UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Aplicación móvil de alarma para discapacitados visuales en plataforma Android

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autor: Mark Perenyi

Director: José Vicente Busquets Mataix

5/12/2013

RESUMEN

Los dispositivos móviles constituyen cada vez más que un aparato para llamar y enviar mensajes. Funciones de comunicación y procesamiento de datos que van mucho más allá que las clásicas de la telefonía móvil. El gigante de Internet Google es una firme alternativa a otros sistemas ya ampliamente extendidos como IPhone o BlackBerry. El presente proyecto busca conocer y comprender las características y el funcionamiento de este sistema operativo, averiguando sus posibilidades y ventajas frente a otras alternativas. Además, abarca el desarrollo completo de una aplicación para los discapacitados visuales, quienes también quieren darle cada vez más uso a estos aparatos.

Palabras clave: android, alarma, aplicación, despertador, eclipse.

1. INTRODUCCIÓN

1.1 MOTIVACIÓN

¿Por qué se ha elegido Android? ¿Por qué el despertador para discapacitados visuales? En esta sección se muestran los aspectos que han empujado a hacer esta elección.

Un informe publicado por la firma de investigación *IDC* [1], revela que más de la mitad de los envíos de teléfonos móviles durante el primer trimestre de 2013 fueron smartphones. Traduciéndolo en cifras, podemos decir que los fabricantes enviaron 418.600.000 de móviles de los cuáles 216.200.000 de ellos fueron smartphones.

Si se revisan estadísticas de smartphones por *Gartner* [2], se comprueba que en el año 2013 un 51% de los móviles inteligentes eran plataforma Android, cifra que supone más de la mitad de la total.

North America Smartphone UB by OS : % of Total	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Android	0.0%	0.9%	6.3%	26.5%	47.2%	52.8%	51.8%	51.0%	49.4%	45.7%	42.1%
Apple iOS	8.8%	14.3%	21.0%	22.6%	27.6%	32.6%	35.6%	34.1%	31.6%	30.5%	29.9%
Bada	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Blackberry OS	47.3%	50.6%	49.7%	35.5%	17.8%	9.4%	5.4%	3.9%	3.3%	3.0%	2.7%
Firefox OS	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.4%	0.7%	1.1%	1.4%
LiMo	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
MeeGo	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Microsoft	29.6%	22.2%	14.2%	8.6%	4.6%	4.3%	6.7%	10.0%	13.4%	16.2%	18.2%
Palm OS / webOS	8.2%	6.9%	4.7%	3.0%	1.0%	0.3%	0.1%	0.0%	0.0%	0.0%	0.0%
Symbian	4.1%	2.4%	2.8%	2.7%	1.5%	0.5%	0.2%	0.1%	0.0%	0.0%	0.0%
Tizen	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.4%	1.1%	1.6%
Others	1.9%	2.7%	1.4%	1.2%	0.3%	0.1%	0.0%	0.3%	1.1%	2.4%	4.1%
Total	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Figura 1.1. Smartphones y sistemas operativos. Estadísticas.

La razón por la que la gente elige utilizar cada vez más un móvil inteligente es, por un lado la relación precio-calidad, y por otra parte la tecnología que ofrecen esos móviles.

Mi decisión de hacer mi aplicación para Android es, precisamente, por su tendencia ganadora en el mercado de móviles.

Por otro lado, los sistemas que engloban las aplicaciones en los diferentes sistemas, apenas ofrecen aplicaciones para gente con discapacidad visual. Sin embargo, según la *OMS* [3], en 2010 existían 285 millones de personas con algún tipo de discapacitad visual. Dicha cifra supone el 4% de la población mundial. Esos datos me empujar a la decisión de preparar aplicaciones para el fuerte mercado que suponen esas personas.

He querido hacer algo compacto, sin embargo sin ser demasiado complicado de utilizar, dado la condición de discapacidad del usuario final. Hojeando las aplicaciones en Android Store, observé cuenta que siendo una de las funcionalidades básicas del móvil, no hay ningúna alarma disponible para los con problemas de visión. Mi decisión por diseñar el despertador ha sido después de haber meditado tan sólo unas horas, ya que en mi opinión, antes de hacer cualquier aplicación con propósito de diversión, cómo un juego, me parece mucho más importante crear algo que sirve fines más importantes.

1.2 Objetivos Globales y Personales

La realización de este proyecto busca cumplir varios objetivos que se detallan a continuación.

El objetivo principal es desarrollar un despertador utilizando la plataforma Android, ofreciendo una alternativa a la versión que ya trae por defecto el propio sistema operativo.

Otro de los objetivos es el de explotar las posibilidades que ofrece Android para personas con discapacidad visual, destacando el uso de pantalla táctil y altavoz incorporado para facilitar la entrada, modificación y borrado de alarmas.

Un tercer objetivo es de proporcionar un punto de partida a futuros estudiantes, que deseen desarrollar aplicaciones de cualquier tipo en la misma plataforma para personas con deficiencia.

En resumen, expuesto en el apartado anterior la cantidad de la publicación total que tiene tal deficiencia, tal número no coincide en proporción con el número de aplicaciones y/o investigaciones para dichas personas. Es una multitud, que igual, o quizás más que cualquier otro tipo de agrupación,

constituye un fuerte mercado preparado para utilizar su Smartphone, pero adaptando a sus necesidades.

Figura 1.2 Captura de pantalla

1.3 Fases de desarrollo

En este apartado se detallan las distintas fases que han formado parte del desarrollo del proyecto.

- *Documentación previa*: dado que el proyecto se desarrolla en una plataforma con la que no se estaba familiarizado, es necesario documentarse sobre la plataforma Android y cómo desarrollar aplicaciones en ella
- *Análisis de requisitos*: se realiza un análisis de requisitos para determinar características tales como las funcionalidades básicas de la aplicación o las opciones disponibles para los usuarios.
- *Diseño*: El diseño del software ha de incluir bocetos de las diferentes pantallas, componentes que lo forman y todo aquello que sirva como guía para la implementación del mismo
- *Implementación*: con el estudio realizado durante las fases de análisis y diseño, se pasa a la fase de implementación. La aplicación se podrá ejecutar desde el primer momento, de modo

que a medida que se avance son su desarrollo, las dificultades de programación se irán incrementando.

1.4 Medios utilizados

Para la realización del proyecto han sido necesarios tanto medios hardware como software. Los elementos hardware se han empleado tanto para la fase de desarrollo, como para la de pruebas, mientras que los recursos software se han empleado para la fase de desarrollo y la fase de elaboración de la memoria.

Elementos hardware:

- ordenador potente para programar y simular el software. Puesto que se implanta para un Smartphone, no es necesario tener una gran tarjeta gráfica. El que se ha utilizado ha sido un HP Intel Core i3 con 3 GB de RAM. Cuenta con el sistema operativo Windows 7.
- Smartphone Sony Xperia Tipo para probar el software durante la fase de desarrollo y pruebas. Aunque se dispone de un simulador en el entorno de desarrollo, probarlo en el dispositivo final permite depurar el software con mucha más precisión. Tiene una pantalla táctil con una resolución de 320 X 480 HVGA.

Elementos software:

- Eclipse: en su versión Kepler (4.3, año 2013)
- JDK: Java SE, no es suficiente con JRE
- Android SDK: para comenzar a programar con Android es necesario su instalación. No es un entorno de desarrollo completo, solo incluye las herramientas del núcleo del SDK, que se pueden usar para descargar el resto de los componentes del SDK (como la última versión de la plataforma Android)
- ADT Plugin para Eclipse: Android ofrece un plugin hecho a medida para el IDE de Eclipse llamado ADT (Android Development Tools), y está diseñado para proporcionar un entorno potente y integrado en el que desarrollar aplicaciones de Android. Extiende las capacidades de Eclipse para configurar fácilmente proyectos de Android, crear interfaces de usuario y depurar las aplicaciones utilizando las herramientas del SDK de Android.

- ADV (Android Virtual Device): en el que se simulará el software. El utilizado es Android 2.3 (API level 8) con una pantalla HVGA, para que simule a la perfección el Sony Tipo.
- Microsoft Windows 7: se han usado herramientas como Paint para hacer modificaciones en imágenes.

2. ESTADO EN CUESTIÓN

En este segundo capítulo se contextualiza el uso de aplicaciones en móviles: se presenta una breve historia del origen y evolución de los smartphones, qué sistemas operativos utilizan, cómo se descargan aplicaciones y juegos y algunos de los más importantes para Android. También se muestra en más detalle el sistema operativo Android y como es su uso en terminales Sony. Para finalizar se detallan algunas de las características del terminal Sony Xperia Tipo.

2.1 Historia de los smartphones

El origen [5] de los smartphones data del año 1993. Aquellos primeros modelos se usaban como teléfonos de empresa y sus precios eran prohibitivos para la mayoría de los consumidores. Pero gracias al enorme éxito del iPhone, las operadoras han visto que pueden comprometer a sus clientes a tener una permanencia si se subvencionan las compras de los últimos modelos de smartphones.

El simon (figura 2.1) fue el primer intento real de crear un teléfono que incorporase voz y datos en el mismo dispositivos (actuaba como teléfono móvil, PDA e incluso fax). También disponía de una pantalla táctil que permitía marcar números de teléfono, siendo así el primer precursor del iPhone que nacería 14 años después. Su precio original era de 899 dólares.

Figura 2.1

El Nokia 9110 Communicator (figura 2.2) todavía tenía una gran pantalla gris y no se podía navegar realmente por la web, pero sí que tenía un diseño muy creativo que ha servido como modelo para smartphones como el Motorola Droid.

Figura 2.2

A finales de los noventa, la compañía canadiense Research in Motion sacó al mercado localizadores que usaron millones de personas por todo el mundo. Y fue en 2002 cuando RIM entró en el mercado de la telefonía móvil con su dispositivo BlackBerry 5810 (figura 2.3) con e-mail y capacidad de navegar por la Web. Requería de cascos para poder hablar. Hasta el año 2004, RIM no saco al mercado un dispositivo que no necesitaba cascos (BlackBerry 6210)

Figura 2.3

El Treo 600 (figura 2.4) fue el primer Smartphone lanzado por Palm. Este dispositivo que tenía modelos GSM y CDMA, poseía además una capacidad de 32MB de RAM y procesador de 144MHz.

figura 2.4

El iPhone (figura 2.5) fue el primer intento de Smartphone de Apple que integraba pantalla táctil y la mejor experiencia para navegar por la red ofrecida hasta entonces. A día de hoy, todavía es comparado con el resto de smartphones.

figura 2.5

Se ha de destacar el crecimiento de Android (figura 2.6), sistema operativo para móviles que fue lanzado en el otoño de 2007. El éxito del sistema operativo de código abierto, es más impresionante si se tiene en cuenta la cantidad de OS muy poderosos que ya estaban en el mercado (iPhone,BlackBerry,Windows Movile o Symbian). Ahora el OS de Google se ha convertido en el mayor proveedor dentro de la industria de los smartphones: a finales de 2012, más de un 60% de los dispositivos en todo USA. Durante una serie de encuestas realizadas en el primer trimestre de 2012, el 60% de los usuarios que pensaban comprar un Smartphone indicaron que preferían el OS de Android en comparación con el 20% que prefería iPhone.

Figura 2.7

El G1 (figura 2.8) de la operadora T-Mobile fue el primer terminal Android [6] y salió al mercado en septiembre de 2008. Fue fabricado y diseñado

por HTC con la supervisión de Google. Este primer terminal tenía un teclado físico además de la pantalla táctil como métodos de control, y se comenzó a comercializar de manera online a través de la web de T-mobile. En su versión para desarrolladores, el G1 fue el responsable de la revolución de Android, ya que los programadores hicieron las primeras aplicaciones para él.

