Escola Universitària d'Enginyeria
Tècnica Industrial de Barcelona
Consorci Escola Industrial de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Memoria

APLICACIÓN DE GESTION Y SEGUIMIENTO VIA GPS SOBRE DISPOSITIVO MOVIL I PLATAFORMA ANDROID

PFC presentado para optar al título de Ingeniero Técnico Industrial especialidad Electrónica Industrial

por: Héctor Martínez Calleja
Gabriel Moreno Postigo

Barcelona, 18 de Enero de 2011

Tutor proyecto: Joan Segura Casanovas
Departamento de ESAII (707)
Universitat Politècnica de Catalunya (UPC)

1. Introducción....3
1.1. Motivación del proyecto....3
1.2. Objetivos del proyecto....3
2. Estado del arte....6
2.1. Introducción....6
2.2. Smartphone:....6
2.3. Plataformas de dispositivos móviles.....6
2.3.1. Symbian:....7
2.3.2. Windows Phone (Anteriormente Windows Mobile Edition)....9
2.3.2.1. Windows Mobile 2003:....10
2.3.2.2. Windows Mobile 2003 Second Edition:....10
2.3.2.3. Windows Mobile 5.0:....10
2.3.2.4. Windows Mobile 6:....11
2.3.2.5. Windows Mobile 6.1:....11
2.3.2.6. Windows Phone 6.5.....11
2.3.2.7. Windows Phone 7:....12
2.3.3. iOS:....13
2.3.3.1. Características:....13
2.3.4. Android:....15
3. Justificación....17
3.1. Estudio de mercado....17
4. Plataforma Android....19
4.1. Componentes de Android....19
4.1.1. El núcleo de Linux....20
4.1.2. Máquina virtual Dalvik....20
4.2. Estructura de una aplicación Android....21
4.2.1. Intent....22
4.2.2. Activity....22
4.2.3. BroadcastReceiver....23
4.2.4. Service....24
4.2.5. ContentProvider....24
4.2.6. AndroidManifest.xml....25
5. GPS....26
5.1. Introducción....26

5.2. Historia....26
5.3. Funcionamiento GPS....27
6. Aplicación WiGo....29
6.1. Diagrama de bloques....29
7. Gantt....30
7.1. Tabla de tareas....30
7.2. Diagrama de Gantt....30
8. Aplicaciones de prueba....32
8.1. Árbol de proyectos....32
8.1.1. Carpeta /src/....32
8.1.2. Carpeta /res/....33
8.1.3. Carpeta /gen/....33
8.1.4. Carpeta /assets/....35
8.2. Aplicación: HelloAndroid....35
8.2.1. Clase: HelloAndroid.java....35
8.2.2. AndroidManifest.xml....36
8.2.3. Layout: main.xml....37
8.2.4. Values: strings.xml....38
8.2.5. Screenshot de la aplicación....39
8.3. Aplicación: GPS_02....40
8.3.1. Clase: GPS_02.java....40
8.3.2. AndroidManifest.xml....41
8.3.3. Layout: main.xml....42
8.3.4. Values: strings.xml....42
8.3.5. Screenshot de la aplicación....42
9. Presupuesto....45
9.1. Costes de Ingeniería....45
9.2. Otros costes....45
1.1. Costes totales....46

1. Introducción

En las siguientes líneas se hace una breve introducción al presente proyecto, exponiendo cuál es su motivación, qué objetivos son los que persigue y cuáles son los ofrecidos en esta memoria.

1.1.Motivación del proyecto

Los teléfonos móviles y los dispositivos manuales portátiles experimentan en la actualidad grandes cambios debidos a diferentes factores. Por un lado, los dispositivos portátiles son más potentes y capaces de realizar tareas que hace unos años nadie podría haber imaginado. Muchos de nosotros llevamos encima un dispositivo capaz de conectarse a la Web para ver películas o juegos en 3D, e incluso de realizar llamadas. Por otra parte, los consumidores demandan estas prestaciones en los nuevos dispositivos.

Una tercera parte de la convergencia es que ahora los dispositivos portátiles constituyen un mayor mercado para los programadores de software y aplicaciones, mayor que el de las principales plataformas informáticas, y la creación de aplicaciones para dichos dispositivos sueles ser más sencilla y racionalizada.

La nueva generación de teléfonos ya incluye hardware de aceleración grafica, conectividad inalámbrica, planes de acceso a datos, GPS, ampliación y conectividad de hardware, pantallas táctiles, etc. Los Sistemas Operativos (SO en adelante) y las aplicaciones se crean para aprovechar estas nuevas prestaciones, al tiempo que los consumidores controlan lo que pueden hacer sus dispositivos, lo que establece una conexión fluida entre programadores y consumidores. Estos últimos consiguen el software que desean y los programadores acceden a un mercado potencialmente ilimitado para sus productos.

A esta transformación subyace una tendencia hacia un mayor aperturismo: en las prestaciones de los dispositivos y como controlarlas, en el desarrollo y comercialización de aplicaciones, en la colaboración entre fabricantes de dispositivos, proveedores de red y de software.

1.2.Objetivos del proyecto

El nuevo sistema operativo para dispositivos móviles de Google, Android, centra el desarrollo de este proyecto final de carrera. Para poder dirigir con mayor éxito los esfuerzos por conocer y comprender las características de este nuevo sistema, es necesario fijar unos objetivos que abarquen las actividades que se pretenden realizar y, además, permitan al final de las mismas conocer el grado de desarrollo y cumplimiento alcanzado.
Los objetivos previstos son:

- **Comparativa de las diferentes plataformas móviles del mercado.** Para poder decidir la plataforma adecuada a nuestro proyecto, deberemos realizar un estudio previo de las características de las plataformas actuales.
    - Symbian
    - Windows Phone
    - iOS
    - Android

- **Justificación de elección de la plataforma Android.** Citar las razones que nos han llevado a seleccionar esta plataforma para nuestro proyecto.

- **Tecnología GPS.** Estudio del funcionamiento de la tecnología de posicionamiento global, para su posterior uso en nuestra aplicación.
    - **Funcionamiento del GPS en dispositivos móviles.** A-GPS, GPS vía Wi-Fi, etc..

- **Adquisición de las herramientas de desarrollo Android.** Es obvio que para crear aplicación software es necesario una serie de herramientas para desarrolladores.
    - Eclipse
    - SDK Android
    - Plugin ADT
    - API's

- **Adquirir conocimientos básicos del lenguaje de programación para Android**. Dado que no hemos estudiado el lenguaje de programación para Android (Java™) en la escuela, será necesaria una previa preparación sobre este lenguaje.

- **Diseño de primeras aplicaciones a nivel de aprendizaje**. Todo aprendizaje comienza por una serie de pruebas, en el caso de la programación, crearemos algunas aplicaciones básicas (HelloWorld, programas de manipulación de ventanas, etc.)

- **Diagrama de bloques de la aplicación.** Idear el funcionamiento básico de la aplicación y su diseño gráfico ajeno a la programación.

- **Desarrollo de la Aplicación.**

    - **Primera versión.** En una primera versión, intentaremos mostrar y manipular los datos obtenidos por el GPS interno del dispositivo.
    - **Segunda versión.** Cartografiar la posición actual y tratamiento de cartografías.
    - **Tercera versión.** Generación de rutas.
    - **Cuarta versión.** Situar puntos conocidos en cartografía.

    - **Quinta versión.** Diseño de la interfaz gráfica

- **Versión Beta de la Aplicación "WiGo"**. Primera versión con todas las funcionalidades finales.

- **Depuración de la versión Beta, y registro de la aplicación final en *Android Market*.** Una vez comprobada la correcta funcionalidad de la aplicación se pretende registrar en Android Market para su libre distribución.

2. Estado del arte

A continuación veremos el estado actual de las plataformas móviles actuales, así como sus expectativas de futuro.

2.1.Introducción

Actualmente, dada la multitud de dispositivos móviles que existen en el mercado y la infinidad de: características, aplicaciones, funciones, etc. Se han desarrollado varias plataformas para gestionar dichos dispositivos.

A día de hoy el dispositivo móvil más extendido en la sociedad es el SmartPhone (Teléfono inteligente).

2.2.Smartphone:

Es una evolución del teléfono móvil común que cuenta con ciertas características y prestaciones que lo acercan más a un ordenador personal que a un simple teléfono.
Entre dichas características, se puede encontrar una mejora en la capacidad de proceso y Almacenamiento de datos, conexión a Internet mediante Wi-Fi o GPRS / 3G, pantalla táctil, Acelerómetro, GPS, teclado QWERTY y diversas aplicaciones de usuario como navegador web, cliente de correo, aplicaciones ofimáticas, reproductores de vídeo y audio, etc. Incluyendo la posibilidad de descargar e instalar nuevas aplicaciones.

No debemos olvidar que a pesar de estas importantes mejoras con respecto a sus predecesores móviles, el reducido tamaño de los *SmartPhones* conlleva inevitablemente limitaciones de hardware que los mantienen claramente diferenciados de los ordenadores convencionales. Estas limitaciones se reflejan principalmente en pantallas más pequeñas, menor capacidad del procesador, restricciones de memoria tanto de datos como de proceso y necesidad de adaptar el consumo de energía a la capacidad de una pequeña batería.

Están limitaciones técnicas obligan a tener muy presente la capacidad real del dispositivo a la hora de desarrollar su software.

2.3.Plataformas de dispositivos móviles.

Como hemos dicho anteriormente, dadas las características cercanas a un ordenador, para la gestión de un SmartPhone es necesario un SO.

Al igual que existen multitud de fabricantes de dispositivos, existen también multitud de plataformas de gestión, entre los más importantes se encuentran: Symbian, Android, iOS, Windows Phone (Anteriormente Windows ME).

Figura 1. Principales plataformas de dispositivos móviles

2.3.1. Symbian:

Symbian es un SO que fue producto de la alianza de varias empresas de telefonía móvil, entre ellas Nokia, Psion, Ericsson, Motorola, Siemens…

El objetivo de Symbian fue crear un SO para terminales móviles que pudiera competir con Palm OS o Windows ME.

La primera versión de Symbian, basada en el sistema EPOC de Psion, nació en 1998. Actualmente la última versión estable es Symbian^3 lanzada el Q1 2010.

El acuerdo bajo el cual se desarrolló Symbian es bastante simple: Symbian Ltd. desarrolla el sistema operativo Symbian, que incluye el microkernel, los controladores, el *middleware* y una considerable pila de protocolos de comunicación e interfaces de usuario muy básicas.

Los desarrolladores que obtienen la licencia correspondiente para trabajar con Symbian implementan sus propias interfaces de usuario y conjuntos de aplicaciones según las necesidades de sus propios dispositivos.

Esto permitió a Symbian posicionarse como un SO muy flexible, que tenía en cuenta los requisitos de la mayoría de los dispositivos fabricados y, al mismo tiempo, permitía un alto grado de diferenciación

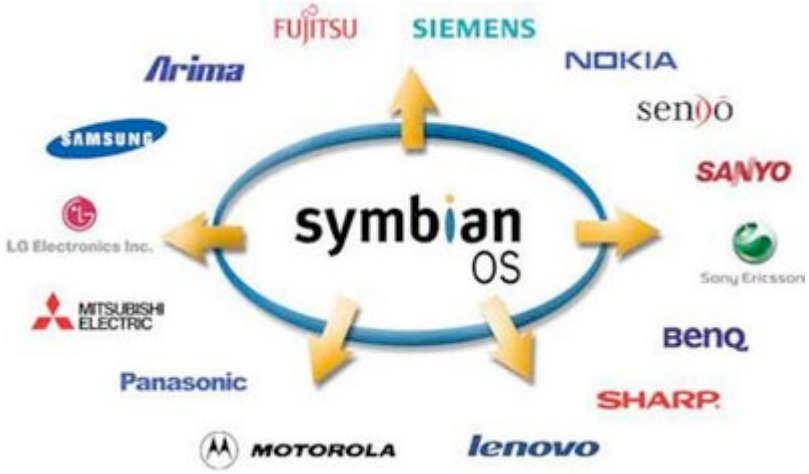

Figura 2. Principales fabricantes de la alianza Symbian.

