UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

Diseño y desarrollo de aplicación de seguridad para el sistema operativo Android

TITULACIÓN: Telemática
TUTOR/ES: Álvaro Suárez Sarmiento
Elsa María Macías López

AUTOR: David Airam Hernández Rodríguez
FECHA: julio del 2014

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

Diseño y desarrollo de aplicación de seguridad para el sistema operativo Android

Presidente: Secretario: Vocal:

Tutores: Autor:

NOTA: ……

TITULACIÓN: Telemática
TUTOR/ES: Álvaro Suárez Sarmiento
Elsa María Macías López

AUTOR: David Airam Hernández Rodríguez
FECHA: julio del 2014

¿Por qué esta magnífica tecnología científica, que ahorra tanto trabajo y nos hace la vida más fácil, nos aporta tan poca felicidad?

La respuesta es simple, porque aún no hemos aprendido a usarla con inteligencia.

Albert Einstein (1879-1955)

Índice

1. Introducción

En este primer capítulo se presenta el alcance y ámbito del trabajo realizado así como los objetivos del *Proyecto Final de Carrera* (*PFC*) y la estructura de esta memoria.

Comenzaremos analizando brevemente la situación actual de los teléfonos móviles, más concretamente la de los *smartphones* Android, para continuar posteriormente centrándonos en los sistemas de seguridad actuales de estos dispositivos. Finalmente abordamos los objetivos de este PFC y la estructura de la memoria completa.

1.1 Los teléfonos móviles

Vivimos en una época en la que los pequeños aparatos electrónicos han ido cobrando cada vez más protagonismo y actualmente estamos viviendo un completo *boom* de *gadgets* tecnológicos que usamos de forma continua en nuestro día a día. Todos estos pequeños aparatos electrónicos o *gadgets* que nos acompañan en el día a día cobran cada vez una mayor importancia y sin quererlo hemos creado un vínculo de dependencia con éstos, ya sea para bien o para mal. En definitiva estos teléfonos móviles hacen la vida un poco más fácil permitiendo a la vez que el ciudadano del siglo XXI siga conectado mientras está en movilidad.

Sin duda alguna, de todos estos nuevos aparatos electrónicos el más importante sigue siendo el s*martphone.* Se considera que dichos aparatos terminaron de reestructurarse y tomar la forma que conocemos hoy en día tras la irrupción en el mercado del primer *iPhone.* Tras la aparición de éste en el año 2007 supuso un cambio radical en el mundo de los teléfonos inteligentes ya que cambió casi por completo el paradigma que conocíamos hasta entonces. Antes del nacimiento del *iPhone* uno de los *smartphone* más avanzado del momento era el Nokia N95, y aunque incluso era superior en algunos aspectos al *iPhone*; el hecho de usar un software que era significativamente inferior debido principalmente a una interfaz gráfica clásica, de menor expresividad y menos *user-friendy* terminaron por decantar la balanza hacia la filosofía propuesta por Apple. En la Figura 1.1 se representa este cambio radical.

Hasta llegar al Nokia N95 hubo otros teléfonos con nombre propio como *Blackberry*, *Motorola* y *Ericsson* que pasaron a un segundo plano. Otro hito importante es la aparición del software *Android* con el cual se produjeron cambios importantes y significativos: muchos teléfonos móviles lo adoptaron lo que facilitó el diseño de nuevos *smartphones* con pantallas táctiles muy potentes. Gracias al empuje de *Google* ha conseguido convertirse en el sistema

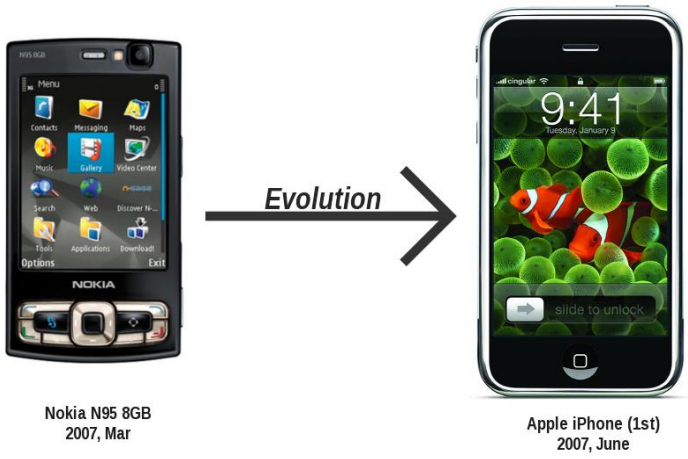


Figura 1.1. La evolución del N95 de Nokia frente al primer

operativo más popular para *smartphones* y consolidarse como el principal rival de *iOS.* Si bien es cierto que no son los dos únicos sistemas operativos para smartphones, es cierto que el resto de sistemas operativos no tiene tanta implantación como éstos.

