2011/2012

Autores
Manuel Báez Sánchez
Jorge Cordero Sánchez
Miguel González Pérez
Directores
Prof. Dr. Victoria López
Prof. Dr. Juan Tejada

[CONTROL DE GASTO TELEFÓNICO]

Aplicación Android para control del gasto telefónico individual y grupal

Manuel Báez Sánchez, Jorge Cordero Sánchez y Miguel González Pérez, alumnos matriculados en la asignatura de Sistemas Informáticos, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria como el código, la documentación y/o el prototipo desarrollado en el proyecto *Control De Gasto Telefónico: Aplicación Android para control del gasto telefónico individual y grupal,* todo ello realizado durante el curso académico 2011-2012 bajo la dirección de María Victoria López López, profesora del Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática y Juan Tejada, profesor del Departamento de Estadística e Investigación Operativa de la Facultad de Ciencias Matemáticas.

Manuel Báez Sánchez

Jorge Cordero Sánchez

Miguel González Pérez

Resumen

En este proyecto se ha desarrollado la aplicación CGTel para teléfonos móviles ejecutable desde dispositivos con sistema operativo Android. La principal funcionalidad de la aplicación es ayudar al usuario individual y a empresas o grupo en la gestión del gasto telefónico de sus terminales. La aplicación ofrece otras funcionalidades como la recomendación de tarifa, recomendación del terminal de llamada, detalle del consumo y predicción del coste de la próxima factura mediante análisis de series temporales. Con estas funcionalidades, tanto el usuario individual como las empresas o grupos pueden optimizar el gasto telefónico.

La aplicación cuenta con una base de datos con todas las tarifas de proveedores de telefonía a nivel nacional y con las llamadas realizadas por los usuarios de la aplicación. Los datos sobre las llamadas son actualizados en tiempo real desde los propios terminales.

CGTel ha sido probada con éxito en varios terminales con distintos operadores y diferentes tarifas. Una de las claves de este éxito es la configuración de los terminales incluyendo fecha de facturación, números vips, etc.

Palabras clave: Aplicaciones móviles, Android, factura telefónica, recomendador de tarifa, predicción, optimización de costes.

Abstract

In this project, it has been developed CGTel, a mobile app for Android OS. CGTel helps users and organizations to optimizing final cost on phone bill. The application offers some functionalities like plan recommendation, call recommendation, stats and billing prediction. The application offers other features such as rate recommendation, recommendation calling terminal, detail of consumption and prediction of the cost of the next bill by time series analysis. With these features, both the individual user or groups and companies can optimize telephone costs.

CGTel has a database with every national price plans and details of each call. This database is updated in real time from the own phones.

The application has a database of all phone providers' rates nationwide and details about calls made by users. Data are updated in real time from their own terminals.

CGTel has been successfully tested in several terminals with different operators and different rates. One key to this success is the configuration of terminals including invoice date, VIP numbers, etc.

Keywords: Mobile apps, Android, telephone bill, plan recommender, prediction, cost optimization, smartphones.

Dedicado a nuestros padres por hacerlo posible.
Mil gracias

Prólogo

Este proyecto se inició en el hall de la Facultad de Matemáticas de la UCM, una tarde tras un encuentro casual entre dos profesores que se produjo sin propósito alguno pero que acabó dando lugar a múltiples ideas y proyectos. En efecto, las ideas pueden nacer en cualquier momento y en cualquier lugar pero lo maravilloso es poder llevarlas a cabo. Gracias a Jorge, a Manuel y a Miguel, autores de esta memoria y de este proyecto, una de esas ideas se ha convertido en una realidad.

El camino no ha sido siempre fácil pero hemos tenido la suerte de formar un equipo bien estructurado con muchas ganas de trabajar y con una actitud extraordinaria ante cualquier escollo encontrado en el camino. Miguel, Manuel y Jorge han sido capaces de desarrollar una aplicación útil, correcta y atractiva. Para ello, no sólo han tenido que estudiar nuevos modelos matemáticos necesarios sino que también han demostrado su habilidad para adaptar y mejorar dichos métodos al problema concreto sobre predicción de costes que resuelve su aplicación. También han demostrado gran capacidad de trabajo en grupo al dividirse las tareas de forma adaptable a cada perfil y manteniendo la comunicación constante entre los miembros del grupo, como si se tratase de un proyecto profesional de ingeniería de software. Han estudiado multitud de vías para solucionar los problemas encontrados y siempre se han mostrado dispuestos a realizar los cambios y mejoras propuestos por los directores del proyecto.

El resultado es CGTel, una aplicación Android para controlar y optimizar el gasto telefónico de grupos con distintos proveedores de telefonía. Una aplicación que cualquiera puede utilizar de forma individual o en grupo optimizando el gasto en telefonía de una familia o empresa. A éste último nivel, se trata de un software de aplicación que resuelve una necesidad de negocio específica, que procesa datos reales sobre las tarifas de telefonía y que por ello facilita la gestión de los recursos de las empresas, controlando en tiempo real el gasto y facilitando de este modo la toma de decisiones. Por iniciativa de Manuel, Jorge y Miguel, la aplicación también incluye diversas funcionalidades interesantes, como el recomendador de tarifas, muy útil en mercados altamente competitivos como es el nuestro en estos momentos. En definitiva, un trabajo bien hecho y de una gran utilidad para cualquier tipo de usuario.

Victoria López y Juan Tejada

Índice

1. Introducción

Idea original y panorama actual

Hoy en día la evolución de las tecnologías móviles, especialmente con la llegada de los teléfonos inteligentes (Smartphone en inglés), ha fomentado la aparición de multitud de aplicaciones destinadas a todo tipo de público. Así, surge una nueva oportunidad en el mercado del desarrollo de aplicaciones, el desarrollo para iPhone y Android, siendo estas las plataformas más extendidas en el mercado. Actualmente en nuestro país, el sector del desarrollo para terminales móviles y la publicidad en los denominados *markets* supone unos ingresos del 18% más que el año pasado, cifras que para los consumidores suponen un gasto cercano a los 1.257,7 millones de euros (datos previstos para 2012).

Cada vez es más frecuente disponer de un terminal de los señalados anteriormente que disponga de una conexión de datos que nos permite estar conectado a Internet desde cualquier punto. Desde un público que necesita ocasionalmente mirar su correo electrónico o hacer alguna consulta hasta los adictos a las redes sociales, todo el mundo tiene a su disposición un Smartphone con todas estas aplicaciones a su disposición.

Es en este panorama donde surge la idea de CGTel. El nombre de CGTel son las siglas de *Control del Gasto Telefónico.* Las operadoras de telefonía móvil ofrecen una amplia gama de combinaciones de tarifas planas de voz y datos. En este ámbito, por el desconocimiento del propio usuario, es bastante habitual que nuestra tarifa de voz contratada sea excesiva. Dado que disponemos de la conexión de datos del terminal podemos usar la información que compartamos entre los terminales para intentar, en la medida de lo posible, reducir el consumo en nuestra factura telefónica.

Sistemas operativos móviles

Los sistemas operativos móviles que podemos encontrar en el mercado están fuertemente ligados a la marca del terminal ante el que nos encontremos. Actualmente, podemos destacar cuatro entre los más utilizados: Android, BlackBerryOS, iOS, Windows Phone 7.

Android es un sistema operativo móvil basado en Linux enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos. Está desarrollado por la Open Handset Alliance[1], la cual es liderada por Google. Tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A fecha de hoy, se han sobrepasado las 400.000 aplicaciones disponibles para la tienda de aplicaciones oficial de Android: Google Play (antes conocido como *Market*).

El anuncio del sistema Android se realizó el 5 de noviembre de 2007. Google liberó la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto. Al tratarse de una comunidad fundamentalmente

libre, muchas son las marcas de móviles que han optado por sacar a la venta sus terminales con el sistema operativo de Google.

BlackBerryOs es un sistema operativo móvil desarrollado por Research In Motion (RIM) para sus dispositivos BlackBerry. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en distintos terminales, particularmente la trackwheel, trackball, touchpad y pantallas táctiles. La principal ventaja respecto a sus competidores, diseñado únicamente para sus propios terminales, el BlackBerry OS está claramente orientado a su uso profesional como gestor de correo electrónico y agenda. Permite sincronizar los dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de Microsoft Exchange Server así como con Lotus Notes y Novell GroupWise.

iOS es un sistema operativo móvil de Apple. Originalmente desarrollado para el iPhone, siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV. Apple, Inc. no permite la instalación de iOS en hardware de terceros. La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles. Los elementos de control consisten de deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida, en definitiva el sistema fue creado para las plataformas portátiles de Apple.

Windows Phone 7 es el último sistema operativo móvil desarrollado por Microsoft, como sucesor de la plataforma Windows Mobile. Está pensado para el mercado de consumo generalista en lugar del mercado empresarial por lo que carece de muchas funcionalidades que proporciona la versión anterior. Este sistema no es compatible con las aplicaciones desarrolladas para Windows Mobile 6. Con Windows Phone 7 Microsoft ofrece una nueva interfaz de usuario e integra varios servicios en el sistema operativo. Junto con Android, son muchas las compañías de Smartphones que están optando por sacar nuevas gamas de terminales que dispongan del sistema operativo de Microsoft, entre las más destacadas se encuentra Nokia.

Como hemos comentado en el punto anterior, los líderes en el mercado, hasta la fecha, son los sistemas iOS de Apple y Android de Google.

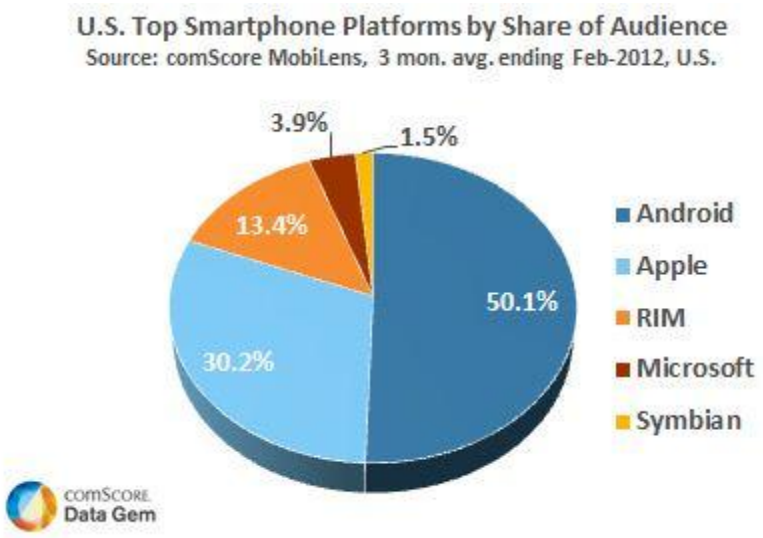


Figura 1. Ventas de Smartphone en Estados Unidos, Febrero 2012. [2]

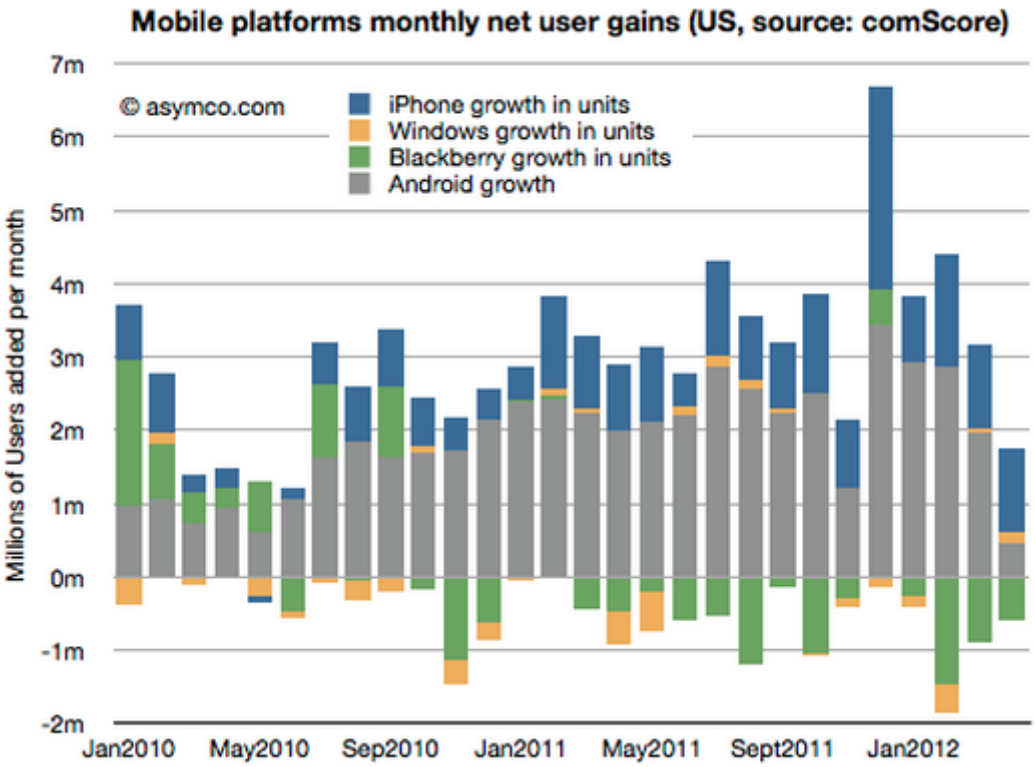


Figura 2. Crecimiento de los distintos sistemas operativos móviles en los dos últimos años. [3]

2. Estado del arte

Introducción

Para llevar a cabo el proyecto hemos realizado un amplio estudio sobre el estado del arte. Se ha analizado el estado de los distintos sistemas operativos móviles así como de las aplicaciones más destacadas en cuanto a la gestión del gasto telefónico. Además se ha realizado un estudio sobre los planes de pago ofertados por las compañías telefónicas. A continuación incluimos el resultado de dicho estudio junto con decisiones y conclusiones que obtuvimos. Analizamos los sistemas operativos para Smartphones más destacados para ver en cuál encajaba mejor nuestra idea.

Android

Teniendo en cuenta que el desarrollo de la aplicación no estaba orientado a un sistema operativo concreto, consideramos la posibilidad de una portabilidad de dicha aplicación a otros sistemas. Android fue considerado como la mejor opción en este aspecto. En comparación con iOS, Android ofrece mayor comodidad a los desarrolladores, siendo utilizado por numerosas empresas para crear aplicaciones, las cuales una vez terminadas y revisadas son portadas a otros sistemas operativos. Esta característica de Android se aprecia en el número de aplicaciones contenidas Google Play Store[4]. Además de la principal tienda de aplicaciones Android existen otras tiendas como AppBrain[5], Amazon Store[6], GetJar[7], AppsFire[8] o SlideMe[9]. A continuación presentamos las distintas tiendas de aplicaciones.

Google Play Store

Antes llamada Android Market se trata de la tienda de aplicaciones oficial de Android. El cambio de nombre coincidió con una ampliación de servicios ofertados por Google, incluyendo una tienda de música, un servicio de alquiler y venta de películas y un una tienda de libros electrónicos con cerca de 3 millones de ejemplares.

Al ser la tienda oficial es muy común relacionarla con el éxito de Android. En la gráfica mostrada en la Figura 3 se puede apreciar un claro aumento de descargas de aplicaciones. Actualmente se superan los 15.000 millones de descargas.

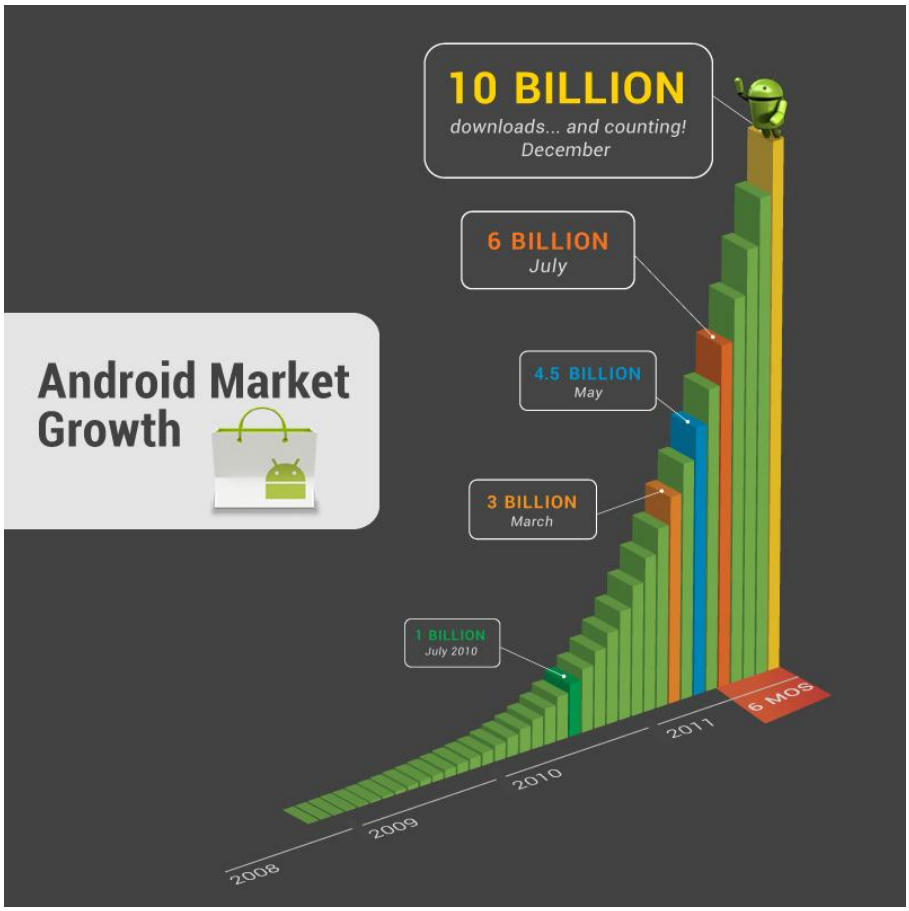


Figura 3. Número de descargas en el Android Market.

Google Play Store ha experimentado una gran evolución desde su creación como se puede apreciar en la gráfica anterior. Actualmente el número de aplicaciones es superior a 500.000.

Hay que destacar que hay un amplio número de aplicaciones de baja calidad por no efectuarse un control de calidad sobre las mismas. En la Figura 4, la Figura 5 y la Figura 6 pueden apreciarse el elevado número de aplicaciones de baja calidad disponibles en Google Play Store.[10]

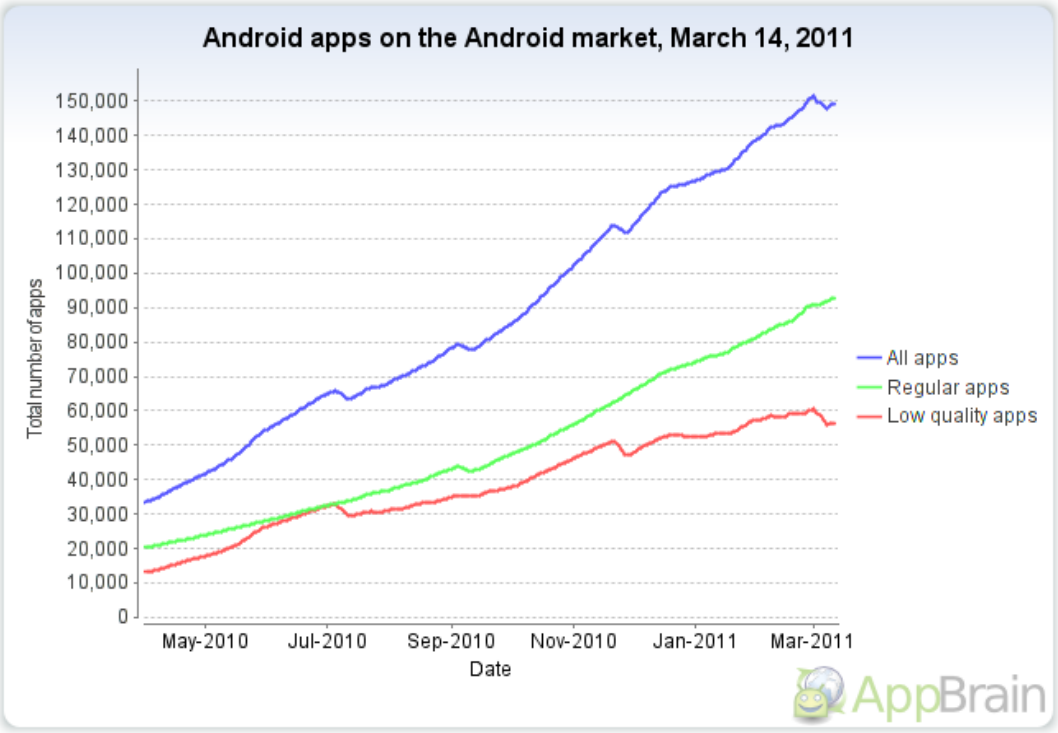


Figura 4. Número de aplicaciones disponibles en Google Play Store según su calidad, Marzo de 2011.

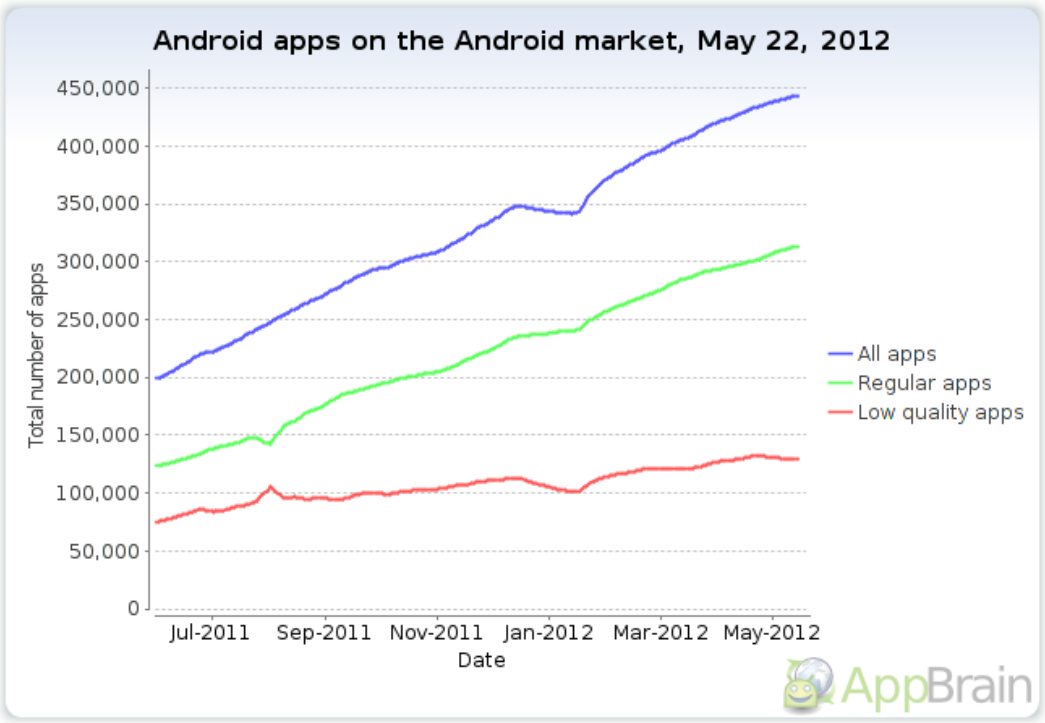


Figura 5. Número de aplicaciones disponibles en Google Play Store según su calidad, Mayo de 2012.

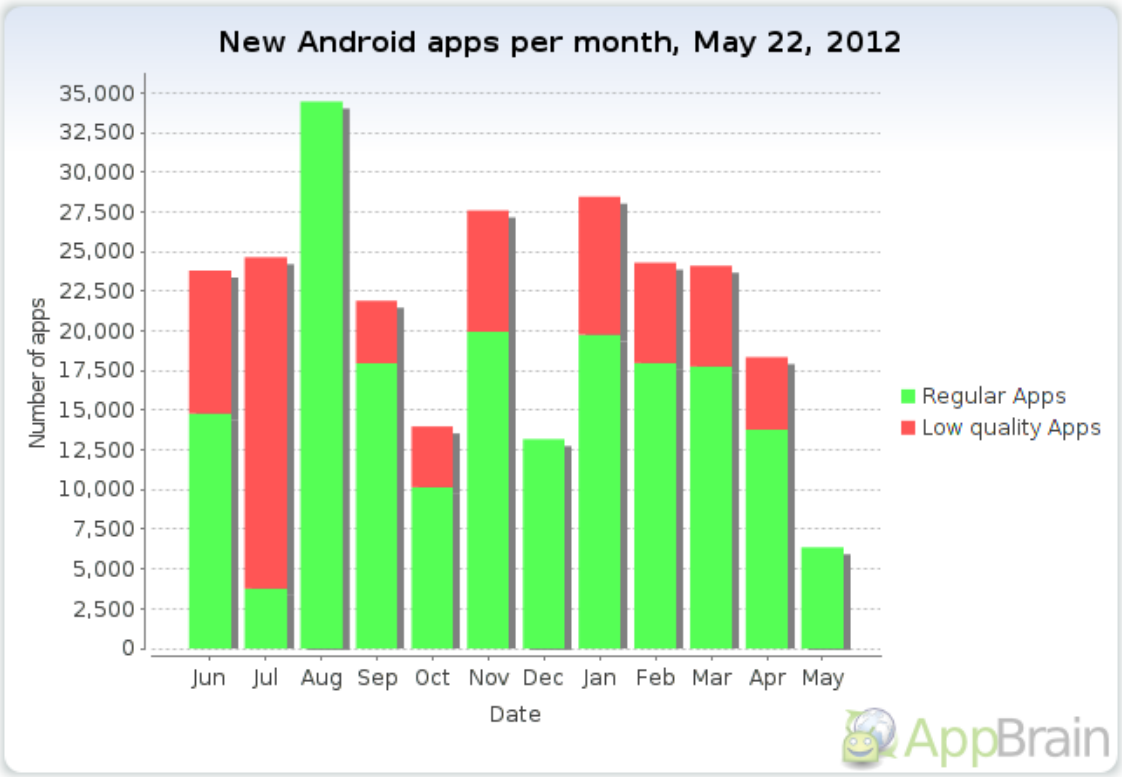


Figura 6. Porcentaje mensual de nuevas aplicaciones de baja calidad.

AppBrain

AppBrain es una de las tiendas de aplicaciones alternativas a Google Play Store más destacadas y antiguas. La oferta de aplicaciones es similar a la de Google Play con la ventaja de una distinta organización de las aplicaciones. Se pueden buscar aplicaciones haciendo uso de numerosos filtros y categorías ofrecidas.

Las aplicaciones pueden buscarse por sus características, permitiendo por ejemplo buscar aplicaciones que permiten la instalación en la tarjeta SD del dispositivo. Entre otras opciones es posible filtrar por países, edad de los usuarios o valoración.

Figura 7. Aplicación AppBrain para Android.

AppBrain fue de las primeras tiendas de aplicaciones en ofrecer instalación de aplicaciones desde el navegador del ordenador. Más tarde fue implementada esta posibilidad en el Android Market (Google Play Store). Esta es una evidencia más de como la variedad de Android lleva a una mejora de los servicios debido a la competencia.

Amazon Appstore

Amazon Appstore actualmente disponible en los Estados Unidos con una oferta de aplicaciones inferior a sus competidoras. Actualmente dispone de cerca de 40.000 aplicaciones. Al contrario que en otras tiendas, Amazon tiene una mayor oferta de aplicaciones de pago y cada día suelen ofertar una aplicación gratuita.

En Mayo de 2012 ha incluido una importante mejora en su tienda de aplicaciones para Android, que la ha hecho única. Ofrece la posibilidad de probar aplicaciones sin necesidad de instalarlas. Si bien esta opción ya estaba disponible

desde su web, han incluido la posibilidad de probarlas desde su aplicación para Android., como se puede apreciar en la Figura 8. Siendo un gran paso frente a sus competidoras a pesar de no estar disponible para todos los dispositivos. Suponiendo un gran avance de cara a la distribución de aplicaciones, sobre todo las de pago, ya que los usuarios podrán probarlas sin necesidad de pagar.

Figura 8. Aplicación Amazon AppStore para Android.

Actualmente en tiendas como Play Store si un usuario quiere probar una aplicación de pago debe comprarla y si no está satisfecho dispone de un tiempo de prueba para devolver la aplicación. Dicho tiempo se redujo a comienzos del 2012 a tan sólo 15 minutos, tiempo muy limitado teniendo en cuenta que gran número de aplicaciones deben descargar datos tras su instalación para poder funcionar (por ejemplo aplicaciones GPS y algunos videojuegos). Para evitar este problema Google aumentó también el espacio disponible para los desarrolladores desde los 50 MB hasta los 4 GB, de esta forma los desarrolladores pueden incluir más datos en los instaladores de sus aplicaciones para evitar descargas tras la instalación. Google recibió varias demandas [11] respecto a estos movimientos, lo que producirá cambios en su política de devoluciones. Tras la salida de Test Drive [12] en la Amazon App Store, Google se verá forzado a incluir mejoras.

Este es otro ejemplo de como las distintas alternativas para distribuir aplicaciones en Android suponen una ventaja a la hora de elegir la que más se adecúe a las necesidades de los desarrolladores y/o usuarios.

GetJar

GetJar proporciona aplicaciones para distintos sistemas operativos móviles: Android, iOS, Windows Mobile, BlackBerry y Symbian. Al contrario que AppBrain esta tienda de aplicaciones ofrece aplicaciones que no están disponibles en Play Store. La instalación de aplicaciones mediante GetJar es más complicada que en sus competidoras.

Figura 9. Aplicación GetJar para Android.

Figura 10. Página web de GetJar.

SlideMe

SlideMe está destinado a pequeños y medianos desarrolladores. Una característica destacable es el foro del que dispone, orientado a consumidores y desarrolladores, facilitando el reporte de errores (bugs) en aplicaciones y el desarrollo. Podemos ver en la Figura 11 su gestor de aplicaciones y como muestran las aplicaciones punteras.

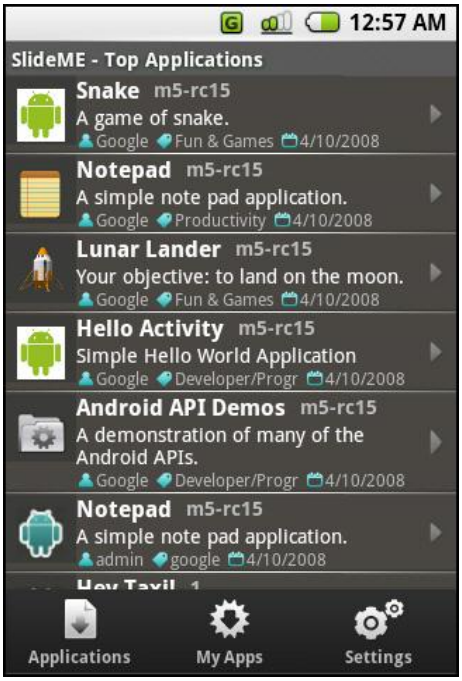


Figura 11. Aplicación SlideMe para Android.

AppsFire

AppsFire es similar a AppBrain. Ofrece una alternativa a Play Store organizando la aplicaciones facilitando a los usuarios la búsqueda. Dispone de una buena aplicación para para Android que optimiza los resultados de las búsquedas. La interfaz de dicha aplicación, en la Figura 12, permite el fácil acceso a aplicaciones recomendadas según las estadísticas de uso del usuario.

Figura 12. Captura de la aplicación appsfire para Android.

Gestión de datos

Con la aparición de los Smartphones se ha difundido el uso de las conexiones de datos en este tipo de dispositivos. Esto ha llevado a la creación de tarifas de datos por parte de las compañías telefónicas.

Tras la inclusión en Android 4.0 de esta nueva utilidad, el desarrollo de una aplicación para el control de datos tiene difícil salida en el mercado, podemos ver en la Figura 13 que no están en el top 10. Es por esto que CGTel no se centra en el control de datos. Además las principales tarifas ofrecidas por las operadoras en España limitan la velocidad de los datos una vez superado el límite, pero no añaden un coste si esto sucede. A continuación mostramos el estudio de las aplicaciones más destacadas en la gestión del consumo de datos.

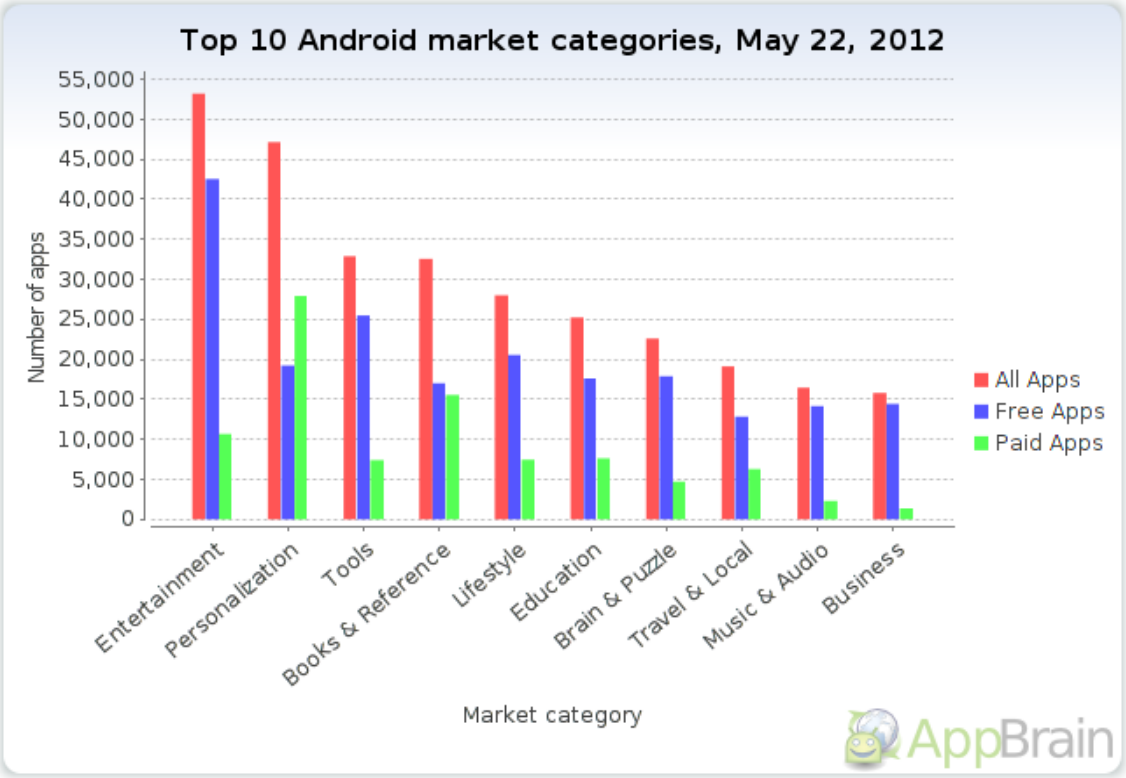


Figura 13. Distribución de las aplicaciones del Android Market según su categoría.[13]

Gestor de datos en Ice Cream Sandwich

En la nueva versión del sistema operativo Android se ha incluido una aplicación nativa para el control de datos. Esto viene a representar la importancia que tiene para los usuarios de Smartphones el control del gasto en sus dispositivos. A esta aplicación se accede desde los ajustes y permite monitorizar el acceso a internet de las aplicaciones.

Para el control de datos existen numerosas aplicaciones de terceros pero la precisión no siempre es buena. Por ello en este campo es difícil competir con una aplicación nativa integrada en el sistema operativo. Además se incluye la posibilidad de limitar el consumo dependiendo del plan de cada usuario. Para cada aplicación se puede seleccionar límites específicos y restringir el consumo a redes Wifi.

Figura 14. Captura de la aplicación nativa de Android 4.0 para la gestión de datos.

Este tipo de opciones son suficientes para una gestión básica del consumo de datos. Pero existen numerosas alternativas que ofrecen mayores posibilidades de configuración.

Onavo

Onavo[14] ofrece alguna característica más respecto a la aplicación nativa de Ice Cream Sandwich. Se encuentra disponible de forma gratuita en Play Store aunque no ofrece las mismas características que para iOS. Su principal punto fuerte es la compresión de los datos de navegación en sus servidores antes de ser enviados al dispositivo móvil. Esta característica supone una drástica reducción en el consumo de datos, pero no está disponible en la aplicación Onavo para Android. A pesar de ello Onavo recibe buenas críticas de los usuarios, reduce el consumo de datos y monitoriza aplicaciones permitiendo limitar el acceso a internet dependiendo de la aplicación o el tipo de conexión.

Se puede apreciar en la Figura 15 como con una intuitiva interfaz se obtienen resúmenes de los datos ahorrados con el uso de la aplicación y los datos de la monitorización de aplicaciones.

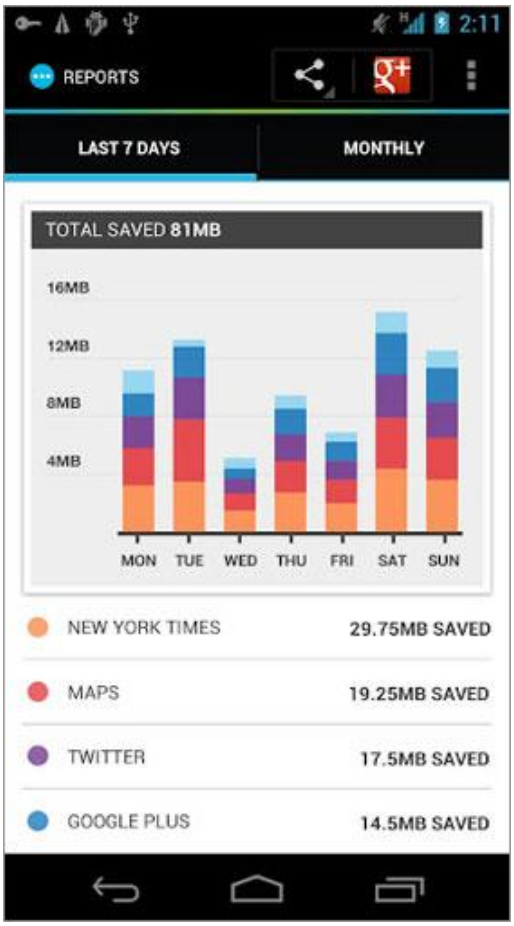


Figura 15. Captura de la aplicación Onavo. [15]

My Data Manager

My Data Manager no ofrece tantas características como Onavo. Está centrada en la monitorización del consumo y no ofrece muchas características de bloqueo o limitación del consumo de datos. Las opciones que ofrece se limitan a bloquear las conexiones 3G una vez superado un límite de datos diario o mensual. En la Figura 16 podemos ver el control exhausto que hace del uso de la red móvil [16].

Figura 16. Captura de la aplicación My Data Manager.

Los resultados de la monitorización se ajustan bastante a los reales. Hay que destacar que no siempre se suelen obtener resultados precisos en el control de los datos y este puede ser uno de los puntos fuertes de My Data Manager.

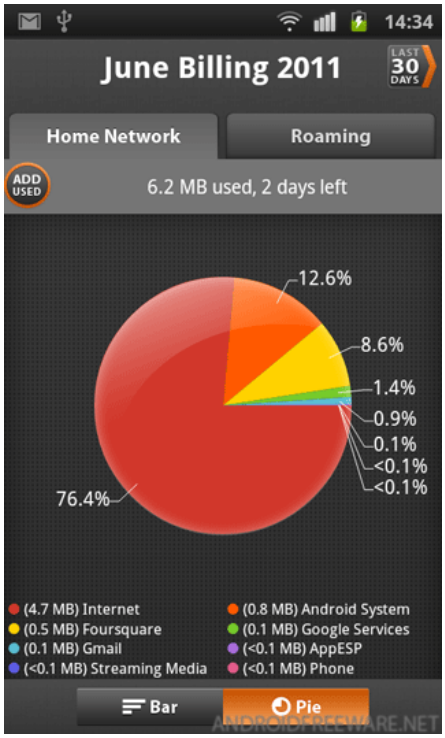


Figura 17. Gráfico con un resumen del consumo de datos generado por la aplicación My Data Manager.

La ventaja de aplicaciones limitadas a la monitorización del consumo de datos es que no tienen ajustar su configuración en base a los planes de facturación ofertados por las compañías. Basta con pedir al usuario el límite de su plan y el periodo de facturación. My Data Manager dispone de una configuración sencilla y eficaz, como se muestra en la Figura 18 consta de 3 sencillos pasos. La monitorización de datos en My Data Manager comienza tras la instalación de la aplicación por lo que hasta el primer día de facturación la aplicación no tiene utilidad ninguna.

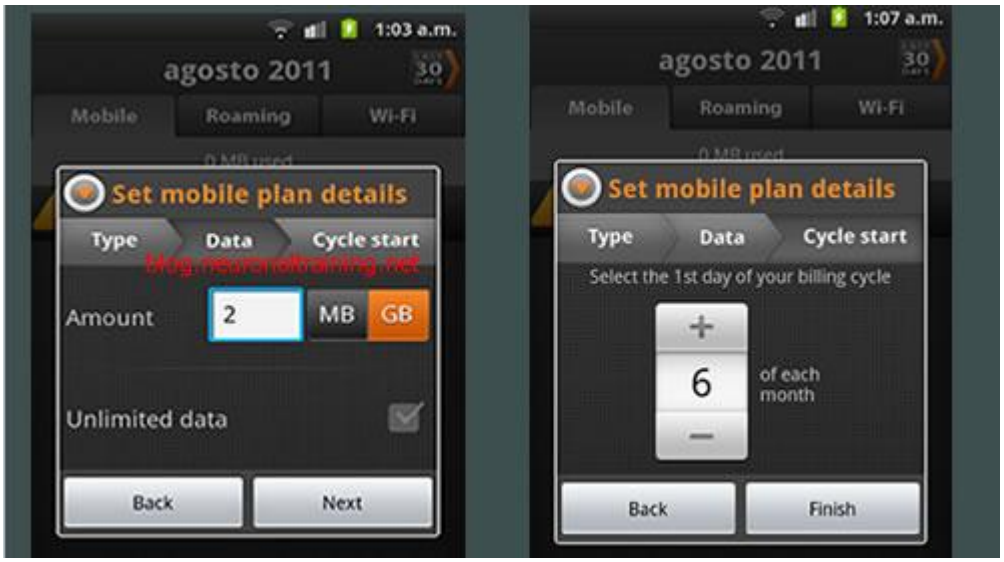


Figura 18. Ventanas de configuración del plan de datos en My Data Manager.

Control de llamadas

Existen numerosas aplicaciones para el control de gasto de las llamadas sin llegar a destacar ninguna. El gran problema de las aplicaciones existentes es la gran variedad de tarifas. Esto ha hecho que las aplicaciones más importantes sean de código abierto, permitiendo a desarrolladores y colaboradores añadir nuevas tarifas. Los problemas a los que se enfrentan este tipo de aplicaciones son sencillos pero tediosos. Las compañías no se limitan a ofertar planes de facturación sino que constantemente los modifican o incluyen ofertas especiales, descuentos porcentuales y otros detalles que influyen en el cálculo del gasto en un terminal. Los planes de facturación se complican con determinadas tarifas en las que las compañías ofrecen distintos precios en función de la compañía a la que pertenezca el receptor de la llamada.

Aplicaciones similares

En este apartado destacamos myPlan y ControlGasto. Ambas son proyectos de código abierto lo que ha contribuido a que ofrezcan una amplia gama de tarifas a nivel nacional. Ambas aplicaciones son muy limitadas y el funcionamiento no suele ser el esperado. Normalmente por la configuración de las tarifas los datos mostrados no se corresponden con los reales.

myPlan

La aplicación myPlan ofrece comparación de facturas para distintos planes de facturación de en base a las llamadas realizadas en el periodo de facturación actual, Figura 19. Muestra el coste de las llamadas realizadas para la tarifa seleccionada de una lista.

Figura 19. Ventana principal y listado de tarifas con su resumen de la factura en la aplicación myPlan.

El punto fuerte de myPlan es la amplia oferta de tarifas de que dispone.

Compañía	Tarifas
EroskiMovil	Conekta, Contigo, ContratoSin, HablaA6.
Jazztel	Habla Navega 5, Jazzmovil 5, Tarifa plana 100 minutos, 200 minutos, 300 minutos, 400 minutos, 500 minutos.
MásMovil	Tarifa 3, Tarifa 5, Tarifa 8, Ligth 1GB, Light 300MB, Light 500MB, Movil Today, Tarifa Plana 200, Sin 1GB, Sin 300MB, Sin 500MB, Tarifa Total, Tarifa Líder.
Movistar	Contrato móviles, Contrato simple, Contrato tiempo libre, Planazo Móviles Movistar, Planazo Tiempo Libre, Movistar 8, Habla, Habla 15, Habla 25, Habla 35, Habla 45, Habla 75, Habla Ocio, Habla Ocio 15, Habla y Navega 21, Habla y Navega 30, Habla y Navega 40, Habla y Navega 50.
Orange	Ardilla 6, Ardilla 8, Ardilla 9, Ardilla 12, Ardilla 15, Básico 6, Delfín 20, Delfín 30, Delfín 32. Delfín 40, Delfín 42, Delfín 59, Delfín 79, León 24, León 25, León 30, León 32, León 49, Mini Básico, Panda 15, Panda 20, Panda 22, Panda 26, Pingüino.
PepePhone	Tarifa 6cent, Tarifa 7cent, Caballo Loco, Cotorra Pepe, Einstein, Lobo Feroz, Lobo&Cordero, Movilonia 9, Movilonia VIP, Pulpo Pepe, Ratoncito&Elefante, Shurperro, Sin Animal, Tarifa Menguante.
Simyo	Tarifa 0 y 8 céntimos, Tarifa 3 céntimos, Tarifa 5 céntimos.
Vodafone	A mi Aire 90x1 24h, A mi Aire 90x1 a Todos, A mi Aire Simple, Contrato 1, Diminuto 8, Mini 24h, Mini a Todos, Plana 24h, Plana a Todos, Super Plana Mini, Super Tarifa Plana, L, M, S, XL, XS, XS6, XS8, @L, @M, @S, @XL, @XS, @XS8.
Yoigo	La del 0, La del 4, La del 6, La del 8, La megaplana 20, La megaplana 30, La megaplana 40, La megaplana 55, La plana 10, La plana 20, La plana 30, La plana 55.

Tabla 1. Resumen de las tarifas contenidas en la aplicación MyPlan [17].

Aquí se aprecia el problema de crear una aplicación de este tipo. Existen un gran número de tarifas que son modificadas constantemente por parte de las operadoras y el mantenimiento de estas aplicaciones se complica.

En myPlan y en otras aplicaciones similares la actualización de la base de datos de tarifas se realiza mediante la actualización de la aplicación, que aunque se puedan realizar automáticamente no resulta muy cómodo ni para el usuario ni para el desarrollador. Una solución sería actualizar las tarifas disponibles en segundo plano desde la aplicación accediendo a una base de datos en internet. Además esto implicaría que todos los usuarios independientemente de su versión de la aplicación instalada accederían a la última versión de la base de datos con los planes disponibles sin necesidad de actualizaciones.

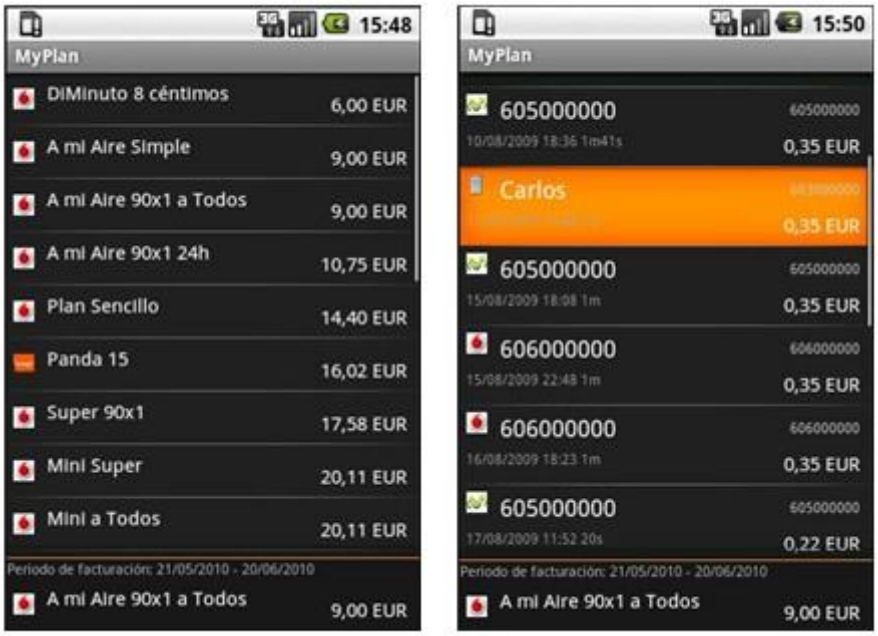


Figura 20. Capturas del listado de tarifas y el listado de llamadas.

Para solucionar el problema de determinadas tarifas que tienen en cuenta el operador del destinatario de la llamada a la hora de facturar, MyPlan ofrece al usuario la posibilidad de seleccionar el operador de sus contactos, como podemos ver en la Figura 20. Esto al usuario no le puede resultar atractivo pero al desarrollador le puede ahorrar mucho tiempo de trabajo ya que no existe una forma sencilla de consultar el operador de móvil de un número de teléfono de forma automatizada.

Desarrollo

El desarrollo de myPlan concuerda con las características de una aplicación de este tipo. En Mayo del 2005 se comenzó el desarrollo de la aplicación se generó gran parte del código de la aplicación (12.000 líneas de las 16.000 actuales). Como se puede apreciar en l gráfica de la Figura 21 tras el primer mes el número de líneas de código añadidas no fue significativo.

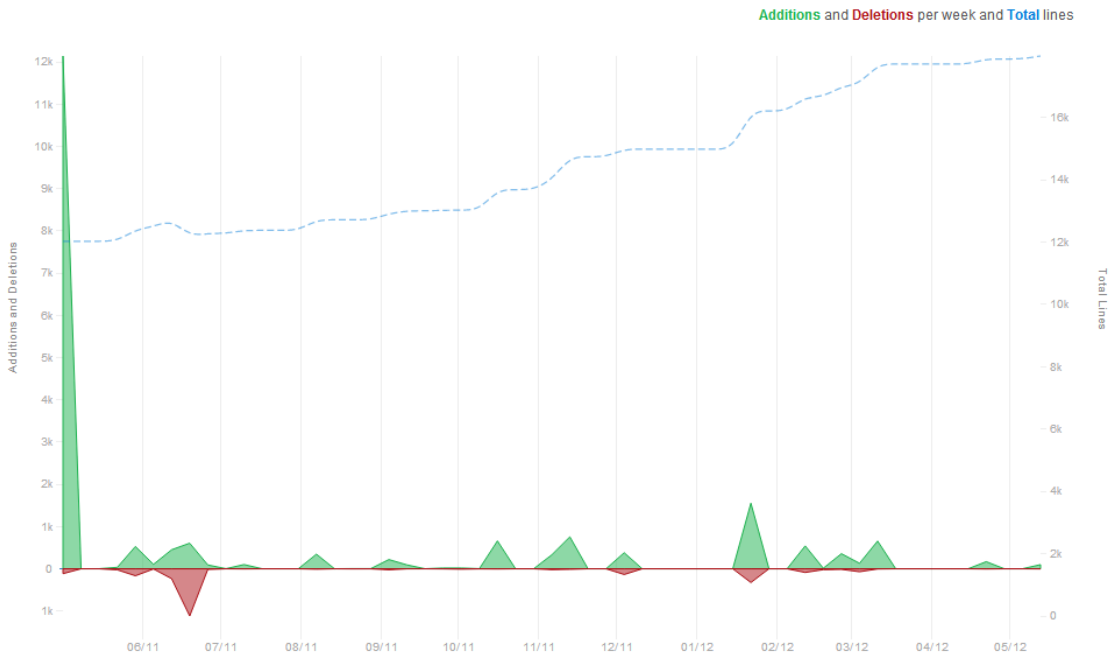


Figura 21. Número de líneas de código. [18]

Las actualizaciones se han mantenido durante el periodo de vida de la aplicación. Esto se debe a la necesaria actualización de la base de datos de tarifas en la aplicación y a la inclusión de determinadas características en las tarifas.

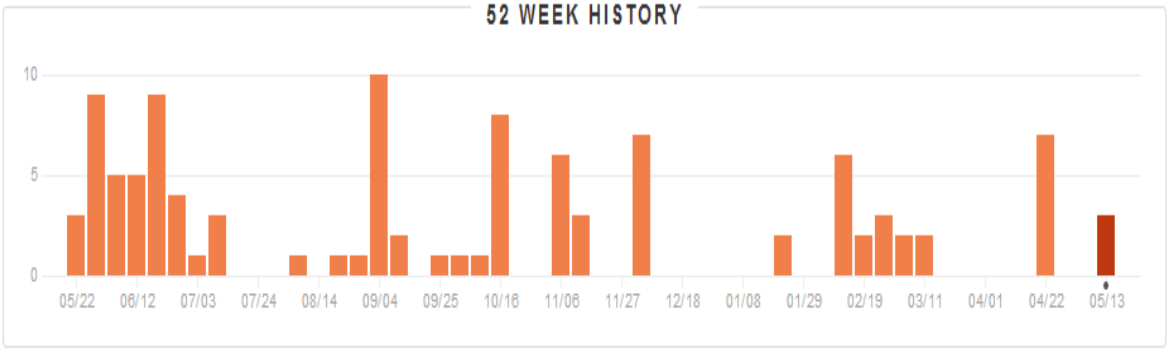


Figura 22. Frecuencia de actualización del proyecto. [19]

La mayoría de problemas a los que se ha enfrentado myPlan, son referentes a las nuevas tarifas [20]. Actualmente sigue teniendo estos problemas y otros como la inclusión de números Vips en determinadas tarifas, o el tratamiento de números de teléfono con prefijos específicos (902, 900) [21]. Estos son otros ejemplos de detalles a incluir en la configuración de una tarifa.