Figura 2.8

Aunque Android llegó al mercado un año antes que el lanzamiento del Droid (figura 2.9), éste fue el primer gran hito para la plataforma Android que le permitió su reconocimiento como marca. Este dispositivo, que fue el primer Smartphone basado en Android que funcionó por la red Verizon, vendió más de un millón de unidades durante sus primeros 74 días en el mercado.

figura 2.9

Con el lanzamiento del HTC EVO 4G de Sprint (figura 2.10), han conseguido que sus usuarios puedan sacar provecho de su red de alta velocidad (tienen la red wifi comercial más rápida de los Estados Unidos). Además de su conectividad, el dispositivo tiene un gran tamaño (800x400 px) y un peso de 170 gr.

Figura 2.10

2.2 Sistemas operativos

El sistema operativo del Smartphone es el que gestiona sus recursos hardware y software. En la actualidad, 7 grandes compañías y sus sistemas operativos, abarcan más del 99% de los dispositivos. Estos 7 sistemas operativos son: Symbian, Research In Motion (RIM), iPhone OS, Microsoft Windows Mobile, Linux, Android y WebOs.

1. Symbian: fue producto de la alianza de varias compañías entre las que se encuentran Nokia, Sony Ericsson, Psion, Samsung, Siemens, Arima, Benq, Lenovo, LG, Motorola, Mitsubishi Eletric, Panasonic o Sharp. La plataforma de Symbian se diseñó con los smartphones en mente y la mayoría de los terminales son de Nokia [7]
2. Research In Motion: compañía de telecomunicaciones y dispositivos inalámbricos canadiense. Es conocida por ser la desarrolladora de la BlackBerry. Su sistema operativo (BlackBerry OS) es propietario y proporciona multitarea. Soporta dispositivos de entrada como pantalla táctil o trackball.
3. iPhone OS: es el sistema operativo para móviles de Apple. Se desarrolló originariamente para el iPhone, pero en la actualidad se ha extendido para su uso en otros dispositivos de Apple como iPod touch, iPad y Apple TV.
4. Microsoft Windows Mobile: sistema operativo desarrollado por Microsoft que se usa tanto en smartphones como en dispositivos móviles. La última versión (Windows Phone 7) establece los requisitos mínimos para pantallas táctiles de alta resolución.
5. Linux: LiMo foundation [8] es un consorcio de la industria cuya finalidad es la de crear un sistema operativo para dispositivos móviles, basado en Linux. Participan en el proyecto empresas como NEC, Panasonic, Vodafone, Telefonica, Orange, Mozilla, Huawei o McAfee, entre otras. Las tecnologías clave de LiMo incluyen una

potente y flexible interfaz de usuario, librerías extendidas para widgets, efectos 3D para widgets, multimedia avanzado, redes sociales y localización.

6. Android: sistema operativo basado en Linux que utiliza el lenguaje de programación Java. Se diseñó para smartphones, pero actualmente soporta otros dispositivos como tablets o netbooks.
7. WebOS: sistema operativo multitarea basado en también en Linux, diseñado para dispositivos de pantalla táctil. Fue desarrollado por Palm Inc, pero actualmente es propiedad de Hewlett-Packard Company.

Las dos mejores actuaciones en 2012 fueron Android y Apple (figura 2.11). Android incrementó su cuota de mercado 22 puntos en 2011-2012 alcanzando un 65%, mientras qe Apple creció 2 puntos desde 2011 llegando a 20%.

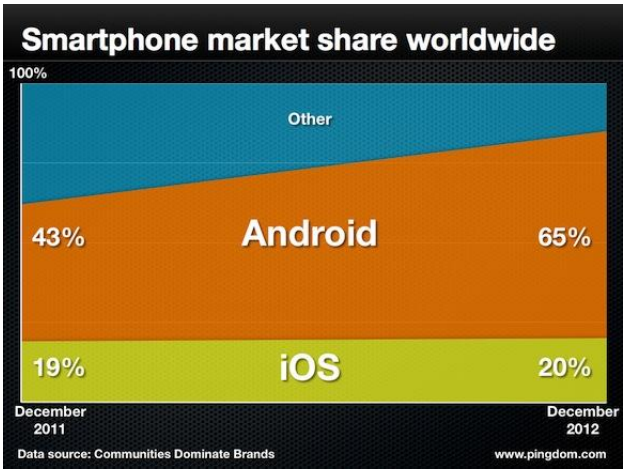


Figura 2.11

2.3 Historia de Android

En junio de 2005 [11] Google compró una pequeña empresa que desarrollaba aplicaciones para móviles llamada Android Inc. Andy Rubin, cofundador de la compañía, pasó a ser el director de la división de plataformas móviles de Google, en ese momento se empezó a especular con que Google tenía la idea de lanzar el "Google pone": el dispositivo y el coste del servicio serían sufragados con publicidad en el mismo.

Aparecieron distintos prototipos y demos no oficiales, y fue en Noviembre de 2007 cuando se anunció la creación de la Open Handset Alliance [12] (organización para la difusión de la plataforma móvil de Android)

Fabricantes de dispositivos y proveedores de servicios se unieron para crear el primer sistema operativo abierto para móviles y que además no estaría atado a un terminal concreto o marca (gracias a su kernel de Linux se podría adaptar a casi cualquier dispositivo) Cinco días después de este anuncio, Google sacó su Software Developmet Kit o SDK que además incluía un emulador.

El primer móvil con Android fue el G1 T-Mobile (HTC Dream) lanzado en septiembre de 2008. Desde este lanzamiento, numerosos fabricantes como Sony Ericsson, LG, Motorola, Samsung o Lenovo han utilizado el sistema operativo de Android.

2.3.1 Qué es Android

Android [13] es un conjunto de software para dispositivos móviles que incluye sistema operativo, middleware y aplicaciones. El SDK de Android proporciona las herramientas y APIs necesarios para empezar a desarrollar aplicaciones en su plataforma utilizado el lenguaje de programación de Java

2.3.2 Arquitectura de Android

El siguiente gráfico (figura 2.12) muestra los distintos componentes del sistema operativo de Android.

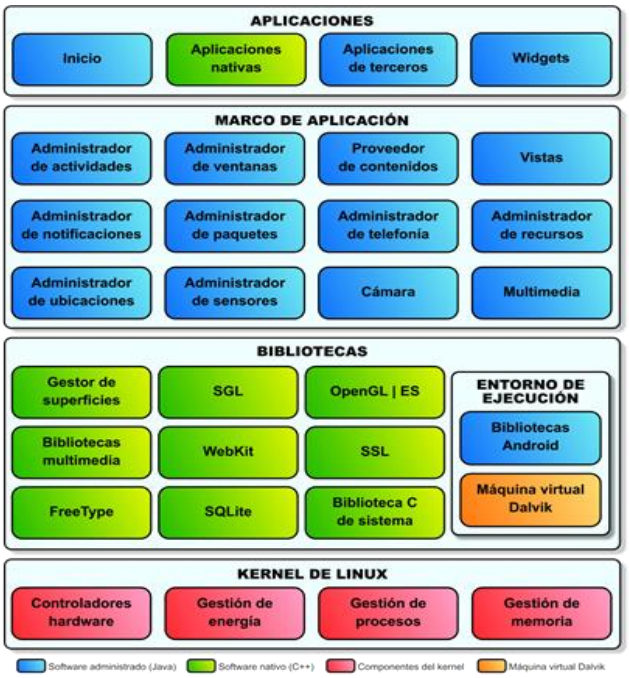


Figura 2.12

Aplicaciones

Android proporciona una serie de aplicaciones de core como cliente de email, programa de SMS, calendario, mapas, explorador, contacto, etc. Todas las aplicaciones están escritas utilizando el lenguaje de programación de Java.

Framework de aplicaciones

Al proporcionar una plataforma de desarrollo abierta, Android ofrece a los desarrolladores la posibilidad de programar aplicaciones muy brillantes e innovadoras. Los desarrolladores son libres para beneficiarse del hardware del dispositivo, acceder a la información de localización, correr servicios en segundo plano, activar alarmas, añadir notificaciones a la barra de estado, y mucho, mucho más.

Los desarrolladores tienen acceso completo al mismo API utilizado para las aplicaciones de core. La arquitectura de aplicación está diseñada para simplificar la reutilización de componentes, cualquier aplicación puede publicar sus capacidades y cualquier otra puede hacer uso de esas capacidades (sujeto a restricciones de seguridad impuestas por el framework). Este mismo mecanismo permite al usuario reemplazar componentes.

Todas las aplicaciones son un conjunto de servicios y sistemas, incluyendo:

- Un conjunto de **Views** que se usa para desarrollar aplicaciones, incluyendo listas, cuadriculas, cuadros de texto, botones e incluso explorador web embebido
- **Proveedores de contenido** que permiten a las aplicaciones acceder a datos de otras aplicaciones (por ejemplo de los contactos), o compartir sus propios datos
- Un **gestor de recursos**, proporcionando acceso a recursos como strings, gráficos, y ficheros de layout
- Un **gestor de notificaciones** que permite a todas las aplicaciones mostrar alertas personalizadas en la barra de estado
- Un **gestor de actividad** que gestiona el ciclode vida de las aplicaciones y proporciona una navegación común

Librerías

Android incluye un conjunto de librerías de C/C++ que usan varios componentes del sistema Android. Algunas de estas librerías:

- **System C**
- **Media**
- **Surface Manager**
- **LibWebCore**
- **SGL**
- **librerías 3D**
- **FreeType**
- **SQLite**

Android Runtime

Android incluye un conjunto de librerías de core que proporcionan la mayoría de las funciones disponibles de las librerías de core del lenguaje de programación de Java.

Cada aplicación de Android corre sobre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik se implementó de forma que un dispositivo puede correr varias máquinas virtuales eficientemente. La Dalvik VM ejecuta ficheros en el formato .dex que está optimizado para un uso mínimo de memoria.

Kernel de Linux

Android utiliza la verión 2.6 para servicios de sistema del core como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de driver. El kernel actúa también como una capa abstracta entre el hardware y el resto de la pila del software.

2.3.3 Fundamentos de las aplicaciones

Las aplicaciones [14] de Android están escritas en lenguaje de programación Java. Las herramientas del SDK de Android compilan el código (junto con otros ficheros de datos y recursos) en un paquete de Android, y genera un fichero con una extensión .apk. Todo el código en un único .apk se considera como la aplicación y es el fichero que los dispositivos basados en Android utilizan para instalar la aplicación.

Una vez instalado en el dispositivo, cada aplicación de Android tiene su propia zona de seguridad:

- El sistema operativo de Android es un sistema de Linux multiusuario en el que cada aplicación es un usuario diferente.

- Por defecto, el sistema asigna a cada aplicación un identificativo de usuario de Linux único (el identificativo es utilizado solo por el sistema y es conocido por la aplicación). El sistema pone permisos a todos los archivos de una aplicación de manera que solo el ID de usuario asignado a la aplicación puede acceder a él.
- Cada proceso tiene su propia VM, así que un código de aplicación corre de manera aislada a otras aplicaciones.
- Por defecto, cada aplicación corre en su propio proceso de Linux. Android comienza el proceso cuando cualquiera de los componentes de la aplicación necesitan ser ejecutado, después finaliza el proceso cuando ya no es necesario o cuando el sistema debe recuperar memoria para otras aplicaciones.

