Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Guía para el desarrollo de aplicaciones para Android

Autor: Frank Andrés Pérez Cou

Tutor: Ing. Arnaldo Moreno Montes de Oca

Santa Clara

2012

"Año 54 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Guía para el desarrollo de aplicaciones para Android

Autor: Frank Andrés Pérez Cou

fperez@uclv.edu.cu

Tutor: Ing. Arnaldo Moreno Montes de Oca

arnaldomm@uclv.edu.cu

Santa Clara

2012

"Año 54 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

Hay dos clases de conocimiento. Podemos conocer un tema por nosotros mismos, o bien conocer dónde encontrar información al respecto.

Samuel Johnson

DEDICATORIA

A mis padres, en especial a mi madre

AGRADECIMIENTOS

A mi familia, en especial a mi madre porque a ella se lo agradezco todo

A mis amigos por apoyarme todo el tiempo

Al Rafa y al Fongo por ser mis hermanos

A Susana por darme más confianza de la que realmente esperaba tener

A Carlos por ayudarme todo el tiempo con el acceso a internet

TAREA TÉCNICA

- Revisión bibliográfica relacionada al tema de las aplicaciones para el sistema operativo Android.
- Estudio y selección de las herramientas adecuadas para la confección de aplicaciones para el sistema operativo Android.
- Confección de una guía para el desarrollo de aplicaciones para el sistema operativo Android.
- Desarrollo de aplicaciones demostrativas para el sistema operativo Android.
- Confección y presentación de un informe final.

Firma del Autor

Firma del Tutor

RESUMEN

Actualmente el área de las comunicaciones móviles ocupa un lugar importante dentro del desarrollo de la sociedad por lo que las investigaciones sobre este sector de la tecnología es de importancia para los ingenieros. En Cuba las investigaciones en el campo de las comunicaciones móviles son escasas. En el caso particular de la Facultad de Ingeniería Eléctrica de la Universidad Central “Marta Abreu” de Las Villas no se cuenta con investigaciones dedicadas a esta rama del saber aun contando con profesionales y estudiantes de la rama de las telecomunicaciones.

Por lo anteriormente dicho en este trabajo se propuso suplir parcialmente esta necesidad a través de la confección de una guía para el desarrollo de aplicaciones para el sistema operativo móvil Android, en la que se especificaran cuestiones como el entorno de desarrollo óptimo para esta finalidad en un ámbito docente y los pasos elementales para la programación de aplicaciones de baja complejidad. Con la confección y análisis de aplicaciones de baja complejidad se obtuvo un material adecuado para la consulta de profesores y estudiantes interesados en desarrollar sus conocimientos y habilidades en este campo, que se recomienda ser incluida como documento de consulta bibliográfica.

Índice

INTRODUCCIÓN

El constante avance de la electrónica ha propiciado, entre otras cosas, la disminución del tamaño de los dispositivos personales aumentando con esto la comodidad para los usuarios que los utilizan. Lo anterior unido al desarrollo de las redes inalámbricas ha puesto en manos de los usuarios actuales los llamados dispositivos móviles, los cuales han evolucionado hasta llegar a los llamados *Smartphones*(Teléfonos inteligentes) y las *Tablet PC*.

Estos dispositivos poseen numerosas utilidades entre las que sobresalen las orientadas a las comunicaciones como el correo electrónico, los mensajes de texto, el servicio telefónico y el acceso a internet, tan necesario para insertar a los usuarios en el creciente flujo mundial de información.

Todos estos dispositivos funcionan con sistemas operativos especiales denominados sistemas operativos móviles, los cuales son más sencillos que sus parientes de computadores personales ya que están orientados básicamente a los servicios de conexión inalámbrica.

La enorme competencia que existe en el mercado actual de los dispositivos móviles ha propiciado la aparición de diversos sistemas operativos móviles, cada uno de los cuales ha ido evolucionando para agregar utilidades a los dispositivos que los utilizan, entre los que sobresalen por la cantidad de usuarios que los emplean: Symbian, WindowPhone, iOS y Android.

Android es un sistema operativo desarrollado por la Open Handset Alliance, la cual es liderada por Google. El detalle más importante de este sistema operativo es que, a

diferencia de otros, se desarrolla con código abierto ya que está basado en el *kernel* de Linux. Esto abre la posibilidad de programar y modificar a gusto el sistema operativo.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un *framework* Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución.

Android cuenta con una gran comunidad de desarrolladores que programan aplicaciones para extender la funcionalidad de los dispositivos y ha contado con numerosas actualizaciones desde su liberación inicial en el año 2008.

En la Facultad de Ingeniería Eléctrica (FIE) de la Universidad Central “Marta Abreu” de Las Villas no existe material para el desarrollo de aplicaciones en este sistema operativo, aun contando con el personal calificado y los medios técnicos para hacerlo.

El objetivo general de este trabajo es crear una guía para el desarrollo de aplicaciones para Android y de esta forma dotar a la FIE de un material que sirva para la iniciación de los estudiantes en este perfil.

Para el logro del objetivo general se plantean los siguientes objetivos específicos: seleccionar las herramientas adecuadas para la confección y emulación de aplicaciones para Android aptas para el aprendizaje, confeccionar aplicaciones que ejemplifiquen el uso de estas herramientas, y describir los pasos básicos para la utilización de estas herramientas.

Organización del informe

El informe de la investigación consta de tres capítulos.

Capítulo 1: Sistemas Operativos Móviles.Presenta una caracterización sobre los sistemas operativos en general.Luego particulariza en los sistemas operativos móviles hasta llegar finalmente al sistema operativo Android, tema principal de esta investigación. Se analizan y diferentes entornos de desarrollo y se selecciona el más adecuado según una serie de criterios.

Capítulo 2: Confección de Aplicaciones.Constituye la guía para programar aplicaciones para Android, en la cual se especifican las acciones básicas para la configuración del entorno de desarrollo de Android, los pasos para crear un proyecto y su estructura, así como los elementos básicos de código para la confección de las aplicaciones.

Capítulo 3: Aplicaciones Desarrolladas.Se muestran aplicaciones de pequeña complejidad que ejemplifican el uso de la guía desarrollada enel Capítulo 2, mediante el análisis de los elementos principales del código de dichas aplicaciones.

CAPÍTULO 1. SISTEMAS OPERATIVOS MÓVILES

En este capítulo se abordan los aspectos teóricos más importantes de los sistemas operativos, en especial de los sistemas operativos móviles. Debido a la gran variedad de sistemas operativos móviles que existen, en este capítulo solo se mencionan los más conocidos.

Además se dedica un epígrafe al estudio del sistema operativo Android, en el cual se menciona la historia, características y estructura del sistema operativo; de forma que pueda contribuir a una mejor comprensión del mismo.

1.1.Sistemas operativos

Un sistema operativo (SO) es un programa que controla el hardware de un computador. Además provee las bases para la programación de aplicaciones y actúa como intermediario entre el usuario y el hardware del computador. El objetivo principal de un sistema operativo es lograr que el sistema de computación se use de manera cómoda y el objetivo secundario es que el hardware del computador se emplee de manera eficiente [**1**].

Uno de los propósitos de un sistema operativo consiste en gestionar los recursos de localización y protección de acceso al hardware, aliviando a los programadores de aplicaciones de tener que tratar con estos detalles.

La mayoría de los equipos electrónicos que utilizan microprocesadores para funcionar, llevan incorporado un sistema operativo. En cada caso manejado por una interfaz gráfica de usuario; un gestor de ventanas o un entorno de escritorio si es un celular, mediante una consola o un control remoto si es un DVD, o mediante una línea de comandos o un navegador web si es un enrutador [**2**].

1.1.1. Servicios de un sistema operativo

Un sistema operativo ofrece ciertos servicios para favorecer la tarea del programador o para hacer eficiente el funcionamiento de la máquina.

Gestión de Procesos:

Un proceso es un programa en ejecución y toda la información necesaria para ejecutar el programa [**3**]. El sistema operativo es el encargado de crear y distribuir los procesos, parar y reanudar los procesos y ofrecer mecanismos para que se comuniquen y sincronicen.

Gestión de memoria principal:

El sistema operativo es el responsable de conocer que partes de la memoria están siendo utilizadas y por quien, decidir que procesos se cargaran en memoria cuando haya espacio disponible y asignar y reclamar espacio de memoria cuando sea necesario.

Gestión de almacenamiento secundario:

Un sistema de almacenamiento secundario es necesario, ya que la memoria principal (almacenamiento primario) es volátil y además muy pequeña para almacenar todos los programas y datos. También es necesario mantener los datos que no convenga mantener en la memoria principal.

El sistema operativo se encarga de gestionar el espacio libre, asignar el almacenamiento y verificar que los datos se guarden en orden.

Gestión de Archivos:

El sistema operativo es responsable de construir y eliminar archivos y directorios, ofrecer funciones para manipular archivos y directorios y establecer la correspondencia entre archivos y unidades de almacenamiento.

Operaciones de entrada/salida:

Como un programa no puede acceder directamente a un dispositivo de E/S el sistema operativo debe facilitarle algunos medios para realizarlo. Para ello, el sistema nos provee de ciertas funciones genéricas para su acceso [**4**].

Detección de errores:

El sistema operativo necesita constantemente detectar posibles errores. Los errores pueden producirse en la CPU y en el hardware de la memoria, en los dispositivos de E/S o bien en el programa de usuario. Para cada tipo de error, el sistema operativo debe adoptar la iniciativa apropiada que garantice una computación correcta y consistente [**5**].

1.1.2. Sistemas operativos actuales

En la actualidad existe una gran variedad de sistemas operativos. Los más conocidos son los sistemas operativos para computadores personales (PC) producto de su amplia difusión en décadas pasadas. Entre estos destacan Windows, Linux y Mac OS.

Mac OS es el nombre del sistema operativo creado por la empresa Apple en el año 1984 para su línea de computadoras Macintosh. Es conocido por haber sido el primer sistema dirigido al gran público en contar con una interfaz gráfica compuesta por la interacción del *mouse* con ventanas, iconos y menús. Destaca por su facilidad de uso y su multitarea cooperativa [**6**].

Microsoft Windows es el nombre de una familia de sistemas operativos desarrollados por Microsoft desde 1981. Microsoft comercializó por primera vez el entorno operativo denominado Windows en noviembre de 1985 como complemento para el sistema operativo MS-DOS y con el transcurso de los años llegó a dominar casi la totalidad del mercado de sistemas operativos a nivel mundial [**7**].

Linux es un sistema operativo para PC's compatible con sistemas Unix. Fue escrito por LinusTorvalds hacia 1991, basándose en otro sistema operativo cuya misión principal era la de enseñar a los estudiantes los principios de éstos sistemas. Es un sistema operativo diseñado por y para programadores aunque cada día son más los usuarios que lo utilizan.

Linux puede funcionar tanto en entorno gráfico como en modo consola. La consola es común en distribuciones para servidores, mientras que la interfaz gráfica está orientada al usuario final tanto de hogar como empresarial. Asimismo, también existen los entornos de escritorio conformado por ventanas, iconos y muchas aplicaciones que facilitan la utilización del computador [**8**].

Estos sistemas operativos tienen un papel imprescindible en el mundo de los dispositivos móviles pues son la base de una serie de sistemas operativos especiales que se desarrollan para estos dispositivos llamados sistemas operativos móviles.

1.2.Sistemas operativos móviles en la actualidad

Un sistema operativo móvil es un sistema operativo que controla un dispositivo móvil al igual que las computadoras personales utilizan Windows o Linux. La diferencia principal radica en que los sistemas operativos móviles son mucho más simples. Estos están orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos [**9**].

Los sistemas operativos móviles se dividen en cuatro capas. La primera capa se denomina *kernel* o núcleo. Esta proporciona el acceso a los distintos elementos del hardware del dispositivo. Esta capa ofrece servicios a las capas superiores como los controladores para el hardware, la gestión de procesos, el sistema de archivos y la gestión de memoria. La segunda capa se denomina *middleware*. Esta es el conjunto de módulos que hacen posible la propia existencia de las aplicaciones para móviles. Esta capa es transparente al usuario y ofrece servicios como el motor de mensajería, códecs multimedia, gestión del dispositivo y seguridad. La tercera capa es el entorno de ejecución de aplicaciones. Esta consiste en un gestor de aplicaciones y un conjunto de interfaces programables abiertas para facilitar la creación de software. Por último está la capa de interfaz de usuario la cual facilita la interacción con el usuario y la presentación visual de cada aplicación. Esta capa ofrece servicio de componentes gráficos y el de marco de interacción [**9**].