Gastos Móvil

La aplicación Gastos Móvil [22] ofrece únicamente el listado de las llamadas y su coste junto con un resumen de la factura.

Figura 23. Resumen de la factura en la aplicación Gastos móvil.

La gran ventaja de Gastos Móvil es su gestión de los planes de facturación. Ya hemos visto anteriormente cómo el gran problema del control de las llamadas y el cálculo de la factura era la gran variedad de tarifas ofertadas y su variedad. Gastos Móvil soluciona este problema de una forma muy inteligente: permitiendo al usuario la creación de tarifas personalizadas. La solicitud de tarifas que no estén definidas en la aplicación e incluso compartiendo con otros usuarios las tarifas creadas. La configuración de tarifas no resulta intuitiva y se complica sobre todo si se trata de una tarifa con franjas, pero no existe una forma sencilla de configurar dichas tarifas y que Gastos Móvil ofrezca esa posibilidad es una gran mejora.

En Gastos Móvil las tarifas están compuestas por el nombre, el coste fijo mensual, el límite de llamadas mensual y los números asociados a la tarifa (números a los que se aplicará dicha tarifa). Dentro de una tarifa se pueden definir franjas las cuales pueden tener costes concretos a aplicar dentro de su horario. Así bien, en una franja podemos definir el horario y los días que se aplica.

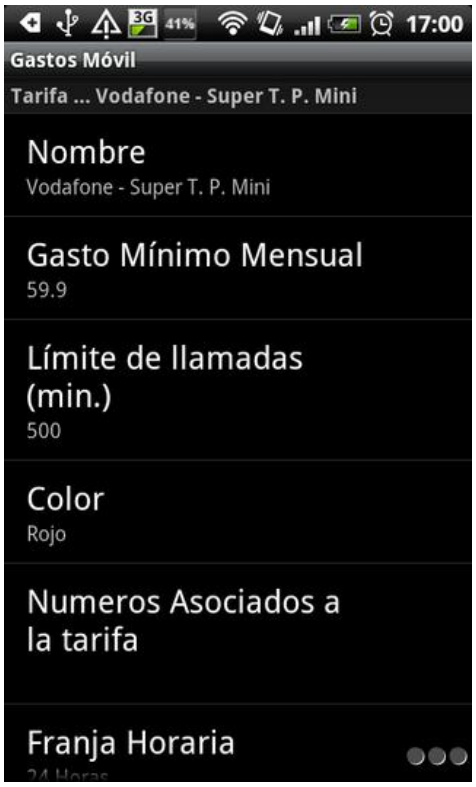


Figura 24. Pantalla de configuración de tarifa en la aplicación Gastos Móvil.

Figura 25. Pantalla de configuración de franja en la aplicación Gastos Móvil [23].

3. Requisitos necesitados

Para poder llevar a cabo el proyecto de CGTel hemos tenido requisitos tanto hardware y software, como requisitos humanos. A continuación, los desglosamos pues según su categoría.

Software

Para poder desarrollar la aplicación ante la que nos encontramos para un dispositivo Android necesitamos poder desarrollar en exclusiva para el sistema operativo de Google. Actualmente la compañía del buscador más utilizado ha sacado a la luz dos formas de poder escribir aplicaciones para su sistema: API de Android basado Java Software Development (SDK)[24] y el Native Development Kit (NDK)[25].

En nuestro caso, hemos elegido desarrollar en la API de Android basada en Java. Esta API es la más extendida y en la que se suelen desarrollar la mayoría de aplicaciones convencionales que podemos encontrar en el market. Está muy bien documentada en Android Developer y cuenta con una gran comunidad de usuarios. Además el SDK se ejecuta sobre Java y todos los integrantes del grupo tenemos experiencia con este lenguaje.

El NDK por el contrario, sirve para programar a bajo nivel en C/C++ sobre Android. El NDK es utilizado sobre todo para aplicaciones de tiempo real y motores para juegos. Dada la naturaleza de CGTel preferimos las facilidades y funcionalidades que nos ofrecía el SDK frente a la versión *nativa* de C/C++.

A continuación se decidió un entorno de desarrollo para poder programar y depurar el código Android. Para nuestra situación hemos escogido el Eclipse. Google proporciona un plugin para el Eclipse para poder crear proyectos en Android con el SDK previamente descargado (No entraremos más en el detalle de este punto ya que se tratará en los siguientes capítulos). Eclipse es una herramienta gratuita y multiplataforma, además incluye muchos plugins que pueden estar activados al mismo tiempo, especialmente interesante para nosotros el uso del plugin de Android acompañado con un plugin *Subversion* para poder usar un repositorio SVN.

En la parte de la aplicación móvil, finalmente lo único que necesitamos para poder desarrollarla es hacer uso de la librería Android Plot. Android Plot es una librería para Android que nos permite dibujar todo tipo de gráficos de una forma fácil e intuitiva. El carácter libre y de código abierto de esta librería nos hizo decantarnos por esta opción entre el resto de librerías para mostrar las estadísticas necesarias para el desarrollo de la aplicación. Android Plot nos otorga la facilidad de crear “widgets” sobre la interfaz de nuestros sistema cuyo contenido sean gráficos, ya sean estos de barras, puntos, circulares, etc.

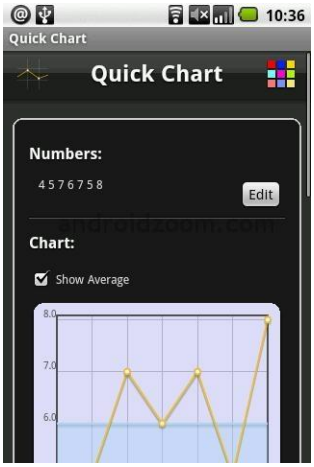


Figura 26. Gráfica generada con Android Plot.

Con todo ello habríamos terminado con los requisitos de cara a la aplicación móvil. Para poder acceder a los datos del resto de los terminales así como a los datos referentes a las tarifas existentes en el mercado necesitaremos una base de datos, elegimos por comodidad una base de datos de MySQL administrada mediante un phpMyAdmin. Estos han sido los elegidos ya que nuestra única necesidad esta poder tener una base de datos a la que poder acceder mediante un servicio web (realizado en php) y desde *webhost.com,* donde hemos alojado la base de datos de manera gratuita, nos proporcionaban los indicados anteriormente.

Hardware

Las herramientas hardware que vamos a necesitar para poder desarrollar la aplicación son: un PC donde poder desarrollar la aplicación y un terminal con Android 2.1 o posterior para poder probar la aplicación.

En nuestro caso hemos usado tres portátiles para el desarrollo de la aplicación, gracias a la compatibilidad que nos ofrecía el comentado anteriormente Eclipse, cada miembro del grupo ha podido usar su equipo y Sistema Operativo habitual. Normalmente hemos usado un equipo con OS X y los dos restantes tanto con Linux como con Windows.

Del mismo modo que con los equipos también hemos usado un terminal para las pruebas por cada miembro del equipo, en nuestro caso disponíamos de 3 terminales HTC con Android: Un HTC Hero (Android 2.1) el de menor gama de

todos con el que hemos podido ver de modo aproximado cuál va a ser el rendimiento mínimo de la aplicación. Con el hemos podido localizar los tiempos críticos de la aplicación y comprobar el correcto funcionamiento y despliegue de la aplicación en una pantalla de una resolución menor a las que podemos encontrar actualmente en el mercado. También hemos utilizado un Desire HD y Desire S (ambos con Android 2.3.3) con los que no hemos tenido ningún problema durante todo el período de desarrollo.

Figura 27. Dispositivos utilizados para el desarrollo de la aplicación.

Durante el período de pruebas de la aplicación hemos contado con usuarios que disponían de terminales de diversas marcas y gamas, por lo que tuvimos un amplio abanico de terminales donde probar los avances del desarrollo de CGTel.

Por ello podemos decir que CGTel es una aplicación que funciona en la mayoría de terminales Android con 2.1 o posterior y por lo tanto es perfectamente compatible sin importar la gama a la que pertenezca el terminal.

Recursos humanos

En el cuanto a los recursos humanos de los que hemos dispuesto durante el período de investigación, desarrollo y pruebas ha sido el equipo que ha estado compuesto en todo momento por Manuel Báez Sánchez, Jorge Cordero Sánchez y Miguel González Pérez y supervisado por María Victoria López López. Este equipo ha trabajado en el proyecto desde octubre de 2011 hasta junio 2012.

Horas invertidas

Para la organización de las tareas entre los integrantes así como para la puesta en común de los distintos avances, hemos mantenido reuniones periódicas, supervisadas en todo momento por María Victoria López y Juan Tejada. Estos encuentros se realizaron bisemanalmente junto a los otros alumnos del Grupo Tecnología UCM[26]. Todos sus integrantes participaron en la creación de varios proyectos de fin de carrera en la Universidad Complutense de Madrid relacionados con aplicaciones de Android con principios estadísticos o basados en técnicas de investigación operativa.

Las reuniones que mantuvimos fueron muy positivas para el avance del proyecto ya que los problemas a la hora de enfrentarnos en un desarrollo de Android fueron similares. Ninguno de los integrantes había desarrollado para el sistema operativo de Google, y muchos de ellos tampoco habían trabajado antes con servicios Web. La mayoría de los problemas que surgieron inicialmente, están vinculados a estas dos categorías.

Podemos considerar, pues que las horas de trabajo estuvieron determinados por los objetivos que nos marcamos cada 15 días. Usamos una metodología de trabajo tipo Scrum. Durante los 15 días que duraba cada Sprint nos proponíamos y asignábamos una serie de tareas simples, para ir avanzando hasta lo determinado en cada hito. Muchas veces estos “compromisos” no fueron cumplidos, ya fuera por desconocimiento o por complejidad de determinadas situaciones. Pese a todo, también hubo muchas veces que se pudieron realizar más tareas que las contempladas inicialmente, por lo que finalmente los tiempos (de desarrollo) se acercaron mucho a la predicción inicial, aunque es cierto que se alargó un par de semanas más de lo esperado con pequeños retoques y cambios finales de estilo.

En resumen, cada miembro del equipo ha dedicado íntegramente al proyecto sobre el que versa esta memoria de 8 a 10 horas semanalmente. Llevando a cómputos globales las horas de trabajo totales se acercan a las 270-320 horas por cada alumno. Este tiempo contabiliza tanto investigación, organización, desarrollo y pruebas.

Otros

Para poder compartir el código fuente durante el período de desarrollo, tuvimos que optar por hacer uso de un repositorio. De entre los formatos disponibles escogimos SVN, por su fácil integración con Eclipse mediante el plugin Subclipse,

y porque todos los integrantes del grupo conocen su funcionamiento ya que lo hemos utilizado en alguna otra ocasión. Nos decantamos por crear el repositorio en XP-Dev[27], ya que nos permitía crear un repositorio privado (código visible sólo para los miembros del grupo) con las características anteriormente descritas.

Por otro lado, pese a la obviedad, hemos necesitado como requisito indispensable, tanto para el uso de la aplicación como el desarrollo con los terminales, la disponibilidad de una conexión Wifi/3G para poder probar la conectividad de aplicación con la base de datos.

4. CGTel

Arquitectura y paquetes básicos

Para poder estructurar todas las funcionalidades que se han comentado en el apartado anterior, decidimos agruparlas en tres módulos básicos y un esqueleto de la aplicación principal. Desde la propia aplicación haremos uso de dichos módulos como se puede apreciar en la Figura 28.

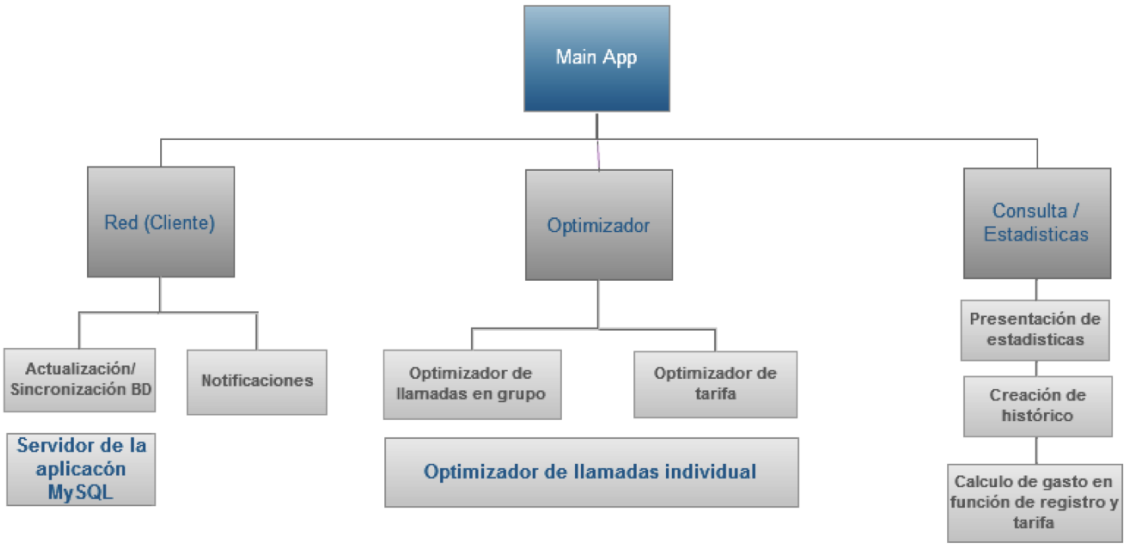


Figura 28. Esquema básico de la estructura de la aplicación.

A continuación explicaremos los tres módulos: bases de datos, predicción y el módulo de red.

Módulo Bases de datos

Para el funcionamiento de la aplicación es necesario un módulo que contenga la base de datos con la información necesaria. La aplicación Android debe disponer de los datos de llamadas de todos los usuarios de CGTel y tener acceso a un catálogo actualizado de las tarifas disponibles. Toda esta información debe estar accesible a través de una conexión a Internet y debe estar actualizada. Para ello se ha confeccionado una base de datos con la siguiente información

- Usuarios registrados en la aplicación e información de acceso. Esto incluye el número de teléfono, IMEI, contraseña y otros datos como la última fecha de acceso a la aplicación.

- Llamadas realizadas por el usuario. Estas son almacenadas en la base de datos para que desde cualquier terminal se pueda calcular el gasto y obtener estadísticas.
- Tarifas. La base de datos contiene la información actualizada de las tarifas que oferta el mercado.
- Información de los grupos. Datos de los grupos e integrantes.

Los datos son almacenados utilizando una base de datos MySQL [28] versión 5.1. La administración de la base de datos se ha realizado utilizando phpMyAdmin. La base de datos se ha implementado mediante varias tablas. Las tablas creadas son una para cada conjunto de datos descrito en el párrafo anterior:

- "usuarios" para almacenar la información de cada usuario de la aplicación.
- "numeros_vip" la información de un usuario se puede ver ampliada por los números vips de usuario.
- "grupos" es una sencilla tabla que almacena la información de los grupos creados y sus miembros.
- "llamadas" incluye el listado de llamadas realizadas.
- "tarifas" donde están definidas las tarifas. Esta tabla necesita de información adicional que esta almacenada las tabla "franjas". Dicha tabla contiene información relativa a las franjas de la tarifa, las cuales pueden ser reutilizadas por distintas tarifas.

En la Figura 29, podemos ver cada una de las tablas así como las relaciones mediante sus claves con las otras tablas.

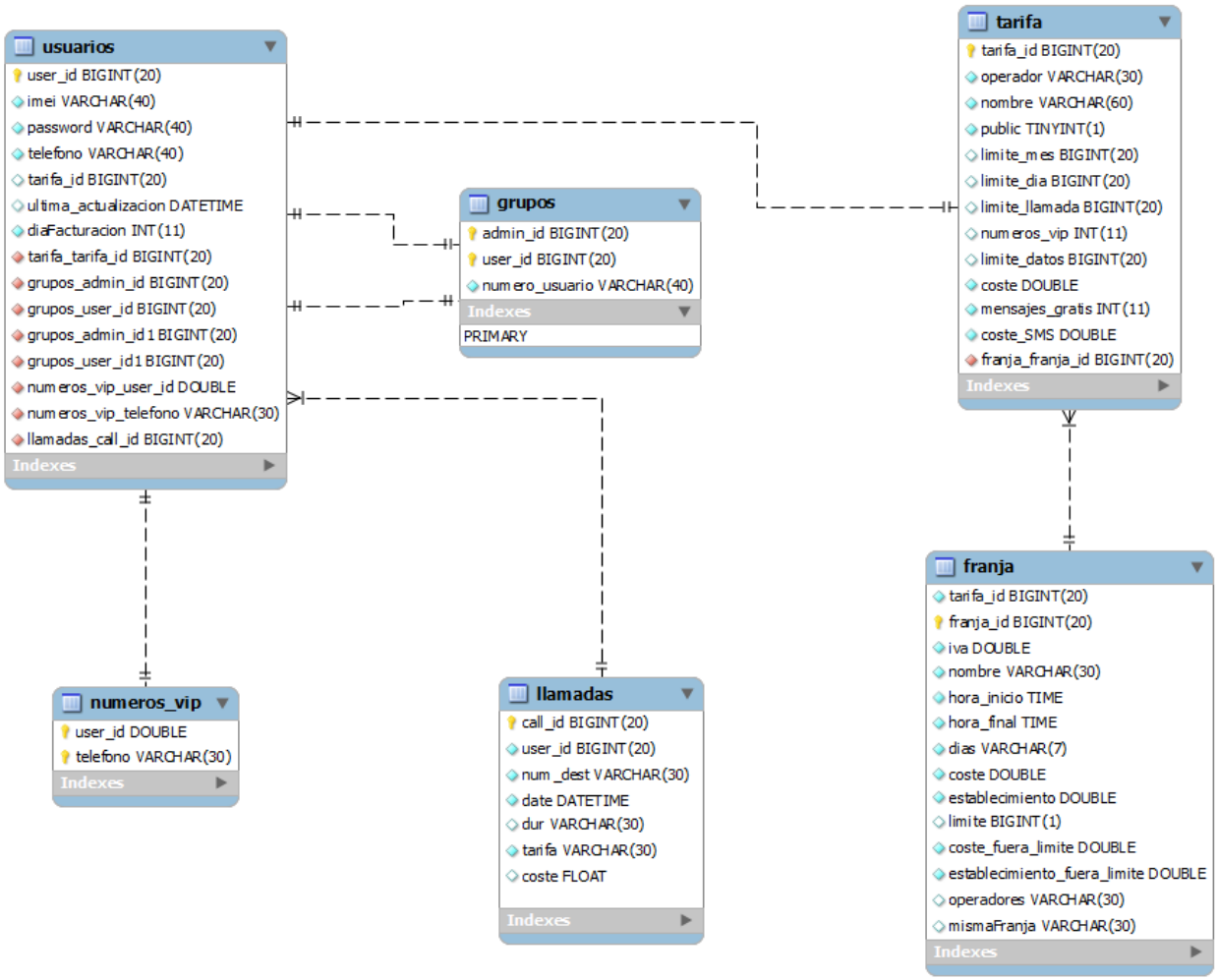


Figura 29. Estructura de la base de datos.

Encriptación de los datos

Para aumentar la seguridad de la base de datos se recurrió a un cifrado del contenido principal. Dicho contenido cifrado fueron los números de teléfono y contraseñas. De esta forma el usuario permanece anónimo ante los administradores de la base de datos.

Los algoritmos Hash son algoritmos que dado un conjunto, en nuestro caso cadenas de caracteres, generan otro conjunto[29]. Este conjunto generado se trata de una cadena de caracteres de longitud fija que es única para cada entrada del algoritmo y no puede ser reconstruida. De esta forma los números almacenados en la base de datos no pueden ser reconstruidos al estar almacenados utilizando una función hash.

Algunos de los algoritmos más comunes son MD5[30] (Message-Digest Algorithm 5) y SHA1[31] (Secure Hash Algorithm 1). El algoritmo MD5 es el que se ha utilizado en la base de datos. Ha sido uno de los algoritmos Hash más utilizado

en los últimos años. Fue desarrollado por Ronald Rivest del Instituto Técnico de Massachusetts (MIT)[32]. Debido a la capacidad de procesamiento de los sistemas actuales la versión básica del algoritmo MD5 es insuficiente.

Hoy en día este tipo de algoritmo es utilizado para comprobación de integridad de ficheros. Es bastante común que junto a la descarga de un fichero se descargue su codificación Hash y se compare con la generada a partir del fichero descargado. Si el código MD5 descargado y el generado son distintos el fichero descargado es erróneo. La versión básica del algoritmo produce 128 bits de salida por cada bloque de 512 bits de entrada, lo cual lleva a otro problema que sucede con los algoritmos de hash, si bien es poco probable existe lo que se conoce como la colisión de Hash. Dicho problema tiene lugar cuando dos entradas distintas producen una misma salida tras aplicarles una función hash. Esto sucede ya que el número menor de posibles salidas es inferior al de posibles entradas. Si bien conociendo el conjunto de datos de entrada es posible crear una función hash que no dé lugar a colisiones. En el caso de nuestra base de datos dicho conjunto de entrada son números de teléfono los cuales los cuales no dan lugar a colisiones en las salidas.

La actualización de la base de datos para incluir este tipo de encriptación ha supuesto un gran paso y una remodelación de la aplicación cliente de Android. Ahora el acceso a la base de datos así como las consultas deben realizarse cifrando los datos previamente. Esto ha ocasionado pequeños cambios en el acceso de los usuarios o en acciones de consulta como por ejemplo la obtención de las llamadas de un usuario. La actualización de la aplicación se complica cuando realizan consultas complejas a la base de datos.

La función hash ha sido utilizada también para el proceso de identificación del usuario al comienzo de la aplicación. Cuando el usuario introduce sus datos estos son registrados tras aplicarles la función hash y son almacenados en las preferencias. Si algún usuario malicioso intentase obtener estos datos, no podría revertir la información y obtener la contraseña del usuario.

PhpMyAdmin[33]

Para el mantenimiento de la base de datos se ha hecho uso de phpMyAdmin. Esta herramienta proporciona lo necesario para la administración de la base de datos con una presentación muy amigable. En la Figura 30 se puede apreciar la vista principal que nos proporcional el panel de administración así como las opciones que permiten administrar de una manera intuitiva los datos y campos de nuestra base de datos.

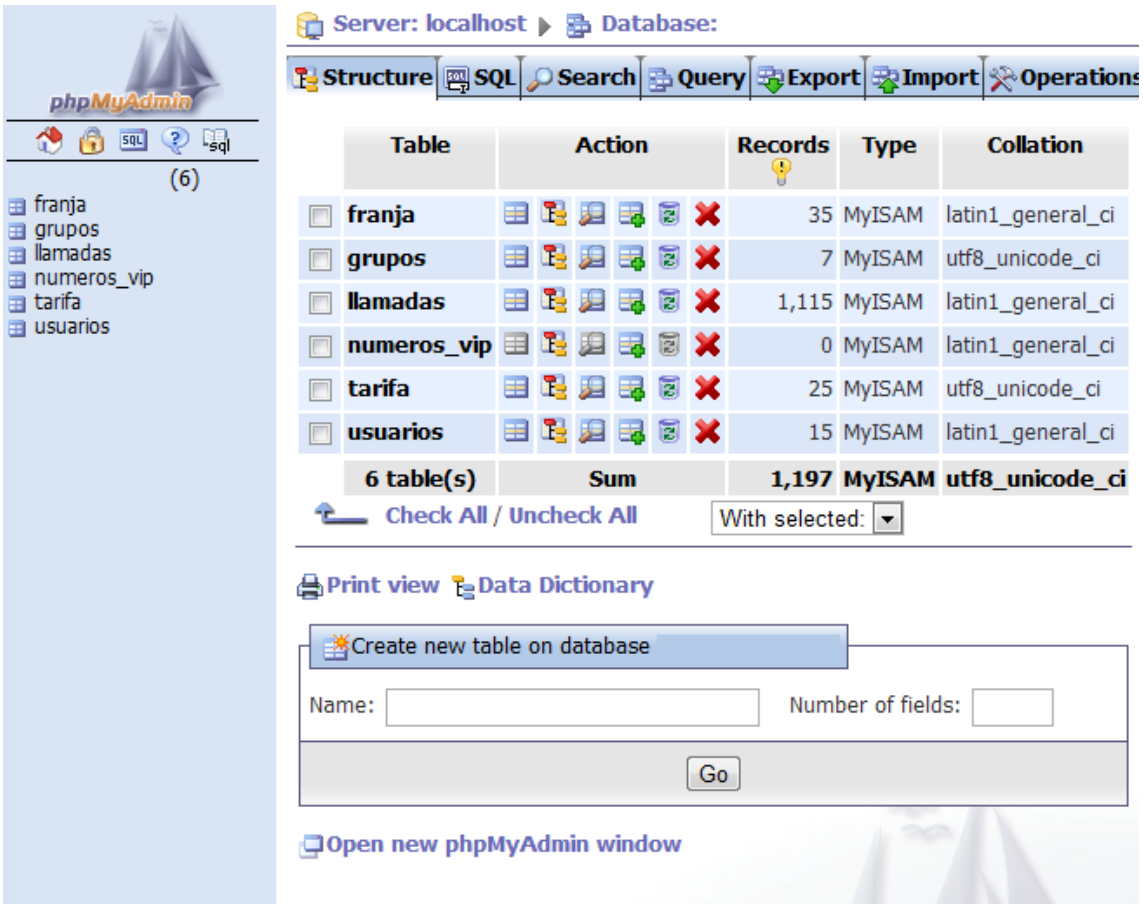


Figura 30. Interfaz web de la aplicación phpMyAdmin.

Módulo Predicción

Para hacer la predicción de gasto mensual basándonos en las llamadas desde facturación hasta la fecha, hemos tenido que dividir las llamadas en las distintas franjas de la tarifa seleccionada por el usuario, ya que el coste de las llamadas depende de la hora y del día a la que se realicen.

Una vez están divididas en franjas, basta con acumular el número de minutos de las llamadas y ver si superan el número de minutos gratuitos en dicha franja, en caso de que esa franja no disponga de minutos gratuitos su valor será 0. La predicción también se hará sobre las franjas. Para ello hemos usado técnicas de series temporales. Cabe destacar que hay un amplio abanico de técnicas predictivas y que pese a que hemos decidido utilizar dos de ellas pero, gracias a la facilidad de modulación de Java, podríamos agregar otra técnica sin mayor

esfuerzo que el de pasarla a código Java en un módulo que reciba como datos de entrada los datos reales y devuelva los predecidos.

La elección entre ambas técnicas se realiza usando la que tenga el menor error cuadrático medio.

Error cuadrático medio (ECM).

El error cuadrático medio es la potencia cuadrada de la desviación de la predicción con respecto al valor real. En nuestro caso tendremos un número n de errores cuadráticos medios, donde n será el número de días transcurridos entre el día de facturación y el día actual. Para hallar el error cuadrático medio de una serie de predicciones a una etapa calculamos la suma de ECM de dicha serie y la mejor serie de las evaluadas será la que tenga el menor ECM.

Series temporales

Los dos métodos de predicción que utilizamos en el estimador son de naturaleza distinta. Uno de ellos es un método sencillo y fácil de calcular al que llamamos método ingenuo y el otro, más complicado, es un método de medias móviles.

Método ingenuo

El método ingenuo, es un método que se aplica sobre una serie cuya característica más destacada es que tiene un cálculo sencillo. Sirve principalmente como caso base para comparar varias técnicas o métodos de series temporales y saber así cuál de ellas es la que mejor se ajusta al caso real.

Para nuestro propósito la vamos a utilizar como alternativa en el caso de que las medias móviles se ajusten muy poco a los resultados reales. Ofreciendo así otra alternativa más cercana al valor real.

Nuestro método ingenuo consiste en suponer que haremos las mismas llamadas en las semanas siguientes que las que hicimos la primera semana desde el inicio de la facturación. Así conseguimos tener un método ingenuo elaborado ya que la predicción variará en función del día de la semana en el que nos hallemos. Por ello este método ingenuo no se utiliza para comparar otros dos métodos elaborados, como sería lo normal, sino que forma parte de los métodos elaborados.

Medias móviles

Las medias móviles es un método que se aplica sobre los últimos n elementos de una serie. Así para calcular el valor correspondiente al día k necesitaremos tener los datos comprendidos entre los k-n/2 y k+n/2 días. En nuestro caso dicho n será igual al número de días en que esté activa la franja sobre la que vamos a calcular la predicción. A continuación vemos la expresión de las medias móviles (MM), siendo k el día a hallar y N el número de días en que opera la franja.

$$\hat{Y}_k = \frac{Y_{k-\frac{N}{2}} + Y_{k-\frac{N}{2}+1} + \ldots + Y_{k+\frac{N}{2}}}{N}$$

Hemos introducido algunas variaciones sobre las medias móviles clásicas para mejorar la predicción calculada. Dichas variaciones consisten por un lado en introducir un sistema de pesos, dando mayor peso al último dato del día de la semana actual a la hora de hacer la estimación. En la podemos apreciar la modificación realizada.

Error! Reference source not found.

Las medias móviles tienen un problema y es que, en principio, vamos a tener menos datos calculados que el número de datos iniciales, ya que, siguiendo con el ejemplo anterior, para calcular el primer dato de las medias móviles vamos a necesitar los n primeros elementos de la serie y para hallar el último haremos uso de los n últimos quedándonos así con 2n-2 elementos de los que tenemos en la serie original.

La segunda modificación trata de resolver este problema y ha sido viable gracias a la forma en que hemos gestionado los datos en Java.

Dichos datos los hemos almacenado en un array, de forma que tenemos un array de tamaño 32, del elemento 0 al elemento 31. Si ignoramos las posiciones que no correspondan a ningún día de la estimación que queremos realizar, como son por ejemplo el 0 en cualquier mes o el 31 en los mes de menos de 31 días, podemos interpretarlo como un array circular donde la posición i corresponde al día i del mes durante el que estamos haciendo la facturación, no desde dicha facturación.

Así podemos comenzar a calcular las medias móviles desde el día n terminando en el día n-1.

Otras series temporales descartadas

En un principio nos planteamos hacer uso de otras técnicas, finalmente hemos decidido utilizar las medias móviles porque es una técnica idónea cuando no existe tendencia, al igual que los alisados. A continuación explicamos en qué consisten éstas últimas y exponemos las razones por las que las hemos rechazado.

Alisado simple

La técnica de alisado simple consiste en realizar un alisado sobre los datos que tenemos. Un alisado consiste en realizar una media móvil ponderando los datos en función de cierto α, que tomará valor en función de cómo queramos repartir los pesos, con un valor alto tomarán mayor importancia los primeros días y con un valor pequeño los últimos. Cabe destacar que la suma de coeficientes de ponderación ha de ser 1 y que el valor tomado por cada coeficiente es $\alpha(1-\alpha)^i$ donde i es el número de elementos que distancian el dato a hallar del dato usado para la predicción. Podemos ver la demostración de que este valor será 1 siempre (4) y la fórmula teórica de infinitos términos del alisado simple (3), ambas con el

parámetro 0 < ω < 1, a dicho parámetro se le da valor y tiene la misma naturaleza que α.

$$S_t = (1-\omega)Y_t + (1-\omega)\,\omega\,Y_{t-1} + (1-\omega)\,\omega^2\,Y_{t-2} + \dots, (3).$$

$$\sum_{j=0}^{\infty}(1-\omega)\omega^j = (1-\omega)\sum_{j=0}^{\infty}\omega^j = (1-\omega)\frac{1}{1-\omega} = 1, (4).$$

¿Por qué hemos descartado el alisado?

Dicha técnica se ha descartado ya que da mayor peso a los datos más recientes, pero vamos a ver que no es la mejor forma de hacer la estimación, para ello vamos a basarnos en un ejemplo muy sencillo. Si mi franja opera durante n días el dato con mayor peso será el del día de ayer, lo cual es una buena referencia, pero por el contrario el dato con menor peso será el del tal día como hoy de la semana anterior, lo cual dada la naturaleza de nuestro problema es algo altamente contradictorio, ya que precisamente debería de ser de los días con mayor peso y no lo contrario. Por contraposición está el caso contrario en el cual el día de hoy de la semana anterior tendrá un peso importante mientras que el día anterior de la franja tendrá una importancia pequeña, lo cual tampoco nos interesa porque posiblemente sean dos de los días más importantes.

Por ello hemos desestimado el alisado como posible estimación si bien es cierto que las serie temporal utilizada es un híbrido entre el alisado y la serie temporal, ya que le damos mayor relevancia al dato del mismo día de hoy en la semana anterior a la estimación a calcular, pero sin perder relevancia el día anterior de la franja.

Módulo de red

En el módulo de red de la aplicación se han recogido todas aquellas interacciones referentes a la conectividad mediante Internet de la aplicación.

Para el caso ante el que nos encontramos las comunicaciones necesarias están muy bien definidas, por un lado la conectividad contra el servidor público donde se hospeda la base de datos con los datos requerido por la aplicación y la conectividad entre los terminales.

Para el funcionamiento de la aplicación es indispensable poder acceder a una base de datos pública de tal manera que en ella queden alojados tanto los datos de los distintos usuarios (desde llamadas, logins, contraseñas) como datos a nivel global para presentar a los usuarios y que son subidos a la base de datos desde la propia administración de CGTel (actualizaciones de tarifas, precios, franjas...) Para todo ello, decidimos conectar cada terminal con la base de datos mediante un servicio web.

Las comunicaciones entre terminales amplían el campo del intercambio de datos a un nivel mucho mayor. Para el propósito de la aplicación, sólo necesitamos comunicar a un determinado usuario que el recomendador de llamada nos indica que una llamada suya sería el gasto óptimo es ese instante. Dada dicha

naturaleza, pensada en el ahorro de la factura telefónica, vimos inviable tratar esta comunicación por mensajes de texto o cualquier otro tipo de conexión que pudiera suponer un coste adicional para el usuario final. Dado que la conexión a internet es requisito indispensable para la aplicación, decidimos crear notificaciones entre terminales usando el framework de Google Cloud To Device Messaging[34] (C2DM).

Tras investigar sobre este framework y conocer su funcionamiento, entendimos que la arquitectura de nuestro servidor complicaba mucho su utilización, por lo que finalmente empleamos el envío de mensajes estándar de Android que nos ofrece un listado de aplicaciones disponibles en el terminal para elegir la que elija el usuario para comunicarse directamente con otros terminales.

Comunicación con la BBDD: Web Services

La W3C, responsable de la definición de estándares para el Web, define así los servicios Web (*Web Services*[35]): son sistemas software pensados para dar soporte a interacciones entre máquinas.

Para que estas interacciones sean posibles todo servicio Web posee una interfaz descrita en un formato procesable por una máquina (utilizando el lenguaje WSDL, *Web Service Description Language*[36]). En esta descripción se indica cómo otros sistemas pueden interactuar con el servicio Web: qué mensajes se intercambian, qué parámetros deben pasarse al servicio para que lleve a cabo su función, qué resultados se obtienen o qué fallos pueden producirse. Normalmente la invocación remota a un servicio Web se lleva a cabo utilizando mensajes SOAP[37], serializados con lenguajes de marcado, fundamentalmente XML, y transmitidos mediante el protocolo HTTP, *HyperText Transfer Protocol*.

Para facilitar la localización de servicios, es posible registrar los mismos en un directorio o registro UDDI, *Universal Description, Discovery and Integration*. La descripción del servicio almacenada en el registro UDDI contiene información como el nombre del servicio, detalles sobre el proveedor del servicio, el tipo de servicio ofrecido, su ubicación (URL) y detalles técnicos sobre el mismo (el WSDL del servicio). La información de esta descripción es utilizada por las aplicaciones cliente para decidir qué servicio pueden invocar.

Los Web Services son una tecnología adecuada cuando se pretenden desarrollar arquitecturas orientadas a servicio. En estas arquitecturas distintos componentes interactúan intercambiando mensajes para conseguir llevar a cabo una tarea compleja. Gracias a la utilización de Web Services, que pueden ser accedidos remotamente, componentes distintos pueden ejecutarse en distintas máquinas distribuidas por toda la red. Actualmente los más extendidos por la red son aquellos referidos a la interacción con bases de datos.

En CGTel, el módulo de red es el encargado de generar la peticiones HTTP (Http Requests) contra las distintas URLs disponibles para interactuar con la base de datos de la aplicación. Mediante POST, añadimos a la petición todos los atributos necesarios para que esta se pueda llevar a cabo, desde los datos de

usuario y contraseña para acceder a la base de datos hasta los distintos requisitos de cada servicio. El usuario y contraseña para poder acceder a la base de datos se envían desde la propia aplicación, así evitamos tener guardado en el propio código php del servicio estos datos y que de alguna forma estos fueran ejecutados desde cualquier equipo con acceso a las URLs que nos acometen.

Una vez los servicios web, desarrollados en php, toman el acceso a la base de datos devuelven una respuesta codificada con el estándar JSON (*JavaScript Object Notation*[38]*).* JSON es un formato ligero para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de Javascript que no requiere el uso de XML. Su simplicidad ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX.

Una de las ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript y Java, un texto JSON se puede analizar fácilmente usando los procedimientos integrados en los propios lenguajes lo que ha sido fundamental para que haya sido aceptado por parte de la comunidad, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

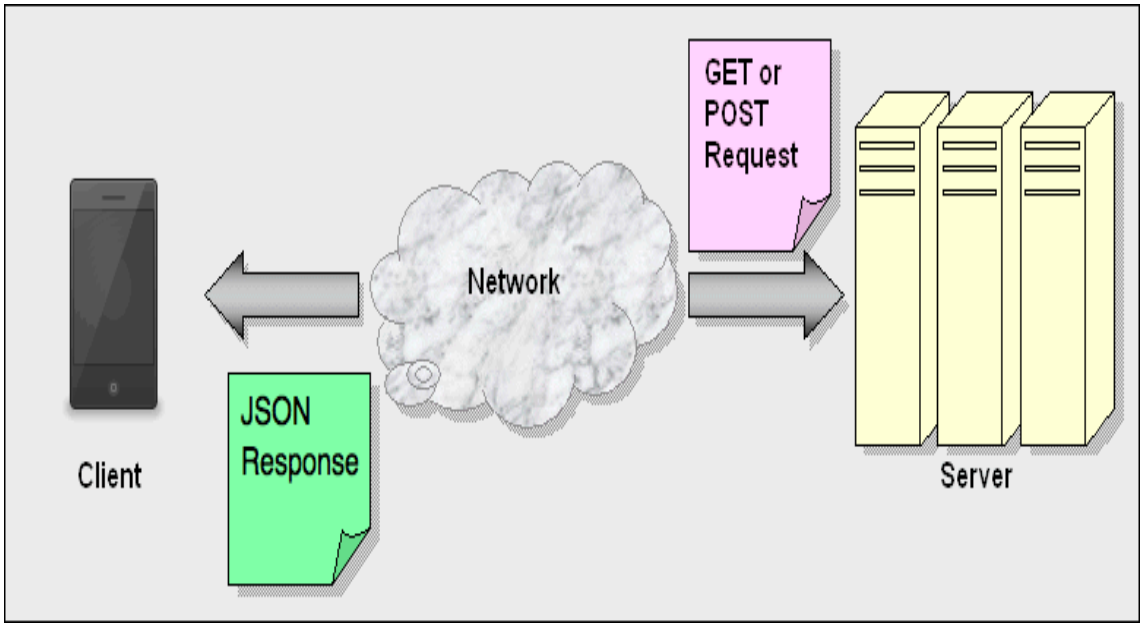


Figura 31. Esquema del funcionamiento de JSON.

En la Figura 31 podemos ver como la aplicación hace uso de las peticiones HTTP a través de Internet y de la codificación JSON para comunicarse con el servidor. Los servicios web utilizados en la aplicación son:

- **acceso**: Este servicio nos proporciona los datos para validar el login del usuario en la aplicación, gestionar las altas y los códigos de error en caso que alguna de las credenciales sea incorrecta.

- **consulta**: Permite realizar consultas genéricas contra la base de datos. También valida las credenciales para acceder a la base de datos y proporciona los correspondientes códigos de error.

- **insert**: Permite insertar y actualizar determinadas tablas de la base de datos (llamadas y usuarios). También valida las credenciales para acceder a la base de datos y proporciona los correspondientes códigos de error.

- **listarTarifas**: Nos proporciona un listado de las tarifas y sus datos asociados para tratarlos dentro de la aplicación.

- **listarFranjas**: Nos proporciona un listado de las distintas franjas, sus tarifas asociadas y sus datos asociados para tratarlos dentro de la aplicación.

- **time**: Sincroniza la hora del terminal con la de la base de datos (diferencia horaria, ya que el servidor utiliza la hora de Estados Unidos).

Comunicación entre terminales: Google C2DM

¿Qué es C2DM?

Android Cloud To Device Messaging (C2DM) es un servicio de notificaciones push, que ofrece a los desarrolladores de aplicaciones para Android la posibilidad de enviar datos desde sus servidores hasta los terminales que tengan la aplicación instalada. Este servicio provee un mecanismo simple y ligero que los servidores pueden utilizar para que los servidores puedan comunicarse directamente con los terminales directamente, para actualizar con nuevos datos o simplemente como mecanismo de sincronización. El servicio C2DM se encarga de todo el trámite de encolado de mensajes y de la entrega a la aplicación destino en los distintos terminales.

Las aplicaciones que usan está tecnología no necesariamente tienen que estar ejecutándose para que puedan recibir los mensajes a través de la red de C2DM. El sistema levantará automáticamente la aplicación ya que el mensaje se recibe a través de un *broadcast receiver* cuando el mensaje llega el dispositivo.

C2DM no proporciona ninguna interfaz de cara al usuario ni ninguna otra facilidad de cara al usuario / desarrollador para manejar los datos de estos mensajes. Sin embargo, proporciona un paso de mensajes directo hasta la aplicación con la que estemos intentando conectar de manera que será esta quien tendrá que controlar y ejecutar las instrucciones recibidas en el mensaje proporcionado.

Para poder utilizar este *framework*, el terminal de destino debe estar ejecutando la versión 2.2 de Android (API 8) con el Market instalado o superior y tener configurada una cuenta de Google. Esto es necesario ya que el C2DM lo utiliza Google para poder enviar notificaciones a través de la web del Market

(ahora Google Play) para lanzar automáticamente instalaciones, actualizaciones y otros procesos.

El cuerpo de los mensajes que actualmente se permiten enviar no pueden exceder el tamaño de 1024 bytes. Para poder utilizarlo requiere registrarse como desarrollador y comentar el uso estimado en mensajes por día y la cadencia de estos para que Google estime si es viable su uso sin licencia adicional, en caso de superar cuantiosamente esta cuota, Google proporciona licencias de C2DM a nivel profesional para poder cursar una cantidades de tráfico mayores. A pesar de esto, este proyecto sigue en Google Labs, siendo una de las grandes funcionalidades que pone a nuestra disposición Google, pese a ser una versión de prueba.

¿Por qué C2DM? Poll y Push

Utilizamos las técnicas de poll y push cuando necesitamos mantener los datos sincronizados con un servidor de manera constante.

El nombre poll, proviene del inglés encuesta, se trata de consultar periódicamente a un servidor externo por el estado de los datos con la intención de mantenerlos en todo momento sincronizados o para tener constancia de los cambios. Es apropiado usar *polling* en situaciones donde el contenido cambie constantemente y para mantener datos coherentes de una manera estable.

Si usamos *polling*, debemos asumir un gasto adicional en red y batería. En un dispositivo móvil en reposo podemos estar hablando de un consumo cercano a los 5-8 mA, si usamos polling, gastamos unos 0.5mAh por una encuesta corta ya que las comunicaciones se establecen durante un breve período de tiempo. Si a todo esto sumamos que se van a establecer esta conexiones con un refresco de unos 5 minutos, asumimos que esta práctica consumirá más de 10% de uso de la batería.

Por el contrario, tenemos las notificaciones push, que describen una comunicación con el cliente una vez la petición ha sido realizada por otro usuario o por un servidor central. Este es el caso de uso dónde aparece C2DM. Google proporciona una manera fácil y gratuita de utilizar notificaciones push, con su propio servicio de notificaciones push (creado inicialmente sólo para el Market) dejando un Framework a disposición del desarrollador.

En el uso de C2DM aparecen 3 componentes fundamentales:

- **Dispositivo Móvil:** El dispositivo en el que se ejecuta la aplicación que usa C2DM. Este debe de ser un dispositivo Android 2.2 o superior, tener Google Play instalado y estar logado al menos con una cuenta Google.

- **Servidor de datos:** Servidor encargado de gestionar el estado de los datos y comunicar el estado de los mismos. El servidor de datos envía los mensajes a la aplicación Android en el dispositivo a través del servidor de C2DM.

- **Servidor C2DM:** Servidor de Google encargado de recibir los mensaje del servidor de dato y enviarlos al dispositivo móvil.

Para la interconexión de estos componentes, se verifica constantemente la autenticación de los usuarios / servidores mediantes los siguientes:

- **Sender ID:** Email asociado a la aplicación y que sirve en el proceso de registro para identificar la aplicación que tiene permisos para mandar mensajes al dispositivo.

- **Application ID:** La aplicación que está registrada para recibir mensajes. Este identificador corresponde con el nombre de paquete indicado en el *manifest* de la aplicación.

- **Registration ID:** Id proporcionado por el servidor C2DM y que permite a la aplicación recibir mensajes. Una vez esta tiene dicho id deberá proporcionárselo al servidor de datos el cual lo usará para identificar cada dispositivo registrado para recibir mensajes. Este id está asociado una instancia concreta de la aplicación en un dispositivo concreto.

- **Google User Account:** Para que C2DM funcione el dispositivo debe tener configurada al menos una cuenta de Google.

- **Sender Auth Token:** Nuestro servidor deberá logarse con la cuenta elegida como “Sender ID” mediante una petición POST recibiendo en la respuesta el ID que la identifica para poder mandar mensajes.

A continuación en la Figura 32 podemos ver un esquema del funcionamiento del paso de mensajes entre el dispositivo, servidor y la nube de Google. En ella aparecen enumerado los pasos que se realizan a la hora de entregar un mensaje al dispositivo. Como se puede apreciar, la autenticación comentada se realiza por pasos: el Sender ID se comprueba en el paso 1, el Registration ID y Application ID en 2, Auth Token y User Account en el paso 4.

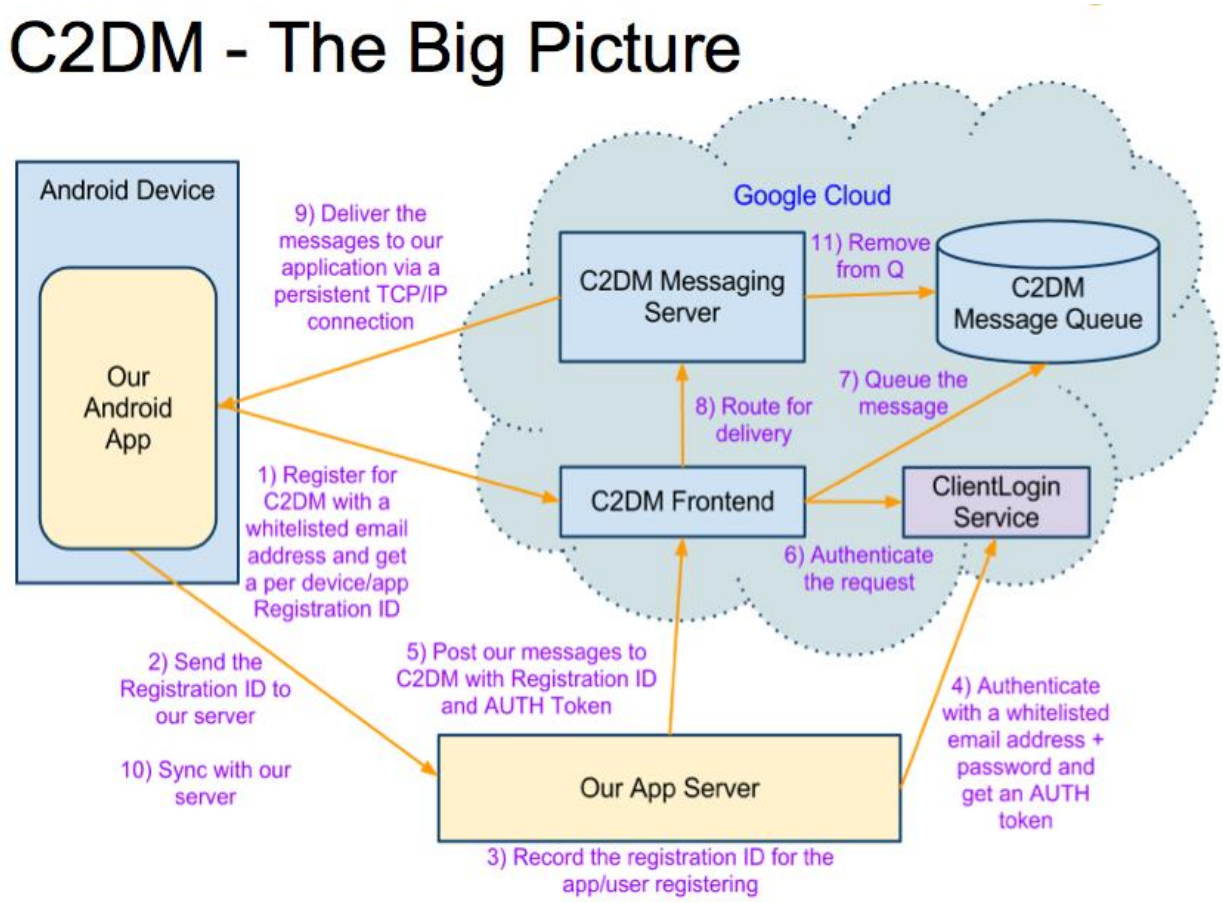


Figura 32. Esquema del funcionamiento del paso de mensajes entre el dispositivo, servidor y la nube de Google. [39]

Problema C2DM, por qué no lo usamos finalmente

Dada la complejidad que nos causaba en la parte del servidor para poder uso de C2DM, junto al poco uso que tiene dentro de nuestra aplicación decidimos no utilizarla. Pese a que el framework ofrece muchas facilidades son necesarias ciertas configuraciones, así como que el servidor permita establecer una conexión segura (HTTPS) y el servidor donde teníamos ya alojada la base de datos no permitía dichas configuraciones. Bien es cierto que si el lector está interesado en dicho Framework para trabajos futuros recomendamos encarecidamente que disponga de un VPS (Virtual Private Server) donde tenga un acceso total a su configuración para poder utilizar la tecnología de notificaciones Push que nos brinda C2DM.

Para la comunicación entre los dispositivos hemos usado finalmente los mensajes nativos del termina. Por defecto aparecen unos servicios mínimos como email, mensaje de texto además de otras aplicaciones instaladas en el propio terminal que cumplan las funcionalidades de mensajería como pueden ser Facebook y WhatsApp [40], siendo ésta la aplicación más popular de su categoría. En la Figura 33 se muestra un ejemplo de la pantalla de selección que Android ofrece de manera nativa para el paso de mensajes entre terminales. Este es un ejemplo

lanzado desde CGTel en un terminal en concreto que dispone de Dropbox, Facebook y WhatsApp como alternativas a las que ofrece Android de por sí, no obstante las opciones variarán en función de las aplicaciones instaladas en cada dispositivo.

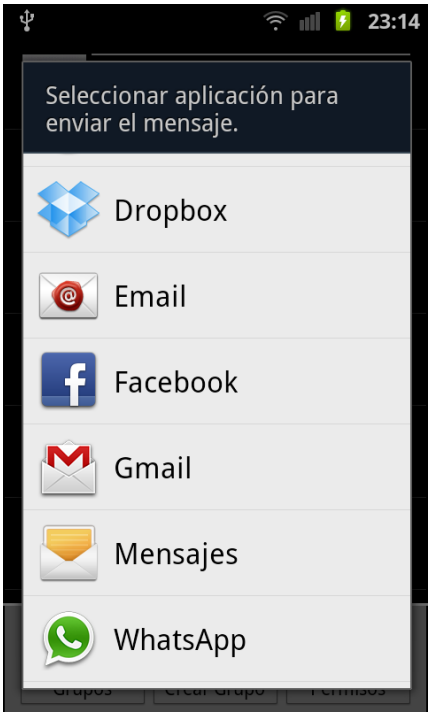


Figura 33. Ejemplo de ventana de diálogo para envió de mensajes en Android.

5. Android

Historia y productos Google

Historia

La historia de Google comenzó en 1997 como una tesis doctoral de Larry Page y Sergey Brin, estudiantes de la Universidad de Stanford. Su objetivo era mejorar las búsquedas en Internet.

El 15 de Septiembre de ese año cuando registraron el dominio Google y un año después se funda la compañía Google Inc. estrenando a su vez el motor de búsqueda. A continuación describiremos brevemente las partes más destacadas de Google.

Productos más destacados de Google

Gmail

Es el servicio de correo electrónico de Google, sus inicios se remontan al 15 de Abril de 2004 y se ha convertido en apenas 8 años en uno de los más destacados. Gmail es necesario para poder explotar otros servicios de Google entre los que destacan Google Calendar, Google Docs y Google Talk, que son un calendario, un sistema de archivos compartidos y un chat respectivamente. También es necesaria una cuenta de Gmail cuando se adquiere un terminal móvil con Android, siendo ésta el nombre de usuario.