De esta forma, el sistema de Android implementa el principio del menor privilegio. Esto es, cada aplicación por defecto, tiene acceso solo a los componentes que requiere para hacer su trabajo y no más. Esto crea un medio muy seguro en el que una aplicación no puede acceder a partes del sistema para las que no tenga permisos.

Los componentes de las aplicaciones son los bloques esenciales para construir una aplicación de Android. Cada componente es un punto diferente a través del cual el sistema puede entrar en la aplicación. No todos los componentes son puntos reales de entrada para el usuario y algunos dependen de otros, pero cada uno existe como una entidad en si mismo y juega un rol especifico.

Hay cuatro tipos de componentes de aplicaciones:

- **Actividades**: una actividad representa una única pantalla con una interfaz de usuario. Una actividad se implemente como una subclase de Activity.
- **Servicios**: no se corresponde con una interfaz de usuario, es un componente que corre de fondo sin bloquear otros procesos. Implemente una subclase de Service.
- **Proveedores de contenido**: gestiona un conjunto de datos compartidos entre diferentes aplicaciones. SE implemente como una subclase de ContentProvider.
- **Broadcast receivers**: es un componente que responde a anuncios de Boradcast de sistema. Se implementa como subclase de BroadcastReceiver.

Las actividades, los servicios y los broadcast receivers se activan por medio de un mensaje asíncrono llamado intent.

El archivo del manifiesto (AndroidManifest.xml) muestra todos los componentes que tiene definidos la aplicación. Además de declarar los componentes, identifica los permisos de usuario que requiere la aplicación, declara el nivel de API mínimo, los usos de hardware y software que necesitan, etc.

3. *ANÁLISIS, DISENO E IMPLEMENTACIÓN*

En este tercer capítulo del Proyecto Fin de Carrera se incluye una descripción del problema y un estudio de las alternativas. Es el más importante ya que describe todo el proceso, desde la primera idea hasta su puesta en marcha en el dispositivo del usuario.

Este capítulo se divide en tres partes. La primera de ella define el análisis, cuáles fueron los primeros pasos del proyecto. La segunda parte muestra el diseño y las características del software que sirven de base para la implementación. La tercera parte muestra la implementación en la que se detallan algunas partes remarcables del código para mejor entendimiento del mismo.

3.1 ANÁLISIS

3.1.1 Propuesta inicial

La primera idea planteada para la realización de este proyecto, es la de implementar una de las funcionalidades más básicas en cualquier teléfono móvil. En el comienzo de este proyecto, la aplicación tan solo contaba con las más básicas, cómo definir alarmas a una hora puntual.

Al tener sus funciones implementadas para personas especiales, las pantallas mostradas más adelante, no cuentan con movimientos dinámicos, y cambios bruscos entre imágenes. En vez de eso, cuentan con imágenes en blanco y negro, que destaca más el contenido de dichas pantallas.

La idea principal es indicar en cada momento donde se encuentra el usuario dentro de la aplicación. Esto se consigue con los colores explicados anteriormente, y con el siguiente método: si el usuario hace un click corto sobre cualquier botón o parte de la pantalla, esta le contesta diciendo donde ha apretado. Si realiza un click largo, hace de confirmación, es decir, entra en dicha función.

3.1.2 Requisitos de usuario

Una vez analizados los objetivos e idea inicial, se realiza la extracción de los requisitos de usuario.

Con los requisitos de usuario, el mismo indica una serie de restricciones o funcionalidades que debe cumplir el programa, juego o aplicación. Las especificaciones siguientes intentar regirse a la Especificación de Requisitos Software (ERS), que es una descripción complete del comportamiento del sistema que se va a desarrollar, incluyendo las siguientes prácticas:

- *Completa*. Todos los requerimientos deben estar reflejados en ella y todas las referencias deben estar definidas.
- *Consistente*. Debe ser coherente con los propios requerimientos y también con otros documentos de especificación.
- *Inequívoca*. La redacción debe ser clara de modo que no se pueda mal interpretar.
- *Correcta*. El software debe cumplir con los requisitos de la especificación.
- *Trazable*. Se refiere a la posibilidad de verificar la historia, ubicación o aplicación de un ítem a través de su identificación almacenada y documentada.
- *Priorizable*. Los requisitos deben poder organizarse jerárquicamente según su relevancia para el negocio y clasificándolos en esenciales, condicionales y opcionales.
- *Modificable*. Aunque todo requerimiento es modificable, se refiere a que debe ser fácilmente modificable.
- *Verificable*. Debe existir un método finito sin costo para poder probarlo.

Los requisitos de usuario están definidos por los siguientes atributos:

- **Identificador**: muestra el tipo de atributo RUC (Requisito de Usuario de Capacidad) o RUR (Requisito de Usuario de Restricción), seguido del número del requisito. Este identificativo es único.

- **Necesidad**: indica la prioridad del requisito que puede ser Esencial, Deseable u Opcional. Para el análisis de la aplicación, no se han tenido en cuenta los requisitos opcionales.
- **Titulo**: frase con la que se da a conocer el nombre del requisito.
- **Comentario**:pequena descripción del requisito.

A continuación se muestran los requisitos de usuario de capacidad:

Identificador:	RUC-01	Necesidad:	ESENCIAL
Título:	Lanzamiento de la aplicación		
Descripción:	La aplicación debe lanzarse al hacer click sobre el icono.		

Figura 3.1

Identificador:	RUC-02	Necesidad:	ESENCIAL
Título:	Instrucciones		
Descripción:	La aplicacion debe contar con claras instrucciones para el usuario		

Figura 3.2

Identificador:	RUC-03	Necesidad:	DESEABLE
Título:	Mensajes de información		
Descripción:	Debe mostrarse información de audio cuándo el usuario se pierde entre las pantallas.		

Figura 3.3

Identificador:	RUC-04	Necesidad:	DESEABLE
Título:	Definir días de semana		
Descripción:	La alarma debe contar con una pantalla a parte para configurar los días que un usuario quiere que suene el despertador.		

Figura 3.4

Identificador:	RUC-05	Necesidad:	ESENCIAL
Título:	Definir hora		
Descripción:	Es una necesidad básica, ya que estamos hablando de un despertador.		

Figura 3.5

Identificador:	RUC-07	Necesidad:	DESEABLE
Título:	Consultar alarmas		
Descripción:	La aplicación debe contar con una consulta sencilla de las alarmas ya definidas		

Figura 3.6

Identificador:	RUC-08	Necesidad:	DESEABLE
Título:	Sonido confirmación		
Descripción:	La aplicación debe responder con audio confirmando si el usuario ejecuta algún comando (entrar en x)		

Figura 3.7

Identificador:	RUC-09	Necesidad:	DESEABLE
Título:	Icono ejecutable		
Descripción:	Aunque es una aplicación home, es deseable también disenar un logotipo bueno para su ejecutable		

Figura 3.8

A continuación se muestra los requisitos de usuario de restricción:

Identificador:	RUR-01	Necesidad:	ESENCIAL
Título:	Plataforma Android		
Descripción:	La aplicación está hecha para Android.		

Figura 3.9

Identificador:	RUR-02	Necesidad:	ESENCIAL
Título:	Sony Xperia Tipo		
Descripción:	La aplicación tiene que ser compatible con el Sony Xperia Tipo		

Figura 3.10

Identificador:	RUR-03	Necesidad:	DESEABLE
Título:	Sonido		
Descripción:	El usuario puede subir y bajar el volumen de multimedia, y llamadas.		

Figura 3.11

Identificador:	RUR-04	Necesidad:	ESENCIAL
Título:	Plataforma Android		
Descripción:	La aplicación está hecha para Android.		

Figura 3.12

Identificador:	RUR-05	Necesidad:	DESEADO
Título:	Formatos		
Descripción:	Las imagenes deben tener un formato universal, por la uniformidad del programa, que será PNG, y los sonidos serán MP3.		

Figura 3.13

3.1.3 Requisitos del software

Los requisitos software indican lo que debe hacer el sistema (requisitos funcionales) y la forma en que se debe llevar a cabo (requisitos de rendimiento, interfaz, operación, recursos, comprobación, documentación, seguridad, calidad, mantenimiento, dano y aceptación de pruebas)

Al igual que los requisitos de usuario, los requisitos software vienen definidos por:

- **Identificador**: muestra el tipo de atributo RS (Requisitos Software Funcional. Este identificativo es único
- **Necesidad**: indica la prioridad del requisito que puede ser Esencial, Deseable u Opcional. Para este análisis, las necesidades opcionales no se ha tenido en cuenta.
- **Descripción**: pequena descripción del requisito

Para el análisis de la aplicación, no se han definido todos los tipos de equisitos software.

A continuación se muestran algunos de los requisitos software funcionales:

Identificador:	RSF-01	Necesidad:	ESENCIAL
Título:	Lanzar aplicación		
Fuente	RUC-01,RUR-01		
Descripción:	La aplicación se lanzará mediante la ejecución de un fichero .apk que se genera al compilar desde eclipse.		

Figura 3.14

Identificador:	RSF-02	Necesidad:	ESENCIAL
Título:	Pantalla principal		
Fuente	RUC-02, RUC-03, RUC-04, RUC-05, RUC-07		

Descripción:	Cuando se lanza la aplicación, se debe mostrar una pantalla con las distintas opciones disponibles, entre ellos: Agenda (llamadas), mensajes (sms), Hora (alarmas, fecha, hora, cronometro), etc

Figura 3.15

Los requisitos software de Rendimiento definen valores relacionados con el rendimiento (velocidad de procesado, o ejecución del sistema software):

Identificador:	RSR-01	Necesidad:	ESENCIAL
Título:	Cargar la aplicación		
Descripción:	La aplicación debe cargar en lo menos tiempo posible. Esto se consigue solo cargando la pantalla principal, y lanzando los servicios y procesos asincronos de fondo.		

Figura 3.17

Identificador:	RSR-02	Necesidad:	ESENCIAL
Título:	Actualización de pantallas		
Descripción:	La aplicación debe avisar claramente entre el cambio de las pantallas, y a su vez debe tardar lo menos posible		

Figura 3.18

Identificador:	RSR-03	Necesidad:	ESENCIAL
Título:	Layouts		
Descripción:	Las diferentes pantallas deben seguir el formato universal Android, definidas en XML.		

Figura 3.19

3.1.4 Casos de Uso

Los casos de uso ayudan a definir a grandes gasgos la relación que hay entre actores (usuario) y el sistema. El siguiente dibujo es un caso concreto, donde se ha tomado cómo ejemplo 4 casos.

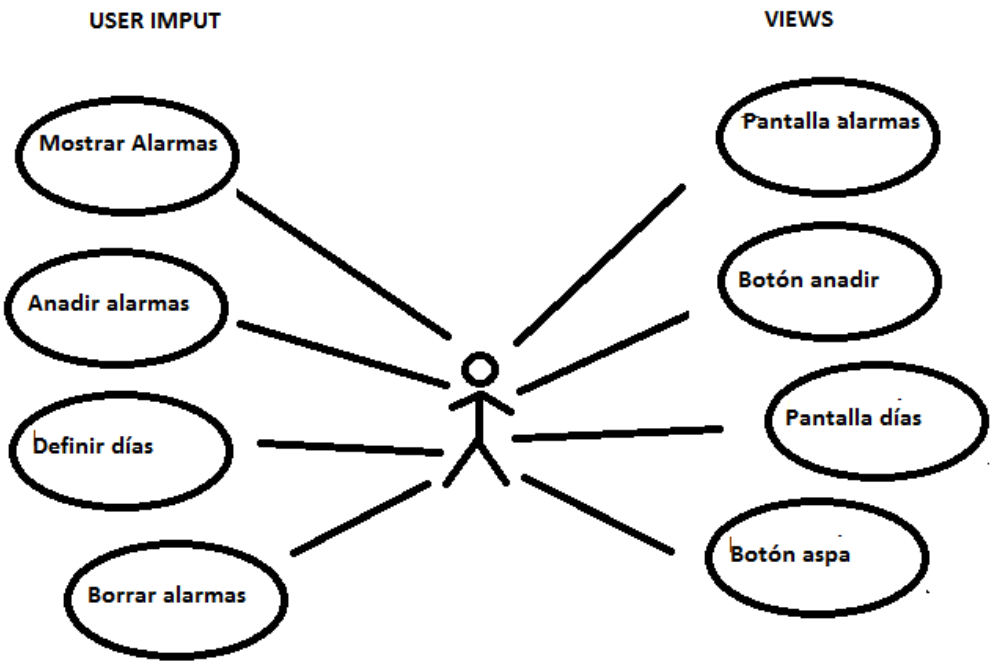


Figura 3.20

3.1.5 Diagrama de actividad del sistema

Una vez se han definido las interacciones entre el sistema y el usuario a través de los casos de uso, se especifica el diagrama de actividad: se muestra mediante diagramas de flujo la secuencia de actividades que seguirá la aplicación.