En Symbian, una pequeña porción del sistema tiene privilegios de Kernel, el resto se ejecuta con privilegios de usuario, de forma que los procesos en ejecución y sus prioridades son manejados por este. Cada una de las aplicaciones corre en su propio proceso y tiene acceso únicamente a una exclusiva zona de memoria.
Symbian contempla diferentes ediciones del SO según las características del dispositivo móvil:

- **Symbian S60:**
  La versión S60, puede ser la más popular de todas, debido fundamentalmente a que Nokia, uno de los fabricantes más importantes del mundo, ha hecho de symbian y de su versión S60 el núcleo de casi todos los modelos de SmartPhones. Normalmente los dispositivos con S60, tienen una pequeña pantalla y un teclado del tipo 0 – 9#.

- **Symbian S80:**
  Edición, utilizada por Nokia, orienta a terminales que disponen de pantalla táctil y permiten multitarea, pudiendo tener varias aplicaciones abiertas simultáneamente.

- **Symbian S90:**
  Serie muy parecida a la S80, solo que está orientada a dispositivos con pantallas más grandes. Utilizan teclados virtuales, reconocimientos de trazos o teclados acoplables mediante, por ejemplo, Bluetooth.

- **Symbian UIQ (User Interface Quartz):**
  La Edición UIQ se encuentra muy influenciada por Palm OS. Implementa una especie de multitarea virtual, dando al usuario la falsa sensación de poder realizar varias acciones simultaneas.

Desarrollar aplicaciones para Symbian es relativamente sencillo, porque permite utilizar lenguajes habituales como Java, C++, Visual Basic o Perl, entre otros, para desarrollar aplicaciones. Este hecho ha permitido que actualmente sean cientos de miles las aplicaciones y utilidades disponibles para Symbian.

2.3.2. Windows Phone (Anteriormente Windows Mobile Edition)

Windows Phone, es un SO, diseñado por Microsoft Corporation, y orientado a multitud de dispositivos móviles.

Figura 3. Logo comercial Windows Phone

Basado en el núcleo de otro gran SO llamado Windows CE (Windows Compact Edition).
Originalmente apareció con el nombre de Pocket PC. Windows ha cambiado varias veces de nombre siendo su última versión la llamada Windows Phone 7 durante el 2010.
A principios de la década de los 90, cuando comenzaron a aparecer los primeros dispositivos móviles, Microsoft tomó la decisión de crear un sistema operativo capaz de hacer frente al entonces recientemente lanzado por Apple, el sistema Newton Message Pad. Fruto de esta iniciativa surgió Pegasus, cuyo nombre comercial definitivo fue Windows Compact Edition, o Windows CE.

El objetivo principal que buscaba Microsoft era que el nuevo sistema fuera lo suficientemente flexible y adaptable para poder ser utilizados en un amplio abanico de dispositivos, cuyo única característica común es la de ser de reducido tamaño y tener, por tanto, una limitación obvia en sus recursos.
Las características principales con las que cuenta Windows CE son las siguientes:

- Es un sistema modular, lo que permite que cada fabricante pueda seleccionar aquellas partes que le benefician más a su dispositivo.
- Contempla una considerable gama de recursos hardware: teclado, cámara, pantalla táctil, etc.
- Tiene un tamaño en memoria relativamente pequeño y de bajo coste.
- Es capaz de trabajar con distintas familias de procesadores de 32 bits.
- Permite interactuar con otros dispositivos móviles.

Un aspecto distintivo de Windows CE con respecto a otros productos desarrollados por Microsoft es que un elevado número de sus componentes se ofrece a los fabricantes y desarrolladores a través del propio código fuente. Esto les permite poder adaptar el sistema a sus dispositivos específicos. Aquellos componentes básicos de Windows CE que no necesitan ningún tipo de adaptación siguen siendo ofrecidos únicamente como código binario.

La arquitectura básica de Windows CE es la explicada a continuación:

- OEM Layer: Es la capa situada entre el hardware del dispositivo y l kernel. Permite a los fabricantes desarrollar sus propios drivers y funciones de control de los elementos de hardware.
- Operating System Layer: incluye el kernel como elemento principal y el conjunto de API Win32 necesarias. En esta capa se sitúan las bibliotecas de comunicaciones, el gestor gráfico, gestor de ficheros y registros, así como otros componentes opcionales.
- Application Layer: donde residen las aplicaciones por defecto de Windows CE y las aplicaciones del usuario.

Algunas de las versiones más conocidas cronológicamente son:

2.3.2.1. Windows Mobile 2003:

Lanzada el 23 de junio de 2003, era el primer lanzamiento bajo el nombre Windows Mobile. Existían tres ediciones diferentes:

- Windows Mobile 2003 Pocket PC Edition.
- Windows Mobile 2003 Pocket PC Phone Edition, diseñado para los Pocket PC con características móviles (HTC Himalaya…).
- Windows Mobile 2003 SmartPhone Edition, plataforma similar a la Pocket PC Edition substancialmente diferente, ya que está limitada por las características especiales de estos dispositivos, tales como: funcionamiento por teclas en vez de pantalla táctil, resolución de pantalla más baja, capacidad de memoria…

2.3.2.2. Windows Mobile 2003 Second Edition:

Apareció el 24 de marzo de 2004. Algunas de sus mejoras respecto a su predecesor son:

- Cambiar la orientación de la pantalla.
- Pocket Internet Explorer.
- Soporte para resolución de pantalla VGA (640 x 480).
- Soporte para Wi-Fi.

2.3.2.3. Windows Mobile 5.0:

Lanzada al mercado el 9 de mayo del 2005. Algunas de las características más relevantes son:

- Una nueva versión de Office llamada "Office Mobile".
- Reproductor "Windows Media 10 Mobile".
- Un paquete multimedia que facilitará la administración de vídeos y fotos.

- Ayuda mejorada de Bluetooth.
- Interfaz de administración GPS para los programas de navegación instalados.
- Soporte para teclados QWERTY incluido por defecto.
- Cliente para PPTP y L2TP/IPsec VPNs.
- La memoria no volátil (ROM) está disponible en Pocket PC permitiendo un aumento de la batería. Anteriormente más del 50% (suficiente para 72 horas de almacenaje) de energía de la batería se reservaba para mantener datos en la memoria RAM (volátil). Los dispositivos basados en Windows usa la memoria RAM como su medio de almacenaje primario al uso de memoria flash.

2.3.2.4. Windows Mobile 6:

Lanzado el 12 de febrero del 2007. Ofrece tres versiones:

- Windows Mobile 6 Standard para Smartphones (teléfonos sin pantalla táctil).
- Windows Mobile 6 Professional para PDAs con la funcionalidad del teléfono (Pocket PC Phone Edition).
- Windows Mobile 6 Classic para PDAs sin telefonía IP.

Algunas de las características más relevantes son:

- Soporta las resoluciones 800x480 y 320x320.
- Opción de 1:1 en la páginas web
- Desarrollo y distribución de aplicaciones más rápida y más fácil.
- Soporte VoIP con los codec del audio AEC (Acoustic Echo Cancelling) y MSRT
- Windows Live para Windows Mobile.9
- La pila Bluetooth de Microsoft ha mejorado notablemente.
- Cifrado de la tarjeta de almacenamiento - Windows Mobile 6 para Pocket PC y Smartphone soportan el cifrado de los datos almacenados en tarjetas externas de almacenamiento.
- Soporte AJAX, JavaScript y XMLDOM en Internet Explorer Mobile.
- .NET Compact Framework v2 SP1 en la ROM.
- SQL Server Compact Edition en la ROM.

2.3.2.5. Windows Mobile 6.1:

Anunciada el 1 de abril de 2008. Pequeña actualización de la plataforma Windows Mobile 6.0, incluye mejoras de rendimiento, pantalla inicial rediseñada, zoom a página completa en Internet Explorer, etc.

2.3.2.6. Windows Phone 6.5.

El 6 de Octubre de 2009 fue el lanzamiento mundial de la nueva versión de Windows Mobile, cabe decir que desde ese día cambió el nombre a Windows Phone.
Algunas de las novedades son:

- Cambio completo de la interfaz de usuario, para poder adaptarse a los nuevos dispositivos táctiles de forma que se pueda utilizar fácilmente con el dedo.
- Windows Marketplace: Acceso a la tienda de aplicaciones de Microsoft que contiene un gran número de aplicaciones
- Internet Explorer Mobile 6.
- Microsoft My Phone: Aplicación que permite disponer de 200 Mb en los servidores de Microsoft para mantener una copia de seguridad de los datos de nuestro teléfono móvil.

2.3.2.7. Windows Phone 7:

Anunciado el 15 de febrero del 2010. Windows Phone 7 ofrece una interfaz completamente diferente, tomando prestados varios conceptos que se pudieron observar en el Zune HD y que han dado un buen resultado. Todo se encuentra administrado en un sistema de bloques que representan un cambio estético muy importante. Estas mejoras visuales representan una demanda importante de hardware.

Figura 4. Terminal con Windos Phone (HTC HD7)

2.3.3. iOS:

El lanzamiento de iOS tuvo lugar el 29 Junio de 2007. Sistema Operativo para dispositivos móviles de Apple. Desarrollado originalmente para el iPhone, siendo usado más tarde por iPod Touch e iPad.

Figura 5. Logo comercial Apple iOS 4.

El iOS tiene 4 capas de abstracción, la capa del núcleo del SO, la capa de servicios principales, la de medios de comunicación y la de "Cocoa Touch".

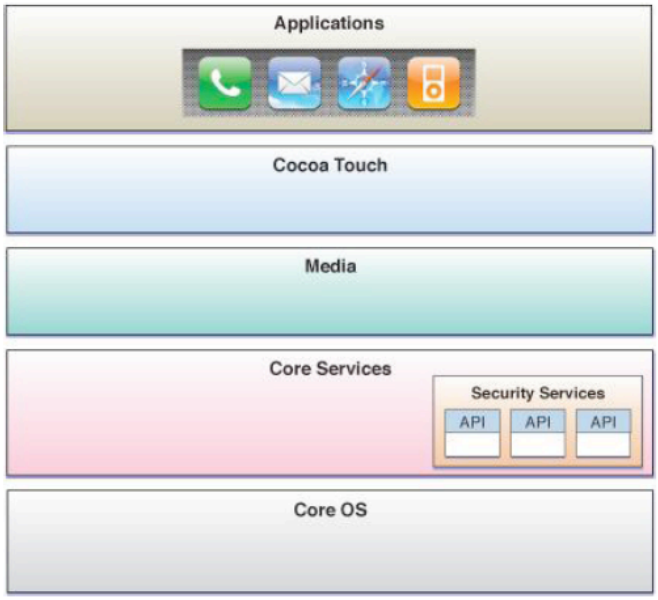

Figura 6. Estructura de capas del iOS 4.

2.3.3.1. Características:

Interfaz de usuario, basada bajo el concepto de manipulación multitáctil. Los elementos básicos se componen por deslizadores, interruptores y botones. La respuesta es inmediata y se provee de una interfaz fluida. La interacción con el SO se realiza mediante gestos como deslizar, tocar y pellizcar. Acelerómetros y Giroscopios internos son utilizados por algunas aplicaciones para responder a movimientos y gestos, como sacudir el aparato (en campos de texto es usado para deshacer y rehacer) o rotarlo (se suele usar para cambiar de posición vertical a modo paisaje).