YouTube

Fundada en Febrero de 2005, inicialmente no era parte de Google si no que debido a su sonado éxito fue adquirida en Octubre de 2006 por 1.650 millones de dólares. En ese momento muchas empresas aprovecharon para denunciar a YouTube por los vídeos que no cumplían la declaración de derechos dichas empresas.

Motorola Mobility

Google compra por 8.800 millones de euros Motorola Mobility en agosto de 2011 consiguiendo así una empresa propia para fabricar terminales móviles. También cabe destacar que una de las principales razones por las cuales Google adquiere Motorola es la compra de ciertas patentes, hasta un montante de 20.000, por lo que en cierto modo estaban obligados a hacer dicha compra. Otra compra sonada relacionada con esta es la de 1000 patentes a IBM. Así es como Google se protege de los posibles ataques por parte de Microsoft o Apple.

En Mayo de 2012 finalizó la compra de Motorola Mobility[41] tras recibir la autorización de China, completando nueve meses de trámites burocráticos que han provocado un coste extra a Google alcanzando los 12.000 millones de dólares. Este movimiento por parte de Google ha llevado a todo tipo de especulaciones como en cuanto al futuro de Android. Para evitar que la situación actual de Android

cambiase, China exigió a Google que garantizase la condición de Android de sistema operativo abierto durante al menos cinco años más, temiendo que al comprar Motorola emprendiese una estrategia similar a las de sus competidores Apple y Microsoft.

Android

La parte que realmente nos importa y posiblemente una de las filiales de Google con mayor proyección en el futuro. La principal razón por la que Google adquirió Motorola así como las patentes de IBM.

Introducción y comienzo de Android

Android es uno de los sistemas operativos móviles más usados en la actualidad y el único que compite en prestaciones con iOS. El nombre Android viene de la novela de Philip K. Dick *¿Sueñan los androides con ovejas eléctricas?*, que versa sobre un grupo de androides con aspecto humano, que se descontrolan y son cazados por unos cazadores llamados Blade Runner, la historia está contada en torno al protagonista Rick Deckard (Harrison Ford).

Android, está desarrollado en Java, adoptando así sus cualidades más destacadas, tanto las positivas como las negativas. En el primer grupo podríamos añadir una mayor portabilidad, mayor fiabilidad y una sencilla modulación del código haciéndolo más sencillo de programar. Por contraposición entre sus cualidades negativas son una mayor ineficiencia y un mayor consumo energético, que en el caso de los terminales móviles es algo bastante relevante. Una de las cualidades más destacadas de Android es que es un sistema operativo back-end que termina de fraguar cada compañía de teléfonos móviles para adaptarlos a sus terminales haciendo ellos la parte front-end. Convirtiéndolo en un sistema operativo muy versátil.

En el año 2003 un grupo formado por Andy Rubin, Rich Milner, Nick Sears y Chris White fundaron, en Palo Alto, Android Inc. comenzando así el idilio de Android. Dos años después, entraron a formar parte de un proyecto de Google que consistía en la compra de startups del sector móvil. En Noviembre de 2007 se presentó el sistema operativo Android y en Octubre de 2008 se estrenó en su primer terminal, un HTC Dream.

En menos de 4 años son ya 8 las actualizaciones que ha habido de Android. Cabe destacar que la versión más extendida es la Gingerbread (versión 2.3).

Versiones

A continuación explicamos brevemente en la Tabla 2 los distintos cambios que ha surgido Android en sus diferentes versiones.

1.0	Liberado el 23 de septiembre de 2008
1.1	Liberado el 9 de febrero de 2009
1.5 (Cupcake) **Basado en el kernel de Linux 2.6.27**	El 30 de abril de 2009, la actualización 1.5 (Cupcake) para Android fue liberada. Entre sus nuevas características y destacan: • Nueva interfaz de usuario • Grabación y reproducción de vídeo • Sincronización con cuentas YouTube y Picasa • Teclado con predicción de texto • Soporte para Bluetooth A2DP y AVRCP • Transiciones de pantalla animadas
1.6 (Donut) **Basado en el kernel de Linux 2.6.29**	El 15 de septiembre de 2009, se liberó el SDK 1.6 (Donut). En esta actualización se incluyó: • Un reformado Android Market • Una interfaz integrada de cámara, filmadora y galería • Actualización de la búsqueda por voz • Experiencia de búsqueda mejorada • Actualización de soporte para CDMA/EVDO, 802.1x, VPN y text-to-speech • Soporte para resoluciones de pantalla WVGA • Mejoras de velocidad en las aplicaciones de búsqueda y cámara • Framework de gestos y herramienta de desarrollo GestureBuilder
2.0 / 2.1 (Eclair) **Basado en el kernel de Linux 2.6.29**	El 26 de octubre de 2009, el SDK 2.0 (Eclair) fue liberado. Los cambios incluyeron: • Velocidad de hardware optimizada • Soporte para más tamaños de pantalla y resoluciones • Interfaz de usuario renovada • Navegador con soporte para HTML5 • Nuevas listas de contactos • Mejoras en Google Maps • Soporte para Microsoft Exchange • Soporte integrado de flash para la cámara • Zoom digital • Teclado virtual mejorado • Bluetooth 2.1 • Fondos de pantalla animados

2.2 (Froyo) **Basado en el kernel de Linux 2.6.32**	El 20 de mayo de 2010, el SDK 2.2 (Froyo) fue liberado. Los cambios incluyeron: • Optimización general del sistema Android, la memoria y el rendimiento • Mejoras en la velocidad de las aplicaciones, gracias a la implementación de JIT • Integración del motor JavaScript V8 • Funcionalidad de Wi-Fi hotspot y tethering por USB • Soporte para pantallas de alta resolución (720p)
2.3 (Gingerbread) **Basado en el kernel de Linux 2.6.35.7**	El 6 de diciembre de 2010, el SDK 2.3 (Gingerbread) fue liberado. Los cambios incluyeron: • Actualización del diseño de la interfaz • Soporte para pantallas grandes • Soporte nativo para telefonía VoIP SIP • Soporte para NFC (Near Field Communication) • Soporte mejorado para desarrollo de código nativo • Cambio de sistema de archivos de YAFFS a ext4
3.0 / 3.1 / 3.2 (Honeycomb)	Esta actualización fue pensada esencialmente para tablets, a destacar: • Mejor soporte para tablets • Sistema multitarea mejorado • Mejoras en el navegador web predeterminado • Soporte para videochat mediante Google Talk • Mejor soporte para redes Wi-Fi, así como guardar una cada configuración independiente para cada SSID • Añade soporte para una gran variedad de periféricos y accesorios con conexión USB: teclados, ratones, hubs, dispositivos de juego y cámaras digitales. Cuando un accesorio está conectado, el sistema busca la aplicación necesaria y ofrece su ejecución. • Los widgets pueden redimensionarse de forma manual sin la limitación del número de cuadros que tenga cada escritorio.
4.0 **(Ice Cream Sandwich)**	La última versión de Android fue liberada el 15 de Noviembre de 2011 • Versión que unifica el uso en cualquier dispositivo, tanto en teléfonos, tablets, televisores, netbooks, etc.

	• Aceleración gráfica por hardware • Añadido un gestor del tráfico de datos de internet. El entorno le permite establecer alertas cuando llegue a una cierta cantidad de uso y desactivación de los datos cuando se pasa de su límite. • Mejora de configuración de las notificaciones • Android Beam es la nueva característica que nos permitirá compartir contenido entre teléfonos. Vía NFC • Reconocimiento de voz del usuario • Soporte nativo para el uso de lápiz táctil.

Tabla 2. Versiones de Android. 42

Aplicaciones Android

Las aplicaciones Android tienen un patrón de construcción impuesto por el propio sistema.

Esquema de una aplicación

El patrón impuesto por el sistema consiste en una parte perteneciente al front-end en XML un el back-end que ejecuta el código desarrollado en Java.

Los archivos XML que gestionarán el front-end de la aplicación se guardarán en diversas carpetas en función de su naturaleza. Las carpetas donde se alojan estos archivos se organizan de la siguiente forma:

- anim: Donde se guardarán las imágenes que queramos introducir en los XML. Estando aquí toman un identificador con el cual se identifican en todos los archivos del proyecto.
- drawable: En las cuales se guardarán las imágenes de la aplicación.
- layout: En esta carpeta están todos los layouts de la aplicación.
- menu: Aquí es donde están todos los menús del proyecto.
- values: Las variables de los XML se guardan en esta carpeta.
- xml: En dicha carpeta se guardan los archivos de preferencias.

Para poder realizar el back-end en Java, Android nos proporciona la API para interactuar con el sistema que ya hemos comentado en puntos anteriores. Con esta estructura, Android nos ofrece la facilidad de desarrollar para Java con la potencia de la programación orientada a objetos y también la facilidad de componer la apariencia de los *"widgets"* que formen parte de la aplicación mediante los layouts en XML, al más puro estilo de los formatos web.

Componentes de una aplicación

Los componentes son las partes principales de una aplicación Android. Cada componente utilizado ha de ser declarado en el Manifest de la aplicación para que pueda ser utilizado, también ha de declararse en el Manfest la versión desde la cual la aplicación es compatible, puede funcionar correctamente o alguna configuración de hardware necesaria.

Los componentes de una aplicación pueden ser de cuatro tipos distintos

Activities

Un a*ctivity* es la componente principal encargada de mostrar al usuario la interfaz gráfica, es decir, una actividad sería el equivalente a una ventana, y es el medio de comunicación entre la aplicación y el usuario. Se define un *activity* por cada interfaz del proyecto. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado (que se guarda en ./res/layout) para poder ser tratados en la clase NameActivity.class, que hereda de la clase Activity.

Services

Un *service* es una tarea no visible que se ejecuta siempre en *background*, incluso cuando la actividad asociada no se encuentra en primer plano. Tiene un hilo propio (aunque no se puede ejecutar solo), lo que permite llevar a cabo cualquier tarea, por pesada que sea. No necesita interfaz. Todos heredan de la clase Service

Content providers

Los *content providers* son el mecanismo que tiene Android para comunicar datos entre distintas aplicaciones. El propio Android trae (desde sus versiones 2.0+) incluidos la agenda, los SMS y el listado de llamadas mediante el método de los Content Providers, simplemente una aplicación debe acceder a la URI donde tenemos el Content Provider y una vez tengamos acceso pedirle los datos que necesitemos. Todos heredan de la clase ContentProvider.

Broadcast receivers

Un *broadcast receiver* es una componente que nos muestra notificaciones de los eventos que suceden en nuestro terminal móvil, como por ejemplo aviso de batería baja o que el teléfono está gastando la batería más rápido de lo que carga. Todos heredan de la clase BroadcastReceiver

Ciclo de vida de una aplicación

El ciclo de vida de una aplicación es el período comprendido desde que nace la idea de la aplicación hasta que se deja de dar soporte a la misma, o lo que es lo mismo desde que deja de hacerse el mantenimiento necesario para que pueda ser usada por los usuarios.

Dicho ciclo de vida se divide en cinco etapas principales, que son las siguientes: análisis, diseño, codificación, pruebas y mantenimiento.

Especificación y análisis

La fase de análisis es la primera fase a realizar, dicha fase consiste en concretar las características principales de la aplicación.

Así una etapa de análisis bien desarrollada y bien definida logrará que queden menos problemas sin resolver en el resto de etapas y por lo tanto el desarrollo de la aplicación sea más rápido ya que reducirá malentendidos en interpretaciones de las funciones a realizar.

En esta etapa el cliente final es parte activa, ya que es el encargado de determinar que las funcionalidades modeladas son las esperadas por el destinatario, él mismo, así la comunicación bidireccional entre cliente y analistas es de vital importancia para esta etapa. Es interesante que los analistas tengan una amplia capacidad de compresión de problemas de cualquier campo para que así la comunicación con el cliente sea más fluida e intentando perder el mínimo tiempo posible en adquirir los conocimientos mínimos para hacer la aplicación.

Esta etapa puede marcar la diferencia entre un fracaso de aplicación, que incluso no llegue a ver la luz, y una aplicación puntera.

Diseño

En esta fase se hará la especificación de requisitos producido en el análisis, definiendo el cómo se cumplirán los requisitos así como la estructura software para conseguirlo.

En la fase de diseño no se escribe código de ningún tipo siendo un paso intermedio entre el análisis y la codificación, que es la fase en la que se desarrolla el código.

En la fase de diseño simplemente se toman decisiones tales como el uso de la arquitectura de hardware o el sistema operativo entre otras. Pudiendo describir la aplicación a distintos niveles de abstracción, si bien es cierto que normalmente el diseño de la arquitectura se hace a muy alto nivel. La modulación del sistema se realiza a un nivel intermedio de detalle, fijando aquí el lenguaje de programación. El nivel más detallado de diseño se considera ya un primer paso de la codificación ya que si fuere un lenguaje orientado a objetos, se definirían las clases, con sus atributos y métodos.

Codificación

En la etapa de codificación se programa la aplicación. Durante esta fase lo único que se hace es plasmar sobre el lenguaje concreto todo lo establecido en las etapas anteriores. El tiempo dedicado a la etapa de codificación depende en gran medida del lenguaje utilizado, ya que a mayor nivel del lenguaje, menor será el tiempo invertido en el desarrollo de un método.

En la etapa de codificación debemos hacer las pruebas mínimas para tener la certeza de que cada módulo cumple con ciertas funcionalidades básicas. Algunos fallos que puedan encontrarse nos obligarán a volver a la fase de diseño.

Pruebas

Las pruebas a realizar, en función de su naturaleza, se pueden dividir en varios tipos

Pruebas unitarias

Son aquellas que realizan pruebas de cada parte del software, Se aplican a procedimientos, funciones y módulos. Sirven para saber que el funcionamiento, a nivel unitario, es el esperado.

Pruebas de integración

En las pruebas de integración intentamos empezar a fusionar distintos módulos y comprobar su correcto funcionamiento una vez toda la aplicación está funcionando.

En esta fase de prueba aparece la llamada Beta Test. Consiste en instalar la última versión estable de la aplicación y probar todas las funcionalidades repetidas veces. Con ello se pretende encontrar fallos, inestabilidades y generalmente respuestas no deseadas de la aplicación. Los errores que aparezcan en la Beta se comunican a los desarrolladores para que los depuren.

Instalación y paso a producción

La penúltima fase es la de instalación. En dicha fase se pone la aplicación en las circunstancias en las que estará normalmente, por lo tanto habrá que instalarlo en la máquina adecuada y con la configuración necesaria para ello. Esta fase tiene mayor o menor complejidad en función de la naturaleza de la instalación de la aplicación.

Mantenimiento

La fase de mantenimiento es la que se dedica a controlar, mejorar y optimizar la aplicación, una vez instalada. En esta fase también depuraremos errores que no hayan sido eliminados previamente, en el caso de que los hubiera.

La fase de mantenimiento es la última del ciclo de vida de una aplicación y su duración y coste dependerá en gran medida del trabajo que se haya realizado en el resto de fases del ciclo de vida.

También son parte del mantenimiento las mejoras del software, así como el lanzamiento de nuevas versiones de la aplicación. Se puede llegar al caso de que al querer agregar una funcionalidad en una nueva versión haya que retocar las fases previas del ciclo de vida de la aplicación. En ese caso será necesario revisar las fases desde la cual se ha introducido la modificación ya que se han podido ver afectadas, hasta que volvamos a tener una versión estable en la fase de mantenimiento.

La duración de la fase de mantenimiento generalmente está fijada con el usuario final, si es una aplicación de escasa distribución, o en el caso de ser una aplicación de distribución masiva, generalmente dura hasta que se saque una nueva versión de dicha aplicación o hasta que deje de ser rentable el mantenimiento de la misma.

Con la finalización de la fase de mantenimiento finaliza el ciclo de vida de la aplicación y pasa a ser una aplicación muerta.

6. La aplicación CGTel

Especificación de requisitos

La especificación de requisitos se divide en dos grandes bloques, entre ambos se logra una descripción del comportamiento de la aplicación que está desarrollándose. Ambos bloques son los siguientes:

Requisitos funcionales

Los requisitos funcionales son el compendio de funcionalidades que tiene la aplicación de cara al usuario.

El conjunto de requisitos funcionales de CGTel divididos principalmente por los módulos del menú, son los siguientes.

Recomendador de tarifas

El usuario puede ver el precio que tendría que pagar con cada una de las tarifas de la base de datos. El primero día desde el que se empiezan a tomar los datos es el día de facturación, por lo que es especialmente útil y fiable los días previos al día de facturación.

Grupos

Los grupos nos permiten ver a quién le va a salir la llamada más barata de entre un conjunto de usuarios de CGTel elegidos por el usuario. Además si el terminal óptimo no fuera el nuestro, podremos hacerle una notificación con una aplicación de mensajería externa seleccionada por el usuario. Del mismo modo, si los usuarios nos proporcionan permisos para poder gestionar su consumo, podríamos ver un desglose del gasto actual de cada uno de los integrantes de dicho grupo.

Facturación

Nos permite ver un desglose detallado de la factura desde el día de facturación hasta el día actual. Incluye, para cada franja de la tarifa, una barra de progreso con el número de minutos hablados y el número de minutos máximos. También habrá una para mensajes, en el caso de tenerlos gratuitos.

Estimación

Nos hace un cálculo estimado del número de minutos que hablaremos a final de mes teniendo en cuenta las llamadas realizadas hasta ahora. Mostrándonos una gráfica que con los minutos hablados hasta ahora y la estimación realizada, así como el gasto actual y el gasto estimado a final de mes.

Notificaciones

Las notificaciones llegarán al usuario cada vez que finalice una llamada saliente. En dicha notificación se reflejará la información sobre el coste de la última llamada realizada.

Coste llamadas

Para todas las funcionalidades comentadas anteriormente, la aplicación ha de ser capaz de calcular el coste de cada llamada realizada, tanto por el usuario del propio terminal, como de otro que utilice CGTel a partir de la duración, el listado del resto de llamadas de ese mes y la tarifa del usuario.

Requisitos no funcionales

Los requisitos no funcionales son atributos de calidad, siendo así características no necesarias para la implementación de la aplicación, pero no por ello menos útiles, como puede ser la usabilidad, o menos importantes, como puede ser la adaptabilidad.

Los requisitos no funcionales CGTel son los siguientes:

Estabilidad

En condiciones normales, disponibilidad de internet y que el terminal se encuentre en un estado óptimo de rendimiento, la aplicación CGTel no tiene ningún problema de estabilidad.

Seguridad

Hemos codificado la base de datos con MD5. MD5 es un algoritmo de codificación de un único sentido, o lo que es lo mismo codifica pero no decodifica. Es muy útil ya que para poder acceder a los datos de la base de datos hace falta tener los datos originales.

Coste

La aplicación es totalmente gratuita convirtiendo el costo en otro requisito no funcional de CGTel.

Usabilidad

Gracias a la estructura del menú principal y la sencillez de los submenús, la aplicación tiene una alta usabilidad.

Escalabilidad

Gracias a la modulación de Java y la correcta implementación del código la escalabilidad de CGTel es alta.

Ya que si necesitáramos introducir alguna nueva funcionalidad solamente tendremos que agregar un nuevo icono en el menú principal, en el caso de que no tenga ninguna relación con cualquiera de los otros cuatro, o incluirlo en el módulo correspondiente si formara parte de alguno.

Requisitos no funcionales que no son viables

En nuestro caso tenemos algunos requisitos no funcionales importantes en los cuales, por la naturaleza de CGTel, es imposible que sean parte de nuestros requisitos no funcionales.

Uno de ellos es la mantenibilidad, que es la facilidad de mantenimiento de una aplicación. Al ser CGTel una aplicación que necesita tener actualizadas las

distintas tarifas que hay en el mercado, tenemos un alto nivel de dependencia de ellas. Ya que si una operadora modifica o agrega alguna de sus tarifas hay que modificarla de inmediato para que la aplicación no se torne en obsoleta.

Por ello es obvio que la mantenibilidad de la aplicación es muy mala.

Diseño

Una vez establecidos todos los requisitos así como los módulos claramente diferenciados de la aplicación pudimos pasar a la fase de diseño. La primera aproximación a la aplicación a modo de prototipo fue determinada en noviembre 2011.

Desde un principio el concepto de cómo diseñar la aplicación estuvo muy claro, debíamos separar las tareas propias de Android de las que fueran propias de las funcionalidades que requería la aplicación. Esta decisión estuvo motivada fundamentalmente por la falta de experiencia con Android y servicios web de los integrantes del grupo, al no saber si podría ser un obstáculo a la hora de avanzar con las distintas funcionalidades que debía implementar la aplicación se opto por separar completamente lo que eran funcionalidades de Android tales como interfaces, permisos o acceso a datos o red.

Antes de comenzar con los trabajos de implementación hubo que diseñar una estructura de datos para las tarifas que ofertan las compañías actualmente de forma que se adaptara de manera genérica a todas por igual. La solución tras estudiarlas distintas tarifas fue crear tarifas compuestas a su vez de franjas horarias. Todas las tarifas telefónicas mantienen eso en común pueden facturar en distintas franjas con límite de gasto y en otras sin él, tener en conjunto un número de minutos, pero lo único que todas tienen en común es su división en franjas. En el caso básico una tarifa estaría compuesta por una única franja que abarcaría las 24h los 7 días de la semana. Para poder entender la estructura es más fácil visualizarlo en el ejemplo de la Figura 34 donde tenemos una tarifa con dos franjas entresemana y una 24h para los sábados y domingos

<table>
<tr><th></th><th>Lunes</th><th>Martes</th><th>Miércoles</th><th>Jueves</th><th>Viernes</th><th>Sábado</th><th>Domingo</th></tr>
<tr><td>00.00-
08.00</td><td colspan="5">Franja
Tardes
18:00 - 8:00</td><td rowspan="3" colspan="2">Fines de semana
y festivos
24h</td></tr>
<tr><td>08.00-
18.00</td><td colspan="5">Franja
Mañanas
8:00 - 18:00</td></tr>
<tr><td>18.00-
00.00</td><td colspan="5">Franja
Tardes
18:00 - 8:00</td></tr>
</table>

Figura 34. Ejemplo de la estructura de una tarifa con cuatro franjas.

En un primer instante, los datos para compartir en la base de datos se determinaron de forma que todos los resultados se pudieran obtener mediante simples consultas. El ejemplo más claro lo tenemos con las cargas de las llamadas de los usuarios a la base de datos, en principio será viable cargar en cada llamada el coste asociado, sin embargo, siempre es preferible cargar el listado de llamadas y viendo la tarifa que tiene activa el usuario proceder al calculo del consumo acumulado. De esta forma conseguimos unos datos más actualizados sin necesidad de cargar información adicional a la base de datos. Al igual que con esta primera concepción de los datos a cargar pronto nos dimos cuenta que estas cargas obtenían datos sin tener en cuenta información actual y que perjudicaba a la veracidad sobre los costes de las llamadas por lo que realizamos cambios similares en la aplicación / base de datos similares al comentado con el caso de las llamadas

La otra parte importante de cara al diseño fue concebir la idea de la gestión y recomendación a nivel de usuarios de la aplicación y grupos. Para simplificar, decidimos que cada usuario sería capaz de conceder permisos al modo grupo y que la aplicación no podría tramitar solicitudes entre usuarios Por otro lado definimos como comunes entre el modo individual y el modo grupo la recomendación de llamada, el funcionamiento aquí debiere ser el mismo sólo que en el modo de grupo se compararía el mejor resultado entre todos los integrantes del grupo.

En el modo grupo dejamos la parte de la administración. Todas las consultas y referencias a conocer datos de consumo y predicciones de otros usuarios irán destinadas a modo grupo y nunca al plan individual. Tiene sentido pensar que la consulta de consumo de varios terminales tiene sentido siempre y cuando se esté administrando una cierta “entidad” con varios miembros que nos permitan ver en todo momento su consumo de voz y mensajes de texto, entendiendo por entidad desde grupos de familias donde se lleve una estadística del consumo familiar hasta una pequeña empresa donde se quiera vigilar el gasto telefónico en una pequeña flota de dispositivos móviles.

Para concluir esta fase, decidimos que la aplicación no tiene sentido sin disponibilidad de una conexión a internet ya que todos los datos comunes entre terminales y tarifas están albergados en una base de datos externa, y no se alberga copia en local. Es por eso, que se optó por agregar un login a la aplicación que validara el número de teléfono del terminal y además autenticara mediante contraseña al usuario, en caso de no disponer de internet no podremos acceder a la aplicación.

Implementación

Como hemos comentado en el punto anterior, la intención del grupo fue desde un principio manejar una interfaz a modo de esqueleto de la aplicación desde la cual llamar a las respectivas funcionalidades que debía brindar la aplicación. Esta interfaz no fue ni mucho menos definitiva, pero nos proporcionaba ese acceso

directo y que diferenciaba y aislaba las distintas funcionalidades. Así desde las primeras versiones podemos ver como la aplicación mostró un menú principal donde podamos acceder a los registros de usuarios, recomendación de tarifa / llamadas, estadísticas y configuración y selección de tarifas.

Así, cada tarea quedaría excluida del resto y se podría trabajar sobre cada una de las funcionalidades por separado.

Los trabajos de implementación se han dividido por funcionalidades:

- Cuerpo de la aplicación
- Obtención de datos del terminal (contactos, listados de llamadas)
- Interconexión con base de datos
- Login del usuario
- Acceso y guardado de configuraciones
- Recomendador de llamadas
- Notificaciones y receivers
- Módulo de predicción y estadísticas
- Recomendador de tarifas
- Administración / modo grupo
- Interfaces gráficas: Layouts y diálogos

En los siguientes apartados podemos ver como se ha estructurado por paquetes y cómo se han ido implementando cada una de las funcionalidades:

Patrones y algoritmos implementados vistos en la carrera

Algoritmos: Ordenación, método de Burbuja.

Al realizar el recomendador de tarifas hemos visto muy útil, de cara al usuario, ordenar las tarifas por precios de manera ascendente. Así las primeras tarifas que verá el usuario serán las óptimas y es mucho más sencillo encontrar tarifas idóneas, es importante destacar que la tarifa propia se marcará con una estrella, así todos los datos relevantes para el usuario serán fácilmente localizables.

Para ello hemos utilizado el método de burbuja ya que era el más sencillo de implementar. Es importante destacar que al estar ordenando facturas no es relevante la diferencia entre un orden n^2 o un orden $n*\log_n$, si no fuera por esto, si hubiéramos querido ordenar llamadas por ejemplo, habríamos implementado otro algoritmo de ordenación con mejores resultados en tiempo.

El método de la burbuja consiste en ir haciendo una “sublista” ordenada dentro de la lista principal. Lo que de un modo más sencillo significa que cuando vamos a ver dónde colocar un elemento que está en la posición j sus j-1 elementos

anteriores estarán ordenados. Por lo que solamente habrá que colocarlo en la posición correspondiente, irá avanzando hacia la posición 0 como una burbuja hasta encontrar su sitio correcto, de ahí el nombre del algoritmo.

Patrones

Modelo - Vista – Controlador

Modelo Vista Controlador [43] (MVC) es un patrón que divide la aplicación en tres componentes independientes interconectados entre sí como se puede ver en la Figura 35.

- Modelo. Contiene la lógica de la aplicación, estructuras de datos y todos los componentes para representar el estado de la aplicación.
- Vista. Según el estado de la aplicación, muestra la interfaz con la que interactúa el usuario.
- Controlador. Desencadena acciones a partir de la interacción del usuario con la vista.

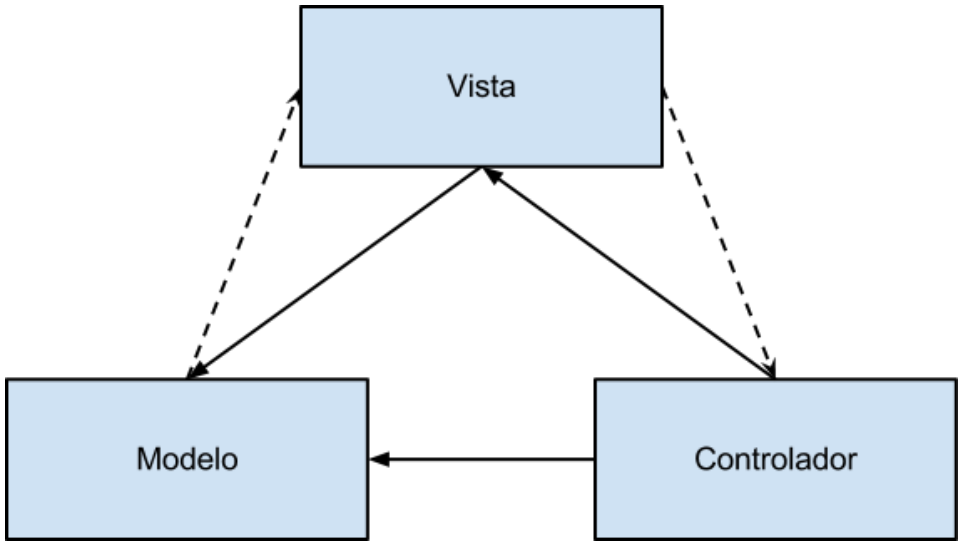


Figura 35. Esquema del patrón MVC.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo, una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original.

El patrón Modelo-Vista-Controlador está implementado en Android por defecto. Teniendo en cuenta que este patrón se puede aplicar de formas muy variadas, en Android se utiliza de una forma específica que puede tener diferencias con la definición original. En una aplicación Android podemos definir los siguientes elementos:

Vista

La interfaz de una aplicación Android se define mediante archivos xml y otro tipo de archivos que son almacenados en la carpeta “res”. Dichos archivos se pueden modificar e incluso se pueden agregar nuevos de forma que la interfaz mejore sin necesidad de modificar el resto de implementación.
Las diferentes pantallas de la interfaz están definidas mediante archivos xml en la carpeta “resources/layout”, así como los diálogos.
Como se ha explicado en puntos anteriores estos recursos son automáticamente catalogados en la clase “R.java”.

Controlador

A esta parte corresponden las clases que extienden de la clase Activity de Android. Dichas clases están relacionadas con la interfaz. Para ello se realiza la carga de un xml que contenga la definición del layout y se definen las diferentes acciones que tomará el controlador en función de la interacción del usuario con la interfaz.

Modelo

En el modelo se define en código Java separado de las actividades. Esta es la parte del código que sería portable a cualquier sistema que lea el mismo código fuente, por ejemplo, a una web donde realicemos las mismas operaciones que desde nuestro dispositivo Android.

Adapter

Para la implementación de este proyecto también hemos hecho uso del patrón Adapter en especial para la presentación de listas de elementos. Este patrón es utilizado en Android para crear la interfaz o vista asociada a un conjunto de datos.

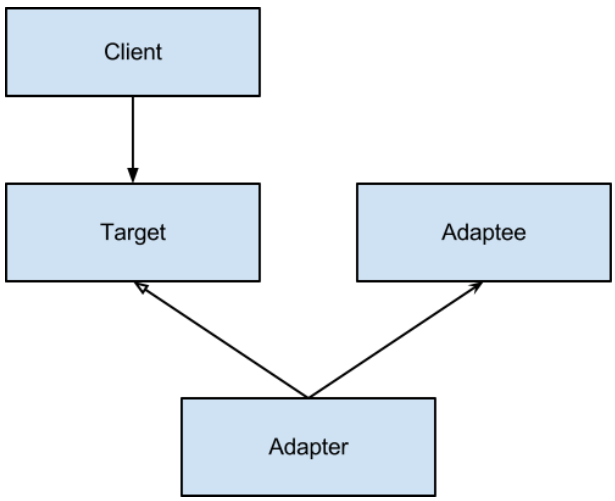


Figura 36. Esquema básico del patrón Adapter.

Android proporciona la interfaz Adapter [44] y un conjunto de clases que la implementan: ArrayAdapter<T>, ListAdapter, CursorAdapter,

SimpleCursorAdapter, HeaderViewListAdapter, BaseAdapter, ResourceCursorAdapter, SimpleAdapter, SpinnerAdapter y WrapperListAdapter.

La aplicación del patrón adapter implica implementar la interfaz Adapter o extender las clases que la implementan redefiniendo sus métodos. En nuestro caso se ha hecho uso de la clase ArrayAdapter para adaptar ArrayList de objetos específicos a una interfaz con un layout específico definido en un xml. Para ello se han implementado los elementos que se muestran en la Figura 36.

- Client. Realiza operaciones sobre "Target". En nuestro caso el Cliente es la interfaz en sí, en concreto la clase que realiza la carga de una interfaz mediante un xml debe asignar un adaptador (Adapter) al elemento de la interfaz correspondiente.
- Target. Define la interfaz específica del dominio que "Client" usa.
- Adaptee. Define una interfaz existente que necesita adaptarse. Por ejemplo una lista de contactos o de llamadas.
- Adapter. Adapta "Adaptee" a la interfaz "Target". Esta clase se encarga de adaptar un conjunto de elementos a una la interfaz definida en "Target". De esta forma se le puede asignar una interfaz a cada objeto adaptado en función de los datos que contenga. Por ejemplo en una lista de llamadas se puede asignar un icono a cada elemento de la lista en función del tipo de llamada o cualquier otro parámetro que se estime oportuno.

Paquetes y clases

A continuación vamos a explicar la distribución interna de la aplicación CGTel para Android, para ello explicaremos cómo están distribuidos los paquetes y clases.

En el caso de los paquetes y clases tienen distribución funcional, por ello hemos dividido todo el conjunto de clases en 6 paquetes principales, dichos paquetes con sus correspondientes clases son los detallamos a continuación.

activity

Este paquete comprende todas las clases referentes a actividades que puede llevar a cabo el usuario. Dichas clases se agrupan por paquetes, cada paquete contiene las clases necesarias para la implementación de la actividad. Paquetes:

- *comparador*
  En este paquete están contenidas las clases que encargadas de presentar al usuario la comparación de tarifas y el listado de llamadas.
- *config*
  Contiene las clases referentes a las preferencias de la aplicación y a la configuración de la tarifa.
- *contactos*
  Comprende la administración de contactos y grupos.

- *estadísticas*
  En esta pantalla se muestran las gráficas del consumo, la predicción de la próxima factura y un desglose de la facturación.
- *login*
  Pantalla de inicio de sesión de la aplicación.
- *main*
  Pantalla principal de la aplicación desde la que se puede acceder a realizar otras actividades principales: comparador, configuración, contactos y estadísticas.

adapter

En la aplicación se hace uso de numerosos adapters para las distintas listas mostradas en la interfaz. Dichas clases adaptadoras están definidas en este paquete facilitando su localización y su reutilización.

dialog

Este paquete contiene los diálogos personalizados que se han utilizado en las distintas notificaciones que se le pueden presentar al usuario en el programa.

reciver

Contiene la implementación del notificador de cada llamada recibida, así como arrancar el servicio de actualización en el arranque del terminal/aplicación si este no estuviera ya disponible.

service

Contiene la implementación del servicio de actualización que permite tener los datos de las llamadas del usuario siempre sincronizado con la base de datos.

util

En él esta contenida la lógica del programa y las clases auxiliares utilizadas.

Manual de usuario

Instalar la aplicación

Para instalar la aplicación solamente tenemos que descargar el .apk, para ello tendremos que acceder a la página web y descargar la última versión disponible de CGTel en nuestro terminal móvil, al finalizar la descarga saldrá el menú de instalación estándar para cualquier aplicación Android. Tras unos segundos la instalación finalizará y podremos exprimir al máximo las utilidades de CGTel.

Ejecutar la aplicación. Primeros pasos en CGTel

Una vez instalada la aplicación vamos a ver cuáles tienen que ser nuestros primeros pasos por CGTel, lo primero que tendremos que hacer es ejecutar la aplicación, ahora nos saldrá la siguiente ventana Figura 37, ahora tendremos que introducir nuestros datos de usuario, para ello introduciremos nuestro número de teléfono y nuestra contraseña, si el número introducido no está en la base de datos se agregará y se le asignará la contraseña introducida.

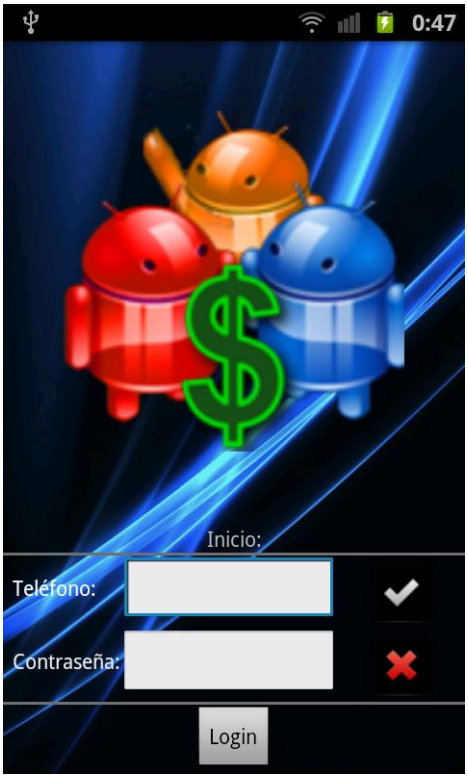


Figura 37. Pantalla Login de la aplicación CGTel.

Una vez introducido el usuario y la contraseña, si es la primera vez que ejecutamos la aplicación se nos llevará directamente a la pantalla de selección de tarifa (Figura 39). De no ser el caso accederemos la ventana principal del programa (Figura 38). Para completar nuestro primer login como usuario ahora deberemos elegir la tarifa y el día de facturación. Para ello seleccionaremos nuestro operador y luego la tarifa que nos corresponda.

Figura 38. Pantalla principal.

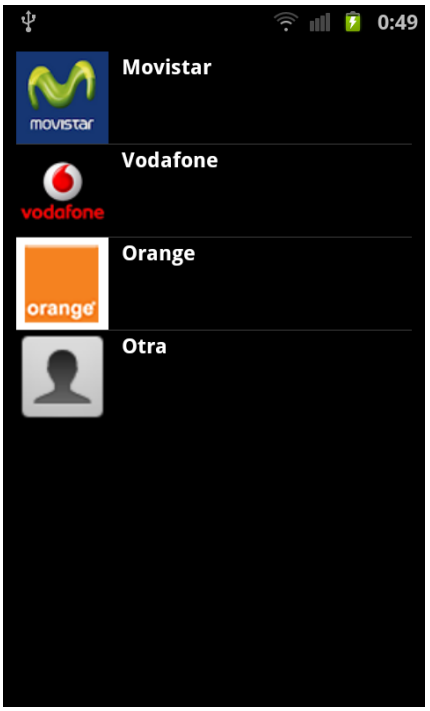


Figura 39. Pantalla de selección de tarifa.

Una vez tengamos seleccionada la tarifa vamos a seleccionar nuestro día de facturación para ello volveremos una vez estemos en el menú de configuración (Figura 40) y a continuación pulsaremos sobre día de facturación y seleccionar el día correspondiente en la lista como en la Figura 41.

Figura 40. Pantalla de Configuración.

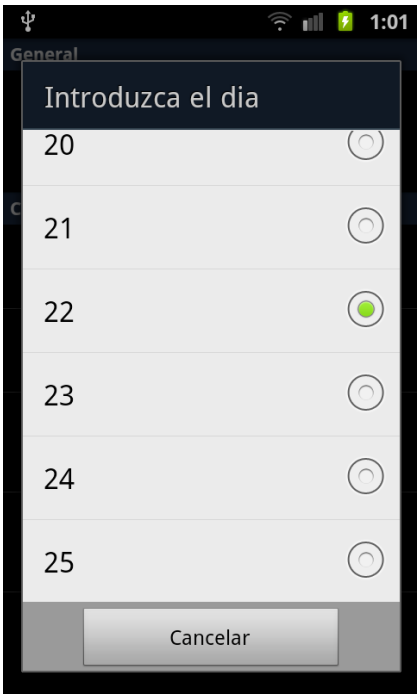


Figura 41. Diálogo de selección del día de facturación.

Ya hemos inicializado la aplicación y configurado, desde este momento podemos disfrutar de sus ventajas, las cuales las explicaremos a continuación.

Funcionalidades más destacadas

CGTel es una aplicación muy útil de cara al usuario ya que le ayuda de manera sencilla a gestionar su consumo telefónico, a continuación vamos a explicar las funcionalidades que hacen que sea así.

Grupos: Gracias a los grupos podemos ver a que usuario de CGTel, de un grupo seleccionado por el usuario, le cuesta menos realizar una llamada.

Recomendador de tarifa: El recomendador de tarifas nos dice, según nuestro consumo actual, con cuál de las tarifas de la base de datos tendremos menor coste a final de mes.

Estimador: El estimador nos hará una estimación del gasto a final de mes en función de los minutos hablados hasta ahora.

Facturación actual: Desglose de la factura actual, incluyendo minutos hablados, llamadas realizadas, mensajes enviados así como los costes y descuentos pertinentes.

Manual de usuario (Paseo guiado por CGTel)

A continuación vamos a dar un paseo parándonos detenidamente a explicar cada una de las funcionalidades de CGTel.

Vamos a comenzar el paseo guiado por el menú principal de la aplicación, podemos apreciar que dicho menú tiene cuatro iconos: configuración, contactos / grupos, recomendador de tarifas y estadísticas. Dichos iconos son los que nos van a llevar a las principales funcionalidades de la aplicación

Si hacemos clic en contactos y grupos, habrá que esperar a que el terminal cargue los contactos con CGTel. Una vez cargados nos mostrará un listado con todos los usuarios que están en la agenda del teléfono y que también tiene la aplicación instalada (Figura 42). Estos serán los usuarios que se podrán beneficiar junto al propio usuario de las funcionalidades de CGTel.

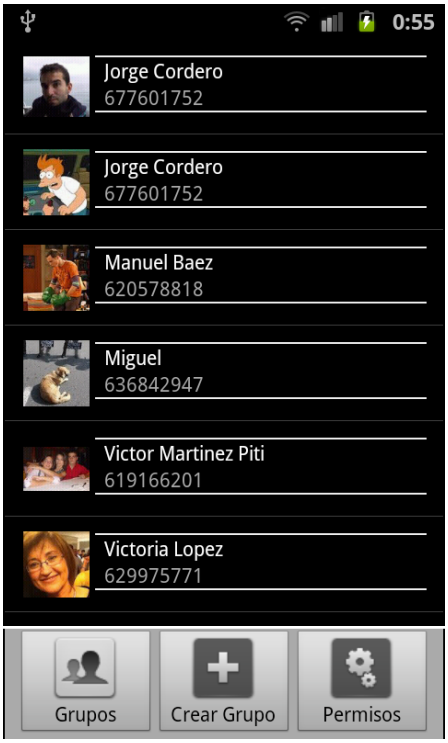


Figura 42. Pantalla de contactos de CGTel.

Entre los usuarios con CGTel tendremos varias funcionalidades, podemos acceder a dichas funcionalidades pulsando sobre el contacto y agregan a llamar las siguientes.

Agregar a grupo: Agrega al terminal a un nuevo grupo o a uno ya existente.

Recomendación de llamada: Comparación uno a uno entre el contacto y el terminal propio para ver a cuál de los dos le costaría menos realizar la llamada.

Conceder permisos de administración: Permite que el contacto seleccionado tenga permisos de administración sobre nosotros pudiendo acceder así a nuestros datos de facturación.

Si ahora hacemos un gesto de derecha a izquierda, arrastramos el dedo por la pantalla de derecha a izquierda o pulsamos sobre el botón “Grupos”, vamos a la ventana de grupos, en dicha ventana tenemos todos los grupos de usuarios que hemos creado, Figura 43. Para acceder a las opciones que nos ofrecen los grupos tenemos que hacer clic sobre uno de ellos, apareciendo una ventana como la que se muestra a continuación en la Figura 44.

Figura 43. Pantalla de grupos.

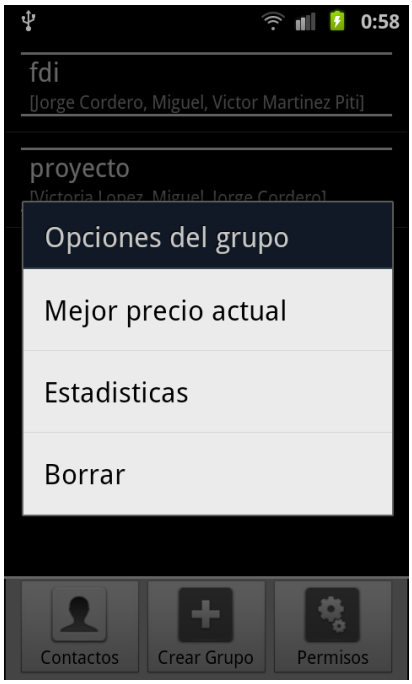


Figura 44. Diálogo de opciones de grupo.

Las opciones que se nos presentan en un grupo son las siguientes:

Mejor precio actual: Nos permitirá ver a que miembro del grupo le sale más barata la siguiente llamada y si no fuéramos nosotros nos da la opción de hacerle una notificación, externa a CGTel por ejemplo mediante un SMS, un correo o cualquier aplicación dedicada a notificaciones como WhatsApp.

Estadísticas: Si tenemos permisos sobre los miembros del grupo aquí podemos ver sus estadísticas de gasto y facturación

Borrar: Permite eliminar el grupo seleccionado

Volvemos ahora al menú principal y pulsemos sobre el icono del recomendador de tarifas. En estos momentos el terminal calculará lo que nos estaría costando nuestro consumo actual, tanto de llamadas como de mensajes, en cualquiera de las tarifas de la base de datos. Una vez calculados los datos se mostrarán ordenadas de menor a mayor coste de tarifa, marcando con una estrella nuestra tarifa actual. Podemos ver en la Figura 45 como se han ordenado las

tarifas por coste y que la aplicación nos recomienda por el consumo actual la tarifa Vodafone XS a un precio de 24,13 €. Del mismo modo podemos ver con facilidad que nuestra tarifa en la captura es Orange Habla y Navega 29 (Autónomos), marcada con la estrella, y que la factura con las llamadas hasta la fecha de hoy ascenderá hasta los 34,75 €.

Figura 45. Listado de tarifas y costes.

Este recomendador es más útil cuantos menos días queden hasta la siguiente facturación ya que la comparativa será más completa.

Vamos a ver ahora las estadísticas que nos muestra CGTel, para ello dentro del menú principal, y como es evidente, hay que pulsar en estadísticas. Podemos ver en la figura 50 que lo primero que hace el terminal al pulsar sobre estadísticas es realizar los cálculos necesarios para mostrar la gráfica de llamadas realizadas (azul) y segundos hablados(verde). Junto a estos datos se nos muestran también una estimación de los segundos que hablaremos en los días sucesivos (rojo) y las llamadas que realizaremos (amarillo), en función de las llamadas que hemos realizado desde el último día de facturación.

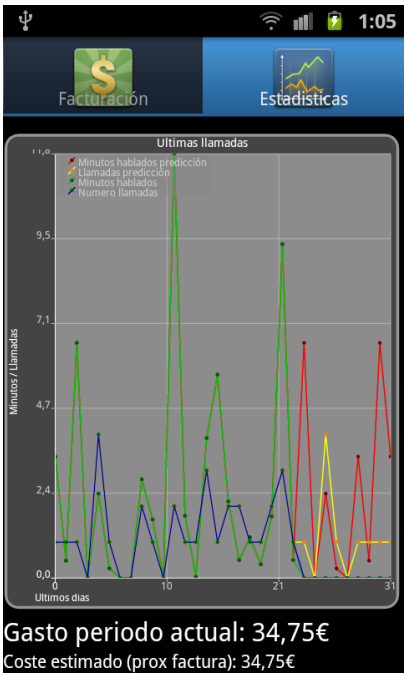


Figura 46. Pantalla de estadísticas.

Si pulsamos sobre la pestaña de facturación tenemos un desglose de lo que sería la facturación actual, llamadas y SMS entre el último día de facturación y hoy. En gasto telefónico va incluido el gasto base de la tarifa correspondiente, incluyendo así los mensajes gratis que pueda tener la tarifa. Por esto es posible que el usuario vea que ha enviado un cierto número de mensajes mayor que 0 sin coste alguno. El desglose obtenido será similar al presentado en la Figura 46.

Figura 47. Resumen de la factura.

Finalmente vamos a ver las distintas opciones de configuración que nos ofrece CGTel, a dichas opciones entrando en preferencias, como ya hicimos en los primeros pasos por CGTel, como podemos apreciar desde aquí podremos cambiar nuestra tarifa actual si hemos cambiado la que elegimos en el primer inicio de la aplicación.

Desde el menú de configuración además de las opciones ya comentadas anteriormente podemos desactivar la pantalla del login en cada inicio de la aplicación, añadir un descuento porcentual y elegir nuestros números VIP. Los números VIP serán aquellos números que la aplicación obviará tanto su coste como duración y sus llamadas no serán tenidas en cuentas a la hora de las estimaciones ni de los desgloses en la facturación. En la pantalla de números VIP podremos agregar distintos números de teléfono (ilimitados) y borrarlos cuando lo veamos necesario. El panel para la administración de números VIP es similar al que podemos ver en la Figura 48.

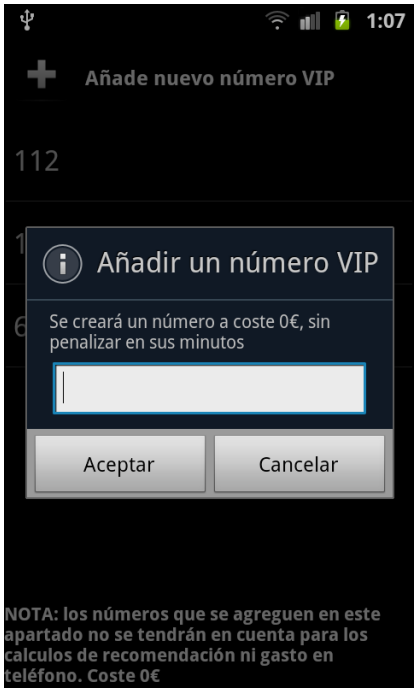


Figura 48. Diálogo para añadir números vip.

Del resto de opciones de configuración:

Mostrar login al inicio: CGTel nos da la posibilidad de guardar los datos del usuario y no tener que pedirlos cada vez que iniciemos la aplicación. La verificación con la base de datos se seguirá haciendo y por lo tanto seguirá siendo necesaria la conexión con la red móvil.

Día de facturación: Aquí podemos seleccionar cual es nuestro día de facturación, imprescindible para el correcto funcionamiento de la aplicación, en el caso de no seleccionar ninguno se seleccionará por defecto el 1 de cada mes.

Descuento porcentual: Aquí el usuario podrá introducir un descuento en el caso de que la compañía telefónica le haga algún descuento concreto sobre la tarifa seleccionada. Inicialmente no hay ningún descuento aplicado.

Para administrar las opciones de administración sobre los usuarios debemos entrar en el menú de Contactos / Grupos y pulsar sobre el botón Permisos allí iremos a un submenú de administración con las siguientes opciones:

Ver contactos que me permiten ver su información, nos muestra una lista de los contactos sobre los cuales tenemos permisos, si los hubiera.

Ver contactos a los que permito administrar es homólogo al caso anterior, mostrándonos en este caso los contactos que tienen permisos sobre nosotros.

Borrar los permisos de administración lo que hace es borrar todos los permisos de administración.

Notificaciones de llamadas con CGTel

CGTel gestiona nuestras llamadas en tiempo real, esto significa que al finalizar cualquier llamada del usuario nos mostrará una notificación (Figura 49) con el teléfono al que se ha realizado la llamada y su duración y coste.

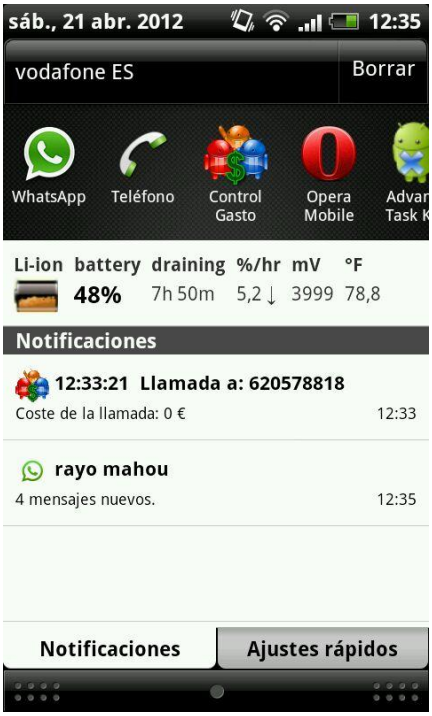


Figura 49. Notificación de llamada de la aplicación CGTel.

Para eliminar la notificación solamente deberemos hacer clic sobre ella.

Opiniones de los usuarios

Tras el lanzamiento de las últimas versiones de la aplicación, se ha realizado un estudio de la opinión de los usuarios en los aspectos generales de la aplicación. Esto ha permitido una mejora de aspectos clave en las últimas fases del desarrollo.

Destacamos la mejora en la experiencia de uso, tras la valoración de las opiniones y la remodelación de la interfaz y el acceso a principales características. En este último ámbito se han mejorado muchos aspectos entre los que destacamos la muestra de información sobre la factura, el acceso a los elementos y la presentación de los contactos y grupos. Así mismo se han incluido notificaciones para mejorar la interacción del usuario con la aplicación, aspecto que fue bastante destacado por los usuarios ya que la aplicación accede en numerosas ocasiones a la base de datos provocando la espera por parte del usuario

Encuesta

A continuación destacamos algunos resultados de la encuesta realizada sobre la última versión de la aplicación (versión 164). La encuesta está disponible para realizarla en la dirección:

https://spreadsheets.google.com/spreadsheet/viewform?formkey=dEZQUkEtN0ppUmJabklKY3VvT0ZxdWc6MQ

Interfaz y experiencia de uso

Se puede apreciar la satisfacción de los usuarios tras las mejoras introducidas en la interfaz en las últimas versiones.