Un diagrama de actividad es una representación gráfica del flujo de trabajo, en el que las actividades y acciones pueden estar asociadas a una elección, una interacción, una base de datos, o una concurrencia. Gracias al diagrama de actividad se comprende mejor las transiciones y eventos del software.

Por simplicidad sólo se muestra una parte pequeña de la aplicación. En los capítulos siguientes se verán imágenes de todas las pantallas disponibles. El siguiente diagrama de esto muestra una profundidad de 3 pantallas, de los cuales se ha querido seleccionar 3 al azar para mostrar el flujo de la aplicación. Al ser una figura reducida de tamaño, para no confundir el lector, se ha decidido mostrar con flechas los caminos hacía atrás, mientras que el flujo hacía abajo no tiene flechas; figura 3.21

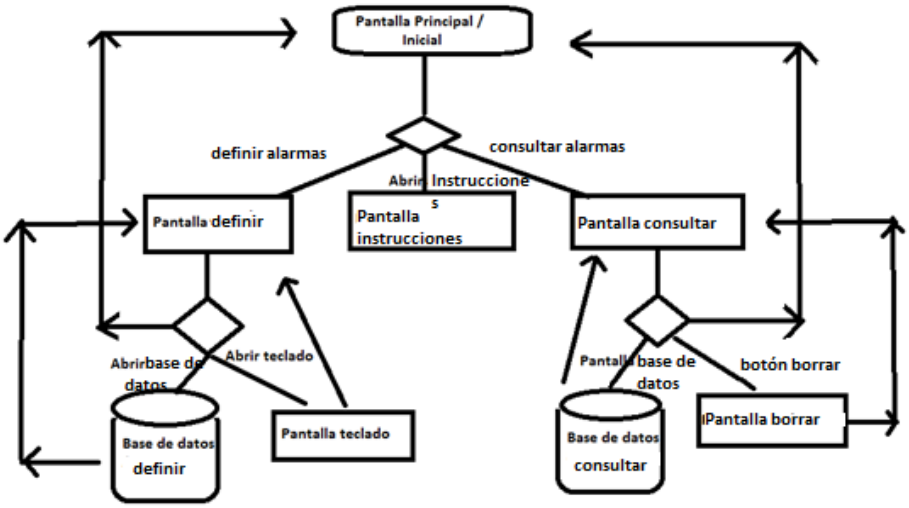


Figura 3.21

3.1.6 Diagramas de secuencia

Gracias a los diagramas de secuencia, se puede representar la forma en que varios objetos se comunican entre sí para la realización de los casos de uso.

Con los diagramas de secuencia se facilita la creación del modelo de clases, ya que contiene información sobre la implementación del escenario (incluyendo objetos y clases, y mensajes intercambiados entre objetos).

La figura 3.22 representa el diagrama de secuencia de la aplicación.

La clase MainActivity es la que inicializa el sistema al comienzo de la aplicación, lleva el control de las alarmas y también la gestión de su definición. Una vez se inicializan los datos, se guardan en una base de datos SQLite, que veremos en la siguiente sección.

La clase ListaDías nos ayuda escoger los días que queremos que suene una alarma especifica. Podemos seleccionar días diferentes para cada una de ellas.

El onBootreceiver es un servicio que se pone en marcha nada más encender el móvil, es decir, independientemente de la aplicación. Esto sirve para mantener las alarmas ya definidas, y no dormirnos☺

El OneShotAlarm gestiona el momento de sonar una alarma, es decir, visualiza una mini-ventana avisando que está sonando la alarma, aparte de emitir un sonido.

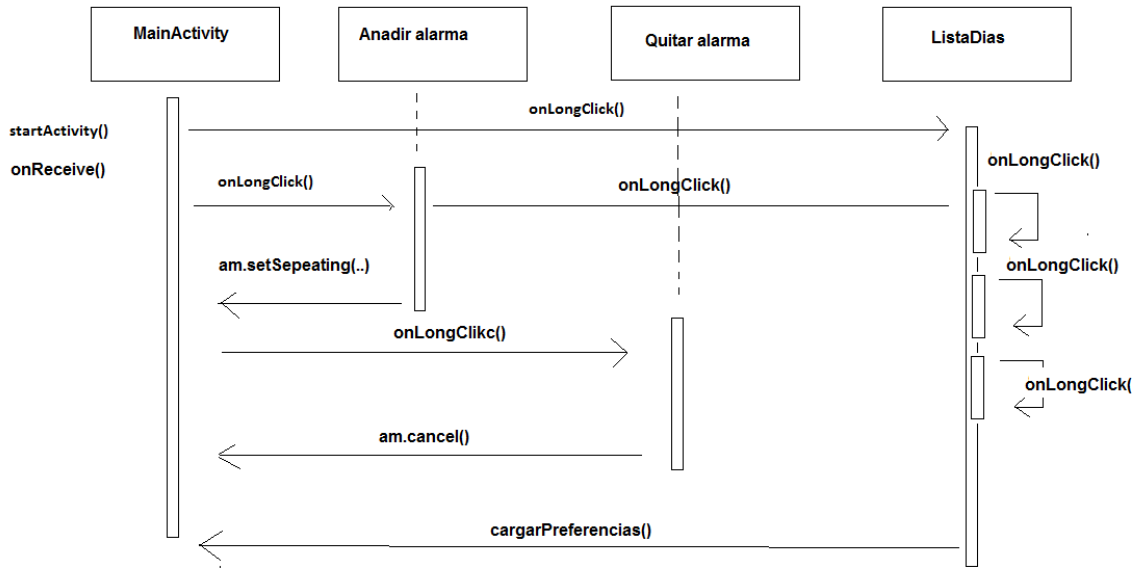


Figura 3.22

3.2 DISEÑO

Con los datos obtenidos durante la fase de análisis, se realiza un diseño de los componentes, de forma que todos los aspectos queden claras y definidos para la fase de implementación. Con toda esta información, durante la fase de implementación no será necesario replantear ningún punto. El diseño se ha de adaptar a las necesidades del software, y las clases se han de definir de manera exhaustiva, de manera que las relaciones entre ellas queden claras.

Si la fase de diseño no se realiza de manera concienzuda, puede repercutir en una necesidad de replantear el diseño, que implicaría un cambio en la planificación del proyecto y por lo tanto en su fecha de entrega.

Esta sección se ha estructurado en varios aparatos:

- **Arquitectura** define a alto nivel la estructura de la aplicación
- **Pantallas** muestra las distintas interfaces a utilizar
- **Carpetas del proyecto** contienen el árbol con la organización de las mismas
- **Diagramas de clases** con sus dependencias entre ellas

3.2.1 Arquitectura

La arquitectura[18] del software proporciona el diseño de más alto nivel de la estructura de la aplicación y define los módulos principales, las responsabilidades de cada módulo, la interacción que existirá entre dichos módulos y el control de flujo y datos. Aporta una visión abstracta de alto nivel, dejando el detalle de cada módulo a fases posteriores del diseño.

La arquitectura del software se divide en tres componentes principales.

El componente más importante es el core de la aplicación: inicializa componentes y gestiona todo el flujo. La entrada/salida engloba las interacciones del usuario con el software y las vistas de los menús.

Figura 3.23

La Figura 3.23 muestra la arquitectura del usuario.

3.2.2 Pantallas

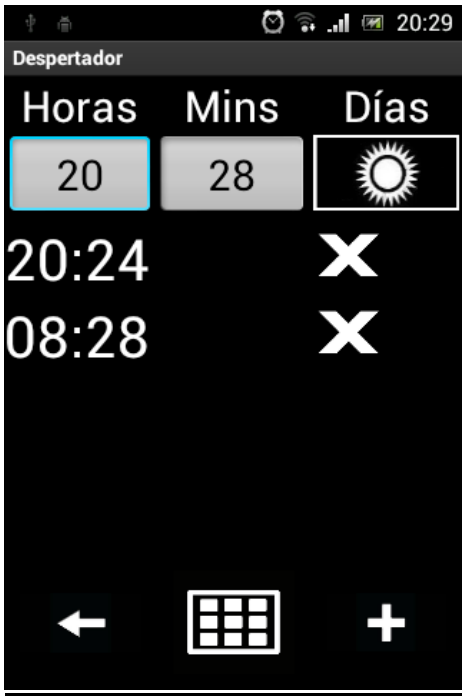
En este apartado se definen los diseños preliminares que se realizaron de las diferentes pantallas.

Puesto que la pantalla del dispositivo móvil es de un tamaño muy limitado, para poder aprovechar este al máximo, todas las pantallas están forzadas a una orientación vertical.

La Figura 3.24 representa el diseño de la pantalla de inicio de la Alarma. Esta pantalla engloba un poco las funciones de la aplicación, permite ir a la pantalla de los días, añadir alarmas, invocar el teclado (un teclado especial con botones grandes, diseñados especialmente para gente con problemas visuales).

La Figura 3.25 muestra la pantalla con los días. En esta pantalla se puede seleccionar los días que uno quiere poner las alarmas. Intenta ser lo más sencilla posible.

La Figura 3.26 muestra el teclado especial mencionado anteriormente.



La Figura 3.24

La Figura 3.25

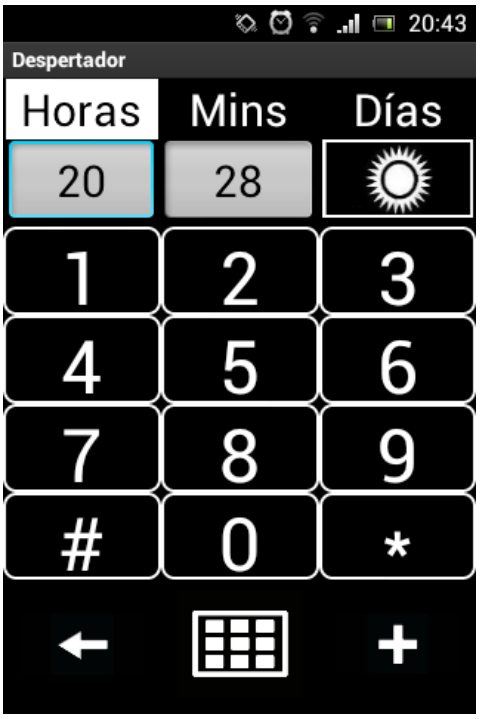


Figura 3.26

3.2.3 Carpetas del proyecto

Al programar con Android y Eclipse, la estructura del proyecto[19] queda definida en la figura 3.27

La carpeta **/src** almacena todos los ficheros fuente de Java que se van creando durante el desarrollo del juego.

