desde otras aplicaciones. Este tipo de servicios son iniciados mediante el método *bindService()*, que permite establecer una conexión con el servicio e invocar alguno de los métodos que son ofrecidos.

Cada vez que un servicio es creado por alguna de las razones anteriores, el sistema instancia el servicio y llama al método *onCreate()*. Corresponde al servicio implementar el comportamiento adecuado, habitualmente creará un hilo de ejecución (*thread*) secundario donde se realizará el trabajo.

Un servicio, como el resto de componentes de una aplicación, se ejecuta en el hilo principal del proceso de la aplicación. Por lo tanto, si el servicio necesita un uso intensivo de CPU o puede quedar bloqueado en ciertas operaciones, como uso de redes, se debe crear un hilo diferente para ejecutar estas acciones. También se puede utilizar la clase *IntentService* para lanzar un servicio en su propio hilo.

Para crear un servicio debemos implementar una clase Java que derive de la clase *Service* y declararla en el manifiesto mediante el elemento *<service>*. También podemos definir un permiso para restringir su acceso. En este caso, las aplicaciones han de declarar este permiso, con el correspondiente *<uses-permission>* en su propio manifiesto.

4.8.1. Estados y ciclo de vida de un servicio.

Al igual que sucede con las actividades los servicios en Android también tienen su propio ciclo de vida. El ciclo de vida de un servicio es más sencillo que el de una actividad pero es importante tener muy en cuenta cuando se crea y se destruye ya que éste puede permanecer corriendo en segundo plano sin que el usuario se percate de ello.

Como acabamos de explicar existen dos tipos de servicios según como hayan sido creados. Las funciones de estos servicios son diferentes, y por lo tanto, también su ciclo de vida.

Si el servicio es iniciado mediante *startService()* el sistema comenzará creándolo y llamando a su método *onCreate().* A continuación llamará a su método *onStartCommand(Intent intent, int flags, int startId)* con los argumentos proporcionados por el cliente (en versiones del API inferiores a 2.0 el método llamado era *onStart()* éste se mantiene en versiones recientes por razones de compatibilidad). El servicio continuará en ejecución hasta que sea invocado el método *stopService()* o *stopSelf()*.

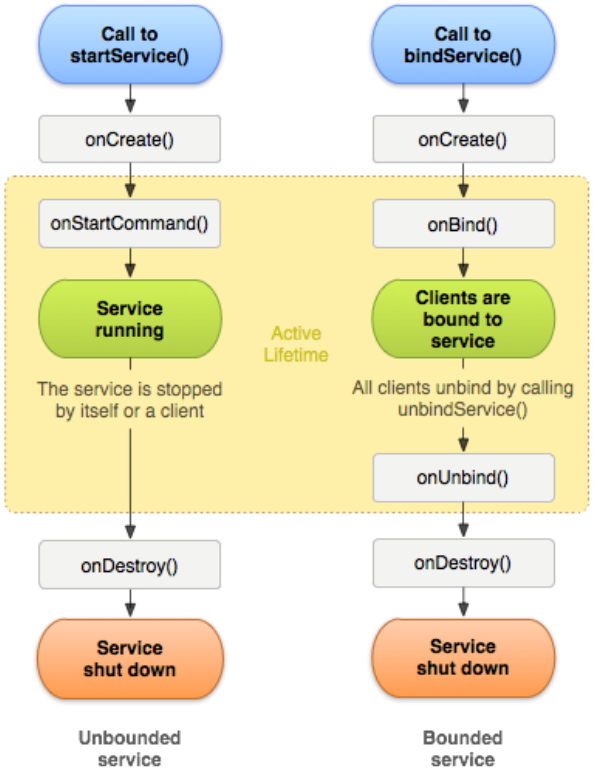


Ilustración 25. Ciclo de vida de un servicio.

Si se producen varias llamadas a *startService()* no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a *onStartCommand()*. No importa cuántas veces el servicio haya sido creado, parará con la primera invocación de *stopService()* o *stopSelf()*. Sin embargo, podemos utilizar el método stopSelf(int startId) para asegurarnos que el servicio no parará hasta que todas las llamadas hayan sido procesadas.

Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria. Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en *onStartCommand()*. Existen dos modos principales: devolveremos START_STICKY si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente. Devolveremos START_NOT_STICKY si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.

Teniendo en cuenta que los servicios pueden estar largo tiempo en ejecución, el ciclo de vida del proceso que contiene nuestro servicio es un asunto de gran importancia. Conviene aclarar que en situaciones donde el sistema necesite memoria conservar un servicio siempre será menos prioritario que la actividad visible en pantalla, aunque más prioritario que otras actividades en segundo plano. Dado que el número de actividades visibles es siempre reducido, un servicio solo será eliminado en situaciones de extrema necesidad de memoria. Por otra parte, si un cliente visible está conectado a un servicio, el servicio también será considerado como visible, siendo tan prioritario como el cliente. En el caso de un proceso que contenga varios componentes, por ejemplo una actividad y un servicio, su prioridad se obtiene como el máximo de sus componentes.

Podemos también utilizar *bindService(Intent servicio, ServiceConnection conexion, int flags)* para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el *flag* BIND_AUTO_CREATE esté activo), llamándose al método *onCreate()*, pero no se llamará a *onStartCommand()*. En su lugar se llamará al método *onBind(Intent intent)* que ha de devolver al cliente un objeto *IBinder* a través del cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de una interfaz escrita en AIDL, que permite el intercambio de objetos entre aplicaciones que corren en procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto *IBinder*.

También es posible diseñar un servicio que pueda ser arrancado de ambas formas (*startService()* y *bindService()*). Este servicio permanecerá activo si ha sido

creado desde la aplicación que lo contiene o si recibe conexiones desde otras aplicaciones.

Todo servicio terminará llamando al método *onDestroy()* cuando vaya a terminar de forma efectiva.

4.8.2 Notificaciones de uso de un servicio.

Una vez en ejecución, un servicio puede notificar al usuario de eventos mediante una notificación Toast o una notificación en la barra de estado del sistema.

Como ya sabemos una notificación Toast es un mensaje que aparece en la superficie de la ventana actual por un momento y luego desaparece, mientras que una notificación de la barra de estado proporciona un icono en la barra de estado con un mensaje que el usuario puede seleccionar con el fin de realizar una acción (por ejemplo, como iniciar una actividad).

Por lo general, una notificación de la barra de estado es la mejor técnica cuando un proceso en el background se ha completado (como una descarga completa de un archivo) y el usuario puede actuar sobre ella. Cuando el usuario selecciona la notificación en el panel correspondiente, la notificación puede iniciar una actividad (por ejemplo, para ver el archivo descargado).

4.9. Localización en Android.

La localización geográfica en Android es uno de esos servicios que, a pesar de requerir poco código para ponerlos en marcha, no son para nada intuitivos ni fáciles de llegar a comprender por completo. Y esto no es debido al diseño de la plataforma Android en sí, sino a la propia naturaleza de este tipo de servicios.

La plataforma Android dispone de un interesante sistema de posicionamiento que combina varias tecnologías:

- **Sistema de localización global basado en GPS**. Este sistema sólo funciona si disponemos de visibilidad directa de los satélites.
- **Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.** Funciona en el interior de los edificios.

El GPS presenta un inconveniente; solo funciona cuando tenemos visión directa de los satélites. Para solventar este problema, Android combina esta información con la recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.

Estos servicios se encuentran totalmente integrados en el sistema y son usados por gran variedad de aplicaciones. Para utilizar estas capacidades en una aplicación usaremos las clases del paquete android.location y el API de Google Maps para Android.

4.9.1. Servicios de localización en Android.

Como ya hemos dicho, Android nos ofrece el código para hacer uso de los servicios de localización a través de las clases contenidas en el paquete android.location. Veremos a continuación las clases e interfaces más importantes de este paquete:

- **Location.** Esta clase representa una posición en el mapa que ha sido recogida por alguno de los proveedores de ubicación que ofrece nuestro dispositivo (normalmente GPS o redes Wi-Fi). Para acceder a la latitud y la longitud usaremos sus métodos *getLatitude()* y *getLongitude()*. También podemos acceder a otros datos como la altura, el instante en que se registró dicha posición, el proveedor de la misma, etc…
- **LocationManager**. El componente central del marco de ubicación es el servicio del sistema *LocationManager*, que proporciona APIs para determinar la ubicación y el rumbo del dispositivo subyacente (si está disponible). Al igual que con otros servicios del sistema, no se crean instancias de un *LocationManager* directamente. Más bien, se solicita una instancia del sistema llamando a *getSystemService(Context.LOCATION_SERVICE*). Éste método devuelve un identificador para una nueva instancia de *LocationManager*. Una vez que su aplicación tiene un *LocationManager*, ésta es capaz de hacer tres cosas:
  - Consultar la lista de todos los *LocationProviders* de la última ubicación conocida del usuario.
  - Registrar/anular el registro de las actualizaciones periódicas de la ubicación actual del usuario a través de un proveedor de ubicación (especificado ya sea por criterios o nombre).
  - Registrar/anular el registro de una intención (intent) determinada si el dispositivo se encuentra próximo a una zona dada (especificada por el radio en metros).

  En Android no existe ningún método del tipo *"obtenerPosiciónActual()"*. Obtener la posición a través de un dispositivo de localización como por ejemplo el GPS no es una tarea inmediata, sino que puede requerir de un cierto tiempo

de procesamiento y de espera, por lo que no tendría sentido proporcionar un método de ese tipo.

Si buscamos entre los métodos disponibles en la clase LocationManager, lo más parecido que encontramos es un método llamado *getLastKnownLocation(String provider)*, que como se puede suponer por su nombre, nos devuelve la última posición conocida del dispositivo devuelta por un provider determinado. Es importante entender esto: este método NO devuelve la posición actual, este método NO solicita una nueva posición al proveedor de localización, este método se limita a devolver la última posición que se obtuvo a través del proveedor que se le indique como parámetro. Y esta posición se pudo obtener hace pocos segundos, hace días, hace meses, o incluso nunca (si el dispositivo ha estado apagado, si nunca se ha activado el GPS,…). Por tanto, hay que tener cuidado cuando se haga uso de la posición devuelta por el método *getLastKnownLocation()*.

Entonces, ¿de qué forma podemos obtener la posición real actualizada? Pues la forma correcta de proceder va a consistir en algo así como activar el proveedor de localización y suscribirnos a sus notificaciones de cambio de posición. O dicho de otra forma, vamos a suscribirnos al evento que se lanza cada vez que un proveedor recibe nuevos datos sobre la localización actual. Y para ello, vamos a darle previamente unas indicaciones sobre cada cuanto tiempo o cada cuanta distancia recorrida necesitaríamos tener una actualización de la posición.

Todo esto lo vamos a realizar mediante una llamada al método *requestLocationUpdates()*, al que deberemos pasar 4 parámetros distintos:

- Nombre del proveedor de localización al que nos queremos suscribir.
- Tiempo mínimo entre actualizaciones, en milisegundos.
- Distancia mínima entre actualizaciones, en metros.
- Instancia de un objeto *LocationListener*, que tendremos que implementar previamente para definir las acciones a realizar al recibir cada nueva actualización de la posición.

Tanto el tiempo como la distancia entre actualizaciones pueden pasarse con valor 0, lo que indicaría que ese criterio no se tendrá en cuenta a la hora de decidir la frecuencia de actualizaciones. Si ambos valores van a cero, las actualizaciones de posición se recibirán tan pronto y tan frecuentemente como estén disponibles. Además, como ya hemos indicado, es importante comprender que tanto el tiempo como la distancia especificadas se entenderán como simples indicaciones o “pistas” para el proveedor (al menos para versiones no recientes de Android), por lo que puede que no se cumplan de forma estricta.

Para mejorar la adquisición de nuestra ubicación, se pueden utilizar estrategias que podemos encontrar en la web de desarrolladores de Android: http://developer.android.com/guide/topics/location/strategies.html.

Por último si queremos dejar de recibir actualizaciones sobre la ubicación de nuestro dispositivo haremos una llamada al método *removeUpdates(LocationListener).*

- **LocationListener**. En cuanto al listener, éste será del tipo *LocationListener* y contendrá una serie de métodos asociados a los distintos eventos que podemos recibir del proveedor:
  - onLocationChanged(location). Lanzado cada vez que se recibe una actualización de la posición.
  - onProviderDisabled(provider). Lanzado cuando el proveedor se deshabilita.
  - onProviderEnabled(provider). Lanzado cuando el proveedor se habilita.
  - onStatusChanged(provider, status, extras). Lanzado cada vez que el proveedor cambia su estado, que puede variar entre OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE, AVAILABLE.

  Por nuestra parte, tendremos que implementar cada uno de estos métodos para responder a los eventos del proveedor, sobre todo al más interesante, *onLocationChanged()*, que se ejecutará cada vez que se recibe una nueva localización desde el proveedor.

4.9.2. Permisos de acceso a la ubicación del dispositivo.

Para que una aplicación pueda acceder a los servicios descritos anteriormente es necesario solicitar permiso en el manifiesto de la misma. Existen dos permisos dedicados para tal efecto:

- **android.permission.ACCESS_COARSE_LOCATION.** Permite acceso al proveedor de ubicación a través de Internet (NETWORK_PROVIDER).
- **android.permission.ACCESS_FINE_LOCATION.** Permite acceso al proveedor de ubicación a través del GPS (GPS_PROVIDER) y también incluye al anterior.

A pesar de que el permiso anterior incluye el derecho de acceso a ambos proveedores de ubicación normalmente y para evitar conflictos se suelen solicitar los dos permisos. Por ejemplo, la documentación de la API de Google Maps, que veremos con detalle a continuación recomienda su uso.

4.10. Google Maps Android API v. 2.

Google Maps nos proporciona un servicio de cartografía *online* que podremos utilizar en nuestras aplicaciones Android. Veremos las claves necesarias para utilizarlo.

Estudiaremos la versión 2 del API que incorpora interesantes ventajas respecto a la versión anterior. Entre estas ventajas destacan:

- Integración con los Servicios de Google Play (*Google Play Services*) y la Consola de APIs.
- Utilización a través de un nuevo tipo específico de *fragment (MapFragment)*, una mejora muy esperada por muchos.
- Utilización de *mapas vectoriales*, lo que repercute en una mayor velocidad de carga y una mayor eficiencia en cuanto a uso de ancho de banda.
- Mejoras en el sistema de caché, lo que reducirá en gran medida las famosas áreas en blanco que tardan en cargar.
- Los mapas son ahora 3D, es decir, podremos mover nuestro punto de vista de forma que lo veamos en perspectiva.

La principal diferencia de esta versión respecto de la anterior es el componente que utilizaremos para la inclusión de mapas en nuestra aplicación. En la anterior versión de la API, para incluir un mapa en la aplicación debíamos utilizar un control de tipo *MapView*, que además requería que su actividad contenedora fuera del tipo *MapActivity*. Con la nueva API nos olvidaremos de estos dos componentes y pasaremos a tener sólo uno, un nuevo tipo específico de *fragment* llamado *MapFragment*. Esto nos permitirá entre otras cosas añadir uno (o varios, esto también es una novedad) mapas a cualquier actividad, sea del tipo que sea, y contando por supuesto con todas las ventajas del uso de fragmentos. Es importante tener en cuenta que como el nuevo control de mapas se basa en fragmentos, si queremos mantener la compatibilidad con versiones de Android anteriores a la 3.0 tendremos que utilizar la librería de soporte *android-support*.

Como inconveniente resaltar que la nueva versión sólo funciona en un dispositivo con Google Play instalado.

Conviene también destacar que a diferencia de Android, Google Maps no es un software libre, por lo que está limitado a una serie de condiciones de servicio. Podemos usarlo de forma gratuita siempre que nuestra aplicación no solicite más de 15.000 codificaciones geográficas al día. Podemos incluir propaganda en los mapas o incluso

podemos usarlo en aplicaciones móviles de pago (para sitios web de pago es diferente). Una información más completa de este API la encontramos en:

https://developers.google.com/maps/documentation/android

Para acabar si utilizamos la API de Google Maps para Android en nuestra aplicación, se debe incluir el texto de atribución de Google como parte de la sección de "Avisos legales" en la aplicación. Se recomienda incluir los avisos legales como una opción de menú independiente o como parte de un elemento del menú "Acerca de".

El texto atribución está disponible al hacer una llamada a *GooglePlayServicesUtil.getOpenSourceSoftwareLicenseInfo*.

4.10.1. Trabajar con la API de Google Maps.

Para trabajar con esta nueva versión de la API debemos seguir una serie de pasos previos para configurar nuestro entorno adecuadamente y así poder desarrollar aplicaciones haciendo uso de la misma. Los pasos que deberemos seguir serán los siguientes:

1. **Instalación de los servicios de Google Play (Google Play Services).**

   En primer lugar, dado que la API v2 se proporciona como parte del SDK de *Google Play Services*, será necesario incorporar previamente a nuestro entorno de desarrollo dicho paquete. Haremos esto accediendo desde Eclipse al Android SDK Manager y descargando del apartado de extras el paquete llamado “Google Play Services”.

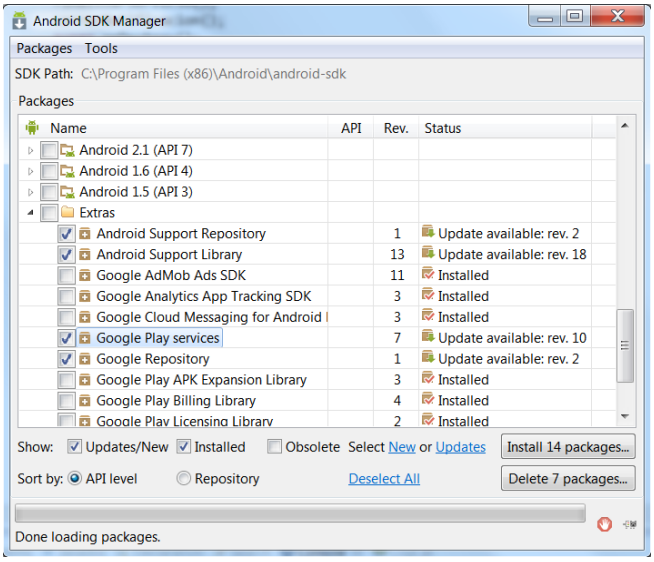


Ilustración 26. Instalación de Google Play Services.

Tras pulsar el botón de *Install* y aceptar la licencia correspondiente el paquete quedará instalado en nuestro sistema, concretamente en la ruta: *<carpeta-sdk-android>/extras/google/google_play_services/*.

2. **Obtención de una API Key.**

Para poder utilizar este servicio de Google, al igual que como ocurre cuando se utiliza desde una página web, va a ser necesario registrar la aplicación que lo utilizará. Tras registrar la aplicación se nos entregará una clave que tendremos que indicar en la aplicación.

Realmente vamos a necesitar dos claves diferentes, una durante el proceso de desarrollo y otra para la aplicación final. La razón es que se genera una clave diferente en función del certificado digital con la que se firma la aplicación. En la fase de desarrollo las aplicaciones también han de ser firmadas digitalmente, pero en este caso el SDK utiliza un certificado especial utilizado sólo en la fase de desarrollo. En caso de querer distribuir una aplicación, una vez terminada tendrá que ser firmada con un certificado digital propio. Será entonces necesario reemplazar la clave de *Google Maps*, por otra, esta última asociada a nuestro certificado digital. Veamos cómo obtener la clave de *Google Maps* para el certificado de depuración:

La nueva API de mapas de Android se ha integrado por fin en la Consola de APIs de Google, por lo que el primer paso será acceder a ella. Una vez hemos accedido, tendremos que crear un nuevo proyecto desplegando el menú superior izquierdo y seleccionando la opción “Create…”.

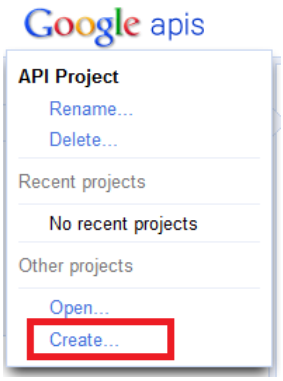


Ilustración 27. Google APIs: creación de un proyecto.

Aparecerá entonces una ventana que nos solicitará el nombre del proyecto. Introducimos algún nombre descriptivo y aceptamos sin más.

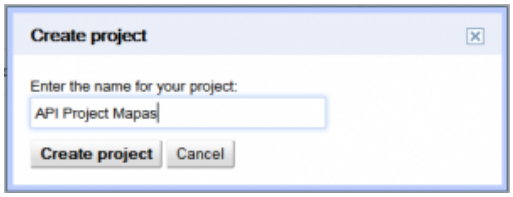


Ilustración 28. Google APIs: nombre de proyecto.

Una vez creado el proyecto, accederemos a la opción “Services” del menú izquierdo. Desde esta ventana podemos activar o desactivar cada uno de los servicios de Google que queremos utilizar. En este caso sólo activaremos el servicio llamado “Google Maps Android API v2” pulsando sobre el botón ON/OFF situado justo a su derecha.

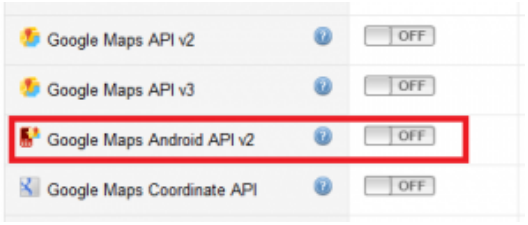


Ilustración 29. Google Maps Android API v2.

Una vez activado aparecerá una nueva opción en el menú izquierdo llamada “API Access”. Accediendo a dicha opción tendremos la posibilidad de obtener nuestra nueva API Key que nos permita utilizar el servicio de mapas desde nuestra aplicación particular.

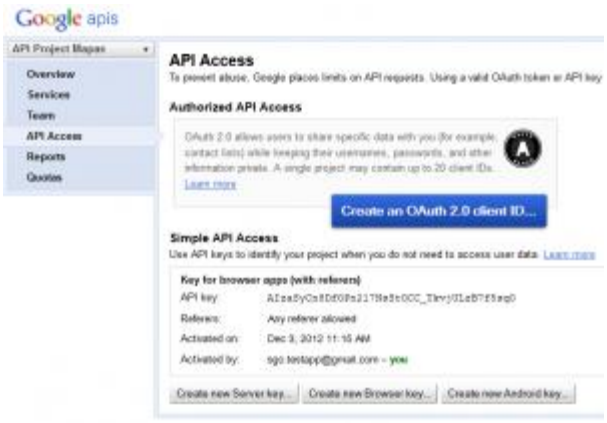


Ilustración 30. Creación de una API Key.

Para ello, pulsaremos el botón "Create new Android key…". Esto nos llevará a un cuadro de diálogo donde tendremos que introducir algunos datos identificativos de nuestra aplicación. En concreto necesitaremos dos: la huella digital (SHA1) del certificado con el que firmamos la aplicación, y el paquete java utilizado. El segundo no tiene misterio, pero el primero requiere alguna explicación.

Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Este proceso de firma es uno de los pasos que tenemos que hacer siempre antes de distribuir públicamente una aplicación. Adicionalmente, durante el desarrollo de la misma, para realizar pruebas y la depuración del código, aunque no seamos conscientes de ello también estamos firmando la aplicación con un "certificado de pruebas". Podemos saber en qué carpeta de nuestro sistema está almacenado este certificado accediendo desde Eclipse al menú Window /Preferences y accediendo a la sección Android / Build. Normalmente el certificado de pruebas está en la ruta: "C:\Users\<nombre_de_usuario>\.android\debug.keystore".

Pues bien, para obtener nuestra huella digital SHA1 deberemos acceder a dicha ruta desde la consola de comando de Windows y ejecutar los siguientes comandos:

- `C:\>cd C:\Users\<nombre_de_usuario>\.android\`
- `C:\Users\<nombre_de_usuario>\.android>"C:\Program Files\Java\jdk1.7.0_17\bin\keytool.exe" -list -v -keystore debug.keystore -alias androiddebugkey -storepass android -keypass android`

Suponiendo que nuestra instalación de Java está en la ruta "C:\Program Files\Java\jdk1.7.0_17". Si no es así sólo debemos sustituir ésta por la correcta. Esto nos deberá devolver varios datos del certificado, entre ellos la huella SHA1.

Pues bien, nos copiamos este dato y lo añadimos a la ventana de obtención de la API Key donde nos habíamos quedado antes, y a continuación **separado por un punto y coma** añadimos el paquete java que vayamos a utilizar en nuestra aplicación, que en mi caso será "us.jvalzae.pfc".

Pulsamos el botón "Create" y ya deberíamos tener nuestra API Key generada, podremos verla en la pantalla siguiente dentro del apartado "Key for Android Apps (with certificates)". Apuntaremos también este dato para utilizarlo más tarde.

Ilustración 31. API Key.

Con esto ya habríamos concluido los preparativos iniciales necesarios para utilizar el servicio de mapas de Android en nuestras propias aplicaciones.

3. **Añadir la API Key al manifiesto de nuestra aplicación**.

Tras esto lo primero que haremos será añadir al fichero AndroidManifest.xml la API Key que acabamos de generar. Para ello añadiremos al fichero, dentro de la etiqueta *<application>*, un nuevo elemento *<meta-data>* con los siguientes datos:

```
…
<application>
…
<meta-data android:name="com.google.android.maps.v2.API_KEY"
android:value="API_KEY"/>
…
</application>
```

Como valor del parámetro android:value tendremos que poner nuestra API Key recién generada.

4. **Añadir los permisos necesarios para el uso de mapas.**

Siguiendo con el manifiesto, también tendremos que incluir una serie de permisos que nos permitan hacer uso de los mapas.