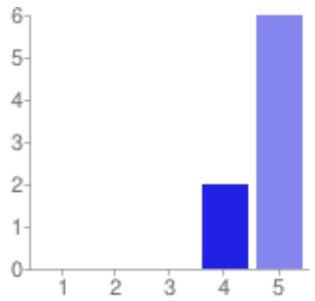


Figura 50. Calificación de la interfaz (1 - Mala, 5 - Buena).

Variedad de tarifas

Las tarifas de nuestros usuarios de prueba están todas incluidas en nuestra base de datos. En principio la base de datos incluye las tarifas suficientes para realizar pruebas pero no hay mucha variedad. Esto no ha supuesto un problema durante el desarrollo del proyecto puesto que la inclusión de nuevas tareas era una labor sencilla.

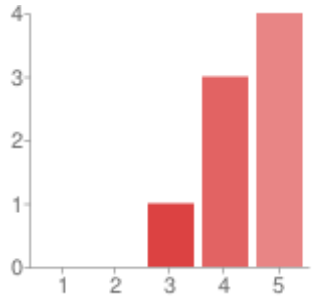


Figura 51. Variedad de tarifas (1 - pocas, 5 - muchas).

Módulos

Los apartados de la aplicación que los usuarios han encontrado más útiles son el recomendador de tarifas y el estimador de factura (valoración en la Figura 53 y en la Figura 54 respectivamente). Los detalles del consumo y las gráficas también son valorados. En cambio la administración de grupos no ha sido utilizada por la mayoría de usuarios los cuales se han mostrado más interesados por el uso del estimador del gasto.

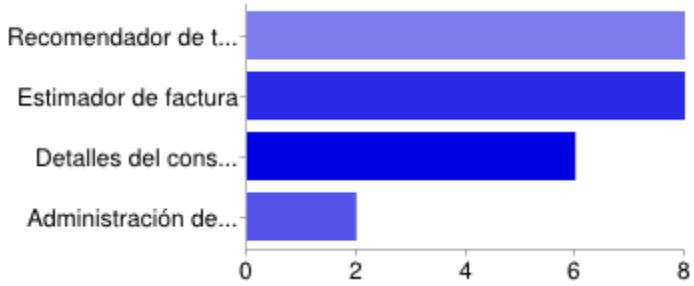


Figura 52. Elementos más utilizados.

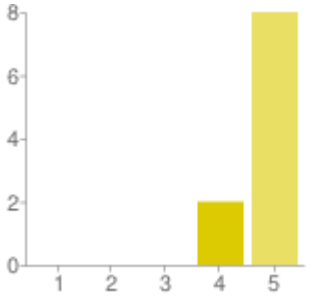


Figura 53. Valoración del recomendador de tarifas.

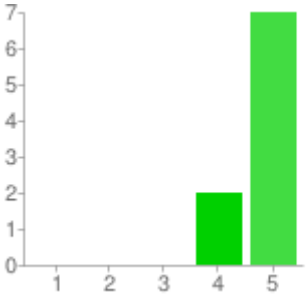


Figura 54. Valoración del estimador de factura.

7. Presente y futuro de CGTel. Conclusiones

Página web

La página web es una parte muy importante de la aplicación ya que actualmente es el nexo de unión con el usuario, desde la página podremos descargarnos el .apk[45] para poder comenzar a utilizar la aplicación.

CGTel comparte la página web con otros proyectos del grupo tecnológico de la UCM, son para públicos muy específicos pero no por ello ni su utilidad ni su interés ha de ser menor.

Los usuarios podrán acceder a la página web y de un modo sencillo descargarse la aplicación. Para ello tendrán que pulsar sobre el logo de descarga que hay a la izquierda del título CGTel.

http://www.ucm.es/info/tecnomovil/proyectos.html

Versiones de la Aplicación

Las versiones estables de CGTel son las siguientes

ControlGasto 0.96.apk
Primera versión de CGTel, prototipo con funciones básicas.

ControlGasto 0.102.apk
Primera versión con el codificado de la página web, agregado también el descuento de Movistar. Es la primera versión completa de CGTel.

ControlGasto 0.141.apk
Corregidos bugs de las versiones anteriores. Aun así es una versión inestable

ControlGasto_v0.143.apk
Versión estable de la 0.141.

ControlGasto_v0.160.apk
Reorganización del menú, modificado el módulo estadísticas, ahora muestra primero la facturación. Incluidos números VIP.

ControlGasto_v0.162.apk
Agregadas ventanas emergentes de aviso, reorganización del menú adquiriendo la estructura actual. No estable.

ControlGasto_v0.164.apk
Versión estable de la 0.162 y última versión de la aplicación hasta el momento.

Colaboraciones y concursos

Cátedra UAM

Para presentarnos a la cátedra UAM enviamos el siguiente documento.

CGTel

Control del gasto telefónico grupal.

Los integrantes que formarán el equipos que desarrollará el proyecto son los alumnos:
Manuel Báez Sánchez, Jorge Cordero Sánchez y Miguel González Pérez.
Dirección:
Dra. Victoria López

Aplicación Móvil para el Control del Gasto Telefónico Grupal

Introducción

Hoy en día los Smartphones están pensados para todo tipo de público, desde los anteriormente señalados hasta los adictos a las redes sociales. Para el uso intensivo de todas las funcionalidades que se nos presentan requerimos de una conexión constante de datos. Las operadores ofrecen una amplia gama de combinaciones de tarifas planas de voz y datos, pero puede que la tarifa de voz contratada sea excesiva. Es aquí donde aparece CGTel.

Descripción

CGTel será una herramienta de telefonía móvil y será desarrollada bajo el sistema operativo Android pensada para uso masivo.

El objetivo de la aplicación será reducir el gasto en nuestra factura telefónica, así como conseguir el uso apropiado del teléfono haciendo uso de la conexión a Internet, que en la mayoría de casos, siempre está presente en este tipo de terminales. Con esta intencionalidad, nuestro foco es llegar al mayor número de público posible, desde una persona que quiera ahorrar en su propia factura, a gente que frecuentemente suele contactar con el mismo grupos de amigos/trabajo o simplemente gente que quiera controlar todas las facturas de su grupo (como pudiera ser el caso de una entidad familiar o de una empresa con una flota de terminales móviles).

Características principales de la aplicación:

1. Cálculo del coste mensual acumulado del grupo**: Se podrá consultar desde cualquier Terminal o desde un Terminal Padre con privilegios especiales.**
2. Cálculo del coste mensual acumulado de cada dispositivo individual**: como en el caso anterior se podrá consultar desde cualquier Terminal o desde un Terminal Padre con privilegios especiales.**
3. Recomendador **por optimización del dispositivo desde el que realizar la siguiente llamada (control inteligente) a partir de perfil de costumbres y comportamiento del consumidor.**
4. Recomendador **de actualizaciones y contratación de ofertas en el mercado así como cambio de proveedor de uno o varios terminales de acuerdo con el perfil del grupo.**
5. Avisadores **en llamada activa de la finalización de saldo con tarifa prioritaria y otros elementos.**

Optimiza el consumo disfrutando siempre de la mejor tarifa.

Las funcionalidades descritas anteriormente permiten realizar cualquier llamada con la mejor tarifa posible. Por ello la aplicación CGTel es de interés en cualquier grupo de personas/teléfonos móviles. A continuación se realizan dos propuestas de explotación a modo de grupos de interés.

- Grupos de Amigos o Familias: **Cada terminal del grupo o familia se conectará con los restantes por la WiFi doméstica y se transferirán los datos de consumo mensual hasta ese momento. De esta forma, desde un terminal se podrá conocer el consumo acumulado e individual de cada miembro. Además el sistema informará (bajo petición) de cuál es el terminal, de los presentes, desde el cual realizar la siguiente llamada optimizando el coste computado.**
- Empresas y sociedades: **Es habitual que las empresas provean a determinados empleados de un móvil de empresa. Puesto que los empleados ya poseen número móvil personal y probablemente no deseen cambiar su proveedor, es habitual encontrar empleados que cargan con dos o más móviles diariamente. Las empresas suelen negociar con un único proveedor que seguramente ya les informa de los gastos acumulados individual y colectivamente por sus empleados con móvil de empresa. Sin embargo, las necesidades de los colectivos a veces llevan a la contratación múltiple. El constante movimiento de nuevas ofertas en el mercado hace que la aplicación de control de gasto grupal CGTel sea de interés para las empresas interesadas en la optimización de este recurso.**
- Compañías telefónicas: **En la actualidad existen muchas ofertas por parte de una gran variedad de operadoras de telefonía. La aplicación podría orientarse únicamente hacia las tarifas de una compañía permitiendo al usuario obtener información de otras tarifas que le podrían ser más rentables y recibir ofertas de la compañía. De esta forma aplicación podría ser utilizada por el grupo de contención de bajas enviando ofertas a determinados usuarios.**

Beneficios

La aplicación se ofrecerá con carácter gratuito en el Market de Android. El interés y uso que genere la aplicación permitirá a usuarios que requieran más funcionalidad comprar una versión de pago. Esta versión de pago estará enfocada al control de grupo, permitiendo que uno o varios usuarios de un grupo puedan ver informes de los gastos de cada terminal del grupo, obtener las facturas a nivel global y presentarle una recomendación de tarifa a nivel global para todo los integrantes. A modo de control, esta utilidad puede generar un gran interés en una entidad familiar y sobretodo en empresas con flotas de móviles desplegadas.

Con la modalidad *Premium* de la aplicación instalada, todos los datos del resto de terminales se le presentarán al usuario desde el propio terminal. Aquí se nos presenta otra oportunidad de negocio, al cliente le puede interesar una solución a medida que le permita acceder a todos esos datos, generar gráficas e informes y así poder tener un total control sobre sus empleados mediante el navegador de internet. De esta forma el usuario administrador, no debe ser necesariamente un terminal más. Dado que todos los datos van a estar alojados en la red, la sencillez para crear un sitio web que le muestre las estadísticas de ciertos terminales a un usuario autorizado se convierte en un desarrollo estándar para cada empresa que lo desee, sin la necesidad de tener que gastar tiempo con desarrollos muy a medida que sean distintos para cada cliente.

Para cubrir los gastos de mantenimiento, se le añadirá publicidad a la aplicación en su versión gratuita, si bien esta opción no pensamos que sea una opción para que sea rentable, será suficiente para mantener el servidor y la

base de datos donde se hospeda la información que maneja la aplicación. En función de los usuarios necesitaremos más o menos espacio, el cual rentaremos gracias a esa publicidad que nos ofrecerán ese número de usuarios.

Algunas Características técnicas

Desarrollo

Los integrantes del equipo de desarrollo serán los que se encargarán tanto del desarrollo de la aplicación, así como de la creación y mantenimiento de la base de datos.

Será necesario disponer de varios terminales Android en los que probar la aplicación, así como una base de datos accesible desde *webservices* provista de un *phpMyAdmin*. El resto de las herramientas restantes necesarias para el desarrollo del proyecto son de carácter gratuito (como el IDE *Eclipse*, o uso de ciertas APIs existentes). Los gastos en materiales se reducirán simplemente al servidor con la base de datos. Estamos pues, hablando de unos costes muy reducidos en cuanto a recursos hardware.

Mantenimiento

Una vez la aplicación se encuentre en producción, además de atender a las incidencias sobre el servicio y arreglos de la propia aplicación, se deberá seguir actualizando la base de datos con un equipo que se encargue de ello. Según el éxito comercial de aplicación este equipo tendrá un volumen variable.

Conclusión

A modo de conclusión, pensamos que una aplicación como CGTel debería estar presente en la vida de los Smartphones. La capacidad de tener internet en el móvil hace que cada vez sea menos necesario el uso de la llamadas. Por todo ello pensamos que la mejor forma de evitar equivocarnos de tarifa por una compañía que nos invade con grandes cantidades de minutos gratuitos o tarifas aparentemente muy atractivas, pero sin ningún interés para el usuario final, sea gracias al uso un estudio fiable y personalizado.

Con todo ello y con la involucración de la empresa en el uso de los datos de la llamadas de sus empleados, CGTel se convertiría en la mejor y más fiable alternativa para el ahorro y el control del gasto de los terminales con contrato de nuestro país.

Contacto

Victoria López
vlopez@fdi.ucm.es
www.mat.ucm.es/~vlopez
Tel. +34 629975771
Facultad de Informática
Universidad Complutense de Madrid
c/ Profesor García Santesmases, s/n
28040 Madrid

SEIO

En el SEIO[46] (Sociedad de Estadística e Investigación Operativaestigación Operativativa) participamos como "CGTel: Control de Gasto Telefónico" y nuestro correo electrónico fue recoletosmb@gmail.com.

El resumen presentado en el SEIO fue el siguiente:

Proyecto CGTel: El principal objetivo de este proyecto es calcular y predecir el gasto acumulativo realizado por un terminal, así como el gasto total de un grupo de móviles asociados en un grupo de interés. CGTel funcionará incluso cuando los móviles tengan diferentes proveedores al ser independiente de las compañías suministradoras de voz y datos. Entre sus funcionalidades destacan recomendadores de tarifas, el cálculo de costes y análisis predictivo del coste al finalizar un periodo.

Wayra

Para presentarnos al proyecto Wayra[47] (Madrid) tuvimos que rellenar una serie de formulario a través de su web donde expusiéramos el caso de negocio del proyecto ante el que nos encontramos. No han salido los resultados todavía.

Trabajos futuros

Ahora bien, es cierto que hay ciertas funcionalidades que podrían hacer de CGTel una aplicación de uso habitual en cualquier terminal. Explicamos estas funcionalidades a continuación.

Añadir algún método de verificación a la hora de agregar usuarios a la base de datos, un método similar al que utiliza WhatsApp valdría. Consiste en un mensaje de confirmación.

Implementar en el propio programa de CGTel las notificaciones entre usuarios utilizando la tecnología C2DM y las notificaciones push en lugar de utilizar un programa externo.

Dar la posibilidad al usuario de modificar su tarifa, para ello tendríamos que incluir algún algoritmo para corroborar que la tarifa no es absurda e incluir dicha tarifa en la base de datos y si fuera verificada que otros usuarios también pudieran seleccionarla.

Incluir Roaming y tarificación en el extranjero, podríamos considerar incluir las principales compañías de los países europeos más importantes o los más cercanos, así CGTel entraría también en el plano internacional y no solo funcionaría en España, para ello sería necesario un despliegue importante en cuanto a las actualizaciones de la base de datos.

Agregar varios idiomas a la aplicación para hacerla multilenguaje. Obviamente sería necesario si se hiciera el punto anterior. Sin duda sería la mejora más vistosa de cara al usuario y más sencilla de aplicar.

Introducir un método de autoaprendizaje para mejorar la estimación del coste a calcular a final de mes, pudiendo incluir por lo tanto varios meses en los

cálculos. Podemos incluso hacer al usuario partícipe de dicho método ofreciéndole la opción de puntuar una estimación y que el algoritmo le de mayor o menor relevancia al resultado en función de dicha puntuación.

Profesionalizar CGTel. Comenzando por mejorar la página web contratando un servicio externo. Así como intentar anunciar en algún sitio el producto. En este punto también tendríamos que migrar la base de datos a un servidor propio.

Una vez hayamos profesionalizado el producto podremos dar servicio de atención al cliente y así ganar agilidad y respuesta a la hora de agregar nuevas tarifas o realizar posibles modificaciones así como depurar bugs futuros.

Hacer reuniones de I+D para que la mejora de CGTel sea constante y pueda llegar a convertirse en una aplicación puntera en su sector.

Modelos avanzados

A continuación vamos a mostrar unos modelos avanzados de cómo se podría mejorar el predictor de la aplicación.

El objetivo es predecir el coste total al final del mes $\sum_{t=1}^{n} y_t = C$ siendo y_t el gasto telefónico t correspondiente a una cierta franja de un cierto día.

Por ejemplo, si hay 30 días y tres franja cada día $\Rightarrow n = 30 * 3 = 90$

$\hat{C} =$ predicción de C utilizando de $Y_1, \dots, Y_k$

$$ECM_k = E\left[\left(C - \hat{C}_k\right)^2\right]$$

mínimo ECM se alcanza en la media condicionada

$$\hat{C}_k = E\left[C/_{Y_1, \dots, Y_k}\right] = E\left[\sum_{r=1}^{m} Y_t/_{Y_1, \dots, Y_k}\right] = \sum_{t=1}^{k} Y_t + \sum_{t=1}^{k} E\left[Y_t/_{Y_1, \dots, Y_k}\right]$$

$E\left[Y_t/_{Y_1, \dots, Y_k}\right]$ depende del modelado que asumamos para Y_t

Modelo ARIMA

Consideramos solamente modelos estacionarios, porque no es de esperar la existencia de tendencia y autorregresividad, por simplicidad.

$$\phi(B) = \phi(B^s)x_t = a_t$$

Siendo S es el período de estacionacionalidad. Se identifica el siguiente modelo:

$$(1 - \phi_1 B - \phi_2 B^2)(1 - \phi B^S)X_t = a_t$$

es decir, Y_t depende de los dos datos anteriores y del mismo datos S períodos antes.

Aunque se puede identificar y estimar un modelo cada mes y utilizarlo en el mes siguiente para predecir, el método parece engorroso y algorítmicamente complicado por lo que, alternativamente se puede optar por el siguiente modelo:

Modelo de suavizado o alisado exponencial

El modelo más adecuado es el de suavizado exponencial estacional simple, es decir, sin tendencia. Se supone que $x_t = n_t + s_t + e_t$

El nivel de la serie se obtiene por: $n_t = \alpha\,(x_t - s_{t-s}) + (1-\alpha)a_{t-1}$, $\alpha \in (0,1)$ y la actualización de la componente estacional por $s_t = \beta(x_t - a_t) + (1-\beta)s_{t-s}$, $\beta \in (0,1)$ un valor típico para $\alpha\, y\, \beta$ sería 0.1, aunque se podrían ajustar de forma preciso. La predicción de y_t sería $\hat{y} = n_k + s_{t-rs}$ siendo r tal que $t - rs < k$.

Conclusiones

CGTel es un aplicación muy útil y sobre todo muy sencilla, con el simple hecho de elegir nuestra tarifa y seleccionar un día de facturación podemos contar con una amplia diversidad de opciones, desde simplemente saber cuál es la tarifa que más se ajusta a tu perfil de usuario, como saber a qué usuario de CGTel va a costarle menos hacer la siguiente llamada.

Por ello y por lo expuesto en el punto anterior CGTel es una aplicación con presente y un gran futuro por delante y consideramos que sería interesante que se continuara con el proyecto en un futuro.

8. Bibliografía y referencias

Bibliografía

Ed. Burnette, Hello, Android: Introducing Google's Mobile Development Platform (2ª Edición) 2009

Narciso Martí, Yolanda Ortega, J. Alberto Verdejo, Estructura de datos y métodos algorítmicos, Prentice Hall, 2003.

Guía para desarrolladores, Android Developers

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

Android Cloud to Device Messaging Framework, Google Developers https://developers.google.com/android/c2dm/

StackOverflow, comunidad de desarrolladores internacional multilenguaje http://stackoverflow.com/

Android-Spa, principal comunidad de desarrolladores de Android en español

http://www.android-spa.com/

Android Plot, documentación web official

http://androidplot.com/

Descargas de distintos paquetes de Eclipse. Web oficial de la plataforma Eclipse. http://www.eclipse.org/

Referencias

[1] "Open Handset Alliance (OHA)" <http://www.openhandsetalliance.com/>

[2] "comScore Data Mine" <http://www.comscoredatamine.com/wp-content/uploads/2012/04/Smartphone-Platform-Share_February-2012-Data.jpg>

[3] "Inside Mobile Apps" <http://www.insidemobileapps.com/wp-content/uploads/Screen-shot-2012-06-04-at-11.37.40-AM.png>

[4] "Google Play." 2007. 22 May. 2012 <http://play.google.com/>

[5] "Top Android Apps and Games in the Android Market | AppBrain.com." 2005. 22 May. 2012 <http://www.appbrain.com/>

[6] "Amazon.com: Appstore for Android." 2011. 22 May. 2012 <http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011>

[7] "GetJar | Móvil |." 2009. 22 May. 2012 <http://m.getjar.com/?lang=es>

[8] "AppsFire" 2009 22 May. 2012 <http://appsfire.com/>

[9] "SlideMe | Android Apps Market: Download Free & Paid Android ..." 2008. 22 May. 2012 <http://slideme.org/>

[10] "AppBrain | Android Market stats" <http://www.appbrain.com/stats/>

[11] "Demandan a Google por su política de devoluciones en Google ..." 2012. 24 May. 2012 <http://www.muycomputerpro.com/2012/03/19/demandan-a-google-por-su-politica-de-devoluciones-en-google-play-store/>

[12] "Amazon.com Test Drive: Try apps right now on your computer." 2011. 24 May. 2012 <http://www.amazon.com/gp/feature.html?ie=UTF8&docId=1000667581>

[13] "AppBrain | Android Market stats" <http://www.appbrain.com/stats/>

[14] "Onavo | Data Monitor for Android." 2012. 24 May. 2012 <http://www.onavo.com/data_monitor>

[15]"Pocket-lint | APP OF THE DAY: Onavo review (Android)" 2012. 5 Mar. 2012< http://www.pocket-lint.com/news/44763/onavo-android-app-review-data>

[16] "My Data Manager - Android Apps on Google Play." 2012. 24 May. 2012 <https://play.google.com/store/apps/details?id=com.mobidia.android.mdm>

[17] "myPlan - Android Apps on Google Play." 2012. 25 May. 2012 <https://play.google.com/store/apps/details?id=com.conzebit.myplan>

[18] "Code Fraquency. jmsanzg/myPlan - GitHub." 25 May. 2012 <https://github.com/jmsanzg/myPlan/graphs/code-frequency>

[19] "Commit Activity. jmsanzg/myPlan - GitHub." 25 May. 2012 <https://github.com/jmsanzg/myPlan/graphs/commit-activity>

[20] "jmsanzg/myPlan - GitHub." 25 May. 2012 <https://github.com/jmsanzg/myPlan/issues?state=closed>

[21] "jmsanzg/myPlan - GitHub." 25 May. 2012 <https://github.com/jmsanzg/myPlan/issues?page=1&state=open>

[22] "Gastos Móvil." 2010. 25 May. 2012 <http://gastosmovil.simahuelva.es/>

[23] "Gastos Móvil, toma el control de tu factura. 4ndroid.com"<http://4ndroid.com/gastos-movil-toma-el-control-de-tu-factura/>

[24] "Android SDK | Android Developers." 2009. 25 May. 2012 <http://developer.android.com/sdk/index.html>

[25] "Android NDK | Android Developers." 2010. 25 May. 2012 <http://developer.android.com/sdk/ndk/index.html>

[26] "Grupo de Tecnologías Móviles - Universidad Complutense de Madrid." 2012. 25 May. 2012 <http://www.ucm.es/info/tecnomovil/proyectos.html>

[27] "Subversion, Git and Mercurial Hosting - Powered by XP-Dev.com." 2007. 25 May. 2012 <http://xp-dev.com/>

[28] "MySQL :: The world's most popular open source database." 6 Jun. 2012 <http://www.mysql.com/>

[29] "Función Hash - Wikipedia, la enciclopedia libre." 2008. 6 Jun. 2012 <http://es.wikipedia.org/wiki/Funci%C3%B3n_Hash>

[30] "The MD5 Message-Digest Algorithm - MIT Laboratory for Computer Science and RSA Data Security, Inc." 1992, Apr <http://www.ietf.org/rfc/rfc1321.txt>

[31] "US Secure Hash Algorithm 1 (SHA1) – Cisco Systems" 2001 Sept <http://www.ietf.org/rfc/rfc3174.txt>

[32] "Ronald L. Rivest : HomePage." 2007. 6 Jun. 2012 <http://people.csail.mit.edu/rivest/>

[33] "phpMyAdmin" 2002. 5 Jun. 2012 <http://www.phpmyadmin.net/>

[34] "Android Cloud to Device Messaging Framework – Android Developers" <https://developers.google.com/android/c2dm/>

[35] "Web Services Architecture - W3C" February 2004 http://www.w3.org/TR/ws-arch/

[36] "Web Services Description Language (WSDL) Version 2.0 - W3C" 26 June 2007 <http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/>

[37] "Latest SOAP versions – W3C"http://www.w3.org/TR/soap/

[38] "Especificación JSON " http://www.json.org/json-es.html

[39] "Mastering C2DM The Android Cloud to Device Messaging Framework by Aleksandar (Saša) Gargenta - Marakana Inc." 2011 <https://docs.google.com/present/view?id=dvp65dh_32cgqkzcdm>

[40] "WhatsApp Messenger – Homepage" http://www.whatsapp.com/

[41] "Cierra Google compra con Motorola." 25 May. 2012 <http://www.elporvenir.com.mx/notas.asp?nota_id=586643>

[42] "Android (operating system) - Wikipedia, the free encyclopedia." 2008. 25 May. 2012 <http://en.wikipedia.org/wiki/Android_(operating_system)>

[43] "El patrón Modelo-Vista-Controlador (MVC) – Programación Orientada a Objetos FDI, UCM" 2008 <http://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf>

[44] "Adapter | Android Developers." 2009. 5 Jun. 2012 <http://developer.android.com/reference/android/widget/Adapter.html>

[45] "The Developer's Guide - Glossary| Android Developers." http://developer.android.com/guide/appendix/glossary.html

[46] "SEIO – Homepage" http://www.seio.es/

[47] "Wayra – Homepage" http://wayra.org/es/

9. Apéndices

Anexo I: Desarrollo en Android

Eclipse

Se trata de un entorno de desarrollo integrado (en adelante IDE) de código abierto multiplataforma para desarrollar aplicaciones pesadas. Esta plataforma, típicamente ha sido usada para desarrollar IDEs, el más conocido es el de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

Eclipse fue desarrollado originalmente por IBM en 2001 y soportado por un consorcio de empresas de software. Actualmente, Eclipse sigue siendo desarrollado por la Eclipse Foundation, organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License, siendo ambas licencias de software libre.

ADT: Android Developer Tools

El ADT (Android Developer Tools) es un plugin para Eclipse que ofrece un paquete de herramientas para integrarlas con Eclipse. El plugin, proporciona utilidades que hacen la experiencia del programador Android mucho más amena, proporciona una interfaz gráfica para la mayoría de los comandos del SDK, del mismo modo presenta en el propio plugin una interfaz gráfica para prototipar, diseñar y crear la interfaz gráfica de una aplicación.

De entre las funcionalidades de este plugin para Eclipse cabe remarcar:

Integración de Android: Creación de proyectos, desarrollo, empaquetado, instalación y depuración. El ADT integra herramientas para gestionar el flujo de trabajo dentro del Eclipse, permitiendo agilizar el desarrollo y el testing de aplicaciones Android.

Integración con el SDK: Las herramientas del SDK quedan integradas dentro de los menús del Eclipse. Generan nuevas perspectivas de procesos ejecutados en segundo plano por el ADT.

Lenguaje Java y editores XML: El editor de Java comparte las funciones comunes del IDE de Java como el autocompletado, formateo de código, revisor de sintaxis y la integración de la documentación de las APIs de Android. El ADT también integra editores personalizados de XML que permiten crear XML específicos para Android, donde se permite la modificación de dichos archivos mediante una interfaz gráfica permitiendo modificarlos hasta con *drag and drop*.

Documentación de Android integrada: Permite un fácil acceso a la documentación al pasar sobre las distintas clases, métodos y variables.

Actualmente la última versión publicada del ADT es la 18.0.0 (release Abril 2012). Esta última versión necesita correr sobre Java 1.6 o superior y del mismo modo disponer de el Eclipse 3.6.2 o superior. Esta última versión corrige bugs y está pensada fundamentalmente para acompañar al último SDK de Android (r18).

Instalación

Para instalarlo debemos seguir una serie de pasos desde Eclipse:

1. Arrancamos Eclipse y, una vez dentro, debemos dirigirnos a "Help" -> "Install New Software".
2. Añadimos el repositorio donde se encuentra el plugin ADT: http://dl-ssl.google.com/android/eclipse/
3. Seleccionamos "*Developer Tools*" y pulsamos "*Next*"
4. Pulsamos siguiente y aceptamos los términos de la licencia, una vez proceda clicamos en "*Finish*".
5. Cuando la instalación lo solicite, reiniciaremos el Eclipse.
6. Para configurarlo y poder empezar a usarlo, una vez haya reiniciado el Eclipse, seleccionamos "*Window -> Preferences*".
7. En el panel situado en el lateral izquierdo, seleccionamos Android (esta opción nos indicará que el plugin se ha instalado correctamente).
8. Nos falta seleccionar la ubicación de SDK que previamente hemos instalado en nuestro sistema, para ello hacemos click en "Browse" y seleccionamos la raíz de la carpeta donde tengamos instalado nuestro SDK.
9. Presionamos "*OK*" y podremos comenzar a utilizar nuestro plugin ADT.

Subclipse

Subclipse es un plugin para Eclipse que permite de manera sencilla interconectar nuestros proyectos locales con un repositorio de tipo Subversion. Este software se distribuye mediante Eclipse Public License (EPL 1.0), licencia de código libre. Eclipse trae, en sus últimas distribuciones, soporte nativo para repositorios CVS, si necesitamos usar un repositorio SVN, Subclipse es sin duda la mejor solución integrada dentro del propio IDE.

Entre las características del plugin destacan:

Inclusión del CollabNet Merge Client, el cual nos permite utilizar potentes "*merge*" o fusiones de código mediante un intuitivo editor gráfico. Esta función para poder seguir la fusión del código al vuelo esta incluido a partir de la release 1.5 de Subversion.

Conector Mylyn que permite a Mylyn crear automáticamente cambios basados en las tareas sobre las que se esté trabajando. Este conector, habilita también una opción para poder generar enlaces a las tareas mientras se ve el histórico de los *commits* sobre el repositorio Subversion.

Gráfico de revisiones, Subclipse posee una interfaz con gráficos muy completo donde se muestran los cambios en las distintas revisiones del proyecto. Este plugin gráfico esta creado con Eclipse GEF/Draw2D. Este gráfico es muy útil ya que permite al usuario poder recorrer de manera muy intuitiva el repositorio y recorrer entre los distintos *commits* y *merges* a través de las ramas de Subversion. En la captura de la Figura 55 podemos ver el gráfico de revisiones de un archivo del proyecto que nos atañe.

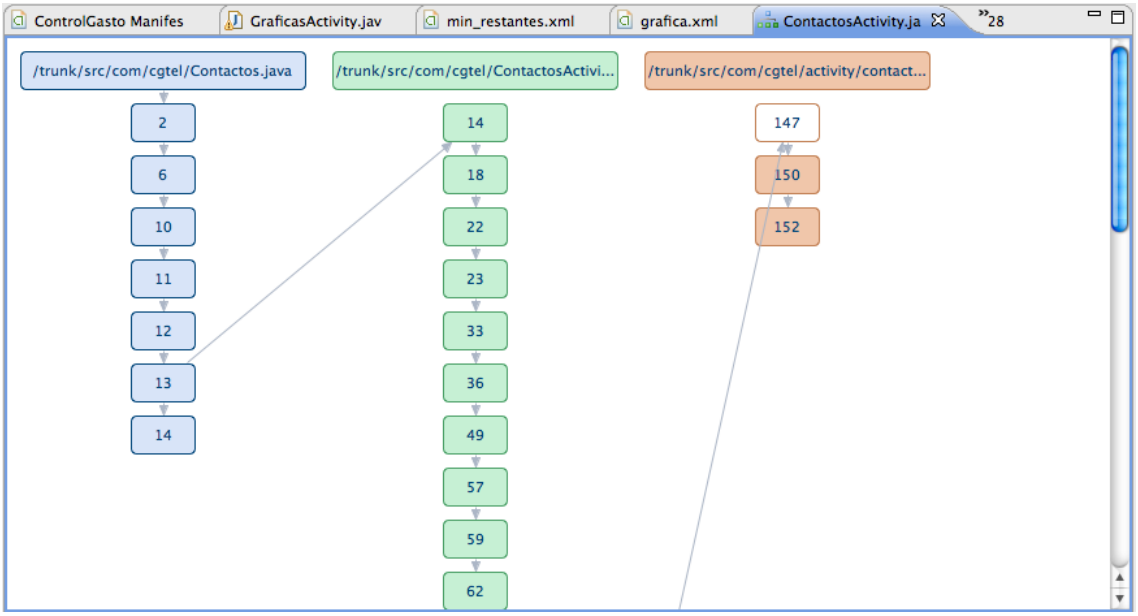


Figura 55. Gráfico de revisiones.

Para más información sobre el uso y posibles dudas ver el resumen de las FAQ de los desarrolladores: http://subclipse.tigris.org/wiki/PluginFAQ.

El SDK de Android

Para poder hacer una aplicación para Android con Eclipse es necesario descargar e instalar el SDK de Android, los pasos para descargarlo, instalarlo y su funcionalidad se explican a continuación.

Funcionalidades, descarga e instalación

Lo primero que tendremos que hacer será descargarlo, podemos descargarlo de la página de Android Developer, una vez que lo descarguemos tendremos que decidir los paquetes y funcionalidades que queremos descargar. Ciertas

funcionalidades son exclusivas de algunas versiones, algunas se pueden quedar obsoletas, en ese caso habrá otras que nos aporten la misma funcionalidad.

Pero a veces podemos tener nuevas funcionalidades a partir de una versión. Por eso es posible que una aplicación para Android solamente sea válida a partir de una versión concreta haciendo que los terminales con versiones anteriores de Android no sean compatibles. Este caso anterior suele pasar sobre todo con terminales relativamente antiguos ya que como bien hemos explicado Android ofrece su versión nativa a las compañías de teléfonos móviles las cuales las adaptan a sus terminales. Pero en el caso de ser un terminal descatalogado es posible que no salga ninguna versión impidiendo el funcionamiento de ciertas aplicaciones en el terminal. Por ello el SDK nos permite tener acceso a la jerarquía de clases de Android para poder realizar las aplicaciones.

Pero siguiendo con el SDK, como bien hemos dicho antes del inciso, podemos decidir qué versiones descargarnos de Android así como múltiples funcionalidades, algunas para terminales concretos y otras para cualquier terminal. La versión de Android más utilizada es la 2.3 por lo que es interesante por lo menos descargarse dicho paquete.

Es importante destacar que la descarga es excesivamente lenta e incluso algunos miembros del equipo de este proyecto han tenido ciertos problemas al realizarla.

Una vez se finalicen las descargas añadiremos dicho software al plugin de Eclipse. (Ver el apartado ADT)

Documentación

El SDK de Android dispone de una excelente guía online para desarrolladores[48]. En esta guía se pueden encontrar desde esquemas de implementación hasta consejos de diseño introducido tras la publicación de Ice Cream Sandwich[49].

NDK

El Android NDK es un conjunto de herramientas que permite al desarrollador crear componentes en código nativo para las aplicaciones Android.

El NDK pone a disposición del usuario implementar partes de código en lenguajes nativos, comúnmente C y C++ evitando durante estas operaciones la interacción con la máquina virtual de Java. Esta metodología permite incrementar la velocidad de ejecución, por el contrario esta implementación supone una mayor complejidad para el desarrollador.

El NDK al igual que el SDK está provisto de documentación, ejemplos y tutoriales. Actualmente la última release del NDK (r8.0) soporta los siguientes juegos de instrucciones: ARMv5TE, ARMv7-A y MIPS

El NDK proporciona cabeceras estables para libc (la librería de C) libm (la librería de operaciones matemáticas), OpenGL ES (librería gráfica) y otras librerías que vienen incluidas en el set de herramientas que proporciona el NDK.

El NDK no beneficia a la mayoría de las aplicaciones. Es importante que el desarrollador ponga en una balanza a la hora de decidirse entre usar el NDK o no, los pros y los contras que va a encontrar en el desarrollo, ya que la complejidad se incrementa notablemente. Su uso está pensado sólo para mejorar determinadas partes de las aplicaciones (por ejemplo, en aplicaciones de tiempo real) , pero se utiliza como estándar para poder programar en C/C++.

Las aplicaciones candidatas para el uso del NDK, son aplicaciones de alto uso de la CPU que no requieren demasiado uso de la memoria, como pueden ser cálculos físicos, tratamiento de la señal, y todo tipo de operaciones multimedia..

Las herramientas de desarrollo contienen una serie de compiladores, linkers, etc, que permiten generar los ejecutables binarios para ARM sobre las plataformas de Linux, Mac y Windows.

Las APIs Nativas cuyo soporte está garantizado en el NDK son:

- libc (librería C)
- libm (librería para cálculos matemáticos)
- JNI (Framework que permite a un código escrito en Java para ser ejecutado en la_máquina virtual pueda interactuar con programas escritos en otros lenguajes como_C, C++ y ensamblador)
- libz (librería de compresión)
- liblog (generador de logs en Android)
- OpenGL ES 1.1 y OpenGL ES 2.0 (librerías de gráficos en 3D)
- libjnigraphics (Acceso a buffer de pixeles)
- Soporte básico para C++
- OpenSL ES (librería de audio nativa)
- APIs de Aplicaciones nativas en para Android

Emulador

Android nos permite programar para ellos sin la necesidad de tener un terminal con su sistema operativo instalado. Para ello tendremos que hacer uso del emulador de Android, en este caso para Eclipse.

Usarlo es muy sencillo simplemente tendremos que probarlo pulsando *Run* (desde Eclipse), como si fuéramos a ejecutar cualquier proyecto. En el caso de que no haya ningún terminal conectado se ejecutará automáticamente el emulador arrancando así una versión de Android compatible con nuestra aplicación. Dentro de dicha versión de Android estará nuestra aplicación para poder ver su correcto funcionamiento.

Dicho emulador, pese a no ser un terminal, no tener tarjeta y por lo tanto no poder realizar llamadas las puede simular, lo cual nos ha resultado bastante útil

dada la naturaleza de nuestra aplicación. En la Figura 56, tenemos un ejemplo del emulador ejecutando CGTel.

Figura 56. Aplicación CGtel ejecutada en el emulador.

Para poder administrar el tipo de terminal que queremos emular tenemos el AVD Manager, el cual viene incorporado con el SDK. Desde esta aplicación podremos elegir qué tipo de terminal queremos crear. En el menú que se muestra en la Figura 60, podemos ver como entre las opciones que se ponen a disposición del usuario se encuentran la versión de la API que queremos que ejecute el terminal, el tipo de arquitectura de CPU del dispositivo, tamaño de la memoria externa e incluso la resolución.

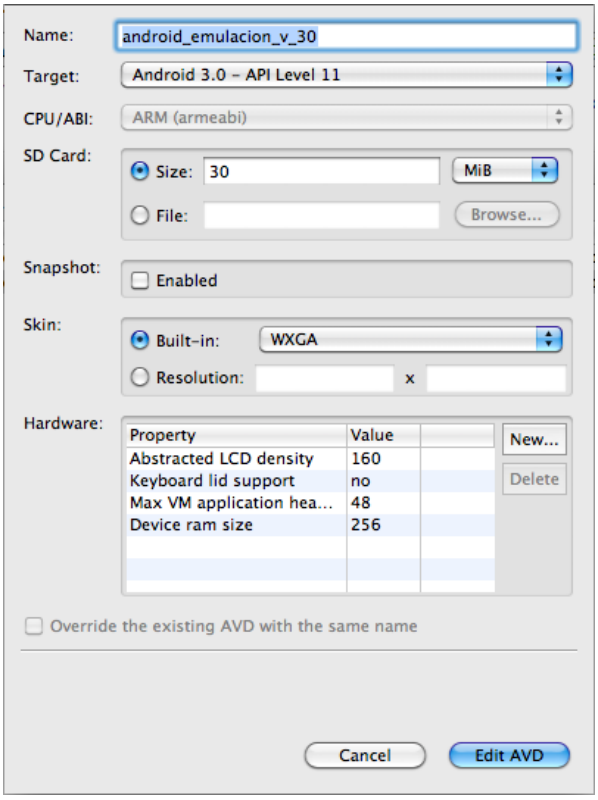


Figura 57. Ventana de configuración de un emulador.

Podremos acceder al AVD Manager más cómodamente desde Eclipse en el menú Window -> AVD Manager.

Programación orientada a objetos y Java

En programación imperativa, los datos se consideran los elementos fundamentales de un programa (las variables). Existen por sí mismos y son usados y modificados por las funciones/procedimientos.

El inicio del paradigma de Programación Orientada a Objetos aparece en la mitad de los años 60 con el lenguaje de programación Simula, creado en Noruega. Simula se definió como un lenguaje de programación orientado a la simulación de procesos, con el que se podían definir distintos tipos de actividades. En este lenguaje aparecen por primera vez los conceptos de *clases* y *objetos*.

Una de las ideas fundamentales del paradigma de programación orientada a objetos es el concepto de *objeto* como una entidad que engloba datos y funciones. El enfoque de la POO permite modelar un dominio (problema a representar) de una forma muy cercana a la realidad. Los objetos del programa simulan los objetos (sustantivos) del dominio. Y los métodos de los objetos permiten modelar perfectamente las acciones que pueden realizar. Para evaluar las funciones hay que *enviar un mensaje* al objeto solicitando que se ejecute alguno de sus métodos. Sólo es posible consultar el estado de un objeto mediante alguno de sus métodos. De esta forma, en POO se refuerza la filosofía de la *barrera de abstracción* y de la *ocultación de información*.

Entre las características de la POO podemos indicar los siguientes conceptos:

- Objetos y clases.
- Los objetos agrupan datos y métodos, realizando de esa forma una ocultación de la información.
- Los métodos de los objetos se invocan mediante mensajes.
- Dispatch dinámico: cuando una operación es invocada sobre un objeto, el propio objeto determina qué código se ejecuta. Dos objetos con la misma interfaz pueden tener implementaciones distintas.
- Herencia: las clases se pueden definir utilizando otras clases como plantillas y modificando sus métodos y/o variables de instancia.

Java

Java, desarrollado en los laboratorios de Sun, es uno de los lenguajes de programación orientado a objetos que mayor repercusión ha tenido en los últimos años.

Está basado en C++ pero simplificado, mucho más fácil de usar, con una mayor abstracción y menos propenso a errores. Java dispone de una amplísima biblioteca estándar de clases predefinidas Las aplicaciones Java pueden ser ejecutadas indistintamente en cualquier plataforma sin necesidad de recompilación

ya que se trata de un lenguaje interpretado haciendo uso de la Java Virtual Machine (JVM).

Sus posibilidades son muy amplias, dado que hace uso de una máquina virtual para ejecutar el código desarrollado en Java, estas aplicaciones son fácilmente portables entre equipos, dispositivos, móviles o contra la propia web. De cara al aprendizaje del lenguaje y para una mayor comodidad del programador, Java controla la gestión avanzada de memoria mediante el uso de un recolector de basura, que eliminará aquellos objetos que queden huérfanos, es decir, que no tengan una referencia en ningún objeto. Gestión avanzada de errores, tanto en tiempo de compilación como de ejecución Soporte sencillo de múltiples hilos de ejecución

Java es uno de los lenguaje más intuitivos de cara a aprender el uso de la POO. La ayuda que ofrece el recolector automático de basura y su amplia extensión le convirtieron en el claro candidato para ofrecer el desarrollo para Android.

Java ofrece una amplia extensión de Kits de desarrollo y documentación gratuitos en la red. Existen infinidad de .jar (librerías o programas escritos en Java) que ofrecen al programador muchas posibilidades y que en este caso son utilizables desde Android. Los únicos .jars que presentan problemas a la hora de importarlos desde el código para Android son los que tienen que ver con los paquetes de interfaces gráficas.

Futuro de Android

Actualmente el sistema operativo Android no tiene problemas de distribución ya que son muchos los fabricantes que actualmente están optando por incorporar Android como sistema operativo de forma nativa tanto en Tablets como en terminales móviles.

De hecho, durante estos últimos años Google de la mano de las principales compañías ha ido sacando nuevos terminales creados con la firme idea de ser la referencia en potencia e innovación del sistema. La aparición de estos terminales siempre ha estado ligada a la salida de un nueva versión del sistema operativo Android, dando la exclusividad (momentánea) a la marca colaboradora de Google.

Así es como han surgido los llamados Google Phones con la certificación "*Powered by Google*", de entre los más destacados están la HTC Magic, HTC Nexus One y Samsung Galaxy Nexus.

Hardware

Con la adquisición de Motorola Mobility, Google tiene asegurado su futuro en los terminales móviles, al menos en lo que al hardware se refiere, sin depender de ninguna empresa ajena como sucedía antes con Samsung o HTC. Pero en el desarrollo del software están surgiendo ciertos problemas, como bien exponemos a continuación.

Software

Actualmente Oracle está presionando a Google para que pague la certificación de Java en el sistema Android, por eso se está planteando seriamente la migración del sistema al lenguaje C#[50].

Java y C# son parte de la gran familia de lenguajes de programación interpretados, esto es, que el código es ejecutado en un intérprete o 'máquina virtual'. La principal ventaja del uso de la máquina virtual es que permite que el lenguaje sea independiente del sistema operativo o el hardware, ya que encargarse de esta tarea es trabajo del intérprete. Por lo tanto, esto abre un gran abanico de posibilidades al programador ya que, aunque haga un programa para una plataforma, será compatible en todos aquellos en los que la máquina virtual esté disponible. Por eso es más que recomendable que este suceda en Android ya que sus destinatarios son terminales móviles con características muy diferentes.

Android es básicamente un sistema Linux sobre el que funciona el intérprete 'Dalvik' de Java que ejecuta todo el sistema operativo. Este intérprete pese a estar altamente modificado para funcionar mejor en terminales móviles que la implementación original de Java, sigue siendo perseguido por Oracle para poder reclamar el importe su certificación. Este es uno de los puntos que promueve como principal ventaja de uso de C# ya que Mono, su intérprete para Linux, lleva cerca de 10 años siendo desarrollado, por lo que está muy optimizado y su uso no estaría ligado a derechos de autor..

C# es un lenguaje creado por Microsoft, se presentó desde el principio como un estándar, por lo que las implementaciones del intérprete no están sujetas a derechos de autor. Actualmente Xamarin, empresa implicada también en el desarrollo de Mono y herramientas asociadas a C#, ha liberado el sistema operativo XobotOS que mantiene la filosofía de Android pero usando este lenguaje. Así, XobotOS, se ha convertido en una pequeña muestra del posible futuro de la plataforma móvil más extendida.

Repositorio

Uno de los principales fines de un repositorio es centralizar en un punto la última versión del código fuente de una aplicación. Soluciones rudimentarias al problema de la centralización utilizan una unidad compartida en la red, pero ¿qué ocurre si dos personas modifican un mismo archivo al mismo tiempo? ¿y si necesitamos volver a una versión anterior del código? para resolver estas y otras cuestiones aparecen los repositorios.

Para poder llevar a cabo el desarrollo de CGTel hemos estimado indispensable el uso de un repositorio que nos permitiera compatibilizar sin problemas los distintos horarios de los integrantes del grupo, para evitar conflictos en el código y poder tener un histórico de versiones mucho más potente que cualquiera que pudiésemos ir generando de manera manual de cara a tener copias de seguridad estables del proyecto.

Funcionalidades

Utilizando un repositorio de código obtenemos de manera inmediata las siguientes ventajas:

- Solución al problema de la concurrencia. Varias personas podrán trabajar con el mismo fichero sin ningún tipo de problema, el repositorio nos ayudará a evitar errores y a tener un control total sobre posibles conflictos y revisiones del código.

- Trazabilidad completa. El repositorio nos otorga la posibilidad de controlar todas las versiones que se han producido de forma automática para cada una de las modificaciones de un fichero. Nos ofrecerá capacidad para compararlas entre ellas, conocer la fecha de modificación y el autor de dicha revisión.

- Histórico. Poder retornar a una versión anterior de la aplicación sin problema, y a partir de dicha versión abrir una nueva rama que difiera de la que se tomó en su momento. Podremos desde lanzar una actualización de una versión ya cerrada aunque esté a mitad de un desarrollo hasta hacer que para la nueva versión que se esté preparando tenga los mismos errores solventados que la de la actualización.

- Seguridad. Control de acceso al código, sólo aquellos usuarios que tenga determinados permisos podrán realizar modificaciones, borrar, crear nuevos ficheros. Además, la redundancia que suele estar presente en estos sistemas nos otorga la tranquilidad de saber que nuestros ficheros no van a desaparecer.

Tipos

Vamos a explicar brevemente los tres tipos que históricamente se han usado más: CVS, Subversion y Git.

- **CVS:** Concurrent Versions System, es una aplicación informática que implementa un sistema de control de versiones. CVS se ha hecho popular en el mundo del software libre. Sus desarrolladores difunden el sistema bajo la licencia GPL. CVS utiliza una arquitectura cliente-servidor, el servidor guarda las versiones actuales del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. Esto se hace para que eventualmente puedan trabajar con esa copia y más tarde ingresar sus cambios con comandos GNU. Típicamente, cliente y servidor se conectan utilizando Internet, pero con el sistema CVS el cliente y servidor pueden estar en la misma máquina. El sistema CVS tiene la tarea de mantener el registro de la historia de las versiones del programa de un proyecto solamente con desarrolladores locales. Actualmente los clientes CVS pueden funcionar en cualquiera de los sistemas operativos más difundidos.

- **Subversion:** es un sistema de control de versiones diseñado específicamente para remplazar al popular CVS. Este sistema también se distribuye como software libre bajo una licencia de tipo Apache/BSD y es quizá más conocido con las siglas SVN por ser este el nombre de la herramienta utilizada en la línea de comando. Una característica importante de Subversion es

que, a diferencia de CVS, los demás archivos que se encuentren bajo control de versión no tienen cada uno un número de revisión independiente. Para resolver este problema, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras.

- Git: es la última tendencia en software de control de versiones diseñado por Linux Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, a destacar entre los más importantes el desarrollo del kernel de Linux y el desarrollo de Ruby. De sus características destacan el almacenaje de forma distribuida, mientras sus competidores usan un servidor centralizado, una mejora considerable de eficiencia en proyectos extensos y su fuerte adaptación para desarrollos no lineales.

Referencias

[48] "The Developer's Guide | Android Developers." 2009. 25 May. 2012 <http://developer.android.com/guide/index.html>

[49] "Android Design - Android Developers." 2012. 25 May. 2012 <http://developer.android.com/design>

[50] “Xamarin | Android ported to C#” 2012. 1 May. 2012 <http://blog.xamarin.com/2012/05/01/android-in-c-sharp/>

Anexo II: Manual de Android

1. INTRODUCCIÓN

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

¿QUÉ ES ANDROID?

En los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales, grandes y pesados, pensados sólo para hablar por teléfono en cualquier parte, a los últimos modelos, con los que el término "medio de comunicación" se queda bastante pequeño.

Es así como nace Android. Android es un sistema operativo y una plataforma software, basado en Linux para teléfonos móviles. Además, también usan este sistema operativo (aunque no es muy habitual), tablets, netbooks, reproductores de música e incluso PC's. Android permite programar en un entorno de trabajo (framework) de Java, aplicaciones sobre una máquina virtual Dalvik (una variación de la máquina de Java con compilación en tiempo de ejecución). Además, lo que le diferencia de otros sistemas operativos, es que cualquier persona que sepa programar puede crear nuevas aplicaciones, *widgets*[1], o incluso, modificar el propio sistema operativo, dado que Android es de código libre, por lo que sabiendo programar en lenguaje Java, va a ser muy fácil comenzar a programar en esta plataforma.

HISTORIA DE ANDROID

Fue desarrollado por Android Inc., empresa que en 2005 fue comprada por Google, aunque no fue hasta 2008 cuando se popularizó, gracias a la unión al proyecto de Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo hardware, software y telecomunicaciones, que decidieron promocionar el software libre. Pero ha sido Google quien ha publicado la mayor parte del código fuente del sistema operativo, gracias al software Apache, que es una fundación que da soporte a proyectos software de código abierto.