Por tanto, es interesante centrar nuestro PFC en el sistema operativo Android y en los *smartphones* modernos con capacidades de cálculo y comunicación muy poderosas. En estos teléfonos móviles un problema importante es el de la privacidad de los datos personales que contienen ya que cada vez estos son mayores y más sensibles y casi sin darnos cuenta depositamos en estos terminales total confianza y seguridad.

1.2 Seguridad de los teléfonos móviles

Los teléfonos móviles hoy día almacenan gran cantidad de información personal. Normalmente dicha información es privada y perderla supone un grave problema para el usuario del teléfono móvil. Por ello, la seguridad en los equipos móviles, en cuanto a robo se refiere, es un problema muy importante.

Aunque en la actualidad existen algunas soluciones disponibles para evitar la pérdida de los teléfonos móviles, sería deseable crear una solución de seguridad integral para teléfonos móviles lo más transparente y fácil de usar posible haciendo hincapié en el uso de servicios gratuitos para lograr los objetivos deseados.

Además, debería exhibir facilidad de uso o que éste sea homogéneo en toda la aplicación. Todo esto se podría conseguir de la siguiente manera:

- Definir una palabra clave de activación del sistema así como otros datos útiles como email de respuesta por defecto...

- La activación del sistema sería siempre vía *Short Message Service* (*SMS*) debido a que su uso es sencillo, es de bajo coste y de alta disponibilidad del servicio, ya que incluso con una tecnología lenta como *General Packet Radio Service* (GPRS) o *Global System for Mobile communications* (*GSM*) es capaz de funcionar de forma rápida.

- Para activar el sistema antirrobo simplemente se debe enviar un SMS con la palabra clave definida al principio de éste.

Las principales características que debería implantar un sistema de estas características son:

- Arranque y ejecución transparente: Ejecución en segundo plano de forma transparente través de un servicio de sistema sin que perjudique esto el rendimiento del teléfono móvil.

- Implantación de diferentes modos de alerta y actuación.

- Sistema de auto-activación al apagarse y/o cambiar la *Subscriber Identity Module* (*SIM*).

- Diferentes formas de activación siendo la principal vía SMS.

- Facilidad de uso a través de una Graphics User Interface (GUI) atractiva e intuitiva.

- Ser eficiente con los recursos del teléfono móvil.

En este PFC se pretende estudiar la seguridad en los teléfonos móviles Android y presentar una solución de desarrollo de software que a su vez sirva como una base en la que se puedan apoyar futuros proyectos. No se pretende superar a otras herramientas existentes, sino exponer una visión particular simple y útil para resolver este problema.

1.3 Objetivo

El objetivo principal del proyecto es poder proteger un teléfono móvil Android a través de una aplicación sencilla, potente e intuitiva así como fácil de configurar y de usar; la cual no dependa de complejos sistemas de registro en un servicio web para poder después rastrear tu teléfono accediendo a servidores de terceros a través de su página web.

La idea es que una vez percatemos de que han robado o hemos perdido nuestro teléfono podamos simplemente enviarnos a través del teléfono del amigo o familiar que tengamos al lado un SMS con nuestra contraseña. Si no tenemos nadie al lado siempre podríamos hacerlo a través de un teléfono fijo o a través de la web mediante páginas de servicios de SMS.

1.4 Estructura de la memoria

La memoria se ha estructurado de forma progresiva ordenada y limpia. Empieza dando una introducción al desarrollo de la telefonía móvil y a la seguridad en los *smartphones* para poco a poco abordar el problema más de lleno, ver las alternativas que hay en el mercado y presentar nuestra solución y explicar el porqué de ésta. Estudiamos las diferentes herramientas de desarrollo empleadas para posteriormente hacer un completo análisis funcional y orgánico de la aplicación implementada sin olvidarnos de analizar toda la interfaz de usuario, hacer una batería de pruebas y finalmente sacar conclusiones y posibles ampliaciones de este proyecto final de carrera. La estructura de la memoria de este PFC es la siguiente:

- En el primer capítulo hacemos una introducción al problema de la seguridad en los *smartphones*.
- En el segundo capítulo profundizamos en el problema y presentamos alternativas y planteamos por primera vez nuestra solución de forma concisa.
- En el tercer capítulo analizamos las herramientas de desarrollo usadas para implantar este Proyecto Final de Carrera (PFC).
- En el cuarto capítulo hacemos un análisis funcional y orgánico del código fuente de la aplicación.
- En el quinto capítulo analizamos en detalle la interfaz de usuario y realizamos varias pruebas al sistema implementado.
- En el sexto sacamos las conclusiones y estudiamos el futuro del PFC.
- Finalmente elaboramos un presupuesto de implementación del PFC con su estudio previo y pliego de condiciones.
- El Anexo I da un apoyo a la memoria ya que explica partes fundamentales del desarrollo Android centradas en este proyecto en particular.

2. Sistemas antirrobo en teléfonos móviles

Los *smartphones* se han convertido en un elemento prácticamente indispensable en la vida de las personas. Con ellos a lo largo del día realizamos una gran variedad de acciones. Estas nuevas operaciones han complementado y probablemente trasladado toda la importancia a la principal acción de llamar del antecesor del *smartphone*.