En primer lugar tendremos que definir y utilizar un permiso llamado "paquete.java.permission.MAPS_RECEIVE". Este permiso los hemos definido y utilizado en nuestra aplicación en el momento de su desarrollo ya que era

necesario, pero actualmente con la nueva actualización de Google Play Services 3.1.59, es totalmente innecesario. Lo que siempre tendremos que añadir son permisos adicionales que nos permitan acceder a los servicios web de Google, a Internet, y al almacenamiento externo del dispositivo (utilizado para la caché de los mapas). Veamos:

```
…
<permission
        android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" >
    </permission>

    <uses-permission
android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
…
```

5. **Añadir el uso de las características necesarias.**

Por último, dado que la API v2 de Google Maps Android utiliza OpenGL ES versión 2, deberemos especificar también dicho requisito en nuestro manifiesto añadiendo un nuevo elemento *<uses-feature>*:

```
…
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />
…
```

6. **Referenciar la librería de Google Play Services.**

Una vez hemos configurado todo lo necesario en el manifiesto, y antes de escribir nuestro código, tenemos que seguir añadiendo elementos externos a nuestro proyecto. El primero de ellos será referenciar desde nuestro proyecto la librería con el SDK de Google Play Services que nos descargamos al principio de este tutorial. Para ello, desde Eclipse podemos importar la librería a nuestro conjunto de proyectos mediante la opción de menú “File / Import… / Existing

Android Code Into Workspace". Este paquete se localiza en la ruta "<carpeta-sdk-android>/extras/google/google_play_services/libproject/google-play-services_lib".

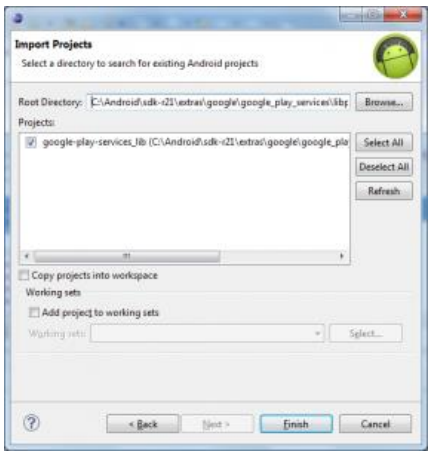


Ilustración 32. Importar librería de Google Play Services al proyecto.

Tras seleccionar la ruta correcta dejaremos el resto de opciones con sus valores por defecto y pulsaremos Finish para que Eclipse importe esta librería a nuestro conjunto de proyectos.

El siguiente paso será referenciar esta librería desde nuestro proyecto de ejemplo. Para ello iremos a sus propiedades pulsando botón derecho / Properties sobre nuestro proyecto y accediendo a la sección Android de las preferencias. En dicha ventana podemos añadir una nueva librería en la sección inferior llamada Library. Cuando pulsamos el botón "Add…" nos aparecerá la librería recién importada y podremos seleccionarla directamente, añadiéndose a nuestra lista de librerías referenciadas por nuestro proyecto.

Como último paso de configuración de nuestro proyecto, si queremos que nuestra aplicación se pueda ejecutar desde versiones "antiguas" de Android (concretamente desde la versión de Android 2.2) deberemos asegurarnos de que nuestro proyecto incluye la librería android-support-v4.jar, que debería aparecer si desplegamos las sección "Android Dependencies" o la carpeta "lib" de nuestro proyecto. Las versiones más recientes de ADT incluyen por defecto esta librería en nuestros proyectos, pero si no está incluida podemos hacerlo mediante la opción del menú contextual "Android Tools / Add Support Library…" sobre el proyecto, o bien de forma manual.

Y con esto hemos terminado de configurar todo lo necesario. Ya podemos usar mapas en nuestra aplicación. En el siguiente punto veremos cómo hacerlo y qué elementos u otras opciones se pueden añadir a éste.

4.10.2. Mapas.

Una vez configurado nuestro proyecto para poder trabajar con mapas sólo nos queda añadirlos. Para añadir un mapa en esta versión de la API usaremos un fragmento, para añadirlo podemos hacerlo de dos formas:

- **Declarándolo en el archivo de diseño de la actividad.**

  Para ello añadimos el siguiente código al archivo XML del layout de la actividad donde queremos mostrar el mapa.

```
…
<fragment
android:id="@+id/map"
android:name="com.google.android.gms.maps.MapFragment"
<!—Compatibilidad con versiones anteriores a Android 3.0:
android:name="com.google.android.gms.maps. SupportMapFragment"-->
android:layout_width="match_parent"
android:layout_height="match_parent" />
…
```

  Si queremos que nuestra aplicación sea compatible con versiones anteriores a la versión 3.0 de Android tenemos que incluir la librería de soporte (normalmente se incluye por defecto) y usaremos el mismo código sólo que en este caso cambiaremos la clase de la etiqueta android:name por su correspondiente en dicha librería.

- **Programáticamente.**

  Podemos también insertar un fragmento en nuestra actividad de manera programática para ello emplearemos el siguiente código:

```
…
mMapFragment = MapFragment.newInstance();
/* Compatibilidad con versiones anteriores a Android 3.0:
 mMapFragment = SupportMapFragment.newInstance();*/
FragmentTransaction fragmentTransaction =
getFragmentManager().beginTransaction();
/* Compatibilidad con versiones anteriores a Android 3.0:
FragmentTransaction fragmentTransaction =
getSupportFragmentManager().beginTransaction();*/
fragmentTransaction.add(R.id.my_container, mMapFragment);
fragmentTransaction.commit();
…
```

  Si queremos compatibilidad con versiones anteriores del sistema a la 3.0 entonces usaremos las clases de la librería de compatibilidad *SupportMapFragment* y *SupportFragmentManager*.

También para ofrecer compatibilidad con las versiones anteriores a la 3.0 nuestra actividad principal tendrá que extender a *FragmentActivity* (en vez de simplemente *Activity* como es lo "normal").

Una vez insertado nuestro mapa sólo tenemos que empezar a trabajar con él. En la antigua versión de la API de Google Maps era bastante poco homogéneo el acceso y modificación de determinados datos de un mapa. Por ejemplo, la consulta de la posición actual o la configuración del tipo de mapa se hacían directamente sobre el control *MapView*, mientras que la manipulación de la posición y el zoom se hacían a través del controlador asociado al mapa (*MapController*). Además, el tratamiento de las coordenadas y las unidades utilizadas eran algo peculiares, teniendo que estar continuamente convirtiendo de grados a micro grados y de estos a objetos *GeoPoint*, etc…

Con la nueva API, todas las operaciones se realizarán directamente sobre un objeto *GoogleMap*, el componente base de la API. Accederemos a este componente llamando al método *getMap()* del fragmento *MapFragment* que contenga nuestro mapa. Podríamos hacerlo de la siguiente forma:

…

```
private GoogleMap mMap;
...
mMap = ((MapFragment)
getFragmentManager().findFragmentById(R.id.map)).getMap();

/* Compatibilidad con versiones anteriores a Android 3.0:

mMap = ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map)).getMap();*/
```

…

Una vez obtenida esta referencia a nuestro objeto GoogleMap podremos realizar sobre él la mayoría de las acciones básicas del mapa. Veremos algunas en los siguientes apartados.

4.10.3. Acciones sobre el mapa.

Llegado a este punto describiremos algunas de las acciones más comunes que se pueden llevar a cabo sobre un mapa, del cual hemos obtenido previamente su referencia como vimos en el punto anterior. Veámoslas:

- **Para modificar el tipo de mapa** mostrado podremos utilizar una llamada a su método *setMapType()*, pasando como parámetro el tipo de mapa:
  - MAP_TYPE_NORMAL.
  - MAP_TYPE_HYBRID.
  - MAP_TYPE_SATELLITE.
  - MAP_TYPE_TERRAIN.
- **En cuanto al movimiento sobre el mapa**, con esta nueva versión de la API vamos a tener mucha más libertad que con la anterior versión, ya que podremos mover libremente nuestro punto de vista (cámara) por un espacio 3D. De esta forma, ya no sólo podremos hablar de latitud-longitud (*target*) y zoom, sino también de orientación (*bearing*) y ángulo de visión (*tilt*). La manipulación de los 2 últimos parámetros unida a posibilidad actual de ver edificios en 3D de muchas ciudades nos abren un mundo de posibilidades. El movimiento de la cámara se va a realizar siempre mediante la construcción de un objeto *CameraUpdate* con los parámetros necesarios. Para los movimientos más básicos como la actualización de la latitud y longitud o el nivel de zoom podremos utilizar la clase *CameraUpdateFactory* y sus métodos estáticos que nos facilitarán un poco el trabajo.

  Así por ejemplo, para cambiar sólo el nivel de zoom podremos utilizar los siguientes métodos para crear nuestro CameraUpdate:
  - CameraUpdateFactory.zoomIn(). Aumenta en uno el nivel de zoom.
  - CameraUpdateFactory.zoomOut(). Disminuye en uno el nivel de zoom.
  - CameraUpdateFactory.zoomTo(nivel de zoom). Establece el nivel de zoom.

  Por su parte, para actualizar sólo la latitud-longitud de la cámara podremos utilizar:
  - CameraUpdateFactory.newLatLng(lat, long). Establece la latitud y la longitud expresadas en grados.

  Si queremos modificar los dos parámetros anteriores de forma conjunta, también tendremos disponible el método siguiente:
  - CameraUpdateFactory.newLatLngZoom(lat, long, zoom). Establece la latitud, la longitud y el zoom.

  Para movernos lateralmente por el mapa (*panning*) podríamos utilizar el método de scroll:
  - CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical). Scroll expresado en píxeles.

  Tras construir el objeto *CameraUpdate* con los parámetros de posición tendremos que llamar a los métodos *moveCamera()* o *animateCamera()* de

nuestro objeto GoogleMap, dependiendo de si queremos que la actualización de la vista se muestre directamente o de forma animada.

Además de los movimientos básicos que hemos comentado, si queremos modificar los demás parámetros de la cámara o varios de ellos simultáneamente tendremos disponible:

- CameraUpdateFactory.newCameraPosition(CameraPosition camera). Recibe como parámetro un objeto de tipo *CameraPosition*. Este objeto puede crearse llamando a *CameraUpdateFactory.newCameraPosition()* con los parámetros deseados.
- getCameraPosition(). Nos devuelve un objeto *CameraPosition*. Accediendo a los distintos métodos y propiedades de este objeto podemos conocer con exactitud la posición de la cámara, la orientación y el nivel de zoom.

4.10.4. Marcadores.

Los marcadores identifican ubicaciones en el mapa. Éstos utilizan un icono estándar, común al estilo y apariencia de los mapas de Google. Es posible cambiar el color, la imagen o el punto de anclaje del icono a través de la API. Los marcadores son objetos de tipo *Marker*, y se añaden al mapa con el método *GoogleMap.addMarker(MarkerOptions).*

Los iconos de los marcadores se dibujan orientados sobre la pantalla del dispositivo en lugar de sobre la superficie del mapa, de modo que si giramos la pantalla, cambiamos la orientación de la misma o hacemos zoom, la orientación del marcador no variara.

Los marcadores están diseñados para ser interactivos. De forma predeterminada, reciben eventos de clic, por ejemplo, que se utilizan a menudo para abrir las ventanas de información. Ajustar la propiedad *draggable* (desplazable) de un marcador a *true* (verdadera) por ejemplo, nos permitirá cambiar la posición del marcador (se utiliza la pulsación larga para activar la capacidad de mover el marcador).

Veremos ahora como añadir un marcador a nuestro mapa, abrir su ventana de información y manejar los eventos correspondientes.

- **Añadir un marcador.** En el siguiente ejemplo se muestra como añadir un marcador a un mapa que se mostrará en las coordenadas de latitud 0 y longitud 0 y qué abrirá una ventana de información al hacer clic que diga “Hola Mundo”.

…

```
private GoogleMap mMap;
mMap = ((MapFragment)
getFragmentManager().findFragmentById(R.id.map)).getMap();
mMap.addMarker(new MarkerOptions()
        .position(new LatLng(0, 0))
        .title("Hola Mundo"));

…
```

- **Ventana de información (InfoWindow).** Una ventana de información le permite mostrar información al usuario cuando toque en un marcador en el mapa. De forma predeterminada, se muestra una ventana de información cuando un usuario pulsa en un marcador, si el marcador tiene un título fijado. Sólo puede mostrarse una ventana de información en un momento. Si un usuario hace clic en otro marcador, la ventana actual se ocultará y se mostrará la nueva ventana de información. Se puede mostrar una ventana de información mediante programación llamando a *showInfoWindow()* desde nuestro objeto *Marker*. Una ventana de información se puede ocultar llamando a *hideInfoWindow()*.

  Una ventana de información se dibuja orientada sobre la pantalla del dispositivo, centrada por encima de su marcador asociado. La ventana de información por defecto contiene el título en negrita, con el texto opcional debajo del título.

  Podemos personalizar el contenido y el diseño de las ventanas de información. Para ello, se debe crear una implementación concreta de la interfaz *InfoWindowAdapter* y luego llamar a *GoogleMap.setInfoWindowAdapter()* para fijar esa ventana como por defecto. La interfaz contiene dos métodos para ser implementados: *getInfoWindow(Marker)* y *getInfoContents(Marker)*. La API llamará primero a *getInfoWindow(Marker)* y si éste devuelve un valor nulo, entonces se llamará a *getInfoContents(Marker)*. Si éste también devuelve un valor nulo, se utilizará la ventana de información por defecto.

  El primero de ellos (*getInfoWindow ()*) le permite ofrecer una vista que se utilizará para toda la ventana de información. El método *getInfoContents()* nos permite simplemente personalizar el contenido de la ventana, pero aún mantiene la información de marco de la ventana por defecto y el fondo.

- **Manejo de eventos.** La API de Google Maps nos permite atender y responder a los eventos de un marcador. Para atender a estos sucesos, se debe establecer el listener correspondiente en el objeto *GoogleMap* al que pertenecen los marcadores. Cuando se produce el evento en uno de los marcadores del mapa,

se invoca al *listener* con el objeto *Marker* correspondiente como parámetro. Se puede establecer un listener para los siguientes tipos de eventos:

- Clic en el marcador. Implementaremos la interfaz *OnMarkerClickListener* para atender a los eventos de clic en el marcador. Para establecer este *listener* en nuestro mapa efectuaremos la llamada al método siguiente: *GoogleMap.setOnMarkerClickListener(OnMarkerClickListener)*. Cuando un usuario hace clic en un marcador, *onMarkerClick(Marker)* se llamará al listener correspondiente al que se le pasará el marcador como argumento. Este método devuelve un booleano que indica si se ha consumido el evento (por ejemplo verdadero si se desea suprimir el comportamiento por defecto). Si se devuelve false, el comportamiento por defecto se producirá, además de nuestro comportamiento personalizado. El comportamiento por defecto es mostrar la ventana de información (si está disponible) y mover la cámara de manera que el marcador está centrado en el mapa.
- Desplazar el marcador. Deberemos implementar los métodos de la interfaz *OnMarkerDragListener* para escuchar los eventos de arrastre en un marcador. Para fijar este *listener* sobre nuestro mapa, llamaremos a *GoogleMap.setOnMarkerDragListener*. Para arrastrar un marcador, un usuario debe presionar mucho en el marcador. Cuando el usuario mueva su dedo por la pantalla, éste se desplazará. Cuando se arrastra un marcador, *onMarkerDragStart(Marker)* se llama principio. Mientras que el marcador se desplaza, *onMarkerDrag(Marker)* será llamado constantemente. Al separar el dedo de la pantalla se llamará a *onMarkerDragEnd(Marker)*. Se puede saber la posición del marcador en cualquier momento llamando al método *Marker.getPosition()*. De forma predeterminada, no es posible desplazar un marcador. Un marcador, debe configurarse explícitamente para ser desplazable antes de que pueda ser arrastrado por un usuario. Esto se puede hacer ya sea con *MarkerOptions.draggable(booleano)* antes de añadirlo al mapa, o *Marker.setDraggable (booleano)* una vez que éste se ha añadido.
- Clic en la ventana de información del marcador. Podemos implementar la interfaz *OnInfoWindowClickListener* para atender a los eventos de clic en una ventana de información. Para fijar este listener sobre nuestro mapa llamaremos al método *GoogleMap.setOnInfoWindowClickListener (OnInfoWindowClickListener)*. La ventana de información se visualiza como una vista completa por lo que no se pueden especificar *listeners* concretos para los elementos incluidos en el interior de esa vista. Es decir,

debemos abstenernos de colocar botones dentro de nuestra ventana de información personalizada.

Además de los marcadores que acabamos de ver se pueden dibujar también otras formas o figuras sobre un mapa. No las describiremos, puesto que no hemos hecho uso de ellas en nuestra aplicación pero si citaremos algunas, son las siguientes:

- **Polilíneas** (líneas poligonales). Para dibujarlas tenemos la clase *Polyline*.
- **Polígonos.** Se utiliza la clase *Polygon* para su dibujado.
- **Círculos.** Para facilitar la construcción de este tipo de polígonos disponemos de la clase *Circle*.

4.11. Persistencia de la información en Android.

Veremos a lo largo de esta sección una breve descripción de los diversos mecanismos que nos ofrece el sistema Android para garantizar la persistencia de nuestra información.

4.11.1. Preferencias compartidas.

Las preferencias compartidas, o *shared preferences*, son un conjunto de funciones de Android que permiten almacenar información de forma local y permanente. Están diseñadas para almacenar datos primitivos, es decir, cadenas de texto, enteros, booleanos, etc… Cada dato almacenado se asocia a una clave única que posteriormente hay que utilizar para acceder a él. Es decir, las preferencias compartidas son esencialmente un diccionario que se puede persistir. Un diccionario de preferencias determinado puede estar asociado a una actividad concreta o a toda la aplicación. En versiones antiguas de Android era posible compartir diccionarios de preferencias entre distintas aplicaciones, incluso si estas tenían identificadores de usuario distintos, pero dicha funcionalidad ha sido deprecada a partir del API level 17.

4.11.2 Almacenamiento en ficheros.

El almacenamiento externo está indicado para almacenar los datos no sensibles de la aplicación y aquellos que sean compartidos entre varias aplicaciones. En algunos dispositivos Android el almacenamiento externo se lleva a cabo en la tarjeta de memoria

extraíble del dispositivo, pero en otros está ubicado en una partición de la memoria interna. En estos últimos la tarjeta de memoria, si existe, se utiliza sólo para almacenar contenido multimedia y las aplicaciones no pueden crear o leer ficheros en ella. En cualquier caso antes de acceder al almacenamiento externo hay que consultar su disponibilidad, para lo cual Android proporciona la función *getExternalStorageState*. El estado del almacenamiento externo puede cambiar en cualquier momento, por ejemplo cuando se extrae la tarjeta de memoria o cuando se conecta el dispositivo a un ordenador. Android gestiona los cambios en la disponibilidad del almacenamiento externo como cambios de configuración del dispositivo.

Dentro del almacenamiento externo hay que distinguir dos casos:

- **Dispositivos Android 2.1 y anteriores.** En ellos la carpeta raíz del almacenamiento externo se obtiene con la función *getExternalStorageDirectory*. Dentro de esta carpeta los contenidos específicos de la aplicación deben almacenarse en una subcarpeta cuyo nombre sea el nombre del paquete de la aplicación. Los contenidos compartidos entre varias aplicaciones deben almacenarse en subcarpetas con nombres preestablecidos de acuerdo a su tipo (*Music*, *Download*, *Movies*, etc…). En estos dispositivos los ficheros del almacenamiento externo no se borran cuando la aplicación es desinstalada.
- **Dispositivos Android 2.2 y posteriores**. En ellos la carpeta adecuada para guardar un contenido en el almacenamiento externo se obtiene con la función *getExternalFilesDir*. Si el contenido es compartido entre varias aplicaciones su ubicación se obtiene con *getExternalStoragePublicDirectory*. Cuando la aplicación se desinstala sus ficheros no compartidos del almacenamiento externo son eliminados.

4.11.3. Bases de datos SQLite.

Android incorpora de serie la librería SQLite 3, que es el estándar de facto en motores de gestión de bases de datos locales. Las bases de datos SQLite son el método más eficaz y flexible para almacenar y procesar grandes cantidades de información de forma eficiente. Esta flexibilidad conlleva una mayor complejidad que el almacenamiento en ficheros o preferencias compartidas, pero aun así es el método de almacenamiento más adecuado para la mayoría de aplicaciones. Aunque la librería SQLite está escrita en lenguaje C, Android proporciona objetos Java que permiten crear y manipular bases de datos sin tener que escribir ningún código en C. Las aplicaciones Android pueden crear tablas, seleccionar registros, y hacer cualquier operación sobre la

base de datos utilizando el lenguaje estandarizado de consulta de datos relacionales SQL.

Cabe destacar que las bases de datos SQLite 3 pueden compartirse entre distintos sistemas que utilicen dicho motor, por lo que es posible crear una base de datos con herramientas externas para posteriormente importarla y utilizarla en una aplicación Android y viceversa. Teniendo en cuenta lo anterior podemos distinguir tres patrones de uso frecuentes de bases de datos SQLite en Android:

- **Base de datos de sólo lectura creada externamente e incluida en la subcarpeta de recursos *raw*.** Puede ser abierta utilizando el método *getReadableDatabase* de la clase *SQLiteOpenHelper*.
- **Base de datos de lectura/escritura creada externamente e incluida en la subcarpeta de recursos raw.** En este caso la aplicación debe copiarla de la carpeta raw a otra carpeta en el área de almacenamiento interno o externo y abrirla utilizando el método *getWritableDatabase* de *SQLiteOpenHelper*.
- **Base de datos de lectura/escritura creada por la propia aplicación.** Se puede crear y abrir utilizando la clase *SQLiteOpenHelper* e importando los datos iniciales dentro del método *onCreate* de dicha clase.

4.11.4. Proveedores de contenido.

Los proveedores de contenido, o *Content Providers*, son uno de los tipos de componentes que pueden formar parte de una aplicación Android. La función de este tipo de componentes es permitir que las aplicaciones puedan publicar un conjunto de datos de modo que otras aplicaciones sean capaces de manipularlos. Cualquier aplicación puede crear su propio proveedor de contenidos, aunque es más frecuente el caso de las aplicaciones que acceden a los proveedores de contenidos que Android proporciona de manera estándar. A través de ellos se puede acceder a la lista de contactos, el registro de llamadas, las alarmas del usuario, la librería de contenido multimedia, etc…

Las aplicaciones se conectan a los proveedores de contenidos utilizando la clase *ContentResolver*, que implementa métodos CRUD (create, read, update, delete) para manipular los datos del proveedor. Con frecuencia los proveedores de contenido del sistema están restringidos y las aplicaciones deben solicitar el permiso correspondiente en el manifiesto.

4.12. MySQL.

Actualmente el gestor de base de datos juega un rol central en la informática, como única utilidad, o como parte de otra aplicación.

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB —desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

MySQL es uno de los gestores de bases de datos de código abierto más populares del mundo. Es una implementación cliente/servidor que consta de un servidor y diferentes clientes (programas/librerías). Podemos agregar, acceder, y procesar datos grabados en una base de datos.

MySQL es muy rápido, robusto, multihilo, multiusuario, multiproceso, fácil de usar, con capacidad para manejar grandes bases de datos, numerosas APIs (C, C++, Eiffel, Java, Perl, PHP, Python and TCL), disponible para varias plataformas, con acceso seguro mediante privilegios y password, soporta ODBC (MyODBC), está basado en ANSI SQL 92, etc…

La conectividad, velocidad y seguridad hacen que MySQL sea altamente conveniente para acceder a bases de datos en Internet. Su última versión es MySQL 5.7, podemos encontrar su manual de referencia en la sección de documentación de la web de desarrolladores de MySQL: http://dev.mysql.com.

En nuestro proyecto haremos uso de esta tecnología para crear nuestra base de datos remota, guardaremos en ella el estado de nuestros envíos. Explicaremos esto con más detalle, más adelante en la sección 5.2.1.

4.13. PHP.

PHP es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo

que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. PHP puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

PHP fue creado originalmente por Rasmus Lerdorf en 1995. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP. Este lenguaje forma parte del software libre publicado bajo la licencia PHP que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término *PHP*.

Las siglas PHP son un acrónimo de "Hypertext Preprocessor" (inicialmente *PHP Tools*, o, *Personal Home Page Tools*). Se trata de un lenguaje cuyos programas se ejecutan en la parte del servidor, y cuyo código escribiremos incrustado con el código HTML. Dispone de múltiples herramientas que te permiten acceder a bases de datos de forma sencilla, por lo que es ideal para crear tus aplicaciones para Internet.

Dado que se ejecuta en el servidor, será necesario pues, que tengamos un servidor web para poder comprobar nuestros programas. La opción que se tomó en la generación del proyecto fue Apache.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite.

Es multiplataforma, lo que significa que funciona tanto para Unix, Windows y MAC OS, de forma que el código desarrollado se puede ejecutar en cualquiera de ellos sin modificaciones.

En ningún caso se envía código PHP al navegador, por lo que todas las operaciones realizadas son transparentes para el usuario, al que le parecerá que está visitando páginas HTML que cualquier navegador puede interpretar.

Programar con PHP tiene muchas ventajas, podemos citar algunas de ellas:

- Muy sencillo de aprender.
- Similar en sintaxis a C y a PERL.
- Clases y herencia.
- Se puede incrustar código PHP con etiquetas HTML.
- Excelente soporte de acceso a base de datos.
- Toda la comprobación de la información enviada por el cliente se hace en el servidor.

- PHP proporciona unos tiempos de ejecución óptimos, en gran parte debido a que está escrito completamente en C++, que utiliza poca memoria.

La última versión es PHP 5.5.0, tenemos acceso a toda la documentación a través de la web de PHP: http://php.net.