Dado que Android está basado en el núcleo de Linux, tiene acceso a sus recursos, pudiendo gestionarlo, gracias a que se encuentra en una capa por encima del Kernel, accediendo así a recursos como los controladores de pantalla, cámara, memoria flash…

En la Figura 1, abajo, se muestran las capas que conforman el sistema operativo Android:

[1] Un *widget* es una pequeña aplicación que facilita el acceso a funciones frecuentes. Más información en el Capítulo 10

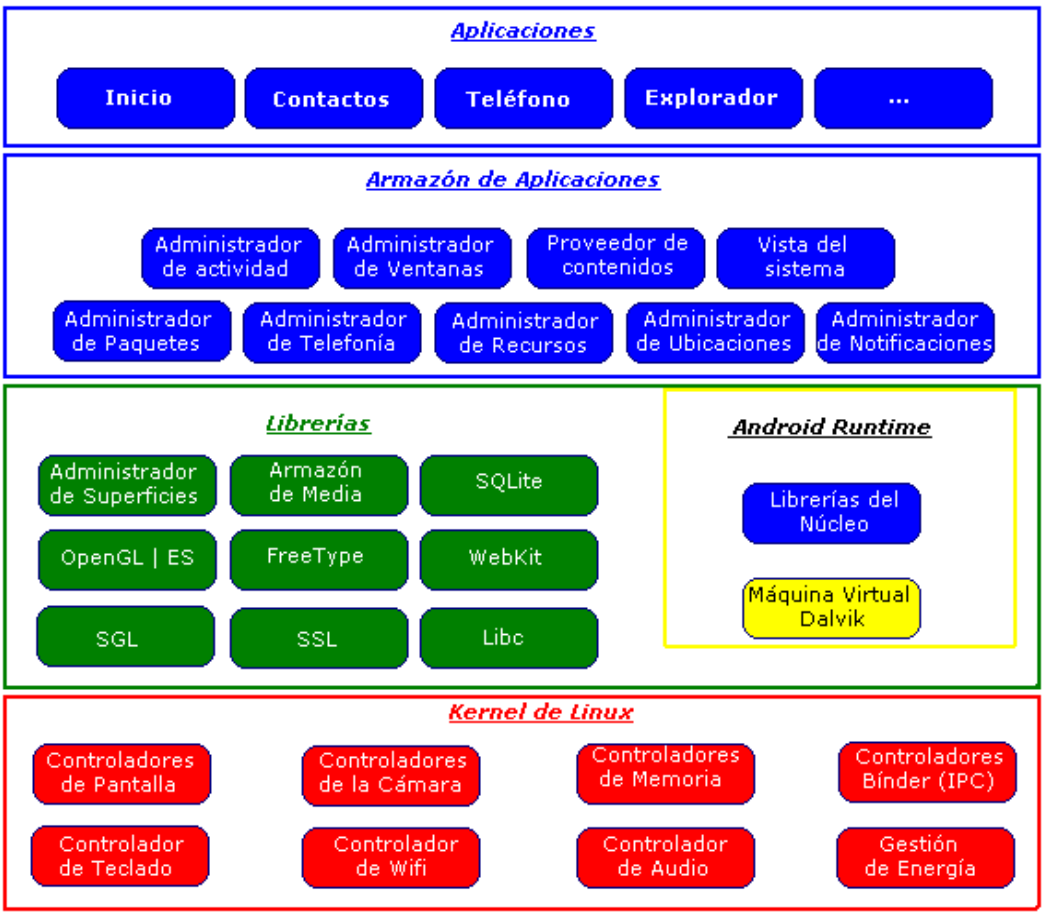


Figura 1. Sistema de capas de Android

En la imagen se distinguen claramente cada una de las capas: la que forma parte del propio Kernel de Linux, donde Android puede acceder a diferentes controladores, las librerías creadas para el desarrollo de aplicaciones Android, la siguiente capa que organiza los diferentes administradores de recursos, y por último, la capa de las aplicaciones a las que tiene acceso.

VERSIONES DISPONIBLES

El sistema operativo Android, al igual que los propios teléfonos móviles, ha evolucionado rápidamente, acumulando una gran cantidad de versiones, desde la 1.0 para el QWERTY HTC G1, hasta la 4.0 que acaba de salir al mercado.

- **Cupcake: Android Version 1.5**

  Características: Widgets, teclado QWERTY virtual, copy & paste, captura de vídeos y poder subirlos a Youtube directamente.

- **Donut: Android Version 1.6**

  Características: Añade a la anterior la mejoría de la interfaz de la cámara, búsqueda por voz, y navegación en Google Maps.

- **Eclair: Android Version 2.0/2.1**

Características: Mejoras en Google Maps, salvapantallas animado, incluye zoom digital para la cámara, y un nuevo navegador de internet.

- **Froyo: Android Version 2.2**

Características: Incluye hostpot Wifi, mejora de la memoria, más veloz, Microsoft Exchange y video-llamada.

- **Ginger Bread: Android Version 2.3**

Características: Mejoras del consumo de batería, el soporte de vídeo online y el teclado virtual, e incluye soporte para pagos mediante NFC[2].

- **Honey Comb: Android Version 3.0/3.4**

Características: Mejoras para tablets, soporte Flash y Divx, integra Dolphin, multitarea pudiendo cambiar de aplicación dejando las demás en espera en una columna, widgets y homepage personalizable.

- **Ice Cream Sandwich: Android Version 4.0**

Características: Multiplataforma (tablets, teléfonos móviles y netbooks), barras de estado, pantalla principal con soporte para 3D, widgets redimensionables, soporte usb para teclados, reconocimiento facial y controles para PS3.

ECLIPSE COMO ENTORNO DE TRABAJO

En este curso de Android, se da por supuesto que el alumno está familiarizado con el entorno Eclipse y que además tiene nociones básicas de programación en el lenguaje Java. Lo primero que necesitaremos para poder programar en Android, es preparar el entorno de trabajo. Es necesario disponer de una versión de Eclipse Galileo 3.5 o superior para poder desarrollar nuestros proyectos. Lo segundo que necesitamos es el kit de desarrollo software para Android o Android SDK, del que se pueden encontrar varias versiones para diferentes plataformas en la página web:

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

Si el sistema operativo es Windows, lo más recomendable, es descargar el instalador automático **installer_rXX-windows.exe**, y simplemente seguir las instrucciones. Una vez se inicia la instalación, el instalador comprueba si el equipo dispone del Java SE Development Kit (JDK). Si no es así, muestra un mensaje como el siguiente:

[2] NFC (Near-Field Communication) es una plataforma abierta para la comunicación instantánea.

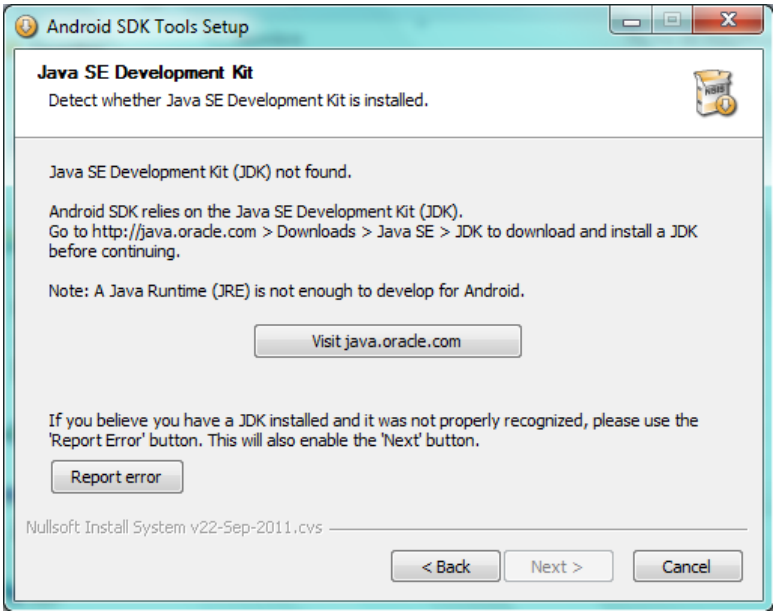


Figura 2. Android Setup

Simplemente pincha sobre el botón "Visit java.oracle.com" (Figura 1.1) que redireccionará a la página mencionada para descargar el paquete necesario. Una vez instalado el JDK, se continúa con la instalación del SDK. Cuando finalice el instalador, se ejecutará el SDK Manager, en el que se deberán seleccionar todas las casillas deshabilitadas, para instalar todas las versiones de Android así como sus herramientas (Tools).

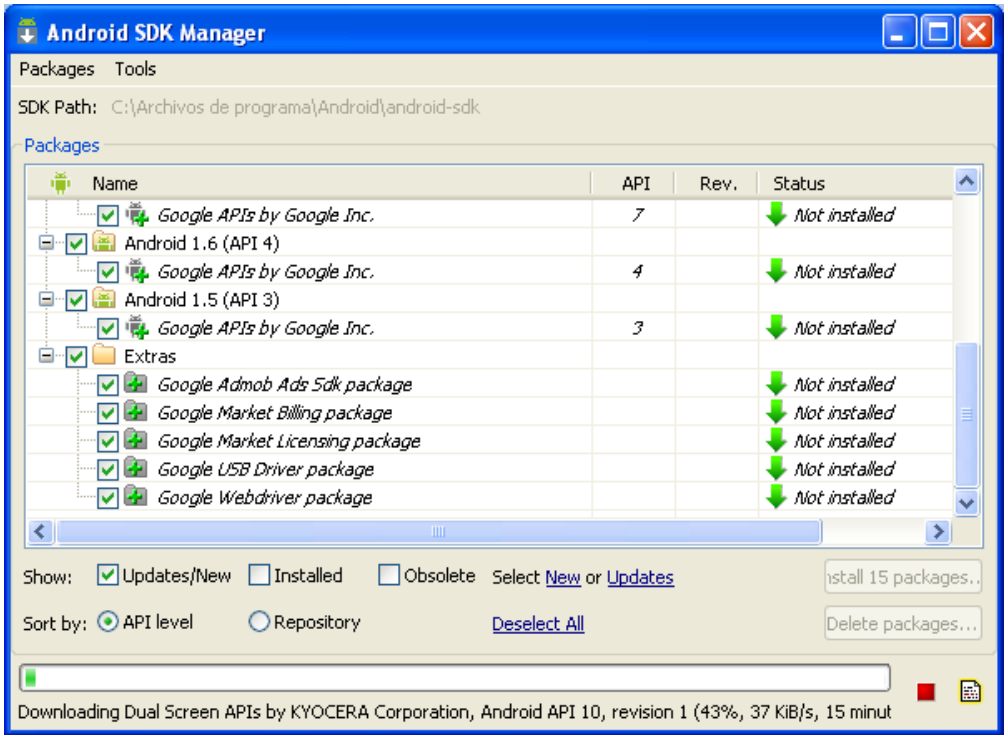


Figura 3. SDK Manager

Una vez todo esté descargado e instalado, abrir Eclipse para descargar el **ADT Plugin** e instalarlo en el entorno de desarrollo. Se deben seguir los siguientes pasos:

1.- En la pestaña "Help", seleccionar "Install New Software".

2.- Presionar el botón "Add" en la esquina superior derecha.

3.- En el cuadro de dialogo que aparece, escribir "ADT Plugin" en el campo "Name", y la siguiente URL

en el campo "Location" y pulsar "OK" (Si existe algún problema para enlazar el entorno con éste link,

probar a poner http: eliminando la 's'):

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

4.- En "Avalaible Software", seleccionar la casilla correspondiente a "Developer Tools" y pulsar "Next".

5.- Leer y aceptar el Acuerdo de licencia y presionar "Finish"(si salta una advertencia de seguridad

informando de que la autenticidad o validez del software no se puede establecer, simplemente pulsar

"OK"), y reiniciar Eclipse.

Lo único que queda es configurar el ADT Plugin. En Eclipse, en la pestaña "Window", seleccionar "Preferences", y elegir "Android" en el panel de la izquierda. Aparecerá un cuadro de dialogo preguntándo si se quiere enviar estadísticas a Google, seleccionar la elección y pulsar "Proceed". Ahora presionar el botón "Browse" y seleccionar la ruta del directorio dónde se haya ubicado el SDK (normalmente C:\Archivos de programa\Android\Android-sdk\) y pulsar "Apply" y "OK".

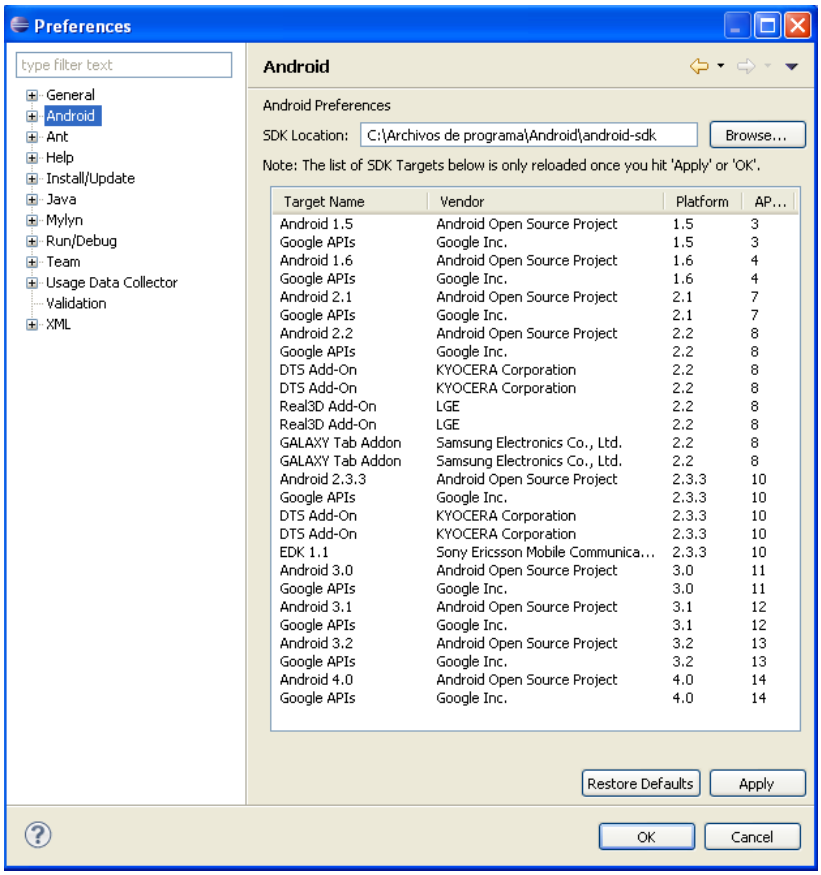


Figura 4. Preferences

Por último, hay que comprobar que el SDK está completamente actualizado. Para ello, en la pestaña "Window", seleccionar "Android SDK and AVD Manager". En la sección "Available

Packages", seleccionar todas aquellas casillas a instalar. Presionar "Install Selected" para comenzar con la descarga e instalación.

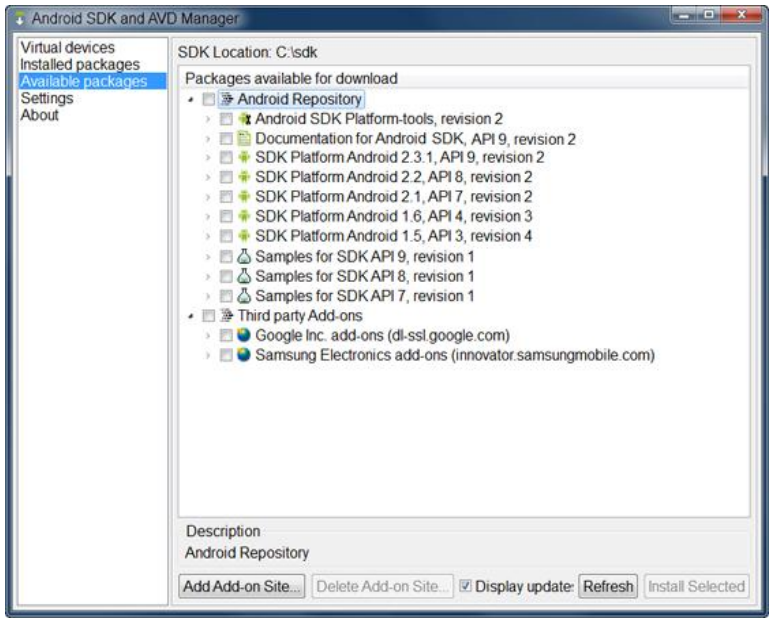


Figura 5. Repository

¡Y ya está! Ya tenemos preparado el entorno para poder programar en Android.

PERSPECTIVAS Y EMULADOR

1. PERSPECTIVA JAVA

Dados por sabidos los conocimientos básicos sobre Eclipse y la programación en Java, ésta perspectiva debe ser conocida por todos.

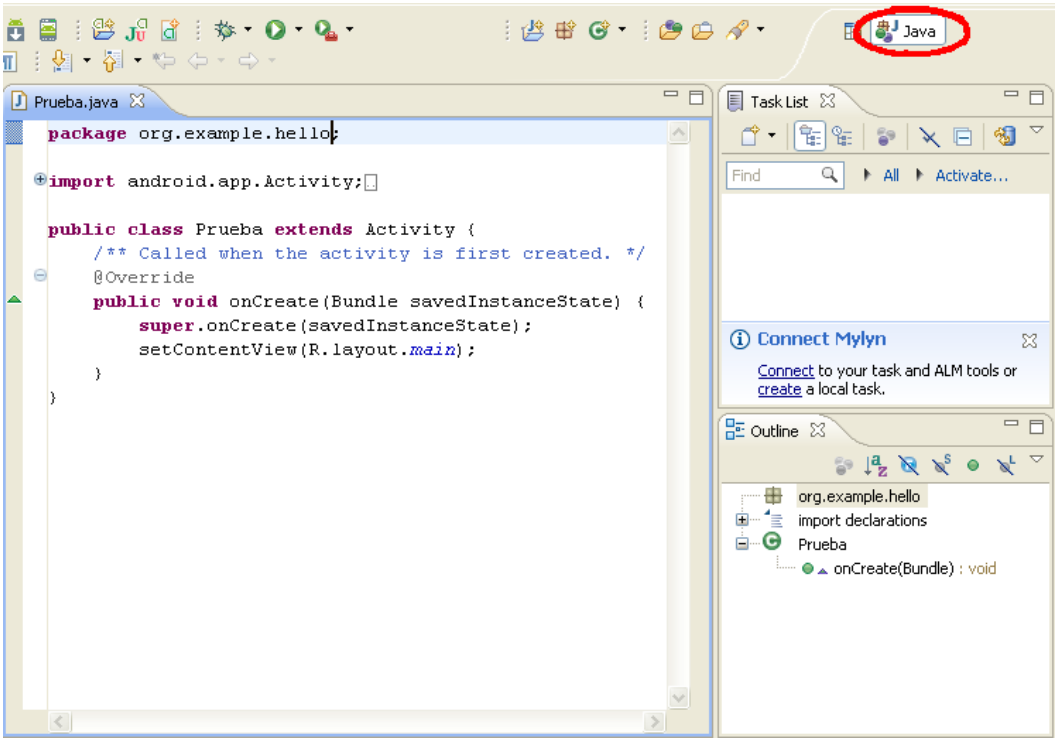


Figura 6. Perspectiva Java

Es la interfaz de usuario (o conjunto de vistas) que provee el JDT Plugin para poder programar en lenguaje Java. Esta interfaz, proporciona una serie de herramientas (se puede considerar como una determinada organización de las vistas), para el correcto desarrollo de programas y aplicaciones, y será la que utilizaremos para programar en este curso de Android.

2. PERSPECTIVA DDMS

En este caso, es el ADT Plugin el que nos proporciona la nueva perspectiva, por lo que lo primero que hay que hacer es habilitarla.

En la pestaña "Window", seleccionar "Open Perspective" -> "Other"-> "DDMS".

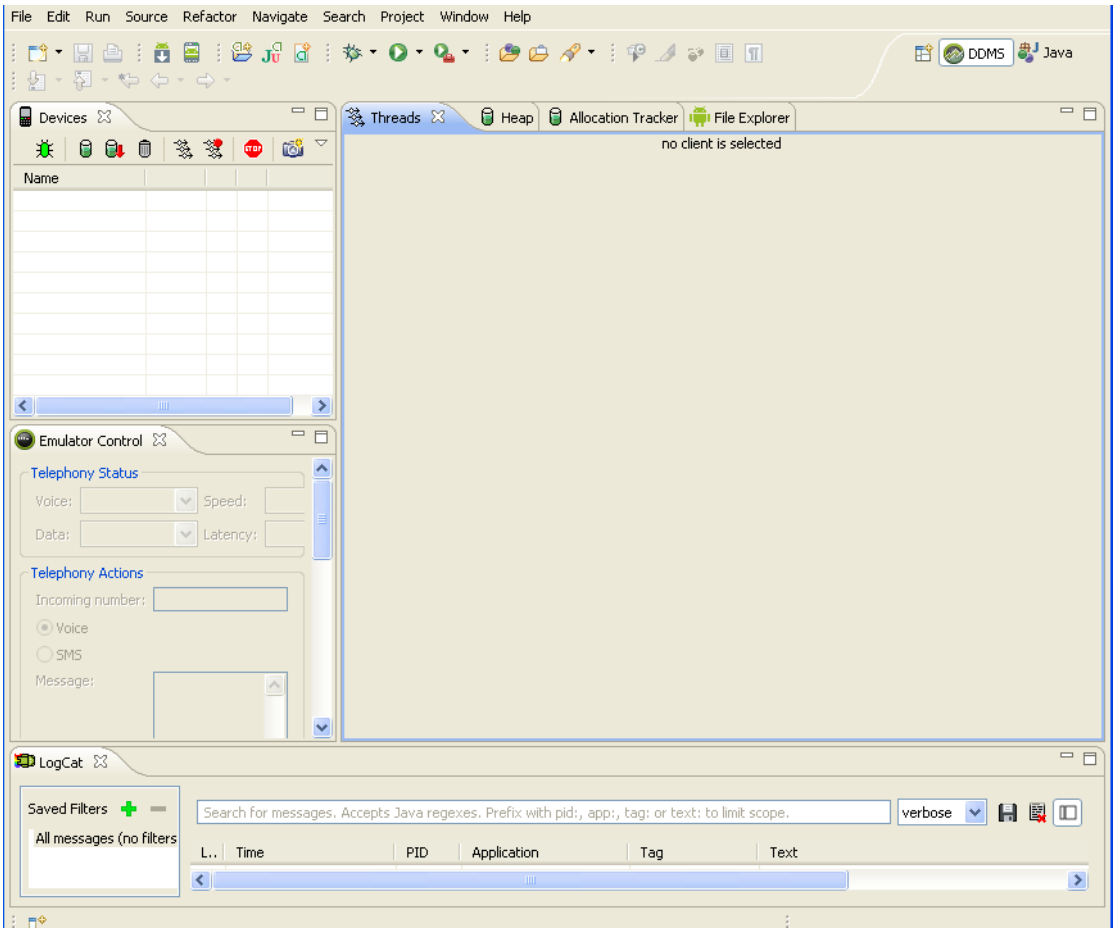


Figura 7. Perspectiva DDMS

Esta perspectiva, sirve para poder programar y realizar debugging al mismo tiempo, lo que es una forma muy efectiva de programar.

Aunque se programará con la perspectiva Java, a la hora de corregir errores se puede pasar a la perspectiva DDMS.

3. EMULADOR

Una vez tengamos el proyecto listo para ejecutar, entra en escena el emulador de Android. Éste proporciona una vista especial para comprobar si la aplicación hace lo que se desea. A continuación se muestra la vista del emulador para la versión 2.2 de Android:

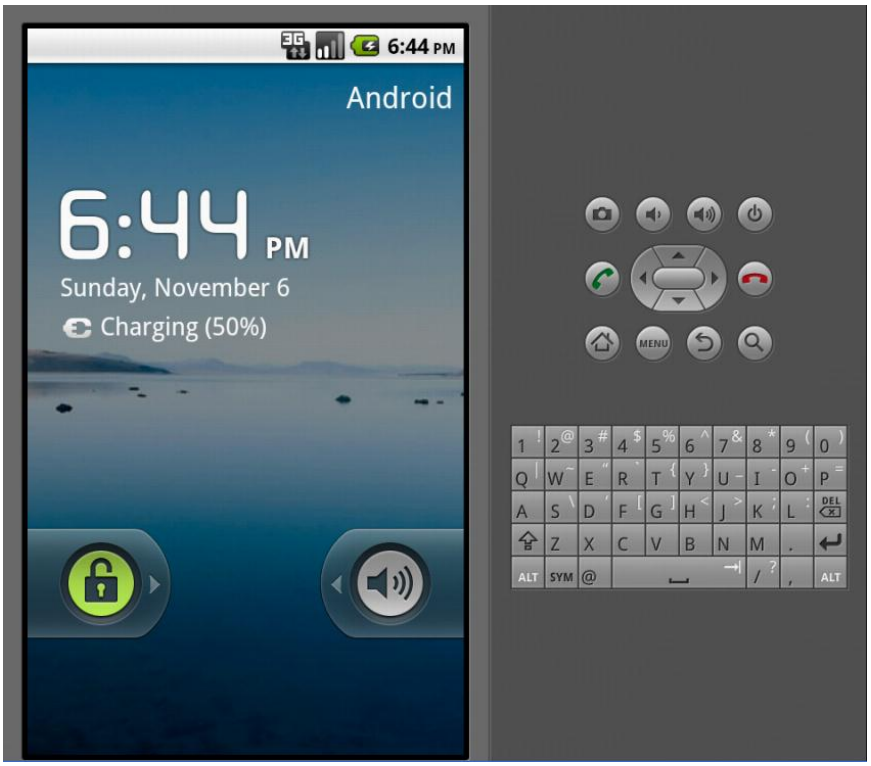


Figura 8. Emulador para Android 2.2

Lo primero que hay que hacer cuando se quiere ejecutar una aplicación, es pinchar sobre el proyecto con el botón derecho, y en "Run as" seleccionar "Android Application", entonces se lanzará el emulador más apropiado siempre que esté creado (más adelante, se explicará cómo generar los emuladores).

No se debe parar la ejecución del emulador, dado que cada vez que se ejecuta el mismo, necesita de muchos recursos del computador, por lo que tarda bastante en lanzarse, y realmente no es necesario cerrarlo, puesto que cada vez que se lleva a cabo una ejecución del proyecto, la aplicación se reinstala en el emulador.

UN EJEMPLO: "HOLA ANDROID"

Vamos a crear nuestro primer proyecto en Android, pero antes veamos de qué se compone cada uno. Al generar un nuevo proyecto de Android, dado que estamos utilizando el entorno Eclipse, éste va a generar automáticamente la distribución de carpetas que contendrá la aplicación, la cuál será común a todos los proyectos Android.

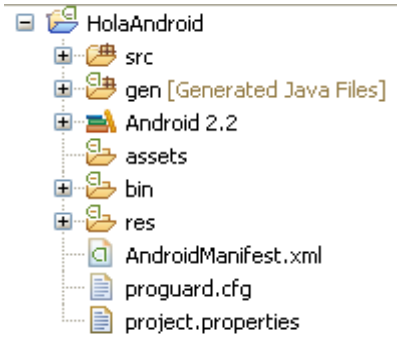


Figura 9. Sistema de carpetas de un proyecto

Veamos el significado de cada carpeta por separado:

- **Carpeta src:**

  Recoge la totalidad del código fuente (Java) de la aplicación. En el ejemplo que vamos a llevar a cabo, Eclipse generará automáticamente el código base de la ventana principal (Activity).

- **Carpeta res:**

Contiene los recursos necesarios para generar una aplicación Android:

.- res/drawable/: Guarda las imágenes y se divide en: drawable-ldpi, drawable-mdpi y drawable-hdpi, que

dependerán de la resolución del dispositivo.

.- res/raw/: Contiene archivos de propósito general, en otro formato que no es XML.

.- res/layout/: Incluye los archivos que definen el diseño de la interfaz gráfica, siempre en XML.

.- res/values/: Guarda los datos y tipos que utiliza la aplicación, tales como colores, cadenas de texto,

estilos, dimensiones…

- **Carpeta gen:**

  Ésta carpeta guarda un conjunto de archivos (de código Java) creados automáticamente cuando se compila el proyecto, para poder dirigir los recursos de la aplicación. El archivo R ajusta automáticamente todas las referencias a archivos y valores de la aplicación (guardados en la carpeta res).

- **Carpeta assets:**

Guarda el resto de archivos necesarios para el correcto funcionamiento de la aplicación, como los archivos de datos o de configuración. La principal diferencia entre los recursos que almacena ésta carpeta y los que guarda la carpeta "res", es que los recursos de ésta última generan un identificador por recurso, identificador que se encargará de gestionar el fichero R y sólo se podrá acceder a ellos a través de determinados métodos de acceso, mientras que los recursos almacenados en la carpeta "assets" no generan identificador alguno y se accederá a ellos a través de su ruta, como se hace con cualquier otro fichero.

- **Archivo AndroidManifest.xml:**

Éste archivo es uno de los más importantes de cualquier aplicación Android. Se genera automáticamente al crear el proyecto, y en él se encuentra definida la configuración del proyecto en XML (Actividades, Intents, los permisos de la aplicación, bibliotecas, etc.). Por ejemplo, el proyecto que vamos a generar ("Hola Android"), contiene un AndroidManifest.xml como el siguiente:

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

   <uses-sdk android:minSdkVersion="8" />

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:label="@string/app_name"
         android:name=".Hola" >
         <intent-filter >
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

Otra vista diferente del manifiesto es la siguiente:

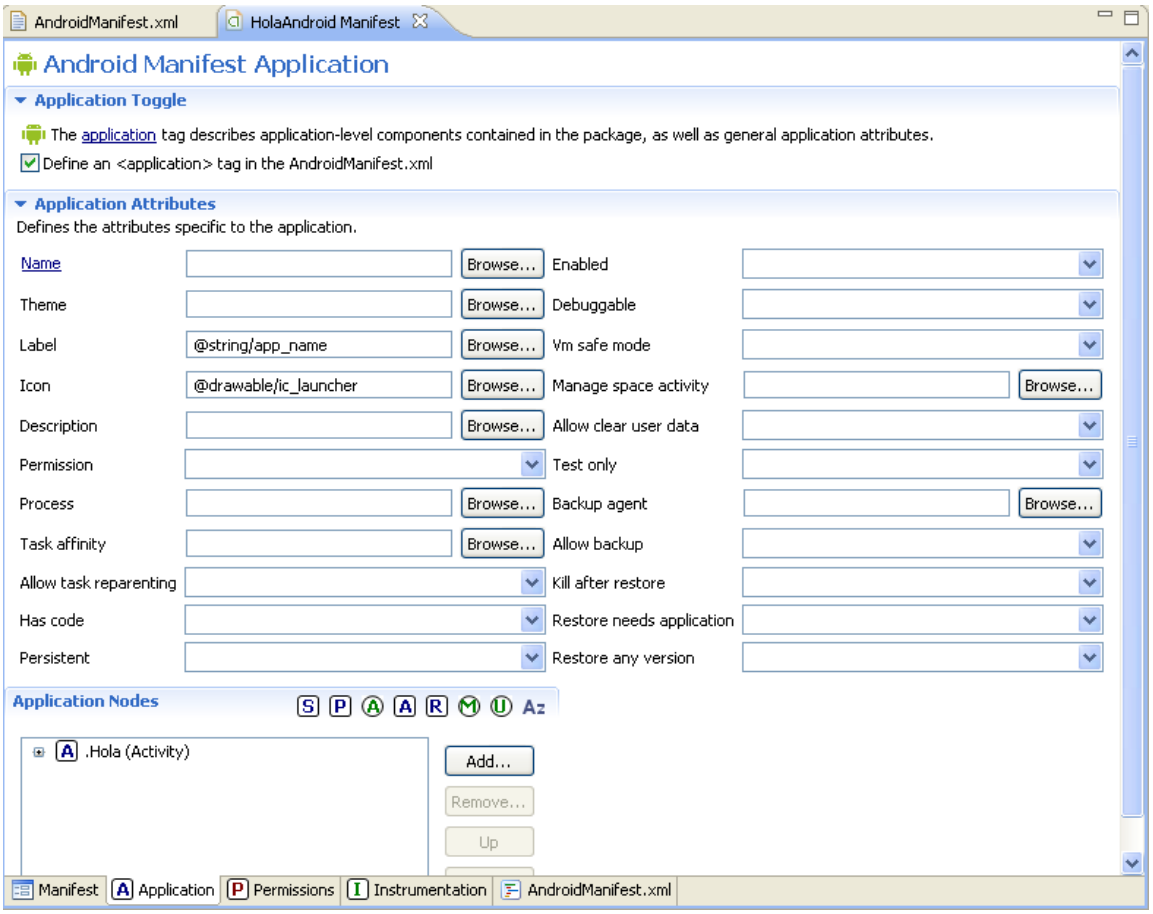


Figura 10. Editor Android Manifest

Y por fin, una vez explicadas cada una de las partes que componen el proyecto, vamos a crear nuestro primer proyecto con Android. Primero pinchar en "File"->"New"->"Other"->"Android Project", saldrá la pantalla que se muestra a continuación ("Create Android Project"). Simplemente en "Project Name" poner: HelloAndroid y pulsar "Next". En la siguiente pantalla, "Select Build Target", seleccionar la versión de Android sobre la que construir el proyecto. Vamos a seleccionar Android 2.2, para que nuestra aplicación pueda correr en cualquier terminal que tenga ésta versión o una posterior.

En la última pantalla antes de dar por concluida la configuración del nuevo proyecto, "Application Info", completar los siguientes campos:

- "Application Name": Hello, Android
- "Package Name": org.example.hello
- "Create Activity": Hello

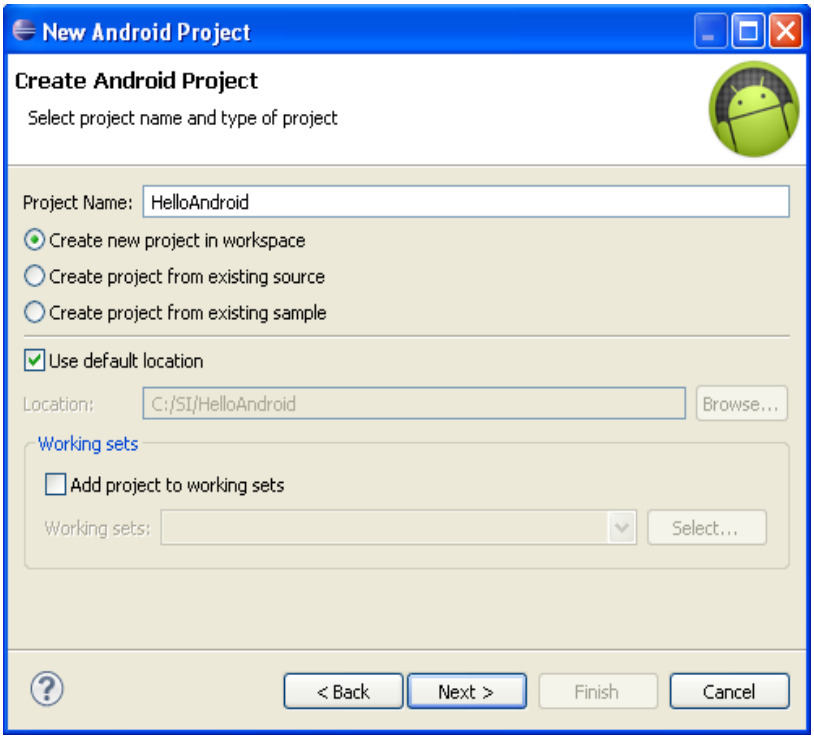


Figura 11. Create Android Project

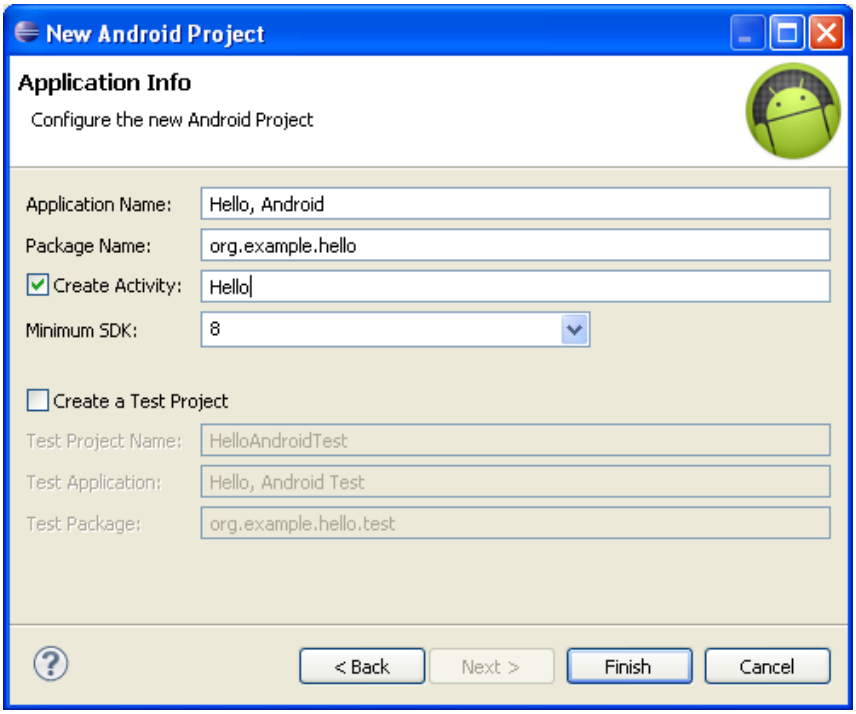


Figura 12. Application Info

Para ejecutar nuestra aplicación, primero debemos tener creado un emulador de nuestra versión de Android. Para ello, pinchar en el símbolo que abre el "Android Virtual Device Manager", y pulsar en "New":

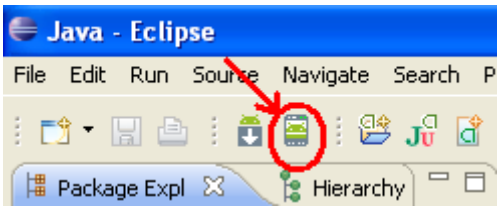


Figura 13. Símbolo Android Virtual Device Manager

En la siguiente pantalla rellenar los siguientes campos:

.- "Name": em2.2

.- "Target": Android 2.2 – API Level 8

.- "Size": 128 MiB

.- "Built-in": Default(WVGA800)

.- Si se quiere añadir funcionalidades Hardware, en "Hardware" pulsar "New", y seleccionar la

la opción/es deseada/s. En el ejemplo se ha añadido la funcionalidad "Camera support"

.- Por último, pulsar "Create AVD".

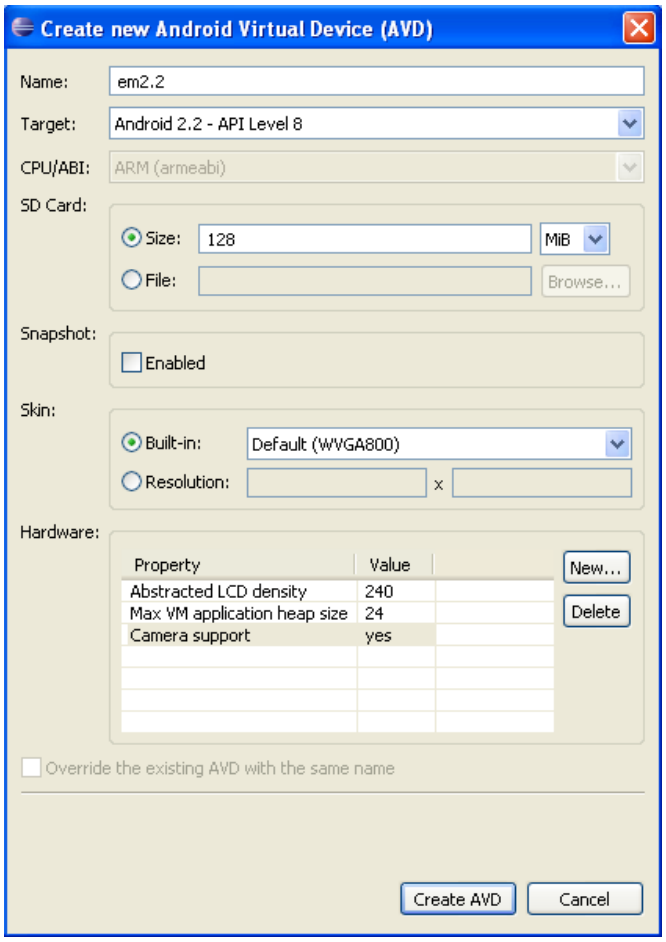


Figura 14. Create AVD

Ahora ya podemos ejecutar nuestra aplicación. Pinchar con el botón derecho del ratón sobre el proyecto, y en "Run As", seleccionar "Android Application". Se lanzará el emulador (hay que tener paciencia, pues debido a que consume muchos recursos, tardará un rato), y pasado un tiempo, se mostrará nuestro primer programa en Android:

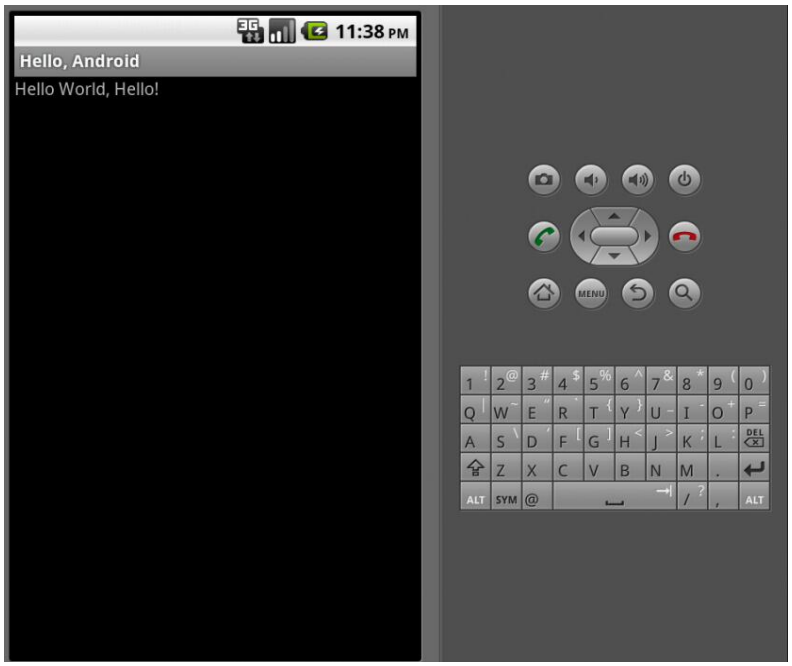


Figura 15. Simulación Hello Android

2. CONCEPTOS BÁSICOS

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

COMPONENTES DE UNA APLICACION

Para diseñar una aplicación en Android, es necesario tener claros los elementos que la componen y la funcionalidad de cada uno de ellos. Ya hemos visto el ejemplo del "Hola Android", por lo que podemos intuir algunos de ellos. Uno de los aspectos más importantes a tener en cuenta es su funcionamiento. Android trabaja en Linux, y cada aplicación utiliza un proceso propio. Se distinguen por el ID, un identificador para que solo ella tenga acceso a sus archivos. Los dispositivos tienen un único foco, la ejecución principal, que es la aplicación que está visible en la pantalla, pero puede tener varias aplicaciones en un segundo plano, cada una con su propia pila de tareas. La pila de tareas es la secuencia de ejecución de procesos en Android. Se componen de actividades que se van apilando según son invocadas, y solo pueden terminarse cuando las tareas que tiene encima están terminadas, o cuando el sistema las destruye porque necesita memoria, por lo que tienen que estar preparadas para terminar en cualquier momento. El sistema siempre eliminará la actividad que lleve más tiempo parada. En caso de que el sistema necesitase mucha memoria, si la aplicación no está en el foco, puede ser eliminada por completo a excepción de su actividad principal.

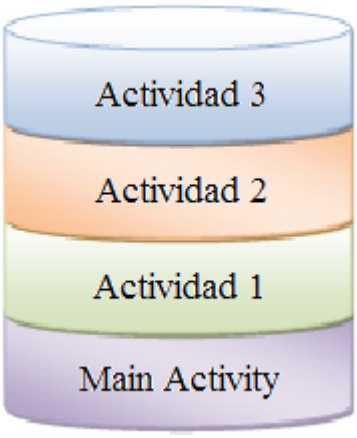


Figura 1. Pila de actividades Android

Una de las características principales del diseño en Android es la reutilización de componentes entre las aplicaciones, es decir, dos aplicaciones diferentes pueden utilizar una misma componente, aunque esté en otra aplicación para así, evitar la repetición innecesaria de código, y la consiguiente ocupación de espacio. Los componentes son los elementos básicos con los que se construyen el proyecto. Hay cuatro tipos, pero las aplicaciones se componen principalmente de actividades. Habrá tantas actividades como ventanas distintas tenga la aplicación. Sin embargo, por si solos, los componentes no pueden hacer funcionar una aplicación. Para ello están los *intents*.

Todos ellos deben declararse en el AndroidManifest.xml (junto con otros elementos que se mostrarán después) con el mismo nombre que lleve la clase asociada. Por ejemplo, la clase MainActivity, será definida en el AndroidManifest con el mismo nombre.

ACTIVIDADES

Una actividad (o Activity) es la componente principal encargada de mostrar al usuario la interfaz gráfica, es decir, una actividad sería el equivalente a una ventana, y es el medio de comunicación entre la aplicación y el usuario. Se define una actividad por cada interfaz del proyecto. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado (que se guarda en ./res/layout) para poder ser tratados en la clase NameActivity.class, que hereda de la clase Activity.

Dentro del fichero xml asociado a la actividad, se definen los elementos tales como ubicación de los elementos en la pantalla (layouts), botones, textos, checkbox, etc., cono se verá en capítulos posteriores. Las actividades tienen un ciclo de vida, es decir, pasan por diferentes estados desde que se inician hasta que se destruyen. Sus 3 posibles estados son:

- Activo: ocurre cuando la actividad está en ejecución, es decir, es la tarea principal
- Pausado: la actividad se encuentra semi-suspendida, es decir, aun se está ejecutando y es visible, pero no es la tarea principal. Se debe guardar la información en este estado para prevenir una posible pérdida de datos en caso de que el sistema decida prescindir de ella para liberar memoria.
- Parado: la actividad está detenida, no es visible al usuario y el sistema puede liberar memoria. En caso de necesitarla de nuevo, será reiniciada desde el principio.

Una vez definido el ciclo de vida, hay que tener en cuenta qué métodos son importantes en cada uno de ellos. Aquí están los métodos más importantes de una actividad:

- OnCreate (Bundle savedInstanceState): es el método que crea la actividad. Recibe un parámetro de tipo Bundle, que contiene el estado anterior de la actividad, para preservar la información que hubiera, en caso de que hubiera sido suspendida, aunque también puede iniciarse con un null si la información anterior no es necesaria o no existe.
- OnRestart(): reinicia una actividad tras haber sido parada (si continúa en la pila de tareas). Se inicia desde cero.
- Onstart(): inmediatamente después de onCreate(Bundle savedInstanceState), o de onRestart() según corresponda. Muestra al usuario la actividad. Si ésta va a estar en un primer plano, el siguiente método debe ser onResume(). Si por el contrario se desarrolla por debajo, el método siguiente será onStop(). Es recomendable llamar al método onRestoreInstanceState() para asegurar la información
- OnResume(): establece el inicio de la interactividad entre el usuario y la aplicación. Solo se ejecuta cuando la actividad está en primer plano. Si necesita información previa, el método onRestoreInstanceState() aportará la situación en que estaba la

actividad al llamar al onResume(). También puede guardar el estado con onSaveInstanceState().

- OnPause(): se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra. Guarda la información, para poder restaurar cuando vuelva a estar activa en el método onSaveInstanceState(). Si la actividad vuelve a primer plano, el siguiente método será onResume(). En caso contrario, será onStop().
- OnStop(): la actividad pasa a un segundo plano por un largo período. Como ya se ha dicho, el sistema puede liberar el espacio que ocupa, en caso de necesidad, o si la actividad lleva parada mucho tiempo.
- OnDestroy(): es el método final de la vida de una actividad. Se llama cuando ésta ya no es necesaria, o cuando se ha llamado al método finish().

Además de estos métodos, cabe destacar dos más, que son de vital importancia:

- OnSavedInstanceState(): guarda el estado de una actividad. Es muy útil cuando se va a pausar una actividad para abrir otra.
- OnRestoreInstanceState(): restaura los datos guardados en onSavedInstanceState() al reiniciar una actividad.

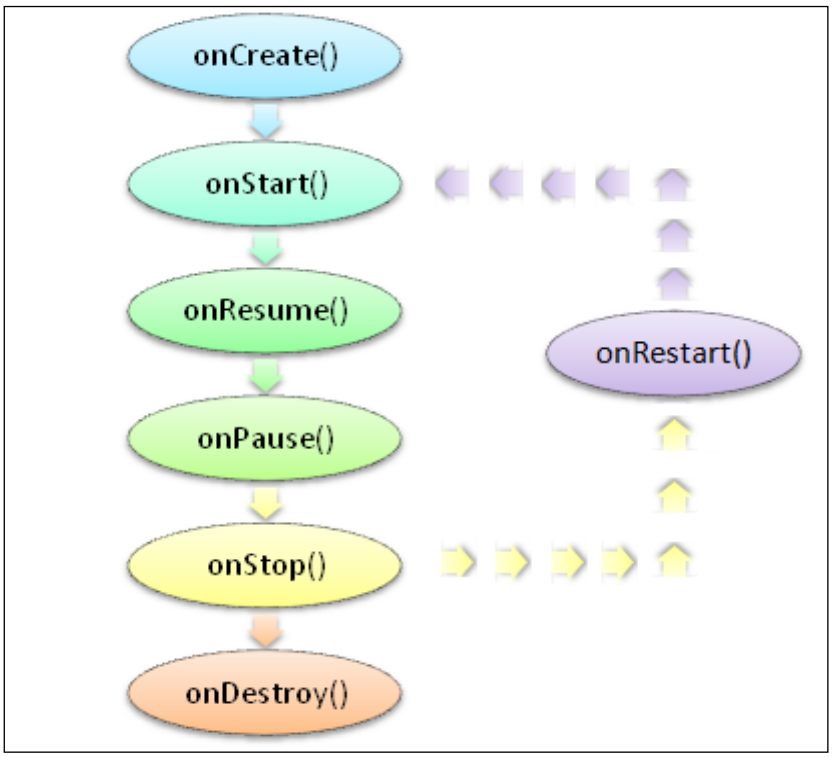


Figura 2. Ciclo de vida de una actividad

SERVICIOS

Los servicios (o service) son tareas no visibles que se ejecutan siempre por debajo, incluso cuando la actividad asociada no se encuentra en primer plano. Tiene un hilo propio (aunque no se pueden ejecutar solo), lo que permite llevar a cabo cualquier tarea, por pesada que sea. No necesita interfaz, a no ser que se pida explícitamente, en cuyo caso la clase Service la exportaría.

El ciclo de vida de un servicio se inicia con el método onCreate(Bundle), y se libera con el método onDestroy(). Sin embargo, el desarrollo puede llevarse a cabo de dos maneras, dependiendo de cómo se lance:

- Si se llama al método startService(), esto implicará que el servicio ejecutará todo su ciclo vital. El siguiente método tras onCreate(Bundle) será onStartComand(Intent, int, int). Para terminar el servicio externamente, se usa stopService(), e internamente, stopSelf() ó stopSelfResult(), ambos de la clase Service.
- En otro caso, si el servicio se llama con bindService(), el usuario podrá interactuar mediante la interfaz que exporta el servicio, y tras onCreate(Bundle) se ejecutará el método onBind(Intent). En este caso, el servicio se termina llamando al método onUnbind(Intent). También es posible reiniciarlo con el método onRebind(Intent).

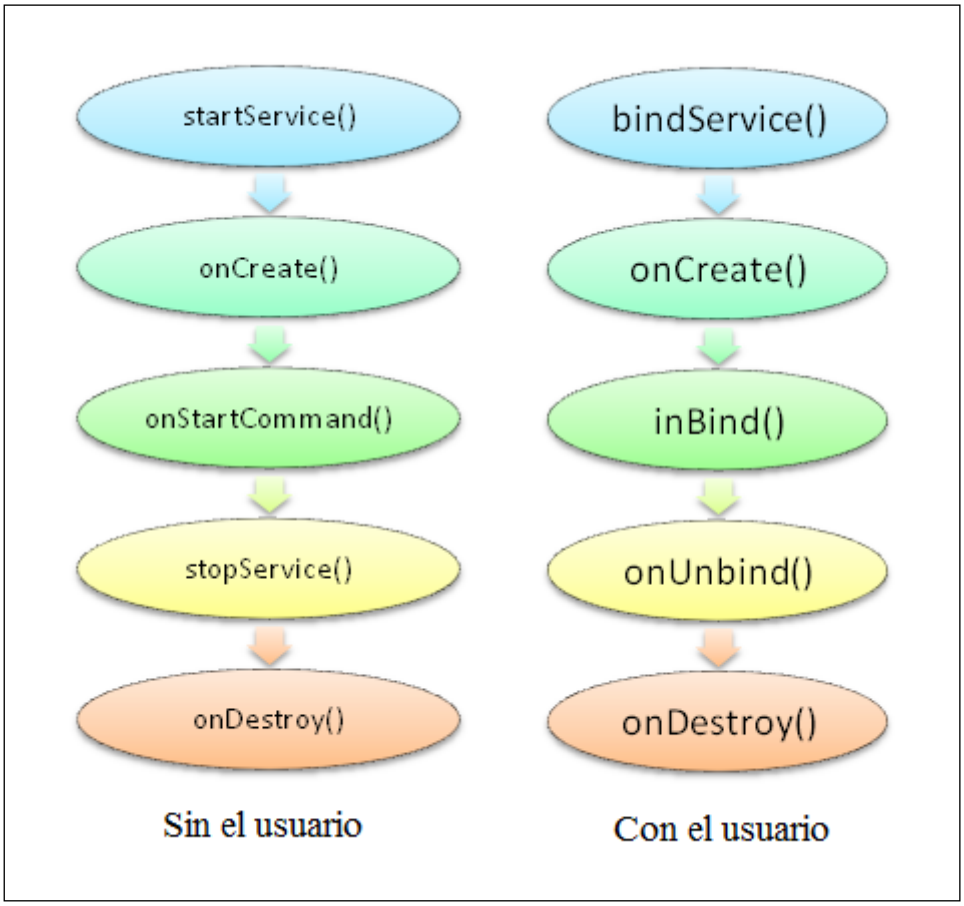


Figura 3. Ciclo de vida de un servicio

Receptores de Mensajes de Distribución

También llamados broadcast receiver o notificaciones, son los encargados de reaccionar ante los eventos ocurridos en el dispositivo, ya sean generados por el sistema o por una aplicación externa. No tienen interfaz, pero pueden lanzar una activity por medio de un evento. La clase que defina estos componentes heredará de la clase BroadCastReceiver. Su ciclo de vida es muy corto, ya que solo están activos mientras se ejecuta el método onReceive (Context, Intent), que es equivalente al onCreate(Bundle) de otros componentes. El objeto Context nos pasa es estado actual, y el intent, nos permitirá lanzar el evento.