Home Screen o pantalla principal, es donde se ubican los iconos de las aplicaciones. La pantalla tiene una barra de estado en la parte superior para mostrar datos, tales como: la hora, el nivel de la batería y la intensidad de señal.

Aplicaciones, puede que sea la característica más importante del iOS. Todos los dispositivos dotados con iOS incorporan varias aplicaciones, cómo pueden ser:

- Mail.
- Navegador Web: Safari.
- Reproductor de medios: iPod.
- Visor de mapas: Google maps.
- Visor de videos: Youtube.
- Descargar aplicaciones: App Store.

Además, existe la AppStore, con más de 300.000 aplicaciones listas para descargar.

Figura 7. Modelo comercial con iOS 4 (Apple iPhone 4).

2.3.4. Android:

En Julio de 2005, Google adquirió Android Inc., una pequeña compañía de Palo Alto, California. En ese entonces, poco se sabía de las funciones de Android, Inc. Únicamente que desarrollaba software para teléfonos móviles. Esto dio pie a rumores de que Google estaba planeando entrar en el mercado de los teléfonos móviles.

Figura 8. Logo comercial Android.

El organismo responsable de Android es la *Open Handset Aliance*, dicha organización es una alianza formada por 30 compañías dispuesta a instaurar una telefonía abierta y de mejor calidad en el mercado.

Android es un entorno de software creado para dispositivos móviles. No es una plataforma de hardware. Incluye un SO basado en Linux, una completa interfaz de usuario, aplicaciones, bibliotecas de código, estructuras para aplicaciones, compatibilidad multimedia y mucho más, ¡Incluso funcionalidad de teléfono móvil!

Aunque los componentes del SO se escriban en C o C++ las aplicaciones para Android se diseñan en Java, incluso las aplicaciones incorporadas están escritas en Java.

Una característica de la plataforma Android es que no existen diferencias entre las aplicaciones incorporadas y las creadas con el SDK, esto significa que se pueden crear completas aplicaciones para aprovechar los recursos disponibles en el dispositivo. Puede que lo más notable de Android se su naturaleza de código abierto: La comunidad de desarrolladores proporciona los elementos de los que carece. El SO basado en Linux no incorpora un entorno sofisticado, pero como la plataforma es abierta, se puede modificar.

En cuanto el mercado al que destinado Android, su objetivo es admitir diversos dispositivos d hardware, no solo los más avanzados, que suele asociarse a los costos *Smartphones*. Evidentemente, Android funcionará mejor en dispositivos más potentes, en especial si tenemos en cuenta que incluye completas funciones informáticas. No obstante, Android puede adaptarse a la funcionalidad de un teléfono tradicional con el objetivo de aumentar su cuota de mercado.

Android tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, existen cerca de 200.000

aplicaciones disponibles para Android. Android Market es la tienda de aplicaciones en línea administrada por Google, aunque existe la posibilidad de obtener software externamente a diferencia de otras plataformas.

El código abierto es un arma de doble filo, por un lado, el poder de individuos y empresas de todo el mundo que se esfuerzan continuamente por ofrecer las funciones deseadas, una iniciativa a tener en cuenta, sobre todo si lo comparamos con el enfoque tradicional y comercial del desarrollo de software, el otro lado de la ecuación, es que sin una base estable de código centralizado, Android podría perder la masa crítica que necesita para irrumpir en la telefonía móvil. Veamos como ejemplo el caso de Linux como alternativa a Windows. Linux a disfrutado de un tremendo éxito; se encuentra en muchos SO, en dispositivos como routers y conmutadores, y en multitud de plataformas móviles como Android. Existen diversas distribuciones para equipos de escritorio e irónicamente, esta variedad de opciones es lo que ha retrasado su adopción como alternativa de escritorio a Windows. Linux es sin duda el mayor éxito de producto en código abierto, como alternativa de escritorio a Windows, se ha fragmentado, en varias versiones como: Ubuntu, openSUSE, Debian… lo que ha dificultado su entrada en el mercado.

	symbian OS	Windows Mobile			
Conectividad	3G,GSM,WiFi	3G,GSM,WiF CDMA	3G,GSM,WiFi, CDMA	3G,GSM,WiFi	3G,GSM,WiFi, GPRS
Creador	Nokia	Microsoft	RIM	Apple	Google
Disponibilidad de aplicaciones	Alta	Alta	Mediana	Alta	Mediana /Alta
Disponibilidad de App Store	Sí	Sí	Sí	Sí	Sí
Dispositivos compatibles	Smartphone / teléfono	PDA / Smartphone	Smartphone	Smartphone	Smartphones /netbook
Licencia	Privada	Privada	Privada	Privada	Abierta
Multitarea	Sí	Sí	Sí	Sí	Sí
Táctil/Multitáctil	Sí	Sí	Sí	Sí	Sí
Tipo de núcleo	Symbian	WindowsCE	Propio	iPhoneOS	Linux

Figura 9. Tabla comparativa diferentes plataformas móviles.

3. Justificación

En esta sección se describirá cual ha sido la plataforma escogida para el desarrollo del proyecto, y cuáles han sido los motivos que nos han llevado a tomar la decisión.

3.1.Estudio de mercado

Para la selección de nuestra plataforma, hemos querido tener en cuenta el estado actual de la tecnología en este campo de aplicación. Sin embargo, no solo debemos fijarnos en el presente sino mirar hacia el futuro de las plataformas existentes.

Los datos mostrados pertenecen al artículo *"Gartner Says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position by 2014"* de la consultora, Gartner, S.A. Esta sociedad es un proyecto de investigación de tecnologías de la información y de firma consultiva con sede en Stamford, Connecticut, Estados Unidos. Se conocía como el Grupo Gartner hasta 2001.Gartner incluye como clientes algunas grandes empresas, agencias de gobierno, empresas tecnológicas y la agencias de inversión como BT, CV, Wall Street journal etc. La empresa se concentra en la Investigación, Programas Ejecutivos, Consultas y eventos. Fue fundada en 1979; Gartner tiene 4.000 socios, incluyendo a 1.200 analistas y consultores en 75 países por todo el mundo.

Hemos podido comprobar que dichos datos distan mucho entre consultoras. Dichas diferencias pueden deberse a intereses propios o que han sido realizadas en diferente fecha.

A continuación vemos la cuota de mercado en Agosto de 2010:

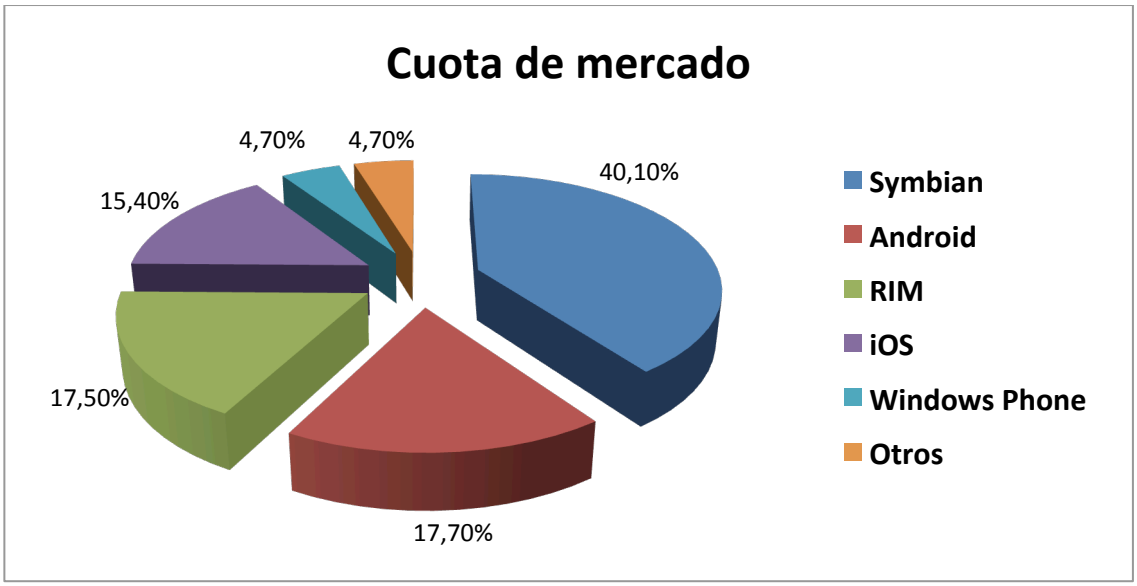

Figura 10. Cuota de mercado según Gartner (Agosto 2010).

En la siguiente gráfica, podemos ver las previsiones de crecimiento para los principales SO del mercado:

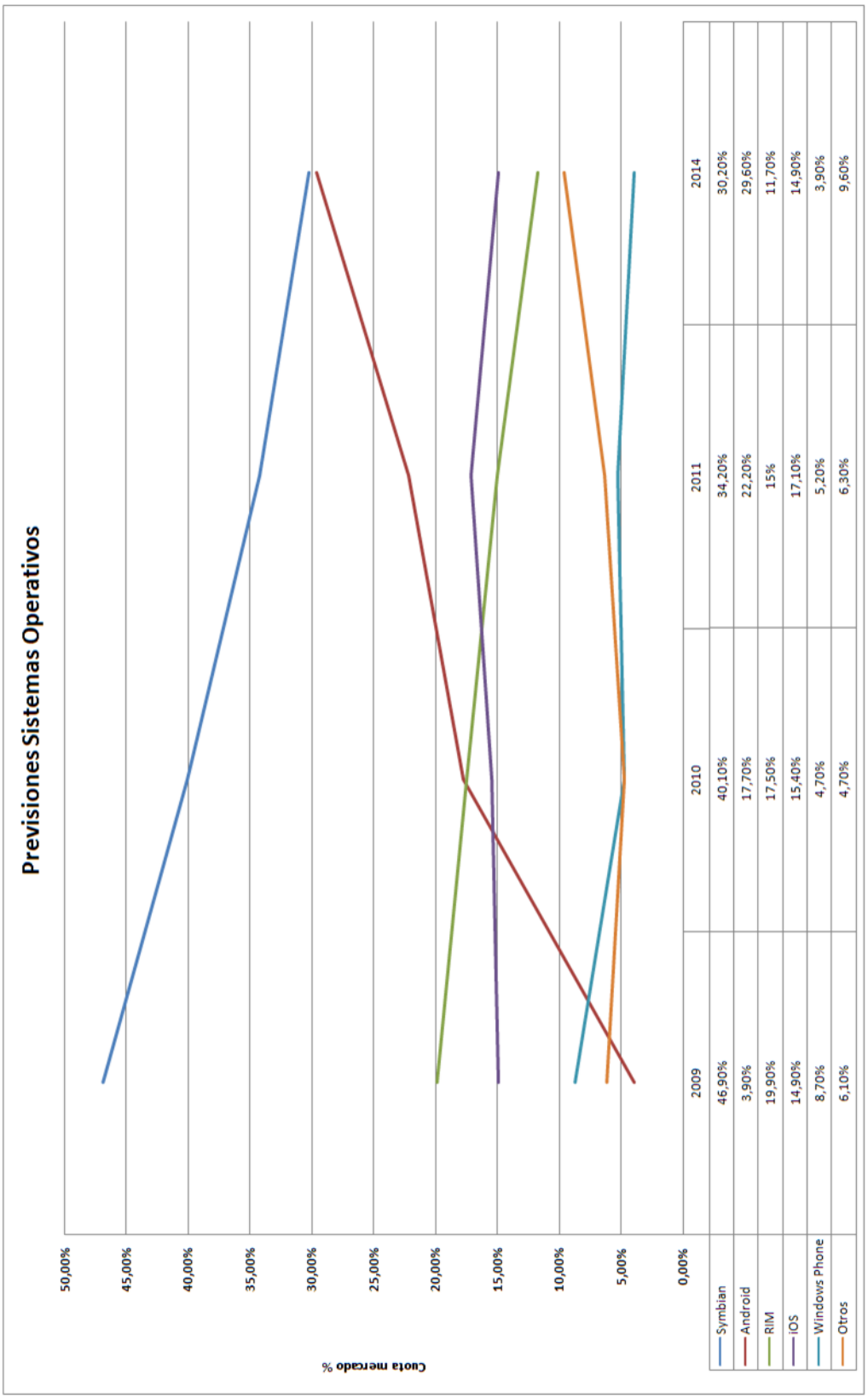

Figura 11. Gráfica de previsión de crecimiento por plataformas (Gartner, Agosto 2010).