1.2.1. Sistemas operativos móviles más utilizados

En el mundo existen varios sistemas operativos móviles que compiten entre sí para dominar la gran demanda que tienen hoy en día los celulares y las *tablet PC*. Entre los más utilizados se encuentran *Windows Phone*, BlackBerry OS, iPhone OS, Symbian y Android, siendo este último el más difundido.

- **BlackBerry OS**

El BlackBerry OS 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 computadoras de mano, particularmente los *trackwheel*, *trackball*, *touchpad* ypantallas táctiles.

El BlackBerry OS está claramente orientado a su uso profesional como gestor de correo electrónico y agenda. Este sistema permite sincronizar el dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de Microsoft Exchange Server además es compatible también con Lotus Notes y Novell GroupWise.

BlackBerry Enterprise Server (BES) proporciona el acceso y organización del email a grandes compañías identificando a cada usuario con un único BlackBerry PIN. Los usuarios más pequeños cuentan con el software BlackBerry Internet Service, programa más sencillo que proporciona acceso a Internet y a correo POP3 / IMAP / Outlook Web Access sin tener que usar BES.

Aparte de los dispositivos de la propia marca, otras marcas utilizan el cliente de correo electrónico de BlackBerry: Siemens, HTC y Sony Ericsson. La mayoría de estos dispositivos cuentan con teclado QWERTY completo.

Al igual que en el sistema operativo (SO) Symbian, desarrolladores independientes también pueden crear programas para BlackBerry pero en el caso de querer tener acceso a ciertas funcionalidades restringidas necesitan ser firmados digitalmente para poder ser asociados a una cuenta de desarrollador de RIM [**10**].

- ***Windows Phone***

Windows Phone, anteriormente llamado *Windows Mobile***,** es un sistema operativo móvil compacto desarrollado por *Microsoft*, y diseñado para su uso en teléfonos inteligentes (*Smartphones*) y otros dispositivos móviles. *Windows Phone* hace parte de los sistemas operativos con interfaz natural de usuario.

Se basa en el núcleo del sistema operativo *WindowsCE* y cuenta con un conjunto de aplicaciones básicas utilizando las interfaces de programación de aplicaciones (API) de *Microsoft Windows*. Está diseñado para ser similar a las versiones de escritorio de *Windows* estéticamente. Además, existe una gran oferta de software de terceros disponible para *Windows Phone*, la cual se puede adquirir a través de *Windows Marketplacefor Mobile*. En la actualidad, la mayoría de los teléfonos con *Windows Phone* vienen con

un puntero, que se utiliza para introducir comandos pulsando en la pantalla. *Windows Phone* ha evolucionado y cambiado de nombre varias veces durante su desarrollo [**10**].

- **Symbian OS**

Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia, Sony Ericsson, PSION, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic y Sharp. Sus orígenes provienen de su antepasado EPOC32, utilizado en PDA's y Handhelds de PSION.

El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Windows Mobile 6.X de Microsoft y ahora Android de Google Inc. y iOS de Apple Inc.

En la actualidad Symbian se mantiene entre los líderes del mercado pero con una tendencia al descenso ante el auge de Android [**11**].

- **iOS**

iOS, anteriormente denominado iPhone OS, es un sistema operativo móvil de Apple. Originalmente fue desarrollado para el iPhone, siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV.

iOS se deriva de MacOS X, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo Unix.

Este sistema operativo tiene una interfaz de usuariobasada 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. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan a sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotarlo en tres dimensiones (un resultado común es cambiar de modo vertical al apaisado o horizontal).

En la actualidad es uno de los sistemas más utilizados en todo el mundo.En mayo de 2010 en los Estados Unidos, tenía el 59% de consumo de datos móviles (incluyendo el iPod Touch y el iPad) [**12**].

Tenía el 26% de cuota de mercado de sistemas operativos móviles vendidos en el último cuatrimestre de 2010, detrás de Google Android y Nokia Symbian [**13**].

De todos los sistemas operativos móviles que existen en la actualidad, Android es el más utilizado en el mundo. En esto influye que Android es un sistema que se desarrolla de forma abierta por lo que es muy fácil de personalizar y de actualizar. En el próximo epígrafe se hará un análisis más profundo de la historia y la actualidad de este sistema operativo.

1.3.Sistema operativo Android

Android es un sistema operativo para dispositivos móviles como teléfonos inteligentes y tabletas. Es desarrollado por la Open Handset Alliance, la cual es liderada por Google.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un *framework* Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución.

Esta plataforma permite a los desarrolladores escribir códigos en Java que se ejecuten en móviles mediante las librerías Java desarrolladas por Google [**14**].

Uno de los detalles a destacar, es que el sistema operativo Android es diferente a otros como el iOS o el *Windows Phone*. Esto debido a que se desarrolla de forma abierta y se puede acceder tanto al código fuente como al listado de incidencias, con lo cual puedes verificar los problemas aún no resueltos y reportar los nuevos.

El nombre Android (androide en español) hace alusión a la novela de Philip K. Dick *¿Sueñan los androides con ovejas eléctricas?*, que posteriormente fue adaptada al cine como *BladeRunner*. El logotipo es el robot Andy [**15**].

1.3.1. Historia

En julio de 2005, Google adquirió Android Inc., una pequeña compañía de Palo Alto, California fundada en 2003. En aquel entonces, poco se sabía de las funciones de Android

Inc. fuera de que desarrollaban software para teléfonos móviles [**16**]. Esto dio pie a rumores de que Google estaba planeando entrar en el mercado de los teléfonos móviles. Entre los cofundadores de Android que se fueron a trabajar a Google están Andy Rubin, RichMilner, Nick Sears y Chris White.

En Google, el equipo liderado por Rubin desarrolló una plataforma para dispositivos móviles basada en el *kernel*de Linux que fue promocionado a fabricantes de dispositivos y operadores con la promesa de proveer un sistema flexible y actualizable. Se informó que Google había alineado ya una serie de fabricantes de hardware y software y señaló a los operadores que estaba abierto a diversos grados de cooperación por su parte [**17**].

La especulación sobre que el sistema Android de Google entraría en el mercado de la telefonía móvil se incrementó en diciembre de 2006. Reportes de BBC y *The Wall Street Journal* señalaron que Google quería sus servicios de búsqueda y aplicaciones en teléfonos móviles y estaba muy empeñado en ello. Medios impresos y en línea pronto reportaron que Google estaba desarrollando un teléfono con su marca.

El anuncio del sistema Android se realizó el 5 de noviembre de 2007 junto con la creación de la Open Handset Alliance, un consorcio de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles [**18**]. Google liberó la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto [**19**].

1.3.2. Actualizaciones

Android ha visto numerosas actualizaciones desde su liberación inicial. Estas actualizaciones al sistema operativo base típicamente arreglan errores y agregan nuevas funciones. Generalmente cada actualización del sistema operativo Android es desarrollada bajo un nombre en código de un elemento relacionado con postres. Estas versiones difieren en las mejoras y la interfaz que presentan.

Android se inició oficialmente a finales de 2008 en Estados Unidos, cuando fue lanzado el primer teléfono con Android a bordo, el HTC *Dream* G1 de T-Mobile.

La ventana de notificación desplegable fue la gran novedad que introdujo Android, apostó desde el principio por un sistema de notificación con el que tener toda la información a la

vista y el tiempo le ha dado la razón ya que ha sido copiado por iOS en su última versión. Android 1.0 supuso la mejor experiencia de correo en el móvil que había en el mercado gracias al apoyo de Gmail a POP e IMAP. Era difícil imaginar un *smartphone* sin tienda de aplicaciones, el primer Android Market salió sin apenas aplicaciones y con un diseño de una sola fila ubicada en la parte superior de la pantalla de inicio de la aplicación.

En febrero de 2009 llegó la primera actualización para Android, unos tres meses después del lanzamiento del G1. La versión 1.1 fue dedicada básicamente a reparar errores.

En abril de 2009 es liberada la actualización 1.5 para Android. Conocida como *Cupcake*, fue la primera versión en utilizar nombre de postres. Cada versión después de *Cupcake* ha sido nombrada con un nombre de postre continuando el orden alfabético.

En esta versión se comenzaron a ver algunos cambios en la interfaz de usuario, por poco que se puedan apreciar como son los cambios de en la barra del buscador y en la barra del menú, también cambió el logo del navegador. Las primeras versiones de Android no contaban con un teclado virtual, ya que el G1 disponía de un teclado físico, en la 1.5 se introdujo el teclado virtual coincidiendo con la salida del primer Android con pantalla táctil y sin teclado físico, el HTC *Magic* [**20**].

Google no se había dado cuenta todavía del potencial de los *widgets* ya que hasta ese momento no había lanzado el SDK para desarrolladores y había muy poca variedad. Eso cambió en *Cupcake* permitiendo que *widgets* de terceros estuvieran disponibles para los usuarios. La plataforma no ofrecía desde el principio la posibilidad de copiar y pegar en las ventanas del navegador. Se integró por primera vez en esta versión y fue terminada en posteriores versiones. La primera versión de Android no ofrecía la posibilidad de grabar vídeo ni de reproducción, si no que se integró con la 1.5. Los fabricantes comenzaron a crear sus propias interfaces para la cámara añadiendo soporte para escenas adicionales, modos, opciones y la posibilidad del enfoque presionando sobre la pantalla táctil [**21**].

En septiembre de 2009 se libera la actualización 1.6 para Android, conocida por *Donut*. Con su llegada vino el soporte para redes CDMA haciendo que Android llegara a Estados Unidos y Asia. Pero tal vez la mejora más significativa fue la posibilidad de correr el sistema operativo en múltiples resoluciones de pantalla y relaciones de aspecto, a raíz de

esta actualización se puede disfrutar hoy en día de pantallas con resolución QVGA, HVGA, WVGA, FWVGA, QHD y 720p.

Donut también introdujo la búsqueda rápida, generalmente conocida en el mundo del móvil como búsqueda universal. Antes de *Donut* la búsqueda se limitaba a Internet pero con las mejoras introducidas en la versión 1.6 se podría buscar además contenido propio del teléfono. Todo ello desde el mismo *widget* de búsqueda.

Un año después del estreno del G1, a principios de noviembre de 2009 fue lanzado Android 2.0 *Eclair*. Fue ofrecido en exclusiva con Verizon y el Motorola Droid, un teléfono que marcó un antes y después para Android y con el que Motorola volvió a ser la gran marca que fue. Por primera vez se podrían añadir varias cuentas en el mismo dispositivo con acceso al correo electrónico y a los contactos de cada una, además también se introdujo soporte para cuentas de *Exchange*. También se abrió la puerta de las sincronizaciones automáticas para los contactos gracias a la información compartida entre los tipos de cuenta; Facebook fue la primera en integrar esta funcionalidad. *Google MapsNavigation* fue publicado junto con la versión 2.0 y fue un paso adelante parar integrar un sistema de navegación de automóviles en el móvil con vistas en 3D, guía de voz e información de tráfico de forma completamente gratuita. *Eclair* agregó una barra de contacto rápido, una barra de herramientas desplegable que se utiliza para realizar múltiples funciones de manera rápida [**21**].

A la versión *Eclair* se le lanzó una mejora 2.1 en enero de 2010 que mantuvo el nombre. Android 2.1 cambió la pantalla de bloqueo y la hizo más al estilo iOS.

La nueva versión de Android se le llamo *Froyo*. Lanzado a mediados de 2010 trajo una gran cantidad de cambios. La pantalla de inicio fue rediseñada, se ampliaron los 3 paneles existentes desde el inicio a 5 con un nuevo grupo de accesos directos dedicados y se agregaron unos puntos para tener un mejor control de la pantalla. *Froyo* también introducía una galería completamente rediseñada con imágenes en 3D que aparecen al inclinar el teléfono. Se mejoró también el soporte para copiar y pegar en Gmail. En esta versión se agregó la posibilidad de poner una contraseña o PIN en la pantalla de bloqueo para los usuarios que no les gustaba el patrón de desbloqueo [**21**].