Las diferentes operaciones que podemos realizar hoy en día con nuestros smartphones van desde la mensajería instantánea hasta comprobar el email pasando por otras tan variadas como: visualizar documentos, navegar por Internet, comprobar el tiempo atmosférico, usar la agenda, usar la cámara de fotos y vídeo, jugar, llevar la contabilidad... Y esto es sólo una muestra de lo que permite hacer hoy en día nuestro *smartphone,* ya que cada día el ecosistema de aplicaciones y posibilidades crece tanto por parte de fabricantes como de desarrolladores dotando a estos teléfonos móvil cada vez de más y más opciones.

En este capítulo mostramos los problemas de la pérdida de la información, las soluciones comerciales existentes y presentamos las ideas generales del proyecto a implantar.

2.1 Estado actual Social de los teléfonos móviles

Actualmente en nuestros *smartphones* depositamos muchísima información importante de nuestra vida diaria, tanto de nuestra vida profesional (correos, contactos, datos de login, documentos…) como personal (fotografías, vídeos, mensajes, aplicaciones...) con lo que se ha convertido en un aparato que alberga información privada tanto personal como profesional de mucha relevancia. Es natural pensar que una persona desea conservar a toda costa tanto su información privada como por extensión su teléfono móvil, tanto por su valor económico como por su valor en contenido digital sensible.

En la Tabla 1.1 se muestra el tipo de información privada (*datos sensibles*), que suele superar los 5,5 GB, que una persona suele almacenar en su teléfono móvil.

Elementos	Tamaño
Fotografías e imágenes	+2,00 GB
Vídeos	~2.00 GB
Mensajería (Whatsapp, Telegram, SMS...)	~1.20 GB
Documentos, emails, otros	+0.30 GB
TOTAL	**+5,50 GB**

Tabla 1.1. Media de datos sensibles por categorías

Además de la cantidad media de datos sensibles cuando estamos evaluando la posible pérdida de un teléfono móvil, también deberíamos tener en cuenta el coste medio del teléfono móvil. El precio medio de los teléfonos inteligentes ha descendido un 8% respecto al ejercicio anterior (2013), pero éste se sigue situando cercano a los 300€ con un valor exacto de 278,20€ para el año 2014. Estos dos últimos datos reflejan la importancia y el valor que adquiere el smartphone, situándose según las encuestas como el objeto personal que más miedo daría perder a los usuarios.

Otro punto muy importante, además del coste del teléfono móvil, es el sistema operativo con el que cuentan los *smartphones* ya que junto con las aplicaciones de éste forman

el principal ecosistema con el que cuenta el usuario para trabajar diariamente. Y es precisamente en este marco en el que *Android* tiene especial importancia ya que hoy por hoy es el número uno en cantidad de aplicaciones disponibles, en variedad de teléfonos móviles y el número uno en ventas de Mercado tal como se muestra en la Figura 2.1.

Además de todo lo anterior, Android cuenta con el *Software Development Kit* (*SDK*) más avanzado de todos los disponibles, brindando a los desarrolladores casi una total libertad a la hora de implantar sus aplicaciones. Por lo que puestos a elegir una plataforma y un sistema operativo para desarrollar la primera versión de este PFC sin duda Android se plantea como la mejor alternativa así como con la que más usuarios y objetivos conseguimos cubrir.

En definitiva, por todo lo comentado anteriormente cada vez es más importante y necesario proteger de una manera fiable y segura tanto nuestro *smartphone* como su contenido y es precisamente esta la razón de ser de este proyecto, crear una aplicación de seguridad para teléfonos móviles Android.

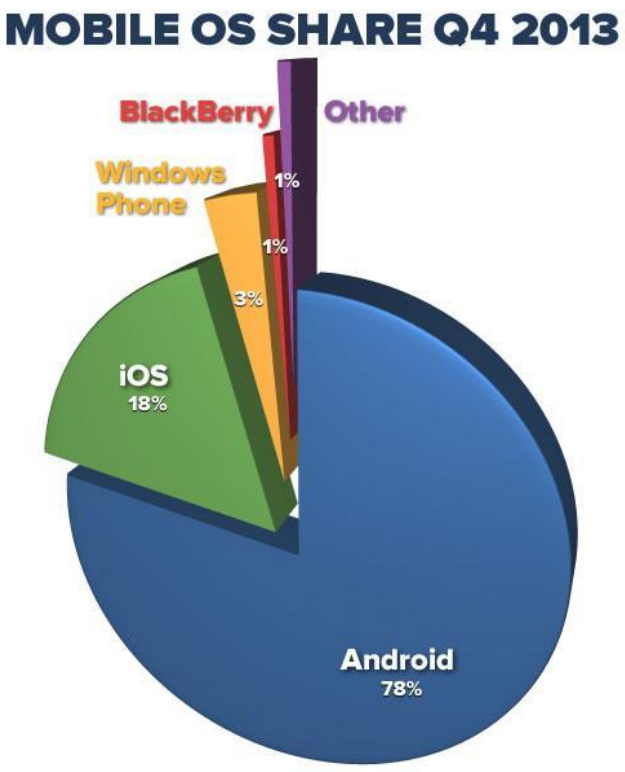


Figura 2.1. Mercado de *smartphones* del último cuatrimestre del 2013.