Utilizaremos PHP para programar los scripts necesarios para manipular nuestra base de datos MySQL. Explicaremos esto en la sección 5.2.2.

5. Desarrollo.

En esta sección comentaremos los detalles de la implementación de nuestro proyecto. Como ya hemos dicho anteriormente en los objetivos, nuestra aplicación Android almacenará la información de los envíos en una base de datos remota. La comunicación se establecerá por medio de peticiones http a páginas web que contienen los scripts necesarios para la manipulación de la base de datos. Por tanto, hemos considerado dividir esta sección en dos partes:

- En la primera hablaremos de cómo hemos llevado a cabo el desarrollo de nuestra aplicación Android. Esta es evidentemente la parte más extensa de la sección.
- En la segunda comentaremos la estructura de la base de datos utilizada para almacenar la información de los envíos y daremos más detalles de cómo hemos establecido la comunicación entre la aplicación y la base de datos remota por medio de los scripts en PHP.

Antes de comenzar, aclarar que todo el código al que se hace referencia en esta sección puede encontrase en los apéndices de este mismo libro. Así como en el cd que se adjunta con éste.

5.1. Aplicación Android.

Vamos a desarrollar una aplicación Android. Antes que nada lo primero que debemos hacer es consultar el número de usuarios por versión del sistema Android que hay en el mundo. Así podremos saber el alcance que tendrá nuestra aplicación y qué grado de compatibilidad debemos ofrecer para alcanzar el máximo posible de usuarios. Esta información y bastante actualizada, podemos afortunadamente consultarla en la web de desarrolladores de Android, concretamente en la siguiente dirección: http://developer.android.com/about/dashboards/index.html. Mostramos ahora a modo de ejemplo, una de sus estadísticas de uso:

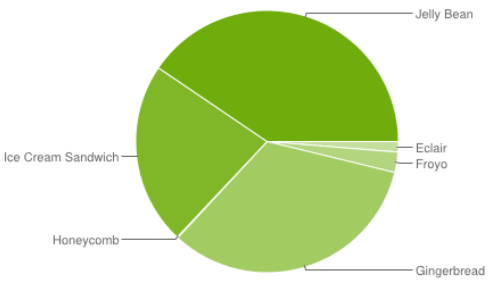


Ilustración 33. Comparativa de uso de versiones del sistema Android.

Versión	Nombre en código	API	Distribución
1.6	Donut	4	0.1%
2.1	Eclair	7	1.2%
2.2	Froyo	8	2.5%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	33.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	22.5%
4.1.x	Jelly Bean	16	34.0%
4.2.x		17	6.5%

Ilustración 34. Porcentaje de uso de las versiones de Android.

Observando estos datos podemos ver claramente que si queremos desarrollar para la última API deberíamos mantener la compatibilidad con las versiones desde al menos la API 10 para alcanzar a un volumen de mercado aceptable. De todos modos, estas consideraciones son tomadas en cuenta de forma automática por el Android SDK, de modo que si instalamos el target de la última API cuando vamos a crear un nuevo proyecto se establece como mínima versión de compatibilidad la API número 8 garantizándose así el mayor abanico de usuarios posible.

Sobre la compatibilidad, aclararemos que toda aplicación Android desarrollada para una API concreta podrá ejecutarse en ésta y en versiones posteriores, de nosotros dependerá en función del código empleado el mantener la compatibilidad con versiones anteriores.

Aclarado todo esto ya podemos comenzar con el desarrollo que nos ocupa. En nuestro caso, vamos a desarrollar para la API 17 y mantendremos la compatibilidad a los valores que establece el SDK por defecto, es decir hasta la API 8.

Para comenzar debemos plantearnos cómo vamos a satisfacer los objetivos, teniendo en cuenta las características de los mismos y la tecnología a emplear. Sabemos, por un lado que la aplicación hará uso de una base de datos remota para guardar la información de los envíos. Por otro que básicamente ofrecerá dos tipos de funcionamiento: el control y el seguimiento de los envíos. Esto nos da la idea de que al menos, deberemos ofrecer dos interfaces gráficas (actividades en Android) una para cada función y evidentemente alguna interfaz que haga de puente entre ellas. Para establecer la comunicación con la base de datos efectuaremos peticiones http a páginas que contendrán los scripts PHP de manipulación necesarios y en el caso de una consulta, éstos devolverán un fichero XML que será parseado por nuestra aplicación obteniendo así los datos de la misma.

En cuanto al diseño de nuestra aplicación, vamos a asemejarlo en la medida de lo posible a lo que sería conocido como el patrón MVC (modelo-vista-controlador). Y digo en la medida de lo posible debido a las particularidades propias de Android. De este modo, tendremos de base (se pueden añadir otros paquetes con utilidades, etc…) tres

paquetes dentro de la carpeta de fuentes (/src), que comprenderán estas tres partes del patrón, es decir el modelo, la vista y el controlador.

Aplicando este patrón a nuestro proyecto tendremos que el modelo contendrá las clases donde se mapearán los datos de nuestra base, la vista contendrá, pues eso las vistas, y el controlador hará de puente y controlará nuestra aplicación, es decir en Android nuestros controladores van a ser por ejemplo nuestros servicios y actividades. Hemos añadido un paquete más de nombre parser, se encargará de parsear los ficheros XML devueltos por nuestros scripts PHP y mapearlos en las clases de nuestro modelo de datos.

Veamos, aplicando este criterio cómo queda el diagrama de clase de los paquetes de nuestra aplicación:

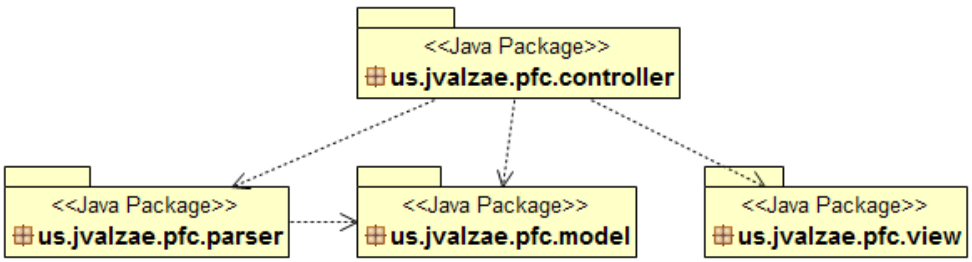


Ilustración 35. Paquetes del proyecto.

En el diagrama se aprecian las relaciones existentes entre los diversos paquetes. Vemos como el controlador se relaciona a su vez con la vista y el modelo (también hará uso del parser) pero como estos dos paquetes, no tienen relación entre ambos, ésta es la base de este patrón de diseño. Buscando así la alta cohesión y el bajo acoplamiento.

El nombre que le hemos dado a nuestra aplicación es ATracker. A continuación, iremos comentando los detalles de la implementación de cada uno de sus paquetes.

5.1.1. Paquete model.

Empezaremos por este paquete ya que su implementación resulta bastante sencilla. Este paquete tan sólo constará de una clase que será la encargada de persistir los datos que lleguen de nuestra base (veremos su estructura más adelante). Como ya hemos comentado con anterioridad, estos datos van a reflejar la información de un

envío con lo que nuestra clase tendrá, básicamente los mismos atributos que campos tenga la tabla de envíos en nuestra base de datos.

La clase que hemos implementado para tal efecto es la clase *Envio* (fichero Envio.java), veamos su diagrama de clases:

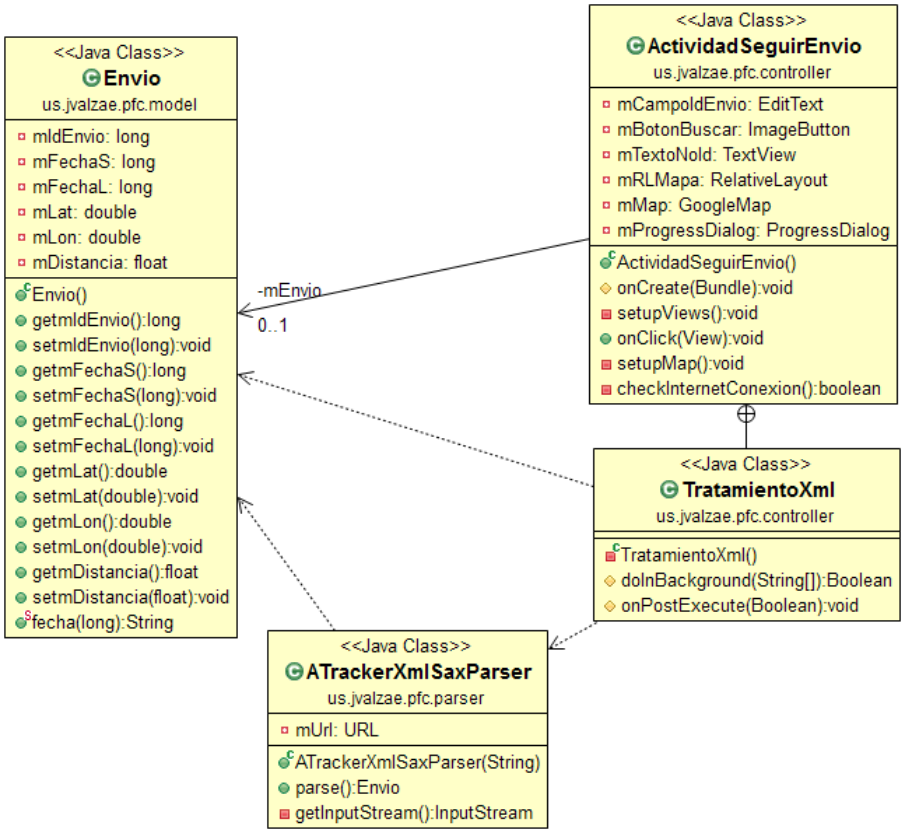


Ilustración 36. Diagrama de clases del paquete model.

Vemos como en la clase disponemos de una serie de atributos privados. Éstos representan cada uno de los campos almacenados en la tabla de nuestra base de datos para los que queremos persistir su valor. También hemos implementado todos los métodos de acceso correspondientes, y un método estático de nombre fecha. Este método recibirá un entero que representa un timestamp (número de segundos transcurridos desde el 1 de Enero de 1970) y devolverá la cadena con el formato de fecha correspondiente al idioma del país que se tenga configurado en las opciones del sistema.

El código correspondiente a la implementación puede verse en los apéndices y se encuentra también en el cd adjunto a esta documentación.

5.1.2. Paquete parser.

La implementación de este paquete también resulta bastante sencilla pues al igual que ocurría con el anterior. Este paquete sólo va a contener una clase que será la que contenga los métodos necesarios para parsear nuestros ficheros XML devueltos por los scripts PHP.

Android nos ofrece varias opciones a la hora de parsear XMLs ya que incluye algunas librerías y clases para tal efecto, que facilitan bastante la tarea. Como los datos que tenemos que parsear no son de una extrema complejidad (sólo una tabla) usaremos el parseador Sax simplificado que nos ofrece la API. El nombre que le hemos dado a la clase es *ATrackerXmlSaxParser* (fichero ATrackerXmlSaxParser.java), este sería el diagrama de la clase:

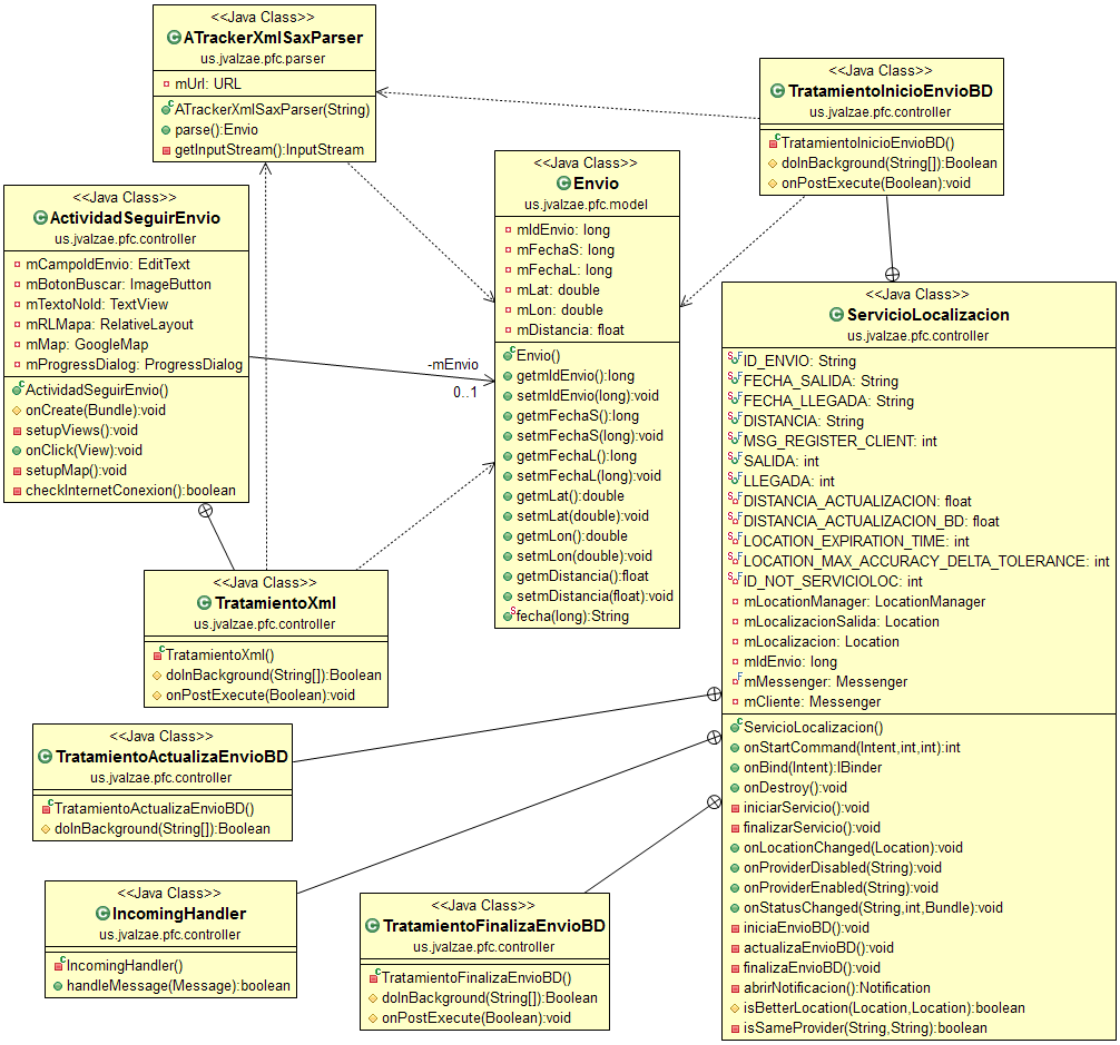


Ilustración 37. Diagrama de clases del paquete parser.

Como se observa esta clase dispone de un atributo que será la URL del script PHP que devuelve un fichero XML. Para obtener ese texto usaremos el método privado *getInputStream()* y para parsearlo el método *parse()*.

En el modelo SAX simplificado de Android, las acciones a realizar para cada evento se definen en la misma clase y asociadas a etiquetas concretas del XML. Y para ello lo primero que haremos será navegar por la estructura del XML hasta llegar a las etiquetas que nos interesan tratar y una vez allí, asignarle algunos de los *listeners* disponibles: de apertura *StartElementListener* o de cierre *EndElementListener* de etiqueta, incluyendo las acciones oportunas.

En nuestro caso, sólo hemos hecho uso de éste último. Lo que haremos al final de cada etiqueta, será asignar a los atributos de la clase *Envio* el correspondiente valor parseado. Para ello usaremos sus métodos de acceso correspondientes (setters).

El código Java correspondiente a la implementación de esta clase puede consultarse en los apéndices y en el cd adjunto.

5.1.3. Paquete view.

El siguiente paquete es el que contendrá la implementación de todas las clases encargadas de representar las vistas de nuestra aplicación. En nuestro caso este paquete sólo incluirá los diálogos de la aplicación. Es aquí cuando recordamos porqué dijimos que intentaríamos acercarnos al patrón MVC en la medida de lo posible. Esto es así porque en Android las vistas correspondientes a las actividades, se encuentran como vimos con anterioridad, dentro de los recursos de la aplicación (/res/layout).

Los diálogos en Android se crean de manera programática aunque también se puede crear un layout específico para los mismos. Por eso, este paquete sólo contendrá el código correspondiente a los diálogos, de los que por supuesto, harán uso nuestras actividades.

En nuestro caso hemos utilizado la clase *DialogFragment* para nuestro cometido. Para ello extendemos de esta clase e implementamos sus método *onCreateDialog* y después construimos el diálogo mediante una clase *builder* (*AlertDialog.Builder*) proporcionada por la API.

Con el *builder* fijaremos un título, un icono, un layout personalizado si queremos, el mensaje del diálogo y sus botones. Todos los diálogos tienen a los sumo tres botones: PositiveButton (si), NegativeButton (no) y NeutralButton (aceptar) para los que definiremos las acciones que queramos.

El código de la implementación de los diálogos se puede ver en los apéndices y en el cd del proyecto.

En total este paquete lo componen siete clases que no se relacionan para nada entre sí, cada una de ellas es uno de los diálogos que puede mostrar ATracker.

5.1.3.1. DialogoActivarInternet (DialogoActivarInternet.java).

Este diálogo se activará cuando queramos hacer uso de los servicios de nuestra aplicación y no dispongamos de conexión a internet. Nos alertará de ello, y nos ofrecerá la posibilidad de acceder a las opciones de configuración del sistema para activar dicha conexión.

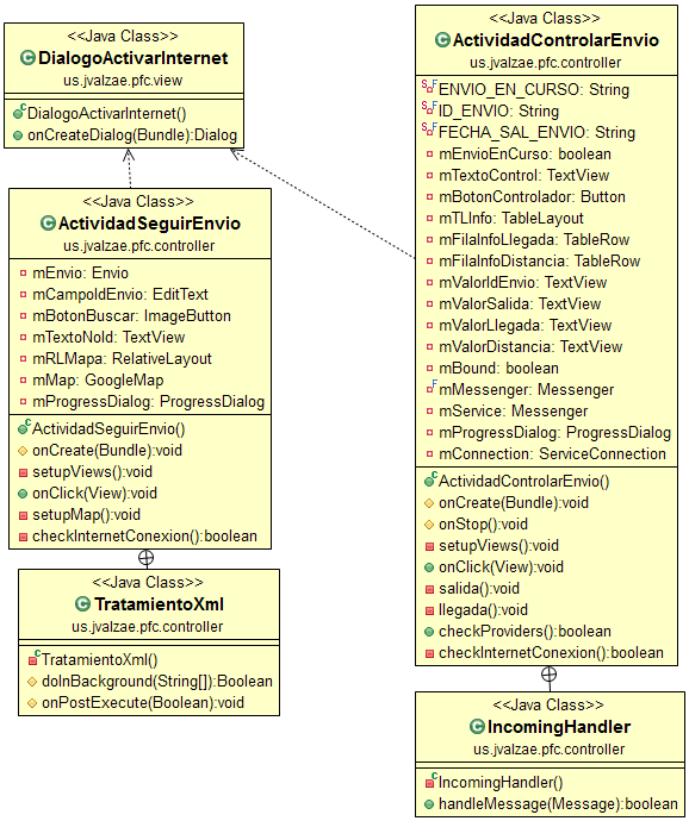


Ilustración 38. Diagrama de clases de DialogoActivarInternet.

Vemos en el diagrama como esta clase se relaciona con las clases *ControlarEnvio* y *SeguirEnvio* del paquete controller.

5.1.3.2. DialogoActivarProviders (DialogoActivarProviders.java).

Este diálogo hace lo mismo que el anterior pero para el caso en el que no tengamos activados ambos proveedores de ubicación (redes y GPS). Se deben tener activados ambos proveedores para que la aplicación funcione correctamente, más adelante veremos por qué.

Este diálogo como puede apreciarse se relaciona únicamente con la clase *ControlarEnvio* del paquete controller. Esto se debe evidentemente, a que para seguir un envío no necesitamos conocer la ubicación de nuestro dispositivo.

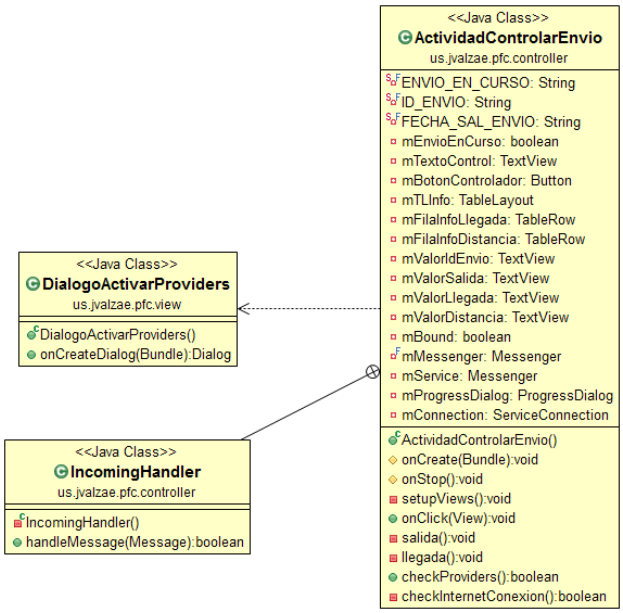


Ilustración 39. Diagrama de clases de DialogoActivarProviders.

5.1.3.3. DialogoAvisoSalida (DialogoAvisoSalida.java) y DialogoAvisoLlegada (DialogoAvisoLlegada.java).

Estos diálogos avisan al usuario que está controlando un envío, de la salida y llegada del mismo. Se activarán por tanto, tras confirmar el usuario la salida o llegada de éste. Su implementación es idéntica y lo único que cambia son los textos que vamos a mostrar (salida/llegada).

Ambos diálogos van a estar relacionados con la clase *ControlarEnvio* del paquete controller por medio de su clase interna *IncomingHandler*.

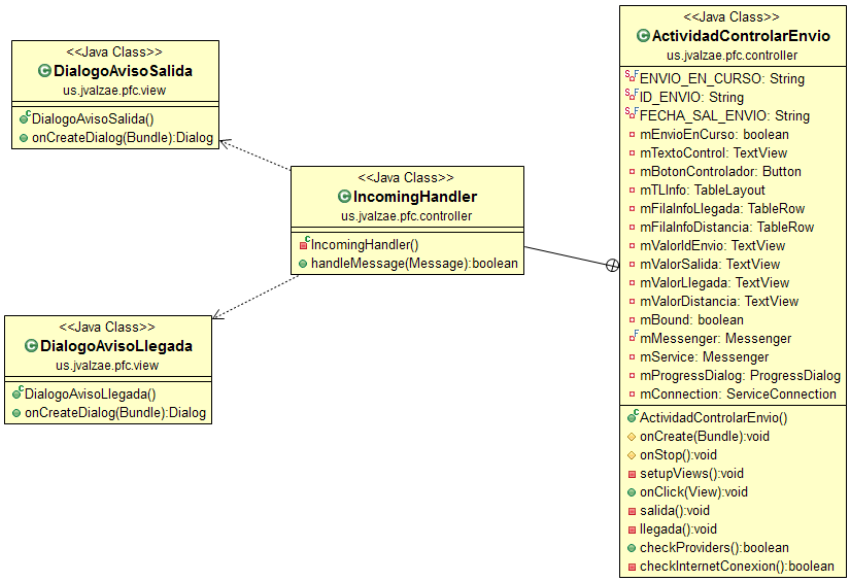


Ilustración 40. Diagrama de clases de DialogoAvisoSalida y DialogoAvisoLlegada.

5.1.3.4. DialogoInfo (DialogoInfo.java).

Este diálogo muestra la información de la aplicación, los créditos y avisos legales de la misma. Se activará, cuando el usuario seleccione en el menú correspondiente acceder a la información de ATracker.

Para este diálogo, hemos diseñado un layout propio (/res/layout) de nombre dialogo_info_layout.xml. Veamos en qué consiste:

Ilustración 41. Diseño gráfico de DialogoInfo.

En las ilustraciones anteriores podemos ver cómo queda nuestro diseño y que elementos hemos utilizado para construirlo. El widget *textoWeb* nos llevará a la web del proyecto al pulsarlo, con lo que hemos implementado un *listener* para el mismo. El *textoGoogle* que no aparece en la pantalla del diseño, se introduce programáticamente y corresponde a la cadena de avisos legales de Google Maps Android API v.2, como vimos en dicha sección.

La clase de este diálogo tiene dos atributos correspondientes a los textos antes mencionados. Está relacionada con la interfaz *OnClickListener* ya que como hemos dicho, implementamos dicha interfaz de forma anónima dentro de la clase para fijar el listener de *textoWeb*. Respecto al resto de paquetes del proyecto esta clase se relaciona con la clase *ActividadPrincipal* del paquete controller. Veamos su diagrama de clases:

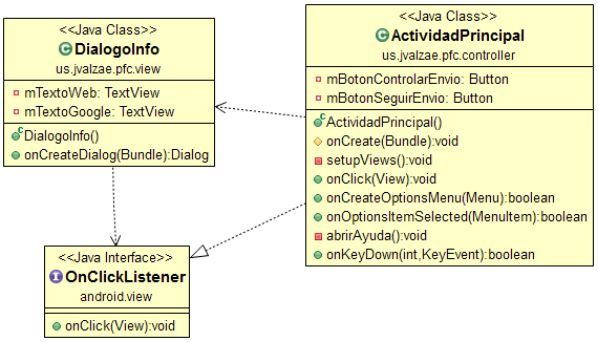


Ilustración 42. Diagrama de clases de DialogoInfo.