Un año y medio después del lanzamiento de *Froyo*, Google lanzó la nueva versión del sistema operativo, Android 2.3. Esta es conocida por *Gingerbread*. Fue una actualización menor en muchos sentidos pero trajo algunos cambios importantes en la interfaz de usuario. Se añadió en esta versión la posibilidad de seleccionar el texto que se desea copiar y pegar. Anteriormente solo se podía copiar el contenido de las cajas completas. Nuevamente Google pone su empeño en mejorar el teclado, cambios en el diseño y de coloración además del soporte *multitouch*. *Gingerbread* fue la primera versión en integrar soporte para varias cámaras, aunque la opción de *videochat* en Google Talk no llegaría hasta mediados de 2011. La nueva versión dio más libertad a los desarrolladores para poder escribir código más rápido y desarrollar juegos con gráficos en 3D que hasta entonces no disponía Android. Google estaba perdiendo la batalla de los juegos con iOS y tenía que reaccionar.

La versión de Android para *tablets* se le llamó *Honeycomb*. Cambió de color, del verde típico de Android al azul que se utilizó para la batería, el *widget* del reloj, indicadores de señal y algunas otras características de la interfaz [**21**].

Se integró una barra en la parte inferior de la pantalla con una serie de botones virtuales que hacen que no se necesiten botones dedicados. Aparece un nuevo botón virtual de Aplicaciones recientes en la parte inferior de la pantalla en la que podemos ver una lista de las últimas aplicaciones utilizadas con capturas de pantalla de las mismas. Android 3.1 y 3.2 fueron versiones de mantenimiento, prueba de ello es que Google no las renombró y continuaron llamándolas *Honeycomb*. Aunque algunas mejoras introducidas en estas actualizaciones se han ido implementando en la mayoría de *tablets* con Android 3.0 del mercado, como la posibilidad de modificar el tamaño de los *widgets* al presionar sobre ellos [**21**].

La última versión de Android es conocida como *Ice CreamSandwish*. La versión 4.0 del sistema operativo tomó prestadas muchas características de *Honeycomb* como los botones virtuales o la transición de tonos verdes a azules, la multitarea con una lista desplegable de miniaturas y las barras de acción dentro de las aplicaciones. Por primera vez se modificó el tipo de letra. *Droid* fue la fuente utilizada desde la versión 1.0 y ahora se modifica por *Roboto*, una fuente que ha sido diseñada para aprovechar la mayor resolución de las pantallas de hoy en día. El teclado virtual también se modificó y esta vez incluye un

sistema de corrección mucho más avanzado que subraya en color rojo las palabras mal escritas e incorpora también un diccionario. La pantalla de notificaciones también recibió una pequeña actualización con las notificaciones individuales extraíbles que permiten deslizar cualquier notificación fuera de la pantalla y acceder a ella. La pantalla adoptó muchos cambios de los que se introdujeron en *Honeycomb* pero añade además algunas características nuevas como la posibilidad de crear carpetas con solo arrastrar un icono a otro. Además la pantalla principal recibe una bandeja de favoritos que puede ser configurada por el usuario. En *Ice CreamSandwich* se busca potenciar el uso de *Near Field Communication* (NFC) con una nueva característica para transferencia de datos entre dos teléfonos con solo tocarlos. Además del bloqueo con contraseña y con patrón de desbloqueo se agregó la opción del desbloqueo facial. Se añadió un gestor para el uso de los datos en el que se informa de las aplicaciones que consumen más datos, se puede ver el uso total desglosado en un periodo de tiempo configurable por el usuario [**21**].

1.3.3. Cuotas de mercado

La compañía de investigación de mercado Canalys estima que en el segundo trimestre de 2009, Android tendría 2,8% del mercado de teléfonos inteligentes a nivel mundial [**22**].

En febrero de 2010, *ComScore* dijo que la plataforma Android tenía el 9% del mercado de teléfonos inteligentes en los Estados Unidos, como estaba tasado por los operadores. Esta cifra fue superior al estimado anterior [**23**]de noviembre de 2009, el cual fue del 9%. Para finales del tercer trimestre de 2010, el mercado de Android en los Estados Unidos había crecido en un 21,4% [**24**].

En mayo de 2010, Android superó en ventas a iPhone, su principal competidor. De acuerdo a un informe del grupo NPD, Android obtuvo un 28% de ventas en el mercado de los Estados Unidos, un 8% más que en el trimestre anterior. En el segundo trimestre de 2010, los dispositivos iOS incrementaron su participación en un 1%, indicando que Android está tomando mercado principalmente de RIM. Adicionalmente, los analistas apuntaron que las ventajas de que Android fuera un sistema multi-canal, multi-operador, le permitiría duplicar el rápido éxito que obtuvo el sistema Windows Mobile de Microsoft [**25**].

A principios de octubre de 2010, Google agregó 20 países a su lista de lugares geográficos donde los desarrolladores pueden enviar aplicaciones. Para mediados de octubre, la compra de aplicaciones estaba disponible en un total de 32 países [**26**].

En diciembre de 2011 Andy Rubin dijo que se activaban 700.000 dispositivos diariamente, anteriormente en julio de 2011 se declaró que se activan unos 550.000 dispositivos Android cada día en comparación con diciembre de 2010 que se activaban 300.000 dispositivos móviles con Android, y los 100.000 que se activaban en mayo de 2010 [**27**].

Estos datos demuestran que este sistema operativo se ha impuesto en muy poco tiempo en un mercado donde existe una gran cantidad de competidores llegando a convertirse el sistema operativo más utilizado en el mundo.

1.3.4. Arquitectura de Android

En la Figura1.1 se muestra la arquitectura del sistema operativo Android.

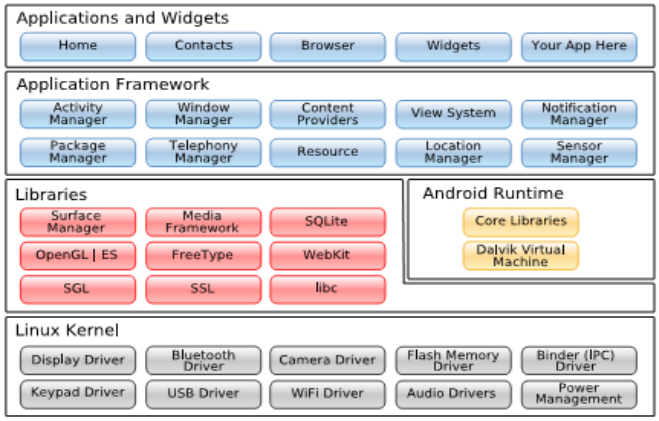

Figura 1.1: Arquitectura de Android

En la base tenemos el *kernel* de Linux. Android lo utiliza por su robustez demostrada y por la implementación de funciones básicas para cualquier sistema operativo como son: seguridad, administración de memoria y procesos, implementación de conectividad de red (*networkstack*) y *drivers* para comunicación con los dispositivos físicos.

Además de todo lo ya implementado en el *kernel* de Linux, Android agrega algunas cosas específicas para plataformas móviles como la comunicación entre procesos (lograda a través del *binder*), la forma de manejar la memoria compartida (*ashmem*) y la administración de energía (con *wakelocks*) [**28**].

Sobre el *kernel*, tenemos un conjunto de librerías de C y C++ utilizadas por el sistema para varios fines como el manejo de la pantalla, mapas de bits y tipos de letra, gráficas en 2D y 3D (SGL y OpenGL), manejo de multimedia, almacenamiento de datos (SQLite) y un motor para las vistas web y el navegador (WebKit).

Junto a estas librerías, encontramos lo necesario para la ejecución de las aplicaciones a través de la máquina virtual Dalvik. Cada aplicación utiliza una instancia de la máquina virtual ejecutando un archivo DEX (*DalvikExecutable*) y el sistema está optimizado para que se ejecuten múltiples instancias de la máquina virtual. Se desarrolla en Java pero no se utiliza una máquina virtual de Sun para su ejecución ni tampoco archivos CLASS [**28**].

Sobre las librerías encontramos una estructura que nos brinda un contexto para desarrollar. Este *framework* permite a los desarrolladores aprovechar un sistema de vistas ya construido, administrar notificaciones y accesar datos a través de proveedores de contenido entre otras cosas [**28**].

La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

Por último tenemos la capa de aplicaciones. Las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros.

1.3.5. Componentes básicos de una aplicación Android

Los componentes principales de una aplicación Android son: *activities*, *intents*, *widgets*, *views*, *services*, *contentproviders* y *broadcast receivers*.

Activity

Las actividades representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana en cualquier otro lenguaje visual [**29**].

Intent

Un *intent* es el elemento básico de comunicación entre los distintos componentes Android. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un *intent* se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje *broadcast*, iniciar otra aplicación, etc [**29**].

Widget

Los *widgets* son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

View

Los objetos *view* son los componentes básicos con los que se construye la interfaz gráfica de la aplicación [**30**]. Diferentes vistas pueden agruparse a través de grupos logrando una jerarquía, esto se logra a través de la disposición de los componentes a través de un archivo XML [**31**].

Android pone a disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear controles personalizados [**29**].

Service

Los servicios son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. *activities*) si se necesita en algún momento la interacción con del usuario [**29**].

ContentProvider

Un *contentprovider* es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los *contentprovider* que se hayan definido [**29**].

BroadcastReceiver

Un *broadcastreceiver* es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: "Batería baja", "SMS recibido", "Tarjeta SD insertada") o por otras aplicaciones [**29**]. Aunque no muestran una interfaz de usuario algunas veces utilizan barras de progreso para mostrar avances [**28**].

1.3.6. Entorno de desarrollo para Android

En la actualidad existen varias herramientas con las que se pueden desarrollar aplicaciones para el SO Android. La principal parte de la comunidad mundial de desarrolladores de aplicaciones emplea una de dos herramientas, Eclipse o NetBeans, siendo por esto las más difundidas y utilizadas mundialmente. Por lo anterior, se compararon estas herramientas y se decidió emplear una de las dos.

NetBeans es un entorno integrado de desarrollo que simplifica algunas de las tareas que, sobre todo en proyectos grandes, pueden ser un poco tediosas. Este asiste parcialmente en la escritura de código. Brinda ayuda en la navegación de las clases predefinidas en la plataforma pero su aprendizaje puede ser un poco costoso [**32**].

Por su parte, Eclipse tiene entre sus características más importantes la opción de autocompletar. Esta opción ayuda mucho a los programadores que se inician en este entorno ya que no precisa un dominio total del lenguaje de programación. Otra característica que hace a Eclipse el entorno utilizado en este proyecto es que no necesita ser instalado, pues para su utilización solo necesita ser descompactado en una dirección deseada por el usuario. Además de estas características, Eclipse tiene una gran cantidad de bibliografía en el mundo, por lo que a la hora de profundizar los conocimientos de algún usuario que se inicia en esta programación le será muy fácil obtener información relacionada con el tema.

Después de analizar algunas de las ventajas de ambos entornos de desarrollo y después de consultar la opinión de programadores en Internet se seleccionó Eclipse como el entorno de desarrollo más óptimo para programar aplicaciones para Android por dos razones básicas: la elección se hace por preferencia pues las dos herramientas tienen sus ventajas, y porque

el sitio oficial de NetBeans bloquea a nuestro país por lo que se hace más difícil encontrar manuales y bibliografía actualizada de esta herramienta.

CAPÍTULO 2. CONFECCIÓN DE APLICACIONES

En este capítulo se explican cuáles son las herramientas necesarias para programar sobre el SO Android, así como el modo de utilizarlas. Se tratará la configuración del entorno para comenzar a programar y se mostrarán los pasos elementales para desarrollar una aplicación.

2.1.Configuración del entorno de desarrollo en Android

Para comenzar a programar en Android, lo primero es disponer del entorno y las herramientas necesarias en una PC. Seguidamente se enumeraran los pasos a seguir para la instalación y configuración del Eclipse.

Paso 1. Instalación de Eclipse

Lo primero es obtener el ZIP del Eclipse. Este se puede descargar desde enlace: http://www.eclipse.org/downloads/.

Después de tener el ZIP del Eclipse, la instalación consiste en descomprimir el archivo en la ubicación deseada.