4. Plataforma Android

En esta sección veremos más al detalle los cimientos de la plataforma Android, es decir, sus componentes, objetivos, estructura y principales características, que han llevado a este SO a ser uno de los más valorados por los usuarios y con grandes expectativas de futuro.

4.1.Componentes de Android

Android incluye una impresionante variedad de funciones para aplicaciones móviles. De hecho, si analizamos únicamente la arquitectura, sin el contexto de Android como plataforma diseñada para entornos móviles, podríamos confundirlo con un entorno informático general. A continuación presentamos los principales componentes de Android:

- Un núcleo Linux que proporciona una capa de abstracción de hardware básica, así como servicios como gestión de procesos, memoria, y sistema de archivos. En el núcleo se implementan controladores de hardware específicos, funciones como Wi-Fi y Bluetooth. La pila de Android tiene un diseño flexible, con diferentes componentes opcionales que dependen de la disponibilidad de hardware en cada dispositivo concreto, como por ejemplo pantallas táctiles, GPS o acelerómetros.

- Entre las principales bibliotecas de código destacan las siguientes:
  - Tecnología de navegar de WebKit, el mismo motor de código abierto de los navegadores Safari de Macintosh y del mobile Safari del iPhone.
  - Compatibilidad con bases de datos a través de SQLite.
  - Compatibilidad gráfica avanzada, animación de SGL y OpenGL ES.
  - Compatibilidad con audio y vídeo a través de OpenCore de Packet Video.
  - Funciones SSL del proyecto *Apache.*

- Diferentes administradores de servicios para:
  - Actividades y vistas.
  - Telefonía.
  - Servicios basados en ubicación.

- El entorno de ejecución de Android proporciona lo siguiente:
  - Paquetes Java para obtener un entorno de programación Java prácticamente completo. No es un entorno J2ME.
  - La máquina virtual Dalvik utiliza servicios del núcleo basado en Linux para proporcionar un entorno de alojamiento para las aplicaciones Android.

Figura 12. Dispositivo con SO Android (Nexus S).

4.1.1. El núcleo de Linux

La pregunta es, ¿Por qué utilizar Linux en un teléfono? El uso de una plataforma tan completa como Linux, proporciona gran potencia y funciones a Android. El uso de una base de código abierta desata la capacidad de individuos y empresas para impulsar la plataforma. Es especialmente importante en el mundo de los dispositivos móviles, donde los productos cambian con tanta rapidez. La velocidad de cambio en el mercado de la telefonía hace parecer lento al sector de la informática general. Y, además, el núcleo de Linux es una plataforma demostrada. En un teléfono móvil la fiabilidad es más importante que el rendimiento, ya que se utiliza principalmente, para comunicaciones por voz.

Otra ventaja del uso de Linux como base de Android es que proporciona un nivel de abstracción de hardware lo que permite conservar los niveles superiores independientemente de los cambios realizados en el hardware subyacente. Evidentemente el diseño de código demanda que las aplicaciones no fallen en caso de que falte un recurso, como por ejemplo la ausencia de una cámara.

Las aplicaciones de usuario, así como las aplicaciones básicas de Android, se escriben en Java y se compilan en código de bytes, que se interpretan en tiempo de ejecución por medio de una máquina virtual.

4.1.2. Máquina virtual Dalvik

La máquina virtual Dalvik es un ejemplo de las necesidades de eficacia, el deseo de un entorno de programación completo e incluso el enfrentamiento de ciertas restricciones de propiedad intelectual, dando todo ello como resultado la innovación. El entorno Java de

Android proporciona una completa plataforma de aplicaciones y resulta muy accesible debido a la popularidad del propio lenguaje Java. Además, el rendimiento de las aplicaciones, en especial en entornos de memoria reducida como son los teléfonos, es imprescindible para el mercado de la telefonía móvil, aunque no sea el único problema.

Como hemos dicho anteriormente Android no es una plataforma J2ME. Sin comentar si esto es positivo o no, existen otras fuerzas presentes. Está el problema de la máquina virtual de Java de Sun Microsystems. El entorno de código de Android es Java. Las aplicaciones se escriben en Java y se compilan en código de Java para después traducirse a una representación diferente denominada archivos *dex*. Estos archivos son los equivalentes lógicos de los códigos de bytes de Java pero permiten que Android ejecute sus aplicaciones en su propia máquina virtual que no depende de las licencias de Sun.

El dato que hay que recordar sobre la máquina virtual Dalvik es que las aplicaciones de Android se ejecutan en su interior y que depende del núcleo de Linux para servicios como procesadores, memoria y administración de sistemas de archivos.

4.2.Estructura de una aplicación Android

La estructura de una aplicación Android está definida por la interacción de distintos componentes, haciendo énfasis en la agrupación de distintas piezas. La aplicación hará uso de las distintas API's expuestas por Android, de forma que los componentes encargados de realizar cada tarea puedan ser manipulados o reemplazados sin problemas, asegurando la máxima flexibilidad. Por ejemplo, una aplicación puede permitir al usuario elegir fotos mediante el componente "Galería" o, por ejemplo, reemplazar esa "Galería" por una selección de fotos a través de un servicio online.

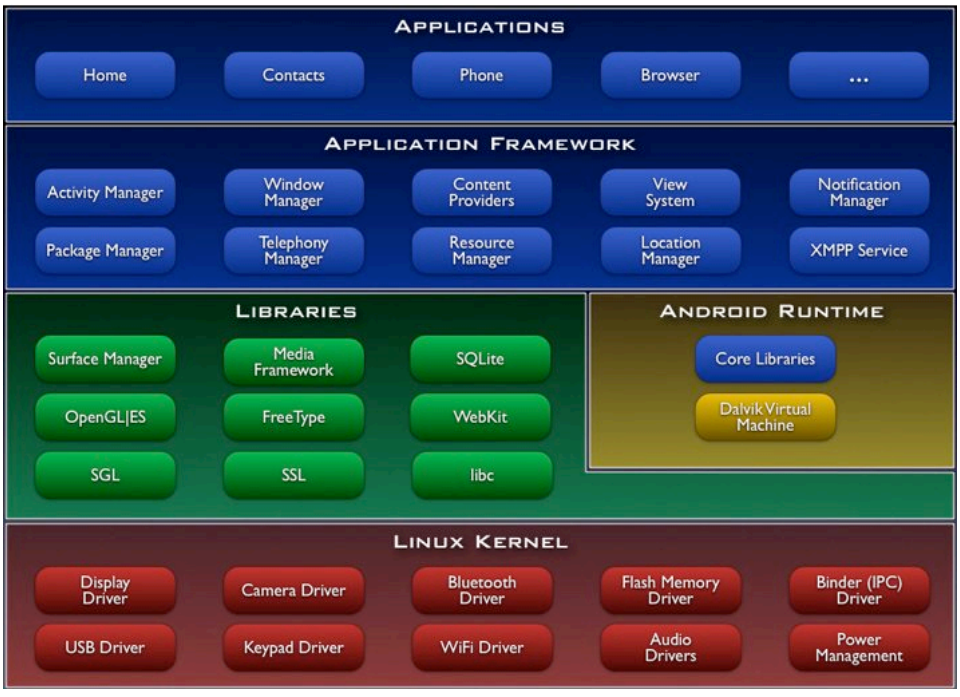

Figura 13. Capas del SO Android.

Los principales componentes de una aplicación serían:

4.2.1. Intent

Android usa una clase especial llamada `Intent` para moverse de una pantalla a otra. Un "`Intent`" describe lo que una aplicación desea hacer. Las dos partes más importantes de la estructura de datos de un "`Intent`" son la acción y los datos sobre los cuales se actuará. Los valores típicos para la acción son "MAIN", "VIEW", "PICK", "EDIT", etc. Los datos son expresados como una URI. Por ejemplo, para ver la información de contacto de una persona podría ser necesario crear un "`Intent`" con la acción "VIEW" y los datos definidos como una URI que representa a esa persona.

Una clase relacionada con "`Intent`" es "`IntentFilter`". Si bien un "`Intent`" es una solicitud para realizar algo, un "`IntentFilter`" es una descripción de lo que intenta hacer una "`Activity`" (o de lo que intenta recibir). Una "`Activity`" que es capaz de desplegar la información de contacto de una persona podría declarar que sabe cómo tratar la acción "`VIEW`" cuando es aplicada a los datos que representan a la persona. Una "`Activity`" declara sus "`IntentFilter`"s en el archivo "`AndroidManifest.xml`"

4.2.2. Activity

Representa cada una de las principales tareas que el usuario puede llevar a cabo en la aplicación. Típica (aunque no necesariamente) corresponderá a una pantalla específica de la aplicación y, también normalmente, una `Activity` será el punto de entrada (pantalla inicial) de nuestra aplicación. Desde ella se invocarán las vistas, específicas o `Layout`, para la aplicación. A continuación podemos ver un ejemplo típico de `Activity`:

```
Package com.example.helloandroid;

Import android.app.Activity;
Import android.os.Bundle;

public class HelloAndroid extends Activity {
/** Called when the Activity is first created. */
  @Override
  Public void onCreate(BundlesavedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
  }
}
```

4.2.3. BroadcastReceiver

Si una aplicación desea recibir y responder a un evento global como por ejemplo una llamada de teléfono o un mensaje de texto entrante, debe registrarse como `BroadcastReceiver`.

Para ello, existen dos técnicas:

- La aplicación puede implementar un elemento `<receiver>` en el archivo AndroidManifest.xml, que describe el nombre de clase de `BroadcastReceiver` y enumera sus `IntentFilter`. Recordar que `IntentFilter` es un descriptor del `Intent` que desea procesar la aplicación. Si el receptor se registra en el archivo `AndroidManifest.xml`, no es necesario ejecutarlo para desencadenarlo. Cuando se produce el evento, la aplicación se inicia automáticamente al notificar el evento. El propio SO de Android se encarga de estas tareas de mantenimiento.

- Una aplicación se puede registrar en tiempo de ejecución a través del modelo `registerReceiver` de la clase `Context`.