2.2 Otras aplicaciones antirrobo

En el año 2009 en el momento de redactar el anteproyecto realmente no existía ninguna alternativa real a lo que se planteó en anteproyecto pero hoy en día, en el año 2014, existen varias alternativas o soluciones similares profesionales a lo que se ha planteado en este PFC. A continuación revisamos algunas soluciones destacando algunas de sus principales características.

Google Android Device Manager

Esta sería la solución nativa para teléfonos móviles Android. Requiere que activemos el *Administrador de Dispositivos* como *Administrador de Sistema* a través de *Ajustes* → *Seguridad* → *Administrador de Dispositivos* (Figura 2.2).

Una vez activado, el sistema permite [11] analizar la ubicación online del teléfono móvil, hacerlo sonar, bloquearlo o borrarlo completamente de forma remota.

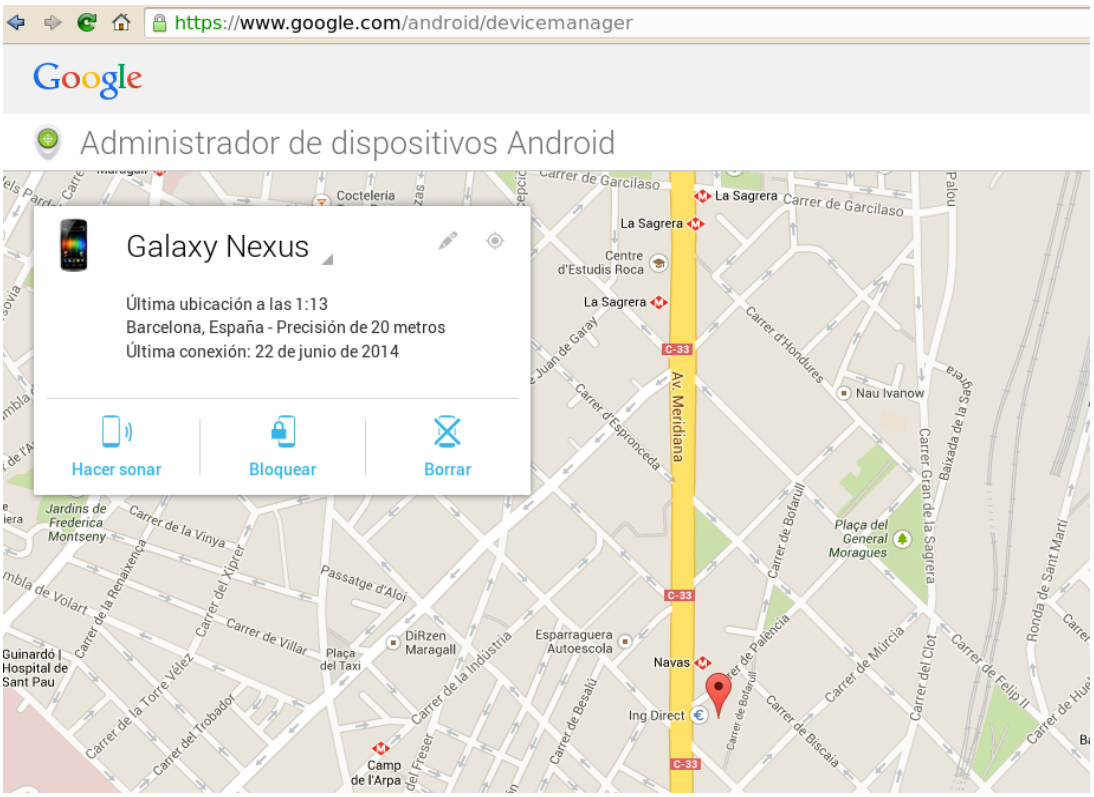


Figura 2.2. Google Android Device Manager

Es una buena alternativa para lograr encontrar el teléfono móvil y/o bloquearlo de forma remota en general como sistema integral y completo antirrobo; pero tiene algunos inconvenientes como por ejemplo la gran limitada cantidad de opciones o el hecho de requerir conectarnos a través de una web para localizar o hacerlo sonar el teléfono móvil de forma inmediata.

En la *Google PlayStore* hoy en día existen bastantes aplicaciones de este tipo pero quizá las dos más importantes son las que mostramos a continuación.

Sophos Antivirus and Security

Las aplicaciones de la serie *Sophos* son un conjunto de aplicaciones conjunta de Antivirus que además cuentan con opciones de seguridad similares a las que estudia y se han desarrollado en este proyecto. Destaca también por su sencillez, y fácil configuración, siendo una buena alternativa (Figura 2.3).