Paso 2. Instalación del SDK

La instalación del SDK también consiste en descomprimir el archivo ZIP en la ubicación deseada. Este ZIP se puede descargar desde el enlace:

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

Luego se configura en el menú *Window/Preferences* del Eclipse como muestra la Figura 2.1:

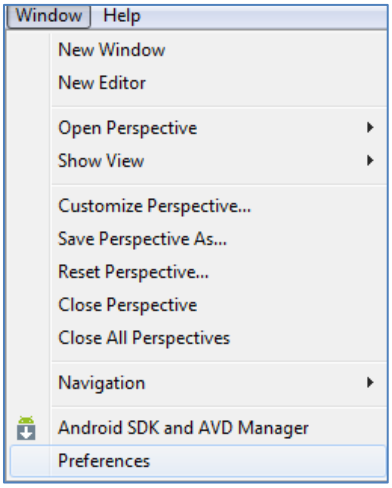

Figura 2.1: Menú *Window* de Eclipse

Luego en el menú Android, como aparece en la Figura 2.2, se indica la dirección donde se descompactó el archivo ZIP.

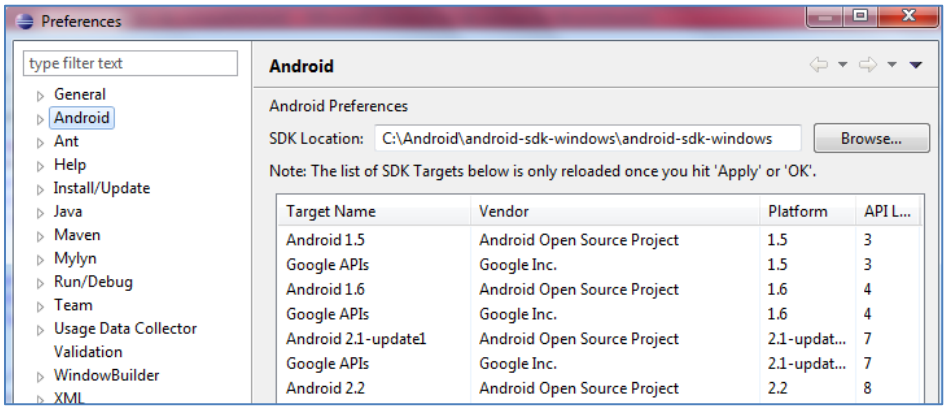

Figura 2.2: Menú *Preferences*

Paso 3. Instalación del ADT

Google ofrece a los desarrolladores un *plugin* para Eclipse llamado *Android Development Tools* (ADT) que facilita en gran medida el desarrollo de aplicaciones para la plataforma. Este se puede descargar desde el URL:

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

Luego se instala mediante el menú *Help/Install New Software…/* (Figura 2.3), indicando la dirección donde se descompactó el ADT, lo cual se muestra en la Figura 2.4.

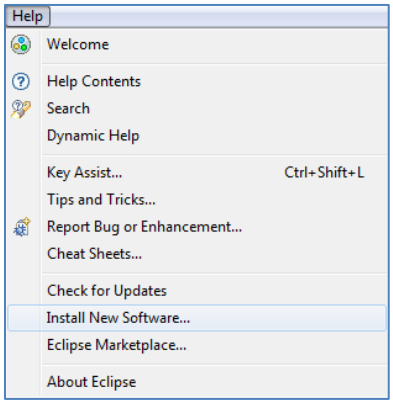

Figura 2.3: Menú *Help*

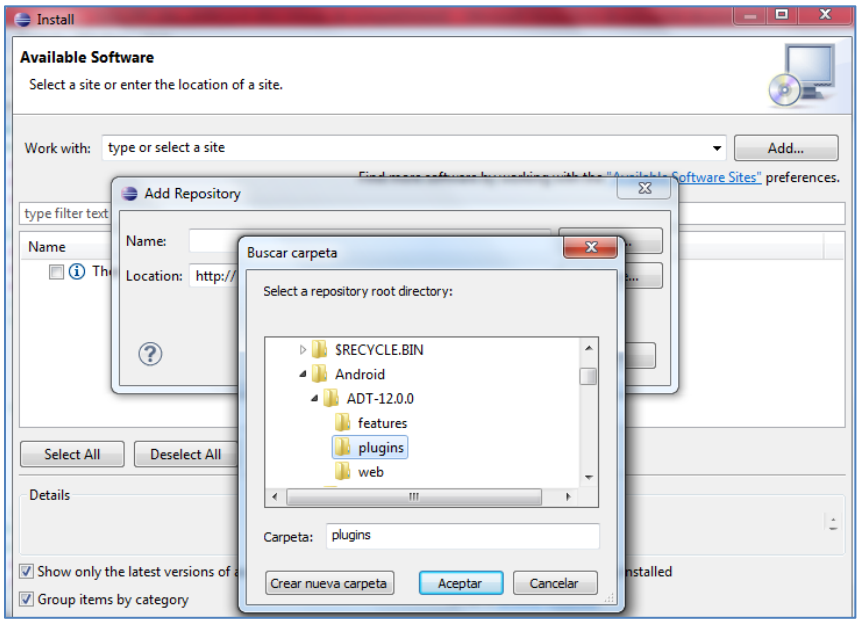

Figura 2.4: Menú *Help/Install New Software…/*

Paso 4. Configuración de un AVD

A la hora de probar y depurar aplicaciones Android no es necesario hacerlo sobre un dispositivo físico, sino que es posible configurar un emulador o dispositivo virtual (*Android Virtual Device*, o AVD) donde realizar fácilmente estas tareas. Para ello se accede al *AVD Manager* desde el menú *Windows* del Eclipse como aparece en la Figura 2.5.

En la sección *Virtual Devices* (Figura 2.6) se pueden añadir tantos AVD como se necesiten (por ejemplo, configurados para distintas versiones de Android). Para configurar el AVD tan sólo es necesario indicar un nombre descriptivo, la versión de Android que se empleará (*Target*), y las características de hardware del dispositivo virtual, como por ejemplo su

resolución de pantalla, el tamaño de la tarjeta SD, o la disponibilidad de GPS como se indica en la Figura 2.7.

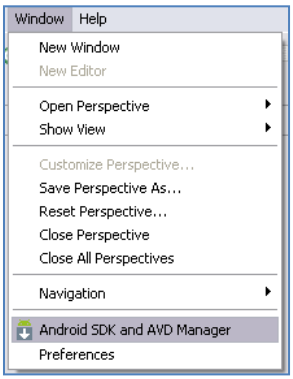

Figura 2.5: Menú *Window*

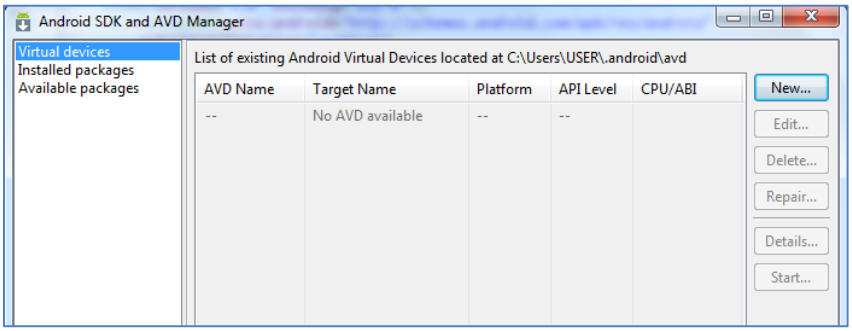

Figura 2.6: Menú *Android SDK and AVD Manager*

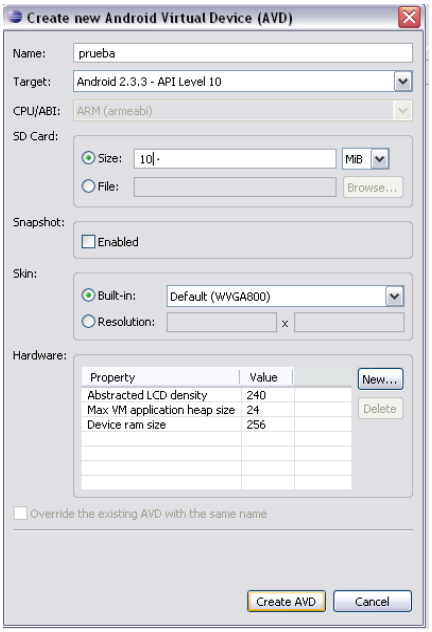

Figura 2.7: Menú *Create new Android Virtual Device (AVD)*

Luego aparecerá en la ventana de *Virtual Devices* el dispositivo creado, como muestra la Figura 2.8:

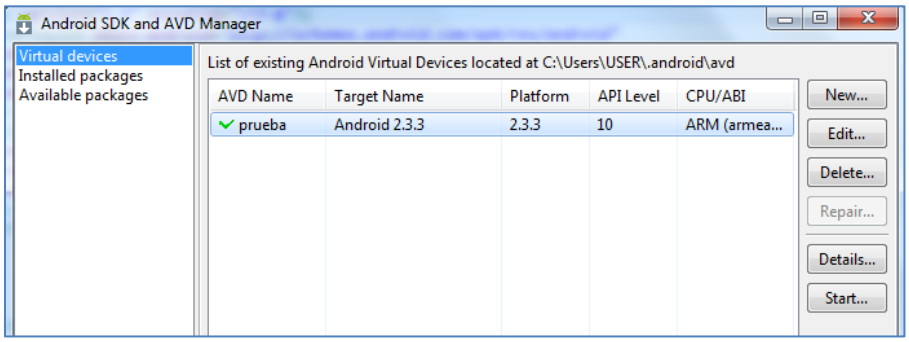

Figura 2.8: Menú *Android SDK and AVD Manager*

Con este paso ya existen las condiciones para para crear el primer proyecto para Android.

2.2.Estructura de un proyecto Android

Para construir una aplicación en Android, primero se requiere entender la estructura general de un proyecto tipo.

Cuando se crea un nuevo proyecto Android en Eclipse, como se mostrará a continuación, se genera automáticamente la estructura de carpetas necesaria para después crear la aplicación. Esta estructura será común a cualquier aplicación, sin importar su tamaño y complejidad.

En la Figura 2.9 se muestran los elementos creados inicialmente para un nuevo proyecto Android:

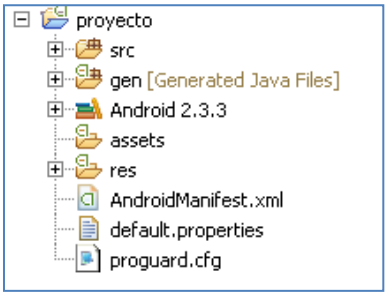

Figura 2.9: Estructura de un proyecto

Carpeta /src/

Contiene todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc. Inicialmente, Eclipse creará el código básico de la pantalla (*Activity*)

principal de la aplicación (Figura 2.10), siempre debajo de la estructura del paquete java definido.

Figura 2.10: Carpeta /src/

Carpeta /res/

Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, videos, cadenas de texto, etc. Los diferentes tipos de recursos de deberán distribuir entre las carpetas tal y como se muestra en la Tabla 2.1:

Tabla 2.1

Carpeta	Descripción
/res/drawable/	Contienen las imágenes de la aplicación. Para utilizar diferentes recursos dependiendo de la resolución del dispositivo se suele dividir en varias subcarpetas: • */drawable-ldpi* • */drawable-mdpi* • */drawable-hdpi*
/res/layout/	Contienen los ficheros de definición de las diferentes pantallas de la interfaz gráfica.
/res/anim/	Contiene la definición de las animaciones utilizadas por la aplicación.
/res/menu/	Contiene la definición de los menús de la aplicación.
/res/values/	Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
/res/xml/	Contiene los ficheros XML utilizados por la aplicación.
/res/raw/	Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

Como ejemplo (Figura 2.11), para un nuevo proyecto Android, se crean los siguientes recursos para la aplicación:

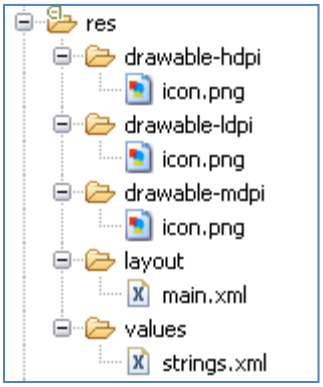

Figura 2.11: Carpeta /res/

Carpeta /gen/

Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que se genera el nuevo proyecto, la maquinaria de compilación de Android crea una serie de ficheros fuente en java dirigidos al control de los recursos de la aplicación, como se ve en la Figura 2.12.