Al igual que `Service`, `BroadcastReceiver` no dispone de IU. Más importante todavía, el código que se ejecuta en el método `onReceive` de `BroadcastReceiver` no debe asumir operaciones de persistencia o ejecución prolongada. Si `BroadcastReceiver` requiere una cantidad mayor de ejecución de código, es recomendable que dicho código solicite un servicio para completar la funcionalidad solicitada.

```
Package com.example.helloBroadcastReceiver;

   Import android.content.Context;
   Import android.content.Intent;
   Import android.content.IntentReceiver;
   Import android.util.Log;

   public class MySMSMailBox extends BroadcastReceiver{
      public static final string tag = "MySMSMailBox";
      @override
         Public void onReceive(Context context, Intentintent){
            Log.i(tag,"onReceive");
            If (intent.getAction().equals
               ("android.provider.Telephony.SMS_RECEIVED")){
               Log.i(tag, "FoundourEvent!!");
            }
        }
   }
```

4.2.4. Service

Si el ciclo de vida de una aplicación es prolongado, debe incluirse en un `Service`. Por ejemplo, una utilidad de sincronización de datos de fondo que se ejecute de forma continuada debe implementarse como `Service`.
Al igual que `Activity`, `Service` es una clase proporcionada en el tiempo de ejecución de Android y que debe ampliarse.

4.2.5. ContentProvider

Establece una capa que permite a las distintas aplicaciones compartir datos. Con independencia del almacenamiento local que utilicen para sus propósitos, las aplicaciones necesitan declarar `ContentProviders` para poner a disposición de otros procesos los datos que consideren necesarios.

`ContentProvider` puede usar cualquier mecanismo de almacenamiento de datos disponible en la plataforma Android, como archivos, base de datos SQL o incluso un mapa de *Hash* basado en memoria si no se necesita persistencia de datos. Básicamente, `ContentProvider` es una capa de datos que proporciona abstracción para sus clientes y centraliza las rutinas de almacenamiento y recuperación. No se recomienda compartir archivos o bases de datos directamente en Android, algo que también impide el sistema de seguridad de Linux, que evita el acceso a archivos desde una aplicación sin permisos explícitos. Los datos almacenados en `ContentProvider` pueden ser tipos de datos fraccionales como enteros y cadenas. Los proveedores de contenidos pueden gestionar datos binarios como imágenes. Al recuperar datos binarios, es aconsejable devolver una cadena que represente el nombre de archivo que contiene los datos binarios. Si se devuelve un nombre de archivo como parte de una consulta `ContentProvider`, no se debe acceder directamente al archivo, sino utilizar el método `openInputStream` de la clase de ayuda `ContentResolver`.

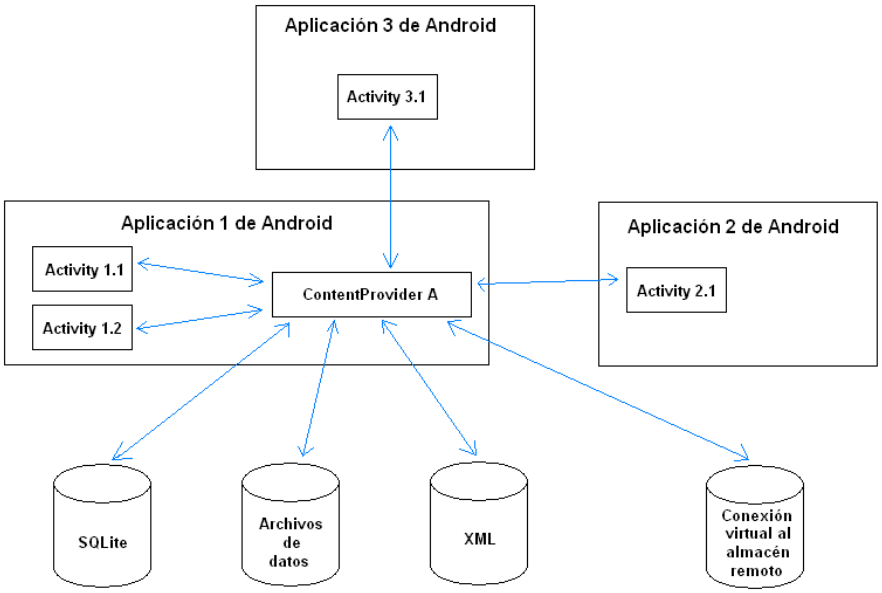

Figura 14. Funciones del Content Provider

A los datos de `ContentProvider` se accede a través del URI Content. Un `ContentProvider` lo define como base pública estática final. Por ejemplo una aplicación puede tener un almacén de datos para gestionar hojas de datos de materiales.

Desde aquí, el acceso a `ContentProvider` es similar al uso de SQL en otras plataformas, aunque no se utiliza una instrucción SQL completa. Se envía una consulta a `ContentProvider`, con las columnas deseadas y clausulas `Where` y `OrderBy` opcionales. Los resultados se devuelven en la clase `Cursor`.

En cierto modo, `ContentProvider` actúa como servidor de base de datos. Aunque una aplicación solamente podría incluir un `ContentProvider` y, en esencia, ser un servidor de base de datos, `ContentProvider` suele ser un componente de aplicaciones Android de mayor tamaño que aloja, al menos, un elemento `Activity`, `Service` y/o `BroadcastReceiver`.

4.2.6. AndroidManifest.xml

En apartados anteriores hemos presentado los elementos comunes de una aplicación Android. Para resumir, hay que tener en cuenta que contiene, al menos, un elemento `Activity`, `Service`, `BroadcastReceiver` o `ContentProvider`. Algunos muestran los `Intent` que desean procesar a través del mecanismo `IntentFilter`. Todos estos fragmentos de información deben combinarse para poder ejecutar la aplicación. El mecanismo que los combina es el archivo `AndroidManifest.xml`. Este archivo se encuentra en el directorio raíz de la aplicación y contiene todas las relaciones de tiempo de diseño y sus `Intent`. Estos archivos actúan como descriptores de implementación de las aplicaciones de Android. Veamos como muestra el siguiente fragmento de código:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.helloandroid"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

5. GPS

Dado que el concepto principal de nuestro proyecto se basa en la tecnología GPS, veremos a continuación algunos conceptos básicos sobre el Sistema de Posicionamiento Global, GPS.

5.1.Introducción

GPS es el acrónimo de Global Positioning System(sistema global de posicionamiento), un sistema formado por una constelación de 24 satélites, llamados NAVSTAR, y 5 estaciones repartidas por la superficie terrestre.

Estos satélites se encuentran en órbitas situadas a 10.900 millas náuticas (20.200 km, aproximadamente) y realizan una circunvalación a la Tierra cada 12 horas. De los 24 en funcionamiento, 21 se encuentran en servicio, mientras que los otros 3 están de reserva. Cada uno de estos satélites emite de manera continua una señal indicando su posición y la hora de sus relojes atómicos.

Las estaciones de tierra se encuentran repartidas en cinco puntos del globo: Hawai, Isla de Ascensión, Diego García, Atolón de Kwajalein y Colorado Springs. Estas estaciones se encargan de vigilar el estado operativo de los satélites y su correcta posición en el espacio. Una de las estaciones cumple las funciones de estación principal y transmite las correcciones a los satélites.

Gracias a este sistema, un usuario puede determinar con muy poco margen de error su posición en la esfera terrestre y la altitud sobre el nivel del mar en las que se encuentra.

5.2.Historia

En 1957, la Unión Soviética lanzó al espacio el satélite Sputnik I, que era monitorizado mediante la observación del efecto Doppler de la señal que transmitía. Debido a este hecho, se comenzó a pensar que, de igual modo, la posición de un observador podría ser establecida mediante el estudio de la frecuencia Doppler de una señal transmitida por un satélite cuya órbita estuviera determinada con precisión.

La armada estadounidense rápidamente aplicó esta tecnología, para proveer a los sistemas de navegación de sus flotas de observaciones de posiciones actualizadas y precisas. Así surgió el sistema TRANSIT, que quedó operativo en 1964, y hacia 1967 estuvo disponible, además, para uso comercial.

Las actualizaciones de posición, en ese entonces, se encontraban disponibles cada 40 minutos y el observador debía permanecer casi estático para poder obtener información adecuada.

Posteriormente, en esa misma década y gracias al desarrollo de los relojes atómicos, se diseñó una constelación de satélites, portando cada uno de ellos uno de estos relojes y estando todos sincronizados con base en una referencia de tiempo determinado.

En 1973 se combinaron los programas de la Armada y el de la Fuerza Aérea de los Estados Unidos (este último consistente en una técnica de transmisión codificada que proveía datos precisos usando una señal modulada con un código de PRN (Pseudo-RandomNoise: ruido pseudo-aleatorio), en lo que se conoció como Navigation Technology Program (programa de tecnología de navegación), posteriormente renombrado como NAVSTAR GPS.

Entre 1978 y 1985 se desarrollaron y lanzaron once satélites prototipo experimentales NAVSTAR, a los que siguieron otras generaciones de satélites, hasta completar la constelación actual, a la que se declaró con «capacidad operacional inicial» en diciembre de 1993 y con «capacidad operacional total» en abril de 1995.

En 2009, este país ofreció el servicio normalizado de determinación de la posición para apoyar las necesidades de la OACI, y ésta aceptó el ofrecimiento.

5.3.Funcionamiento GPS

Como ya se ha mencionado anteriormente, el sistema está formado por 30 satélites y cinco estaciones terrestres, además del receptor del usuario. Estos satélites, a partir de la información incluida en ellos y la que reciben de las estaciones, generan una señal que transmiten a los receptores. Una vez los receptores reciben esta señal, calculan la posición.

La base para determinar la posición de un receptor GPS es la triangulación a partir de la referencia proporcionada por los satélites en el espacio. Para llevar a cabo el proceso de triangulación, el receptor GPS calcula la distancia hasta el satélite midiendo el tiempo que tarda la señal en llegar hasta él. Para ello, el GPS necesita un sistema muy preciso para medir el tiempo. Además, es preciso conocer la posición exacta del satélite. Finalmente, la señal recibida debe corregirse para eliminar los retardos ocasionados.

Una vez que el receptor GPS recibe la posición de al menos cuatro satélites y conoce su distancia hasta cada uno de ellos, puede determinar su posición superponiendo las esferas imaginarias que generan.

Podemos comprender mejor esta explicación con un ejemplo. Imaginemos que nos encontramos a 21.000 km de un primer satélite. Esta distancia nos indica que podemos encontrarnos en cualquier punto de la superficie de una esfera imaginaria de 21.000 km de radio. Ahora, imaginemos que nos encontramos a 24.000 km de un segundo satélite. De este modo, también nos encontramos en cualquier punto de la superficie de esta segunda esfera imaginaria de 24.000 km de radio. La intersección de estas dos esferas generará un círculo que disminuirá las posibilidades de situar nuestra posición. Por otro lado, imaginemos que un tercer satélite se encuentra a 26.000 km. Ahora nuestras posibilidades de posición se reducen a dos puntos, aquellos donde se unen la tercera esfera y el círculo

generado por las otras dos. Aunque uno de estos dos puntos seguramente dará un valor absurdo (lejos de la Tierra, por ejemplo) y puede ser rechazado sin más, necesitamos un cuarto satélite que determine cuál de ellos es el correcto, si bien no es necesario por la razón anteriormente mencionada.

6. Aplicación WiGo

A continuación veamos una breve explicación del funcionamiento de la aplicación a diseñar en este proyecto.

Se trata de una aplicación que nos permitirá conocer los principales atractivos turísticos de una ciudad determinada, indicándonos además algunas rutas a seguir, es decir, sabremos a dónde debemos ir, de ello deriva su nombre; *WiGo*(**W**here**IGo**?).

Nos mostrará rutas que podremos seleccionar bien mediante un método filtrado, según nuestra posición actual, de una distancia determinada, a coche, a pie, etc. O bien podremos ver todas las rutas almacenadas por defecto.

Otra función posible será el de crear nuestra propia ruta, siguiendo nuestros pasos para almacenar esta nueva ruta creada por nosotros mismos.