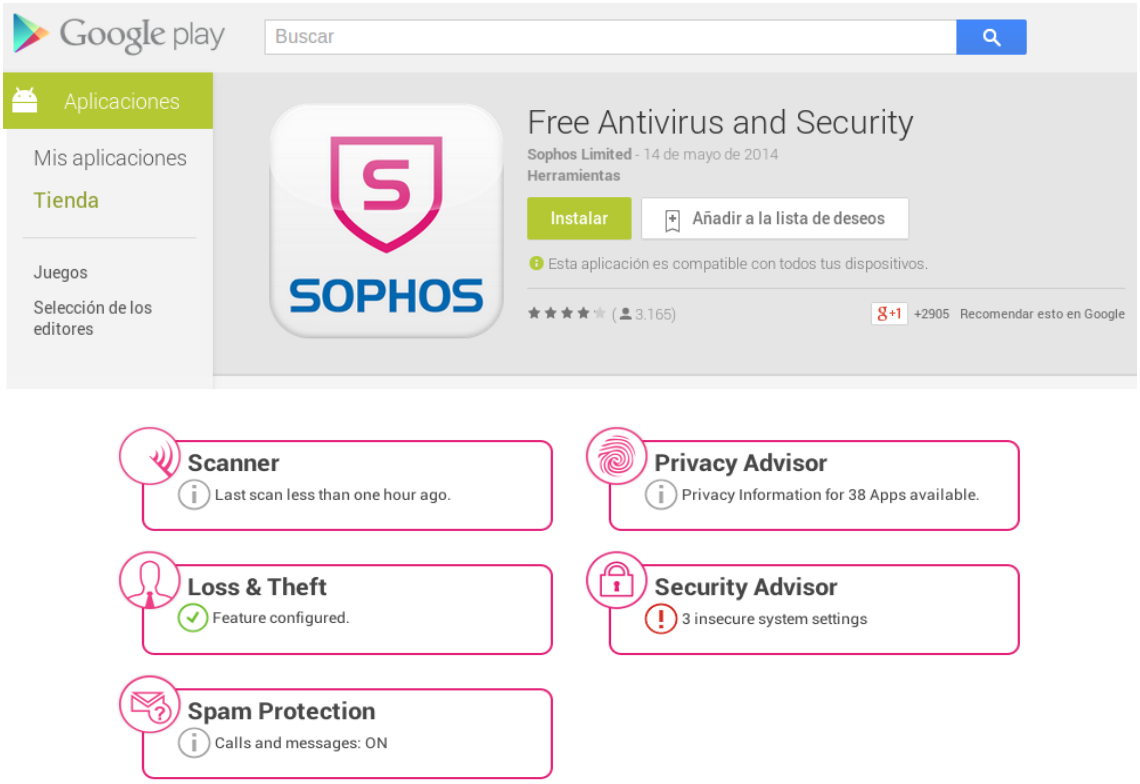


Figura 2.3. Sophos Antivirus and Security

Cerberus & Cerberus SMS

Es la aplicación que más coincide con los objetivos de nuestro PFC: permite tanto controlar el teléfono móvil desde su página web (servidor de cerberus) de forma remota, como vía SMS y además cuenta con muchas de las funciones de las que dispone nuestro PFC, es por eso que es probablemente la mejor referencia comercial de nuestro PFC (Figura 2.4).

Cerberus es una aplicación profesional, que cuenta con más de 2.000.000 de descargas y su licencia cuesta 2,99€ (permitiendo gestionar hasta 5 teléfonos móvil). En resumen, Cerberus es la prueba de que la idea inicial del proyecto en 2009 era un rotundo éxito.

En cualquier caso Cerberus no es perfecto con lo que sigue habiendo bastante margen de mejora.