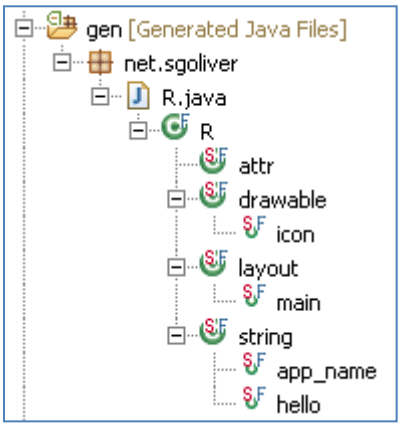

Figura 2.12: Carpeta /gen/

El más importante es el que se puede observar en la imagen anterior, el fichero R.java, y la clase R.

Esta clase R contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta */res/*, de forma que se pueda acceder fácilmente a estos recursos desde el propio código a través de este dato. Así, por ejemplo, la

constante *R.drawable.icon* contendrá el ID de la imagen “icon.png” contenida en la carpeta */res/drawable/*.

A continuación, un ejemplo de la clase R creada por defecto para un proyecto nuevo en la Figura 2.13:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package net.sgoliver;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Figura 2.13: Fichero R.java

Carpeta /assets/

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

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

Fichero AndroidManifest.xml

Contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono), sus componentes (pantallas, mensajes), o los permisos necesarios para su ejecución [**33**].

2.3.Cómo crear un proyecto en Android

Para comenzar un proyecto de Android se deberá seleccionar *File/New/AndroidProyect* desde el Eclipse. Esto abrirá la ventana mostrada en la Figura 2.14. En dicha ventana se procede a especificar el nombre del proyecto y seleccionar la versión (*Target*) de Android deseada.

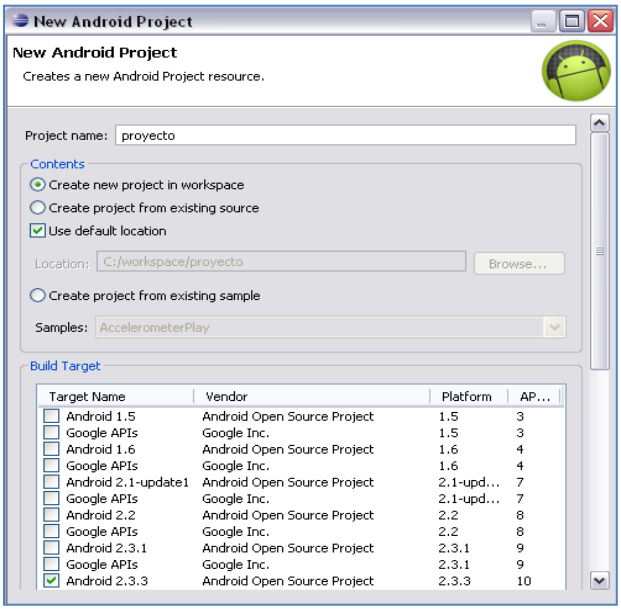

Figura 2.14: Ventana *New Android Project*

Luego se le da nombre a la aplicación, se define el paquete Java por defecto para las clases y el nombre de la clase principal, como aparece en la Figura 2.15:

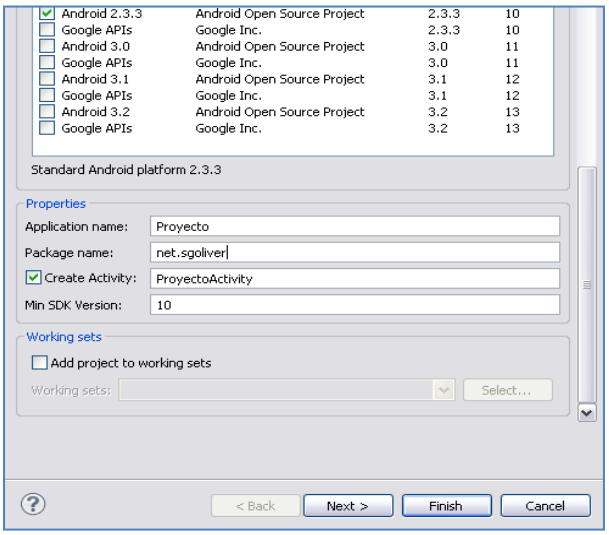

Figura 2.15: Ventana *New Android Project*

Esta acción creará toda la estructura de carpetas necesaria para compilar un proyecto Android.

2.4.Programar en Android

Después crear el nuevo proyecto se le da forma a la aplicación. Para modificar un fichero del proyecto se hace doble clic encima del mismo y este aparece en la ventana de trabajo, como se muestra en la Figura 2.16:

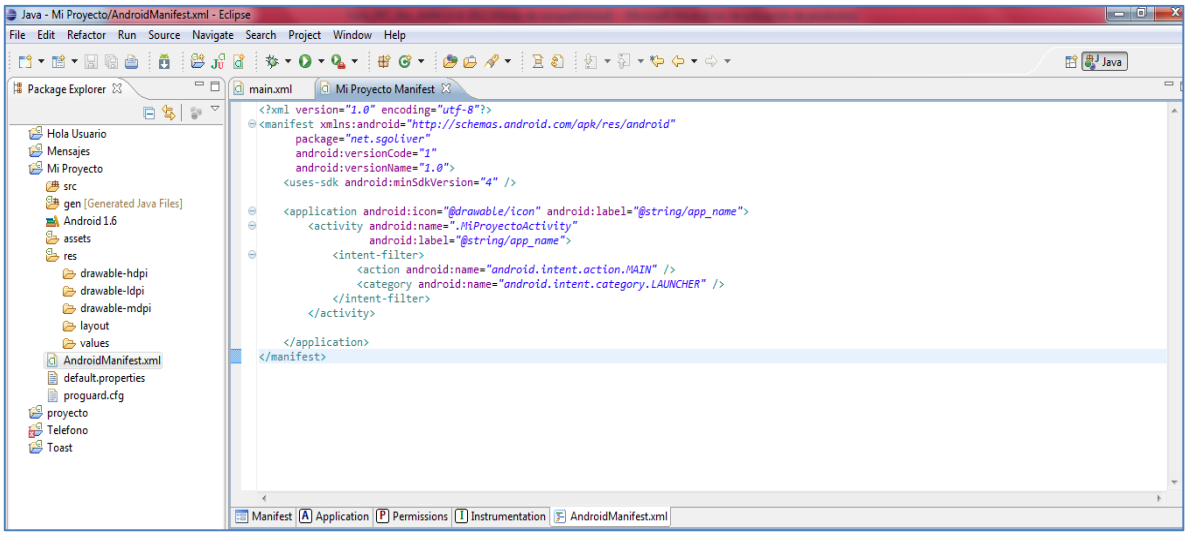

Figura 2.16: Ventana de trabajo del Eclipse

2.4.1. AndroidManifest.xml

Como se explicó anteriormente, el *AndroidManifest.xml* es un fichero requerido en cada aplicación, pues en él se definen los componentes de la aplicación, su identificación y los permisos necesarios para su ejecución.

Un *AndroidManifest.xml* básico luce de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.sgoliver"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MiProyectoActivity"
                  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 2.17: *AndroidManifest.xml*

Loprimero que se observa es el namespace de Android (**xmlns:android="http://schemas.android.com/apk/res/android"**). El *namespace* crea una variedad de atributos estándares de Android disponibles en este fichero que serán utilizados para proporcionar la mayoría de los datos para los elementos en el fichero.

Después está definido el paquete Java que será considerado la base de la aplicación (**package="net.sgoliver"**). Este se define en la creación del proyecto.

Después están definidos los elementos de la aplicación, es decir, el icono (**android:icon="@drawable/icon"**) y la etiqueta (**android:label="@string/app_name"**).

El elemento principal de la aplicación es la *activity*. En este caso solo hay una bandera, pero toda *activity* tiene que estar definida en el *AndroidManifest.xml*, así como la etiqueta de esta y su nombre (**android:name=".MiProyectoActivity"**).

Un elemento importante son los *Intent-filter*, los cuales determinan cuándo la *activity* puede comenzar.

Otro elemento a tener en cuenta son los permisos (**uses-permission**). El *AndroidManifest* puede incluir ninguno o varios permisos. Para declarar un permiso solo se utiliza el atributo **android:name** del permiso que usará la aplicación. Porejemplo:

<uses-permission android:name=" android.permission.RECEIVE_SMS"/>.

2.4.2. Interfaz de usuario

Un paso importante en la aplicación consiste en crear la interfaz de usuario en el archivo *.xml*, creado por defecto en la carpeta */res/layout/*. Para crear la pantalla de la aplicación, Android ofrece un grupo elementos de los cuales se mencionarán aquellos considerados como básicos.

Estos elementos visuales se pueden adicionar a la pantalla de la aplicación introduciéndolos manualmente de manera gráfica en la vista previa que Eclipse proporciona de la pantalla de la aplicación (Figura 2.18), o escribiendo directamente el código en el archivo *.xml* de la pantalla (Figura 2.19):

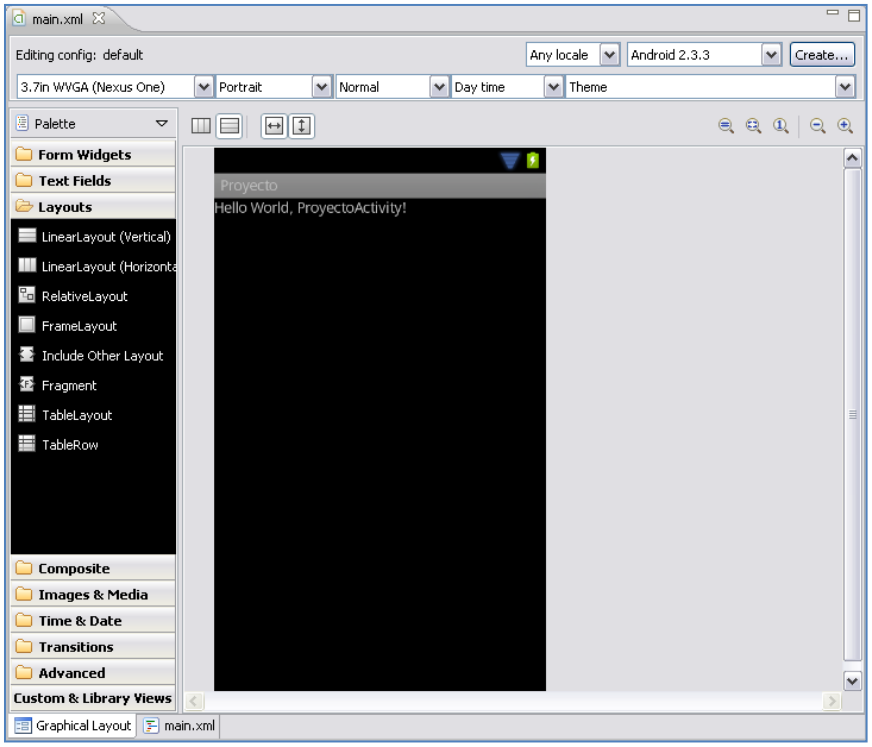

Figura 2.18: Vista previa

Figura 2.19: Fichero *main.xml*

2.4.2.1.Layouts

En primer lugar están los *layouts*, elementos no visuales destinados a controlar la distribución, posición y dimensión de los controles que se insertan en su interior.

FrameLayout

Este es el más simple de todos los *layouts* de Android. Coloca sus controles hijos alineados con la esquina superior izquierda, de forma que cada control queda oculto por el control

siguiente (a menos que éste último tenga transparencia). Por ello, suele utilizarse para mostrar un único control en su interior a modo de contenedor sencillo para un solo elemento sustituible, por ejemplo una imagen.

A los componentes incluidos en un FrameLayout se le establecen sus propiedades **android:layout_width** y **android:layout_height**, que podrán tomar los valores **fill_parent** (para que el control hijo tome la dimensión de su *layout* contenedor) o **wrap_content** (para que el control hijo tome la dimensión de su contenido).

LinearLayout

El siguiente *layout* Android en cuanto a nivel de complejidad es el *LinearLayout*, el mismo apila uno tras otro todos sus elementos hijos de forma horizontal o vertical según se establezca su propiedad **android:orientation**.