La carpeta **/gen** contiene el archivo R.java que se genera automáticamente por la plataforma Android. Este archivo es un índice con todos los recursos definidos en el fichero y ayuda a a localizar rápida e interactivamente la referencia específica que se está buscando

/**Android 2.2** contiene las librerías (jars) que son necesarias para el proyecto. El 2.2 indica el número de versión.

La carpeta **/res** contiene todos los recursos externos como imágenes, ficheros de datos, archivos de audio, etc. En **/res/drawable** se guardan todas las imágenes, iconos o recursos visuales que se usan en el software. **/res/layout** contiene los layouts (en formato XML) de las diferentes interfaces de usuario /res/menu almacene el recurso de menú: se trata de un archivo XML

que define las opciones posibles cuando se pulsa el botón menú del smartphone. Para guardar los recursos de audio (.mp3) se utiliza la carpeta **/res/raw. /res/valus** que también contiene archivos XML, declaran arrays y strings que se referencian en la aplicación, de tal manera que se pueden cambiar sus valores localmente, sin necesidad de cambiar el código fuente. La carpeta **/assets** también puede contener recursos externos como en la carpeta /res, sin embargo estos archivos se han de guardar en formato raw, por lo que se ha hecho uso de la misma.

Existen archivos sueltos, que no están en ninguna de las carpetas descritas anteriormente, sino que cuelgan de la raíz. Uno de esos archivos es uno de los más importantes, **AndroidManifest.XML** Dicho fichero controla los permisos que la aplicación puede tener, en este caso: Establecer alarmas, Recibir la señal del encendido del móvil, y más. La implementación se encuentra en el siguiente capítulo.

En las siguientes imágenes observamos las carpetas anteriormente descritas, y en otra las carpetas RES.

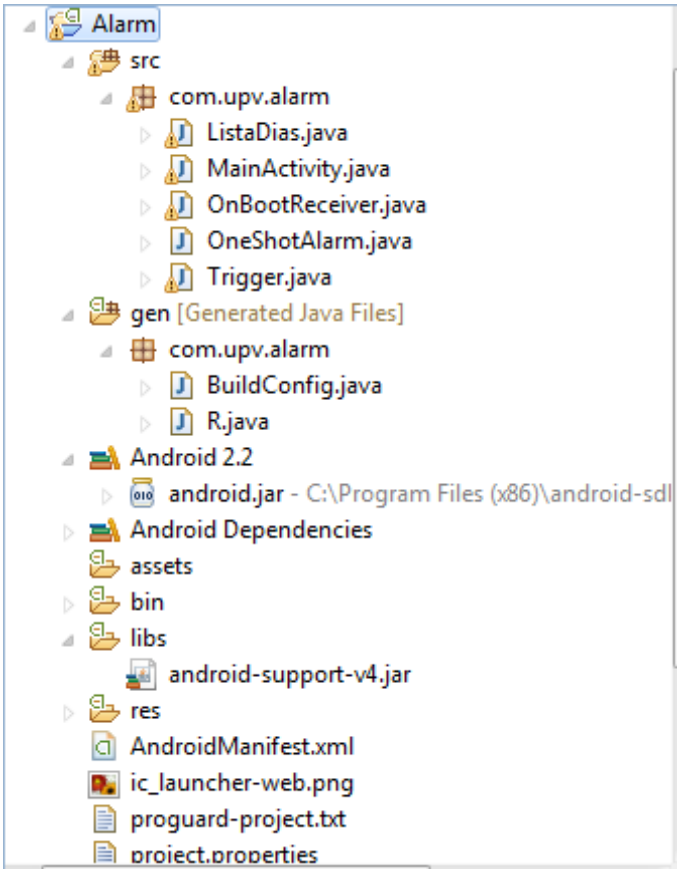


Figura 3.27

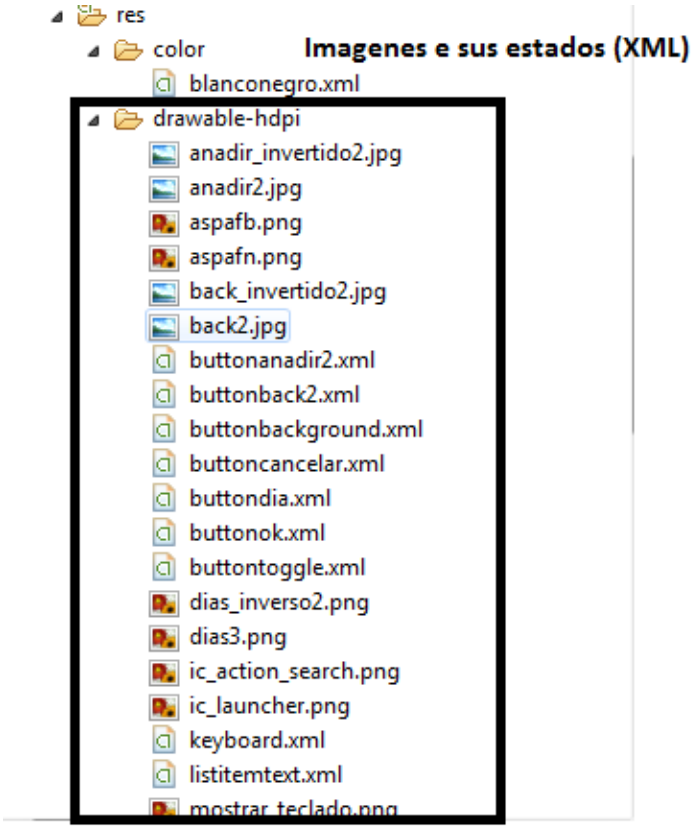


Figura 3.28

En la siguiente imagen vemos las pantallas en xml, y los valores de los textos. Esos valores están disponibles en varios idiomas, en este caso español, e inglés. Esto sirve para separar claramente código e texto, y disponer de varios idiomas para poder comercializar el producto en varios países;

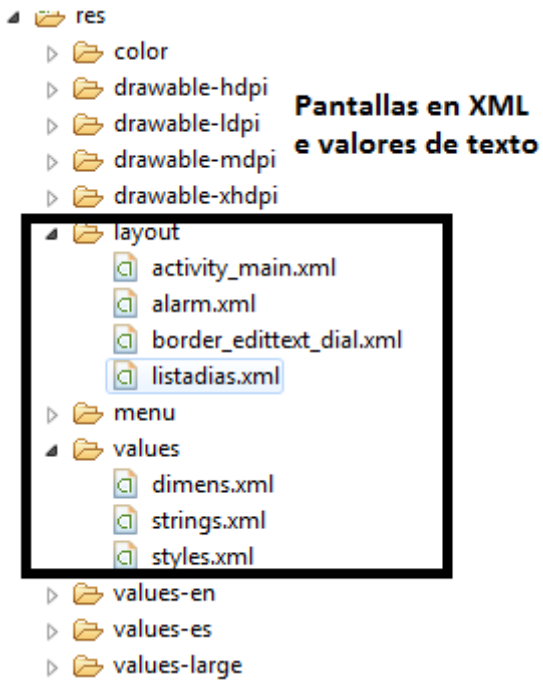


Figura 3.29

3.2.4 Diagrama de clase

En este apartado se definen las clases a utilizar y cuáles son las relaciones entre ellas. Según las buenas prácticas para programar una aplicación móvil con Android, se necesita una clase por cada vista o pantalla. Además de estas clases, se tienen otras clases auxiliares necesarias para la correcta gestión del software. El diagrama de clases representado en la figura 3.29, muestra todas las que se han definido en este proyecto.

Todas las clases que extienden de Activity han de ir definidas en un archivo .xml llamado AndroidManifest. Una activity es una entidad sencilla que se usa para llevar a cabo acciones. Una aplicación puede tener como es el caso, muchas activities diferentes, pero el usuario sólo interactúa con ellas de una en una.

Otras clase extienden de View, y otras son las que se han definido para ser auxiliares.

La clase OneShotAlarm e OnBootReceiver extienden de BroadcastReceiver, que es para recibir señales externas cómo una llamada, recibir sms, o encender el móvil, que es el caso.

Además de los métodos indicados, también se han definido gets y sets para recuperar o editar los valores de los atributos privados.

A continuación se explica las clases empleadas en el proyecto:

MainActivity: es la clase principal, también define la pantalla principal. Tiene muchos atributos, entre ellos:

```
public class MainActivity extends Activity {

    private Vibrator vibr;
    private LinearLayout layoutAlarmas, layoutNumbers, linlay;
    private TextToSpeech mTts;
    private int soundStatus;
    private ImageButton buttonOn, buttonB, buttonkey;
    private TextView hor, minu;
    private static final long[] patternCancel = { 0, 200, 50, 200 };
    private static final long[] patternOk = { 0, 100, 50, 100 };
    private static final long[] NOTIFICATION_OR_WINDOW_STATE_CHANGED_PATTERN = new long[] {
            0, 25, 50, 25, 50, 25 };
    private static final long[] patternMiddle2 = { 0, 100, 100, 100, 50, 100 };
    private final String BD_NOMBRE = "baseDatosAlarmas";
    private final String BD_TABLA = "alarmas";
    int alarmCount;
    private SQLiteDatabase myDB;
    private Button button1;
    private Button button2;
    private Button button3;
    private Button button4;
    private Button button5;
    private Button button6;
    private Button button7;
    private Button button8;
    private Button button9;
    private Button button10;
    private Button button_al;
    private Button button_as;
    private ImageButton button_fecha;
    private int iadd;
    private TextView horastxt, minstxt;
    private String diasalarma;
    private int dia;
```

Figura 3.30

Estos variables globales son privados por el principio de encapsulación. Vemos los botones que se utiliza para mostrar el teclado, una base de datos SQLiteDatabase, que utiliza para guardar las alarmas, patrones para la vibración, e campos de texto (hora, minuto).

A continuación explican algunos de los métodos que esta clase utiliza:

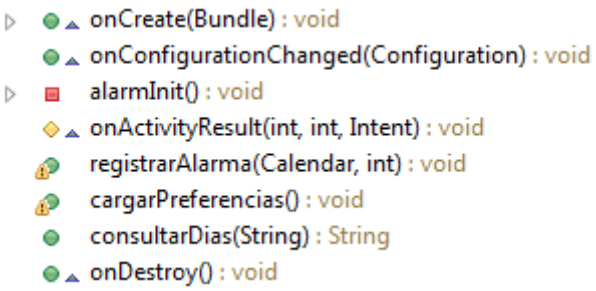


Figura 3.31

Donde onCreate es un método propio para cargar la pantalla, alarmInit() es para iniciar alarmas, registrarAlarma es para registrar una nueva alarma, constularDias es para consultar los días activados en la otra pantalla. Finalmente onDestroy es para destruir la clase al salir de la pantalla.