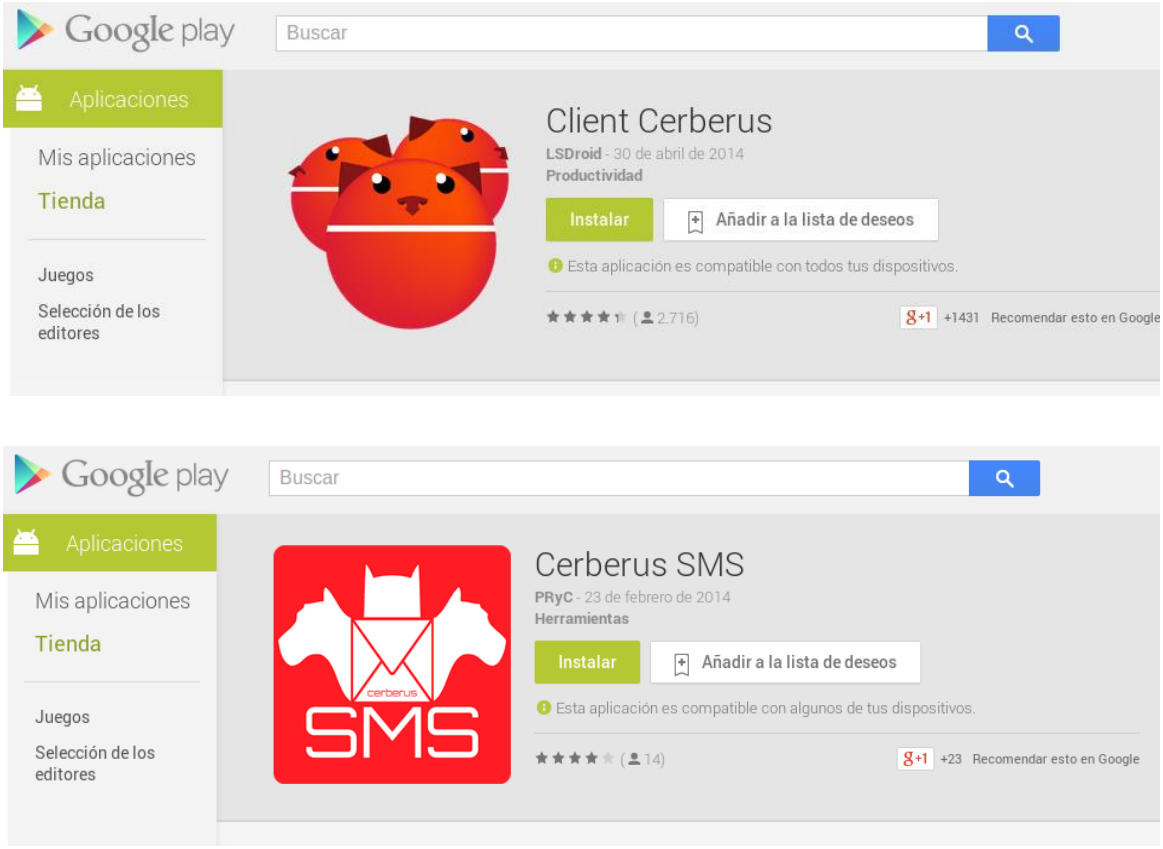


Figura 2.4. Client Cerberus and Cerberus SMS

2.3 Ideas generales de la aplicación a implementar

La aplicación móvil que se ha planteado como solución al robo o pérdida del teléfono móvil pretende ser una base didáctica de desarrollo para estudiar y mejorar el concepto de la seguridad en teléfonos móviles hasta el máximo punto posible que permita la tecnología Android y el ingenio, sabiendo además que las soluciones comerciales existentes no son definitivas.

Aprovechando estas características, se ha intentado que la solución planteada sea lo más profesional posible y de uso elegante.

Se pretende que la aplicación sea una primera versión de un sistema de seguridad integral que incluya las siguientes características: control remoto, opciones multimedia, notificaciones por varios canales, alta disponibilidad… Además de esto debe de ser:

- *Fácil de usar***:** diseñando un menú fácil de entender por el usuario inexperto, no exigiendo un sistema de registro en ningún servidor o web: configurando únicamente parámetros básicos de la aplicación localmente.

- *Fácil activación con alta disponibilidad:* usando la activación, en caso de robo o pérdida, del envío el de un mensaje de texto (SMS). Aprovechando que el SMS funciona incluso cuando la cobertura es muy baja y no es estable lo que aporta una alta disponibilidad. Además se puede recibir el mensaje de forma totalmente transparente sin levantar sospechas en el usuario infractor (en caso de robo).

- *Diferentes modos de funcionamiento:* debería funcionar de diferentes formas, por ejemplo: dando una alarma, geolocalizando al teléfono móvil, capturando audio e imagen y controlando el arranque no autorizado.

- *Funcionamiento sigiloso:* el usuario infractor no debe notar que el teléfono móvil está siendo monitorizado remotamente, esto es, los servicios de la aplicación antirrobo deben funcionar sigilosamente para evitar que el usuario infractor reacciones y evite su funcionamiento. Por ejemplo, se debe ocultar la activación vía SMS, el auto-arranque, la captura de audio y de fotografía… que analizamos en profundidad en capítulos posteriores.

- *Modular:* el control de la privacidad se implanta modularmente: cada elemento de privacidad tiene su correspondiente módulo. Cada módulo debe ser lo más independiente posible para facilitar cambios o futuras actualizaciones.

- *Sistema de bloqueo***:** se debe implantar en varios niveles: a) proteger todo el teléfono móvil con una contraseña y b) filtrar los usuarios que pueden enviar un mensaje activador tanto mediante un sistema de *whitelist* como de *blacklist.*

Cumpliendo con las ideas anteriores se lograría una solución buena, fácil de usar y funcional. Ya que evita largos procesos de registro y se activa fácilmente al recibir un *SMS activador* de una persona determinada.