5.1.3.5. DialogoSalir (DialogoSalir.java).

Este diálogo solicita confirmación para salir de la aplicación. Se activará cuando el usuario solicite salir o pulse la tecla back del dispositivo, estando dentro de la interfaz principal de la misma. Este es su diagrama de clases:

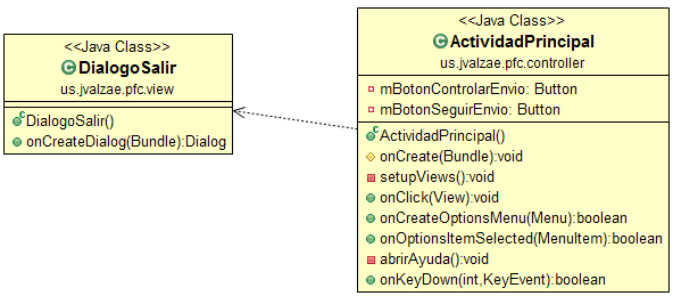


Ilustración 43. Diagrama de clases de DialogoSalir.

5.1.3.6. DialogoSLEnvio (DialogoSLEnvio.java).

Este diálogo se muestra para solicitar confirmación cuando un usuario quiere comenzar o finalizar un envío que está controlando. Una vez confirmado se procederá al inicio o fin de dicho envío, por tanto los textos a mostrar y las acciones resultantes de pulsar la confirmación en el diálogo variarán en función de lo que se haga.

En nuestra clase, hemos creado tres atributos, correspondientes a los textos (título y mensaje) y también un *listener*. Además hemos añadido un constructor estático al estilo *builder* (devuelve el propio objeto) para pasar dichos atributos (sólo los textos) como parámetro. Por otro lado, también hemos creado el método setter para el *listener*, donde pasaremos la clase que lo implementa como parámetro. Como vemos en el diagrama de clases es la *ActividadControlarEnvio* la que implementará la funcionalidad de dicho *listener* dependiendo del caso (salida/llegada). Con todo esto hemos resuelto los inconvenientes antes mencionados. Veamos ahora el diagrama de clases:

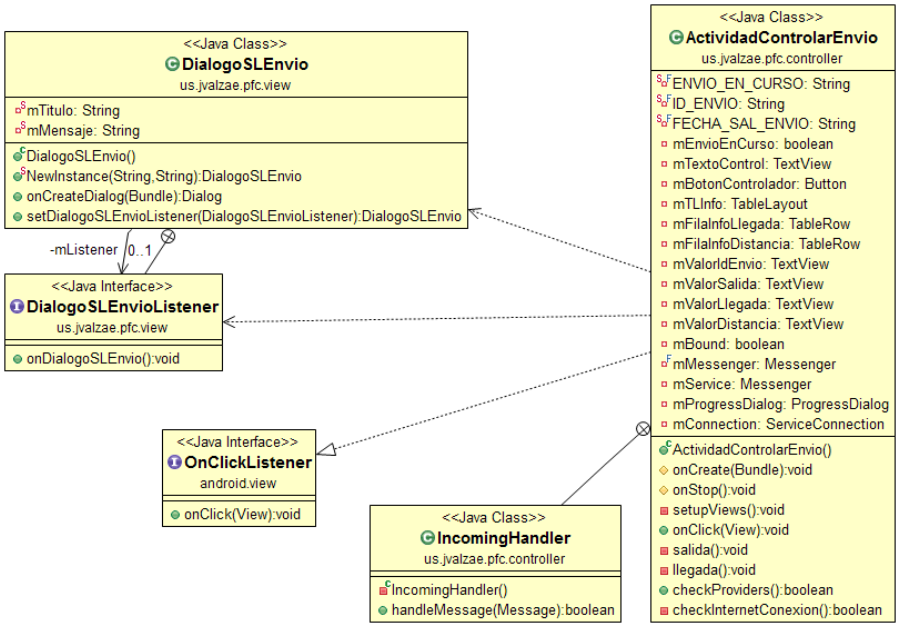


Ilustración 44. Diagrama de clases de DialogoSLEnvio.

5.1.4. Paquete controller.

Este paquete es el más importante de toda la aplicación. Controlará toda nuestra aplicación y es en éste, donde se haya la implementación de los componentes fundamentales de la misma (actividades y servicios).

Estará relacionado (como procede al patrón MVC aplicado para el diseño) con las vistas y el modelo de nuestra aplicación por lo que de su correcta implementación depende el funcionamiento del proyecto, de ahí la importancia del mismo antes mencionada.

Como solución de implementación hemos decidido, como ya comentamos con anterioridad crear una actividad por servicio ofrecido. Es decir, tendremos una actividad para controlar los envíos, otra para seguirlos y una más a modo de interfaz principal. También hemos creado un servicio, que será iniciado por la actividad encargada de controlar los envíos y que se encargará de establecer la ubicación de nuestro dispositivo.

En total este paquete lo compondrán cuatro ficheros Java, correspondientes a las tres actividades y el servicio antes citados. Iremos comentando los detalles de su implementación en las líneas siguientes.

Pero antes, para comprender mejor la estructura del mismo, veamos cuál sería la relación de las clases de éste entre sí. La tenemos en el siguiente diagrama:

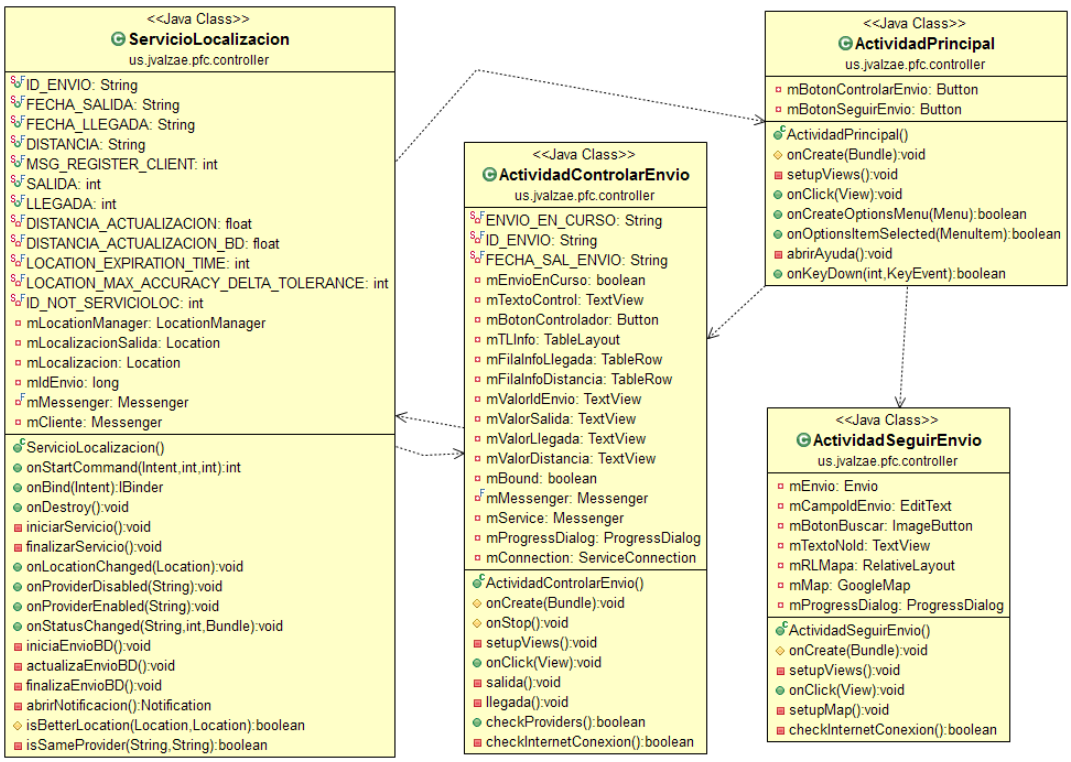


Ilustración 45. Diagrama de clases del paquete controller.

Como vemos en el diagrama es la actividad principal la que se relaciona con las demás actividades que ofrecen el seguimiento y control de los envíos. Esta última como

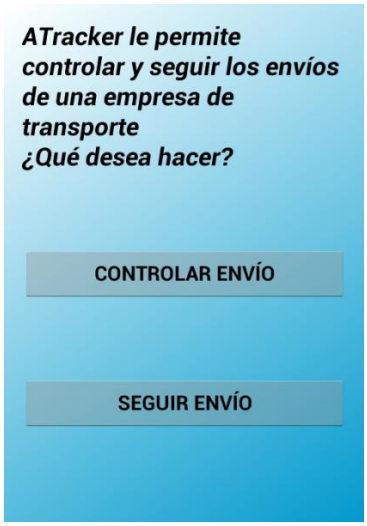
ya dijimos, se relaciona con el servicio que ofrece la localización y que a su vez se relaciona con ésta. El resto de relaciones para con otras clases de fuera del paquete lo veremos en los detalles de la implementación de cada una de éstas.

Pueden consultar el código Java de este paquete en los apéndices de este libro y en el cd que adjuntamos con el mismo.

5.1.4.1. ActividadPrincipal (ActividadPrincipal.java).

Esta clase constituye el punto de entrada de nuestra aplicación como veremos también más tarde cuando hablemos del manifiesto de la misma. Para su implementación hemos extendido la clase *FragmentActivity* (para compatibilidad con versiones anteriores a 3.0) en vez de *Activity*. Desde esta clase, incluida en la librería de soporte y a través de la interfaz que proporciona, podremos acceder a los servicios ofrecidos por ATracker. Es decir a los demás componentes que implementan la funcionalidad del control y seguimiento de los envíos.

El layout de esta actividad se encuentra en la carpeta de recursos a tal efecto (/res/layout) y tiene el nombre de: actividad_principal_layout.xml. El diseño del mismo con sus elementos, es el que se observa en las siguientes ilustraciones:



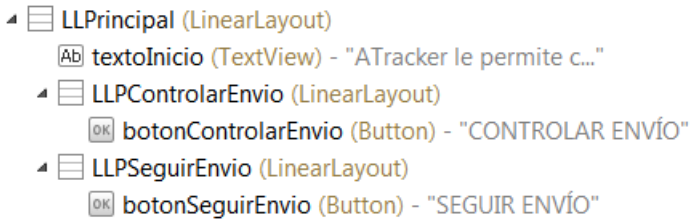


Ilustración 46. Diseño gráfico de ActividadPrincipal.

Como vemos esta actividad tiene una interfaz con dos botones que al pulsarlos nos conducirán a las actividades mencionadas.

Esta actividad y es la única de nuestra aplicación, también incluye un menú de opciones. En las versiones Android 3.0 o superiores este menú aparecerá en la parte superior de la aplicación en la llamada *ActionBar*, donde se muestra el título de la misma. En versiones anteriores deberemos pulsar la tecla menú de nuestro dispositivo para poder visualizarlo. Pesé a estas diferencias no hace falta contemplarlas en nuestra implementación, será el sistema quien se encargue en función de su versión, de mostrar dicho menú de una u otra manera.

Para crear en Android un menú de opciones dentro de una actividad debemos sobrescribir dos métodos:

- **boolean onCreateOptionsMenu(Menu menu).** En éste método, que se autogenera al crear la actividad con el SDK, sólo tendremos que fijar el archivo XML que contendrá el layout del menú.
- **boolean onOptionsItemSelected(MenuItem item).** Con este método definiremos las acciones a realizar para cada una de las opciones de nuestro menú.

El layout correspondiente a nuestro menú se encuentra en la carpeta pertinente (/res/menu) y su nombre es: actividad_principal_menu.xml. El SDK nos proporciona una herramienta para la creación de los diseños de los menús al igual que ocurre con las actividades. Veamos en la siguiente ilustración qué hemos definido para nuestro menú:

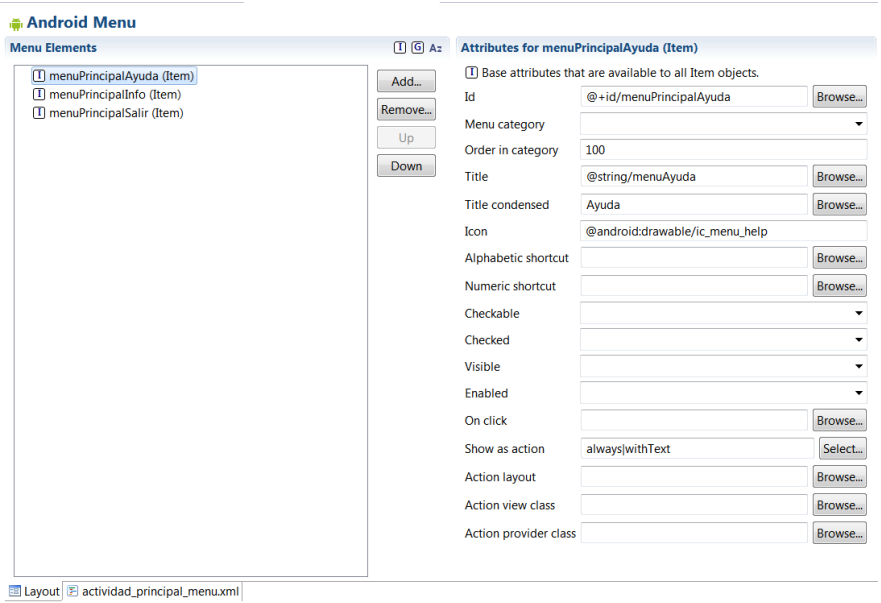


Ilustración 47. Android Menu Editor.

Como podemos observar con la herramienta antes citada podemos construir nuestros menús de forma sencilla. En nuestro caso hemos creado un menú con tres ítems u opciones que serán:

- **menuPrincipalAyuda.** Nos llevará a la web donde se muestra la ayuda de nuestra aplicación. Esta implementado con el método *abrirAyuda*.
- **menuPrincipalInfo.** Esta opción abrirá el diálogo de información de la aplicación. Hemos descrito la implementación del mismo en líneas anteriores.
- **menuPrincipalSalir.** Abrirá el diálogo de confirmación de salida antes descrito en el paquete view. Su implementación puede verse en la descripción de dicho paquete.

Definiremos estas acciones con la función *onOptionsItemSelected.*

Dicho esto veamos ahora cuál es nuestro diagrama de clases para esta actividad:

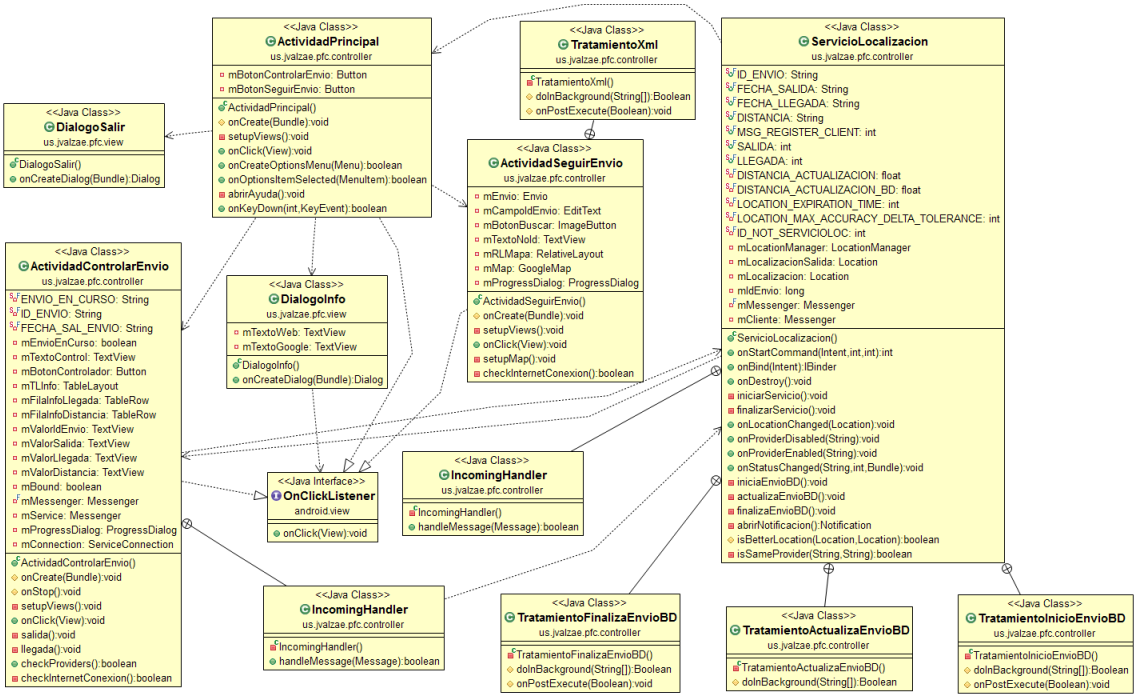


Ilustración 48. Diagrama de clases de ActividadPrincipal.

Vemos básicamente como nuestra actividad tiene dos atributos. Estos corresponden a los dos botones de la interfaz. Implementa la interfaz *OnClickListener* para fijar el *listener* de ambos botones, que no harán sino conducirnos a las otras actividades con las que se relaciona nuestra clase. La relación con la clase *ServivioLocalizacion* se debe a que la notificación del mismo tiene en cuenta cuál es la actividad que se apila por debajo de *ControlarEnvio*. También se relaciona con el

paquete view, con los diálogos que se abren por medio del uso del menú y la tecla back. Para esto último hemos sobrescrito el método *onKeyDown* indicándole que ejecute el diálogo si dicha tecla es pulsada.

Como siempre, se puede consultar el código de la implementación de esta clase en los apéndices de este libro o en el cd adjunto.

5.1.4.2. ActividadControlarEnvio.

Esta es una de las clases y actividades fundamentales de nuestra aplicación. Esta clase proporciona la interfaz y es la encargada de conectarse al servicio, que nos proveerá la ubicación de nuestro dispositivo con el fin de establecer un control sobre nuestros envíos.

Al igual que la ocurría con la actividad anterior hemos heredado de la clase *FragmentActivity* para implementar esta actividad para evitar problemas de compatibilidad con anteriores versiones del sistema.

Veamos ahora que *layout* hemos diseñado como interfaz de esta actividad. Dicho *layout* se encuentra en la carpeta de recursos al efecto (/res/layout) y su nombre es: actividad_controlar_envio_layout.xml.

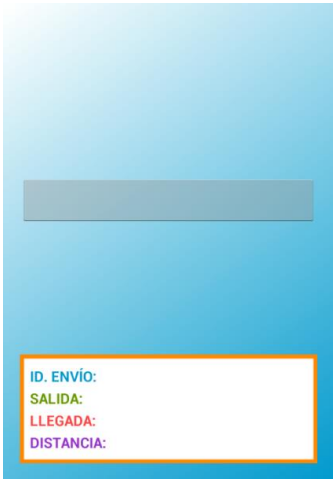


Ilustración 49. Diseño gráfico de ActividadControlarEnvio.

Como podemos observar los textos que variaran a lo largo de la ejecución del programa no están fijados. Su valor se asignará programáticamente cuando la aplicación lo estime oportuno ya que éste no siempre será el mismo.

El diagrama de clases es el siguiente:

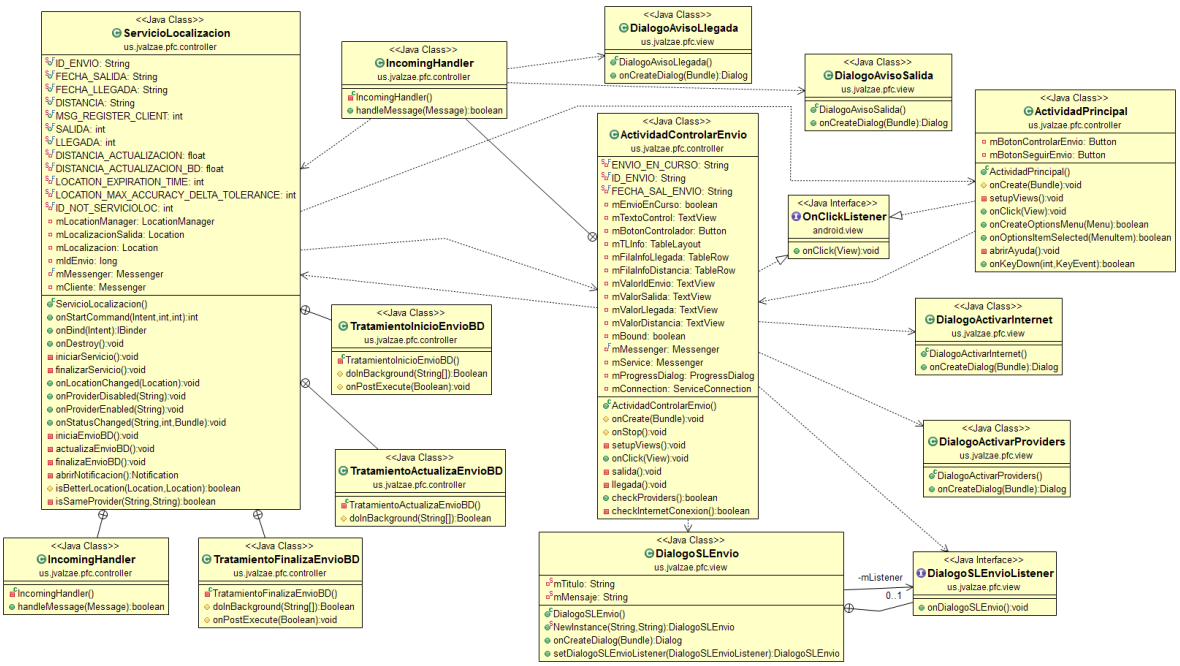


Ilustración 50. Diagrama de clases de ActividadControlarEnvio.

Como se observa nuestra clase contendrá los atributos necesarios para recuperar los elementos de la interfaz gráfica anterior que van a ser manipulados. Además de eso, tenemos otros atributos que guardaran lo siguiente (obviaremos las constantes):

- **private boolean mEnvioEnCurso.** Este atributo contiene un booleano que nos dirá como su propio nombre indica si hay o no un envío en curso. Nuestra aplicación sólo puede controlar un envío por dispositivo en el que se ejecute.
- **private boolean mBound.** Aquí guardaremos otro booleano que nos dirá si estamos o no conectados al servicio, éste nos proporcionará los datos referentes a la ubicación del dispositivo y escribirá éstos en nuestra base de datos remota.
- **private final Messenger mMessenger.** Contendrá una referencia al Messenger o buzón de nuestra clase donde se recibirán los mensajes provenientes del servicio al que estamos conectados.
- **private Messenger mService.** Igual que el anterior pero en este caso es la referencia al buzón del servicio al que nos hemos conectado.
- **private ProgressDialog mProgressDialog.** Contendrá la referencia al diálogo de progreso que mostraremos en función de la tarea desempeñada.

- **private ServiceConnection mConnection.** Referencia a la interfaz *ServiceConnection* cuya implementación es necesaria para la conexión al servicio.

Hablaremos ahora de los métodos de nuestra clase. Además de los métodos propios de nuestra clase padre (*FragmentActivity*) hemos implementado la interfaz *OnClickListener* para atender al botón y comentaremos los siguientes métodos por su relevancia:

- **private void salida().** Este método se ejecutará al pulsar sobre el botón de salida y procesará la salida de un envío. Para ello iniciará el servicio se conectará a él y fijará el valor correspondiente de los atributos de la clase para tal efecto. Todo esto mientras muestra a su vez el diálogo de progreso correspondiente.
- **private void llegada().** Lo mismo que el método anterior, pero cuando se pulsa llegada en el botón de nuestra interfaz. En este caso se finalizará la conexión con el servicio y se detendrá el mismo.
- **private boolean checkProviders().** Éste método se ejecuta al pulsar sobre el botón de salida. Su función es chequear el estado del sistema para ver si están activados los proveedores de ubicación. De no ser así saltará el diálogo correspondiente definido en el paquete view y que ya comentamos anteriormente.
- **private boolean checkInternetConexion().** Hace lo mismo que el anterior y se ejecuta en las mismas condiciones pero para el caso de que no exista ninguna conexión a Internet.

Por último y no menos importante es nuestra clase interna *IncomingHandler* que implementará la interfaz (*Handler.Callback*) y que contendrá el manejador de los mensajes que lleguen de la conexión con el servicio. Estos serán dos: el que nos aporta la información del envío cuando se produce su salida y el que nos aporta la información cuando éste llega.

Respecto a nuestra implementación he de mencionar que normalmente los manejadores se implementan extendiendo de la clase *Handler*, esto sin embargo puede producir fallos de memoria si la clase es interna como en nuestro caso y la cola de mensajes puede alcanzar un tamaño considerable. Aunque en nuestro caso la cola de mensajes es pequeña (un mensaje para la llegada y otro para la salida) el compilador genera el *warning* correspondiente. Para evitarlo, a pesar de que en el caso que nos ocupa no es necesario pueden hacerse dos cosas: convertir la clase en estática y crear una referencia débil a nuestra clase padre dentro de ella, o implementar como hemos hecho la interfaz *Handler.Callback*. Esto evitará la aparición de dicho *warning* y el

consecuente riesgo de pérdida de memoria, aunque curiosamente estemos programando en Java.

Como el lector ya sabrá puede consultar el código de la implementación en los apéndices o si lo desea puede encontrar el archivo de la misma en el cd adjunto al libro.

5.1.4.3. ServicioLocalizacion.

Esta clase es la encargada de implementar el servicio que nos ofrecerá la posición de nuestro dispositivo y escribirla periódicamente en la base de datos remota del proyecto. Está por tanto íntimamente relacionada con la actividad encargada de controlar nuestros envíos que acabamos de ver.

Este servicio tendrá los dos tipos de uso que comentamos estos podían tener (ver servicios aptdo. 4.8). Es decir, será un servicio iniciado y también permitirá la conexión al mismo para obtener la información que éste se encarga de extraer.