Otra de las clases que extienden de Activity es **ListaDias**. Esta clase tiene muchos menos variables:

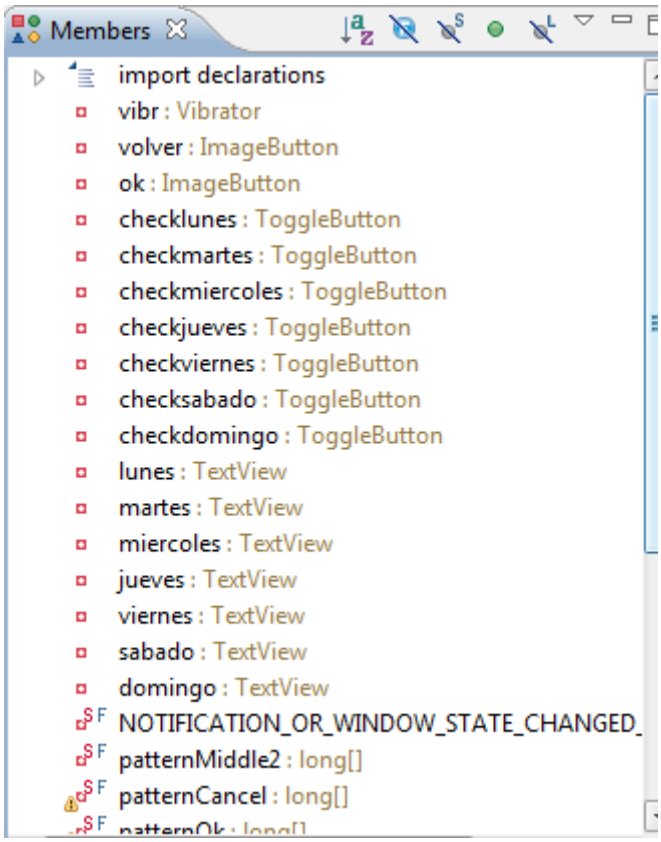


Figura 3.32

Donde el nombre de los días son los textos, y los días empezando en „check" son las que ponen la flecha o aspa, dependiendo si se activa el día o no.

Sus métodos son los siguientes:

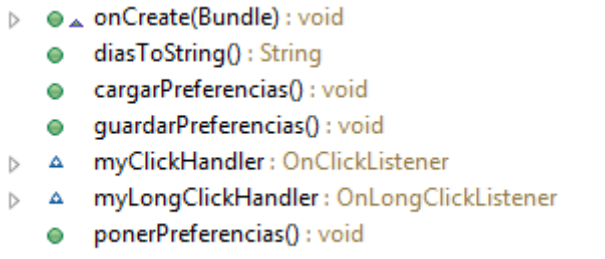


Figura 3.33

Donde los importantes son diasToString que convierte los campos de texto a String, cargarPreferencias es para cargar los días que ya estaban activos anteriormente, y guardarPreferencias es para guardar la nueva configuración.

En las dos clases que extienden de broadCastReceiver tienen pocas variables y métodos, el más importante es onReceive, que es el que controla las señales recibidas de Android.

En la siguiente imagen mostramos un diagrama de clases, relación entre clases;

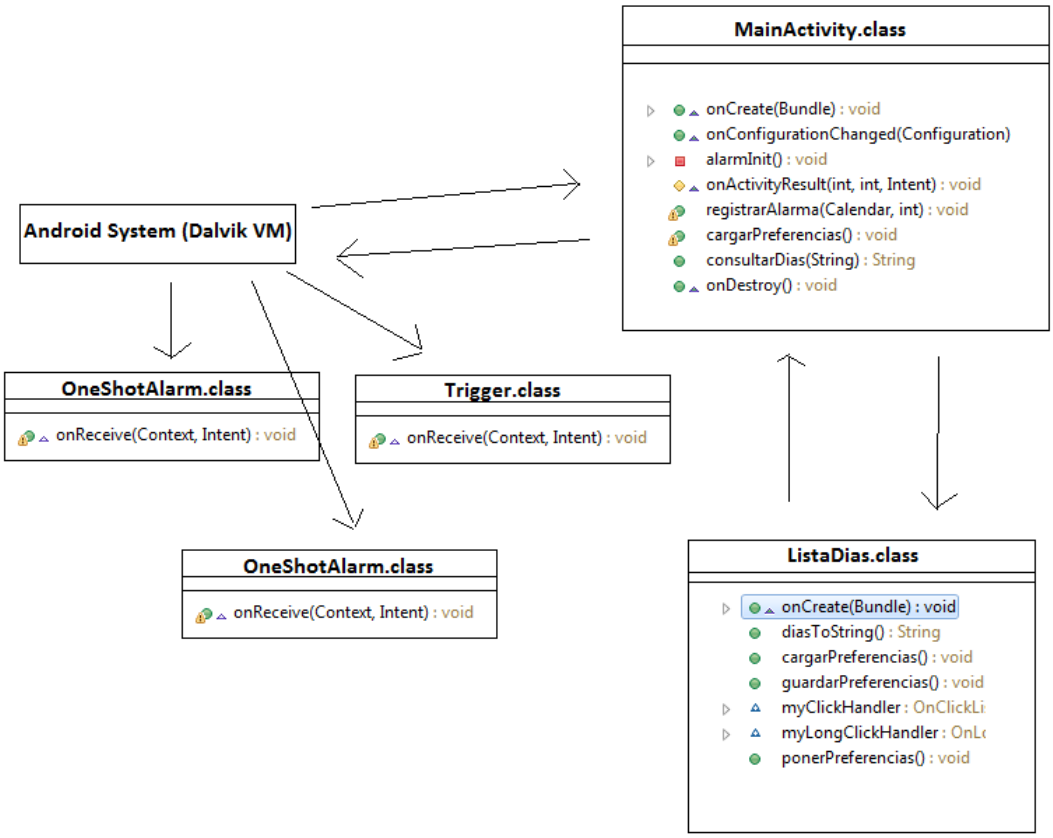


Figura 3.34

3.3 IMPLEMENTACIÓN

En esta sección voy a comentar el código de la aplicación, y detallaré los aspectos importantes y relevantes de esta fase. Sirve como guía de esta fase el estudio anterior de análisis y diseño, y se sigue un ciclo de vida funcional: la aplicación es operativo desde las primeras etapas de la implementación y se van incrementando las funcionalidades.

El capítulo se organiza de forma paralela al desarrollo de la alarma: en primer lugar se implementó una correcta navegación entre pantallas, a continuación se desarrollaron las vistas e interfaces de las distintas pantallas y por último de desarrolló en núcleo de la aplicación. En último lugar se añadieron detalles como cuadros de diálogo, sonidos. Además de estos apartados, el primero de ellos muestra que cambios fueron necesarios durante esta fase.

3.3.1 Implementación previa y cambios

Antes de comenzar con la implementación, fue necesario seguir tutoriales de Android para llegar a familiarizarse con el entorno de trabajo y la plataforma [20].

Durante la fase de implementación se tuvo que cambiar el diseño de sistema de teclado. En un primer momento se planteó como un teclado normal, al que puede ser llamado a través de un método (API) de Android. Dado que la aplicación intentó ser en su totalidad adaptado a discapacitados, el teclado tuvo que ser rediseñado.

Otro problema que surgió durante la fase de implementación fue el de la orientación de las pantallas. Android permite que la orientación de la pantalla se cambie automáticamente cuando el teléfono se gura y esto es factible de implementar, ya que se pueden definir diferentes layouts dependiendo de la orientación (los layouts se ven en apartados sucesivos). Dado que el tamaño de la pantalla es muy limitado, si la orientación es variable no se aprovechaba la pantalla en su totalidad. Sin embargo la razón principal de prohibir esa propiedad es la de confusión, si un usuario se acostumbra a tener los botones y campos de texto en un sitio, al girar eso cambiaría, y confundiría al usuario.

Por ello se cambió la orientación (Figura 3.35) para que fuese siempre vertical;

```
android:screenOrientation="nosensor" >
```

Figura 3.35

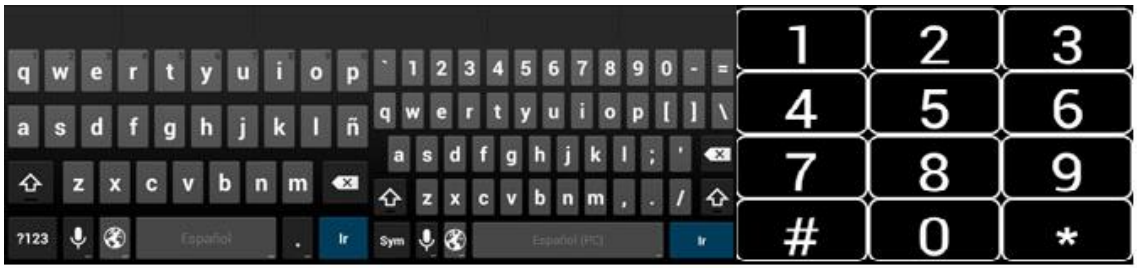


Figura 3.36 – Comparación teclados

3.3.2 Navegación entre distintas pantallas

La primera fase de implementación se centró en la navegación entre pantallas (figura 3.37). Se buscaba que pulsando los distintos botones, se fuese navegando de unas pantallas a otras de forma adecuada. Como se ha comentado, es necesaria al menos una Activity por pantalla.

Recordemos que el flujo entre las distintas pantallas es sencilla:

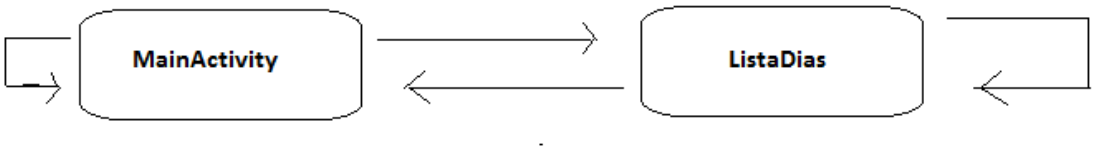


Figura 3.37

El cambio entre pantallas en Android siempre se consigue con un objeto especial llamado Intent (intento), que precisamente indica que es un intento de pasar a otra pantalla, y que puede ser un intento fallido:

```
Intent i = new Intent();
i.setClassName("com.upv.alarm", "com.upv.alarm.ListaDias");
vibr.vibrate(NOTIFICATION_OR_WINDOW_STATE_CHANGED_PATTERN, -1);
startActivityForResult(i,4);
```

Figura 3.37

Donde startActivityForResult indica que queremos marcarle dicha pantalla (con el identificador “4”) para su futura conexión y paso de parámetros entre las 2 clases.

Cuando queremos cerrar una actividad y volver a la pantalla anterior, simplemente tenemos que invocar el siguiente método:

```
finish();
```

3.3.3 Interfaz de usuario

En el apartado de Interfaz de usuario se defina como se realizó la implementación de las pantallas, teniendo en cuenta el funcionamiento de los archivos de layout, la gestión de eventos de la interfaz y los menus.

Layouts

Los layouts[21] son la arquitectura para la interfaz de usuario en una Activity. Define la estructura del layout y contiene todos los elementaos que aparecerán al jugador. Se pueden declarar layouts de dos formas:

- **Declarar elementos de interfay de usuario en XML**: Android proporciona vocabulario XML que corresponde a las calses y subclasese de View. Este tipo de layout se han empleado en las interfaces sencillas como las que muestra el teclado, algunos botones, etc.
- **Instanciar elementos de layout en tiempo de ejecución**: la aplicación puede crear en tiempo de ejecución objetos de View y manipular sus propiedades programando. Esto se ha empleado para interfaces más complicadas como la vista de la pantalla de inicio, donde el teclado tiene alternar con las alarmas ya definidas con sus correspondientes botones. Por ejemplo para la case ListaDías se define un layout con 7 campos de texto días, y 7 toggleBottones. Dichos botones lo que hacen es controlar el estado de cada día, es decir cambiar de estado (por eso toggle), y pueden tener 2 valores binarios. A continuación vemos una de las filas de dicha pantalla, con un campo de texto (lunes) y su correspondiente togglebotton (checklunes).