6.1.Diagrama de bloques

Veamos a continuación el diagrama de bloques donde veremos una explicación básica de las funciones de la aplicación:

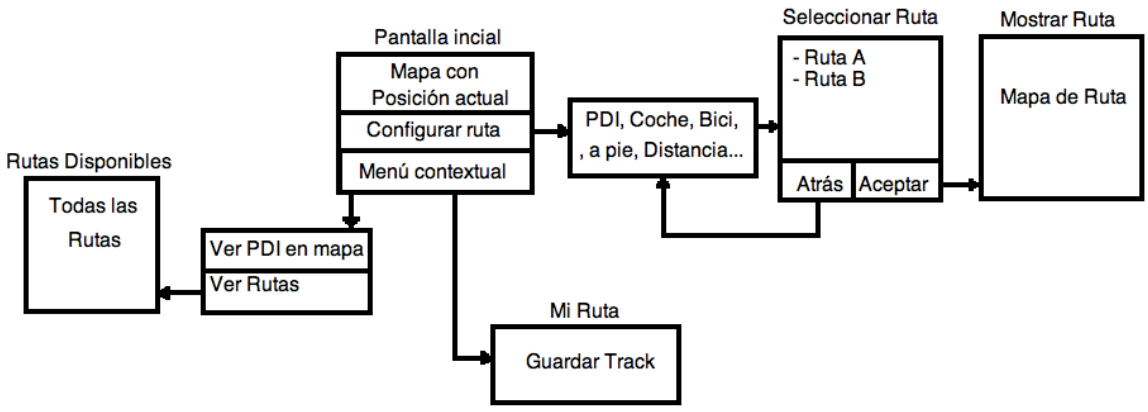

Figura 15. Diagrama de bloques de la aplicación WiGo

7. Gantt

En esta sección veremos el diagrama de Gantt del proyecto y sus diferentes tareas.

7.1.Tabla de tareas

Nombre	Inicio	Fin	Hito
Proyecto Final de Carrera	18/10/10	20/06/11	
Diagrama de Gantt	25/10/10	28/10/10	
PFC1	18/10/10	18/01/11	
Definir proyecto	18/10/10	20/10/10	
Diagrama de bloques de la Aplicación	20/10/10	26/10/10	
Estudio plataformas	28/10/10	5/11/10	
Recopilación Información Android	5/11/10	6/11/10	
Adquisición software	8/11/10	9/11/10	
Aprendizaje Lenguaje Java	8/11/10	18/01/11	
Estudio tecnología GPS	5/11/10	10/11/10	
Programación aplicaciones de prueba	8/11/10	16/12/10	
Redacción memoria PFC1	16/12/10	23/12/10	
Entrega Memória PFC1	23/12/10	24/12/10	*
PFC2	18/01/11	19/01/11	
Diseño aplicación final	19/01/11	20/01/11	
Perfeccionar conocimientos Android	8/11/10	1/01/11	
Perfeccionar conocimiento Java	18/01/11	15/03/11	
Programación aplicación Final	20/01/11	28/04/11	
Depuración Aplicación Final	28/04/11	25/05/11	
Prueba en dispositivo real	20/05/11	4/06/11	
Estudio Normativa Android Market	6/06/11	8/06/11	
Redacción Memoria PFC2	19/01/11	29/04/11	
Entrega Memoria PFC2	29/04/11	2/05/11	*
Preparación Presentación	17/06/11	21/06/11	
Presentación Final Proyecto	21/06/11	22/06/11	*

Tabla 1. Tabla de tareas.

7.2.Diagrama de Gantt

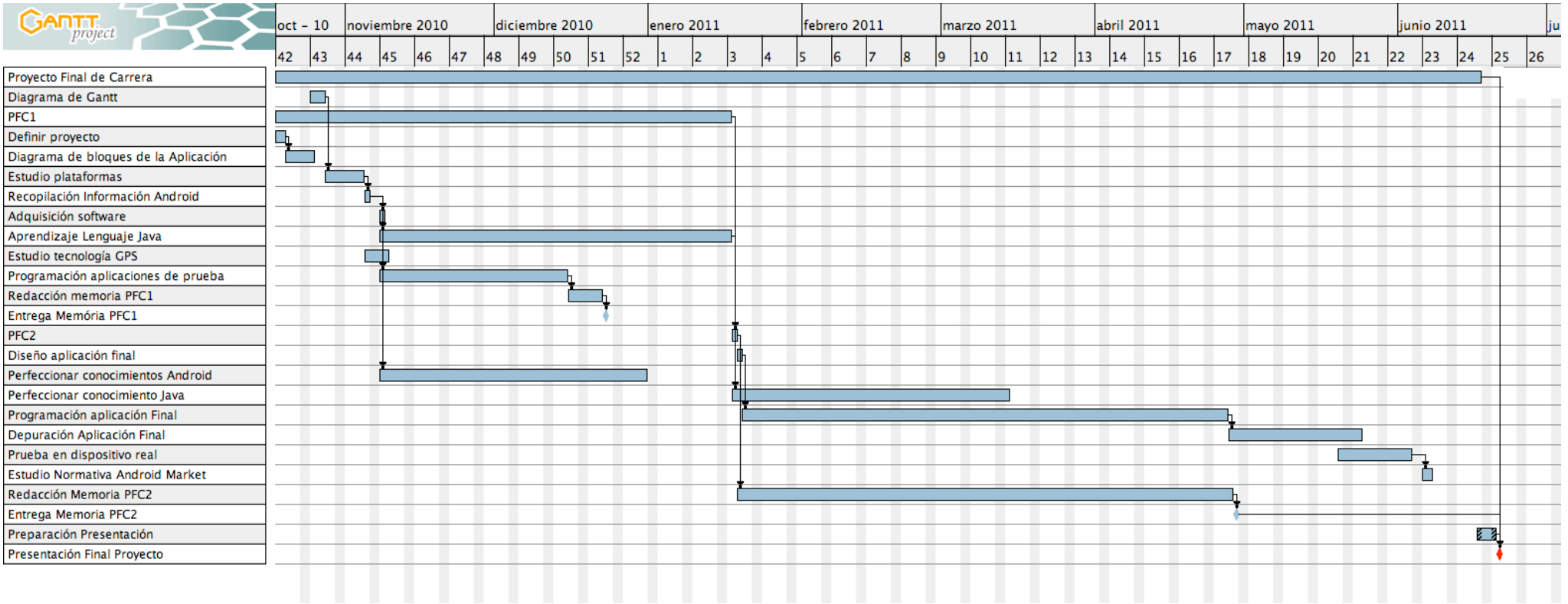

Figura 16. Diagrama de Gantt.

8. Aplicaciones de prueba

En esta sección se expondrán, algunas aplicaciones básicas de prueba. Anteriormente se verá la estructura de una aplicación de Android, tratando de darle un enfoque practico a la explicación.

8.1.Árbol de proyectos

Cuando creamos un nuevo proyecto de Android en Eclipse (IDE recomendado por Google), se genera automáticamente una estructura de carpetas necesaria para poder generar posteriormente la aplicación. Está estructura es común para cualquier aplicación, independientemente del tamaño y la complejidad.
En la siguiente imagen vemos los elementos creado inicialmente para un proyecto nuevo en Android, en este caso la aplicación es el `HelloAndroid`:

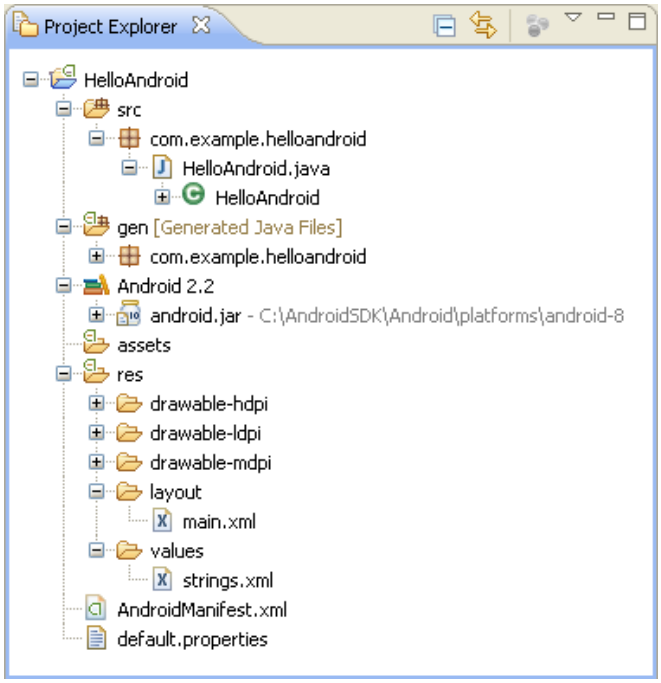

Figura 17. Arbol de proyectos de Eclipse.

8.1.1. Carpeta `/src/`

Contiene todo el código fuente de la aplicación, desde el código de la interfaz gráfica, las clases auxiliares, los `string` correspondientes, etc.… Inicialmente, Eclipse, creará por nosotros el código básico de la pantalla (`Activity`) principal de la aplicación, siempre bajo la estructura del paquete Java definido. En el caso que nos ocupa, la clase principal es la `HelloAndroid.java`:

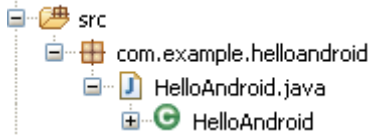

Figura 18. Contenido carpeta src

8.1.2. Carpeta /res/

Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos de deberán distribuir entre las siguientes carpetas:

`/res/drawable`/: Contienen las imágenes de la aplicación. Se puede dividir en `/drawable-ldpi`, `/drawable-mdpi` y `/drawable-hdpi` para utilizar diferentes recursos dependiendo de la resolución del dispositivo.

`/res/layout/`: Contienen los ficheros de definición de las diferentes pantallas de la interfaz gráfica.

`/res/anim/`: Contiene la definición de las animaciones utilizadas por la aplicación.

`/res/menu/`: Contiene la definición de los menús de la aplicación.

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

`/res/xml/`: Contiene los ficheros XML utilizados por la aplicación.

`/res/raw/`: Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

Como ejemplo, para el proyecto `HelloAndroid`, se crean los siguientes recursos para la aplicación:

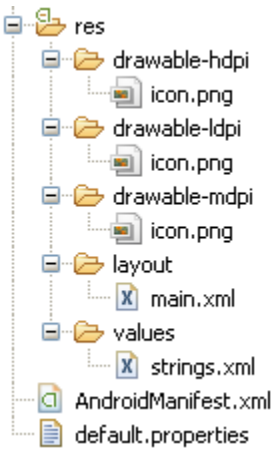

Figura 19. Contenido carpeta res

8.1.3. Carpeta /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.

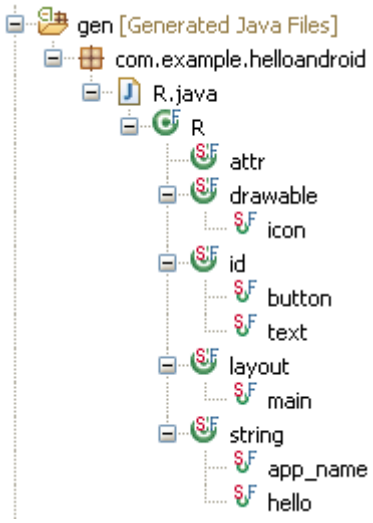

Figura 20. Contenido carpeta gen

El más importante es el que se puede observar en la imagen, el fichero `R.java`, y la clase `R`. Esta clase `R` contendrá en todo momento una serie de constantes con los `ID` de todos los recursos de la aplicación incluidos en la carpeta `res`, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato.