Proveedores de contenidos

Estos proveedores en inglés llamados content provider, se encargan de que la aplicación pueda acceder a la información que necesita, siempre que se haya declarado el correspondiente provider en el AndroidManifest , compartiendo información sin revelar estructura u orden interno. Implementan una interfaz, pero se comunica con ella a través de la clase ContentResolver. Cada vez que se usa un ContentResolver, se activa un ContentProvider. Para obtener los datos necesarios, es necesario conocer la URI (identificador) del dato, los campos que tiene, y los tipos de esos campos. Con esto ya podemos llamar al método ContentResolver.query().

Intents

Los intents son el medio de activación de los componentes (excepto los content provider, que se activan usando ContentResolver). Contiene los datos que describen la operación que desarrollará el componente a quien va dirigido. Se declaran en el AndroidManifets con la etiqueta <Intent>. Pueden ser explícitos o implícitos. Los implícitos no especifican el componente al que va destinado, mientras que el explícito, si. Según el componente, los intents se tratan de diferentes maneras:

- Activity: los intents se lanzan desde el método starActivity(Intent) ó startActivitForResult(Intent). La información se extrae con el método getIntent().

  Los intents tienen definidas algunas acciones para las activity, es decir, informan de la acción a realizar. Entre ellas, por ejemplo se encuentra ACTION_CALL que inicia una llamada.

- Service: para este tipo de componentes, los intents se pasan a los métodos startService(Intent) o bindService(Intent) dependiendo del tipo de ciclo que escojamos. La información será extraída por el método getIntent() en el primer caso y onBind() en el segundo.

  Otra posibilidad es que el servicio sea lanzado por un intent, si aun no esta en funcionamiento.

- Broadcast Receiver: en este caso, el intent será enviado a todos los métodos que pueden recibir el intent : sendBroadcast(), sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle),

sendStickyBroadcast()…, que lo analizarán en su método onReceive(Context, Intent).

También tienen acciones definidas para este componente, aunque en este caso lo que hacen es informar de que ha ocurrido el evento. Por ejemplo tenemos ACTION_BATTERY_LOW, que informa de que la batería esta baja, o ACTION_SCREEN_ON, para cuando la pantalla se ilumina.

Intent-filters

Utilizados únicamente por los intents implícitos, los intent-filters definen (y delimitan) qué tipos de intent puede lanzar la actividad, o qué tipos de intent puede recibir un broadcast. Por ejemplo, para un intent que no especifica a que actividad va dirigido, se consulta el intent filter de una de ellas, y si lo satisface, el intent usará lanzará esa actividad. Se definen en el AndroidManifest con la etiqueta <intent-filter>. La información que pasan los intents debe estar contenida en la definición del intent filter para que la componente pueda ser activada (o pueda recibirlo en el caso del broadcast). Esta información se compone de tres campos:

- Action: string que informa del tipo de acción llevada a cabo. Las acciones pueden ser dadas por la clase Intent, por una API de Android o definidas por el diseñador.
- Data: informa del identificador (URI) del dato que se asocia a la acción y del tipo de ese dato. Es importante la coherencia ya que si la acción requiere un dato de tipo texto, un intent con un dato de tipo imagen no podría ser lanzado.
- Category: string que contiene información adicional sobre el tipo de componente al que va dirigido el intent. La lista de categorías esta incluida en la clase Intent

AndroidManifest

Como ya se introdujo en el tema anterior, este fichero es un documento xml en el que se declaran los elementos de la aplicación, así como sus restricciones, permisos, procesos, acceso a datos e interacciones con elementos de otras aplicaciones. Cada elemento se declara con una etiqueta única. No debe confundirse este documento con el xml asociado a cada actividad. Los elementos gráficos y distribución de la pantalla serán definidos para cada actividad dentro de su xml, pero no en el AndroidManifest. Al implementar el AndroidManifest se deben seguir unas pautas para hacer más comprensible el documento:

Código Android

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

```
<uses-sdk android:minSdkVersion="8" />
<application
        //Aquí se añadirán las acitivdades y sus permisos, acciones, etc…
</application>
```

Figura 4. Código generado automáticamente al crear el AndroidManifest

Este código es generado por el SDK a partir de la información que se ha proporcionado al crear el proyecto. Se declara el manifiesto con la etiqueta <*manifest* > y dentro se incluye el paquete en que se encuentra la aplicación y la versión del código. También incluye la versión del sdk que usa (con la etiqueta <uses-sdk>). A continuación, el usuario definirá la aplicación, incluyendo todos sus componentes en la etiqueta <application>. La declaración de componentes puede ser desordenada, pero para un mejor manejo de este fichero, se recomienda seguir algún tipo de orden. Las activities se declaran con la etiqueta <activity>. En ellas, lo primero es añadir el nombre de la actividad (android:name), que coincidirá con el de la clase en que se define el comportamiento. Además se pueden añadir imágenes, así como cambiar los atributos de que se dispone. A continuación, se declararían los intent filters asociados a la actividad, en caso de que los haya.

Los service se declaran con la etiqueta <Service> y aunque tienen menos atributos que las actividades, lo principal es darles un nombre y especificar si el sistema puede o no utilizarlo mediante el atributo enabled (android:enabled). Después irían los intent filters. Los broadcast receiver utilizan <receiver> y al igual que service, necesita los atributos name y enabled, así como intent filter en caso de necesitarlos. Todos los componentes anteriores declaran del mismo modo sus intent filters. Los content provider utilizan la etiqueta <provider> y son los únicos componentes en los que no se declaran intent filters, ya que no son necesarios. De nuevo el único atributo necesario es el nombre.

Un ejemplo práctico

A continuación se describe un ejemplo en Android aplicando los recursos aprendidos anteriormente. Al comienzo de un proyecto, tras crearlo en eclipse, lo único que tenemos es la MainActivity, o clase principal. En la siguiente imagen se muestra un ejemplo en el que se puede ver la estructura básica de una activity. Se sobrescribe el método onCreate(Bundle) y se muestra con el método setContentView( ). En la actividad se definirán las acciones propias de los elementos del xml. En este caso, tenemos un botón que, al presionar, abre una nueva actividad llamada ProductoActivity.

Código Android

```
package example.app


import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
        /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Button boton = (Button) findViewById(R.id.botonInicial);
      boton.setOnClickListener(new OnClickListener(){
         public void onClick(View v){
                  Intent i = new Intent();
                  i.setClass(getApplicationContext(), ProductoActivity.class);
                  startActivity(i);
         }
      });
   }
```

Figura 5. Ejemplo de Activity

Es conveniente implementar la acción del botón back, pausando la actividad, sin destruirla. Además, también se puede incluir la típica pregunta "¿Está seguro de que desea salir?":

Código Android

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      switch (keyCode){
            case KeyEvent.KEYCODE_BACK:
            builder.setMessage("¿Seguro que deseas salir?");
```

```
            builder.setCancelable(false);
            builder.setPositiveButton("Sí", new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
                  onPause();
                  System.exit(RESULT_OK);
               }  });
             builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
                        //Acciones que se ejecutan al presionar el botón No
               }
            });
            break;
      }
      AlertDialog alert = builder.create();
      alert.show();
      return true;
   }
```

Figura 6: Implementación del botón back.

El archivo xml asociado a la MainActivity contendría el botón con el id botonInicial.

Código xml

```
<?xml version="1.0" encoding="utf-8"?>
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:background="@drawable/prod_background"
      android:gravity="center"
      android:orientation="vertical" >
      <Button
            android:id="@+id/botonInicial"
```

```
            android:layout_width="112dp"
            android:layout_height="146dp"
            android:layout_gravity="center"
            android:background="@drawable/logo"
            android:gravity="center" />
</LinearLayout>
```

Figura 7. Ejemplo xml de Main activity

Una vez la actividad está definida, hay que tener claro qué recursos le serán necesarios. En este caso, y para que sirva de ejemplo, la actividad activará un Service (adecuadamente añadida en el androidManifest) que no tendrá que interactuar con el usuario, por lo que no habrá que sobrescribir el método onBind(Intent). Los métodos que se necesitan son onCreate(), startService(), onStartCommand y onDestroy(). En este ejemplo, el servicio reproducirá un sonido al inicio de la aplicación. De la gestión del sonido se ocupa la clase Reproductor.

Código Android

```
package example.app


import android.app.Service;
import android.content.Intent;
import android.os.IBinder;


public class MyService extends Service {
    private static Reproductor sonido;
    @Override
    public void onCreate() {
        sonido = Reproductor.create(this, R.raw.sonido_inicio);
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        sonido.start();
        onDestroy();
        return Service.START_NOT_STICKY;
    }
```

```
    @Override
    public IBinder onBind(Intent arg0) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Figura 8: Implementación de un servicio.

Cuando se ha creado el servicio, se añade en la actividad en la que será llamado, en este caso, la actividad Main:

Código Android

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
     @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button boton = (Button) findViewById(R.id.botonInicial);
        Intent servicio = new Intent(this, MyService.class);
        startService(servicio);
        boton.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                Intent i = new Intent();
                i.setClass(getApplicationContext(), ProductoActivity.class);
                startActivity(i);
            }
        });
}
```

Figura 9. Uso de Service en la clase MainActivity

Otro componente que se puede incluir en la aplicación es el broadcast receiver, por ejemplo para que al registrar una llamada entrante, la aplicación actúe de un modo determinado: se cierre, se suspenda o guarde la información existente.

Código Android

```
package com.rec.app;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.TelephonyManager;

public class MyBroadcastReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            String llamando =
    intent.getStringExtra(TelephonyManager.EXTRA_STATE_RINGING);
             if (!llamando.equals(TelephonyManager.EXTRA_STATE_RINGING)){
                 //Acciones llevadas a cabo por la aplicación cuando se recibe una llamada
            }
        }
    }
```

Figura 10. Ejemplo de Broadcast Receiver

Se puede añadir tantos componentes como sean necesarios. También es conveniente incluir algún proveedor de contenidos, pero para ello es necesario estudiar previamente el manejo de bases de datos. No se debe olvidar incluir todos estos componentes en el AndroidManifest.

Código xml

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

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



<application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".MainActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".LocalizaActivity" />
        <activity android:name=".ProductoActivity" />

        <service android:name=".MyService" />

        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE" />
            </intent-filter>
        </receiver>
        <uses-permission android:name =
                "android.permission.READ_PHONE_STATE"/>
    </application>
</manifest>
```

Figura 11. Ejemplo AndroidManifest

En resumen, las aplicaciones en Android tienen diverso grado de dificultad, dependiendo de su funcionalidad, pero la estructura siempre es la misma. Uniendo estos conocimientos al uso de

recursos gráficos, bases de datos, mapas, y otros elementos, las posibilidades de estas aplicaciones son bastante amplias.

3. INTERFAZ DEL USUARIO

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

BREVE INTRODUCCIÓN

La interfaz de usuario es la principal sección de interacción entre persona y dispositivo. A todas las funcionalidades disponibles se accede a través de la pantalla, que es por donde se muestra. Es muy importante conseguir que el manejo sea intuitivo y sencillo, y que el aspecto visual sea atractivo.

Para construirla, se emplean diferentes objetos que veremos a continuación, todos ellos descendientes de la clase View. Fundamentalmente hay 2: los propios objetos de tipo View, como por ejemplo botones o etiquetas, y que son la base de una subclase llamada widgets; y los de tipo ViewGroup, que es una clase que extiende a View, y que son la base de una subclase llamada layouts. La estructura de la interfaz puede resumirse en el cuadro que se muestra a continuación.

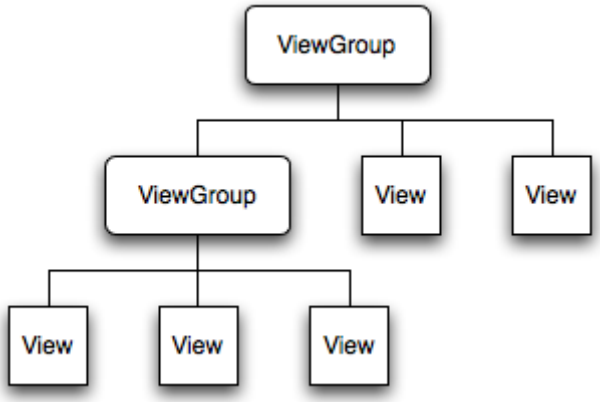


Figura 1. Estructura de la interfaz

Como podemos comprobar, los nodos ViewGroup son contenedores de elementos de tipo View e incluso de otros elementos de su mismo tipo. Los controles de tipo View (obsérvese que los nodos de este tipo son siempre hojas del árbol) son con los que el usuario interactúa.

LAYOUTS EN XML

Un layout es un recurso con el que puedes describir lo que quieres mostrar por pantalla y cómo lo quieres mostrar. La manera más común de crearlo es a través de un archivo XML (en el directorio res/layout del proyecto), con un formato muy similar a HTML, que sigue este patrón: <Nombre_del_layout atributo1=”valor1”… atributoN=”valorN”> elementos/componentes </Nombre_del_layout>. Una vez se ha creado el archivo XML que definirá el layout, hay que cargarlo desde el código de la aplicación, en el onCreate() de la actividad. Debe ser similar a esto:

```
public void onCreate(Bundle savedInstance) {
   super.onCreate(savedInstance);
   setContentView(R.layout.nombre_del_layout);
}
```

Todos los layouts tienen unos parámetros específicos. Los únicos que son comunes a todos son layout_height y layout_width, que sirven para especificar el valor de la altura y anchura del entorno, respectivamente. Aunque se pueden emplear valores numéricos, con “wrap_content”, el tamaño se ajusta a las dimensiones del contenido. Con “fill_parent”, será tan grande como lo permita su padre o contenedor. Estas 2 opciones son más recomendables que el ajuste manual. También se pueden establecer y consultar los márgenes, bordes y la posición del layout, entre otras opciones, mediante métodos y funciones a los que, en esta ocasión, se llaman desde el código de la aplicación.

Cada componente tiene su propia variedad de atributos en XML. El atributo “ID” se encarga de distinguirlos del resto, otorgándole un nombre único. Una vez establecido (mediante, por ejemplo, android:id="@+id/nombre"), para referenciarlo desde el código de la aplicación es preciso hacerlo de esta manera:

```
Tipo myTipo = (Tipo) findViewById(R.id.nombre);
```

Existen otros muchos atributos, que varían dependiendo del componente con el que estemos tratando, y con los que podemos establecer el color de fondo, tamaño, gravedad y un sinfín de opciones más.

FRAME LAYOUT

Es el más simple de todos los existentes. Todos los objetos que se introduzcan se situarán en la esquina superior izquierda, por lo que si hay más de uno, se ocultarán total o parcialmente entre ellos, salvo que los declaremos como transparentes. Por este motivo, su uso ideal es el de mostrar una sola imagen que complete toda la pantalla.

LINEAR LAYOUT

Se trata del Layout que viene por defecto, y uno de los más sencillos. Los objetos son estructurados horizontal o verticalmente, dependiendo del atributo “orientation” de su archivo XML correspondiente, y siempre en una única fila o columna, con un comportamiento similar al de una pila. Aquí podemos ver un código de ejemplo:

Figura 2. Frame Layout

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

   <EditText android:id="@+id/NombreEditText"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />

   <Button android:id="@+id/NombreButton"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent" />
</LinearLayout>
```

TABLELAYOUT

Utilizando esta opción, se consigue una distribución tabular de los elementos de nuestra interfaz. El comportamiento es similar al empleado en HTML: se definen las filas, y dentro de ellas, las columnas. La tabla tendrá tantas columnas como la fila con un mayor número de celdas. En cada casilla, se podrá introducir el objeto deseado (e incluso dejarla vacía). También existe la posibilidad de combinar celdas.

Por lo general, la dimensión de cada casilla la determina el elemento contenido en ella. No obstante, este comportamiento puede variarse utilizando diversos atributos.

Figura 3. Table Layout

RELATIVE LAYOUT

Es la opción que ofrece más posibilidades, y por tanto, la más compleja. Básicamente, empleando este Layout podremos colocar cada elemento en el lugar que deseemos, basándonos en su posición relativa al padre (contenedor) o a otros elementos existentes, pudiendo modificar las distancias entre objetos al antojo del programador.

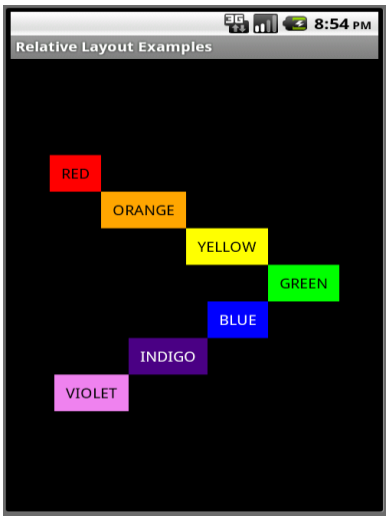


Figura 3. Relative Layout

OTROS

Existen, además, otros elementos con comportamiento similar a los Layouts que hemos visto hasta ahora. Algunos de ellos proporcionan estructuras invisibles para el usuario (como los aquí descritos hasta ahora) y otros sí que las muestran. Algunos ejemplos son Gallery, GridView, Spinner o ViewFlipper.

EVENTOS DE USUARIO

Los eventos de usuario sirven para capturar la interacción del usuario con una determinada aplicación. Existen varias maneras de realizarlo.

EVENT LISTENERS

Un Event Listener es una interfaz de la clase View a través de la cual se pueden detectar diferentes tipos de pulsación del usuario sobre el dispositivo. Los métodos más comunes que incluye esta interfaz son:

-**onClick():** este método es llamado cuando el usuario hace una pulsación simple, ya sea por contacto, con teclas de navegación o de cualquier manera posible, sobre un determinado elemento de la interfaz. Una posible implementación, sobre un botón:

```
private OnClickListener nombreEvento = new OnClickListener() {
   public void onClick(View v) {
     // hacer lo que se desee
   }
};


Button boton = (Button)findViewById(R.id.corky);
boton.setOnClickListener(nombreEvento);
```

-**onKey():** este método es llamado si el usuario presiona determinada tecla o botón de su dispositivo mientras el cursor está situado sobre el elemento deseado.

-**onTouch():** este método es llamado cuando el usuario toca la pantalla y realiza una presión, se despega o se mueve por ella.

Para implementar los métodos aquí vistos y otros existentes, es preciso implementarlos en la Activity correspondiente o definirlos como una nueva clase anónima. Tras esto, deberemos pasar una instancia de la implementación a su respectivo método, con, por ejemplo, setOnClickListener() de OnClickListener.

EVENT HANDLERS

Si se está creando un componente de la clase View sobre cualquier tipo de Layout, se pueden definir varios métodos por defecto, denominados Event Handlers, que son llamados cuando se produce un evento de cualquier tipo, como puede ser una pulsación sobre la pantalla.

TOUCH MODE

Determinados dispositivos pueden ser controlados a través de un cursor con sus respectivas teclas de movimiento, tocándoles la pantalla, o ambas. En el primer y último caso, es preciso contar con un selector de componente cuya función sea advertir sobre el elemento con el que se está tratando en cada momento, dependiendo del movimiento y situación del cursor. Sin embargo, en el momento que el usuario "toca" la pantalla, no es necesario resaltar el elemento con el que se va a trabajar. Por ello, en caso de tratar con un dispositivo que permita ambos tipos de navegación a través de su interfaz, existirá un modo llamado "touch mode", que determinará si es necesario resaltar el componente con el que se va a trabajar o no.

HANDLING FOCUS

Por lo comentado en el punto anterior, el sistema debe encargarse del selector de componente, manejando una rutina, en respuesta a la entrada dada por el usuario. Esto incluye el dónde situar el foco si un elemento es suprimido u ocultado, basándose en un algoritmo que selecciona al componente vecino más cercano.

MENÚS Y BARRAS DE ACCIONES

Los menús son la forma más habitual de proporcionar al usuario una serie de acciones a realizar, ya sea sobre una aplicación o sobre las opciones del propio dispositivo. Para crearlo, la mejor opción es definirlo en un archivo XML que irá contenido en el directorio res/menú del proyecto. Los elementos que lo componen son <menú>, que es el contenedor principal del resto de elementos; <ítem>, que representa cada una de las opciones que ofrece el menú, y <group>, que posibilita agrupar los "ítems" del menú a gusto del usuario, y que puede ser visible o no. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu  xmlns:android="http://schemas.android.com/apk/res/android
">
 <item android:id="@+id/Opcion1" android:title="Opción1"
      android:icon="@drawable/miIcono1"></item>
<item android:id="@+id/ Opcion2" android:title="Opción2"
      android:icon="@drawable/ miIcono2"></item>
</menu>
```

Hay 3 tipos de menús de aplicación:

-**Menú principal:** es la colección primaria de una actividad, que aparece cuando el usuario acciona el botón "Menú" del dispositivo. Una forma de crearlo es esta:

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.miMenu, menu);
    return true;
}
```

Figura 4. Menú principal

-**Menú contextual:** es una "lista" que surge en la pantalla que aparece cuando el usuario mantiene presionado un elemento determinado. Se implementa como el caso anterior, pero añadiendo un nuevo elemento de tipo <menu> al mismo nivel que los elementos <ítem>.

Figura 5. Menú contextual

-Submenús: es una lista de opciones que surge cuando el usuario acciona un elemento del menú, que contiene otro menú anidado.

Las barras de acciones son barras de título que, además de mostrar algún tipo de información, pueden ejecutar acciones. El aspecto de ésta es acorde con el de la aplicación a la que pertenece. Es importante saber que sólo están disponibles para versiones de Android a partir de la 3.0 (versión 11 del sdk).

DIÁLOGOS Y NOTIFICACIONES

Los diálogos son avisos o comprobaciones que surgen de una determinada aplicación, en forma de ventana. La principal diferencia entre ellos y las notificaciones es que las segundas son meramente informativas, no se ejecutan en primer plano, y no requieren interacción directa con el usuario, mientras que los primeros sí se realizan en primer plano y pueden necesitar esa interacción para llevar a cabo una tarea, aunque se puede decir que un diálogo es un tipo de notificación. Un diálogo se crea con la instrucción "onCreateDialog(numero)" como parte de una actividad, y también se muestra como tal, con "showDialog(numero)", donde "numero", en ambos casos, representa un identificador único de cada dialogo. Para rechazarlos, se puede conseguir con dismiss() o dismissDialog(int). Si se desea modificar el texto a mostrar o cualquier otro atributo, existe una colección de métodos, variable en función del tipo concreto del diálogo, que facilitan la labor. Podemos enumerar 3 tipos de diálogos:

-**AlertDialog:** Se recomienda usarlo para mostrar un título, un mensaje de texto, uno o varios botones, o una lista de elementos para seleccionar.

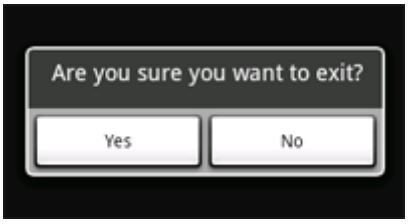


Figura 6. Diálogo de Alerta

En un diálogo con botones, se añaden también a través de métodos específicos del tipo concreto, y es preciso incorporar un evento que detecte su pulsación. Como ejemplo, el código de la última imagen mostrada en este manual:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
              MyActivity.this.finish();
          }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
              dialog.cancel();
          }
```

```
        });
AlertDialog alert = builder.create();
```

En caso de que se desee incorporar una lista de elementos, es preciso crear un array de strings, donde cada elemento corresponderá a cada opción que se mostrará por pantalla, y posteriormente añadirla al diálogo de manera similar a como se ha visto para los botones.

-**ProgressDialog:** Su función es indicar que una tarea está siendo llevada a cabo. Si se conoce el esfuerzo que va a requerir, se mostrará una barra de progreso que suministre la información sobre lo que se lleva completado y lo que resta; y si no se conoce, se mostrará un haz que describirá un movimiento circular. Este tipo de diálogo puede incorporar también botones, cuya función puede ser cancelar el progreso. Aquí, un ejemplo de cada (el código y la imagen no están relacionados):

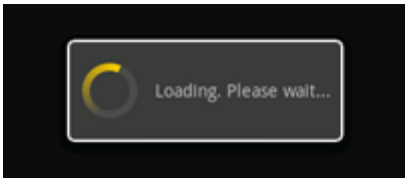


Figura 7. Diálogo de Progreso

```
ProgressDialog dialogo = new ProgressDialog(contexto);
dialogo.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
dialogo.setMessage("Cargando...");
```

-**Diálogo personalizado:** Android da la opción de crear un diálogo con los elementos/aspecto que el usuario desee. Para ello, es preciso crearse un layout en XML, guardarlo como "custom_dialog.xml", y añadirlo como vista al crear el diálogo. Además, se pueden modificar otros atributos de la misma manera que hemos visto para otros casos.

Hay 2 tipos de notificaciones:

-**Toast Notification:** Esta notificación aparece sobre la ventana en la que estemos trabajando, y solamente ocupa el espacio que necesite el mensaje que muestra. No cierra ni modifica la actividad que estemos llevando a cabo, ni acepta interacción con el usuario, por lo que se desvanecerá en un periodo corto de tiempo.

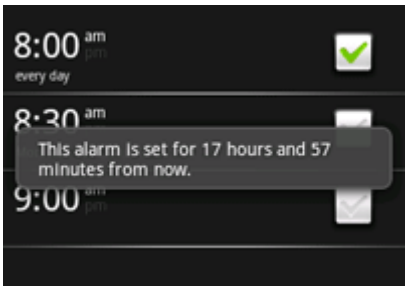


Figura 8. Toast Notification

Para crearla, mostrarla y situarla, es suficiente con incorporar unas instrucciones similares a las mostradas a continuación:

```
Toast toast = Toast.makeText(context, text, duration);
toast.show();
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

También se puede crear un layout (en este manual ya se ha visto cómo) y utilizarlo como una notificación de este tipo.

-**Status Bar Notification:** Esta notificación añade un mensaje en la ventana de notificaciones del sistema, y un icono en la barra de estado que, al seleccionarlo, lanzará la aplicación que lo ha activado. Además, se pueden configurar de tal modo que al producirse, el dispositivo suene, vibre o alerte al usuario de alguna manera.

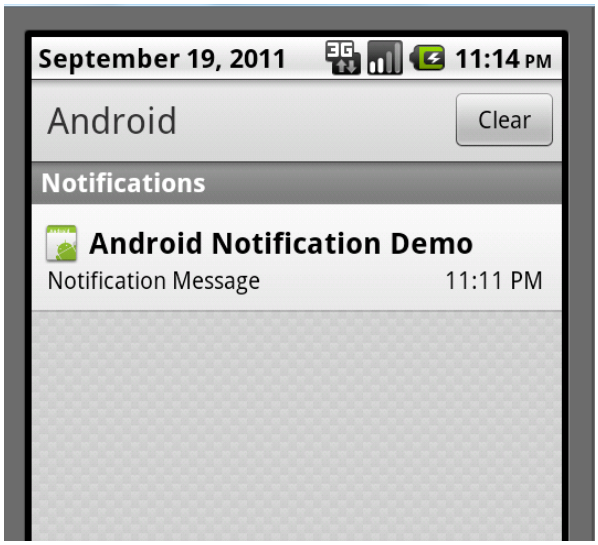


Figura 9. Status Bar Notification

Este tipo de notificaciones sólo deben referirse a actividades que se estén ejecutando en "background", y no podrán pasar a "foreground" sin la acción del usuario.

ESTILOS Y TEMAS

Un estilo es una colección de propiedades que permite especificar el aspecto y formato de una vista o una ventana. Un tema es lo mismo que un estilo, pero aplicado a una actividad al completo, no sólo a una vista.

Para definir uno o varios estilos, se crea un archivo XML en el directorio res/values/ del proyecto. Por cada uno nuevo, se debe incorporar un par de etiquetas <style></style>, que debe ir, a su vez, entre unas globales dentro de ese documento (<resource> </ resource >). Por cada propiedad del estilo que se esté creando, debemos añadir un elemento <ítem>, otorgándole un nombre con "name". Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="NombreEstilo1"
parent="@android:style/TextAppearance.Medium">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#00FFFF</item>
      <item name="android:typeface">monospace</item>
   </style>

  <style name="NombreEstilo2" parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#FF0000</item>
  </style>
</resources>
```

Para aplicar un estilo a una vista individual, es suficiente con añadir el atributo "style" al elemento deseado dentro del layout, en el archivo XML correspondiente. Por ejemplo:

```
<TextView    style="@style/NombreEstilo1"   android:text="@string/hola" />
```

Para aplicar el estilo a una actividad (aplicación) al completo, es preciso incorporar el atributo "android:theme" en el <activity> (<application>) del Android Manifest. Por ejemplo:

```
<activity android:theme="@style/ NombreEstilo1">
```

Además, Android contiene una gran cantidad de estilos y temas predefinidos a completa disposición del usuario.

4. RECURSOS DE APLICACIÓN

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

DEFINIENDO RECURSOS

Para la programación en Android, resulta conveniente separar los recursos que vaya a necesitar la aplicación (como imágenes u otro tipo de variables), de su código, de tal modo que se puedan mantener independientemente.

Conviene agruparlos en carpetas. El directorio res/, contiene todos los grupos de recursos, y permite contener dentro otros tales como: animator/, anim/, color/, drawable/, layout/, menú/, raw/, values/, xml/. Excepto la 5ª y la 7ª opción de la lista anterior, que soportan recursos de tipo Bitmap o archivos arbitrarios, respectivamente, el resto sólo pueden contener documentos en XML.

También se debería especificar un recurso para diferentes configuraciones posibles de los dispositivos, puesto que mientras se ejecuta una aplicación, éste sistema operativo utiliza los recursos apropiados para la configuración activa. Es decir, que conviene tener recursos alternativos (drawable, por ejemplo), por si el que tenemos por defecto no se ajusta apropiadamente en caso de utilizar un dispositivo no habitual.

Para especificar esta configuración alternativa, se ha de crear una nueva carpeta dentro de res/, del estilo <resources_name>-<config_qualifier>, que representan el directorio por defecto de un recurso determinado, y una configuración individual sobre en qué caso se puede usar ese recurso, respectivamente. Se permite añadir más de un <config_qualifier>, pero siempre separados con "-". Posteriormente, sólo queda incorporar los recursos alternativos a la nueva carpeta, que deben tener el mismo nombre que los que son "por defecto". Por ejemplo, donde hdpi se refiere a dispositivos con densidad alta de pantalla:

```
res/
    drawable/
        icono.png
        fondo.png
    drawable-hdpi/
        icono.png
        fondo.png
```

Respecto a las configuraciones disponibles que pueden ajustarse dentro del ya mencionado <config_qualifier>, las posibilidades son amplísimas. La anchura, el idioma, el tamaño de la pantalla, la orientación o la densidad en píxeles, son sólo alguno de los campos existentes, y dentro de cada uno de ellos, hay diferentes opciones.

Es muy importante conocer las restricciones, en caso de utilizar más de una configuración. El orden de los elementos en el nombre debe ser el correcto, puesto que de no ser así no serían tenidos en cuenta (siguiendo una tabla/lista que se puede consultar en la web de Android). Tampoco se permiten anidar las carpetas. Todas deben ir en res/. Las mayúsculas no son tenidas en cuenta, puesto que se transforma todo a minúsculas al compilar. Por supuesto, no se puede incorporar más de un valor por cada tipo.

Ejemplos no válidos:

```
drawable -hdpi-port / (orden)
res/ drawable /drawable-en/ (anidar)
drawable-rES-rFR/ (sólo un elemento por tipo)
```

Sus respectivos ejemplos válidos:

```
drawable-port-hdpi/
res/drawable-en/
drawable-rES /
```

Cuando existe un recurso que se quiere utilizar en más de una configuración (sin ser el "por defecto"), Android permite no tener que duplicarlo. Para ello, es preciso asignarle un nombre que lo diferencie del resto, e introducirlo en la carpeta por defecto. Esto se puede hacer con los tipos "layout", "drawable" y "strings y otros valores simples".

¿CÓMO ENCUENTRA ANDROID EL RECURSO MÁS APROPIADO PARA CADA DISPOSITIVO?

Android selecciona qué recurso utilizar durante la ejecución de una aplicación, obteniendo la configuración del dispositivo en que se realiza, y comparándola con los recursos disponibles en el proyecto, siguiendo este algoritmo:

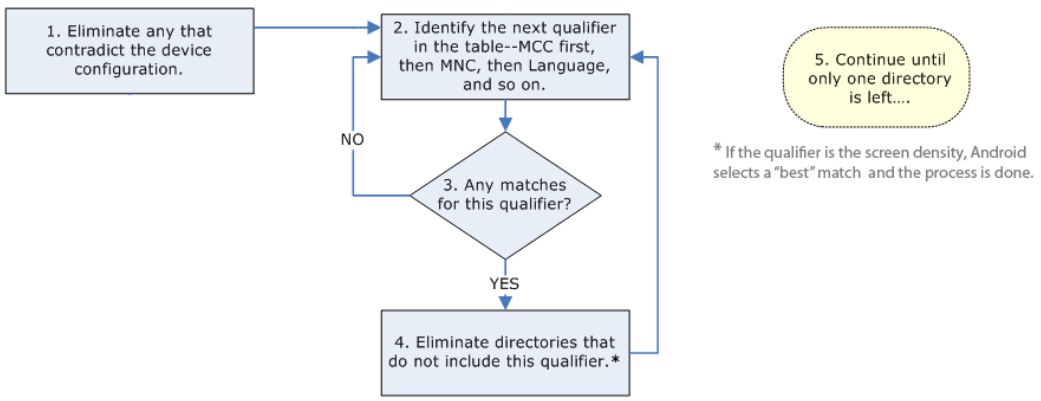


Figura 1. Algoritmo de búsqueda de recursos

1. Eliminar las carpetas con recursos contradictorios a la configuración del dispositivo.
2. Coger el siguiente calificador con precedencia más alta, en la tabla que podemos encontrar en la Web de Android, mencionada anteriormente
3. ¿Está ese recurso en algún directorio?
    - Si no es así, volver al paso 2. - Si está, seguir con el paso 4.

4. Eliminar los directorios que no contengan ese calificador.
5. Repetir los pasos 2, 3 y 4 hasta que sólo quede 1 directorio.

Además de este algoritmo, el sistema, más adelante, optimiza algunos aspectos, siempre en función de la configuración del dispositivo.

UTILIZANDO LOS RECURSOS

Una vez que se han creado todos los recursos, se puede referir a ellos a través de su identificador. Todos ellos están definidos en la clase R del proyecto. Ésta, se genera automáticamente al compilar, y contiene identificadores únicos, compuestos siempre por un tipo y un nombre, para cada recurso existente en el directorio res/. Hay 2 formas de acceder a ellos:

- A través del código, como una subclase de R. Se puede usar un recurso pasando su identificador como parámetro de una función, siguiendo el patrón

  ```
  [<package_name>.]R.<resource_type>.<resource_name>
  ```

  que representan el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), la subclase de R con el tipo y el nombre del recurso sin su extensión, respectivamente. Este método se puede utilizar para establecer una imagen de fondo, un layout para una pantalla o establecer el título de una actividad. Por ejemplo:

  ```
  getWindow().setBackgroundDrawableResource(R.drawable.fondo) ;
  ```

- Desde XML, usando la sintaxis correspondiente a la configuración del identificador del recurso. Se pueden definir elementos o atributos de un fichero XML como referencia a ellos. La sintaxis sigue el patrón

  ```
  @[<package_name>:]<resource_type>/<resource_name>
  ```

  que representan el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), la subclase de R con el tipo y el nombre del recurso sin su extensión, respectivamente. Este método se puede utilizar para establecer el color de un texto o qué debe mostrar, o incluso para crear alias. Por ejemplo:

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <EditText
  xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:textColor="@color/azul"
      android:text="@string/hola" />
  ```

Con esta última opción, se puede establecer un atributo como referencia a un estilo perteneciente al tema que está en uso, incluyendo, por ejemplo,

```
android:textColor="?android:colorSecundario".
```

LOCALIZACIÓN

Android está a disposición del usuario en muchas regiones y dispositivos diferentes. Lo ideal, es que las aplicaciones utilicen los textos, la moneda, los números… de acuerdo a donde se estén ejecutando. Todo eso se controla incorporando la mayor parte del contenido de la interfaz de usuario a los directorios de recursos (como hemos visto anteriormente), y manejando el comportamiento de la interfaz desde el código Java. Siempre que una aplicación se ejecute de forma "local", y no haya definidos un texto específico para ella, Android cargará el "string.xml" por defecto. Sin embargo, si no existe ese archivo, se mostrará un error y la aplicación no se ejecutará.

La prioridad de los recursos en función de la localización es absoluta. Por ejemplo, si nos encontramos en Francia, y tenemos un dispositivo que soporta alta calidad de gráficos, dará prioridad al directorio res/drawable-fr/, independientemente de si en su interior los recursos definidos son para dispositivos que soportan exclusivamente gráficos de baja calidad, frente a un directorio denominado res/drawable-hdpi/, que inicialmente parecería más adecuado.

Se debe saber que los calificadores MCC y MCN son una excepción y son siempre prioritarios, por lo que los recursos contenidos en semejantes carpetas serán cargados por delante de cualquier otro. En ocasiones es preciso crear un layout flexible, de tal modo que se adecúe a un contexto, pero que no se limite a él. Por ejemplo, si el entorno local es España, la agenda de contactos es preciso que tenga campos para nombre y 2 apellidos, sin embargo, si el entorno es Irlanda, sobraría uno de los huecos para los apellidos. Pues bien, la única solución para llevar a cabo esta situación no es duplicar los recursos, puesto que también se permite crear un único layout en el que un campo se active o desactive en función del idioma. Es decir, evaluando una condición sobre la configuración del dispositivo, y se recomienda esta opción por encima de crear más recursos. Se puede utilizar Android para buscar recursos locales, a través de la siguiente instrucción:

```
String locale =
context.getResources().getConfiguration().locale.getDisplayName();
```

PROBAR APLICACIONES LOCALIZADAS

Para probar cómo funcionarían aplicaciones en otra localización, se puede cambiar la configuración del emulador con adb shell, siguiendo éstos pasos:

1- Elegir qué región se quiere simular y determinar su lenguaje y códigos de región, por ejemplo fr de francés y CA de Canadá.

2- Lanza el emulador.

3- Ejecuta mediante comandos, desde el ordenador que lanza el emulador, "abd Shell", o "adb –e Shell", si existe un dispositivo adjunto.

4- Una vez dentro, se debe ejecutar este comando:

```
setprop persist.sys.language [language code];setprop
persist.sys.country [country code];stop;sleep 5;start
```

reemplazando los corchetes por los códigos apropiados (fr y Ca respectivamente, para este ejemplo).

Estos pasos reiniciarán el emulador, y cuando se vuelva a ejecutar la aplicación, se hará sobre la nueva configuración. De hecho, cuando esté lista, se pueden publicar en el Market para diferentes localizaciones, ya sea con diferentes apk's, o incluyendo todos los recursos en la misma.

¿ESTÁN TODOS LOS RECURSOS BÁSICOS NECESARIOS EN LA APLICACIÓN?

Una forma sencilla de comprobar si la aplicación tiene los recursos necesarios para ejecutarse, independientemente de la configuración consiste en establecer el emulador en una localización para la que la aplicación no tenga recursos definidos. Si al ejecutarla nos sale un mensaje de error, es que faltan los recursos por defecto, o no están bien definidos.

TIPOS DE RECURSOS

En Android existen varios tipos de recursos. A lo largo de este manual ya se han ido viendo, más o menos detalladamente, algunos de ellos.

MENU

Este recurso define al Menú Principal, Menú Contextual y Submenú.

- **Localización**: res/menú/nombre_del_archivo.xml
- **Referencia**: R.menu.nombre_del_archivo (en JAVA) y [package:]menú.nombre_del_fichero (en XML).
- **Sintaxis** (con todas las opciones):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@[+][package:]id/resource_name"
          android:title="string"
          android:titleCondensed="string"
          android:icon="@[package:]drawable/drawable_resource_name"
          android:onClick="method name"
          android:showAsAction=["ifRoom" | "never" | "withText" |
"always" | "collapseActionView"]
          android:actionLayout="@[package:]layout/layout_resource_name
"
          android:actionViewClass="class name"
          android:actionProviderClass="class name"
          android:alphabeticShortcut="string"
          android:numericShortcut="string"
          android:checkable=["true" | "false"]
          android:visible=["true" | "false"]
          android:enabled=["true" | "false"]
          android:menuCategory=["container" | "system" | "secondary" |
"alternative"]
          android:orderInCategory="integer" />
    <group android:id="@[+][package:]id/resource name"
           android:checkableBehavior=["none" | "all" | "single"]
           android:visible=["true" | "false"]
           android:enabled=["true" | "false"]
           android:menuCategory=["container" | "system" | "secondary"
| "alternative"]
           android:orderInCategory="integer" >
        <item />
    </group>
    <item >
        <menu>
          <item />
        </menu>
    </item>
</menu>
```

- **Elementos**:

  1- <menu>: Requerido. Tiene que ser el nodo raíz e incluir el atributo xmlns:android obligatoriamente.

  2- <ítem>: Tiene que ser el hijo de un elemento de tipo <menu> o <grupo>. Si además contiene un elemento del 1er tipo, existirá un Submenú.

  3- <group>: Para crear una colección de elementos (de tipo <item>. Los contiene), que pueden compartir unas características. Es necesario que sea hijo de un elemento <menú>.

RESTO DE RECURSOS

El recurso "Menu" es un ejemplo, escogido en este manual por su importancia y por haberse visto parcialmente con anterioridad, pero todos los demás tienen un comportamiento análogo. Aquí se detallan más brevemente:

- **Animation**: Define animaciones predeterminadas. Las denominadas "Tween" se guardan en res/anim/ y son accedidas mediante la clase R.anim. Las "Frame" se almacenan en res/drawable/ y se accede a ellas desde R.drawable.
- **Color State List:** Define unos colores que cambian basándose en la "View". Se guardan en res/color/ y se accede a ellos desde la clase R.color.
- **Drawable**: Define diferentes tipos de gráficos con Bitmaps o con XML. Son accedidos a través de R.drawable y almacenados en res/drawable/.
- **Layout**: Definen la estructura general de la interfaz de usuario. Se guardan en res/layout/ y se accede a ellos desde la clase R.layout.
- **String**: Define strings, arrays de strings y plurales, incluyendo formato y estilo de ellos. Se almacenan en res/values/ y se pueden acceder desde las clases R.string, R.array y R.plurals respectivamente.
- **Style**: Define el aspecto y formato de los elementos de la interfaz de usuario. Se guardan en res/values/ y se accede a ellos desde la clase R.values.
- **Otros**: Definen valores tales como booleanos, enteros, dimensiones, colores u otros arrays. Todos ellos se almacenan en res/values/, pero cada uno se accede desde subclases de R únicas, tales como R.bool, R.integer, R.dimen, etcétera.

Android permite definir recursos que se ajusten a diferentes configuraciones en diferentes dispositivos. A la hora de ejecutar una aplicación, sólo se utilizarán los adecuados y necesarios en función de cada situación. El resultado de ello es una optimización en la relación esfuerzo-calidad, para cada dispositivo.

5. DATOS EN ANDROID. IDEAS PRINCIPALES

Manuel Báez – Jorge Cordero – Miguel González

INTRODUCCIÓN

Hay cuatro maneras de almacenar los datos en Android, usaremos una u otra dependiendo de la función que tengan dichos datos. Es importante destacar que los datos de cada aplicación son privados a dicha aplicación, pero tenemos la posibilidad de compartirlos si así lo deseamos, como veremos más adelante.

Tenemos cuatro tipos de datos distintos, en función de lo que queramos almacenar. Dichos tipos son:

- *Preferencias*: Conjunto de datos, clave/valor
- *Archivos locales: Almacenamiento interno y externo (SD)*
- *Base de datos*: SQLite
- *Proveedores de contenidos*: Único mecanismo para compartir datos, no se trata con ellos directamente.

PREFERENCIAS COMPARTIDAS

Tenemos a nuestra disposición un tipo de datos llamado *preferencias*. Estas *preferencias*, son una forma ágil para poder guardar datos simples de la aplicación. Estos datos son un conjunto clave/valor que perdura después de que la aplicación se cierre, por lo que es principalmente útil para guardar, como su nombre indica, las preferencias de una aplicación.

Por ejemplo guardar el idioma, configuraciones de sonido de la aplicación, fuentes, colores, etc. Los datos que permite guardar son primitivos y se invocan con los siguientes métodos putInt(), putBoolean(), putFloat(), putLong(), putString(), putStringSet() de la forma (String key, tipo value).

SharedPreferences

Ahora vamos a ver cómo guardar los datos de las preferencias y cómo acceder a ellos al inicializar la aplicación.

Para ello utilizaremos una clase que hereda de PreferenceActivity, la cual guarda en un xml las preferencias de la aplicación que, en principio, serán cargadas cuando la iniciemos y guardadas cuando la cerremos. Dicho xml se guarda en SharedPreferences. Es importante destacar que podemos tener una colección de preferencias, para ello tendremos que asociarles un identificador. Almacenándose los datos en:

/data/data/nombre_paquete/shared_prefs/preferencias_por_defecto.xml

Vamos a ver primero cómo guardar los datos. Tenemos a nuestra disposición para ello los métodos onSaveInstance() o onStop(), éste último será el que usemos para el ejemplo. Ambos se invocan antes de cerrar la aplicación, siempre y cuando se cierre de manera correcta.

Para guardar los datos seguiremos el siguiente patrón:

1. Llamamos a edit(void) para editar las preferencias con SharedPreferences.Editor.
2. Añadimos valores con los métodos put anteriormente nombrados.
3. Actualizamos los nuevos valores usando commit(void)

Código Android

```
protected void onStop(){
    super.onStop();
     // Necesitamos un objeto Editor para poder modificar las preferencias
    SharedPreferences preferencias = getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = preferencias.edit();
    editor.putString("NumeroAlAzar", "58472");
    // Por ejemplo un número al azar
    // Actualizamos las nuevas preferencias
    editor.commit();
}
```

Ahora que tenemos las preferencias guardadas vamos a ver cómo podemos cargarlas.

Código Android

```
public class PreferenciasActivity extends PreferenceActivity implements
OnSharedPreferenceChangeListener {
//...     Tu código
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
              // Carga las preferencias del XML.
        addPreferencesFromResource(R.xml.preferencias);
getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }
//...     Tu código
}
```

Preference Activity

Por ahora que sabemos modificar las preferencias en código, vamos a ver cómo hacer nuestras preferencias en xml. Dichos xml se almacenan en /res/xml, y pueden constar de los siguientes tipos de opciones:

1. CheckBoxPreference: Check, valor booleano.
2. EditTextPreference: Cadena de texto.
3. ListPreference: Lista de selección única.
4. MultiSelectListPreference: Lista de selección múltiple.

Código xml

```
<PreferenceScreen  xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory>
            TU CÓDIGO DE ESTA CATEGORÍA
        </PreferenceCategory>
        <PreferenceCategory>
            TU CÓDIGO DE ESTA CATEGORÍA
        </PreferenceCategory>
</PreferenceScreen>
```

Cuando hayamos definido nuestras preferencias, tendremos que implementar una nueva actividad, para ello tendremos que crear una clase que extienda de PreferenceActivity de la siguiente forma

Código Android

```
public class Opciones extends PreferenceActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.opciones);
   }
}
```

También tendremos que añadirla al Manifest como actividad, para ello bastará con añadir el siguiente código en el Manifest.

Código Android

```
<activity android:name=".PantallaOpciones"
    android:label="@string/app_name">
</activity>
```

ARCHIVOS LOCALES

A la hora de guardar distintos tipos de datos, que no se ajusten al patrón clave/valor de las preferencias, tendremos que guardarlos como *archivos locales*, dichos archivos, por defecto, no son accesibles desde otras aplicaciones, regulado por permisos Unix. El acceso es parecido al de Java estándar, debemos crear tanto input stream como output stream. Puede soportar solamente archivos creados en la carpeta de la aplicación. Por lo que necesitamos saber dónde están almacenadas las aplicaciones. Están almacenadas en:

/data/data/nombre_paquete/files/nombre_fichero,

pudiendo en las últimas versiones almacenarlo en la SDCard. Es importante ser conscientes de que algunos dispositivos tienen una memoria interna limitada, por lo que no deberíamos de abusar de este espacio guardando archivos de gran tamaño.

Memoria interna.

Para empezar vamos a ver cómo crear los output stream y los input stream, hay que destacar que en el output stream tenemos que seleccionar el modo de acceso, que puede ser:

1. MODE_PRIVATE(por defecto) acceso privado desde nuestra aplicación.
2. MODE_APPEND para poder añadir datos a un fichero existente.
3. MODE_WORLD_READABLE para permitir lectura a otras aplicaciones.
4. MODE_WORLD_WRITABLE para permitir escritura a otras aplicaciones.

A continuación veremos cómo crear los output:

Código Android

```
try
{
   OutputStreamWriter fOut=
      new OutputStreamWriter(
         openFileOutput("texto_ejemplo.txt", Context.MODE_PRIVATE));

   fOut.write("Cualquier cosa");     // Escribimos “Cualquier cosa” en fOut
```

```
    fOut.close();                    // Cerramos la escritura en el fichero fOut
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al escribir el fichero.");
}
```

Ahora para los input hay que tener en cuenta que solamente podremos leer aquellos ficheros para los que tengamos permisos, el código viene detallado a continuación.

Código Android

```
try
{
    BufferedReader fIn =
        new BufferedReader(
            new InputStreamReader(
                openFileInput("texto_ejemplo2.txt")));

    String texto = fIn.readLine();     // Cogemos la línea de fIn
    fin.close();     // Salimos para guardar la línea
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al leer el fichero");
}
```

RECURSOS

Otra forma de almacenar datos en la memoria interna es almacenarlo como recurso en la carpeta "/res/raw" de nuestro proyecto, en eclipse dicha carpeta no se crea por defecto, tendremos que crearla. Es importante destacar que este método de almacenamiento es útil para ficheros que no vamos a modificar, ya que posteriormente serán de solo lectura. La forma de acceder a dichos archivos viene detallada a continuación.

Código Android

```
try
{
   InputStream fRaw =
      getResources().openRawResource(R.raw.ejemplo_raw);
      //ejemplo_raw es el nombre del fichero
    BufferedReader bRIn =
      new BufferedReader(new InputStreamReader(fRaw));
    String linea = bRIn.readLine();
    fRaw.close();
}
catch (Exception e)
{
   Log.e("Error de fichero", "Error al leer fichero desde recurso");
}
```

MEMORIA EXTERNA

La última alternativa para almacenar archivos locales será la de guardarlos en la memoria externa, que generalmente será una tarjeta de memoria SD. Es importante destacar que en el caso de la memoria externa, y al contrario que con la memoria interna, puede no estar disponible ya sea porque no está presente o porque el dispositivo no la reconoce. Para ello tendremos que cercionarnos de que está disponible antes de utilizarla como sistema de almacenamiento, con ese fin usaremos el método getExternalStorageStatus() que nos devuelve el estado de la memoria externa. Según queramos leer o escribir en ella los estados que nos interesarán son MEDIA_MOUNTED que nos dice que podemos escribir y leer en ella o MEDIA_MOUNTED_READ_ONLY que nos dice que está disponible con permisos de solo lectura.

El siguiente fragmento de código nos muestra una forma sencilla de verificar el estado de la memoria externa y almacenarlo en dos booleanos.

Código Android

```
boolean tarjetaDisponible = false;
boolean tarjetaEscritura = false;

//Comprobamos el estado de la memoria externa (tarjeta SD)
String estadoTarjeta = Environment.getExternalStorageState();

if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED))
{
   // Tarjeta disponible y habilitada para escritura
   tarjetaDisponible = true;
   tarjetaEscritura = true;
}
else if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
{
   // Tarjeta disponible y deshabilitada para escritura
   tarjetaDisponible = true;
   tarjetaEscritura = false;
}
else
{
```

```
    // Tarjeta NO disponible
    tarjetaDisponible = false;
    tarjetaEscritura = false;
}
```

También tendremos que indicar en el manifest que nuestra aplicación necesita permisos para almacenar datos en memoria externa, para ello pondremos:

Código xml

```
<uses-permission android.permission.WRITE_EXTERNAL_STORAGE>
</uses-permission>
```

Ahora que sabemos el estado de la memoria externa y tenemos permisos vamos a ver la ruta absoluta de la tarjeta de memoria. Para ello usaremos el método getExternalStorageDirectory() que nos devolverá un File, ahora ya podremos crear un fichero en la dirección de dicho File como mostramos a continuación.

Código Android

```
try
{
    File ruta_tarjeta = Environment.getExternalStorageDirectory();
    File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
     OutputStreamWriter fOut =
        new OutputStreamWriter(
            new FileOutputStream(f));
     fOut.write("Smile! :D.");
    fOut.close();
}
catch (Exception e)
{
    Log.e("Error de ficheros", "Error al escribir fichero en la tarjeta.");
}
```

Para leer de un fichero el código es muy similar a la escritura. Mostramos el código a continuación:

Código Android

```
try
{
   File ruta_tarjeta = Environment.getExternalStorageDirectory();


   File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
   BufferedReader fIn =
      new BufferedReader(
         new InputStreamReader(
            new FileInputStream(f)));
   String linea = fIn.readLine();
   fIn.close();
}
catch (Exception e)
{
   Log.e("Error de ficheros", "Error al leer fichero de la tarjeta.");
}
```

BASES DE DATOS: SQLITE

Android tiene integrado en el propio sistema una API completa que nos permite manejar BBDD en SQLite. SQLite es un motor de bases de datos que se ha ido popularizando en los últimos años dado que maneja archivos de poco tamaño, no necesita ejecutarse en un servidor, cumple el estándar SQL-92 y, además, es de código libre.