  Esto se muestra en la Figura 3.38:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="10" >
    <TextView
        android:id="@+id/lunes"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lunes"
        android:textSize="50dp"
        android:textColor="@color/blanconegro"
        android:background="@drawable/buttonbackground"
        android:clickable="true"
        android:longClickable="true" >
    </TextView>

    <ToggleButton
        android:id="@+id/checklunes"
        android:background="@drawable/buttontoggle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:textOff=""
        android:textOn=""
        android:clickable="false"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="10dp"/>

</RelativeLayout>
```

Figura 3.38

Una vez se han definido el archivo XML, se carga desde el código (Figura 3.39). EN el método onCreate(), se llama a setContentView, pasando como referencia el layout correspondiente con el formato R.layout.listaDias. A continuación se crean instancias de los objetos de vistas que se han definido, 7 textboxes en este caso:

```
setContentView(R.layout.listadias);

lunes   = (TextView) findViewById(R.id.lunes); lunes.setOnLongClickListener(myLongClickHandler);
                                               lunes.setOnClickListener(myClickHandler);
martes  = (TextView) findViewById(R.id.martes);martes.setOnLongClickListener(myLongClickHandler);
                                               martes.setOnClickListener(myClickHandler);
miercoles = (TextView) findViewById(R.id.miercoles);miercoles.setOnLongClickListener(myLongClickHandler
                                               miercoles.setOnClickListener(myClickHandler);
jueves  = (TextView) findViewById(R.id.jueves);jueves.setOnLongClickListener(myLongClickHandler);
                                               jueves.setOnClickListener(myClickHandler);
viernes = (TextView) findViewById(R.id.viernes);viernes.setOnLongClickListener(myLongClickHandler);
                                               viernes.setOnClickListener(myClickHandler);
sabado  = (TextView) findViewById(R.id.sabado);sabado.setOnLongClickListener(myLongClickHandler);
                                               sabado.setOnClickListener(myClickHandler);
domingo = (TextView) findViewById(R.id.domingo);domingo.setOnLongClickListener(myLongClickHandler);
                                                domingo.setOnClickListener(myClickHandler);
```

Figura 3.39

En esta última imagen vemos cómo se enlazan XML e código, es decir, enlazar la clase con el XML. A continuación mostramos objetos dinámicos definidos desde el propio código. En Android, aunque disponemos de separar gráficos de código (igual que texto de código), podemos definir objetos gráficos desde el propio JAVA;

```
buttonkey = new ImageButton(this); //TODO BOTON TECLADO
buttonkey.setClickable(true);
buttonkey.setImageResource(R.drawable.keyboard);
buttonkey.setBackgroundResource(R.drawable.buttonbackground);
buttonkey.setAdjustViewBounds(true);
buttonkey.setScaleType(ScaleType.CENTER_INSIDE);
buttonkey.setMaxHeight(120);
buttonkey.setId(iadd);
buttonkey.setOnLongClickListener(new OnLongClickListener() {
```

Figura 3.40

Aquí creamos desde un botón desde el propio código, y luego he colgamos todos los parámetros que disponemos desde XML, por ejemplo, darle un fondo (R.drawable.keyboard), o darle un color de fondo (R.drawable.buttonbackground), entre otros.

Una vez tenemos definido este botón en memoria, tenemos que visualizarlo en pantalla. Es lo que muestra la siguiente figura:

```
linlay.addView(buttonB, linlayP);
linlay.addView(buttonkey, linlayP);
linlay.addView(buttonOn, linlayP);
```

Figura 3.41

Aquí añadimos dichas “vistas”, que no son más que objetos, a la vista actual de la pantalla, linlay. A parte, tenemos que pasarle un parámetro (linlayP), que define el ancho y alto del objeto.

Durante la implementación de la aplicación se han llevado a cabo todas las variantes relativas a layouts: con archivo XML, desde el código y combinando las dos.

En el software se han empleado distintos tipos de layout (realative,linear) para poder explorar las distintas posibilidades que ofrecen.

Como se definió en el apartado de carpetas del proyecto 3.2.3, en la figura 3.2.9 vienen definidos todos los layouts XML del software:

- Listadías: En este layout empleamos relativelayouts para ordenar los elementos dentro de la misma fila, es decir, los campos de texto y los togglebottones. También empleamos linearlayouts para dividir la pantalla entre 8 partes, de los cuales 7 son para los 7 reative layouts

correspondientes a los días, y la ultima parte es para los botones de confimación y volver:

Figura 3.42

- Alarma.xml: al ser esta la pantalla principal, se ha empleado más tipos que en el caso anterior. Entre ellos, LinearLayouts, RelativeLayouts, que ya hemos explicado lo que son. Otros son, EditText, que son campos editables por el usuario, por ejemplo hora e minuto. A continuación tenemos ScrollView, que es un objeto para agrupar otros objetos con la posibilidad de deslizar en pantalla. Cómo ultimo tenemos muchos botones, uno para salir de la aplicación, otro para añadir alarmas, etc.
- BorderEdittexDial.xml: este xml realmente no es una pantalla, sino que agrupa propiedades de objetos. Este en especial define los bordes de los botones, su radio, su grosor, color ,etc. Para ilustrarlo mejor, adjuntamos una imagen:

```
<item android:state_focused="true"
      android:state_pressed="false" >
    <shape xmlns:android="http://schemas.android.com/apk/res/android">
       <solid android:color="#ffffff"/>
       <corners
               android:radius="8dp"/>

       <stroke
       android:width="2dp"
       android:color="#000000" />
       <padding android:left="4dp" android:right="4dp" />
    </shape>
</item>
```

Figura 3.43

Gráficos en pantalla

Una vez se han definido los layouts de las distintas pantallas, el paso siguiente es dibujar las pantallas completas. Todos los recursos de imágenes están en la carpeta **res/drawable** y son objetos de tipo Drawable (una abstracción general para algo que se puede “pintar”)

Es importante darnos cuenta del siguiente hecho: las imágenes colocadas en la carpeta drawable no se emplea directamente en el código de las pantallas. Este hecho puede cambiar dependiendo de la aplicación. Este caso, la implementación sigue la siguiente pauta: cada imagen tiene dos variantes: una que tiene fondo negro, y texto blanco, y la otra los mismos colores pero en orden

invertido. La razón principal de ello es que cuando el usuario pincha sobre cualquiera de los objetos, dicho objeto cambia de colores para indicar el lugar donde se está clickeando.

Para lograr esto, en primer lugar se crea un xml que controla los estados con las dos imágenes:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_focused="true"
          android:state_pressed="false"
          android:drawable="@drawable/anadir_invertido2" />
    <item android:state_focused="true"
          android:state_pressed="true"
          android:drawable="@drawable/anadir_invertido2" />
    <item android:state_focused="false"
          android:state_pressed="true"
      android:drawable="@drawable/anadir_invertido2" />
    <item android:drawable="@drawable/anadir2" />
</selector>
```

Figura 3.44

En realidad aunque vemos 3, solo hay dos estados. Uno para el “presionado” (state_pressed=true) u otro para el “suelto” (state_pressed=false).

El siguiente paso es utilizar estos XMLes para injertarlo en código JAVA:

```
buttonOn.setImageResource(R.drawable.buttonanadir2);
```

Figura 3.45

Diálogos

Hasta ahora hemos visto el diseño de las dos principales pantallas. Sin embargo, una de las cosas importantes es controlar el momento de sonar una de las alarmas definidas. En ese momento se activa “trigger”, clase mencionada anteriormente, que recibe la señal de Android. En este momento trigger punta una ventana pequeña, que extiende de Dialog. Dialog es cómo una actividad pero no toma el control de toda la aplicación, de hecho deja el fondo en estado vivo. En la mayoría de los casos se utiliza para interactuar con el usuario, hacerle preguntas, darle opciones, etc. Este el nuestro caso, trigger avisa que la alarma está sonando, aparte de emitir un sonido para avisar al usuario.

En esta sección comentaremos el código del método “dialog” para entender cómo funciona:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage(getString(R.string.alarma_sonando))
        .setCancelable(false)
        .setNeutralButton(getString(R.string.detener),
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        sound.stopTone();
                        dialog.cancel();
                        finish();
                    }
                });
// AlertDialog alert = builder.create();

alert = builder.create();
```

Figura 4.46

En primer lugar le pasamos cómo parámetro el título que queremos que tenga, en este caso es R.string.alarma_sonando. En segundo lugar le decimos que no pueda ser cancelable, por supuesto sí que se podrá cancelar, pero no apretando en cualquier parte de la pantalla. A continuación le damos forma (.setNeutralButton) y le pasamos el texto que queremos que tenga el botón que le ponemos, en este caso R.string.detener.

Con la llamada builder.create() lo pintamos en pantalla.

Eventos de interfaz de usuario

La implementación de la interacción con el usuario ha querido seguir la línea de sencillez para facilitar su uso. Por ello, toda interacción de la aplicación sigue la siguiente línea: si el usuario hace un click corto sobre cualquier parte de la pantalla, la aplicación le lee en voz alta donde ha apretado. Si el usuario mantiene pulsado el botón durante más de 1 segundo, hace de función “entrar”.

Por eso tan solo hemos tenido que emplear 2 tipos de eventos, onClick y onLongClickEvent, ambos sobre todos los objetos empleados. EN la siguiente figura mostramos unos ejemplos:

He aquí un ejemplo de longclick:

```
button_fecha.setOnLongClickListener(new OnLongClickListener() {
                    public boolean onLongClick(View v) {

             //Intent to launch ListaDias.
                    Intent i = new Intent();
                    i.setClassName("com.upv.alarm",
"com.upv.alarm.ListaDias");
                    vibr.vibrate(NOTIFICATION_OR_WINDOW_STATE_CHANGED_PATTERN,
-1);
```

```
startActivityForResult(i,4);
return true;
}
```

Figura 4.47

Vemos que si el usuario presiona largo sobre el botón fecha, la aplicación lanza la pantalla listaDias con un intento (anteriormente explicado).

Sin embargo, si aprieta corto, lo que ocurre es lo siguiente:

```
button_fecha.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                              cargarPreferencias();
                              vibr.vibrate(patternMiddle2, -1);
                              mTts.speak(diasalarma,
TextToSpeech.QUEUE_FLUSH, null);
                        }
```

Figura 4.48

Aquí la aplicación tan sólo lee los días anteriormente definidos (mTts.speak(diasalarma…)).

Icono

El icono es la imagen que Android muestra en su lista de aplicaciones. Este icono [25] es un gráfico que representa la aplicación en la pantalla principal del dispositivo y en el "Launcher". El usuario abre el launcher presionando en casi todos los casos en el medio. En este caso no se ha querido crear un icono propio, sino que se decidió utilizar uno de google sin derechos de propiedad. El icono se muestra a continuación:

Figura 4.48

Strings y arrays

Para optimizar los recursos, es posible guardar valores de arrays y strings en los archivos xml contenidos en la carpeta **res/values**

Para poder tener acceso a los strings y arrays, se hace uso de la etiqueta name XXX, de modo que se toma y usa e valor YYY.