Al igual que en un *FrameLayout*, a los elementos contenidos en un *LinearLayout* se le pueden establecer sus propiedades **android:layout_width** y **android:layout_height** para determinar sus dimensiones dentro del *layout*. Pero en el caso de un *LinearLayout*, existe otro parámetro a considerar, la propiedad **android:layout_weight**. Esta propiedad permite dar a los elementos contenidos en el *layout* unas dimensiones proporcionales entre ellas.

Además de estos dos, existen el *TableLayout* y *RelativeLayout*.

Un *TableLayout* permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.

El *RelativeLayout* permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio *layout*. De esta forma, al incluir un nuevo elemento A, se puede indicar, por ejemplo, que debe colocarse debajo del elemento B y alineado a la derecha del *layout* padre.

2.4.2.2.Botones

El SDK de Android proporciona tres tipos de botones: el clásico (*Button*), el de tipo on/off (*ToggleButton*), y el que puede contener una imagen(*ImageButton*). En la Figura 2.20 se muestra el aspecto por defecto de estos tres controles.

Figura 2.20: Vista de botones

Control Button

Un control de tipo *Button* es el botón más básico que se puede utilizar. A este se le asigna el texto mediante la propiedad **android:text**. Además de esta propiedad se pueden utilizar muchas otras como el color de fondo (**android:background**), estilo de fuente (**android:typeface**), color de fuente (**android:textcolor**) o el tamaño de fuente (**android:textSize**). En la Figura 2.21 se muestra una declaración sencilla de un *Button*.

```
<Button android:text="Saludar"
        android:id="@+id/BtnBoton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
```

Figura 2.21: Declaración de un *Button*

Control ToggleButton

Un control de tipo *ToggleButton* es un tipo de botón que puede permanecer en dos estados, pulsado/no_pulsado. En este caso, en vez de definir un sólo texto para el control, se definen dos, dependiendo de su estado. Así, se pueden asignar las propiedades **android:textOn** y **android:textoOff** para definir ambos textos (Figura 2.22).

```
<ToggleButton android:textOn="On"
              android:textOff="Off"
              android:id="@+id/BtnBoton2"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"></ToggleButton>
```

Figura 2.22: Declaración de un *ToggleButton*

Control ImageButton

En un control de tipo *ImageButton* se puede definir una imagen a mostrar en vez de un texto, para lo que se debe asignar la propiedad **android:src**. Normalmente se asigna esta propiedad con el descriptor de algún recurso que hayamos incluido en la carpeta */res/drawable* como se muestra en la Figura 2.23.

```
<ImageButton android:layout_width="wrap_content"
             android:id="@+id/imageButton1"
             android:src="@drawable/icon"
             android:layout_height="wrap_content"></ImageButton>
```

Figura 2.23: Declaración de un *ImageButton*

2.4.2.2.1. Eventos de un botón

Todos estos controles de botones pueden lanzar varios eventos, pero el más común de todos y el que casi siempre se utiliza es el evento **onClick**. La lógica de este evento se tiene que implementar definiendo un nuevo objeto **View.OnClickListener()** y asociarlo al botón mediante el método **setOnClickListener()**. La forma más habitual de hacer esto se muestra en la Figura 2.24.

```
final Button btnBoton1 = (Button)findViewById(R.id.BtnBoton1);

btnBoton1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0)
    {
      lblMensaje.setText("Botón 1 pulsado!");
    }
});
```

Figura 2.24: Lógica de un *Button*

En el caso de un botón de tipo *ToggleButton* suele ser de utilidad conocer en qué estado ha quedado el botón tras ser pulsado, para lo que se puede utilizar su método **isChecked()**. En la Figura 2.25 se comprueba el estado del botón tras ser pulsado y se realizan acciones distintas según el resultado.

```
final ToggleButton btnBoton2 = (ToggleButton)findViewById(R.id.BtnBoton2);

btnBoton2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0)
    {
        if(btnBoton2.isChecked())
            lblMensaje.setText("Botón 2: ON");
        else
            lblMensaje.setText("Botón 2: OFF");
    }
});
```

Figura 2.25: Lógica de un *ToggleButton*

2.4.2.3.Cuadros de texto

El control *EditText* es el componente de edición de texto que proporciona la plataforma Android. Permite la introducción y edición de texto por parte del usuario, por lo que en

tiempo de diseño una propiedad interesante a establecer, además de su posición/tamaño, es el texto a mostrar por defecto, atributo **android:text**. En la Figura 2.26 se muestra la declaración de un *EditText* sencillo.

```
<EditText android:id="@+id/TxtTexto"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content">
    <requestFocus></requestFocus>
</EditText>
```

Figura 2.27: Declaración de un *EditText*

Desde el código se puede recuperar y establecer este texto mediante los métodos **getText()** y **setText(nuevoTexto)** respectivamente como se muestra en la Figura 2.28.

```
final EditText txtTexto = (EditText)findViewById(R.id.TxtTexto);
String texto = txtTexto.getText().toString();
txtTexto.setText("Hola mundo!");
```

Figura 2.28: Métodos getText y setText

2.4.2.4.Etiquetas

El control *TextView* se utiliza para mostrar un determinado texto al usuario. A parte de esta propiedad, la naturaleza del control hace que las más interesantes sean las que establecen el formato del texto mostrado, que al igual que en el caso de los botones son las siguientes: **android:background** (color de fondo), **android:textColor** (color del texto), **android:textSize** (tamaño de la fuente) y **android:typeface** (estilo del texto). En la Figura 2.29 se muestra la declaración de un *TextView* sencillo.

```
<TextView
    android:id="@+id/LblEtiqueta"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Introducir nombre"
    />
```

Figura 2.29: Declaración de un *TextView*

De igual forma, también se puede manipular estas propiedades desde el código. Como ejemplo, en el fragmento de la Figura 2.30 se recupera el texto de una etiqueta con **getText()**, posteriormente se le concatena unos números y se actualiza su contenido mediante **setText()**.

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);
String texto = lblEtiqueta.getText().toString();
texto += "123";
lblEtiqueta.setText(texto);
```

Figura 2.30: Recuperación del texto

2.4.2.5.Imágenes

El control *ImageView* permite mostrar imágenes en la aplicación. La propiedad más interesante es **android:src**, que permite indicar la imagen a mostrar. Lo normal será indicar como origen de la imagen el identificador de un recurso de la carpeta */res/drawable*, por ejemplo **android:src="@drawable/unaimagen"**.

Además de esta propiedad, existen algunas otras útiles en algunas ocasiones como las destinadas a establecer el tamaño máximo que puede ocupar la imagen, **android:maxWidth** y **android:maxHeight**. En la Figura 2.31 se muestra la declaración de un control *ImageView*.

```
<ImageView android:layout_width="wrap_content"
           android:id="@+id/ImgFoto"
           android:layout_height="wrap_content"
           android:src="@drawable/icon"></ImageView>
```

Figura 2.31: Declaración de un *ImageView*

En la lógica de la aplicación, se puede establecer la imagen mediante el método **setImageResorce(…)**, pasándole el ID del recurso a utilizar como contenido de la imagen (Figura 2.32).

```
ImageView img = (ImageView)findViewById(R.id.ImgFoto);
img.setImageResource(R.drawable.icon);
```

Figura 2.32: Método setImageResorce

2.4.3. Mensajes de alerta

En Android existen varias formas de notificar mensajes al usuario, como por ejemplo los cuadros de diálogo modales o las notificaciones de la bandeja del sistema (o barra de estado). Pero este apartado se va a centrar en la forma más sencilla de notificación: los llamados *Toast*.

Un *toast* es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento). Aunque son personalizables, aparecen por defecto en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido. Por sus propias características, este tipo de notificaciones son ideales para mostrar mensajes rápidos y sencillos al usuario, pero por el contrario, al no requerir confirmación por su parte, no deberían utilizarse para hacer notificaciones demasiado importantes.

Su utilización es muy sencilla, concentrándose toda la funcionalidad en la clase *Toast*. Esta clase dispone de un método estático **makeText()** al que se debe pasar como parámetro el contexto de la actividad, el texto a mostrar, y la duración del mensaje, que puede tomar los valores **LENGTH_LONG** o **LENGTH_SHORT**, dependiendo del tiempo que se desee mostrar la notificación en pantalla. Tras obtener una referencia al objeto *Toast* a través de este método, ya sólo queda mostrarlo en pantalla mediante el método **show ()**.

En la Figura 2.33 se muestra una aplicación de ejemplo para demostrar el funcionamiento de este tipo de notificaciones. Para empezar se incluye un botón que muestre un *toast* básico de la forma que se acaba de describir.

```
Button btnDefecto = (Button)findViewById(R.id.button1);
btnDefecto.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        Toast toast1 =
            Toast.makeText(getApplicationContext(),
                    "Se oprimió el botón Toast", Toast.LENGTH_SHORT);