3. Herramientas de desarrollo

Para el desarrollo de la aplicación móvil hemos tenido que estudiar las diferentes herramientas y posibilidades que teníamos disponibles para una implementación lo más eficiente posible de este PFC. En este capítulo presentamos las herramientas estudiadas y usadas para la implantación de la aplicación.

3.1 Análisis de alternativas de herramientas

La parte positiva a la hora de desarrollar este PFC es que el sistema operativo y la plataforma de desarrollo está totalmente definida, Android. Con lo que esto acota bastante las posibles herramientas y entornos de desarrollo que podemos usar con garantías de éxito.

Aunque si bien es cierto que podríamos desarrollar con cualquier entorno de desarrollo Java e incluso podríamos desarrollar en un editor de texto plano es obvio que tales alternativas quedan descartadas por las siguientes razones:

- No disponer de un *plugin* específico de Android para el entorno.
- Tener que exportar las bibliotecas de Android nativas a mano en un proyecto Java, con todos los problemas de dependencias que esto generaría.
- No generar automáticamente el archivo *manifest* (Anexo I.2)) que además es la columna vertebral de un proyecto Android.
- No tener acceso a gestores de XML de Android.
- Tener que generar manualmente la clase de *Resources*.
- Evitar el complejo proceso de *Build* de una aplicación Android manualmente (Figura 3.1).

En definitiva programar directamente con un entorno de desarrollo no adaptado para Android generaría infinidad de problemas. En la Figura 3.1 se muestra el esquema del proceso de generación de un *Application PacKage File* (*APK*) para Android con el que nos tendríamos que enfrentar en tal caso.

Desestimados los entornos de desarrollo que no cumplen con las características comentadas anteriormente contemplamos dos opciones claras: Eclipse [11] y Android Studio [14].

Entre las ventajas de Eclipse están:

- Es un estándar de desarrollo recomendado por Google para aplicaciones Android.
- Rápido y estable.
- Lleva años en el mercado para lenguajes como Java y C++.
- La instalación del *plugin* Android Developer Tools (ADT) ya no supone problema alguno.