```
<resources>

    <string name="app_name">Alarma</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">Despertador</string>

</resources>
```

Figura 3.49 – Strings

4. CONCLUSIONES

Finalizado el proyecto, se analiza el resultado para comprobar si se han cumplido con las expectativas. Se puede afirmar que los objetivos propuestos se han logrado.

4.1 Implementación de una aplicación

El primero de los objetivos, implementar una aplicación bajo la plataforma Android, se ha completado satisfactoriamente. Se ha creado una aplicación que funciona según los requisitos planeados en un Sony Xperia Tipo.

4.2 Posibilidades de Android

El segundo de los objetivos buscaba explorar las opciones que ofrecía Android. También se ha cumplido con este objetivo, ya que se han implementado distintas funcionalidades para aprender su uso y aplicación:

- **Botones**: se ha aprendido a incluirlos en la pantalla, capturar la pulsación, realizar alguna acción una vez pulsados y optimizarlos visualmente mediante el uso de imágenes
- **Cuadros de diálogo:** se han utilizado para mostrar tanto mensajes informativos o de error, como listas para hacer selecciones.
- **Activities**: se ha aprendido que cada pantalla a mostrar requiere de una Activity, y cómo se ha de hacer la gestión entere las distintas Avtivities dependiendo del resultado que se quiera obtener.
- **Imágenes**: se ha ahondado en la forma de gestionar el uso de imágenes, instanciándolas, modificándolas y refrescando sus vistas.
- **Sonidos**: se han incluido sonidos en bucle, para que se oigan de forma permanente en una pantalla determinada
- **BroadCastReceivers**: Se ha utilizado para recibir las señales q vienen de fuera de la aplicación, mensajes de parte de Dalvik VM.

Gracias a que Android se basa su sistema operativo en código libre, un gran número de programadores desarrollan aplicaciones de manera que es más

sencillo encontrar documentación y ejemplos de código. También esto hace que a la hora de gestionar los problemas o errores, la búsqueda de soluciones es más rápida que en sistemas operativos con código cerrado o propietario.

4.3 Futuros desarrollos

En el caso de que futuros estudiantes deseen implementar una aplicación o juego en la misma plataforma, este proyecto les sirve de base o guía para dar esos primer pasos.

También se ha cumplido el objetivo del que el software sirviese como base a futuros desarrolladores de Android. La aplicación implementada cumple unas funcionalidades básicas que lo hace utilizable, sim embargo da pie a que se realicen ampliaciones y mejores en el mismo. Esos puntos se tratan en el capítulo de Líneas Futuras 5.

4.4 Otras conclusiones

Adicionalmente a los objetivos que se habían marcado, dado que en Android se usan muchos recursos con formato **XML**, se ha podido profundizar en este lenguaje y en las capacidades y funcionalidades que ofrece.

También se han revisado las posibilidades que ofrece el lenguaje **UML** para la fase de diseño del proyecto. Gracias al mismo se pueden hacer un diseño y exhaustivo de clases, flujos, etc, que hacen la fase de implementación más eficiente y sencilla.

A nivel personal, es muy satisfactorio el poder desarrollar por primera vez una aplicación y en una plataforma en la que no se tenía ninguna base. Una vez conocida la plataforma, y gracias a los conocimientos adquiridos, se espera seguir desarrollando aplicaciones y juegos, ya que resulta muy gratificante una vez se llega al resultado final.

5. *LINEAS FUTURAS*

Puesto que había una limitación en tiempo para la realización del proyecto, no ha sido posible implementar todas las funcionalidades que se hubiera deseado. Es por ello que en este capítulo se muestran algunas de esas posibles mejores para realizar en un futuro.

Con estas mejores, se conseguiría una aplicación más completo y con más funcionalidades que permitiría al usuario disfrutar más del mismo

5.1 Elección de día

Una de las mejores podría ser la posibilidad de elegir un día concreto del mes o del ano. Aunque originalmente no hubo ni opción de elegir día de semana, una notable mejora fue la posibilidad de la misma. Sin embargo, sin poder elegir un día concreto, este despertador no puede competir con otros que sí tiene esta opción.

5.2 Cuadro de diálogo

Otra mejora podría ser el cuadro de diálogo que salta al sonar un despertador. Es el único elemento que no se podido adaptar a las necesidades. La razón principal, es que a los cuadros de diálogo no se les puede cambiar la imagen de fondo. En su lugar se tendría que emplear una actividad pequeña, con una propiedad llamada “transparent” que significa transparente. Así la ventana dejaría ver lo que tiene detrás, camuflándose de cuadro de dialogo. El problema es que una actividad debe pausar necesariamente otras actividades, es decir solo puede haber una actividad activa.

5.3 Otras mejores

- **Inteligencia Artificial**: La aplicación podría entender las palabras del usuario, siendo así adaptado para gente con ceguera total, y no solo discapacitados. El problema es que implementar ese sistema requiere

todo un estudio y quizás un master. Por ahora se podría trabajar con la librería de Android para interpretar las palabras, pero cómo dicha biblioteca funciona a través de internet, por ahora se ha dejado fuera del diseño.

- **Zoom**: la aplicación podría tener una función “zoom” para acercar parte de pantalla a los ojos del usuario.

ANEXO

Planificación

Antes de comenzar el proyecto, es necesario realizar una planifcación que permita dividirlo en distintas fases y asociar a cada una de ellas tanto recursos materiales como personales, costes y plazos estimados. Gracias a esta planifcación en etapas del proyecto, se consigue hacer un seguimiento del mismo y así comprobar si se están cumpliendo los fechas comprometidas en un primer momento.

En el anexo, se muestran el ciclo de vida de la realización de una aplicación , el ciclo de vida de Alarma y un diagrama de planifcación inicial.

A.1. Ciclo de vida de un juego comercial

En este apartado se definen las distintas etapas que componen el ciclo de vida de un juego comercial. Estas etapas no se han seguido exactamente en el proyecto, pero sirven de ayuda y guía para la planifcación de cualquier juego.

- **Concepto**: durante esta primera fase, se define el concepto del juego a partir de una idea de origen o un "brainstorming", creándose la propuesta del juego y el arte conceptual. Esto servirá como una primera base que define la historia del juego. Además se incluyen en esta fase, la definición de diferentes características del juego como ambientación, género, plataformas de desarrollo, cronograma estimado presupuesto y análisis de riesgos. Diseñadores, analistas y artistas conceptuales, participan en esta fase.

- **Pre-producción**: en la fase de pre-producción se estudia si es viable realizar el software, definiendo un estudio completo con la mecánica del mismo, niveles, pantallas, personajes, etc. Se ha de tener también un calendario aproximado para la realización de las tareas del proyecto.

- **Producción**: es la parte más importante del proyecto ya que es cuando se construye e implementa la aplicación. Cuando se llega a un nivel sufciente para poder probar, los testeadores se encargan junto con desarrolladores de corregir errores, realizar avances y añadir nuevas funcionalidades. Consta de varias fases como son el diseño, la programación, la creación de niveles, el diseño artístico, la producción de audio o el testeo.

- **Alfa**: durante la fase alfa, el software ya es operativo por completo (motor, interfaz

de usuario, etc), pero todavía pueden quedar detalles o errores por depurar.

- **Beta**: fase cuya finalidad es dejar el software libre de errores y fallos. El incremento de características se ha finalizado, y solo quedan por corregir los errores.

- **Congelación del código**: se congela la versión de la aplicación de forma que ya no se apliquen más cambios. En este punto queda pendiente de la aprobación.

- **Liberación**: una vez se ha aprobado, se libera a los medios de distribución para acercarlo al público.

- **Parches**: cuando los usuarios encuentran fallos o errores en el software que deben ser reparados, se lanzan parches para solucionar el problema.

- **Actualizaciones**: pueden ir desde pequeños detalles, hasta niveles o mundos completamente nuevos e innovadores.

A.2. Planificación

Antes de planificar, es necesario elegir un modelo de ciclo de vida que se seguirá durante el proyecto. Después de evaluar las distintas posibilidades, se ha optado por el modelo de desarrollo evolutivo (Figura A.1).

El modelo evolutivo tiene como objetivo principal construir un prototipo de una forma estructurada, de manera que se pueda ir refinando y reconstruyendo. Esto permite al equipo de desarrollo añadir características o hacer cambios, que no se concibieron durante la fase de requerimientos y diseño.

Gracias a este modelo, los desarrolladores se pueden centrar en una parte del sistema que comprendan, en lugar de tener que abarcar el sistema completo.

Figura A.1

Con este modelo es posible el centrarse en módulos, de forma que la aplicación sea operativo y jugable desde el principio, y se vayan añadiendo distintas funcionalidades y características.

El proyecto consta de varios prototipos o componentes, y en cada uno de ellos se ha aplicado el modelo evolutivo. Como se muestra en la Figura A.1, al comienzo del desarrollo se recogen y refinan los requerimientos, se realiza un diseño rápido para construir el prototipo, se evalúa y se refina.

Los componentes son:

- **Formación previa**: estudio de qué elementos serán necesarios para la realizaci ón del proyecto. Para el desarrollo del software, es necesario tener conocimientos de Android, por lo que esta fase no aplica en caso de haber implementado con anterioridad alguna aplicación con Android.

- **Desarrollo en la documentación y memoria:** a medida que se van realizando o modificando prototipos, se recoge su documentación asociada (análisis, diseño, implementación, evaluación y problemas encontrados), para ir componiendo así este documento fonal.

- **Estructura básica del software**: proporcionará la funcionalidad básica para que el software arranque y sirva de base para el resto de componentes. Incluye la navegación entre pantallas, eventos simples (botones) e interrupción y salida de la aplicación.

- **Interfaz de usuario**: en este apartado se define el diseño gráfico de pantallas, los cuadros de diálogo, los menús y los iconos.

- **Núcleo del software**: en este prototipo se diseña e implementa todo lo relacionado con la gestión del tablero, la detección de las posiciones más cercanas y los movimientos del ladrón.

- **Otros**: para esta parte del desarrollo, se tienen en cuenta los sonidos y la gestión del texto.

- **Evaluación y pruebas generales**: una vez se tienen completos todos los prototipos, en el de evaluación y pruebas generales se comprueba el correcto funcionamiento del aplicación.

BIBLIOGRAFÍA

[1] http://www.idc.com/getdoc.jsp?containerId=prUS24085413

[2] http://www.gartner.com/newsroom/id/2482816

[3]http://www.who.int/features/factfiles/blindness/blindness_facts/es/index.html

[5] B. Reed. (2010, Jun.) A brief history of smartphones. [Online]. Available: http://www.networkworld.com/slideshows/2010/061510-smartphone-history.html#slide1

[6] E. Pacheco. (2010) T-mobile g1, el primer terminal android llega al fin de su vida comercial. [Online]. Available: http://www.xataka.com/moviles/tmobile-g1

[7] symbian.nokia.com. (2011) Symbian at nokia. [Online]. Available: http://symbian.nokia.com/

[8] L. Foundation. (2011) Bienvenido a limo. [Online]. Available: http://www.limofoundation.org/es/bienvenido-a-limo.html

[11] http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html

[12] O. H. Alliance. (2011) Open handset alliance. [Online]. Available: http://www.openhandsetalliance.com/

[13] Android.com. (2011) What is android? [Online]. Available: http://developer.android.com/guide/basics/what-is-android.html

[14] ___. (2011) Application fundamentals. [Online]. Available: http://developer.android.com/guide/topics/fundamentals.html

[18] C. S. . S. E. S. Committee, IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. Luxembourg: IEEE Computer Society, 2000.

[19] S. Muthu. (2010) The structure of an android project. [Online]. Available: http://sudarmuthu.com/blog/the-structure-of-an-android-project

[20] Android.com. (2011) Technical resources - tutorials. [Online]. Available: http://developer.android.com/resources/browser.html?tag=tutorial

[21] ___. (2011) User interface - declaring layouts. [Online]. Available: http://developer.android.com/guide/topics/ui/declaring-layout.html

[22] ___. (2011) User interface - creating dialogs. [Online]. Available: http://developer.android.com/guide/topics/ui/dialogs.html

[23] ___. (2011) User interface - handling ui events. [Online]. Available: http://developer.android.com/guide/topics/ui/ui-events.html

[25] ___. (2011) Launcher icons. [Online]. Available: http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html