Como de un servicio se trata no tendrá por tanto una interfaz asociada. Además dado que el usuario tiene constancia de su ejecución lo hemos programado para su ejecución en el *foreground* y no en el *background* como suele ocurrir. Por supuesto, como todo servicio que se precie lanzará su notificación asociada al mismo en la barra del sistema designada a tal efecto. Veamos su diagrama de clases:

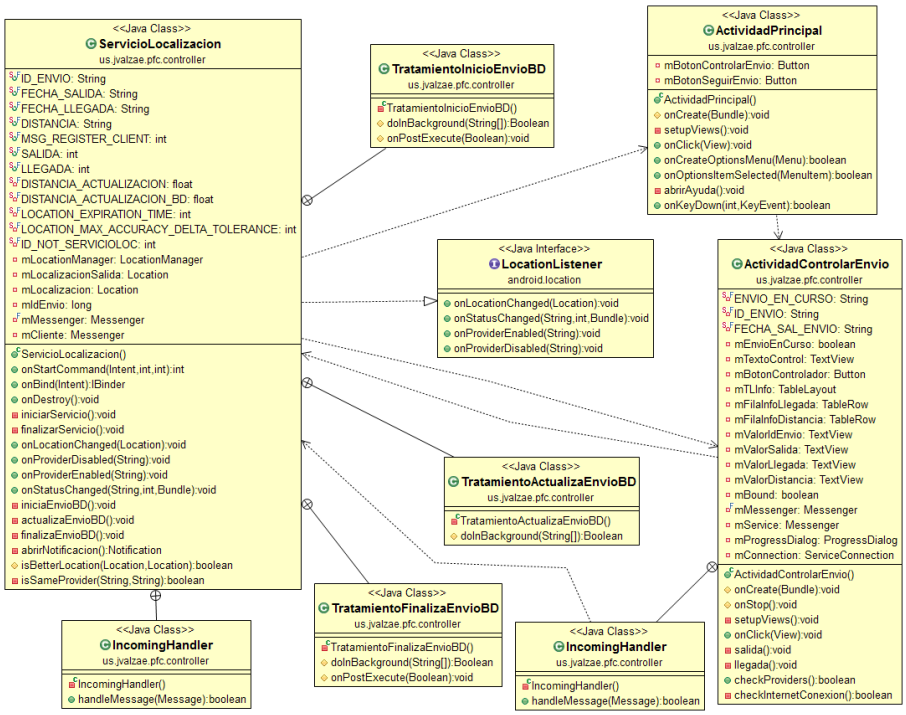


Ilustración 51. Diagrama de clases de ServicioLocalizacion.

El servicio extiende a la clase *Service* e implementa la interfaz *LocationListener* (contiene los métodos para escuchar nuestra ubicación). Pero hablemos primero a parte de las constantes, de los atributos más importantes que hemos definido:

- **private LocationManager mLocationManager.** Referencia al servicio del sistema encargado de proporcionarnos la ubicación de los distintos proveedores de ubicación activados.
- **private Location mLocalizacionSalida.** Referencia al objeto que contendrá los datos de la localización de salida de nuestro envío.
- **private Location mLocalizacion.** Igual que el anterior pero para la localización actual o última localización detectada por nuestro dispositivo.
- **private long mIdEnvio. Contiene el valor del identificador del envío en curso.**
- **private final Messenger mMessenger.** Referencia al buzón de nuestro servicio donde recibimos los mensajes provenientes de la actividad conectada al mismo. Será el manejador el encargado de actuar en consecuencia.
- **private Messenger mCliente.** Igual que el anterior pero del cliente o sea de la actividad que se conecta al servicio.

Estos son los métodos que hemos implementado con mayor relevancia para nuestro servicio:

- **private void iniciarServicio().** Este método como su nombre indica hará todos los ajustes necesarios al comenzare nuestro servicio. Estos son: comenzar el servicio en el foreground (por medio de la función *startForeground*) y lanzar la notificación asociada, fijar a nulos las referencias de nuestras localizaciones y solicitar al servicio del sistema el envío de ubicaciones.
- **private void finalizarServicio().** Lo contario que el anterior. Además escribiremos en la base de datos la última localización antes de destruir el servicio.
- **public void onLocationChanged(Location location**). Este método definido en la interfaz LocationListener se ejecutará al detectar un cambio de ubicación. Lo escribiremos de tal modo que escriba en nuestra base de datos cuando se recorra una distancia mínima de actualización dada en las constantes de la clase.
- **private void iniciaEnvioBD().** Ejecuta la tarea encargada de escribir los datos de un envío en la base datos cuando este se inicia. Para ello hará uso de la clase interna *TratamientoInicioEnvioBD*.
- **private void actualizaEnvioBD().** Lo mismo que el anterior pero para cuando se trata de actualizar los datos de nuestro envío en la base. Se utiliza la clase interna *TratamientoActualizaEnvioBD.*
- **private void finalizaEnvioBD().** Igual que los dos anteriores se ejecutará antes de destruirse el servicio. Se usa la clase interna *TratamientoFinalizaEnvioBD***.**

- **private Notification abrirNotificacion().** Esta función crea la notificación asociada a nuestro servicio y la devuelve como parámetro a ser utilizado por la función *startForeground* cuando éste arranca.
- **protected boolean isBetterLocation(Location location, Location currentBestLocation).** Este método es una copia exacta del que podemos encontrar en la página de documentación para desarrolladores de Android siguiente: http://developer.android.com/guide/topics/location/strategies.html. Como su propio nombre indica compara la bondad de dos localizaciones en base a una serie de criterios.

Como hemos visto durante la descripción de los métodos anteriores existen una serie de clases internas a nuestro servicio. Veremos a continuación cuáles son y a que se debe la necesidad de su implementación.

- **private class IncomingHandler implements Handler.Callback.** Esta clase al igual que ocurría en la actividad anterior se encargará de manejar los mensajes procedentes de nuestro cliente y actuar en consecuencia. Sólo recibirá un tipo de mensaje que contendrá la referencia al buzón del cliente.

Las siguientes clases internas realizan tareas asíncronas en hilos de ejecución diferentes al hilo en el que se ejecuta nuestro servicio. Son necesarias debido a que las tareas que realizan pueden bloquear la actividad del hilo principal y debido a ello es mejor ejecutarlas en otro hilo. De todos modos, no sería obligatorio hacer esto si nuestra aplicación sólo va a ser válida para versiones anteriores del sistema a las 3.0 lo cual, hoy en día es absurdo existiendo ya versiones superiores. Si no se hace la aplicación no funcionará en versiones superiores, dado que éstas incorporan un mecanismo de seguridad que impide la ejecución de este tipo de tareas en el hilo principal.

Para su implementación es necesario extender de la clase *AsyncTask* y sobrescribir si se desea sus dos métodos *doInBackground* (obligatorio ya que indica lo que se hará en background) y *onPostExecute* (se ejecuta al acabar el anterior). Las clases son:

- **private class TratamientoInicioEnvioBD extends AsyncTask<String, Integer, Boolean>.** Esta tarea se encargará de escribir los datos de la localización de salida de nuestro envío en la base de datos. Una vez hecho, ayudándose de nuestro parseador obtendrá el identificador y la fecha de salida de nuestro envío. Estos se comunicarán al cliente en el método *onPosExecute*.
- **private class TratamientoActualizaEnvioBD extends AsyncTask<String, Integer, Boolean>.** Esta tarea actualizará los datos del envío en nuestra base con la ubicación actual de nuestro dispositivo cuando sea necesario. No es necesario por tanto sobrescribir su método *onPostExecute*.

- **private class TratamientoFinalizaEnvioBD extends AsyncTask<String, Integer, Boolean>.** Su función será escribir los datos necesarios para concluir un envío en nuestra base. Terminado esto, comunicará al cliente los datos que proceden por medio de su método *onPostExecute*.

Todas las conexiones con la base de datos se harán por medio de peticiones http a los scripts PHP alojados en nuestro servidor (hablaremos de ello más adelante). Para ello hemos hecho uso de las clases Java *HttpClient* y *HttpPost*.

Para ver el código de la implementación de estas clases vaya a los apéndices de este documento o consulte los archivos del proyecto que se incluyen dentro del cd adjunto.

5.1.4.4. ActividadSeguirEnvio.

Esta es la última clase del total de este paquete. Su importancia es vital ya que implementa la actividad encargada de ofrecer el otro objetivo pendiente de nuestro proyecto: el seguimiento de los envíos.

Su implementación al igual que la del resto de actividades de nuestro proyecto se hace extendiendo de la clase *FragmentActivity*, dado que tendrá que lanzar diálogos implementados con la clase *DialogFragment* evitamos así problemas de compatibilidad.

Veamos ahora que diseño hemos llevado a cabo para la interfaz de esta actividad. Como de costumbre éste está contenido dentro de los recursos (/res/layout) y el nombre de tal archivo es: actividad_seguir_envio_layout.xml.

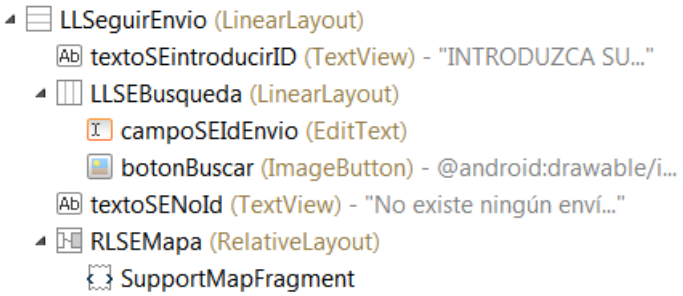


Ilustración 52. Diseño gráfico de ActividadSeguirEnvio.

Como podemos ver hemos incluido en el layout el fragmento con el mapa de Google que mostrará la posición de nuestros envíos. Además está el cuadro de texto donde introduciremos nuestro identificador de envío y el botón de buscar correspondiente. Los widgets que aquí vemos se irán o no visualizando en función de nuestras acciones por tanto esto está programado dentro del código de la clase.

Pasaremos ahora al análisis de nuestra clase, comentaremos como de costumbre los atributos, métodos y clases escritos a fin de obtener nuestros objetivos. Pero antes de todo esto echemos un vistazo al diagrama de clases que se muestra en la ilustración siguiente:

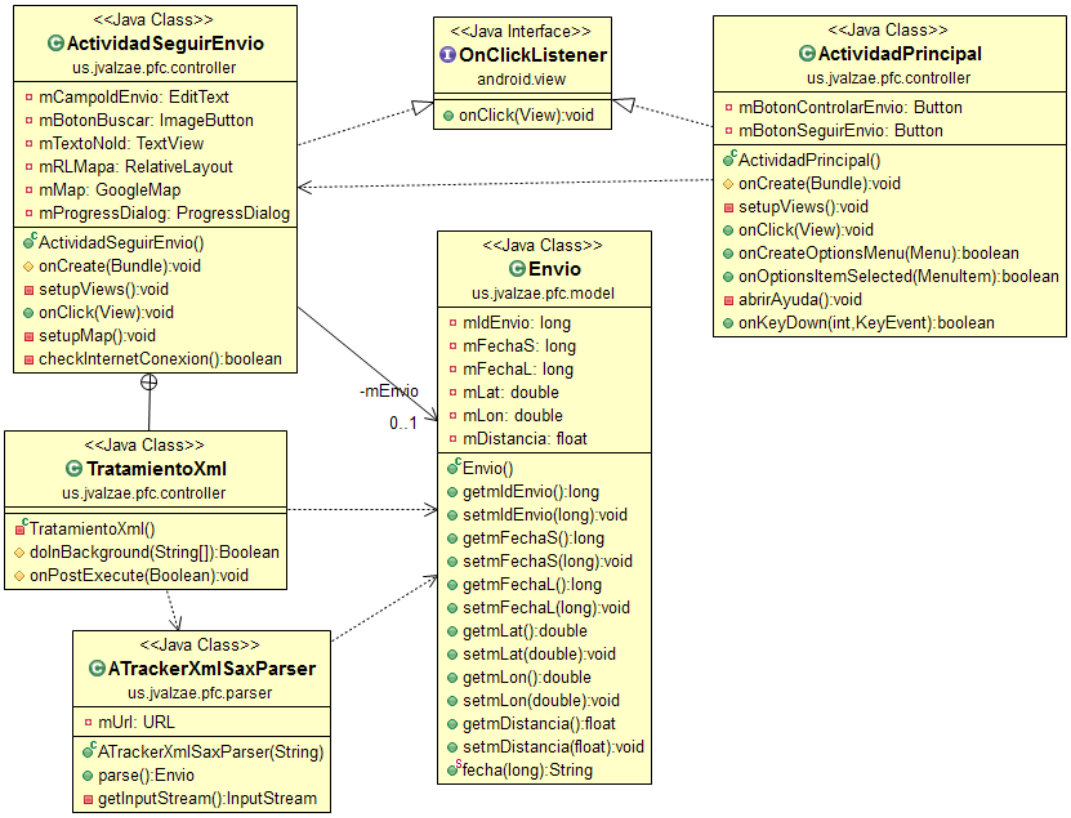


Ilustración 53. Diagrama de clases de ActividadSeguirEnvio.

Ahora sí, diremos que atributos son los más relevantes dentro de la clase (obviaremos entre otros las constantes como de costumbre):

- **private Envio mEnvio**. Contiene una referencia a nuestra clase Envio del paquete model y será aquí donde se parseen los datos del envío a buscar.
- **private GoogleMap mMap.** Es la referencia a nuestro mapa que necesitaremos para manipularlo.

- **private ProgressDialog mProgressDialog.** Referencia al diálogo de progreso que mostraremos mientras se identifica el envío introducido en el cuadro de texto.

Comentemos a continuación los métodos que hemos implementado excluyendo, como solemos hacer en nuestras descripciones, los métodos que implementan la interfaz OnClickListener (utilizados para el botón de buscar). Son:

- **private void setupMap().** Éste es el método más extenso de nuestra actividad sino el de todo el proyecto. Se encarga de configurar el mapa con los datos de nuestro envío para mostrarlo correctamente. Añade el marcador con la posición de nuestro envío y le crea a su vez una ventana de información (con su propio layout: ventana_info_mapa_layout.xml) con los datos de éste.
- **private boolean checkInternetConexion**(). Es idéntico al método encontrado en la clase *ActividadControlarEnvio*. Se lanzará en caso de que no haya conexión a Internet al pulsar el botón de búsqueda.

Dado que en esta actividad deberemos, como sucedía con el servicio conectarnos con la base de datos y esta operación puede ser bloqueante dispondremos de la clase interna *TratamientoXml*. Su función será en el background acceder y parsear los datos del envío de nuestra base de datos remota y después (*onPostExecute*) fijar la interfaz. Para esto, llamará al método *setupMap* antes descrito si el envío existe y si no es así, hará visible el texto que muestra un identificador no válido (ver el *layout* de la actividad).

El código Java puede verse en los apéndices o dentro del cd que se adjunta con el documento.

5.1.5. Recursos de la aplicación.

En este punto diremos que recursos hemos utilizado en nuestra aplicación aparte de los contenidos en la carpetas /res/layout y /res/menu que hemos mencionado durante la anterior descripción del código fuente del proyecto.

Tenemos un icono personalizado para la aplicación que se encuentra en las carpetas drawable correspondientes y que hemos generado con el asistente para nuevo proyecto del Android SDK. Por otro lado hemos añadido los siguientes archivos a la carpeta /res/drawable:

- **borde.xml.** Archivo donde definimos el borde anaranjado que envuelve a las ventanas de información de nuestra interfaz. Al hacerlo con un archivo XML nos libramos de tener en cuenta distintos dispositivos.

- **fondo.xml.** Este fichero contiene el fondo degradado en tonos azules que caracteriza a nuestra interfaz. Al igual que en el caso anterior, así no tendremos que preocuparnos por diferentes dispositivos con tamaños y resoluciones de pantalla diferentes.
- **logo_atc.png.** Imagen con el logo del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla que se muestra en el diálogo de información acerca de la aplicación.
- **logo_etsii.png.** Igual que el anterior (se muestra en el mismo diálogo) pero éste es el logo de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla.
- **logo_us.png.** Lo mismo, pero es el logo de la Universidad de Sevilla.

En la carpeta /res/values también hemos añadido el archivo **colors.xml** que contiene la definición de todos los colores usados por la aplicación. Además hemos introducido la definición de todas y cada una de las cadenas de texto utilizadas en el archivo **strings.xml** de esta misma carpeta. Con esto será muy sencillo traducir la aplicación al idioma que deseemos facilitando así futuras actualizaciones.

Finalmente hemos incluido la carpeta /UML en la raíz del proyecto donde se pueden encontrar todos los diagramas de clases con los que hemos ido ilustrando el desarrollo.

5.1.6. Manifiesto de la aplicación.

Antes de acabar con el desarrollo de nuestra aplicación, y antes de subirla a algún market (si se desea), deberemos repasar el manifiesto. Es importante, porque como ya comentamos, este archivo es el que aporta toda la información al sistema cuando se instala una aplicación.

En nuestro archivo AndroidManifest.xml, deberemos incluir y tener en cuenta todas las consideraciones que no hayamos hecho con anterioridad durante el desarrollo.

Para editar el manifiesto, podemos hacerlo manualmente o haciendo uso de la herramienta que nos facilita el SDK de Android para tal efecto. Aun así, algunas líneas y etiquetas sólo pueden añadirse manualmente al mismo. Veremos a continuación cuáles han sido en nuestro caso.

En la etiqueta *<manifest>* hemos añadido la siguiente propiedad (en negrita) para permitir la instalación de la aplicación no sólo en la memoria del dispositivo sino también en la de la tarjeta de memoria.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="us.jvalzae.pfc"
    android:installLocation="auto"
    android:versionCode="1"
    android:versionName="1.0" >
…
```

En la siguiente etiqueta (*<uses-sdk>*), hemos indicado cuál es la API mínima del sistema Android que será compatible con nuestra aplicación y para qué API ha sido programada. En nuestro caso, tras observar las estadísticas de uso de las diferentes versiones del sistema como vimos al principio de esta sección, hemos decidido programar para la última versión, pero manteniendo la compatibilidad hacia atrás hasta la versión 2.2 Android Froyo como recomienda automáticamente el propio SDK.

```
…
<uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
…
```

Las siguientes etiquetas que hemos añadido incluyen todo lo necesario para trabajar con la API v2 de Google Maps como ya explicamos anteriormente.

```
…
<!-- Permisos -->
<permission
        android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" >
    </permission>
    <uses-permission android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission    android:name="android.permission.ACCESS_NETWORK_STATE"
/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

```
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
    <uses-permission    android:name="android.permission.ACCESS_FINE_LOCATION"
/>
<!--Uso de líbrerias OpenGL ES v.2.0 -->
 <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />
…
<!--API Key -->
…
<application>
…
<meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyA3dkcLUDtpCs0lsAvA2ijf9NgA02k9g84" />
…
</application>
…
```

Para acabar también hemos añadido las siguientes propiedades a nuestras actividades:

- ***android:screenOrientation="nosensor"*** a la etiqueta *<activity>* de todas las actividades del proyecto. Esto, impedirá que la pantalla cambie de orientación al girar el dispositivo y hará que se mantenga fija en posición vertical.
- ***android:launchMode="singleTop"*** en la etiqueta *<activity>* de *ActividadControlarEnvio*. Servirá para indicar que no se cree la actividad si ya existe una en la pila, esto es así porque la notificación del servicio abrirá dicha actividad y si existe, no queremos que esté dos veces en la pila.

Una copia íntegra de todo el código de este archivo puede verse en los apéndices y en el cd adjunto a esta documentación.

5.1.7. Firmado y publicación de la aplicación en Google Play.

Aunque no hemos llevado a cabo este paso, no vamos a publicar la aplicación en Google Play, aunque si la firmaremos. Dada su importancia de cara al desarrollo de aplicaciones Android, si comentaremos los pasos necesarios para llevarlo a cabo.

Toda aplicación que quiera subirse a un market, debe estar previamente firmada digitalmente con la firma de los desarrolladores. Para ello, una vez finalizado nuestro proyecto en Eclipse haremos clic con el botón derecho del ratón sobre el icono del mismo y seleccionaremos la opción *"Export Signed Application Package…"* dentro del menú *"Android Tools"*, indicamos el nombre del proyecto a exportar pulsamos siguiente y se abrirá entonces el cuadro de diálogo que vemos en la ilustración de abajo.

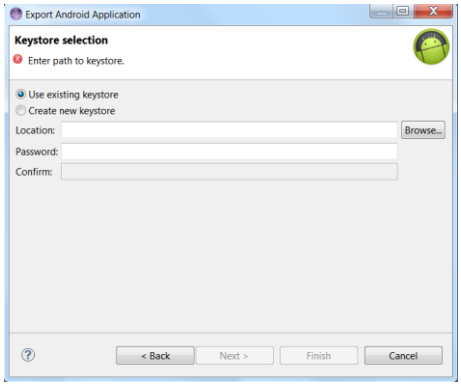


Ilustración 54. Firmado de aplicación Android.

Podemos usar un propio *keystore* o crear uno nuevo (siempre con la extensión .keystore) y le añadimos nuestra contraseña. Éste será el almacén de claves de nuestras aplicaciones.

En nuestro caso antes de firmar la aplicación, como hacemos uso de la API de Google Maps v.2 deberemos solicitar una nueva API Key para producción, ahora sí usando nuestro propio keystore y no, el de depuración proporcionado por el SDK de Android. De lo contrario los mapas que hayamos utilizado no se visualizarán. El proceso es el mismo al descrito en el apartado 4.10.1, cambiando el nombre del archivo .keystore, el alias de la key y las claves por las nuestras.

Solucionado el problema anterior, continuaremos con el proceso de firmado. En la siguiente ventana asignaremos un alias a la key de nuestra aplicación y daremos una validez al certificado durante una duración que sea mayor o igual a 25 años. Deberemos rellenar también otros campos, podemos verlo en la imagen siguiente.

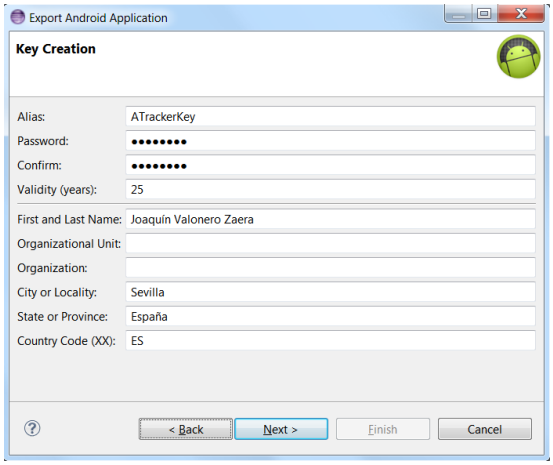


Ilustración 55. Creación de un almacén de llaves.

Lo siguiente será indicar al asistente la ruta donde queremos guardar el fichero .apk de nuestra aplicación firmado y listo para su publicación.

Una vez firmada la aplicación ya podemos subirla a algún Android market. En el caso de Google Play deberemos registrarnos previamente con una cuenta de desarrollador (son 25 $ de por vida) antes de poder subir la aplicación.

Si como es lógico en un futuro realizamos actualizaciones de nuestra aplicación, antes de firmarla hemos de modificar en el manifiesto el atributo *android:versionCode* por un número entero.

Con esto queda expuesto todo el proceso necesario para la firma y puesta en mercado de una aplicación Android.

5.2. Base de datos MySQL y Sitio Web.

En esta sección comentaremos la parte del desarrollo que esta fuera del contexto de Android pero necesaria para el funcionamiento de nuestra aplicación.

Primeramente hablaremos sobre nuestra base de datos y finalmente sobre los scripts PHP que hemos programado para su manipulación a través de las llamadas http que efectuamos desde nuestra aplicación Android.

La base de datos y los respectivos scripts PHP se encuentran alojados en el servidor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla. Podemos ver el código en los apéndices y también en el cd que se adjunta con la presente memoria.

5.2.1. Base de datos MySQL.

La base de datos que necesitamos debe guardar los datos sobre los diferentes envíos que se vayan produciendo. Estará alojada en el servidor antes citado. Este servidor dispone de su propio panel *phpMyAdmin*, que usaremos para construir nuestra base.

El nombre de la base de datos que hemos creado es **jvalzae**. Para su implementación, hemos abordado el problema de manera sencilla y por tanto, su estructura constará de una sola tabla, que contendrá los siguientes campos de información relativos a nuestros envíos.

	Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
☐	idEnvio	bigint(20)			No	*None*	AUTO_INCREMENT
☐	latitud	double			No	*None*	
☐	longitud	double			No	*None*	
☐	altura	double			No	*None*	
☐	latitudS	double			No	*None*	
☐	longitudS	double			No	*None*	
☐	alturaS	double			No	*None*	
☐	fechaS	bigint(20)			No	*None*	
☐	fechaL	bigint(20)			No	*None*	
☐	distancia	float			No	*None*	

Ilustración 56. Estructura de la base de datos.

TABLA envios

- **idEnvio.** Identificador del envío. Clave primaria y auto-incremental.
- **latitud.** Latitud actual del envío en grados.
- **longitud.** Longitud actual del envío en grados.
- **altura**. Altura actual si la hubiere, en metros sobre el nivel del mar del envío. La guardamos para futuros usos.
- **latitudS.** Latitud de salida del envío en grados.
- **longitudS.** Longitud de salida del envío en grados.
- **alturaS.** Altura de salida si la hubiere, en metros sobre el nivel del mar del envío. Guardada para futuros usos.
- **fechaS.** Fecha de salida del envío en formato timestamp.
- **fechaL.** Fecha de llegada del envío en formato timestamp.
- **distancia.** Distancia en metros total recorrida por el envío.

5.2.2. Sitio Web.

Comentaremos a continuación el código HTML y los scripts PHP que hemos alojado en el sitio web del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Sevilla. La URL del sitio, que contiene las páginas de nuestro proyecto es la siguiente: http://www.atc.us.es/~jvalzae.