Así, por ejemplo, la constante `R.drawable.icon` contendrá el `ID` de la imagen `icon.png` contenida en la carpeta `/res/drawable/`. Veamos como ejemplo la clase `R` creada por defecto para el proyecto `HelloAndroid`:

```
1 /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2  *
3  * This class was automatically generated by the
4  * aapt tool from the resource data it found.  It
5  * should not be modified by hand.
6  */
7
8 package com.example.helloandroid;
9
10 public final class R {
11     public static final class attr {
12     }
13     public static final class drawable {
14         public static final int icon=0x7f020000;
15     }
16     public static final class id {
17         public static final int button=0x7f050001;
18         public static final int text=0x7f050000;
19     }
20     public static final class layout {
21         public static final int main=0x7f030000;
22     }
23     public static final class string {
24         public static final int app_name=0x7f040001;
25         public static final int hello=0x7f040000;
26     }
27 }
```

Figura 21. Código incluido en la clase R.java

8.1.4. Carpeta /assets/

Contiene los ficheros auxiliares necesarios para la aplicación, como por ejemplo los ficheros de configuración, de datos, etc.…

La diferencia entre los recursos incluidos en la carpeta `/res/raw/` y los incluidos en la carpeta /assets/, es que para los primeros se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Para los segundos, sin embargo, no se generarán ID y se podrá acceder a ellos por su ruta como a cualquier otro fichero del sistema. Usaremos uno u otro según las necesidades de nuestra aplicación.

8.2.Aplicación: HelloAndroid

Todo aprendizaje de un lenguaje de programación implica una serie de programas de prueba. Uno de los primeros programas básicos la aplicación Hola Mundo.
Esta aplicación es muy sencilla y su código fuente sería el siguiente:

8.2.1. Clase: HelloAndroid.java

La clase HelloAndroid.java, se trata de la clase principal de la aplicación. En ella podemos comprobar la estructura de una aplicación:

1. En la línea 1 encontramos la definición del `package` (nombre del proyecto).
2. En la línea 3 / 4 encontramos los `import` necesarios en nuestra aplicación. Un `import` no es más que una instrucción para apuntar a una librería externa a la aplicación.
3. A partir de la línea 6, podemos ver que la clase `HelloAndroid` extiende una `Activity`, en la que vemos que en el método `onCreate` de la aplicación cargará la vista `R.layout.main` de nuestra aplicación.

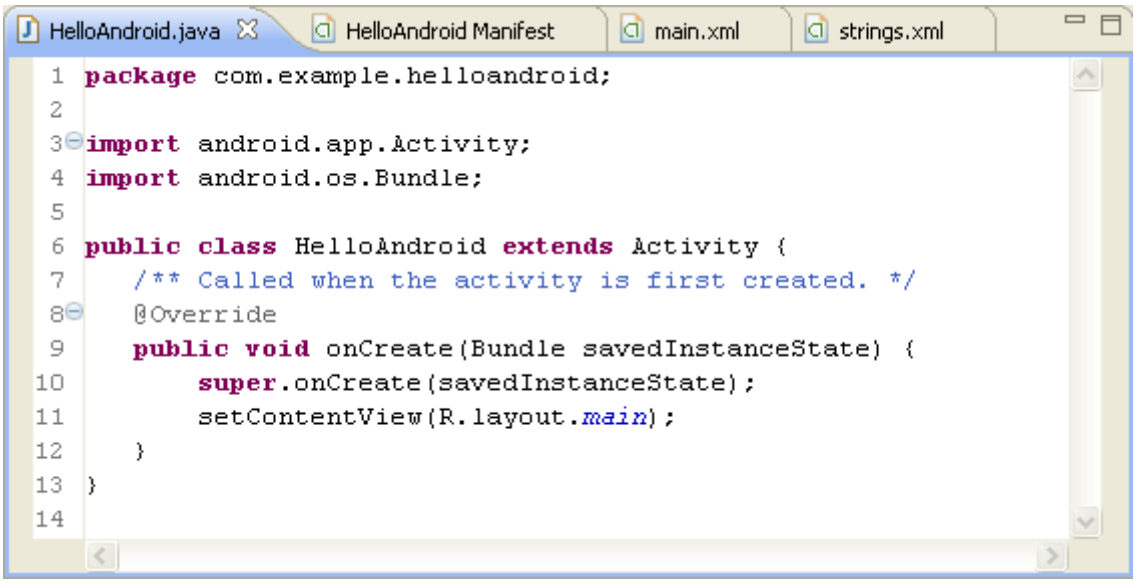

Figura 22. Código incluido en la clase HelloAndroid.java

8.2.2. AndroidManifest.xml

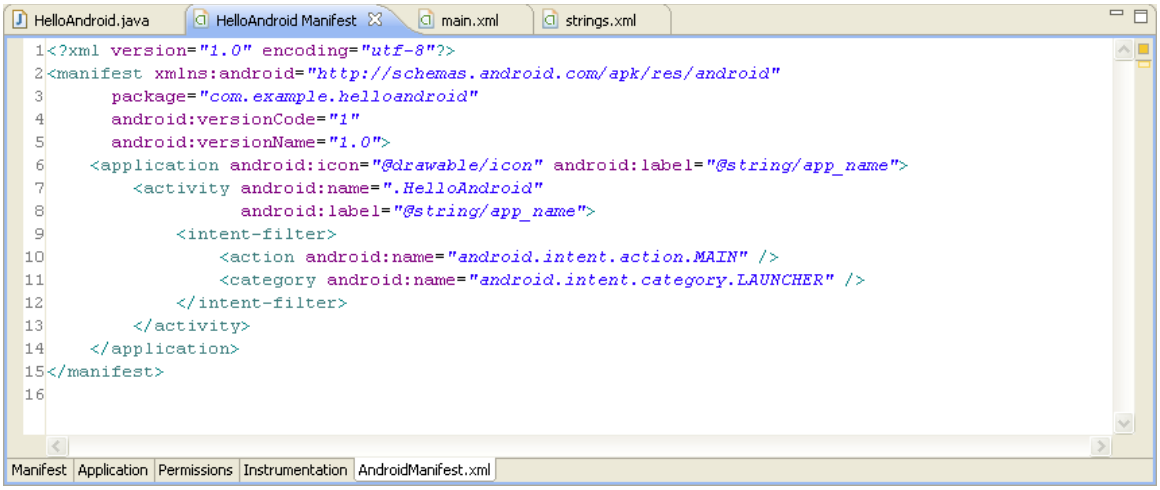

Figura 23. Código incluido en AndroidManifest.xml

Vemos que el elemento `AndroidManifest.xml`, incluye el espacio de nombres obligatorio así como el nombre del paquete de Java que contiene esta aplicación. Dicha aplicación contiene una única `Activity` con el nombre de la clase `HelloAndroid`.

Se puede observar que la sintaxis `@string`, hace referencia a información almacenada en uno de los archivos de recurso (`/res/values/`). En el caso que nos ocupa, el atributo `label`, se obtiene del recurso de cadena `app_name` definido en otro punto de la aplicación.

El único elemento `Activity` de la aplicación, contiene una definición `IntentFilter`, del tipo más habitual utilizado en aplicaciones Android. La acción `android.intent.action.MAIN` indica que es un punto de entrada a la aplicación.

La categoría `Android.intent.category.LAUNCHER` añade esta `Activity` en la ventana de inicio.

Además de los elementos vistos en el fragmento de código anterior, existen otras etiquetas habituales en el `AndroidManifest.xml`:

- La etiqueta <`Service`> representa un servicio: sus atributos son su clase y una etiqueta.

- La etiqueta <`Receiver`> representa un elemento `BroadcastReceiver`.

- La etiqueta <`uses-permission`> indica a Android que esta aplicación requiere determinados privilegios de seguridad. Por ejemplo si una aplicación tiene que acceder a los datos del GPS de un dispositivo, requiere la siguiente etiqueta en su archivo `AndroidManifest.xml`:

```
<!-- GPS permission -->
<uses-permission
      android:name="android.permission.ACCESS_FINE_LOCATION">
</uses-permission>
```

8.2.3. Layout: main.xml

En este `xml` se definen los elementos visuales que componen la interfaz de nuestra pantalla principal y se especifican todas sus propiedades.

Lo primero que nos encontramos es un elemento `LinearLayout`. Los `layout` son elementos no visibles que determinan cómo se van a distribuir en el espacio los controles que incluyamos en su interior. En este caso, un `LinearLayout` distribuirá los controles uno tras otro y en la orientación que indique su propiedad `android:orientation`.

Dentro del `layout` hemos incluido 3 controles: dos etiquetas (`TextView`) y un botón (`Button`). En todos ellos hemos establecido las siguientes propiedades:

- `android:id`. ID del control, con el que podremos identificarlo más tarde en nuestro código. Vemos que el identificador lo escribimos precedido de `@+id`. Esto tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase `R` para dicho control.

- `android:text`. Texto del control. El texto de un control se puede especificar directamente o bien utilizar alguna de las cadenas de texto definidas en los recursos del proyecto (fichero `strings.xml`), en cuyo caso indicaremos su identificador precedido del prefijo `@string`.
- `android:layout_height` y `android:layout_width`. Dimensiones del control con respecto al `layout` que lo contiene. Esta propiedad tomará normalmente los valores `wrap_content` para indicar que las dimensiones del control se ajustarán al contenido del mismo, o bien `fill_parent` para indicar que el ancho o el alto del control se ajustará al ancho o alto del `layout` contenedor respectivamente.

HelloAndroid.java | HelloAndroid Manifest | *main.xml | strings.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical"
              >
    <TextView android:id="@+id/TextView_01"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hola, Soy un TextView sin values.string"
              />
    <TextView android:id="@+id/Hello"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="@string/hello"
            >
            </TextView>
    <Button android:id="@+id/button"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hola, Soy un Boton!!!"
              />
</LinearLayout>

```

Layout | main.xml

Figura 24. Código incluido en main.xml

8.2.4. Values: strings.xml

En el fichero `res/values/strings.xml`, se definen las cadenas de texto que vamos a utilizar en nuestra aplicación. Como hemos visto en la definición de los elementos del `layout`, la propiedad `android:text`, se puede definir directamente, es decir estableciendo un texto fijo en la propiedad. O bien, se puede definir indirectamente, haciendo una referencia con `@string`.

Una posible aplicación práctica de los archivos `strings.xml` podría ser la de hacer un cambio de idioma en la aplicación. Para ello tendríamos diferentes `strings.xml`, uno para cada idioma de la aplicación, y haciendo una selección podríamos ir cambiando el texto.

HelloAndroid.java | HelloAndroid Manifest | main.xml | *strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">"Hola Android! Este es el primer programa en Android!"</string>
    <string name="app_name">Hello, Android! By Gabri &amp;amp; Héctor</string>
</resources>

```

Resources | strings.xml

Figura 25. Código incluido en strings.xml

8.2.5. Screenshot de la aplicación

En este screenshot observamos la aplicación de prueba `HelloAndroid` en el emulador. Vemos los dos `TextView` y un `Button`.