El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un único fichero en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción. Esta es, al mismo tiempo, su gran virtud y su mayor inconveniente, ya que gracias a ello dispone de unas latencias muy bajas, pero también impide el acceso múltiple a la base de datos.

Centrándonos en el lenguaje que abordamos en este tutorial, la manera más fácil de acceder a la información de una base de datos local es creando una clase que herede de SQLiteOpenHelper sobre la cual tendremos que adaptar/sobreescribir los métodos proporcionados para obtener la

funcionalidad con la base de datos deseada. Básicamente en esta clase definimos los atributos de nuestra base de datos y el comportamiento ante creación y actualización de la misma. Los métodos que deberemos sobrescribir serán onCreate() y onUpgrade(), además de la constructora, donde podremos incluir todo lo que creamos necesario.

A modo de ejemplo, vamos a crear una base de datos de una entidad Domicilio que incluirá como atributos: calle, ciudad, CP y número. Así pues crearíamos la clase DomicilioSQLiteHelper:

Código Android

```
public class DomicilioSQLiteHelper extends SQLiteOpenHelper {


    //Aqui creamos el String que creará la base de datos en el onCreate
    String creaBD= "CREATE TABLE Domicilio (calle TEXT, ciudad TEXT, CP INTEGER, numero INTEGER)";


    public DomicilioSQLiteHelper(Context context, String nombre,
                                 CursorFactory factory, int version) {
       super(context, nombre, factory, version);
    }


    @Override
    public void onCreate(SQLiteDatabase db) {
       //Aqui se crea la BD, se llamaría solo cuando no exista
       db.execSQL(sqlCreate);
    }


    @Override
    public void onUpgrade(SQLiteDatabase db, int a, int b) {
       /*Utilizamos como opción de actualización la de borrado de la tabla anterior, para luego crearla vacia con la nueva versión*/
       db.execSQL("DROP TABLE IF EXISTS Domicilio"); //Borrado anterior
       db.execSQL(sqlCreate); //Creación nueva
    }
}
```

Cabe mencionar cuando se usarán los dos métodos anteriores, el método onCreate() se ejecutará de maneara automática cuando se necesite crear la base de datos, lo que se reduce a cualquier llamada o referencia a la base de datos mientras esta todavía no exista. Y el método onUpgrade() que también será llamado automáticamente cuando se produzca un cambio en la estructura de la base de datos, como podría ser la inclusión/eliminación de un campo en una tabla, en el código de ejemplo anterior hemos simplemente borrado la tabla anterior y creado una tabla nueva vacía, sin embargo, en la mayoría de los caso es necesario migrar los datos del modelo anterior al nuevo.

Teniendo nuestra clase heredada SQLiteOpenHelper, estamos en disposición de utilizar los métodos getReadableDatabase() o getWritableDatabase() para acceder a la base de datos en modo lectura o lectura/escritura. Veamos un ejemplo simple en el que abramos la base de datos para escritura y creemos un par de registros:

Código Android

```
Int version = 1;
String nombreDB = "DomiciliosDB";
/****** Nuestro código anterior aquí *******/
    DomicilioSQLiteHelper domicilioHelper =
      new UsuariosSQLiteHelper(this, nombreDB, null, version);

    // Abrimos la Base de datos, en casa de que no existiera, se crearía ahora
    SQLiteDatabase db = domicilioHelper.getWritableDatabase();

      //Insertamos 2 domicilios para inicializar la tabla
      db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
            VALUES ('C/Mayor', 'Madrid', 28001, 13)");
      db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
            VALUES ('Avda. Grande', 'Ibiza', 26050, 45)");

      //Cerramos la base de datos ya actualizada
      db.close();

/****** Resto de código *******/
```

Pese a haber aparecido antes, no se ha comentado el uso del método execSQL(String query). Como el lector habrá podido imaginar, este método recibe Strings de los query que se pueden ejecutar sobre la base de datos y los ejecuta. La intención de este tutorial, no es ni mucho menos enseñar la utilización de bases de datos con el estandar de SQLite, para obtener información sobre él por favor, visite la página oficial de SQLite y encontrará información sobre su sintaxis:

http://www.sqlite.org/lang.html

A pesar de lo citado anteriormente, para no quedarnos sólo con estas breves pinceladas sobre la sintaxis de SQLite, estas son las funcionalidades más comunes de una base de datos en modo escritura:

- Insertar un registro

```
db.execSQL("INSERT INTO Usuarios (usuario,email) VALUES ('usu1','usu1@email.com') ");
```

- Eliminar un registro

```
db.execSQL("DELETE FROM Usuarios WHERE usuario='usu1' ");
```

- Actualizar un registro

```
db.execSQL("UPDATE Usuarios SET email='nuevo@email.com' WHERE usuario='usu1' ");
```

Ahora vamos a crear otro código en el que mostremos las funcionalidades de los cursores. Los cursores simplemente nos servirán para recorrer el resultado de una base de datos como si un iterador se tratase. Tratamos de ejecutar algunas querys sobre la clase domicilio, para ello, los permisos con los que abriremos la base de datos de los domicilios serán sólo de lectura.

Código Android

```
Int version = 1;
String nombreDB = "DomiciliosDB";
/****** Nuestro código anterior aquí *******/
DomicilioSQLiteHelper domicilioHelper = new UsuariosSQLiteHelper(this, nombreDB, null, version);
// Abrimos la Base de datos, en casa de que no existiera, se crearía ahora
SQLiteDatabase db = domicilioHelper.getReadableDatabase();
Cursor c = db.rawQuery("SELECT ciudad, CP FROM Domicilio WHERE calle= "C/Mayor");
//Obtenemos los indices de las columnas
int ciudadIndex = mCursor.getColumnIndexOrThrow("ciudad");
int CPIndex = mCursor.getColumnIndexOrThrow("CP");
```

```
// Posicionamos el cursor al principio de la lista
if (c.moveToFirst()) {
    do {
    //Obtenemos en nuestras variables los datos del registro que está leyendo.
        String ciudad= c.getString(ciudadIndex);
        int CP = c.getString(CPIndex);
    } while(c.moveToNext());
    /*Lo seguimos adelantando mientras tengamos registros que leer,
    Aunque parezca extraño, por la falta de uso, es muy común utilizar aqui la
    estrucutra DO- WHILE*/
/****** Resto de código *******/
```

CONTENT PROVIDERS

Los Content Providers son el mecanismo que tiene Android para comunicar datos entre distintas aplicaciones. Funcionalmente suelen ser muy parecidos a las bases de datos en SQLite que hemos comentado en el punto anterior. El propio Android trae (desde sus versiones 2.0+) incluidos la agenda ,los SMS y el listado de llamadas mediante el método de los Content Providers, simplemente una aplicación debe acceder a la URI donde tenemos el Content Provider y una vez tengamos acceso pedirle los datos que necesitemos.

Al igual que para las bases de SQLite para los Content Providers deberemos crear una clase que herede de ContentProvider y declararla en el manifest.

AndroidManifest.xml

```
<application>....
<provider                                    android:name="DomiciliosProvider"
   android:authorities="com.pruebasandroid.ejemplo" />
 <application/>
```

La facilidad de su uso reside en tener una base de datos que realice las mismas funciones que los métodos que tenemos que sobreescribir, para llamarlos basta con hacer un buen uso de las URIs. A continuación se muestra el código, a modo de ejemplo, de DomiciliosProvider:

Código Android

```
package com.pruebasandroid.ejemplo;
…
public class DomiciliosProvider extends ContentProvider {
      //Establecemos como una constante la URI para aeder a estos datos
   private static final String uri = "content://com.pruebasandroid.ejemplo/domicilios";
      //Uri del String anterior
   public static final Uri CONTENT_URI = Uri.parse(uri);
      //Necesario para UriMatcher
   private static final int DOMICILIOS = 1;
   private static final int DOMICILIOS_ID = 2;
   private static final UriMatcher uriMatcher;
      //Clase interna para declarar las constantes de columna
   public static final class Clientes implements BaseColumns
   {
      private Clientes() {}
            // Inicializamos las columas, para poder acceder a ellas con un mnemotécnico
      public static final String COL_CALLE = "calle";
      public static final String COL_CP = "cp";
      public static final String COL_NUMERO = "numero";
      public static final String COL_CIUDAD = "ciudad";
   }
      //Los datos referentes a la base de datos, que creamos en el ejemplo anterior
   private UsuariosSQLiteHelper helper;
   private static final String NOMBRE_BD = "DomicilioDB";
   private static final int VERSION = 1;
   private static final String TABLA_DOMICILIOS = "Domicilios";
      //Inicializamos el UriMatcher
   static {
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI("com.pruebasandroid.ejemplo", "domicilios",     DOMICILIOS);
```

```
    uriMatcher.addURI("com.pruebasandroid.ejemplo", "domicilios/#",
DOMICILIOS_ID);
  }
  /*Sobreescribimos los métodos de ContentProvider*/
  @Override
  public boolean onCreate() {
    helper = new UsuariosSQLiteHelper(
                                this.getContext(), NOMBRE_DB, null, VERSION);
    return true;
  }
  /*Aqui devolvemos el cursor de nuestra BD de domicilios*/
  @Override
  public Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs, String sortOrder) {

    /*Construimos un where si se refiere a un id concreto */
    String where = selection;
    if(uriMatcher.match(uri) == DOMICILIOS_ID){
        where = "_id=" + uri.getLastPathSegment();
    }
    /*Escritura y lectura y devolvemos el cursor de nuestra base de datos*/
    SQLiteDatabase db = helper.getWritableDatabase();

    return (Cursor)db.query(TABLA_DOMICILIOS, projection, where,
                        selectionArgs, null, null, sortOrder);
  }
  @Override
  public Uri insert(Uri uri, ContentValues values) {
    /*para escribir, insertamos en la base de datos que nos proporciona*/
    long id = 1;
    SQLiteDatabase db = helper.getWritableDatabase()
```

```
    id = db.insert(TABLA_DOMICILIOS, null, values);


    /*Devolvemos su correspondiente Uri*/
    Uri newUri = ContentUris.withAppendedId(CONTENT_URI, id);
    return newUri;
  }
  @Override
  public int update(Uri uri, ContentValues values,
                    String selection, String[] selectionArgs) {
          /* Aqui el código para el update*/
  }
  @Override
  public int delete(Uri uri, String selection, String[] selectionArgs) {
     /* Aqui el código para el delete*/
  }

  @Override
  public String getType(Uri uri) {
    int tipo = uriMatcher.match(uri);
    switch (tipo)
    {
        case DOMICLIOS:
                return "vnd.android.cursor.dir/vnd.pruebasandroid.domicilio";
        case DOMICILIOS_ID:
                return "vnd.android.cursor.item/vnd.pruebasandroid.domicilio"";
        default:
                return null;
    }
  }
}
```

DATOS EN RED

A pesar de lo que hemos comentado sobre SQLite anteriormente, muchas son las aplicaciones que necesitan la integración con una base de datos existente en red como un MySQL. La diferencia principal respecto a los datos que podríamos manejar con SQLite reside en que estos datos se guardaban en un archivo de forma local mientras que con las BBDD que podemos manejar a través de la red tenemos cambios de otros usuarios de manera inmediata.

La mejor forma de conectar nuestro terminal con una base de datos de manera remota es mediante el uso de un servicio web. Para nuestro tutorial, explicaremos como crear de manera sencilla un servicio web en php que nos proporcione datos desde una base de datos MySQL y como interactuar desde Android con ese servicio. Este tipo de servicio es simplemente un ejemplo, si el lector conoce o prefiere utilizar cualquier otro tipo de lenguaje en la creación del servicio web es una opción totalmente válida, pero a modo de seguir acorde con el nivel del tutorial lo crearemos en php ya que es un lenguaje muy sencillo para el manejo de bases de datos que esperemos sea de gusto del lector.

A modo de ejemplo vamos a crear una base de datos que tenga una única tabla que contenga los datos de una persona. A cada persona le vamos a dar los atributos DNI, nombre, sexo y fecha de nacimiento. Entramos en la consola de SQL de nuestro servidor y escribimos:

Código SQL.

```
CREATE TABLE `nuestraBD`.`persona` (
`DNI` VARCHAR( 10 ) NOT NULL ,
`nombre` VARCHAR( 30 ) NOT NULL ,
`sexo` VARCHAR( 10 ) NOT NULL ,
`fechaNacimiento` DATE NOT NULL
)
```

Hasta aquí tendríamos creada ya en la base de datos la entidad persona, vamos ahora a introducir 3 registros:

Código SQL.

```
INSERT INTO `nuestraBD`.`persona` (
`DNI` ,
`nombre` ,
`sexo` ,
```

```
`fechaNacimiento`
)
VALUES (
'000000000X', 'Jorge', 'Hombre', '1980-11-20'
), (
'11111111Y', 'Francisco', 'Hombre', '1977-09-10'
), (
'222222222Z', '', 'Mujer', '1987-01-15'
);
```

Ahora con nuestra base de datos terminada, vamos a crear uns servicio web obtenga todos los datos de la tabla. Obviamente el servicio web aceptará cualquier sintaxis y restricción que le pidamos como al propio SQL (Where, Grouopby, limit ,sort....)

Código PHP.

```
<?php
mysql_connect("host","username","password");
mysql_select_db("nuestraBD");
$pregunta = “select * from persona”
$sql=mysql_query($pregunta);
while($row=mysql_fetch_assoc($sql))
        $output[]=$row;
print(json_encode($output));
mysql_close();
?>
```

Publicamos en la web el código anterior y lo renombramos a “personas.php”. Recordar que los datos de la conexión especificados por "host","username","password", deberán corresponder con los de la base de datos que el usuario requiera para su aplicación. En el código anterior, simplemente hemos lanzado una query que nos devuelva todos los datos de la tabla persona y después hemos ido creando un array que contenga cada entrada a la propia tabla. Como se puede observar, al final siempre cerramos la conexión, pero antes de ello hemos escrito print(json_encode($output)); para poder obtener los datos como salida en el estandar JSON.

JSON (JavaScript Object Notation) es un lenguaje de etiquetas como XML que se utiliza como sustituto para intercambio de contenidos dada su sencillez permite crear parsers mucho más sencillos que en XML. La estructura que nos devolvería JSON para el ejemplo anterior sería:

Respuesta JSON

```
[{"DNI":"000000000X","nombre":"Jorge","sexo":"Hombre","fechaNacimiento":"1980-11-20"},
{"DNI":"11111111Y","nombre":"Francisco","sexo":"Hombre","fechaNacimiento":"1977-09-10"},
{"DNI":"222222222Z","nombre":"Patricia","sexo":"Mujer","fechaNacimiento":"1987-01-15"}]
```

Después de todo esto, ahora sí volvemos a Android. Lo primero que vamos a hacer es introducir en el *Manifest* permisos para que la aplicación pueda acceder a Internet, de esta forma podremos concertar con nuestro servicio Web:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Ahora vamos a crear una función que consiga conectar con el servicio web y pueda enviar/recibir datos:

Código Android

```
public class Persona extends Activity {

@Override
public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.main);
try{
    HttpClient httpclient = new DefaultHttpClient();

    HttpPost httppost = new HttpPost("URL/persona.php");

    HttpResponse response = httpclient.execute(httppost);

    HttpEntity entity = response.getEntity();

    InputStream is = entity.getContent();

    }catch(Exception e){

       System.out.println("Error en la conexión");

       // En esta caputra de excepción podemos ver que hay un problema con la

       // conexión e intentarlo más adelante.

    }
try{
```

```
    BufferedReader reader = new BufferedReader(new InputStreamReader
  (is,"iso-8859-1"),8);


        StringBuilder  sb = new StringBuilder();
        sb.append(reader.readLine() + "\n");
        String line="0";
        while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
         }
         is.close();
         String cadena =sb.toString();
         }catch(Exception e){
              System.out.println("Error al obtener la cadena desde el buffer");
         }
//Creamos los atributos de nuestra clase persona
String DNI; String nombre; String sexo; String fechaNac;
try{
    JSONArray jsonArray = new JSONArray(cadena);
    JSONObject jsonObject=null;
    for(int i=0;i<jsonArray.length();i++){
         jsonObject= jsonArray .getJSONObject(i);
         DNI=jsonObject.getString("DNI");
         nombre=jsonObject.getString("nombre");
         sexo=jsonObject.getString("sexo");
         fechaNac=jsonObject.getString("fechaNacimiento");
      }
    }
    catch(JSONException e1){
          Toast.makeText(getBaseContext(), "No se encuentran los datos"
,Toast.LENGTH_LONG).show();
    } catch (ParseException e1) {
                        e1.printStackTrace();
         }
}
}
  }
```

Así obtendremos los valores en las variables DNI, nombre, sexo y fechaNac del contenido de la tabla que nos ha proporcionado el servicio Web. Con ello ya podremos usarlas en nuestra aplicación, mostrarlas en un ListView o todo aquello que veamos necesario.

Ahora bien, hasta lo que hemos visto, solo podemos pedirle al servicio Web datos pero ¿qué pasa si queremos introducir algún tipo de información? La respuesta es muy sencilla en php una de las cosas más comunes es pedir el usuario y la contraseña para acceder a la base de datos, o simplemente los datos a introducir en un *insert* o cualquier otro dato que necesitemos desde php. Para obtener datos externos desde PHP utilizamos $_REQUEST:

Código PHP

```
<?php
        $host =  $_REQUEST[host];
        $user =  $_REQUEST[user];
        $password = $_REQUEST[password];
        mysql_connect($host , $user, $password);
//    ….....Nuestro código intermedio aqui.....
        mysql_close();
?>
```

Para introducir estos datos desde Android también es muy sencillo, sirviéndonos como código el ejemplo anterior, sólo anotamos aquí lo que deberemos añadir al HttpPost:

Código Android

```
//Creamos un ArrayList de Nombre Valor
// e introducimos los datos que deseamos pasar al web service
ArrayList<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("host","url de nuestro host"));
nameValuePairs.add(new BasicNameValuePair("user","usuario de la BD"));
nameValuePairs.add(new BasicNameValuePair("password","contaseña"));
// Añadimos los valores al HttpPost
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost =  HttpPost("URL/persona.php");
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
```

6. MAPAS Y GPS

Álvaro Borrego – Francisco Hernández – David Palomero

USO DE MAPAS

Para poder usar los mapas, Android provee de una librería externa que se encuentra en el paquete ***com.google.maps***

¿COMO OBTENER LA API KEY PARA USAR GOOGLE MAPS?

Para poder acceder a los datos desde el MapView es necesario registrarse en el servicio de Google Maps y aceptar los términos de uso. Obtendremos una clave alfanumérica que nos dará acceso.

El registro para obtener la clave consta de dos partes:

1. Registrar la huella digital MD5 de la aplicación para que pueda acceder a los datos de Google Maps.
2. Agregar una referencia a la clave en cada MapView (en el XML o en código).

INFORMACIÓN GENERAL

Para asegurar que los aplicaciones utilizan los datos de manera adecuada, el MapView necesista una clave para poder usar la API. Esta clave es una secuencia alfanumérica que identifica la aplicación y el desarrollador. Sin esta clave, el MapView no podrá descargar los datos de los mapas.

Cada ApiKey de Google Maps es el único asociado a un certificado en concreto. Y cada MapView debe hacer referencia a un clave de la API (ApiKey).

Varias vistas Map puede referirse al mismo o distintos si se han registrado varios certificados a la misma aplicación.

¿CÓMO OBTENER LA HUELLA DIGITAL MD5 DEL CERTIFICADO?

Para generarla necesitamos usar la herramienta ***keytool*** del SDK de java. Los parámetros para el ***keytool*** se muestran en la siguiente tabla:

Tabla 1. Parámetros de Keytool

Opciones Keytool	Descripción
`-list`	Muestra la huella MD5 del certificado
`-keystore <keystore-name>.keystore`	El nombre del keystore que contiene la clave
`-storepass <password>`	Una clave para el keystore.
`-alias <alias_name>`	The alias for the key for which to generate the MD5 certificate fingerprint.
`-keypass <password>`	The password for the key. As a security precaution, do not include this option in your command line unless you are working at a secure computer. If not supplied, Keytool prompts you to enter the password. In this way, your password is not stored in your shell history.

La siguiente figura muestra un ejemplo de la obtención de la clave.

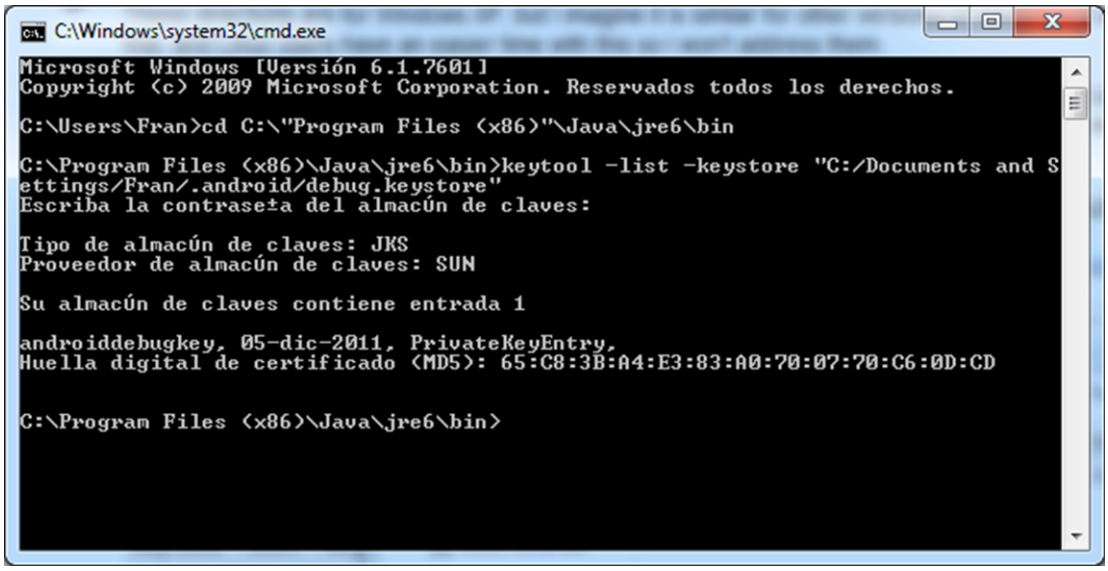


Figura 1. Ejemplo de la obtención de la clave

REGISTRAR LA HUELLA MD5 EN EL SERVICIO DE GOOGLE MAPS

Cuando la tengamos, accedemos a la siguiente dirección Web:

http://code.google. com/android/maps-api-signup.html

Una vez en la página, leemos las condiciones, pegamos la huella digital, y generamos la clave.

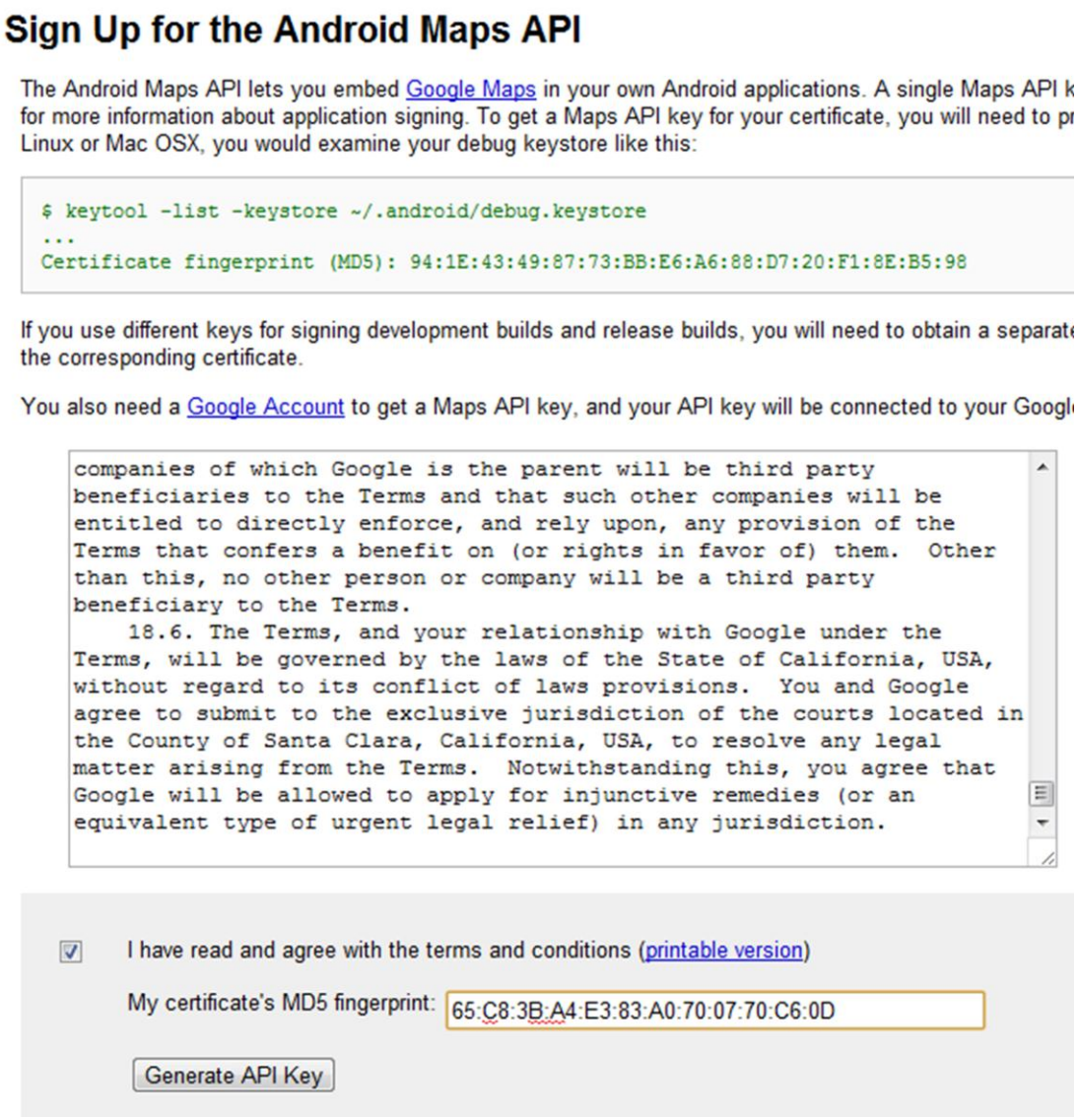


Figura 2. Registro huella MD5 en Google Maps

Figura 3. Api de Google Maps

AÑADIR LA CLAVE DEL API DE ANDROID MAPS A NUESTRA APLICACIÓN Y AÑADIRLA AL LAYOUT

Al generar nuestra clave, ya hemos visto como añadirla desde el XML del layout:

```
<com.google.android.maps.MapView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="example_Maps_ApiKey_String"
 />
```

Pero si quisiéramos añadirla desde una actividad, el código sería el siguiente:

```
MapView mMapView = new MapView(this, "example_Maps_ApiKey_String");
```

PASOS FINALES PARA HABILITAR EL USO DEL MAPVIEW

Hay que añadir la referencia a la librería externa com.google.android.maps en el Manifest. Será un hijo del elemento **<applicationd>:**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="es.ucm.fdi.localyza"
      android:versionCode="1"
      android:versionName="1.0">
       ...
    <application android:icon="@drawable/icon"
                    android:label="@string/app_name"
                    android:theme="@style/OrangeTheme">
            <uses-library android:name="com.google.android.maps" />
```

Cuando la aplicación esté lista para ser distribuida en el Market, debermos modificar la clave de los mapas por la generada con el keystore con el que hemos firmado nuestra aplicación.

MOSTRAR EL ZOOM VIEW

Para añadir los controles de Zoom, solamente tenemos que llamar a la función displayZoomControls:

```
private MapView mapa;
mapa = (MapView)findViewById(R.id.mapa);
mapa.displayZoomControls(true);
```

Y con la siguiente función, se hace zoom o se aleja el mapa:

```
public boolean onKeyDown(int keyCode, KeyEvent event)
{
	MapController mc = mapa.getController();
	switch (keyCode){
		case KeyEvent.KEYCODE_3:
			mc.zoomIn();
			break;

		case KeyEvent.KEYCODE_1:
			mc.zoomOut();
			break;

	  }
	return super.onKeyDown(keyCode, event);

}
```

CAMBIAR LAS VISTAS DEL MAPA

Activación de la vista satélite:

```
mapa.setSatellite(true);
```

Activación de la vista de calles:

```
mapa.setStreetView(true);
```

SITUAR LA POSICIÓN DEL MAPA EN UN PUNTO CONCRETO

Se define la longitud y latitud, creamos un GeoPoint, y mediante el MapController posicionamos la vista en las coordenadas.

```
mapa.setBuiltInZoomControls(true);
Double latitud = 40.452723*1E6;
Double longitud = -3.733253*1E6;
GeoPoint loc =
            new GeoPoint(latitud.intValue(), longitud.intValue());
MapController controlMapa = mapa.getController();
controlMapa.animateTo(loc);
```

AÑADIR CAPAS A LOS MAPAS

Primero se define la capa Overlay que pintará el icono.

```
class MapOverlay extends com.google.android.maps.Overlay
{
      …
}
```

Y se redefine el método draw:

```
public boolean draw(Canvas canvas, MapView mapView, boolean shadow,
long when)
```

```
{
    super.draw(canvas, mapView, shadow);
    //---transformamos el geopoint a pixeles---
    Point screenPts = new Point();
    mapView.getProjection().toPixels(loc, screenPts);

    //--- añadimos el marcador---
    Bitmap bmp = BitmapFactory.decodeResource(
          getResources(), R.drawable.point_mor);

   canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
    return true;

}
```

Ahora, solo falta crear una lista de MapsOverlays e indicar sobre que mapView ha de pintarse:

```
//---Añadimos el marcador de la localización---

MapOverlay mapOverlay = new MapOverlay();
List<Overlay> listOfOverlays = mapa.getOverlays();
listOfOverlays.clear();
listOfOverlays.add(mapOverlay);

mapa.invalidate();
```

GPS

Actualmente, existe una gran cantidad de aplicaciones basadas en localización, y día a día siguen aumentando. Actualmente, existen treinta y un satélites sin nada mejor que hacer que proveernos de estos servicios.

SOLICITUDES DE PERMISOS

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

OBTENIENDO LAS ACTUALIZACIONES DE LA LOCALIZACIÓN

Para obtener la localización en Android funciona por medio de la devolución de la llamada. Lo recibimos por medio de las actualizaciones del LocationManager, llamando al método requestLocationUpdates, y pasándole un LocationListener.

```
// Obtención de una referencia al LocationManager
        LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

        //Definición del listener que responde a las actuzalizaciones
de la localización.

        LocationListener locationListener = new LocationListener() {

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

            public void onProviderEnabled(String provider) {}

            public void onProviderDisabled(String provider) {}

                    public void onLocationChanged(Location location) {

                          //Que hacer con la nueva localización
                    }
          };

        // Registrar el listener con el LocationManager para recibir
actualizaciones

locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDE
R, 0, 0, locationListener);
```

En esta última línea, tenemos que indicar si queremos usar la red o los GPS como proveedor sustituyendo

LocationManager.NETWORK_PROVIDER

por

LocationManager.GPS_PROVIDER.

Los otros dos parámetros son:

- minTime: es el minimo intervalo de tiempo para las notificaciones en milisegundos. Se usa para evitar hacer demasiadas peticiones al servicio de localización y ahorrar batería.
- minDistance: el minimo intervalo de distancia para las notificaciones, en metros.

Para detener las actualizaciones, tenemos que eliminar el listener que previamente habíamos indicado al LocationManager:

```
locationManager.removeUpdates(locationListener);
```

OBTENER LA ÚLTIMA LOCALIZACIÓN CONOCIDA

En ocasiones, podemos querer una rápida localización y en un primer instante, puede ser útil utilizar las últimas coordenadas válidas. Podemos obtenerlas de la siguiente forma:

```
locationProvider = LocationManager.NETWORK_PROVIDER;
        Location lastKnownLocation =
locationManager.getLastKnownLocation(locationProvider);
```

GEOCODER

Es una clase que permite transformar una dirección o descripción de un lugar en las coordenadas (longitud y latitud). La geo codificación inversa es el proceso de transformar las coordenadas en una dirección parcial. La cantidad de detalles obtenidos puede variar según el lugar.

¿CÓMO OBTENER EL LUGAR A PARTIR DE UNAS COORDENADAS?

```
Geocoder gc=new Geocoder(getApplicationContext());
List<Address> address= gc.getFromLocation (latitude, longitude,
                        maxResults);
```

¿CÓMO OBTENER LUGARES A PARTIR DE UNA DESCRIPCIÓN?

```
Geocoder gc=new Geocoder(getApplicationContext());
List<Address> address=gc.getFromLocationName(locationName,
maxResults);
```

7. TELEFONÍA

Manuel Báez – Jorge Cordero – Miguel González

MENSAJES DE TEXTO

La posibilidad de enviar y recibir mensajes de texto provocó una revolución en los teléfonos móviles, siendo una de las principales formas de comunicación durante la primera década del SXXI hasta la llegada del internet móvil. Como es de esperar Android nos da la posibilidad de enviar SMS. El encargado de ello es el Android.telephony.SmsManager que soporta tanto GSM(*Groupe spéciale mobile*) como CDMA(*Code Division Multiple Access*).

Dicha clase tiene únicamente cinco métodos públicos que son los siguientes.

- ArrayList<String> divideMessage(String text)

Su función es la de recibir como parámetro un *String text* y convertirlo en un *ArrayList* de Strings. El tamaño de estos Strings será menor o igual al máximo permitido por un SMS

- static SmsManager getDefault()

Devuelve la instancia por defecto de SmsManager

- void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[ ] data, PendingIntent sentIntent, PendingIntent deliveryIntent)

Es el método que se encarga de enviar el SMS. Le pasamos como parámetro la dirección de destino, el puerto de destino, el SMS en sí y dos *PendingIntent* que sirven para el envío y el acuse de recibo.

- void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)

En el caso de tener un SMS de longitud mayor que lo permitido para uno tendremos que usar este método. Su funcionamiento es el mismo que el de *sendDataMessage*, la diferencia es que hay un array list para cada PendingIntent así como para cada parte(parts).

- void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Vamos a ver ahora como enviar un SMS sirviéndonos de dichos métodos.

Empezamos suponiendo que tenemos ya el texto que queremos enviar en un String text y el número al que lo queremos enviar en un String phoneNumber, también necesitaremos un PendingIntent que lo obtendremos del siguiente modo:

PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, SMS.class), 0);

Ahora vamos a utilizar SmsManager.getDefault() para tener la instancia por defecto de SmsManager que llamaremos sms.

Tenemos dos opciones ante nosotros, que el texto quepa en un SMS o que no, en el caso de que quepa, la solución es más sencilla, solamente tendremos que llamar a sms.sendTextMessage(phoneNumber, null, text, pi, null) como podemos apreciar hemos dejado el campo de scAddress como null, para que tome un valor por defecto y el de deliveryIntent también.

En el caso de que text no quepa en un único SMS tendremos que dividir el texto del SMS en varios mensajes, para ello utilizaremos el método sms.divideMessage(text) y guardaremos el resultado obtenido en arrayText. Ahora procederemos a enviar el mensaje al número deseado, en este caso usaremos el método sms.sendMultipartTextMessage(phoneNumber, null, arrayText, pi, null) y ya habremos enviado el mensaje múltiple.

Es importante destacar que en el caso de los sms y mms, al ser información que ha de ser accesible desde cualquier aplicación, es necesario que se almacenen como Content Providers, y desde la versión 2.0+ es así. Están almacenados en la ruta /data/data/com.android.providers/telephony/databases/mmssms.db, y podemos ver cómo acceder a ellos en el punto del manual de los Content Providers.

Es importante destacar que el URI de los sms es el siguiente

Código Android

```
String url = "content://sms/";
Uri uri = Uri.parse(url);
getContentResolver().registerContentObserver(uri, true, new MyContentObserver(handler));
```

LLAMADAS A TELÉFONO

La razón por la que se crearon los teléfonos móviles era la de poder tener una línea no fija desde la que pudiéramos hacer llamadas desde cualquier punto, siempre y cuando haya cobertura por supuesto, sin necesidad de cables ni nada por el estilo. Por ello es lógico suponer e incluso obvio que Android nos de la posibilidad de llamar desde una aplicación propia. A continuación vamos a ver cómo realizar dichas llamadas. Para poder explicarlo primero tendremos que explicar qué es un Intent y cómo usarlos con un startActivity.

Un Intent (intento) es un paquete, dicho paquete contiene información del componente que reciba el intento. La constructora de Intent que vamos a utilizar en este caso es de la forma Intent(String action, Uri uri) siendo el primer argumento, action, la acción a realizar y el segundo, uri, el dato sobre el que realizar dicha acción. Por su parte startActivity(Intent), hay otras formas de llamar a startActivity pero la que nos interesa es ésta, lo que hace es intentar ejecutar la actividad determinada por Intent, poniéndola en la cima de la pila de actividad.

A continuación tenemos un fragmento de código que lo que muestra un AlertDialog cuando pulsamos sobre un contacto, el cual recibimos por parámetro en este método. Dicho AlertDialog tiene varias opciones de contacto entre ellas está la de "Llamar" cuyo código está enmarcado en rojo.

Código Android

```
private void launchContactOptions(final ContactListItem  contacto) {
    // Elementos del Dialog
    final String items[] = {"Llamar","Otro"};
    AlertDialog.Builder dialog = new AlertDialog.Builder(ContactosActivity.this);
    dialog.setTitle("Opciones de contacto");
    dialog.setItems(items, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface d, int choice) {
            switch (choice) {
            case 0:     // Éste es el código que nos interesa
                //Elegida opción "Llamar" del AlertDialog
                String url = "tel:" + contacto.getNum();
                Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(url));
                ContactosActivity.this.startActivity(intent);
                break;
            case 1:
            //Elegida opción "Otro" del AlertDialog
```

```
                break;
                default:
                break;
                }
            }
        });

        dialog.show();
}
```

Registro de llamadas (CallLog) e información de las llamadas.

Para obtener la lista de llamadas tendremos que llamar al android.provider.CallLog el cual se encarga de guardar un registro de las llamadas tanto recibidas como realizadas. Algunos campos importantes del registro CallLog son:

Campo	**Descripción**
android.provider.CallLog.Calls.NUMBER	El número involucrado
android.provider.CallLog.Calls.TYPE	Tipo de llamada. Entrante (1) o saliente (0).
android.provider.CallLog.Calls.CACHED_NAME	Nombre asociado con el número en caso de existir en la agenda.
android.provider.CallLog.Calls.DATE	Fecha de la llamada
android.provider.CallLog.Calls.DURATION	Duración de la llamada en segundos

Código ejemplo: Consulta de llamadas salientes de duración superior a 0 segundos.

Código Android

```
ArrayList<String[]> result = new ArrayList<String[]>();


  // Consulta SQL
                String[] strFields = {
                      android.provider.CallLog.Calls.NUMBER,
                      android.provider.CallLog.Calls.TYPE,
                      android.provider.CallLog.Calls.CACHED_NAME,
                      android.provider.CallLog.Calls.DATE,
                      android.provider.CallLog.Calls.DURATION
                      };
//Condición llamadas salientes:
          String where = new String("1=" + android.provider.CallLog.Calls.TYPE + "");
          where += " AND " + android.provider.CallLog.Calls.DURATION + ">0";
          String strOrder = android.provider.CallLog.Calls.DATE + " DESC";
                Cursor mCursor = getContentResolver().query(
                      android.provider.CallLog.Calls.CONTENT_URI,
                      strFields,
                      where,
                      null,
                      strOrder
                      );
          // Iteramos sobre la lista de resultados a través del cursor.
                int nameIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.CACHED_NAME);
                int numberIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.NUMBER);
                int typeIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.TYPE);
```

```
                int dateIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DATE);
                int durIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DURATION);
                if (mCursor.moveToFirst()) {
                   do {
                      String name = mCursor.getString(nameIndex);
                      String number = mCursor.getString(numberIndex);
                      String type = mCursor.getString(typeIndex);
                      String date = mCursor.getString(dateIndex);
                      String dur = mCursor.getString(durIndex);
                      result.add(new String{type, name, number, date, dur});

                   } while (mCursor.moveToNext());
                }
```

ACCEDER A LA AGENDA

Accedemos a la agenda mediante la clase android.provider.**ContactsContract**, Disponible desde la versión 2.0+ (API Level 5). Para poder acceder a los contactos debemos añadir permisos a la aplicación en el archivo AndoridManifest.xml.

Código Andorid Manifest.xml

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Para obtener datos de la agenda de contactos almacenamos una instancia del ContentResolver y sobre este ejecutamos query(). La llamada al método query() tiene como primer argumento la URI que identifica el conjunto de datos sobre el que queremos ejecutar la query. Como segundo argumento se introduce un array con las columnas que queremos obtener. Adicionalmente se pueden añadir condiciones a la consulta y el orden en el queremos que devuelva los resultados. A continuacón un método que obtiene informacion de los contactos y la devuelve en un array con el nombre y número de los contactos que tienen número de móvil.

Código Android

```
public ArrayList<String[]> dameContactos() {
        ArrayList<String[]> result = new ArrayList<ContactListItem>();
        ContentResolver cr = getContentResolver();
    // Consulta SQL
    Cursor mCursor = cr.query(
          Data.CONTENT_URI,
          //Columnas a seleccionar
           new String[] { Data.CONTACT_ID, Data.DISPLAY_NAME,
                    Phone.NUMBER,  Phone.TYPE,
                   ContactsContract.CommonDataKinds.Photo.CONTACT_ID},
          //Condiciones
          Phone.NUMBER + " IS NOT NULL",
          null,
          //Orden
          Data.DISPLAY_NAME + " ASC");
    startManagingCursor(mCursor);

    // Iteramos sobre la lista de resultados a través del cursor.
     int nameIndex = mCursor.getColumnIndexOrThrow(Data.DISPLAY_NAME);
     int numberInde x = mCursor.getColumnIndexOrThrow(Phone.NUMBER);
     int contactoIndex = mCursor.getColumnIndex(Data.CONTACT_ID);

     if (mCursor.moveToFirst()) {
        do {
           String name = mCursor.getString(nameIndex);
           String number = mCursor.getString(numberIndex);
           result.add(new String[] {name, number});
        } while (mCursor.moveToNext());
     }
     return result;
}
```

8. SENSORES

Manuel Báez – Jorge Cordero – Miguel González

Android dispone del paquete ***Android.Hardware*** para dar soporte a la cámara del dispositivo y a otros sensores.

<uses-feature>

Los sensores utilizados por una aplicación deberán de estar declarados en el AndroidManifest.xml haciendo uso de la etiqueta <uses-feature> para que la aplicación pueda hacer uso de ellos. Está información es utilizada por el Android Market para saber si una aplicación es compatible con un dispositivo concreto.

Por ejemplo si la aplicación que queremos desarrollar necesita usar el acelerómetro, incluiremos una linea en el AndroidManifest.xml e indicaremos que es requerido (atributo android:required) como se muestra en el siguiente código.

Código de AndroidManifest.xml

```
...
<uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true" />
<uses-feature android:name="android.hardware.bluetooth" android:required="false"/>
<uses-feature android:name="android.hardware.camera"/>
...
```

El atributo android:required está por defecto a *true*, por lo que si no lo incluimos Android entenderá que el hardware indicado será necesario para la aplicación como sucede con la cámara en el código anterior.

Las clases e interfaces disponibles en android.hardware para el control de sensores, son:

Nombre	Tipo	Descripción
SensorManager	Clase	Gestiona los sensores del dispositivo.
Sensor	Clase	Representa a un sensor
SensorEvent	Clase	Representa el evento de un gestor
SensorEventListener	Interfaz	Recibe los cambios de un sensor.

Para hacer uso de un sensor hay que implementar la clase SensorEventListener y obtener una instancia del SensorManager. Esta instancia la utilizaremos para indicar que queremos registrar los cambios del sensor mediante el metodo registerListener(SensorEventListener listener, Sensor sensor, int rate, Handler handler) del SensorManager. El primer argumento de este método es la instancia de la clase que implementa la interfaz Sensor EventListener. El segundo parámetro es el sensor que queremos registrar. El último parámetro es para indicar la frecuencia ideal con la que nos gustaría registrar el sensor, la cual puede no corresponder con la real.

Una vez indicado al SensorManager que sensor vamos a registrar disponemos de dos métodos para manejar los eventos registrados:

- public void onAccuracyChanged(Sensor sensor, int accuracy)
- public void onSensorChanged(SensorEvent event)

Una vez queramos dejar de registrar los cambios en los sensores debemos llamar a la función unregisterListener(SensorEventListener listener) de la instancia del SensorManager.

El SensorEvent contiene todos los datos en relación al evento.

Tipo	Nombre	Descripción
public Sensor	sensor	Sensor implicado en el evento
public int	accuracy	Precisión del sensor
public long	timestamp	Tiempo (ns) en el que ocurrió el evento
public final float[]	values	Contiene la información del sensor dependiendo del sensor

Método	Valor devuelto
public String getName ()	Nombre del sensor
public float getMaximumRange ()	Valor máximo que alcanza el sensor
public float getPower ()	Potencia (mA) consumida por el sensor cuando está en funcionamiento
public int getMinDelay ()	Tiempo (ns) mínimo que transcurre entre dos eventos de un Sensor
public int getType()	Tipo del sensor

public int getVersion()	Versión del sensor
public float getResolution()	Resolución del sensor

Ejemplo de código:

Código Android

```
public class miActividad extends Activity implements SensorListener {
SensorManager miSensorManager = null;
//... resto de código
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
     //... resto de código

    //Obtenemos la instancia del SensorManager

    miSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
}

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.getType() == Sensor.TYPE_ACCELEROMETER) {
          //Tratamiento del evento si es

    }
    //Tratamiento de eventos de otros sensores
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
protected void onResume() {
    super.onResume();
    //Indicamos al sensor manager que registre los cambios del acelerómetro
    miSensorManager.registerListener(this,

                SensorManager.SENSOR_ACCELEROMETER,
                SensorManager.SENSOR_DELAY_NORMAL);
}
   @Override
protected void onStop() {
    miSensorManager.unregisterListener(thejamos de registrar los cambios del sensor);
```

```
        super.onStop();
    }   }
```

GESTOS

Android es un sistema operativo para dispositivos móviles, donde el puntero clásico de los sistemas operativos convencionales pasa a ser el contacto táctil del usuario en la pantalla. Podemos ver que algunas aplicaciones tienen funcionalidades distintas cuando un usuario toca la pantalla, incluso variando en función de la duración o forma de dicho toque. Ésto es lo que se llama un gesto en Android y vamos a ver cómo implementarlos.

Estos gestos dan a Android un gran potencial a la hora de poder realizar un gran abanico de acciones con un escaso número de gestos, hay aplicaciones que su única funcionalidad consiste en ahorrarnos tiempo haciendo que ciertos gestos se conviertan en una especie de accesos directos. Incluso hay un teclado llamado Swype que es un teclado mediante gestos haciendo que sea mucho más ágil que un teclado tradicional.

Hay varios tipos de gestos que podemos capturar, para ello tendremos que utlizar GestureDetector.OnGestureListener que nos notificará cuando salta un evento dentro del listener. Tendremos a nuestra disposición los siguientes subtipos del listener:

Métodos:

abstract boolean	onDown(MotionEvent e)	Notifica cuando se produce un down devolviéndolo en el MotionEvent e.
abstract boolean	onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)	Notifica una pulsación mantenida devolviendo el primer punto en MotionEvent e1 y el final en MotionEvent e2.
abstract void	onLongPress(MotionEvent e)	Notifica cuando se produce una pulsación larga devolviendo los datos en MotionEvent e.
abstract boolean	onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)	Notifica cuando se está moviendo un scroll edvolviendo los valores de la posición inicial en MotionEvent e1 y la final en MotionEvent e2

abstract void	onShowPress(MotionEvent e)	Indica que se ha seleccionado un elemento pero no se ha dejado de pulsar todavía, para mostar al usuario se ha reconocido su interacción.
abstract boolean	onSingleTapUp(MotionEvent e)	Notifica cuando se produce un up devolviéndolo en MotionEvent e

A continuación vamos a ver un ejemplo con el onFling haciendo el gesto de izquierda a derecha:

Código Android

```
public class ContactosActivity extends Activity {
    private static final int SWIPE_MIN_DISTANCE = 120;
    private static final int SWIPE_MAX_OFF_PATH = 250;
    private static final int SWIPE_THRESHOLD_VELOCITY = 200;
    private ContactArrayAdapter listAdapter = null;
    private GestureDetector gestureDetector = null;
    private ContactosActivity actividad = null;
    //private ProgressDialog progressDialog = null;
    class MyGestureDetector extends SimpleOnGestureListener {
      @Override
      public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
                        float velocityY) {
                Intent intent = new Intent(ContactosActivity.this, GruposActivity.class);
                if (Math.abs(e1.getY() - e2.getY()) > SWIPE_MAX_OFF_PATH) {
                        return false;
                }
                // right to left swipe
                else if (e1.getX() - e2.getX() > SWIPE_MIN_DISTANCE
                                && Math.abs(velocityX) >
SWIPE_THRESHOLD_VELOCITY) {
    // Tu código para cuando hagas el gesto de izquierda a derecha
                }
```

```
            return false;
    }
    // Es necesrio devolver true en el onDown para que el evento onFling lo registre
    @Override
    public boolean onDown(MotionEvent e) {
            return true;
    }}}
```

9. MULTIMEDIA

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

MULTIMEDIA EN ANDROID

Hoy en día, los dispositivos móviles no solo se usan para hablar y mandar mensajes de texto. La gran mayoría de los mismos permiten, grabar y reproducir audio y video, así como realizar fotografías. En este tema, se intenta presentar todas las posibilidades mencionadas anteriormente, proporcionadas por Android, así como dar a conocer algunos ejemplos de implementación de pequeñas funciones para reproducir audio o hacer fotografías.

El Framework[1] proporcionado por Android para multimedia, permite capturar y reproducir audio, vídeo o imágenes en los formatos más comúnmente usados, pudiendo interactuar con archivos contenidos en el propio terminal Android, con archivos externos al dispositivo, o con aquellos dispuestos a través de Internet. Para la reproducción utiliza MediaPlayer o JetPlayer, siendo MediaRecorder el usado para grabar audio o vídeo, o hacer fotografías a través de la cámara del terminal. Android soporta varios formatos de audio, video e imágenes. Algunas de ellas como mp3, midi y flac para audio; 3gp y mpeg para videos; y jpg y png entre otros para imágenes.

REPRODUCCIÓN DE AUDIO

En la reproducción de audio, Android proporciona sus propias clases destinados a ello. A continuación, se describirá como usar la clase "**MediaPlayer**" para programar en nuestra propia aplicación y así crear un reproductor propio.

Para poder usar la clase "MediaPlayer", primero hay que especificar los permisos necesarios para que no dé problemas a la hora de compilar el programa. Para ello, en el AndroidManifest se incluye lo siguiente:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Esto permitirá la conexión a Internet en el caso de que se reproduzca audio o video extraído de la web. Para llevar un mejor manejo de la reproducción, se puede incluir un par de botones como son "start" y "pause"

(1) Framework es un término que significa "marco". Es decir, proporciona un conjunto de funciones para implementar aplicaciones con características semejantes.

Si además se quiere poder reproducir multimedia con el dispositivo en suspensión o bloqueado (usando los métodos setScreenOnWhilePlaying() o setWakeMode() que más adelante serán explicados), se añadirá la línea:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Como ya se ha indicado antes, MediaPlayer permite reproducir audio o video desde diferente puntos:

- Recursos locales (los cuales se encontrarán guardados en la carpeta "raw" del proyecto).
- URI interna (Identificador Uniforme de Recurso en inglés, ya sea guardado en el dispositivo o en una tarjeta SD).
- URL externa (obtenida de una página web).

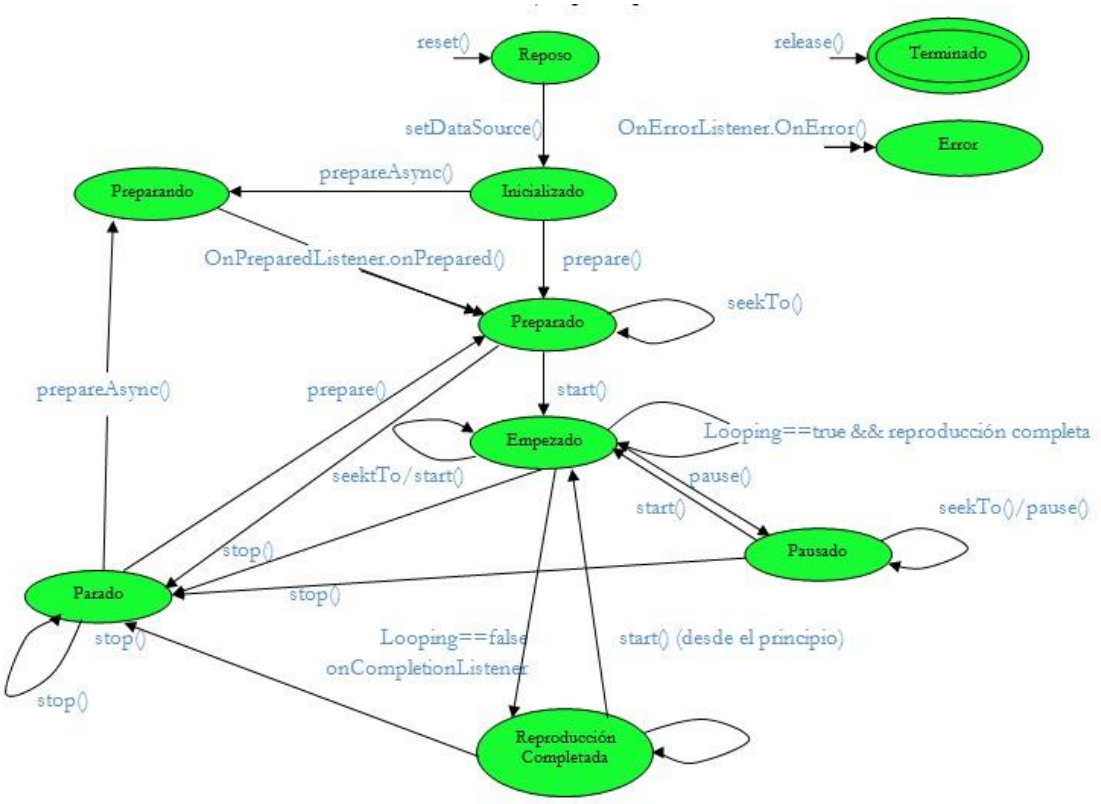


Figura 1. Funcionamiento de la clase MediaPlayer

El funcionamiento de esta clase funciona como una máquina de estados. Cuando el objeto MediaPlayer es creado mediante new() o después de llamar a la función release(), se dice que está en un estado de reposo. Hasta llegar a este estado mediante una de las dos llamadas, el objeto pasará por todo su ciclo de vida.

El ciclo de vida de una reproducción sin tener errores pasaría por los siguientes estados: **Reposo** (cuando se crea el objeto MediaPlayer), **Inicializado** (se le indica al objeto cuál va a ser la pista o video a reproducir), **Preparado** (el MediaPlayer ya tiene todo listo para reproducir), **Empezado** (MediaPlayer está reproduciendo), **Parado** (la pista terminó y MediaPlayer espera una nueva instrucción).

Para poder usar un objeto MediaPlayer, se invoca la constructora por defecto que proporciona Android. Una vez construido el objeto, se indica desde donde va a reproducir la pista. En el caso de que se quiera reproducir desde la carpeta raw del proyecto, hay que especificarle el lugar (en este caso la carpeta raw) y el nombre del archivo que se quiera reproducir:

```
MediaPlayer mp = new MediaPlayer();
mp = MediaPlayer.create(R.raw.cancion_prueba.mp3);
mp.start();
```

Si el archivo que se va a usar se encuentra en la memoria del dispositivo o en una tarjeta SD, mediante una variable de tipo URI se indica la ruta del teléfono en la cual se encuentra el archivo a reproducir:

```
Uri miUri = .... ;//Inicializa la variable apuntando al archivo deseado
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(getApplicationContext(), miUri);
mp.prepare();
mp.start();
```

Si por el contrario el archivo se desea reproducir desde una url, mediante una conexión http indicamos al objeto desde donde descargarlo (hay que tener en cuenta que primero debe descargar el archivo, por lo que puede tardar un tiempo dependiendo de la velocidad de conexión):

```
String url = "http://...";//Se introduce la url desde la que operar
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(url);
mp.prepare();
mp.start();
```

Como se ha podido comprobar, tanto para reproducir desde una URI interna como desde una url externa, es necesario preparar el MediaPlayer para obtener el archivo, al contrario que con un archivo local que se encuentra dentro del proyecto y listo para usarse. En el caso de querer manejar también el audio de la reproducción, se incluirá la clase "AudioManager" la cual primero hay que escribir los permisos necesarios para poder utilizarla:

```
<accion android:name="android.media.AUDIO_BECOMING_NOISY" />
```

Y así poder obtener y manejar a nuestro antojo el volumen de la aplicación. En el siguiente ejemplo se puede observar la forma de obtener el volumen máximo permitido por la aplicación y el volumen actual:

```
AudioManager aManager = (AudioManager) getSystemService(AUDIO_SERVICE);
                //Captura las especificaciones del volumen de la aplicación
float actualVolumen = (float) aManager.getStreamVolume(aManager.STREAM_MUSIC);
float maxVolumen = (float) aManager.getStreamMaxVolume(aManager.STREAM_MUSIC);
```

Una vez obtenidos estos datos, se podrá manejar el volumen de forma sencilla.

REPRODUCCIÓN DE VIDEO

Para programar un reproductor de video en nuestra aplicación, será necesario incluir en la interfaz (archivo .xml) un control de "VideoView" que se encuentra en la paleta de "Images & Media". Para llevar un mejor manejo de la reproducción del video, además podemos añadir un par de botones para reproducir y pausar el video.

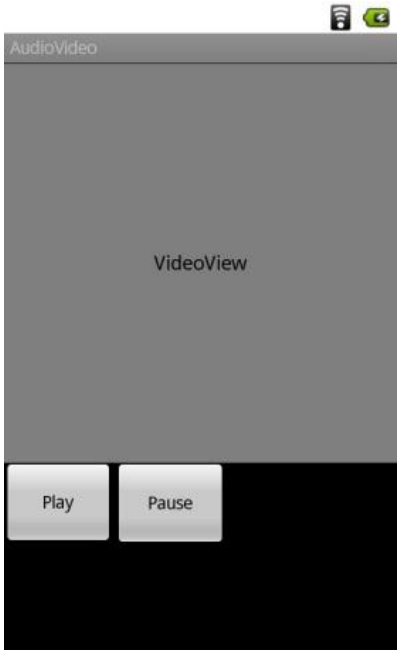


Figura 2. Reproducción de video

En este caso, podremos reproducir un video que esté alojado en la tarjeta SD de nuestro dispositivo (URI) o a través de una URL externa de la web. Para ello, hay que diferenciar los dos métodos de linkado para reproducir uno u otro video.

Para la reproducción a través de un objeto de tipo URI, escribimos las siguientes instrucciones:

```
Uri uri = ....;  //Se incluye la ruta del teléfono donde se encuentre el video
VideoView video = ..... //linkado con su id correspondiente al archivo .xml
video.setVideoUri(uri);
```

Y en el caso de que reproduzca a través de una url externa:

```
String url = ....;  //URL desde donde se descargará el video
VideoView video = .....;
video.setVideoPath(url);
```

Para terminar, proporcionamos a los dos botones un setOnClickListener() para que cada uno realice su función. En el caso del botón play, la instrucción a escribir será video.play(). Y en el caso del botón pause la instrucción será video.pause().

GRABAR SONIDO

El framework multimedia de Android incluye un soporte que permite capturar y codificar una gran variedad de formatos comunes de audio, de modo que estos se puedan integrar fácilmente en la aplicación. Para ello, se puede usar la API "MediaRecorder" siempre y cuando sea compatible con el hardware del dispositivo (que tenga micrófono o no). Sin embargo, el emulador de Android no posee la capacidad de capturar audio, por lo que toda prueba que se precie se deberá hacer en un dispositivo que disponga de micrófono.

Antes de nada, se deberán incluir los permisos necesarios para poder escribir desde una entrada externa en la tarjeta SD y para poder grabar audio. En el archivo AndroidManifest se escribirá:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

La captura de audio desde el dispositivo es más complicado que la reproducción de video o audio. Aún así, se puede programar de forma sencilla. En primer lugar, hay que crear un objeto de tipo MediaRecorder. Del mismo modo se puede crear otro objeto MediaPlayer (visto en este tema), para reproducir la pista capturada:

```
MediaRecorder mRecorder = new MediaRecorder();
MediaPlayer mPlayer = new MediaPlayer();
```

A continuación, se definirá el micrófono como medio para capturar el audio y se define que el archivo se va a guardar con la especificación 3GPP, con extensión .3gp y el codec que se va a emplear:

```
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

Una vez en este punto, hay que indicar la dirección de la tarjeta SD; indicándole al objeto MediaRecorder que, una vez realizada la captura de audio, debe almacenarse en esta dirección. A su vez, creamos un archivo temporal 3gp almacenado en la tarjeta SD donde se guardará el audio:

```
File direccion= new File(Environment.getExternalStorageDirectory().getPath());
File archivo = File.createTempFile("temporal", ".3gp", direccion);
mRecorder.setOutPutFile(archivo.getAbsolutePath());
```

Ahora, ya podemos preparar la captura y empezar con ella. Y pararla una vez se haya terminado de grabar lo deseado:

```
mRecorder.prepare();
mRecorder.start();
.....
mRecorder.stop();
mRecorder.release();
```

Al igual que se ha visto en la reproducción de video, se puede incluir en la interfaz unos botones para tener un mejor control del inicio, parado y reproducción de la grabación de audio.

GRABAR VIDEO

Al igual que pasaba en el apartado anterior, para la captura de video se necesita un dispositivo que disponga de una cámara. También se usará un objeto de tipo "MediaRecorder" solo que esta vez le indicaremos que la captura se obtendrá de la cámara. En este caso, a parte de los permisos que se declararon para la grabación de audio, hay que añadirle los permisos para grabar video y para usar la cámara:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.RECORD_VIDEO" />
<uses-permission android:name="android.permission.CAMERA" />
```

Figura 3. Grabación de video

En la interfaz, hay que añadir un "SurfaceView", el cual permitirá tener un espacio dedicado a dibujar frames del vídeo para la vista y la reproducción (en la activity principal hay que implementar la interfaz "SurfaceHolder.Callback"). Para este apartado, servirá para la vista previa de la grabación y para reproducirla posteriormente. Además, se podrá añadir tres botones que implementen las funciones grabar, detener y reproducir del "SurfaceView". Como consejo, se puede configurar la rotación de la pantalla para que se quede bloqueada y no re-cree la activity al girar el dispositivo. Para se escribe en el manifiesto la siguiente línea:

```
android:screenOrientation="portrait"
```

En esta aplicación se definirán cuatro variables importantes, el "MediaPlayer" y "MediaRecorder" vistos anteriormente y una variable con el nombre del archivo donde se va a guardar la grabación.

De los métodos heredados de la interfaz, importan dos de ellas. El método "surfaceCreated", en el cual se inicializaran las variables y se propone el "SurfaceHolder" como display para la grabación y la reproducción. Y el método "surfaceDestroyed", en el que se liberarán las variables usando el método release() de éstas.

```
mRecorder.setPreviewDisplay(holder.getSurface());
mPlayer.setDisplay(holder);
```

Ahora se necesitará agregar la cámara para que el objeto "MediaRecorder" use ésta para la grabación. Se configura las fuentes de audio y video. Y se llama a los métodos unlock() y lock() de la cámara. Esto sirve para que ninguna otra aplicación pueda utilizar la cámara mientras esté la nuestra en funcionamiento. El primero se llamará cuando se quiera usar la cámara, y el segundo cuando se haya terminado.

```
mCamara = getCameraInstance();
mCamara.unlock();
mRecoder.setCamera(mCamara);
mRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
mRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
mRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toSting());
```

Una vez llegados a este punto, ya se pueden configurar los botones para el uso de la grabación, usando los comandos start() y stop() del "MediaRecorder" para tener manejo sobre la grabación del video, los mismos comandos para el "MediaPlayer" para la previsualización de la grabación y, una vez terminado el uso de la cámara para grabar el video, el método lock() para bloquearla de nuevo.

10. CREACIÓN DE UN WIDGET

Álvaro Borrego – Francisco Hernández – David Palomero

CREACIÓN DE UN WIDGET SENCILLO

En esta sección del manual se va a explicar cómo crear un widget para Android. En primer lugar se mostrará y explicará con detalle la manera de crear un widget estático, es decir, sin ninguna funcionalidad. De esta manera se pretende que se consiga entender de manera sencilla y clara la estructura de este tipo de componentes muy utilizado en cualquier aplicación Android.

Una vez entendida la estructura de un widget, se añadirá una funcionalidad más avanzada, como puede ser, la realización de una determinada acción al ser pulsado por el usuario o su actualización automática.

El widget que se va a implementar en esta primera parte consistirá en una imagen de un reloj y un texto con un mensaje ('Hora Actual'). Con este ejemplo se pretende comprender de una manera sencilla la estructura de un widget en Android, sin tener en cuenta aspectos más avanzados, como puede ser la interacción con el usuario.

DEFINICIÓN DE LA INTERFAZ GRÁFICA

Como se ha comentado anteriormente, el widget consistirá en una imagen de un reloj y un mensaje de texto, que en un principio, solamente mostrará el mensaje 'Hora Actual'. La imagen general del widget en esta primera parte será:

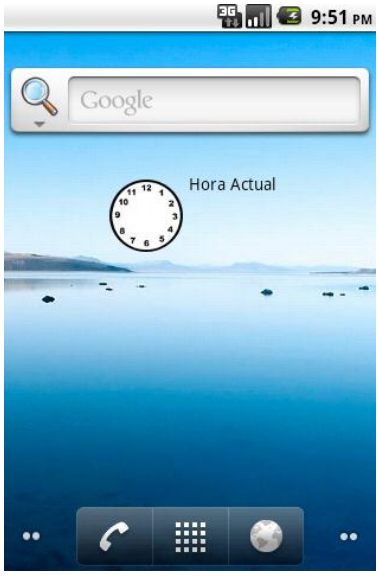


Figura 1. Creación de un widget

Para conseguir este aspecto, se creará en primer lugar, un layout xml llamado *widgethora.xml*. Éste layout está formado sencillamente por un contenedor *LinearLayout* en el que se sitúan una imagen *ImageView* y una etiqueta de texto *TextView* que muestra el mensaje.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:layout_width="wrap_content"
	android:layout_height="wrap_content"
	android:orientation="horizontal"
	android:padding="5dip">

		<ImageView
			android:layout_width="wrap_content"
			android:src="@drawable/reloj"
			android:layout_height="wrap_content"
			android:layout_gravity="right"
			android:id="@+id/imageView1"/>

		<TextView android:id="@+id/textViewHora"
			android:layout_width="fill_parent"
			android:layout_height="fill_parent"
			android:textColor="#000000"
			android:text="Hora Actual" />
</LinearLayout>
```

NOTA: En el ejemplo se ha utilizado una imagen llamada reloj.png, por lo que es necesario crear una imagen con ese nombre e incluirla en el proyecto. Otra opción posible, es sustituir android:src="*@drawable/reloj*" por android:src="*@drawable/icon*", de esta manera el icono del widget no será el de un reloj sino el icono por defecto de Android.

ASIGNAR PROPIEDADES AL WIDGET

En esta parte se da la posibilidad de definir algunas de las propiedades que va a tener el widget, como por ejemplo, el tamaño en pantalla, el nombre que aparecerá en el menú etc... Para ello, se crea un nuevo xml (que se ubicará en la carpeta \res\xml del proyecto) llamado *widgethora_provider.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:initialLayout="@layout/widgethora"
	android:label="Widget de la Hora"
	android:minWidth="146dip"
	android:minHeight="72dip"/>
```

En este caso se han definido las siguientes propiedades:

- **initialLayout:** hace referencia al layuout xml creado anteriormente, en este caso, *widgethora*.
- **label:** será el nombre que aparecerá en el menú de aplicaciones de Android.
- **minWidth/minHeight:** define el ancho y el alto respectivamente del widget en pantalla. En Android, la pantalla se divide en pequeñas celdas donde se pueden colocar iconos de aplicaciones, iconos de contactos, widgets, etc.

En el ejemplo, se quiere tener un widget con unas dimensiones de dos celdas de ancho y una celda de alto (2x1). Para conseguir estas dimensiones, existe una fórmula que indica el valor exacto que deben tener las propiedades minWidth y minHeight:

dimensión = (número de celdas * 74) - 2

Teniendo en cuenta esta fórmula, y sustituyendo el número de celdas deseadas para cada dimensión, se obtienen los valores minWidth="146dip" y minHeight="72dip". Existen otras propiedades muy interesantes que no se han tenido en cuenta en este ejemplo, pero que serían de gran utilidad en la mayoría de aplicaciones con widgets, como pueden ser:

- **icon:** icono que se muestra para el widget en el selector de AppWidget.
- **minResizeWidth/minResizeHeight:** anchura/altura mínima que se puede cambiar de tamaño para el widget.
- **resizeMode:** Las reglas por las que se puede cambiar el tamaño de un widget.
- **updatePeriodMillis**: frecuencia en la que se actualizará el widget, definida en milisegundos.

IMPLEMENTACIÓN DE LA CLASE PRINCIPAL

En este paso, se declarará la clase principal (que será la que se mostrará al pulsar sobre el icono en el menú). En este ejemplo, se mostrará una pantalla en negro con un mensaje que mostrará la forma de añadir el widget creado al escritorio.

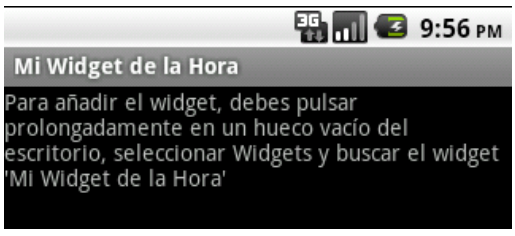


Figura 2. Información de un widget

El código de esta clase es el siguiente:

```
public class WidgetHoraMain extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

El layout *main* estará formado únicamente por una etiqueta de texto, explicando la forma de añadir el widget al escritorio.

```
<?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">

      <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/mensaje"/>

</LinearLayout>
```

Se debe incluir en el archivo *\res\values\strings.xml* la siguiente entrada:

```
<string name="mensaje">"Para añadir el widget, debes pulsar
      prolongadamente en un hueco vacío del escritorio, seleccionar
      Widgets y buscar el widget 'Mi Widget de la Hora'"
</string>
```

Como se ha comentado en repetidas ocasiones a lo largo de la explicación, en esta primera parte del ejemplo no se va a implementar ninguna funcionalidad del widget, no obstante, se va a declarar y a explicar brevemente la clase necesaria para esta tal efecto ya que será implementada más adelante. Es necesario crear la clase *MiWidgetHora.java:*

```
public class MiWidgetHora extends AppWidgetProvider {
      public void onUpdate(Context context,
                           AppWidgetManager appWidgetManager,
                           int[] appWidgetIds) {
      }

}
```

Los principales eventos en un widget son:

- **onEnabled():** lanzado cuando se añade al escritorio la primera instancia de un widget.
- **onUpdate():** lanzado periódicamente cada vez que se debe actualizar un widget.
- **onDeleted():** lanzado cuando se elimina del escritorio una instancia de un widget.
- **onDisabled():** lanzado cuando se elimina del escritorio la última instancia de un widget.

AÑADIR EL WIDGET AL MANIFEST DEL PROYECTO

Por último, queda declarar el widget en el archivo *manifest* del proyecto.

```
<receiver android:name=".MiWidgetHora" android:label="Mi Widget de la
Hora">
<intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
</intent-filter>
<meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/widgethora_provider" />
</receiver>
```

WIDGET CON FUNCIONALIDAD

Una vez explicadas las características más importantes para la creación de un widget, llega el momento de añadirle funcionalidad, permitiendo su actualización e interacción con el usuario. En esta segunda parte, se va a utilizar el widget creado anteriormente, añadiendo una serie de eventos que permitan realizar una acción concreta al ser pulsado por el usuario.

La idea de este widget más avanzado, será mostrar la hora actual del dispositivo al ser pulsado por el usuario. También se añadirá la opción de actualizarse automáticamente pasado un periodo de tiempo determinado. El layout *widgethora.xml* no se ha modificado, por lo que el widget seguirá teniendo la misma apariencia. El objetivo, es ahora, mostrar la hora actual del dispositivo en lugar de un mensaje estático.

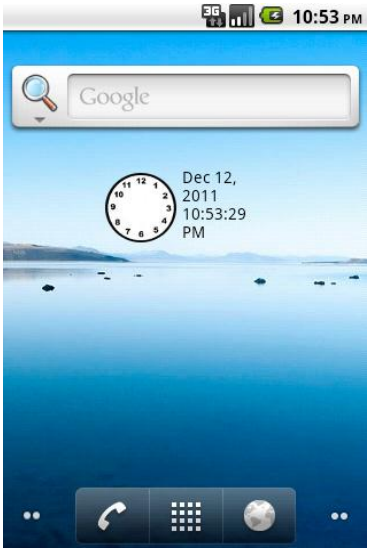


Figura 3. Widget más avanzado

DEFINIR EL TIEMPO DE ACTUALIZACIÓN DEL WIDGET

Como una de las funciones del widget será la de actualizarse automáticamente, habrá que definir el tiempo de actualización (en milisegundos). Para ello, simplemente hay que añadir la siguiente propiedad en el widgethora_provider.xml:

```
android:updatePeriodMillis="1800000"
```

Una opción muy interesante, pero que no se implementará en este caso debido a la simplicidad del ejemplo, es la de mostrar una pantalla de configuración del widget, de esta manera el usuario podrá seleccionar las características que le interesen. Para mostrar una pantalla de configuración, bastaría con añadir, a este mismo archivo:

```
android:configure="rutaDelPaquete.ConfigWidget"
```

Siendo ConfigWidget la actividad donde se definen las distintas opciones de configuración. Con esto, al colocar el widget en el dispositivo, se ejecutará automáticamente la pantalla de configuración antes de ser colocado.

AÑADIR FUNCIONALIDAD AL WIDGET

Una vez realizadas estas pequeñas modificaciones, es el momento de implementar la funcionalidad. En la primera parte, se creó una clase llamada MiWidgetHora.java que quedó prácticamente vacía (ya que no se deseaba implementar ninguna funcionalidad). Ahora es necesario completar esa clase, ya que va a ser la encargada de realizar todas las acciones relacionadas con el widget.

EVENTO ON_UPDATE()

El evento onUpdate (lanzado periódicamente cada vez que se debe actualizar un widget) recibe como parámetro una lista de todas las instancias añadidas al escritorio. Cuando se actualiza, es necesario actualizar todas estas instancias. Para ello, se recorre dicha lista y se van actualizando una a una llamando al método actualizar, que será explicado más adelante.

```
public void onUpdate(Context context,AppWidgetManager
                     appWidgetManager,int[] appWidgetIds) {

      int i=0;
      while (i<appWidgetIds.length)
      {
            int id=appWidgetIds[i];
            actualizar(context, appWidgetManager, id);
            i++;
      }
}
```

EVENTO ON_RECEIVE()

El evento *onReceive*, es el encargado de capturar los mensajes enviados por las componentes. Dependiendo del tipo de mensaje recibido, se realizará una función u otra. En este ejemplo sólo se va a capturar un tipo de mensaje, que será el enviado al pulsar sobre el icono del reloj de la interfaz. Para indicar esto, se creará al principio de la clase un atributo que hará referencia a este tipo de mensajes.

```
private static final String ACCION_PULSAR="PULSAR";
```

El primer paso a realizar en éste método, es capturar la acción asociada al *intent*:

```
final String action = intent.getAction();
```

Una vez que se tiene esta acción, es necesario saber de qué tipo es. Come se ha comentado anteriormente, en este ejemplo, sólo se tiene un tipo de mensaje (ACCION_PULSAR), por lo que bastará con comprobar si el mensaje capturado es de este tipo. En caso de ser así, se debe conocer la id del widget pulsado, ya que es posible tener más de un widget de este mismo tipo en pantalla. Finalmente, sólo queda llamar al método actualizar, encargado de realizar la acción sobre el widget pulsado. Dicho método, necesita como uno de sus argumentos un widget manager, por lo que es necesario obtenerlo antes de realizar la llamada:

```
AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);
```

El código completo de *onReceive* se muestra a continuación:

```
public void onReceive(Context context, Intent intent) {
  final String action = intent.getAction();

  if (action.equals(ACCION_PULSAR)){
      final int id = intent.getIntExtra
                        (AppWidgetManager.EXTRA_APPWIDGET_ID,
                         AppWidgetManager.INVALID_APPWIDGET_ID);

      AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);
      actualizar(context, widgetManager, id);

  }
  super.onReceive(context, intent);
}
```

MÉTODO ACTUALIZAR

El método actualizar, es el encargado de realizar la actualización del widget. Un widget está formado por una serie de componentes llamadas Remote Views. Para hacer referencia a cada una de estas vistas, es necesario acceder a la lista de componentes:

```
RemoteViews componentes = new RemoteViews(context.getPackageName(),
                                          R.layout.widgethora);
```

El objeto componentes, contiene una lista con todas las vistas que forman el widget, en este caso, un *TextView* y un *ImageView*. La forma de hacer referencia a cada una de ellas es la siguiente:

```
componentes.setTextViewText(R.id.textViewHora,"...");
```

Con esto, se consigue asignar al elemento de tipo *TextView* con id *textViewHora* (definido en el *layaout widgethora.xml*) el string encerrado entre comillas. Ahora, toca el turno de hacer lo mismo con el componente *imageviewReloj* (la imagen del reloj). En este caso, lo que se desea es definir el evento *OnClick* sobre la imagen, para así poder actualizar el widget cuando es pulsada por el usuario. Como se ha comentado anteriormente en el *OnReceive*, la forma de conseguir asignar un evento de tipo *OnClick* a una componente de un widget, es enviando un mensaje de tipo ACCION_PULSAR. Para ello, lo primero que hay que hacer es crear un ***intent*** y asociarle la acción comentada. También será necesario conocer el id del widget pulsado.

```
final Intent onClickIntent = new Intent(context, MiWidgetHora.class);
onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);
```

Como la finalidad de este widget es la de mostrar la hora actual del dispositivo, va a ser necesario implementar un método que realice dicha tarea:

```
private static String dameHora(){
      Calendar calendario = new GregorianCalendar();
      return calendario.getTime().toLocaleString();
}
```

Para asegurar la correcta actualización de los componentes del widget en la interfaz, es necesario añadir al final de este método, la siguiente línea:

```
appWidgetManager.updateAppWidget(id, componentes);
```

De esta manera se consigue refrescar de manera adecuada todos los componentes del widget.

El código completo del método *actualizar* queda de la siguiente manera:

```
public static void actualizar(Context context, AppWidgetManager
                              appWidgetManager, int id){

    RemoteViews componentes = new RemoteViews(context.getPackageName()
                                              ,R.layout.widgethora);

    final Intent onClickIntent = new Intent(context,
                                            MiWidgetHora.class);

    onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
    onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);

    final PendingIntent onclickPendingIntent =
            PendingIntent.getBroadcast(context, id, onClickIntent,
            PendingIntent.FLAG_UPDATE_CURRENT);

    componentes.setOnClickPendingIntent(R.id.imageViewReloj,
                                        onclickPendingIntent);

    String horaActual= dameHora();
    componentes.setTextViewText(R.id.textViewHora, "...");
    appWidgetManager.updateAppWidget(id, componentes);
}
```

11. PUBLICANDO EN EL MARKET

Álvaro Borrego – Francisco Hernández – David Palomero

EL ANDROID MARKET

El Android Market es la tienda de aplicaciones de Android. Tiene un acceso rápido y ágil a aplicaciones creadas por desarrolladores de todo el mundo. Desde el punto de vista del desarrollador, se puede usar el Market para publicar sus propias aplicaciones. Tiene las siguientes características destacables:

- Es libre, cualquier usuario puede apuntarse.
- Las aplicaciones publicadas pueden ser puntuadas y comentadas por los usuarios.
- Cuenta con estadísticas de cada aplicación, tales como número de descargas, puntuaciones, comentarios…
- Las aplicaciones publicadas en él, pueden ser de pago, gratuitas, gratuitas con publicidad o gratuitas con limitaciones. Éstas últimas cuentan con su versión completa de pago.

Una de las grandes ventajas que ofrece el Android Market es la posibilidad de instalar una aplicación desde un PC teniendo el dispositivo conectado.

ACCEDER AL ANDROID MARKET

Existen dos maneras por las que se puede acceder al Market para descargar aplicaciones a los dispositivos: mediante la aplicación preinstalada en los dispositivos Android, señalizada con el icono mostrado en la Figura 1., o mediante la Web oficial https://market.android.com cuya imagen inicial se muestra en al Figura 2.

Figura 1. Icono del Market

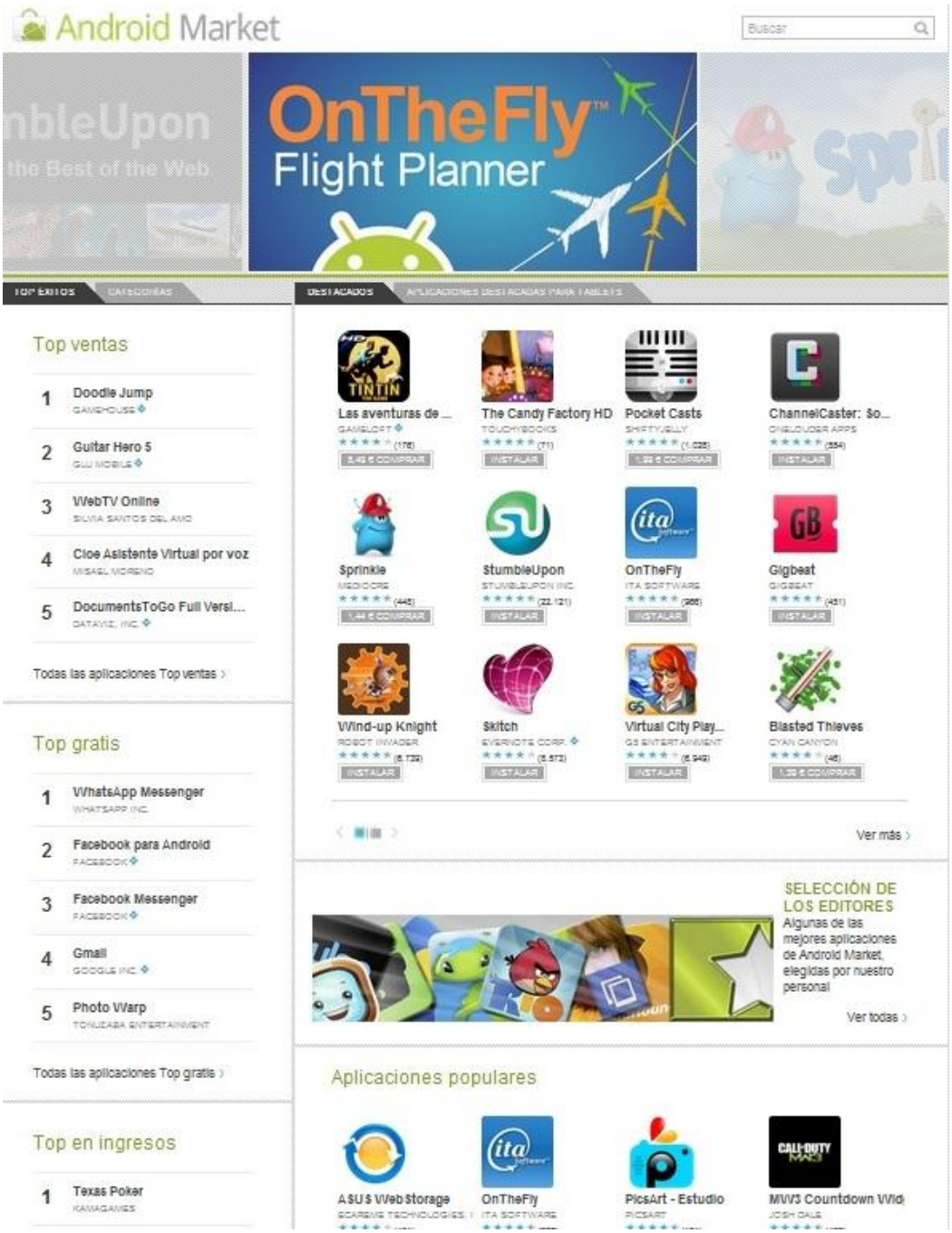


Figura 2. Web oficial de Android Market

En ambas, hay diferentes opciones y las aplicaciones están estructuradas por categorías.La primera opción que presenta es la búsqueda, mediante la cual se puede buscar la aplicación deseada. Una vez introducido el nombre, se mostrarán todas las aplicaciones encontradas, y se podrán clasificar según relevancia, popularidad, precio, etc. A continuación se muestran algunas de las aplicaciones clasificadas por categorias, destacados o aplicaciones top según ventas, nuevas, etc.

PUBLICAR EN ANDROID MARKET

REQUISITOS

Para publicar una aplicación en el Android Market, primero hay que registrarse con una cuenta de desarrollador de Google (Google Checkout 25 $) de acuerdo con los términos del servicio. Se puede registrar como desarrollador en:

http://market.android.com/publish

Una vez registrado, se puede subir la aplicación, actualizar tantas veces como quiera, y publicarla cuando esté lista. Es necesario residir en uno de los países soportados: Australia, Austria, República Checa, Francia, Alemania, Italia, Países bajos, Polonia, Singapur, España, Reino Unido, Estados Unidos. Los requisitos impuestos por el servidor de Android Market son los siguientes:

1. La aplicación debe tener definidos dos atributos en el archivo <manifest>:

   Android:versionCode y android:versionName

2. El atributo versionCode es un número entero creciente que lo utiliza el servidor para identificar internamente la versión de la aplicación y para el manejo de cambios.

3. El atributo versionName muestra a los usuarios la versión de la aplicación. Consiste en una cadena de texto que representa la versión tal y como la verán los usuarios.

4. Además, hay que definir otros dos atributos, android:icon y android:label, dentro del elemento <application> del archivo *manifest*.

5. La aplicación debe estar firmada con una clave criptografica privada

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.prueba"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <activity android:name=".pruebaActivity"
                  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>
```

Figura 3. Requisitos de configuración en Manifiest

PASOS

Se publica la aplicación a través de la herramienta web disponible en el portal de desarrolladores del Android market, cumplimentando los datos necesarios en tres partes:

1. La parte de "**subir recursos**", donde se sube el .apk, las imágenes promocionales, un icono, un gráfico promocional, y un video de youtube (Figura 4).
2. La parte de "**especificación de detalles**" con la descripción de la aplicación en los diferentes idiomas, tipo, categoría, descripción (Figura 5).
3. Y la parte de '**opciones de publicación**' con la protección y clasificación por edades, además de la información de contacto y aceptaciones correspondientes (Figuras 6 y 7).

Una vez publicada la aplicación (opción publish), en breves minutos ya se encontrará en el Market.

Figura 4. Subir recursos

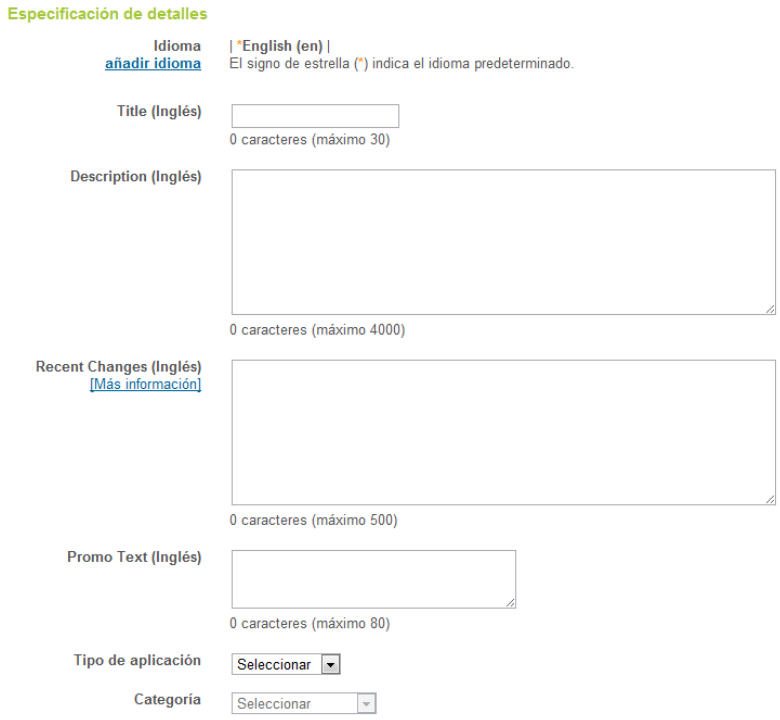


Figura 5. Especificación de detalles

Figura 6. Opciones de publicación

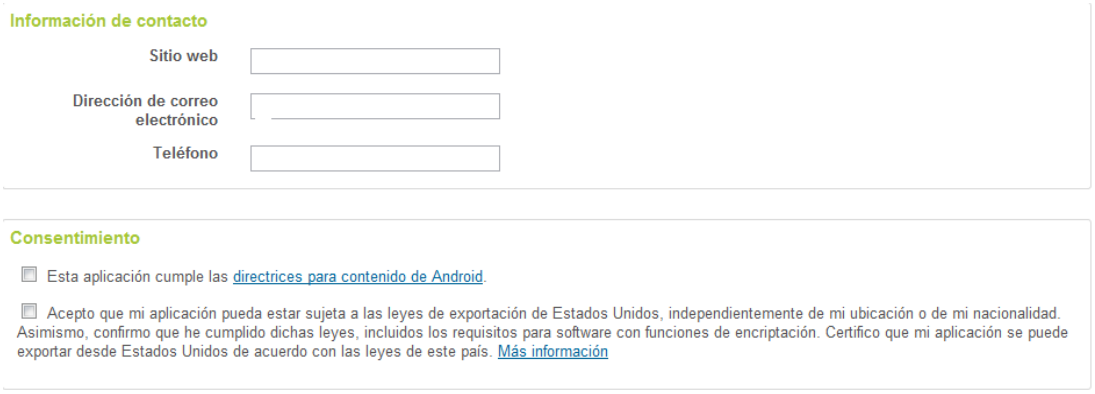


Figura 7. Almacenamiento de información de contacto

FIRMA DIGITAL

Las aplicaciones deben estar firmadas digitalmente para poder distribuirse. Esto es una medida de seguridad, así solo el desarrollador de la aplicación puede modificarla y actualizarla. El certificador puede ser autofirmado, no es necesaria una autoridad certificadora. Existen dos modos para firmar las aplicaciones:

- Debug: mientras desarrollamos. El plug-in ADT se encarga de la firma automáticamente, gestionando claves generadas.
- Release: Crea una clave proporcionando el usuario un password. Para esto se puede utilizar el ADT export wizard desde eclipse.

Firmar aplicaciones desde eclipse a través de export wizard:

Se abre la aplicación que se quiere firmar con eclipse, y en el árbol de directorios se abre el archivo AndroidManifest. Una vez abierto, se abre la primera pestaña llamada *Manifest* y en la sección Exporting hay dos opciones para firmar la aplicación.

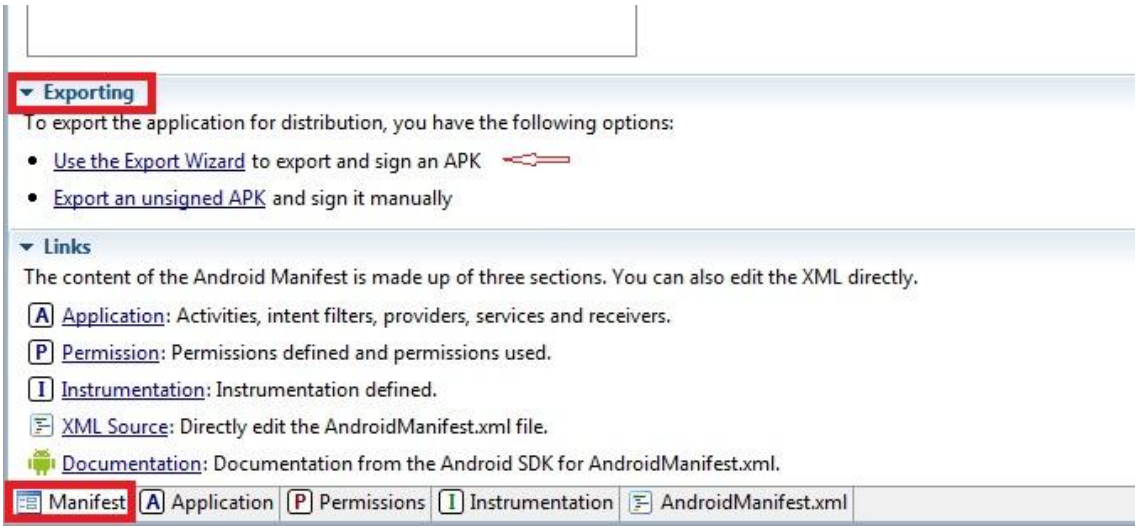


Figura 8. Sección Exporting de Manifiest

Se selecciona la opción Use the Export Wizard.

A continuación, en la siguiente pantalla aparecerá automáticamente el proyecto a firmar, y si no se detecta ningún error, se pulsa en siguiente

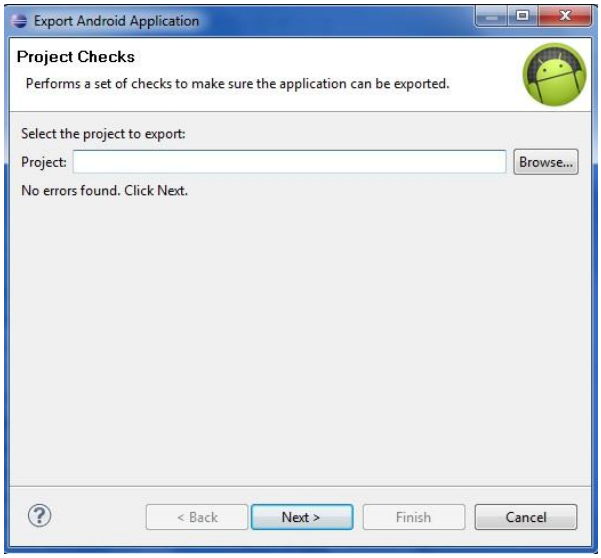


Figura 9. Firma proyecto (1)

Para firmar la aplicación es necesario tener una keystore. Si no se ha creado con anterioridad ninguna, hay que crear una nueva keystore con la opción Create new keystore, rellenando los datos solicitados: Location (directorio donde se guardará la keystore) y password.

Figura 10. Firma proyecto (2)

En la siguiente pantalla, se rellena el formulario de datos de la keystore y del desarrollador de la aplicación.

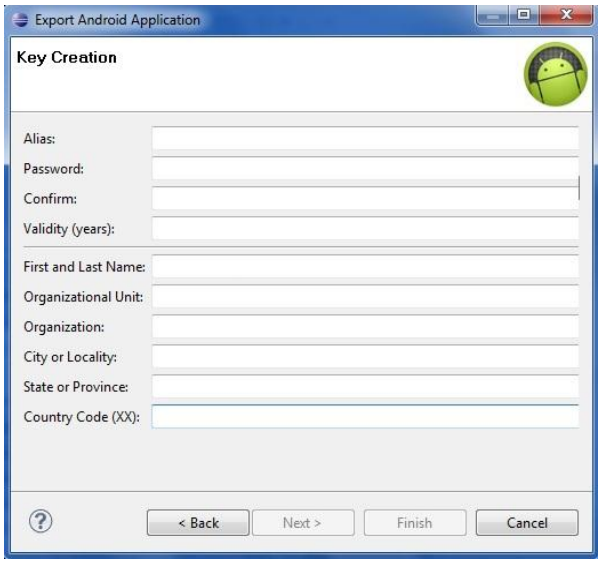


Figura 9. Firma proyecto (3)

Por último, se escoge el directorio en el que se guardara la aplicación ya firmada.

En el caso en el que se quieran firmar más aplicaciones, no será necesario volver a crear una keystore, sino que se podrá utilizar la creada anteriormente, sin necesidad de crear una keystore por aplicación.

ACTUALIZACION

Una vez publicadas las aplicaciones es recomendable actualizarlas corrigiendo errores detectados y añadiéndolas nuevas funcionalidades.

Esto en importante por dos motivos:

- Mantiene la aplicación en las primeras posiciones de la lista de aplicaciones del market, lo que da posibilidad a nuevos usuarios a conocer la aplicación y descargarla.

- Aumenta la confianza los usuarios en el desarrollador, ya que demuestra que tiene en cuenta sus opiniones para mejorar la aplicación.

Para actualizar la aplicación tan solo se debe tener en cuenta que hay que cambiar los atributos android:versionCode y android:versionName.

EL ANDROID MARKET PARA DESARROLLADORES

Cuando se accede al Market con una cuenta de desarrollador, se tienen opciones acerca de las aplicaciones desarrolladas.

Figura 10. Acceso al Market de Android

La Figura 10 muestra dicho acceso. Algunas características y funcionalidades se describen a continuación.

1. **Listado de aplicaciones:** Muestra el número de descargas de cada aplicación, así como el número de descargas activas, que será el número más importante, ya que indica la cantidad de personas que tienen instalada la aplicación (importante si tiene publicidad la aplicación porque aumentará el beneficio).

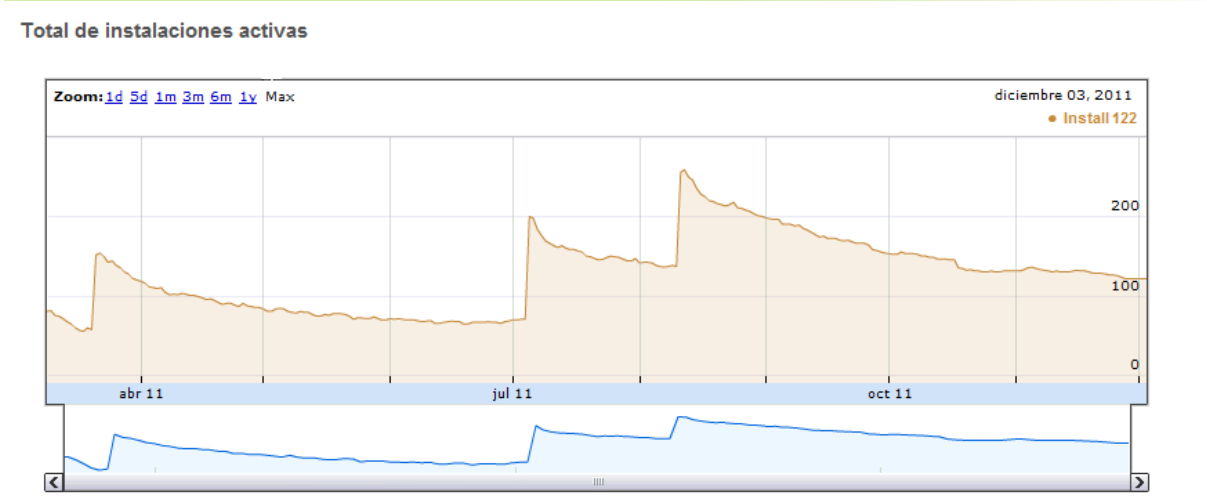


Figura 11. Cargas y descargas sobre las instalaciones activas

En la imagen anterior se observa un gran crecimiento en determinados momentos. Esto es debido a la publicación de actualizaciones de la aplicación, lo que conlleva un aumento del número de descargas al aparecer la aplicación entre las más nuevas, de ahí la importancia de desarrollar actualización de las aplicaciones.

2. **Comentarios:** Para cada aplicación, se puede acceder a las valoraciones y comentarios de los usuarios acerca de esa aplicación.

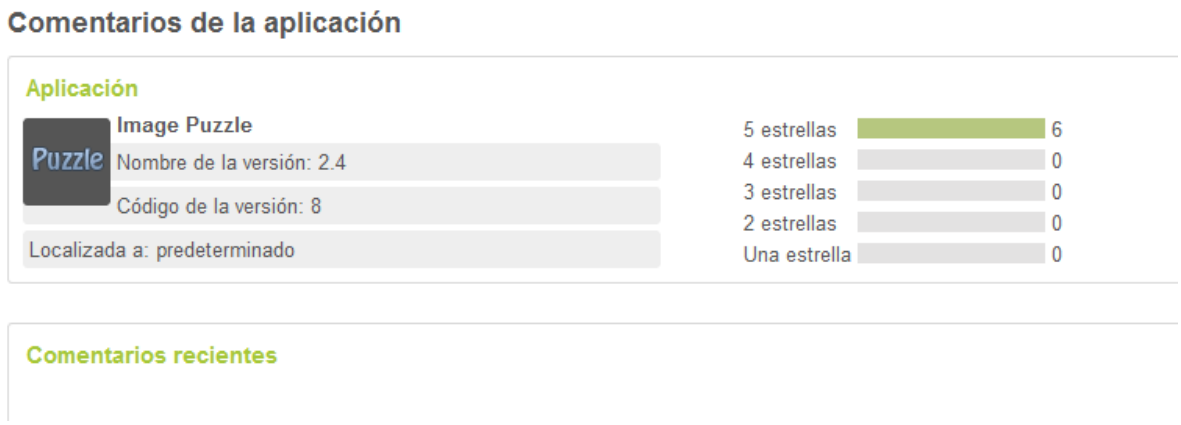


Figura 12. Comentarios de la aplicación

3. **Errores:** Muestra los errores que se producen en la aplicación y son enviados por los usuarios a los desarrolladores. Generalmente se producen porque determinadas partes de la aplicación no han sido debidamente testeadas, o son errores inesperados.

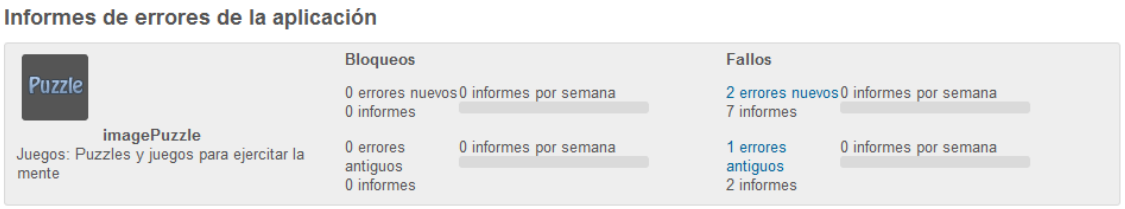
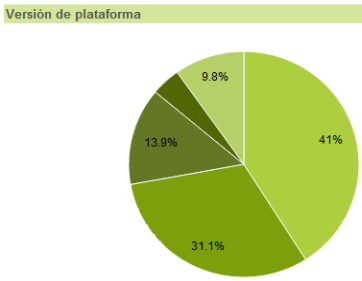


Figura 13. Informe de errores de la aplicación

4. **Disponibilidad de dispositivo:** Indica que dispositivos son compatibles dependiendo del hardware utilizado pudiendo excluir algunos de ellos.

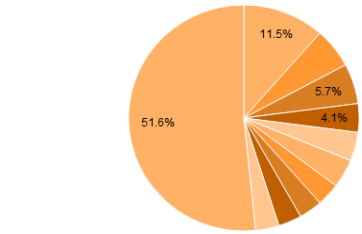
5. **Estadísticas:** Muestra estadísticas actualizadas diariamente de cada aplicación, tales como el número de instancias activas, distribución por versión de Android, por dispositivos, por país o por idioma.



Versión de plataforma

	imagePuzzle	
1	Android 2.2	41,0% (50)
2	Android 2.3.3+	31,1% (38)
3	Android 2.1	13,9% (17)
4	Android 1.6	4,1% (5)
5	Android 3.1	2,5% (3)
6	Android 2.3	0,8% (1)
7	Android 3.2	0,8% (1)
8		
9		
10		

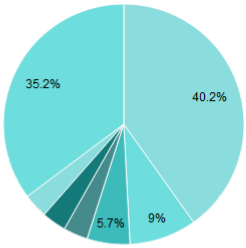
Todas las aplicaciones de mi categoría	
Android 2.3.3+	39,5%
Android 2.2	34,4%
Android 2.1	14,6%
Android 1.6	3,5%
Android 1.5	3,0%
Android 3.1	1,2%
Android 3.2	0,7%
Android 2.3	0,3%
Android 3.0	0,2%
Android 1.1	0,1%



Dispositivo

	imagePuzzle	
1	Samsung Galaxy S (GT-I9000)	11,5% (14)
2	HTC Desire HD	5,7% (7)
3	Samsung Galaxy Tab	5,7% (7)
4	HTC Evo 4G	4,1% (5)
5	Samsung Galaxy S2	4,1% (5)
6	Samsung Galaxy Ace	4,1% (5)
7	Samsung Nexus S	3,3% (4)
8	Motorola Droid II	3,3% (4)
9	Motorola Droid X	3,3% (4)
10	SEMC Xperia X10	3,3% (4)

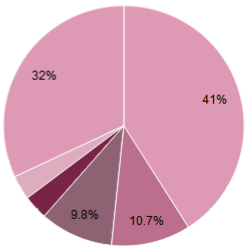
Figura 14. Estadísticas (1)



País

	imagePuzzle	
1	Estados Unidos	40,2% (49)
2	España	9,0% (11)
3	Reino Unido	5,7% (7)
4	Brasil	3,3% (4)
5	Alemania	3,3% (4)
6	Malasia	3,3% (4)
7	Australia	2,5% (3)
8	Arabia Saudí	2,5% (3)
9	Taiwán	2,5% (3)
10	Suiza	2,5% (3)

Todas las aplicaciones de mi categoría	
Estados Unidos	47,5%
Reino Unido	4,8%
Japón	4,8%
Corea del Sur	4,0%
Alemania	3,4%
Francia	3,0%
España	2,1%
Rusia	1,9%
Taiwán	1,9%
Australia	1,8%



Idioma

	imagePuzzle	
1	inglés (Estados Unidos)	41,0% (50)
2	español (España)	10,7% (13)
3	inglés (Reino Unido)	9,8% (12)
4	chino (Taiwán)	3,3% (4)
5	alemán (Alemania)	3,3% (4)
6	ruso (Rusia)	2,5% (3)
7	francés (Francia)	2,5% (3)
8	inglés (Irlanda)	2,5% (3)
9	japonés (Japón)	2,5% (3)
10	portugués (Brasil)	2,5% (3)

Todas las aplicaciones de mi categoría	
inglés (Estados Unidos)	49,1%
inglés (Reino Unido)	7,6%
japonés (Japón)	4,7%
coreano (Corea del Sur)	4,1%
alemán (Alemania)	3,6%
francés (Francia)	3,3%
español (España)	2,4%
chino (Taiwán)	2,3%
ruso (Rusia)	2,2%
español (Estados Unidos)	1,6%

Figura 15. Estadísticas (2)