El código HTML alojado se reduce a dos archivos:

- **index.html.** Contiene la página de presentación del proyecto y la información referente al mismo que también puede verse desde nuestra aplicación Android en el menú de información. Las imágenes que se ven se encuentran dentro de la carpeta /images en la raíz del sitio.
- **ayuda.html.** Contiene la página de ayuda del proyecto a la que se accede desde nuestra aplicación Android pulsando en el menú de ayuda. Sus indicaciones son las mismas que las que veremos en el manual de usuario en el apartado siguiente. Las imágenes que se ven se encuentran dentro de la carpeta /images en la raíz del sitio.

El código PHP para la manipulación de nuestra base está comprendido por los siguientes ficheros:

- **conexion.php.** Contiene el código necesario para establecer una conexión con la base de datos del proyecto.
- **iniciar_envio.php.** Aquí tenemos el código para iniciar un nuevo envío en la base de datos. Este script devolverá un fichero XML de salida que contendrá el identificador del envío creado y su fecha de salida. Éste es parseado por nuestra aplicación para obtener dicha información.
- **actualizar_envio.php.** Este script será llamado por nuestra aplicación para ir actualizando en la tabla la posición de nuestro envío y eso, es precisamente de lo que se encarga el código que contiene.
- **finalizar_envio.php.** Su código finalizará el envío en la base de datos escribiendo la posición del destino, la fecha de llegada y la distancia total recorrida.
- **consultar_envio.php.** Este script es llamado desde la *ActividadSeguirEnvio* y buscará los datos de un envío dado un identificador. Devuelve un fichero XML de salida con los datos, que es parseado por la aplicación para obtener así estos.

Como de costumbre, puede visualizar todo el código al que aquí se hace referencia en los apéndices de este documento o dentro del cd de material adjunto.

6. Manual de usuario.

Aunque hemos desarrollado la aplicación con el fin de que su uso resulte de lo más sencillo para el usuario, incluimos en esta sección un breve manual describiendo su modo de empleo para que éste, pueda servir como referencia a los usuarios en caso de asistencia. Estas pautas, son las mismas que el usuario puede encontrar al pulsar sobre la tecla de ayuda del menú en la pantalla principal de la aplicación. Esto nos lleva a la página web de ayuda del proyecto, http://www.atc.us.es/~jvalzae/ayuda donde están alojadas las instrucciones de manejo de la aplicación que siguen a continuación:

INSTRUCCIONES DE MANEJO DE ATRACKER

ATENCIÓN: ATracker requiere que su dispositivo conste de sensor GPS, es por eso que para su correcto funcionamiento debe tener habilitada dicha característica en los ajustes de su terminal.

Como se indica al inicio de esta aplicación usted puede utilizar ATracker para dos propósitos diferentes que se complementan en la tarea de establecer el seguimiento de un envío concreto.

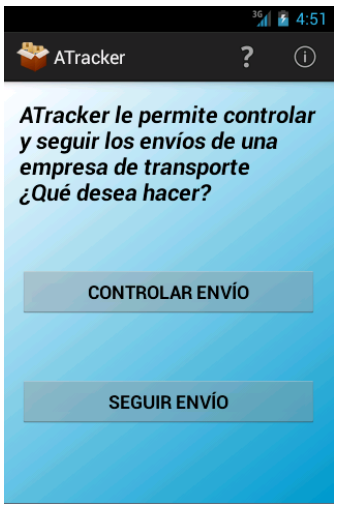


Ilustración 57. Manual: interfaz principal.

Si usted pulsa sobre el primer botón: **CONTROLAR ENVÍO** se abre una nueva pantalla que le permite controlar un envío.

El terminal se comporta entonces como un señuelo detectando su posición a través del sensor GPS de su terminal y escribiéndola en una base de datos remota. Esta actividad es la que deben llevar a cabo los operadores de la agencia de transporte cuando vayan a realizar un nuevo porte.

Para comenzar un envío pulsaremos sobre el botón **SALIDA**.

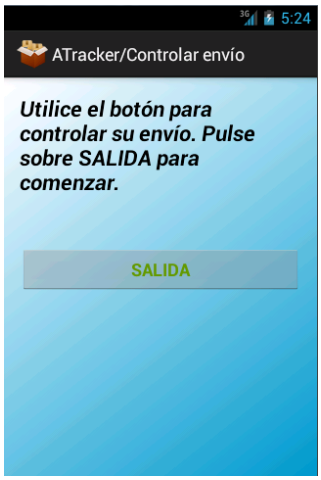


Ilustración 58. Manual: salida.

Una notificación aparecerá en la barra de estado del sistema indicando que el envío se haya en curso. Espere unos instantes mientras el envío se inicia.

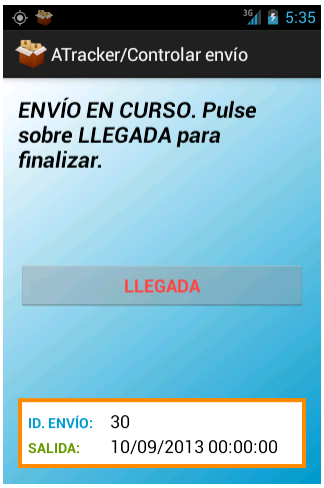


Ilustración 59. Manual: llegada.

En el cuadro de la parte inferior podrá visualizar los datos referentes al mismo.

Para finalizar un envío pulsaremos sobre el botón **LLEGADA**.

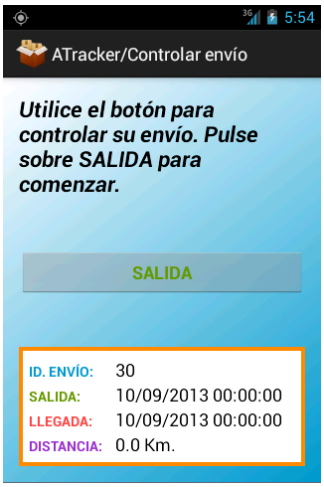


Ilustración 60. Manual: datos de un envío.

Espere unos instantes mientras se finaliza, entonces la notificación desaparecerá de la barra de estado. Podrá ver los datos relativos a la llegada del envío en el cuadro inferior de la pantalla.

Por el contrario si estando en la pantalla principal, usted pulsa sobre el segundo botón: **SEGUIR ENVÍO** se abre una nueva pantalla que le permite seguir un envío.

A continuación, introduzca en el cuadro de texto el número correspondiente a su identificador de envío y pulse sobre el botón adyacente (imagen de una lupa) para averiguar la posición del mismo.

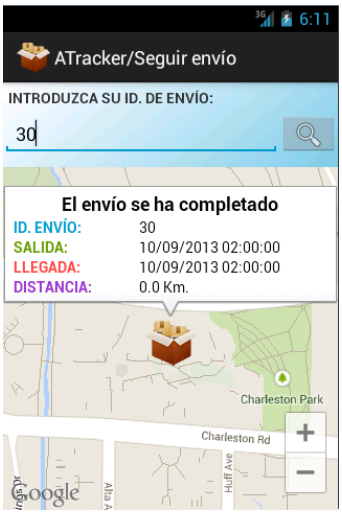


Ilustración 61. Manual: seguimiento de envíos.

Si el identificador que ha introducido es correcto entonces se mostrará un mapa con la posición de su envío: imagen de un paquete. Puede hacer clic sobre la misma para ver la información concerniente a dicho envío.

De lo contrario, se le indicará que el identificador introducido no existe.

GRACIAS por usar ATracker.

Este es todo el contenido que puede encontrarse en las instrucciones de manejo de la aplicación. Esperamos que en el caso de necesitar ayuda, este manual aclare las dudas sobre el manejo de la aplicación que los usuarios puedan plantearse.

7. Conclusiones.

Cuando empecé a plantear mi Proyecto Fin de Carrera pensé el utilizar una tecnología que fuese nueva para mí y que así a su vez, me aportase nuevos conocimientos.

En mi formación, a lo largo del estudio de la carrera y en mi experiencia laboral he trabajado mayoritariamente con el lenguaje de programación Java. Se me ocurrió entonces, tras conocer como usuario el sistema operativo Android, que haría el proyecto para la citada plataforma. Sin duda los conocimientos de Java y de las distintas arquitecturas estudiadas en la carrera han sido determinantes para poder realizar el proyecto.

La adquisición de los conocimientos para abordar el proyecto, ha llevado su tiempo ya que las tareas a emprender eran nuevas para mí. La situación, me ha obligado a ser autodidacta y es muy de agradecer la estupenda documentación que puede encontrase en la web de desarrolladores de Android, ya que facilita enormemente el aprendizaje de la citada tecnología.

El reto más difícil a lo hora de implementar la aplicación ha sido sin duda, la implementación del servicio encargado de detectar la ubicación y escribirla en la base de datos, así como el establecimiento de la comunicación entre éste y la actividad conectada al mismo. Sin embargo, gracias a la maravillosa documentación y a la enorme cantidad de información disponible en Internet, la complicación no ha transcendido más allá.

La nueva versión de la API de Google Maps que hemos empleado en el desarrollo, requiere de un proceso largo y quizás un poco complicado para su configuración en el entorno de trabajo, pero ésta facilita mucho el uso y manejo de los mapas respecto de la anterior versión.

A mi juicio el proyecto está bastante completo pues abarca el uso de los componentes más importantes de una aplicación android (actividades, servicios, intents, sensores, acceso a base de datos, localización, mapas, etc...). Esto claramente me permitirá afrontar futuros desarrollos para Android con alguna experiencia. Sin duda ahora tengo una visión más completa de esta tecnología, de sus capacidades y también de sus limitaciones.

Me siento contento con el trabajo realizado, puesto que los objetivos a alcanzar por el proyecto, han quedado completamente satisfechos. Por supuesto, siempre se

pueden añadir mejoras y actualizaciones sobre la base que hemos desarrollado. Algunas de éstas podrían ser:

- Mejoras en el diseño gráfico y en la adaptación de la pantalla para diferentes dispositivos.
- Traducción a otros idiomas. Fácil, ya que todas las cadenas de texto están referenciadas en el archivo strings.xml.
- Mejoras en la base de datos y aumento de la información almacenada referente a los envíos. Por ejemplo, la inclusión de cuentas de usuarios y un registro de envíos asociado a los mismos. Esto, permitiría también el uso de un sistema de notificaciones *push* que avise a los usuarios de los incidentes acaecidos sobre sus envíos.

Para acabar y como conclusión más importante, quería mostrar mis agradecimientos hacia mi familia y también al exquisito trato recibido por parte de mis tutores, así como a la ayuda que me han ofrecido para poder emprender y abordar este proyecto. Espero con entusiasmo que también el mismo, pueda servir de base a otros alumnos o profesores que quieran llevar a cabo un desarrollo para el sistema Android.

8. Apéndices.

Incluimos en esta sección, a modo de referencia para el lector, todo el código que hemos escrito durante el desarrollo del proyecto. Éste también puede encontrarse en el cd que se adjunta con el presente documento.

8.1. Código de la aplicación Android.

En esta sección de los apéndices encontramos todo el código del proyecto desarrollado con el IDE Eclipse. El código se encuentra libre de errores, sin avisos (*warnings*) y debidamente documentado.

8.1.1. Código Java del paquete model.

Este paquete sólo contiene un fichero de nombre *Envio.java* su contenido es el siguiente:

```
package us.jvalzae.pfc.model;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Envio {

	private long mIdEnvio, mFechaS, mFechaL;
	private double mLat, mLon;
	private float mDistancia;

	public long getmIdEnvio() {
		return mIdEnvio;
	}

	public void setmIdEnvio(long mIdEnvio) {
		this.mIdEnvio = mIdEnvio;
	}

	public long getmFechaS() {
		return mFechaS;
	}

	public void setmFechaS(long mFechaS) {
```

```
		this.mFechaS = mFechaS;
	}

	public long getmFechaL() {
		return mFechaL;
	}

	public void setmFechaL(long mFechaL) {
		this.mFechaL = mFechaL;
	}

	public double getmLat() {
		return mLat;
	}

	public void setmLat(double mLat) {
		this.mLat = mLat;
	}

	public double getmLon() {
		return mLon;
	}

	public void setmLon(double mLon) {
		this.mLon = mLon;
	}

	public float getmDistancia() {
		return mDistancia;
	}

	public void setmDistancia(float mDistancia) {
		this.mDistancia = mDistancia;
	}

	// DEVUELVE UNA CADENA CON LA FECHA ACTUAL:

	public static String fecha(long time) {
		DateFormat format = SimpleDateFormat.getDateTimeInstance();
		Date date = new Date(time);
		String fecha = format.format(date);
		return fecha;
	}

}
```

8.1.2. Código Java del paquete parser.

Al igual que el paquete anterior éste como ya sabemos sólo contiene un fichero de nombre ATrackerXmlSaxParser.java que contiene lo siguiente:

```
package us.jvalzae.pfc.parser;

import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import android.sax.EndTextElementListener;
import android.sax.RootElement;
import android.util.Xml;

import us.jvalzae.pfc.model.Envio;

public class ATrackerXmlSaxParser {

	private URL mUrl;

	public ATrackerXmlSaxParser(String url) {
		try {
			mUrl = new URL(url);
		} catch (MalformedURLException e) {
			throw new RuntimeException(e);
		}
	}

	public Envio parse() {
		final Envio envio = new Envio();
		RootElement root = new RootElement("envio");
		root.getChild("idEnvio").setEndTextElementListener(
				new EndTextElementListener() {
					public void end(String body) {
						envio.setmIdEnvio(Long.valueOf(body));
					}
				});
		root.getChild("latitud").setEndTextElementListener(
				new EndTextElementListener() {
					public void end(String body) {
						envio.setmLat(Double.valueOf(body));
					}
				});
		root.getChild("longitud").setEndTextElementListener(
				new EndTextElementListener() {
					public void end(String body) {
						envio.setmLon(Double.valueOf(body));
					}
				});
		root.getChild("fechaS").setEndTextElementListener(
				new EndTextElementListener() {
					public void end(String body) {
						envio.setmFechaS(Long.valueOf(body));
					}
				});
		root.getChild("fechaL").setEndTextElementListener(
				new EndTextElementListener() {
					public void end(String body) {
						envio.setmFechaL(Long.valueOf(body));
					}
```

```
                        });
                root.getChild("distancia").setEndTextElementListener(
                        new EndTextElementListener() {
                                public void end(String body) {

        envio.setmDistancia(Float.valueOf(body));
                                }
                        });
                try {
                        Xml.parse(this.getInputStream(), Xml.Encoding.UTF_8,
                                root.getContentHandler());
                } catch (Exception e) {
                        throw new RuntimeException(e);
                }
                return envio;
        }

        private InputStream getInputStream() {
                try {
                        return mUrl.openConnection().getInputStream();
                } catch (IOException e) {
                        throw new RuntimeException(e);
                }
        }

}
```

8.1.3. Código Java del paquete view.

El código del presente paquete está compuesto por un total de siete ficheros cuyo contenido es:

8.1.3.1. DialogoActivarInternet.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

//DIÁLOGO DE ACTIVACIÓN DE INTERNET:

public class DialogoActivarInternet extends DialogFragment {

        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
```

```
            // Use the Builder class for convenient dialog construction
            AlertDialog.Builder builder = new
            AlertDialog.Builder(getActivity());
            builder.setTitle(R.string.avisoInternet)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setMessage(R.string.configurarInternet)
                        .setCancelable(false)
                        .setPositiveButton(R.string.si,
                                    new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {
                                Intent intent = new
Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS);
                                                startActivity(intent);
                                                dismiss();
                                            }
                                    });
            builder.setNegativeButton(R.string.no,
                        new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
int whichButton) {}
                        });
            // Create the AlertDialog object and return it
            return builder.create();
      }

}
```

8.1.3.2. DialogoActivarProviders.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

// DIÁLOGO DE ACTIVACIÓN DE PROVEEDORES DE UBICACIÓN:

public class DialogoActivarProviders extends DialogFragment {

      @Override
      public Dialog onCreateDialog(Bundle savedInstanceState) {
            // Use the Builder class for convenient dialog construction
            AlertDialog.Builder builder = new
            AlertDialog.Builder(getActivity());
            builder.setTitle(R.string.avisoUbicacion)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setMessage(R.string.configurarUbicacion)
                        .setCancelable(false)
                        .setPositiveButton(R.string.si,
```

```
                                        new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {
                                                Intent intent = new
Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                                                startActivity(intent);
                                                dismiss();
                                            }
                                        });
            builder.setNegativeButton(R.string.no,
                        new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog,
int whichButton) {}
                        });
            // Create the AlertDialog object and return it
            return builder.create();
      }

}
```

8.1.3.3. DialogoAvisoSalida.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

// NOTIFICA LA SALIDA DE UN ENVÍO:

public class DialogoAvisoSalida extends DialogFragment {

      @Override
      public Dialog onCreateDialog(Bundle savedInstanceState) {
            // Use the Builder class for convenient dialog construction
            AlertDialog.Builder builder = new
            AlertDialog.Builder(getActivity());
            builder.setTitle(R.string.estadoEnvio)
                        .setIcon(android.R.drawable.ic_dialog_info)
                        .setMessage(R.string.avisoSalida)
                        .setNeutralButton(R.string.aceptar,
                                    new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {}
                                    });
            // Create the AlertDialog object and return it
            return builder.create();
      }

}
```

8.1.3.4. DialogoAvisoLlegada.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

// NOTIFICA LA LLEGADA DE UN ENVÍO:

public class DialogoAvisoLlegada extends DialogFragment {

	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {
		// Use the Builder class for convenient dialog construction
		AlertDialog.Builder builder = new
		AlertDialog.Builder(getActivity());
		builder.setTitle(R.string.estadoEnvio)
				.setIcon(android.R.drawable.ic_dialog_info)
				.setMessage(R.string.avisoLlegada)
				.setNeutralButton(R.string.aceptar,
					new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {}
					});
		// Create the AlertDialog object and return it
		return builder.create();
	}
}
```

8.1.3.5. DialogoInfo.java.

```
package us.jvalzae.pfc.view;

import com.google.android.gms.common.GooglePlayServicesUtil;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;
```

```
// DIÁLOGO DE INFORMACIÓN DE LA APLICACIÓN:

public class DialogoInfo extends DialogFragment {

	private TextView mTextoWeb, mTextoGoogle;

	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {
		View customView = getActivity().getLayoutInflater().inflate(
				R.layout.dialogo_info_layout, null);
		mTextoWeb = (TextView) customView.findViewById(R.id.textoWeb);
		mTextoWeb.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				Intent intent = new Intent(Intent.ACTION_VIEW);
		 intent.setData(Uri.parse("http://www.atc.us.es/~jvalzae"));
				startActivity(intent);
			}
		});
		mTextoGoogle = (TextView)
		customView.findViewById(R.id.textoGoogle);
		mTextoGoogle.setText(GooglePlayServicesUtil
				.getOpenSourceSoftwareLicenseInfo(getActivity()));
		// Use the Builder class for convenient dialog construction
		AlertDialog.Builder builder = new
		AlertDialog.Builder(getActivity());
		builder.setTitle(R.string.infoAcercaDe)
				.setIcon(android.R.drawable.ic_dialog_info)
				.setView(customView)
				.setNeutralButton(R.string.aceptar,
					new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {}
					});
		// Create the AlertDialog object and return it
		return builder.create();
	}

}
```

8.1.3.6. DialogoSalir.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

// DIÁLOGO DE SALIDA DE LA APLICACIÓN:

public class DialogoSalir extends DialogFragment {
```

```
	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {
		// Use the Builder class for convenient dialog construction
		AlertDialog.Builder builder = new
		AlertDialog.Builder(getActivity());
		builder.setTitle(R.string.menuSalir)
				.setIcon(android.R.drawable.ic_delete)
				.setMessage(R.string.salidaApp)
				.setPositiveButton(R.string.si,
						new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {
                                                    getActivity().finish();
                                                }
                                    })
                            .setNegativeButton(R.string.no,
                                    new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {}
                                    });
            // Create the AlertDialog object and return it
            return builder.create();
        }

}
```

8.1.3.7. DialogoSLEnvio.java.

```
package us.jvalzae.pfc.view;

import us.jvalzae.pfc.R;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

// DIÁLOGO DE SALIDA/LLEGADA DE UN ENVÍO:

public class DialogoSLEnvio extends DialogFragment {

	private DialogoSLEnvioListener mListener;
	private static String mTitulo, mMensaje;

	public interface DialogoSLEnvioListener {
		public void onDialogoSLEnvio();
	}

	public static DialogoSLEnvio NewInstance(String titulo, String
	mensaje) {
		mTitulo = titulo;
		mMensaje = mensaje;
		return new DialogoSLEnvio();
```

```
	}

	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {
		// Use the Builder class for convenient dialog construction
		AlertDialog.Builder builder = new
		AlertDialog.Builder(getActivity());
		builder.setIcon(android.R.drawable.ic_dialog_alert)
				.setTitle(mTitulo)
				.setMessage(mMensaje)
				.setPositiveButton(R.string.si,
						new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {
						 mListener.onDialogoSLEnvio();
						 }
					})
				.setNegativeButton(R.string.no,
						new DialogInterface.OnClickListener()
{
public void onClick(DialogInterface dialog, int whichButton) {}
					});
		// Create the AlertDialog object and return it
		return builder.create();
	}

	public DialogoSLEnvio setDialogoSLEnvioListener(
			DialogoSLEnvioListener listener) {
		mListener = listener;
		return this;
	}

}
```

8.1.4. Código Java del paquete controller.

El siguiente paquete lo componen un total de cuatro ficheros, todos componentes de Android, son:

8.1.4.1. ActividadPrincipal.java.

```
package us.jvalzae.pfc.controller;

import us.jvalzae.pfc.R;
import us.jvalzae.pfc.R.id;
import us.jvalzae.pfc.view.DialogoInfo;
import us.jvalzae.pfc.view.DialogoSalir;
import android.content.Intent;
import android.net.Uri;
```

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class ActividadPrincipal extends FragmentActivity implements
		OnClickListener {

	private Button mBotonControlarEnvio;
	private Button mBotonSeguirEnvio;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.actividad_principal_layout);
		setupViews();
	}

	// SETUP DE LAS VISTAS:

	private void setupViews() {
		mBotonControlarEnvio = (Button)
		findViewById(id.botonControlarEnvio);
		mBotonControlarEnvio.setOnClickListener(this);
		mBotonSeguirEnvio = (Button) findViewById(id.botonSeguirEnvio);
		mBotonSeguirEnvio.setOnClickListener(this);
	}

	// LISTENERS DE LAS VISTAS:

	@Override
	public void onClick(View v) {
		// BOTÓN CONTROLAR ENVÍO:
		if (v == mBotonControlarEnvio) {
			Intent intent = new Intent(ActividadPrincipal.this,
					ActividadControlarEnvio.class);
			startActivity(intent);
		}
		// BOTÓN SEGUIR ENVÍO:
		else if (v == mBotonSeguirEnvio) {
			Intent intent = new Intent(ActividadPrincipal.this,
					ActividadSeguirEnvio.class);
			startActivity(intent);
		}
	}

	// MENÚ DE OPCIONES:

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is
		//present.
```

```
		getMenuInflater().inflate(R.menu.actividad_principal_menu,
		menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		switch (item.getItemId()) {
		case R.id.menuPrincipalAyuda:
			abrirAyuda();
			return true;
		case R.id.menuPrincipalInfo:
			new DialogoInfo().show(getSupportFragmentManager(),
			"INFO");
			return true;
		case R.id.menuPrincipalSalir:
			new DialogoSalir().show(getSupportFragmentManager(),
			"SALIDA");
			return true;
		default:
			return super.onOptionsItemSelected(item);
		}

	}

	// ABRE LA PÁGINA WEB DE AYUDA DEL PROYECTO:

	private void abrirAyuda() {
		Intent intent = new Intent(Intent.ACTION_VIEW);

	intent.setData(Uri.parse("http://www.atc.us.es/~jvalzae/ayuda.html"));
		startActivity(intent);
	}

	// SE EJECUTA AL PULSAR SOBRE LA TECLA BACK:

	@Override
	public boolean onKeyDown(final int keyCode, final KeyEvent event) {
		if (keyCode == KeyEvent.KEYCODE_BACK)
			new DialogoSalir().show(getSupportFragmentManager(),
			"SALIDA");
		return false;
	}

}
```