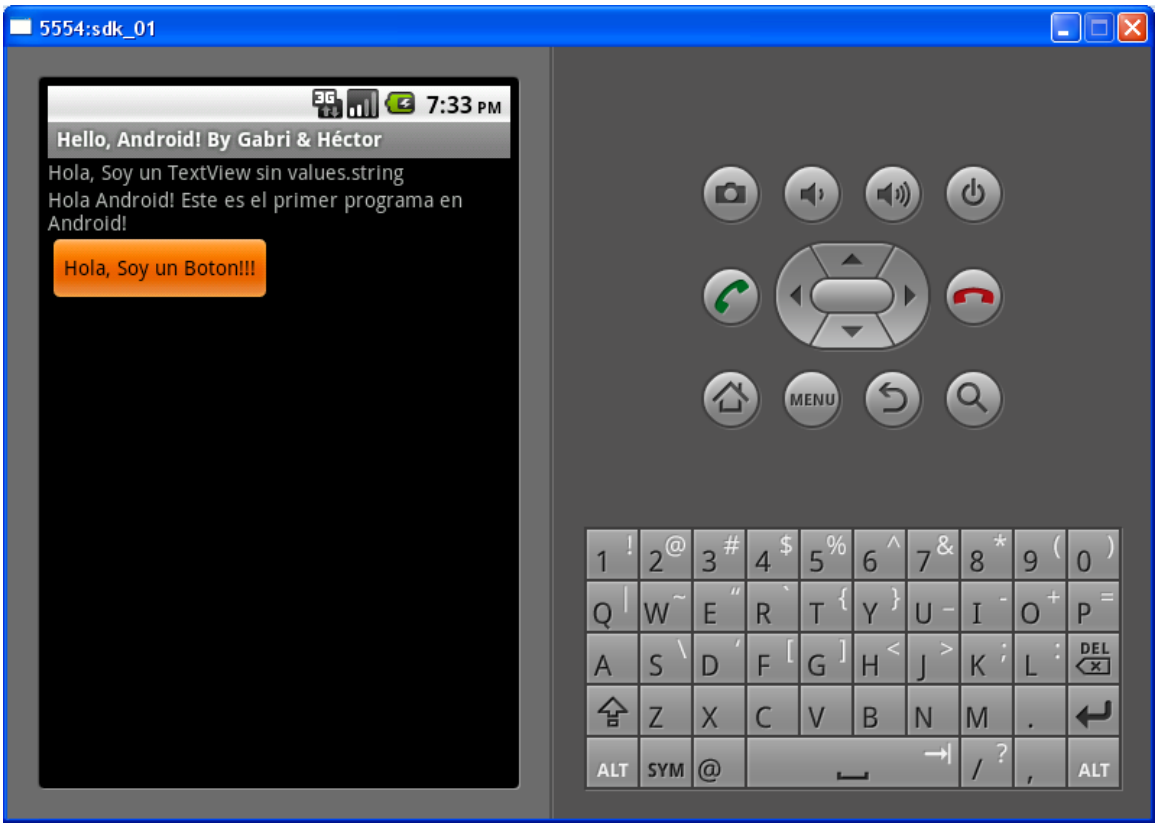

Figura 26. Screenshot de la aplicación Hello Android

8.3.Aplicación: GPS_02

La siguiente aplicación de prueba, ha estado orientada al uso del GPS interno del dispositivo y del tratamiento de los datos proporcionados por el mismo.

Una vez hemos conseguido obtener los datos necesarios del GPS (Latitud y Longitud), los hemos implementado en la cartografía disponible, concretamente en Google maps.

8.3.1. Clase: GPS_02.java

La clase GPS_02.java, se trata de la clase principal (y única) de la aplicación.

```
1  package PFC.GPS_02;
2
3  import com.google.android.maps.GeoPoint;
4  import com.google.android.maps.MapActivity;
5  import com.google.android.maps.MapController;
6  import com.google.android.maps.MapView;
7  import android.content.Context;
8  import android.location.Location;
9  import android.location.LocationListener;
10 import android.location.LocationManager;
11 import android.os.Bundle;
12 import android.widget.Toast;
13
14 public class GPS_02 extends MapActivity
15 {
16     //Se definen los parametros para localización GPS
17     private LocationManager lm;
18     private LocationListener locationListener;
19
20     //Se definen los parametros para el Mapa y el control del mismo
21     private MapView mapView;
22     private MapController mc;
23
24     /** Called when the activity is first created. */
25     @Override
26     public void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.main);  //Crea el Layout
29
30         //Usa el servicio LocationManager para obtener la localización por GPS
31
32         lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
33         locationListener = new MyLocationListener();
34         lm.requestLocationUpdates(
35             LocationManager.GPS_PROVIDER,
36             0,
37             0,
38             locationListener);
39
40         mapView = (MapView) findViewById(R.id.mapview);
41         mc = mapView.getController();
42     }
43     @Override
44     protected boolean isRouteDisplayed() {
45         // TODO Auto-generated method stub
46         return false;
47     }
48     private class MyLocationListener implements LocationListener{
49         @Override
50         public void onLocationChanged(Location loc) {
51             if (loc != null) {
52                 Toast.makeText(getBaseContext(),
53                     "Location changed : Lat: " + loc.getLatitude() +
54                     " Lng: " + loc.getLongitude(),
55                     Toast.LENGTH_SHORT).show();//LENGHT_SHORT = El Toast dura poco tiempo
56                                                //LENGHT_LONG = El Toast dura más tiempo
```

```
                GeoPoint p = new GeoPoint(
                        (int) (loc.getLatitude() * 1E6),
                        (int) (loc.getLongitude() * 1E6));
                mc.animateTo(p);
                mc.setZoom(16);
                mapView.setBuiltInZoomControls(true);
                mapView.setSatellite(true); //Si .setSatellite = True -> Vista Satelite = ON!!!!
                mapView.invalidate();
            }
        }
        @Override
        public void onProviderDisabled(String provider) {
            // TODO Auto-generated method stub
        }
        @Override
        public void onProviderEnabled(String provider) {
            // TODO Auto-generated method stub
        }
        @Override
        public void onStatusChanged(String provider, int status,
            Bundle extras) {
            // TODO Auto-generated method stub
        }
    }
}
```

8.3.2. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="PFC.GPS_02"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <uses-library android:name="com.google.android.maps"></uses-library>
        <activity android:name="GPS_02"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
<!-- Internet y GPS permission -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

Como dato interesante, tenemos los `uses-permission`, son los privilegios de seguridad de nuestra aplicación. En este caso tenemos utilizamos dos, uno para que la aplicación pueda acceder a los servicios de localización y otro para que puede enviar y recibir datos desde internet.

8.3.3. Layout: main.xml

En este `xml` se definen los elementos visuales que componen la interfaz de nuestra pantalla principal y se especifican todas sus propiedades.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:enabled="true"
        android:apiKey="02qb63n9QcVM9uZYz1qLLLOPjnb8FLA11N3k7AA"
        ></com.google.android.maps.MapView>
    <TextView
        android:id="@+id/longitud"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        >
    </TextView>
    <TextView
        android:id="@+id/latitud"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        >
    </TextView>
</LinearLayout>
```

8.3.4. Values: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">GPS_02</string>
</resources>
```

8.3.5. Screenshot de la aplicación

En este screenshot observamos la aplicación de prueba `GPS_02` corriendo en el emulador. Vemos la vista del mapa y el componente Toast (pop-up) indicando la nueva posición del dispositivo.

Figura 27. Screenshot de la aplicación GPS (1)

Figura 28. Screenshot de la aplicación GPS (2)

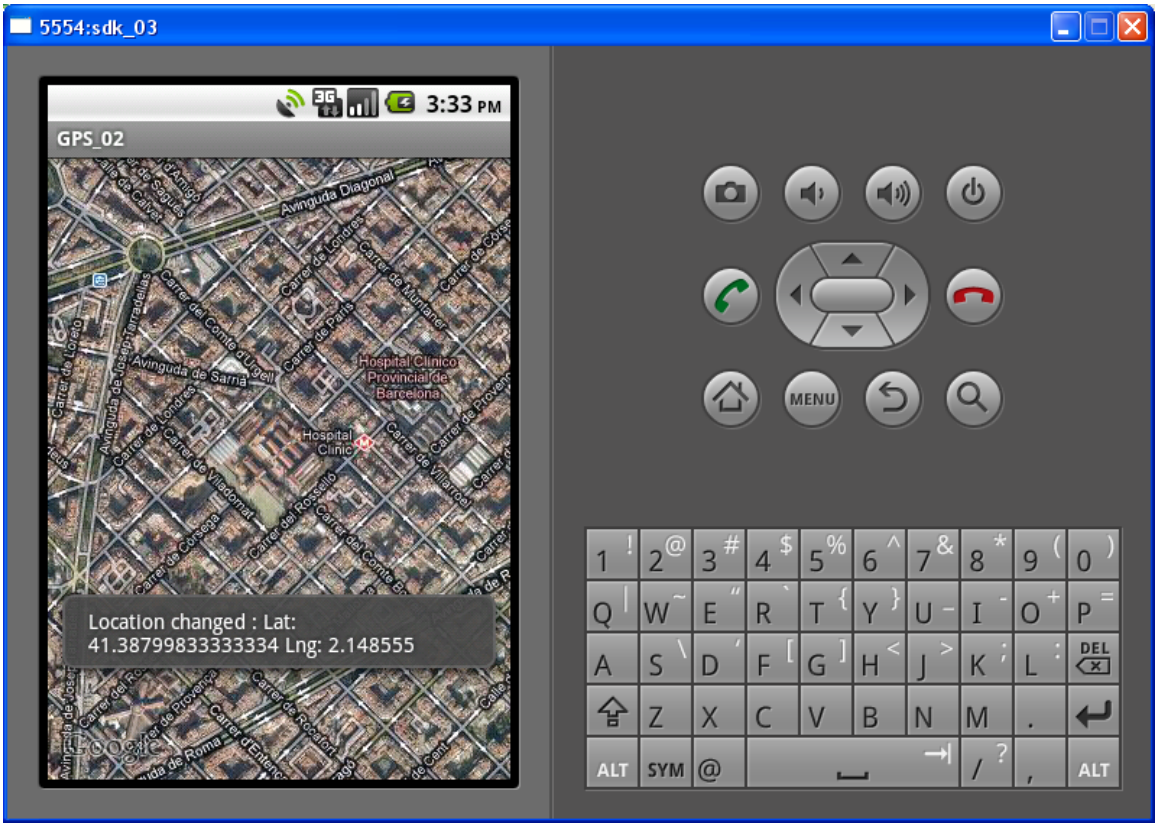

Figura 29. Screenshot de la aplicación GPS (3)

9. Presupuesto

En esta sección se detallará un presupuesto estimado para de los costes de ingeniería y otros costes.

9.1. Costes de Ingeniería

Qt (d)	Qt (d/h)	Referencia	Descripción	Precio unitario (€)	Precio Total (€)
60	2	SWIngTec	Ingeniero técnico, especializado en software	25,28	3.033,60
60	2	SWProg	Programador, implementación de software	14,44	1.732,80
5	3	SWDis	Diseñador grafico, trabajos de diseños de pantallas	14,44	216,60
9	5	DocInResp	Ingeniero responsable, dirección i redacción de proyecto	36,11	1.624,95
9	5	DocInTec	Ingeniero técnico, multiespecializado, ayuda en la redacción del proyecto	25,28	1.137,60
				Sub total	7.745,55
				IVA (18%)	1.394,20
				TOTAL	9.139,75

9.2. Otros costes

Qt	Referencia	Descripción	Precio unitario (€)	Precio Total (€)
0,10	PC	Uso de equipos informáticos genéricos. Su uso no es exclusivo del proyecto	920	92
1	Móvil	Dispositivo móvil, para pruebas	505	505
0,13	Papel	Paquete de 500 folios de papel de 80 g/cm^2. Su uso no es exclusivo del proyecto	4,75	0,59
50	ImpA4	Impresión de folios A4 en color.	0,10	5
1	EnquadA4	Encuadernación memoria A4	5	5
			Sub total	607,59
			IVA (18%)	108,36
			TOTAL	716,95

1.1.Costes totales

Concepto	Precio (€)
Costes de ingeniería	9.139,75
Otros costes	716,95
TOTAL	9.856,70