        toast1.show();
    }
});
```

Figura 2.33: Declaración de un *Toast*

Si se ejecuta esta sencilla aplicación en el emulador y se pulsa el botón que se acaba de añadir se ve como en la parte inferior de la pantalla aparece el mensaje “Se oprimió el botón Toast”, que tras varios segundos desaparecerá automáticamente (Figura 2.34).

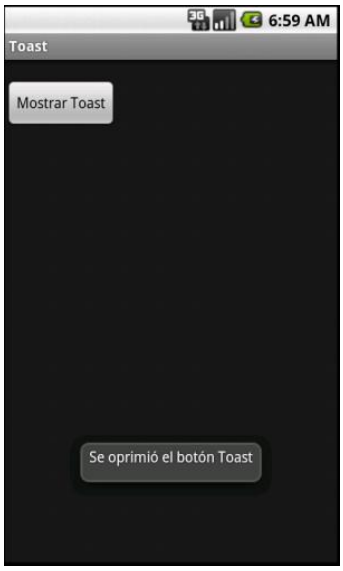

Figura 2.34: Vista de un *Toast*

2.4.4. Cómo cargar aplicaciones en un AVD

Antes de pasar a ver los ejemplos de aplicaciones se explicará cómo se carga una aplicación en un AVD creado.

Lo primero que se hace es ir a la opción *RunNew_configuration* y seleccionar la opción *RunConfigurations…* como se muestra en la Figura 2.35.

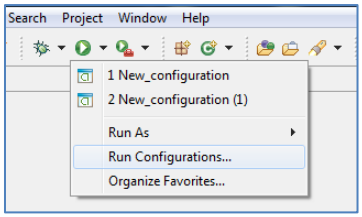

Figura 2.35: Menú *RunNew_configuration*

Luego se hace doble clic en la opción *Android Application* y saldrá una nueva configuración en la cual se especifica el proyecto que se va a simular y el AVD que se va a utilizar. Esto se muestra en las Figuras 2.36 y 2.37 respectivamente.

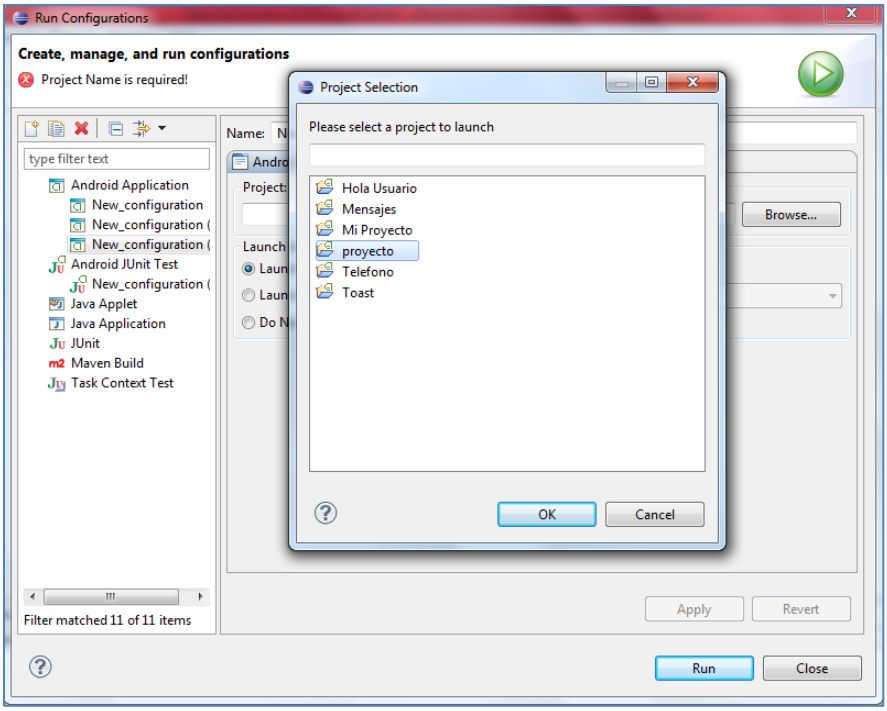

Figura 2.36: Ventana Project Selection

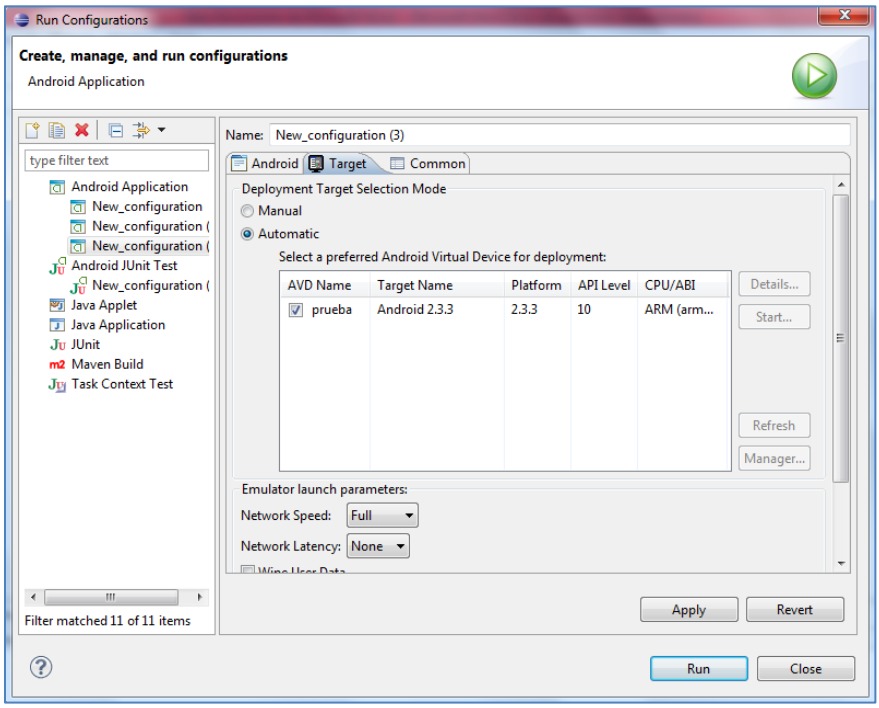

Figura 2.37: Selección del AVD

Después de haber seleccionado el proyecto y el AVD se inicia el dispositivo virtual en la opción *Android SDK and AVD Manager* seleccionándolo y marcando la opción *Start*.

Luego se especifica el tamaño en pantalla del dispositivo mediante la *ScreenSize* y se corre el AVD (Figura 2.38).

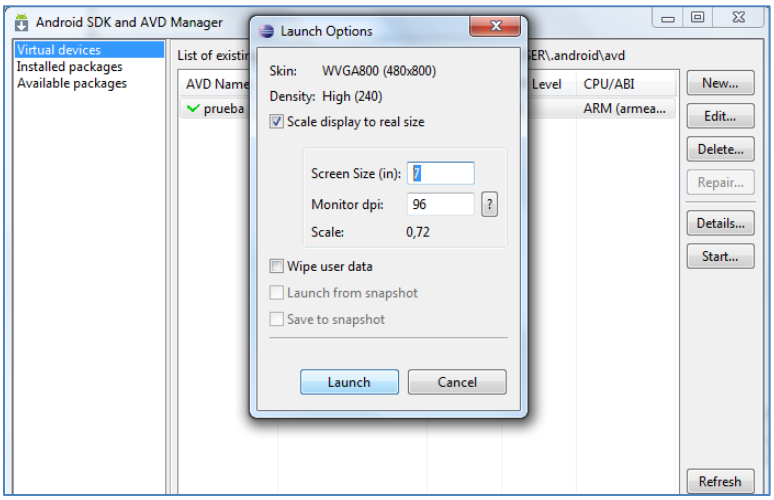

Figura 2.38: Ventana Launch Options

Cuando el dispositivo termine de cargar se verá en la pantalla de aplicaciones de este la aplicación que se quiere simular (Figura 2.39).

Figura 2.39: Vista de aplicaciones en el emulador

Es necesario aclarar que existen más elementos para programar para Android. En este proyecto solo se mencionan los que se consideraron elementales para comenzar a programar aplicaciones de pequeña complejidad.

En el próximo capítulo se verán aplicaciones para entender mejor cómo utilizar esta guía.

CAPÍTULO 3. APLICACIONES DESARROLLADAS

En este capítulo se analizarán aplicaciones de pequeña complejidad que demostraran el uso de la guía del capítulo anterior y se explicarán los principales aspectos de cada una de ellas.

3.1.Hola Usuario

Esta primera aplicación es muy sencilla. Consta de pocos elementos visuales y de una lógica muy sencilla.

El fin de esta aplicación es mostrar una pantalla con un cuadro de texto en el cual el usuario escribirá su nombre y un botón para saludarlo. La aplicación debe quedar como se muestra en la Figura 3.1.

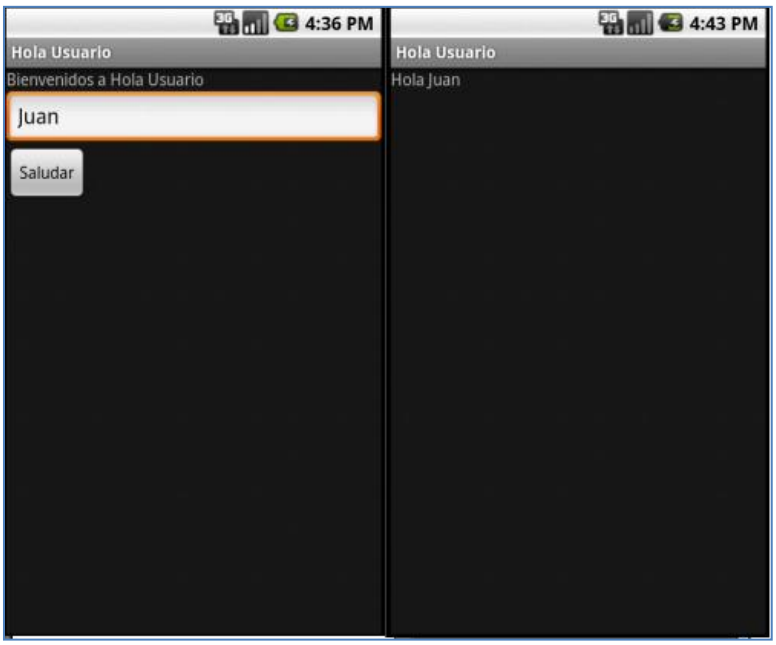

Figura 3.1: Aplicación Hola Usuario

Lo primero que se hace es modificar la interfaz de usuario. Esto se puede hacer arrastrando hacia ella los elementos que necesitamos. Como se observa en la figura anterior, se necesitan dos pantallas, lo que quiere decir que también se necesitan dos *Activities*. Cada *Activity* necesita un *.xml* por separado para su interfaz de usuario. Por esta razón se crea un archivo *.xml* en la carpeta */res/layout/* llamado *frmmensaje*. Este fichero se crea haciendo clic derecho en la carpeta mencionada y seleccionando *New/Android XML File*. En este último fichero solo se añade una etiqueta como se hizo en el capítulo 2 para mostrar el saludo al usuario.

Para modificar las características de cada elemento visual se puede ir directamente al *main.xml* o hacer doble clic en el elemento.

Lo primero que se modifica en el *main.xml* es la etiqueta de la parte superior. En esta solo se cambia el texto a mostrar.

Debajo de la etiqueta se encuentra el cuadro de texto. Lo más importante es definir el **id** para el elemento. En este caso se identifica como **TxtNombre**.

El último elemento es el botón de saludar. A este se le identifica por el **idBtnHola** y se le modifica el texto a mostrar.

Después estos cambios, el fichero *main.xml* queda como muestra la Figura 3.2.

```
<?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:text="Bienvenidos a Hola Usuario"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

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

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Saludar"
        android:id="@+id/BtnHola"/>

</LinearLayout>
```

Figura 3.2: Fichero *main.xml*

El próximo paso es modificar el fichero *frmmensaje.xml*. Las modificaciones en este fichero son solo para la etiqueta y el fichero quedará como se muestra en la Figura 3.3:

```
<?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">

<TextView android:id="@+id/TxtMensaje"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="$mensaje"
    ></TextView>

</LinearLayout>
```

Figura 3.3: Fichero *frmmensaje.xml*

Después de definir la interfaz de usuario de la aplicación se programa la lógica de los elementos. Eclipse crea por defecto la clase de la pantalla principal, en este caso *HolaUsuarioActivity*, y la salva en el fichero *HolaUsuarioActivity.java*. Además se va a crear otra clase para la pantalla secundaria en la carpeta */src/* a la que se le llamará FrmMensaje. Esto se logra haciendo clic derecho en la carpeta y seleccionado *New/Class*.

Se comienza modificando el archivo *HolaUsuarioActivity.java*, en el cual se define todo menos la interfaz gráfica, pues esto lo hace el código incluido por defecto en cada una de las clases. Lo primero es obtener la referencia de los controles que se van a manipular. Esto se logra mediante el método **findViewById()** indicando el ID de cada control, definidos como siempre en la clase R (Figura 3.4):

```
final EditText txtNombre = (EditText)findViewById(R.id.TxtNombre);
final Button btnHola = (Button)findViewById(R.id.BtnHola);
```

Figura 3.4: Referencia de los controles

Una vez que se tiene acceso a los diferentes controles sólo queda implementar las acciones a tomar cuando se pulse el botón de la pantalla. Para ello se implementa el evento **onClick** de dicho botón como se muestra en la Figura 3.5.

```
btnHola.setOnClickListener(new OnClickListener() {

    public void onClick(View arg0) {
        Intent intent = new Intent(HolaUsuarioActivity.this, FrmMensaje.class);

        Bundle bundle = new Bundle();
        bundle.putString("NOMBRE", txtNombre.getText().toString());
        intent.putExtras(bundle);

        startActivity(intent);
    }
});
```

Figura 3.5: Evento onClick

La comunicación entre los distintos componentes y aplicaciones en Android se realiza mediante *intents*, por lo que el primer paso será crear uno objeto de este tipo. Existen varias variantes del constructor de la clase *Intent*, cada una de ellas dirigida a unas determinadas acciones, pero en este caso particular se va a utilizar el *intent* para llamar a una actividad desde otra de la misma aplicación, para lo que se le pasara al constructor una referencia a la propia actividad llamadora (**HolaUsuario.this**), y la clase de la actividad llamada (**FrmMensaje.class**).

Si se quisiera tan sólo mostrar una nueva actividad ya tan sólo quedaría llamar a **startActivity()** pasándole como parámetro el *intent* creado. Pero en este ejemplo se desea también pasarle cierta información a la actividad, concretamente el nombre que introduzca el usuario en el cuadro de texto. Para hacer esto se va a crear un objeto *Bundle*, que puede contener una lista de pares clave-valor con toda la información a pasar entre las actividades. En este caso sólo se añade un dato de tipo *String* mediante el método **putString(clave, valor)**. Tras esto se añade la información al *intent* mediante el método **putExtras(bundle).**

Finalizada la actividad principal de la aplicación se pasa a la secundaria. Se comienza de forma análoga a la anterior, ampliando el método **onCreate** obteniendo las referencias a los objetos que se manipularán, esta vez sólo la etiqueta de texto. Tras esto viene lo más interesante, se debe recuperar la información pasada desde la actividad principal y asignarla como texto de la etiqueta. Para ello se accede en primer lugar al *intent* que ha originado la actividad actual mediante el método **getIntent()** y se recupera su información asociada (objeto *Bundle*) mediante el método **getExtras()**.

Hecho esto tan sólo queda construir el texto de la etiqueta mediante su método **setText(texto)** y recuperando el valor de la clave almacenada en el objeto *Bundle* mediante **getString(clave)**.

El archivo *FrmMensaje.java* queda como se muestra en la Figura 3.6.