8.1.4.2. ActividadControlarEnvio.java.

```
package us.jvalzae.pfc.controller;

import us.jvalzae.pfc.R;
import us.jvalzae.pfc.model.Envio;
import us.jvalzae.pfc.view.DialogoActivarInternet;
import us.jvalzae.pfc.view.DialogoActivarProviders;
```

```
import us.jvalzae.pfc.view.DialogoAvisoLlegada;
import us.jvalzae.pfc.view.DialogoAvisoSalida;
import us.jvalzae.pfc.view.DialogoSLEnvio;
import us.jvalzae.pfc.view.DialogoSLEnvio.DialogoSLEnvioListener;
import android.app.ProgressDialog;
import android.app.Service;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.location.LocationManager;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class ActividadControlarEnvio extends FragmentActivity implements
        OnClickListener {

    private static final String ENVIO_EN_CURSO = "Envío en curso";
    private static final String ID_ENVIO = "Identificador del envío";
    private static final String FECHA_SAL_ENVIO = "Fecha de salida del
    envío";
    private boolean mEnvioEnCurso;
    private TextView mTextoControl;
    private Button mBotonControlador;
    private TableLayout mTLInfo;
    private TableRow mFilaInfoLlegada;
    private TableRow mFilaInfoDistancia;
    private TextView mValorIdEnvio;
    private TextView mValorSalida;
    private TextView mValorLlegada;
    private TextView mValorDistancia;
    private boolean mBound;
    private final Messenger mMessenger = new Messenger(new Handler(
            new IncomingHandler()));
    private Messenger mService = null;
    private ProgressDialog mProgressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.actividad_controlar_envio_layout);
        setupViews();
```

```
	}

	@Override
	protected void onStop() {
		super.onStop();
		SharedPreferences preferences =
		getPreferences(Context.MODE_PRIVATE);
		SharedPreferences.Editor editor = preferences.edit();
		editor.putBoolean(ENVIO_EN_CURSO, mEnvioEnCurso);
		editor.putString(ID_ENVIO, mValorIdEnvio.getText().toString());
		editor.putString(FECHA_SAL_ENVIO,
		mValorSalida.getText().toString());
		editor.commit();
	}

	// SETUP DE LAS VISTAS:

	private void setupViews() {
		mTLInfo = (TableLayout) findViewById(R.id.TLCEInfo);
		mFilaInfoLlegada = (TableRow)
		findViewById(R.id.filaInfoCELlegada);
		mFilaInfoDistancia = (TableRow)
		findViewById(R.id.filaInfoCEDistancia);
		mTextoControl = (TextView)
		findViewById(R.id.textoControlEnvios);
		mBotonControlador = (Button)
		findViewById(R.id.botonControlador);
		mBotonControlador.setOnClickListener(this);
		mValorIdEnvio = (TextView) findViewById(R.id.valorCEIdEnvio);
		mValorSalida = (TextView) findViewById(R.id.valorCESalida);
		mValorLlegada = (TextView) findViewById(R.id.valorCELlegada);
		mValorDistancia = (TextView)
		findViewById(R.id.valorCEDistancia);
		SharedPreferences preferences =
		getPreferences(Context.MODE_PRIVATE);
		mEnvioEnCurso = preferences.getBoolean(ENVIO_EN_CURSO, false);
		if (!mEnvioEnCurso) {
			mTLInfo.setVisibility(View.INVISIBLE);
			mTextoControl.setText(R.string.controlEnviosSalida);

	mBotonControlador.setText(R.string.botonControladorSalida);
			mBotonControlador.setTextColor(getResources().getColor(
					R.color.verde));
		} else {
			bindService(new Intent(ActividadControlarEnvio.this,
					ServicioLocalizacion.class), mConnection,
					Context.BIND_AUTO_CREATE);
			mTextoControl.setText(R.string.controlEnviosLlegada);

	mBotonControlador.setText(R.string.botonControladorLlegada);
			mBotonControlador.setTextColor(getResources()
					.getColor(R.color.rojo));
			mTLInfo.setVisibility(View.VISIBLE);
			mFilaInfoLlegada.setVisibility(View.GONE);
			mFilaInfoDistancia.setVisibility(View.GONE);
```

```
                mValorIdEnvio.setText(preferences.getString(ID_ENVIO,
                null));

        mValorSalida.setText(preferences.getString(FECHA_SAL_ENVIO, null));

              }
        }

        // LISTENER DEL BOTÓN CONTROLADOR:

        @Override
        public void onClick(View v) {
              // PULSAMOS SOBRE SALIDA:
              if (mBotonControlador.getText().equals(
              getResources().getString(R.string.botonControladorSalida))) {
                    if (!checkProviders())
                          new
              DialogoActivarProviders().show(getSupportFragmentManager(),
                                      "PROVIDERS");
                    else if (!checkInternetConexion())
                          new
              DialogoActivarInternet().show(getSupportFragmentManager(),
                                      "INTERNET");
                    else {
                          DialogoSLEnvio
                                      .NewInstance(

              getResources().getString(R.string.salidaEnvio),
                                                  getResources().getString(

        R.string.confirmarSalida)).setDialogoSLEnvioListener( new
        DialogoSLEnvioListener() {
                                    @Override
                    public void onDialogoSLEnvio() {
                                                  salida();
                                                        }

        }).show(getSupportFragmentManager(), "ENVÍOS");
                    }
              }
              // PULSAMOS SOBRE LLEGADA:
              else if (mBotonControlador.getText().equals(
              getResources().getString(R.string.botonControladorLlegada))) {
                    DialogoSLEnvio
                                .NewInstance(
        getResources().getString(R.string.llegadaEnvio),
                    getResources().getString(R.string.confirmarLlegada))
                                .setDialogoSLEnvioListener(new
        DialogoSLEnvioListener() {
                                        @Override
                                        public void onDialogoSLEnvio() {
                                              llegada();
                                        }
                                }).show(getSupportFragmentManager(),
"ENVÍOS");
              }
```

```
	}

	// EJECUTA LA SALIDA DE UN ENVÍO:

	private void salida() {
		mProgressDialog = new ProgressDialog(this);
		mProgressDialog.setMessage(getResources().getText(
				R.string.iniciandoEnvio));
		mProgressDialog.setIndeterminate(true);
		mProgressDialog.setCancelable(false);
		mProgressDialog.show();
		mEnvioEnCurso = true;
		startService(new Intent(ActividadControlarEnvio.this,
				ServicioLocalizacion.class));
		bindService(new Intent(ActividadControlarEnvio.this,
				ServicioLocalizacion.class), mConnection,
				Context.BIND_AUTO_CREATE);
		mTextoControl.setText(R.string.controlEnviosLlegada);
		mBotonControlador.setText(R.string.botonControladorLlegada);

	mBotonControlador.setTextColor(getResources().getColor(R.color.rojo));
		mTLInfo.setVisibility(View.VISIBLE);
		mValorIdEnvio.setVisibility(View.INVISIBLE);
		mValorSalida.setVisibility(View.INVISIBLE);
		mFilaInfoLlegada.setVisibility(View.GONE);
		mFilaInfoDistancia.setVisibility(View.GONE);
	}

	// EJECUTA LA LLEGADA DE UN ENVÍO:

	private void llegada() {
		mProgressDialog = new ProgressDialog(this);
		mProgressDialog.setMessage(getResources().getText(
				R.string.finalizandoEnvio));
		mProgressDialog.setIndeterminate(true);
		mProgressDialog.setCancelable(false);
		mProgressDialog.show();
		if (mBound) {
			unbindService(mConnection);
			mBound = false;
		}
		stopService(new Intent(ActividadControlarEnvio.this,
				ServicioLocalizacion.class));
		mEnvioEnCurso = false;
		mTextoControl.setText(R.string.controlEnviosSalida);
		mBotonControlador.setText(R.string.botonControladorSalida);
	mBotonControlador.setTextColor(getResources().getColor(R.color.verde))
;
	}

	// CONEXIÓN CON EL SERVICIO:

	private ServiceConnection mConnection = new ServiceConnection() {

		public void onServiceConnected(ComponentName className, IBinder
		service) {
```

```
                //This is called when the connection with the service has
                //been established, giving us the object we can use to
                // interact with the service. We are communicating with
                //the service using a Messenger,
                //so here we get a client-side
                // representation of that from the raw IBinder object.
                mService = new Messenger(service);
                mBound = true;
                Message msg = Message.obtain(null,
                            ServicioLocalizacion.MSG_REGISTER_CLIENT);
                msg.replyTo = mMessenger;
                try {
                      mService.send(msg);
                } catch (RemoteException e) {
                      e.printStackTrace();
                }
          }

          public void onServiceDisconnected(ComponentName className) {
                // This is called when the connection with the service has
                //been
                // unexpectedly disconnected -- that is, its process
                //crashed.
                mService = null;
                mBound = false;
          }
    };

    // MANEJADOR:

    private class IncomingHandler implements Handler.Callback {

          @Override
          public boolean handleMessage(Message msg) {
                Bundle datos;
                switch (msg.what) {
                case ServicioLocalizacion.SALIDA:
                      datos = (Bundle) msg.obj;
                      mValorIdEnvio.setText(String.valueOf(datos.getLong(
                                  ServicioLocalizacion.ID_ENVIO, 0)));
          mValorSalida.setText(String.valueOf(Envio.fecha(datos.getLong(
                                  ServicioLocalizacion.FECHA_SALIDA,
                                  0))));
                      mValorIdEnvio.setVisibility(View.VISIBLE);
                      mValorSalida.setVisibility(View.VISIBLE);
                      mProgressDialog.dismiss();
                      new
                DialogoAvisoSalida().show(getSupportFragmentManager(),
                                  "ENVÍOS");
                      return true;
                case ServicioLocalizacion.LLEGADA:
                      datos = (Bundle) msg.obj;
                      mValorLlegada.setText(Envio.fecha(datos.getLong(
                                  ServicioLocalizacion.FECHA_LLEGADA,
                                  0)));
                mValorDistancia.setText(String.valueOf(datos.getFloat(
```

```
                                        ServicioLocalizacion.DISTANCIA, 0) /
                                1000) + " Km.");
                                mFilaInfoLlegada.setVisibility(View.VISIBLE);
                                mFilaInfoDistancia.setVisibility(View.VISIBLE);
                                mProgressDialog.dismiss();
                                new
                        DialogoAvisoLlegada().show(getSupportFragmentManager(),
                                        "ENVÍOS");
                                return true;
                        default:
                                return false;
                        }
                }
        }

        // COMPRUEBA QUE ESTÉN ACTIVADOS LOS PROVEEDORES DE UBICACIÓN:

        private boolean checkProviders() {
                boolean provOk = true;
                LocationManager locManager = (LocationManager)
                getSystemService(Service.LOCATION_SERVICE);
                if (!locManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
                                || !locManager
                .isProviderEnabled(LocationManager.NETWORK_PROVIDER))
                        provOk = false;
                return provOk;
        }

        // COMPRUEBA LA CONEXIÓN A INTERNET:

        private boolean checkInternetConexion() {
                boolean internetOk = false;
                ConnectivityManager conManager = (ConnectivityManager)
                getSystemService(CONNECTIVITY_SERVICE);
                if (conManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE)
                                .getState() == NetworkInfo.State.CONNECTED
                                ||
                conManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
                                        .getState() ==
                NetworkInfo.State.CONNECTED)
                        internetOk = true;
                return internetOk;
        }

}
```

8.1.4.3. ServicioLocalizacion.java.

```
package us.jvalzae.pfc.controller;

import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
```

```
import us.jvalzae.pfc.R;
import us.jvalzae.pfc.model.Envio;
import us.jvalzae.pfc.parser.ATrackerXmlSaxParser;
import android.app.Notification;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.TaskStackBuilder;
import android.view.Gravity;
import android.widget.Toast;

public class ServicioLocalizacion extends Service implements LocationListener
{

	public static final String ID_ENVIO = "Identificador del envío";
	public static final String FECHA_SALIDA = "Fecha de salida del envío";
	public static final String FECHA_LLEGADA = "Fecha de llegada del
	envío";
	public static final String DISTANCIA = "Distancia recorrida";
	public static final int MSG_REGISTER_CLIENT = 1;
	public static final int SALIDA = 1;
	public static final int LLEGADA = 2;
	private static final float DISTANCIA_ACTUALIZACION = 20;
	private static final float DISTANCIA_ACTUALIZACION_BD = 10;
	private static final int LOCATION_EXPIRATION_TIME = 1000 * 60 * 2;
	private static final int LOCATION_MAX_ACCURACY_DELTA_TOLERANCE = 200;
	private static final int ID_NOT_SERVICIOLOC = 1;
	private LocationManager mLocationManager;
	private Location mLocalizacionSalida;
	private Location mLocalizacion;
	private long mIdEnvio;
	private final Messenger mMessenger = new Messenger(new Handler(
			new IncomingHandler()));
	private Messenger mCliente = null;

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {
		iniciarServicio();
		return super.onStartCommand(intent, flags, startId);
	}

	@Override
	public IBinder onBind(Intent intent) {
		return mMessenger.getBinder();
	}
```

```
@Override
public void onDestroy() {
	finalizarServicio();
	super.onDestroy();
}

// SE EJECUTA AL INICIARSE EL SERVICIO:

private void iniciarServicio() {
	startForeground(ID_NOT_SERVICIOLOC, abrirNotificacion());
	mLocalizacionSalida = null;
	mLocalizacion = null;
	mLocationManager = (LocationManager)
	getSystemService(LOCATION_SERVICE);
	mLocationManager.requestLocationUpdates(
			LocationManager.NETWORK_PROVIDER,
			LOCATION_EXPIRATION_TIME,
			DISTANCIA_ACTUALIZACION, this);
mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
			LOCATION_EXPIRATION_TIME, DISTANCIA_ACTUALIZACION,
this);
}

// SE EJECUTA AL FINALIZAR EL SERVICIO:

private void finalizarServicio() {
	finalizaEnvioBD();
	mLocationManager.removeUpdates(this);
	stopForeground(true);
}

// LOCATION LISTENERS:

@Override
public void onLocationChanged(Location location) {
	if (location != null) {
		if (isBetterLocation(location, mLocalizacion)) {
			if (mLocalizacionSalida == null) {
				mLocalizacionSalida = location;
				mLocalizacion = location;
				iniciaEnvioBD();
			} else if (mLocalizacion.distanceTo(location) >=
				DISTANCIA_ACTUALIZACION_BD) {
				mLocalizacion = location;
				actualizaEnvioBD();
			} else
				mLocalizacion = location;
		}
	}
}

@Override
public void onProviderDisabled(String provider) {
	Toast toast = Toast.makeText(ServicioLocalizacion.this, provider
			+ " "
```

```
                    + getResources().getString(R.string.inhabilitado),
                    Toast.LENGTH_LONG);
            toast.setGravity(Gravity.CENTER, 0, 0);
            toast.show();
    }

    @Override
    public void onProviderEnabled(String provider) {
            Toast toast = Toast.makeText(ServicioLocalizacion.this, provider
                    + " "
                    + getResources().getString(R.string.habilitado),
                    Toast.LENGTH_LONG);
            toast.setGravity(Gravity.CENTER, 0, 0);
            toast.show();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle
    extras) {}

    // CREA UN NUEVO ENVÍO EN LA BD REMOTA:

    private void iniciaEnvioBD() {
            TratamientoInicioEnvioBD tarea = new TratamientoInicioEnvioBD();
            tarea.execute();
    }

    // TAREA ASÍNCRONA QUE INICIA EL ENVÍO EN LA BD REMOTA:

    private class TratamientoInicioEnvioBD extends
                    AsyncTask<String, Integer, Boolean> {

            @Override
            protected Boolean doInBackground(String... params) {
                    ATrackerXmlSaxParser parser = new ATrackerXmlSaxParser(
                            "http://www.atc.us.es/~jvalzae/iniciar_envio?lat="
                                            +
                    String.valueOf(mLocalizacionSalida.getLatitude())
                                            + "&lon="
                                            +
                    String.valueOf(mLocalizacionSalida.getLongitude())
                                            + "&alt="
                                            +
                    String.valueOf(mLocalizacionSalida.getAltitude())
                                            + "&fechaS="
                                            +
                    String.valueOf(mLocalizacionSalida.getTime()));
                    Envio envio = parser.parse();
                    mIdEnvio = envio.getmIdEnvio();
                    return true;
            }

            // COMUNICAMOS EL ID. Y LA FECHA DE SALIDA DEL ENVÍO:
            @Override
            protected void onPostExecute(Boolean result) {
                    Bundle datos = new Bundle();
```

```
                datos.putLong(ID_ENVIO, mIdEnvio);
                datos.putLong(FECHA_SALIDA,
                mLocalizacionSalida.getTime());
                try {
                      mCliente.send(Message.obtain(null, SALIDA, datos));
                } catch (RemoteException e) {
                      e.printStackTrace();
                }
          }
    }

    // ACTUALIZA LOS DATOS DEL ENVÍO EN LA BD REMOTA:

    private void actualizaEnvioBD() {
          TratamientoActualizaEnvioBD tarea = new
          TratamientoActualizaEnvioBD();
          tarea.execute();
    }

    // TAREA ASÍNCRONA QUE ACTUALIZA EL ENVÍO EN LA BD REMOTA:

    private class TratamientoActualizaEnvioBD extends
                AsyncTask<String, Integer, Boolean> {

          @Override
          protected Boolean doInBackground(String... params) {
                String url =
                "http://www.atc.us.es/~jvalzae/actualizar_envio?id="
                            + String.valueOf(mIdEnvio) + "&lat="
                            + String.valueOf(mLocalizacion.getLatitude())
                            + "&lon="
                            +
                String.valueOf(mLocalizacion.getLongitude()) + "&alt="
                            +
                String.valueOf(mLocalizacion.getAltitude());
                HttpClient httpclient = new DefaultHttpClient();
                HttpPost httppost = new HttpPost(url);
                try {
                      httpclient.execute(httppost);
                } catch (Exception e) {
                      throw new RuntimeException(e);
                }
                return true;
          }
    }

    // FINALIZA UN ENVÍO EN LA BD REMOTA:

    private void finalizaEnvioBD() {
          TratamientoFinalizaEnvioBD tarea = new
          TratamientoFinalizaEnvioBD();
          tarea.execute();
    }
```

```
// TAREA ASÍNCRONA QUE FINALIZA EL ENVÍO EN LA BD REMOTA:

private class TratamientoFinalizaEnvioBD extends
		AsyncTask<String, Integer, Boolean> {

	@Override
	protected Boolean doInBackground(String... params) {
		String url =
		"http://www.atc.us.es/~jvalzae/finalizar_envio?id="
				+ String.valueOf(mIdEnvio)
				+ "&lat="
				+ String.valueOf(mLocalizacion.getLatitude())
				+ "&lon="
				+
				String.valueOf(mLocalizacion.getLongitude())
				+ "&alt="
				+ String.valueOf(mLocalizacion.getAltitude())
				+ "&fechaL="
				+ String.valueOf(mLocalizacion.getTime())
				+ "&distancia="
				+ String.valueOf(mLocalizacionSalida
						.distanceTo(mLocalizacion));
		HttpClient httpclient = new DefaultHttpClient();
		HttpPost httppost = new HttpPost(url);
		try {
			httpclient.execute(httppost);
		} catch (Exception e) {
			throw new RuntimeException(e);
		}
		return true;
	}

	// COMUNICAMOS LA FECHA DE LLEGADA Y LA DISTANCIA RECORRIDA:
	@Override
	protected void onPostExecute(Boolean result) {
		Bundle datos = new Bundle();
		datos.putLong(FECHA_LLEGADA, mLocalizacion.getTime());
		datos.putFloat(DISTANCIA,
			mLocalizacionSalida.distanceTo(mLocalizacion));
		try {
			mCliente.send(Message.obtain(null, LLEGADA,
			datos));
		} catch (RemoteException e) {
			e.printStackTrace();
		}
	}
}

// MANEJADOR:

private class IncomingHandler implements Handler.Callback {

	@Override
	public boolean handleMessage(Message msg) {
		switch (msg.what) {
		case MSG_REGISTER_CLIENT:
```

```
                        mCliente = msg.replyTo;
                        return true;
                        default:
                        return false;
                }
        }

}

// NOTIFICACIÓN DE USO DEL SERVICIO:

private Notification abrirNotificacion() {
        Notification notif;
        NotificationCompat.Builder mBuilder = new
        NotificationCompat.Builder(
                        this)
                        .setSmallIcon(R.drawable.ic_launcher)
                        .setContentTitle(
                        getResources().getText(R.string.notificacionEnvio))
.setContentText(getResources().getText(R.string.botonControlarEnvio));
        //Creates an explicit intent for an Activity in your application
        Intent resultIntent = new Intent(this,
        ActividadControlarEnvio.class);
        //The stack builder object will contain an artificial back stack
        //for the started Activity.
        //This ensures that navigating backward from the Activity leads
        //out of your application to the Home screen.
        TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
        //Adds the back stack for the Intent (but not the Intent itself)
        stackBuilder.addParentStack(ActividadPrincipal.class);
        //Adds the Intent that starts the Activity to the top of the
        //stack
        stackBuilder.addNextIntent(resultIntent);
        PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(0,
                        PendingIntent.FLAG_UPDATE_CURRENT);
        mBuilder.setContentIntent(resultPendingIntent);
        notif = mBuilder.build();
        notif.defaults |= Notification.DEFAULT_SOUND;
        return notif;
}

// DETERMINA QUE LOCALIZACIÓN ES LA MEJOR:

/**
 * Determines whether one Location reading is better than the current
 * Location fix
 *
 * @param location
 *            The new Location that you want to evaluate
 * @param currentBestLocation
 * The current Location fix, to which you want to compare the new one
 */

protected boolean isBetterLocation(Location location,
                Location currentBestLocation) {
```

```
            if (currentBestLocation == null) {
                  // A new location is always better than no location
                  return true;
            }
            // Check whether the new location fix is newer or older
            long timeDelta = location.getTime() -
            currentBestLocation.getTime();
            boolean isSignificantlyNewer = timeDelta >
            LOCATION_EXPIRATION_TIME;
            boolean isSignificantlyOlder = timeDelta < -
            LOCATION_EXPIRATION_TIME;
            boolean isNewer = timeDelta > 0;
            //If it's been more than two minutes since the current location,
            // use the new location because the user has likely moved
            if (isSignificantlyNewer) {
                  return true;
                  // If the new location is more than two minutes older, it
                  // must be worse
            } else if (isSignificantlyOlder) {
                  return false;
            }
            // Check whether the new location fix is more or less accurate
            int accuracyDelta = (int) (location.getAccuracy() -
            currentBestLocation.getAccuracy());
            boolean isLessAccurate = accuracyDelta > 0;
            boolean isMoreAccurate = accuracyDelta < 0;
            boolean isSignificantlyLessAccurate = accuracyDelta >
            LOCATION_MAX_ACCURACY_DELTA_TOLERANCE;
            // Check if the old and new location are from the same provider
            boolean isFromSameProvider =
            isSameProvider(location.getProvider(),
                        currentBestLocation.getProvider());

            // Determine location quality using a combination of timeliness
            // and accuracy
            if (isMoreAccurate) {
                  return true;
            } else if (isNewer && !isLessAccurate) {
                  return true;
            } else if (isNewer && !isSignificantlyLessAccurate
                        && isFromSameProvider) {
                  return true;
            }
            return false;
      }
      /** Checks whether two providers are the same */
      private boolean isSameProvider(String provider1, String provider2) {
            if (provider1 == null) {
                  return provider2 == null;
            }
            return provider1.equals(provider2);
      }

}
```

8.1.4.4. ActividadSeguirEnvio.java.

```
package us.jvalzae.pfc.controller;

import us.jvalzae.pfc.R;
import us.jvalzae.pfc.model.Envio;
import us.jvalzae.pfc.parser.ATrackerXmlSaxParser;
import us.jvalzae.pfc.view.DialogoActivarInternet;
import android.app.ProgressDialog;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.RelativeLayout;
import android.widget.TableRow;
import android.widget.TextView;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.InfoWindowAdapter;
import com.google.android.gms.maps.GoogleMap.OnMarkerClickListener;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.CameraPosition;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

public class ActividadSeguirEnvio extends FragmentActivity implements
        OnClickListener {

    private Envio mEnvio;
    private EditText mCampoIdEnvio;
    private ImageButton mBotonBuscar;
    private TextView mTextoNoId;
    private RelativeLayout mRLMapa;
    private GoogleMap mMap;
    private ProgressDialog mProgressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.actividad_seguir_envio_layout);
        setupViews();
    }

    // SETUP DE LAS VISTAS:

    private void setupViews() {
        mCampoIdEnvio = (EditText) findViewById(R.id.campoSEIdEnvio);
        mBotonBuscar = (ImageButton) findViewById(R.id.botonBuscar);
```

```
		mBotonBuscar.setOnClickListener(this);
		mTextoNoId = (TextView) findViewById(R.id.textoSENoId);
		mTextoNoId.setVisibility(View.INVISIBLE);
		mRLMapa = (RelativeLayout) findViewById(R.id.RLSEMapa);
		mRLMapa.setVisibility(View.INVISIBLE);
		mMap = ((SupportMapFragment) getSupportFragmentManager()
				.findFragmentById(R.id.mapa)).getMap();
	}

	// LISTENER DEL BOTÓN BUSCAR:

	@Override
	public void onClick(View v) {
		if (!checkInternetConexion())
			new
	DialogoActivarInternet().show(getSupportFragmentManager(),
					"INTERNET");
		else {
			// SI SE HA INTRODUCIDO UN ID. DE ENVÍO:
			if (!mCampoIdEnvio.getText().toString().equals("")) {
				mProgressDialog = new ProgressDialog(this);
				mProgressDialog.setMessage(getResources().getText(
						R.string.identificandoEnvio));
				mProgressDialog.setIndeterminate(true);
				mProgressDialog.setCancelable(false);
				mProgressDialog.show();
				TratamientoXml tarea = new TratamientoXml();
				tarea.execute();
			}
			// SI NO SE INTRODUCE NADA:
			else
				;
		}
	}

	// TAREA ASÍNCRONA QUE SE CONECTA A INTERNET Y PARSEA XML:

	private class TratamientoXml extends AsyncTask<String, Integer,
Boolean> {

		@Override
		protected Boolean doInBackground(String... params) {
			try {
				ATrackerXmlSaxParser parser = new
		ATrackerXmlSaxParser(
				"http://www.atc.us.es/~jvalzae/consultar_envio?id="
						+
		mCampoIdEnvio.getText().toString());
				mEnvio = parser.parse();
			} catch (Exception e) {
				return false;
			}
			return true;
		}
```

```
		@Override
		protected void onPostExecute(Boolean result) {
			mProgressDialog.dismiss();
			if (result)
				setupMap();
			else {
				mRLMapa.setVisibility(View.INVISIBLE);
				mTextoNoId.setVisibility(View.VISIBLE);
			}
		}
	}

// PINTA LA POSICIÓN EN EL MAPA:

	private void setupMap() {
		mTextoNoId.setVisibility(View.GONE);
		mRLMapa.setVisibility(View.VISIBLE);
		mMap.clear();
		// CREAMOS LA VENTANA DE INFORMACIÓN DEL ENVÍO:
		mMap.setInfoWindowAdapter(new InfoWindowAdapter() {

			@Override
			public View getInfoWindow(Marker arg0) {
				return null;
			}

			@Override
			public View getInfoContents(Marker arg0) {
				View customView = getLayoutInflater().inflate(
						R.layout.ventana_info_mapa_layout,
				null);
				TableRow filaInfoLlegada = (TableRow) customView
						.findViewById(R.id.filaInfoLlegada);
				TableRow filaInfoDistancia = (TableRow) customView
						.findViewById(R.id.filaInfoDistancia);
				TextView textoEstado = (TextView) customView
					.findViewById(R.id.textoInfoEstadoEnvio);
				TextView valorIdEnvio = (TextView) customView
						.findViewById(R.id.valorIdEnvio);
				TextView valorSalida = (TextView) customView
						.findViewById(R.id.valorSalida);
				TextView valorLlegada = (TextView) customView
						.findViewById(R.id.valorLlegada);
				TextView valorDistancia = (TextView) customView
						.findViewById(R.id.valorDistancia);
				String estado;
				if (mEnvio.getmFechaL() > 0) {
					estado =
				getResources().getString(R.string.envioEnDestino);
			valorLlegada.setText(Envio.fecha(mEnvio.getmFechaL()));
					String distancia =
				String.valueOf((mEnvio.getmDistancia() / 1000))
							+ " Km.";
					valorDistancia.setText(distancia);
				} else {
```

```
                        estado =
                    getResources().getString(R.string.envioEnCamino);
                        filaInfoLlegada.setVisibility(View.GONE);
                        filaInfoDistancia.setVisibility(View.GONE);
                    }
                    textoEstado.setText(estado);
            valorIdEnvio.setText(String.valueOf(mEnvio.getmIdEnvio()));
                valorSalida.setText(Envio.fecha(mEnvio.getmFechaS()));
                    return customView;
                }
            });
            // AÑADIMOS MARCADOR AL MAPA:
            mMap.setOnMarkerClickListener(new OnMarkerClickListener() {

                @Override
                public boolean onMarkerClick(Marker marker) {
                    mMap.animateCamera(CameraUpdateFactory
                    .newCameraPosition(CameraPosition.fromLatLngZoom(new
            LatLng(mEnvio.getmLat(), mEnvio.getmLon()), (int) (0.8 *
            mMap.getMaxZoomLevel())));
                    marker.showInfoWindow();
                    return true;
                }
            });
            mMap.addMarker(new MarkerOptions().position(
                    new LatLng(mEnvio.getmLat(),
mEnvio.getmLon())).icon(
            BitmapDescriptorFactory.fromResource(R.drawable.ic_launcher)));
mMap.animateCamera(CameraUpdateFactory.newCameraPosition(
CameraPosition.fromLatLngZoom(new LatLng(mEnvio.getmLat(), mEnvio.getmLon()),
                                    mMap.getMinZoomLevel())));
        }

        // COMPRUEBA LA CONEXIÓN A INTERNET:

        private boolean checkInternetConexion() {
            boolean internetOk = false;
            ConnectivityManager conManager = (ConnectivityManager)
            getSystemService(CONNECTIVITY_SERVICE);
            if (conManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE)
                    .getState() == NetworkInfo.State.CONNECTED
            || conManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
            .getState() == NetworkInfo.State.CONNECTED)
                internetOk = true;
            return internetOk;
        }

}
```

8.1.5. Código XML de los recursos de la aplicación.

En este apartado se incluye todo el código XML que hemos desarrollado y que se encuentra dentro de la carpeta de recursos (/res) del proyecto.

8.1.5.1. borde.xml.

Este fichero está alojado dentro de la carpeta /res/drawable, su contenido es:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <solid android:color="@color/blanco" />

    <stroke
        android:width="4dp"
        android:color="@color/naranja" />

</shape>
```

8.1.5.2. fondo.xml.

También está dentro de /res/drawable, éste es el código:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >

    <gradient
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:angle="315"
        android:endColor="@color/azul"
        android:startColor="@color/blanco" />

</shape>
```

8.1.5.3. actividad_principal_layout.xml.

Está ubicado dentro de la carpeta /res/layout, su contenido:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LLPrincipal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo"
```

```
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ActividadPrincipal" >

    <TextView
        android:id="@+id/textoInicio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="@string/Inicio"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textStyle="bold|italic" />

    <LinearLayout
        android:id="@+id/LLPControlarEnvio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:orientation="vertical" >

        <Button
            android:id="@+id/botonControlarEnvio"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="@string/botonControlarEnvio"
            android:textStyle="bold" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/LLPSeguirEnvio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:orientation="vertical" >

        <Button
            android:id="@+id/botonSeguirEnvio"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="@string/botonSeguirEnvio"
            android:textStyle="bold" />
    </LinearLayout>

</LinearLayout>
```

8.1.5.4. actividad_controlar_envio_layout.xml.

Ubicado dentro de /res/layout contiene lo siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LLControlarEnvio"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ActividadControlarEnvio" >

    <TextView
        android:id="@+id/textoControlEnvios"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textIsSelectable="false"
        android:textStyle="bold|italic" />

    <LinearLayout
        android:id="@+id/LLCEBotonControlador"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:orientation="vertical" >

        <Button
            android:id="@+id/botonControlador"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:textStyle="bold" />
    </LinearLayout>

    <TableLayout
        android:id="@+id/TLCEInfo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/borde"
        android:padding="10dp"
        android:shrinkColumns="0,1"
        android:stretchColumns="0,1" >

        <TableRow
            android:id="@+id/filaInfoCEIdEnvio"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/textoCEIdEnvio"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/idEnvio"
```

```
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/azul"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorCEIdEnvio"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textIsSelectable="true" />
    </TableRow>

    <TableRow
        android:id="@+id/filaInfoCESalida"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoCESalida"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/salida"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/verde"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorCESalida"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textIsSelectable="false" />
    </TableRow>

    <TableRow
        android:id="@+id/filaInfoCELlegada"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoCELlegada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/llegada"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/rojo"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorCELlegada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textIsSelectable="false" />
    </TableRow>
```

```
        <TableRow
            android:id="@+id/filaInfoCEDistancia"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="2" >

            <TextView
                android:id="@+id/textoCEDistancia"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/distancia"
                android:textAppearance="?android:attr/textAppearanceSmall"
                android:textColor="@color/lila"
                android:textStyle="bold" />

            <TextView
                android:id="@+id/valorCEDistancia"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:textIsSelectable="false" />
        </TableRow>
    </TableLayout>

</LinearLayout>
```

8.1.5.5. actividad_seguir_envio_layout.xml.

Otro recurso más dentro de /res/layout éste es su código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LLSeguirEnvio"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo"
    android:orientation="vertical"
    tools:context=".ActividadSeguirEnvio" >

    <TextView
        android:id="@+id/textoSEintroducirID"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5dp"
        android:text="@string/introduzcaIdEnvio"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textStyle="bold" />

    <LinearLayout
        android:id="@+id/LLSEBusqueda"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content" >

        <EditText
            android:id="@+id/campoSEIdEnvio"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="numberDecimal" >

            <requestFocus />
        </EditText>

        <ImageButton
            android:id="@+id/botonBuscar"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:contentDescription="@string/buscar"
            android:src="@android:drawable/ic_menu_search" />
    </LinearLayout>

    <TextView
        android:id="@+id/textoSENoId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:text="@string/noexisteEnvio"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="@color/rojo"
        android:textStyle="bold|italic" />

    <RelativeLayout
        android:id="@+id/RLSEMapa"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2" >

        <fragment
            android:id="@+id/mapa"
            android:name="com.google.android.gms.maps.SupportMapFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </RelativeLayout>

</LinearLayout>
```

8.1.5.6. dialogo_info_layout.xml.

Diseño contenido en /res/layout cuyo código es:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/SVDialogoInfo"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:id="@+id/LLDialogoInfo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="vertical"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin" >

        <ImageView
            android:id="@+id/logous"
            android:layout_width="116dp"
            android:layout_height="116dp"
            android:contentDescription="@string/logous"
            android:src="@drawable/logo_us" />

        <ImageView
            android:id="@+id/logoetsii"
            android:layout_width="123dp"
            android:layout_height="71dp"
            android:layout_margin="5dp"
            android:contentDescription="@string/logoetsii"
            android:src="@drawable/logo_etsii" />

        <TextView
            android:id="@+id/textoDescripcionApp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:gravity="center"
            android:text="@string/descripcionApp"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/textoRealizadoPor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:gravity="center"
            android:text="@string/realizadoPor"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/textoDirigidoPor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_margin="5dp"
            android:gravity="center"
            android:text="@string/dirigidoPor"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textStyle="bold" />

        <ImageView
            android:id="@+id/logoatc"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:contentDescription="@string/logoatc"
            android:src="@drawable/logo_atc" />

        <TextView
            android:id="@+id/textoWeb"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:text="@string/textoWeb"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/azul" />

        <TextView
            android:id="@+id/textoGoogle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textIsSelectable="false" />
    </LinearLayout>

</ScrollView>
```

8.1.5.7. ventana_info_mapa_layout.xml.

Último recurso contenido dentro de /res/layout, contiene el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TLInfoWindow"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="0,1"
    android:stretchColumns="0,1" >

    <TableRow
        android:id="@+id/filaInfoEstadoEnvio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" >
```

```
        <TextView
            android:id="@+id/textoInfoEstadoEnvio"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_span="2"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textIsSelectable="false"
            android:textStyle="bold" />
    </TableRow>

    <TableRow
        android:id="@+id/filaInfoIdEnvio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoIdEnvio"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/idEnvio"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/azul"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorIdEnvio"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textIsSelectable="false" />
    </TableRow>

    <TableRow
        android:id="@+id/filaInfoSalida"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoSalida"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/salida"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/verde"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorSalida"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textIsSelectable="false" />
    </TableRow>
```

```
    <TableRow
        android:id="@+id/filaInfoLlegada"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoLlegada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/llegada"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/rojo"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorLlegada"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textIsSelectable="false" />
    </TableRow>

    <TableRow
        android:id="@+id/filaInfoDistancia"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textoDistancia"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/distancia"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textColor="@color/lila"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/valorDistancia"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textIsSelectable="false" />
    </TableRow>

</TableLayout>
```

8.1.5.8. actividad_principal_menu.xml.

Diseño de menú ubicado en /res/menú y que contiene lo siguiente:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
```

```
    <item
        android:id="@+id/menuPrincipalAyuda"
        android:icon="@android:drawable/ic_menu_help"
        android:orderInCategory="100"
        android:showAsAction="ifRoom|withText"
        android:title="@string/menuAyuda"
        android:titleCondensed="Ayuda">
    </item>
    <item
        android:id="@+id/menuPrincipalInfo"
        android:icon="@android:drawable/ic_menu_info_details"
        android:orderInCategory="200"
        android:showAsAction="ifRoom|withText"
        android:title="@string/menuInfo"
        android:titleCondensed="Info">
    </item>
    <item
        android:id="@+id/menuPrincipalSalir"
        android:icon="@android:drawable/ic_menu_close_clear_cancel"
        android:orderInCategory="300"
        android:showAsAction="ifRoom|withText"
        android:title="@string/menuSalir"
        android:titleCondensed="Salir">
    </item>

</menu>
```

8.1.5.9. colors.xml.

Colores empleados por la aplicación, el archivo se encuentra dentro de /res/values y éstos (los colores) son:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <color name="azul">#0099CC</color>
    <color name="blanco">#FFFFFF</color>
    <color name="rojo">#FF4444</color>
    <color name="verde">#669900</color>
    <color name="naranja">#FF8800</color>
    <color name="lila">#9933CC</color>

</resources>
```

8.1.5.10. strings.xml.

Éste fichero que se encuentra dentro de la carpeta /res/values contiene la definición de todas las cadenas de texto de nuestra aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">ATracker</string>
    <string name="Inicio">ATracker le permite controlar y seguir los envíos
      de una empresa de transporte\n¿Qué desea hacer?</string>
    <string name="botonControlarEnvio">CONTROLAR ENVÍO</string>
    <string name="botonSeguirEnvio">SEGUIR ENVÍO</string>
    <string name="title_activity_controlar_envio">ATracker/Controlar
      envío</string>
    <string name="title_activity_seguir_envio">ATracker/Seguir envío</string>
    <string name="menuSalir">Salir de ATracker</string>
    <string name="si">Si</string>
    <string name="no">No</string>
    <string name="salidaApp">¿Desea salir de la aplicación?</string>
    <string name="controlEnviosSalida">Utilice el botón para controlar su
      envío. Pulse sobre SALIDA para comenzar.</string>
    <string name="idEnvio">ID. ENVÍO:</string>
    <string name="salida">SALIDA:</string>
    <string name="llegada">LLEGADA:</string>
    <string name="distancia">DISTANCIA:</string>
    <string name="menuAyuda">Ayuda</string>
    <string name="menuInfo">Información</string>
    <string name="introduzcaIdEnvio">INTRODUZCA SU ID. DE ENVÍO:</string>
    <string name="buscar">Buscar</string>
    <string name="noexisteEnvio">No existe ningún envío con ese
      identificador</string>
    <string name="envioEnCamino">El envío está de camino</string>
    <string name="envioEnDestino">El envío se ha completado</string>
    <string name="salidaEnvio">Salida de un envío</string>
    <string name="confirmarSalida">¿Desea iniciar un nuevo envío?</string>
    <string name="botonControladorSalida">SALIDA</string>
    <string name="botonControladorLlegada">LLEGADA</string>
    <string name="descripcionApp">APLICACIÓN DE SEGUIMIENTO DE ENVÍOS DE UNA
      AGENCIA DE TRANSPORTE PARA ANDROID</string>
    <string name="realizadoPor">Realizado por\nJOAQUÍN VALONERO
      ZAERA</string>
    <string name="dirigidoPor">Dirigido por\nÁNGEL FRANCISCO JIMÉNEZ
      FERNÁNDEZ\nMANUEL JESÚS DOMÍNGUEZ MORALES</string>
    <string name="textoWeb">www.atc.us.es/~jvalzae</string>
    <string name="logous">logous</string>
    <string name="logoetsii">logoetsii</string>
    <string name="logoatc">logoatc</string>
    <string name="aceptar">Aceptar</string>
    <string name="avisoUbicacion">Proveedores de ubicación
      inhabilitados</string>
```

```
    <string name="configurarUbicacion">ATracker requiere tener habilitados
      ambos proovedores de ubicación (Redes y GPS) para su
      funcionamiento.\n¿Desea acceder a las opciones de configuración de su
      ubicación?</string>
    <string name="llegadaEnvio">Llegada de un envío</string>
    <string name="confirmarLlegada">¿Desea dar por finalizado el
      envío?</string>
    <string name="avisoSalida">ENVÍO INICIADO.\n\nRECUERDE:\n· Mantener su
      terminal cerca del pedido.\n· NO inhabilite las opciones de acceso a
      su ubicación ni su conexión a Internet.\n· NO borre la RAM mientras el
      envío esté en curso.</string>
    <string name="avisoLlegada">ENVÍO FINALIZADO.\n\nPulse sobre SALIDA para
      comenzar un nuevo envío.</string>
    <string name="notificacionEnvio">ATRACKER: ENVÍO EN CURSO</string>
    <string name="estadoEnvio">Estado de su envío</string>
    <string name="infoAcercaDe">Acerca de ATracker</string>
    <string name="controlEnviosLlegada">ENVÍO EN CURSO. Pulse sobre LLEGADA
      para finalizar.</string>
    <string name="habilitado">habilitado</string>
    <string name="inhabilitado">inhabilitado</string>
    <string name="avisoInternet">Sin conexión a Internet</string>
    <string name="configurarInternet">ATracker requiere de conexión a
      Internet para su funcionamiento.\n¿Desea acceder a las opciones de
      configuración de su conexión?</string>
    <string name="identificandoEnvio">Identificando&#8230;</string>
    <string name="iniciandoEnvio">Iniciando envío&#8230;</string>
    <string name="finalizandoEnvio">Finalizando envío&#8230;</string>

</resources>
```

8.1.6. AndroidManifest.xml.

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <permission
        android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" >
    </permission>

    <uses-permission android:name="us.jvalzae.pfc.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
```

```
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="us.jvalzae.pfc.controller.ActividadPrincipal"
            android:label="@string/app_name"
            android:screenOrientation="nosensor" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="us.jvalzae.pfc.controller.ActividadControlarEnvio"
            android:label="@string/title_activity_controlar_envio"
            android:launchMode="singleTop"
            android:screenOrientation="nosensor" >
        </activity>
        <activity
            android:name="us.jvalzae.pfc.controller.ActividadSeguirEnvio"
            android:label="@string/title_activity_seguir_envio"
            android:screenOrientation="nosensor" >
        </activity>

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyA3dkcLUDtpCs0lsAvA2ijf9NgA02k9g84" />

        <service
android:name="us.jvalzae.pfc.controller.ServicioLocalizacion" >
        </service>
    </application>

</manifest>
```

8.2. Código alojado en el servidor.

En esta sección incluímos todo el código HTML y PHP que hemos subido al servidor del proyecto. Lo conforman un total de siete ficheros:

8.2.1. index.html.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>PFC - ATracker</title>

<link href="./images/favicon.ico" rel="shortcut icon" type="image/x-icon"/>

</head>

<body style="text-align:center">

<p><img src="./images/logoUS.jpg"/></p>

<p><img src="./images/logoETSII.png" /></p>

<p>INGENIERÍA EN INFORMÁTICA</p>

<p>ATracker</p>

<p><img src="./images/ATracker.png" /></p>

<p>APLICACIÓN DE SEGUIMIENTO DE ENVÍOS DE UNA AGENCIA DE TRANSPORTE PARA
ANDROID</p>

<p>Realizado por<br/>JOAQUÍN VALONERO ZAERA</p>

<p>Dirigido por<br/>ÁNGEL FRANCISCO JIMÉNEZ FERNÁNDEZ<br/>MANUEL JESÚS
DOMÍNGUEZ MORALES</p>

<p>Departamento<br/>

ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES</p>

<p><img src="./images/logoATC.jpg" /></p>

</body>

</html>
```

8.2.2. ayuda.html.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>ATracker - Ayuda</title>

<link href="./images/favicon.ico" rel="shortcut icon" type="image/x-icon"/>

</head>

<body style="text-align:center">

<b><p style="color:#0099CC">INSTRUCCIONES DE  MANEJO DE ATRACKER<br />

</p>

</b>

<p><b style="color:#FF4444">ATENCIÓN:</b> ATracker requiere que su dispositivo
conste de sensor GPS, es por eso que para  su correcto funcionamiento debe
tener habilitada dicha característica en los  ajustes de su terminal.

</p>

<p>Como se indica  al inicio de esta aplicación usted puede utilizar ATracker
para dos propósitos  diferentes que se complementan en la tarea de establecer
el seguimiento de un  envío concreto.</p>

<p><img src="./images/ayuda_1.png"/></p>

<p>Si usted pulsa  sobre el primer botón: <b>CONTROLAR ENVÍO</b> se abre una
nueva pantalla que le permite controlar un  envío. </p>

<p>El terminal se comporta entonces como un señuelo  detectando su posición a
través del sensor GPS de su terminal y escribiéndola en una base de datos
remota. Esta actividad es  la que deben llevar a cabo los operadores de la
agencia de transporte cuando  vayan a realizar un nuevo porte.</p>

<p>Para  comenzar  un  envío  pulsaremos  sobre  el  botón  <b
style="color:#669900">SALIDA</b>.</p>

<p><img src="./images/ayuda_2.png"/></p>

<p>Una  notificación aparecerá en la barra de estado del sistema indicando que
el envío  se haya en curso. Espere unos instantes mientras el envío se
inicia.</p>

<p><img src="./images/ayuda_3.png"/></p>
```

```
<p>En el  cuadro de la parte inferior podrá visualizar los datos referentes al mismo.</p>

<p>Para finalizar un envío pulsaremos sobre el botón <b style="color:#FF4444">LLEGADA</b>.</p>

<p><img src="./images/ayuda_4.png"/></p>

<p>Espere  unos instantes mientras se finaliza, entonces la notificación desaparecerá de  la barra de estado. Podrá ver los datos relativos a la llegada del envío en el  cuadro inferior de la pantalla.</p>

<p>Por el  contrario si  estando en la pantalla  principal, usted pulsa sobre el segundo botón: <b>SEGUIR ENVÍO</b> se abre una nueva pantalla que le permite seguir un  envío.</p>

<p>A continuación, introduzca en el cuadro de texto el número correpondiente a su identificador de envío y pulse sobre el botón adyacente (imagen de una lupa) para averiguar la posición del mismo.</p>

<p><img src="./images/ayuda_5.png"/></p>

<p>Si el identificador que ha introducido es correcto entonces se mostrará un mapa con la posición de su envío: imagen de un paquete.<br  /> Puede hacer clic sobre la misma para ver la información concerniente a dicho envío. </p>

<p>De lo contrario, se le indicará que el identificador introducido no existe.</p>

<p>GRACIAS por usar ATracker.</p>

</body>

</html>
```

8.2.3. conexion.php.

```
<?
	// Conectamos con el servidor:
	$conexion =  mysql_connect('icaro.eii.us.es','jvalzae','120981') or
	die('No se ha podido conectar con el servidor: ' . mysql_error());
	// Seleccionamos la base de datos:
	$bd = mysql_select_db('jvalzae', $conexion) or
	die ('No se ha podido seleccionar la base de datos: ' . mysql_error())
?>
```

8.2.4. iniciar_envio.php.

```
<?
	ob_start();
	include ('./conexion.php');
	//Recuperamos los valores de la querystring:
	$lat = $_GET['lat'];
	$lon = $_GET['lon'];
	$alt = $_GET['alt'];
	$fechaS = $_GET['fechaS'];
	//Escribimos el envio en la base de datos:
	$query = "INSERT INTO envios (idEnvio, latitud, longitud, altura,
	latitudS, longitudS, alturaS, fechaS ) VALUES ('', $lat, $lon, $alt,
	$lat, $lon, $alt, $fechaS)";
	$result = mysql_query($query) or die('Errant query:  '.$query);
	//XML de salida:
	ob_end_clean();
	header('Content-type: text/xml');
	$xml = "<envio>\n\t";
	$xml .= "<idEnvio>".mysql_insert_id()."</idEnvio>\n";
	$xml .= "</envio>";
	$xmlobj = new SimpleXMLElement($xml);
	print($xmlobj->asXML());
?>
```

8.2.5. actualizar_envio.php.

```
<?
	include ('./conexion.php');
	//Recuperamos los valores de la querystring:
	$idEnvio = $_GET['id'];
```

```
	$lat = $_GET['lat'];

	$lon = $_GET['lon'];

	$alt = $_GET['alt'];

	//Actualizamos el envio en la base de datos:

	$query = "UPDATE envios SET latitud = $lat, longitud = $lon, altura =
	$alt WHERE idEnvio = $idEnvio";

	$result = mysql_query($query) or die('Errant query:  '.$query);

?>
```

8.2.6. finalizar_envio.php.

```
<?

	include ('./conexion.php');

	//Recuperamos los valores de la querystring:

	$idEnvio = $_GET['id'];

	$lat = $_GET['lat'];

	$lon = $_GET['lon'];

	$alt = $_GET['alt'];

	$fechaL = $_GET['fechaL'];

	$distancia = $_GET['distancia'];

	//Actualizamos el envio en la base de datos:

	$query = "UPDATE envios SET latitud = $lat, longitud = $lon, altura =
	$alt, fechaL = $fechaL, distancia = $distancia WHERE idEnvio = $idEnvio";

	$result = mysql_query($query) or die('Errant query:  '.$query);

?>
```

8.2.7. consultar_envio.php.

```
<?

	ob_start();

	include ('./conexion.php');
```

```
        //Recuperamos los valores de la querystring:
        $idEnvio = $_GET['id'];
        //Consultamos el envio en la base de datos:
        $query = "SELECT latitud, longitud, fechaS, fechaL, distancia FROM envios
        WHERE idEnvio = $idEnvio";
        $result = mysql_query($query) or die('Errant query:  '.$query);
        $datos = mysql_fetch_array($result);
        //XML de salida:
        ob_end_clean();
        header('Content-type: text/xml');
        $xml = "<envio>\n\t";
        $xml .= "<idEnvio>".$idEnvio."</idEnvio>\n\t";
        $xml .= "<latitud>".$datos['latitud']."</latitud>\n\t";
        $xml .= "<longitud>".$datos['longitud']."</longitud>\n\t";
        $xml .= "<fechaS>".$datos['fechaS']."</fechaS>\n\t";
        $xml .= "<fechaL>".$datos['fechaL']."</fechaL>\n\t";
        $xml .= "<distancia>".$datos['distancia']."</distancia>\n";
        $xml .= "</envio>";
        $xmlobj = new SimpleXMLElement($xml);
        print($xmlobj->asXML());
?>
```

9. Bibliografía.

Android Developers.

http://developer.android.com

Celularis.

http://www.celularis.com/mercado/sistemas-operativos-moviles-2013

Curso de programación Android - sgoliver.net blog.

http://www.sgoliver.net/blog/?page_id=2935

Departamento de Arquitectura y Tecnología de Computadores.

http://www.atc.us.es

Departamento de Lenguajes y Sistemas Informáticos.

http://www.lsi.us.es

Google Developers.

https://developers.google.com

Java.com: Java + You.

http://www.java.com

MySQL :: Developer Zone.

http://dev.mysql.com

ObjectAid UML Explorer.

http://www.objectaid.com

PHP: Hypertext Preprocessor.

http://php.net

Stack Overflow.

http://stackoverflow.com

Wikipedia.

http://es.wikipedia.org