```
package net.sgoliver;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;




public class FrmMensaje extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.frmmensaje);

        TextView txtMensaje = (TextView)findViewById(R.id.TxtMensaje);

        Bundle bundle = getIntent().getExtras();

        txtMensaje.setText("Hola " + bundle.getString("NOMBRE"));
    }
}
```

Figura 3.6: Archivo *FrmMensaje.java*

Con esto termina la lógica de las dos pantallas de la aplicación y tan sólo queda un paso para finalizar su desarrollo. Como ya se indicó en alguna ocasión, toda aplicación Android utiliza un fichero especial en formato XML (*AndroidManifest.xml*) para definir, entre otras cosas, los diferentes elementos que la componen. Por tanto, todas las actividades de la aplicación deben quedar convenientemente recogidas en este fichero. La actividad principal ya debe aparecer puesto que se creó de forma automática al crear el nuevo proyecto Android, por lo que se debe añadir tan sólo la segunda como se muestra en la línea decimoquinta de la Figura 3.7.

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

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

        <activity android:name=".FrmMensaje"></activity>
     </application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

Figura 3.7: Fichero *AndroidManifest.xml*

Después de haber terminado el código de la aplicación ya se puede simular en el emulador de Android.

3.2.Mensajes

La próxima aplicación es un poco más compleja pero todavía es lo suficientemente fácil para entender los elementos básicos de la guía.

Se trata de una aplicación para enviar SMS (*Short MessagesService*). Casi todos los dispositivos que utilizan Android son teléfonos, por tanto estos seguro tienen por defecto aplicaciones para el envío de SMS. En este caso se verá una aplicación muy sencilla que mostrará cómo utilizar los permisos que se comentar en la guía del capítulo anterior.

Esta aplicación consta de una pantalla única en la cual se coloca un cuadro de texto para insertar el número de destino, otro para insertar el texto del mensaje, dos etiquetas para identificar cada uno de los cuadros de texto y un botón para enviar el mensaje.

La aplicación en pantalla deberá verse como muestra la Figura 3.8.

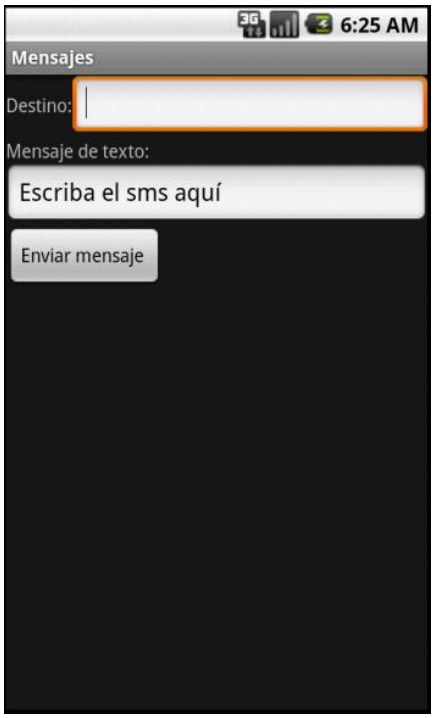

Figura 3.8: Vista de Mensajes

Como siempre se comienza por la interfaz gráfica. Para esto solo se agrega cada elemento que se necesita y el id que se desee. Después de eso el archivo *main.xml* queda como se muestra en la Figuras 3.9 y 3.10.

```
<?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">

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

        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Destino:" />

        <EditText
        android:id="@+id/addrEditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"

        android:text="" />
    </LinearLayout>
```

Figura 3.9: Fichero *main.xml*

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

        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mensaje de texto:" />

        <EditText
        android:id="@+id/msgEditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Escriba el sms aquí" />

    </LinearLayout>

    <Button
    android:id="@+id/sendSmsBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Enviar mensaje " />

</LinearLayout>
```

Figura 3.10: Fichero *main.xml*

Para esta aplicación el único cambio que se necesita hacer en el archivo *AndroidManifest.xml* es adicionar el permiso de envío de mensajes. Esto se logra adicionando la línea:

<uses-permissionandroid:name="android.permission.SEND_SMS"/>.

El *AndroidManifest.xml* queda como se muestra en la Figura 3.11.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.sgoliver"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="4" />
<uses-permission
android:name="android.permission.SEND_SMS" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Mensaje"
                  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 3.11: Fichero *AndroidManifest.xml*

Como ya se vio en la aplicación anterior, Eclipse crea en el *Mensaje.java*, la llamada a la interfaz de la aplicación por defecto. Por esto se va a comenzar por el botón de enviar llamando a la referencia de este. Luego se implementa el evento **onClick** de dicho botón. Luego se llama a la referencia de los cuadros de texto. Después de esto se crea un bloque *try* que llamará al método **sendSmsMessage**, el cual se define después. Además se envía un mensaje al usuario de que el mensaje fue enviado. En caso de que exista algún error enviando el mensaje se necesita hacérselo saber al usuario y para esto se utiliza el bloque *catch* mediante el que se le envía al usuario un mensaje de fallo. Todo esto se muestra en la Figura 3.12.

```
package net.sgoliver;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Mensaje extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        Button sendBtn = (Button)findViewById(R.id.sendSmsBtn);
        sendBtn.setOnClickListener(new OnClickListener(){


            public void onClick(View view) {
                EditText addrTxt =
                      (EditText)Mensaje.this.findViewById(R.id.addrEditText);

                EditText msgTxt =
                      (EditText)Mensaje.this.findViewById(R.id.msgEditText);

                try {
                    sendSmsMessage(
                        addrTxt.getText().toString(),msgTxt.getText().toString());
                    Toast.makeText(Mensaje.this, "SMS Enviado",
                        Toast.LENGTH_LONG).show();
                } catch (Exception e) {
                    Toast.makeText(Mensaje.this, "Fallo al enviar SMS",
                        Toast.LENGTH_LONG).show();
                    e.printStackTrace();
                }
            }});
    }
```

Figura 3.12: Fichero *Mensaje.java*

La parte más interesante de esta aplicación es el método **sendSmsMessage()**. Este utiliza a su vez el método **sendTextMessage()** de la clase **SmsManager**. En este caso se le pasa a este método los objetos del tipo *string***address** y **message** que contienen la el número de destino y el contenido del mensaje respectivamente (Figura 3.13).

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

    private void sendSmsMessage(String address,String message)throws Exception
    {
        SmsManager smsMgr = SmsManager.getDefault();
        smsMgr.sendTextMessage(address, null, message, null, null);
    }
}
```

Figura 3.13: Método sendSmsMessage

Con esto la aplicación está lista para emular.

Las aplicaciones mostradas en este capítulo tienen pocas líneas de código, emplean elementos sencillos y son fáciles implementar por lo que resultan idóneas para el aprendizaje de la programación para Android. Los interesados en implementar aplicaciones de una mayor complejidad podrían remitirse luego a la carpeta de ejemplos que forma parte del SDK de Android.

Con este objetivo también pueden consultar otros textos como *TheBusyCoder´s Guide to Android Development*, del autor Mark L. Murphy, *Pro Android 2*, de los autores SayedHashimi, SatyaKormatineni y DaveMacLean, y *Hello, Android* de Ed Burnette.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Se analizaron y evaluaron las diferentes herramientas afines y se determinó que el Eclipse constituye el entorno de desarrollo más adecuado para la confección de aplicaciones para el SO Android.Esto debido a sus características y a la abundante bibliografía que posee.

Se confeccionaron aplicaciones de baja complejidad, que permitirán una mejor comprensión de las facilidades del entorno de desarrollo Eclipse para programar aplicaciones para el SO Android.

Se elaboró una guía para orientar a los estudiantes de la carrera Ingeniería en Telecomunicaciones y Electrónica de la FIE sobre los pasos básicos de la programación para Android utilizando Eclipse, tema sobre el que no existe bibliografía en la facultad.

Recomendaciones

Confeccionar otras guías similares donde se tengan en cuenta otros sistemas operativos de mucho auge en la actualidad como iOS y Windows Phone.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Sistemas Operativos, conceptos fundamentales*.: Editorial Wiley, 2005.

[2] (2012) Wikipedia. [Online]. http://es.wikipedia.org/w/index.php?title=Sistema_operativo&oldid=54474717

[3] "Sistemas Operativos," Dpto Informática y Automática. Universidad de Salamanca, 2007.

[4] W. Stallings, *Sistemas Operativos*, 2nd ed.: Prentice-Hall, 1997.

[5] A. Marconcini. Sistemas Operativos.

[6] (2012) Wikipedia. [Online]. http://es.wikipedia.org/w/index.php?title=Mac_OS&oldid=54489396

[7] (2007) The Unusual History of Microsoft Windows. [Online]. http://inventors.about.com/od/mstartinventions/a/Windows.htm?rd=1

[8] A. Alea, *Manual de Linux*., 2003.

[9] (2012, Marzo) Wikipedia. [Online]. http://es.wikipedia.org/w/index.php?title=Sistema_operativo_m%C3%B3vil&oldid=54187263

[10] A. Baz, Ferreira I., M. Álvarez, and R. García, *Dispositivos móviles*., 2008.

[11] (2012) Wikipedia. [Online]. http://es.wikipedia.org/w/index.php?title=Symbian_OS&oldid=54143903

[12] S. Perez. (2010, junio) Android Steals Market Share from iPhone. [Online]. http://www.readwriteweb.com/archives/android_steals_market_share_from_iphone.php

[13] (2011, enero) Google´s Android becomes the world´s leadineg smart phone plataform. [Online]. http://www.canalys.com/pr/2011/r2011013.html

[14] I. Pérez, "Estudio de la plataforma de software Android para el desarrollo de una aplicación social," Universidad Politécnica de Cataluña , 2009.

[15] (2009, diciembre) New York Times. [Online]. http://bits.blogs.nytimes.com/2009/12/15/is-

the-google-phone-an-unauthorized-replicant/

[16] B. Elgin. (2005, agosto) Google Buys Android for Its Mobile Arsenal. [Online]. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm

[17] (2007, marzo) Google admits to mobile phone plan. [Online]. http://www.directtraffic.org/OnlineNews/Google_admits_to_mobile_phone_plan_18094880.html

[18] (2010, junio) Open Handset Alliance. [Online]. http://www.openhandsetalliance.com/

[19] (2008, septiembre) Android Overview. [Online]. http://www.openhandsetalliance.com/android_overview.html

[20] (2009, abril) Android 1.5 Platform Highlights. [Online]. http://developer.android.com/sdk/android-1.5-highlights.html

[21] J. Hernández. (2011, diciembre) El Androide Libre. [Online]. http://www.elandroidelibre.com/

[22] (2009, agosto) Canalys: iPhone outsold all Windows Mobile phones in Q2 2009. [Online]. http://www.appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html

[23] (2010, noviembre) comScore Reports February 2010 U.S. Mobile Subscriter Market Share. [Online]. http://www.mycomscore.net/Press_Events/Press_Releases/2010/4/comScore_Reports_February_2010_U.S._Mobile_Subscriber_Market_Share

[24] (2010, noviembre) comScore Reports September 2010 U.S. Mobile Subscriber Market. [Online]. http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Reports_September_2010_U.S._Mobile_Subscriber_Market_Share

[25] G. Sandoval. (2010, agosto) More signs iPhone under Android attack. [Online]. http://news.cnet.com/8301-13579_3-20012418-37.html

[26] (2010, octubre) Google expands Android´s reach, accepting paid apps from 20 more countries, selling to 18 more. [Online]. http://www.engadget.com/2010/10/01/google-expands-androidss-reach-accepting-paid-apps-from-20-mor/

[27] C. Arthur. (2010, junio) Eric Schmidt´s dog whistle to mobile developers: abandon Windows Phone. [Online]. http://www.guardian.co.uk/technology/2010/jun/25/android-schmidt-mobile-platform

[28] A. Catalán, *Curso Android: Desarrollo de aplicaciones móviles*, 1st ed., 2011.

[29] S. Gómez, *Curso Programación Android*., 2011.

[30] E. Burnette, *Hello, Android*.: PI.O printing, 2009.

[31] S. Hashimi, S. Komatineni, and D. MacLean, *Pro Android 2*.: apress, 2010.

[32] J. Gimeno and J. González, Introducción a Netbeans, 2010.

[33] M. Murphy, *The Busy Coder's Guide to Android Development*., 2008.