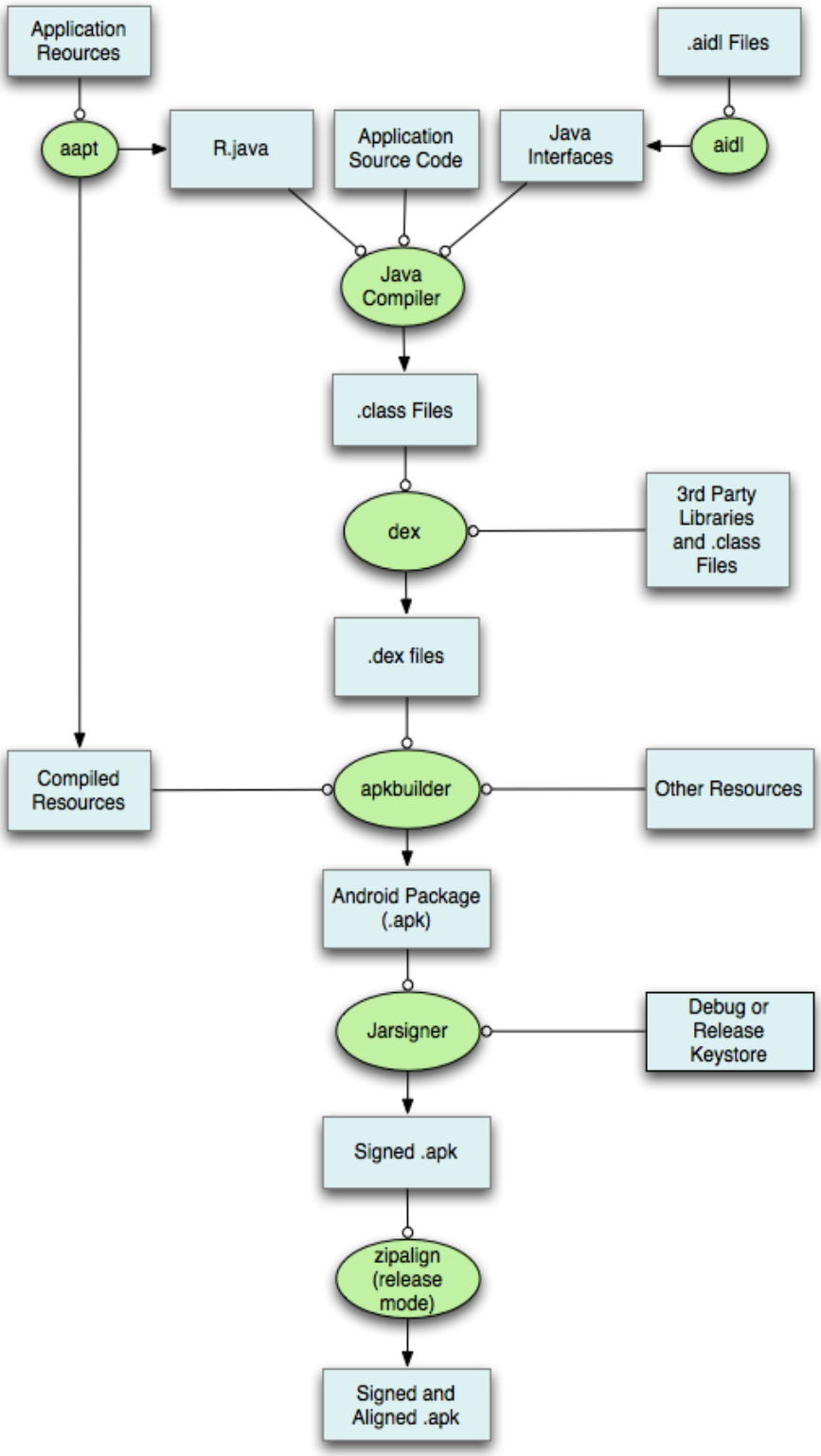


Figura 3.1. Esquema de diseño de un APK.

- Versión de Google totalmente preparada y funcional.
- Gestión de bibliotecas de forma tradicional o a través de repositorios Maven.

Entre las desventajas de Eclipse están:

- Editor de XMLs algo anticuado.
- No tiene la potencia de Android Studio para generar diferentes entornos de trabajo.
- No cuenta con un potente sistema de *refactoring.*
- Gestión de bibliotecas así como sistema de *Build* tradicionales.

Entre las ventajas de Android Studio están:

- Basado en intelliJ Idea (rival directo de Eclipse y gran alternativa).
- Google confirma que será el sucesor de Eclipse.
- Potencia bruta.
- Mejor conexión con el *Android Debug Bridge* (ADB) que Eclipse.
- Totalmente adaptado y diseñado para proyectos Android.
- Nuevo sistema de gestión de dependencias basado en Gradle.
- Mejor sistema de refactoring.
- Mejor editor de Layouts XML.
- Mejor gestión de las bibliotecas *Google Cloud Messaging* (GCM*).*

Finalmente entre las desventajas de Android Studio están:

- Lentitud.
- Estabilidad.
- En fase de desarrollo aún. Versión Beta.
- Complejidad en el sistema Gradle si no se conoce o para proyectos cuyas dependencias no se encuentran en el sistema Gradle.

Después de lo comentado anteriormente, nuestra elección ha sido usar la versión de Eclipse propia de Google [15]. Las razones principales que han llevado a esta elección son: la mayor estabilidad, la rapidez, el editor XML de Eclipse parece correcto y suficiente y finalmente el hecho de que no se obtenga partido en este proyecto al nuevo sistema de *Building* basado en la tecnología *Gradle* ni usemos tampoco servicios como Google Cloud Messaging (GCM) con lo que las ventajas de Android Studio se reducen drásticamente.

3.2 Setup y plugin ADT de Eclipse

Para desarrollar en Android utilizamos Eclipse junto con el plugin de *Android Development Tools* (ADT). Una vez tengamos el Eclipse con el plugin ADT necesitaremos descargar también la última versión del Standard Development Kit (SDK) de Android. Explicamos cada paso detalladamente.

Una vez descargado y descompromido el Eclipse Google Edition [15] obtenemos lo siguiente: Eclipse (una de las últimas versiones), Android Platform Tools (ADT), el SDK de la última plataforma Android y la última imagen del sistema Android que podremos usar con el emulador por defecto que viene con las ADT. Para descargar otras versiones de Android o algunas bibliotecas específicas para el uso de otras *Application Program Interfaces* (*APIs*) de Google como los mapas o el servicio GCM ejecutaríamos el gestor del SDK de Android e instalaríamos lo necesario, en cualquier caso esto lo explicamos con detalle más adelante.

Al reiniciar el Eclipse después de instalar el ADT Plugin debemos indicarle donde se encuentra nuestra carpeta con el Android SDK descargado previamente. Para ello simplemente hacemos click en *Window* → *Preferences* y a continuación a la izquierda clicamos en la sección de *Android* para después en la derecha, donde aparece *SDK Location* hacer clic en el botón de *Browse* e indicarle donde se encuentra nuestro Android SDK previamente descargado y descomprimido (Figura 3.2).

Con los pasos anteriores, el Android SDK Manager se encuentra en una carpeta llamada *android-sdk* en nuestra carpeta de usuario o en la ruta *c:\Program Files\android-sdk* en equipos con Microsoft Windows. Una vez hecho esto ya podríamos empezar a trabajar en nuestro proyecto.

3.3 Android SDK Manager

A través del SDK Manager podemos gestionar y descargar las diferentes versiones de la API de Google dependiendo del *target* que queramos usar así como otras herramientas de desarrollo de Google, ejemplos, drivers USB para teléfonos móviles de la familia Nexus, bibliotecas de compatibilidad, bibliotecas de Google Play, Google Analytics…

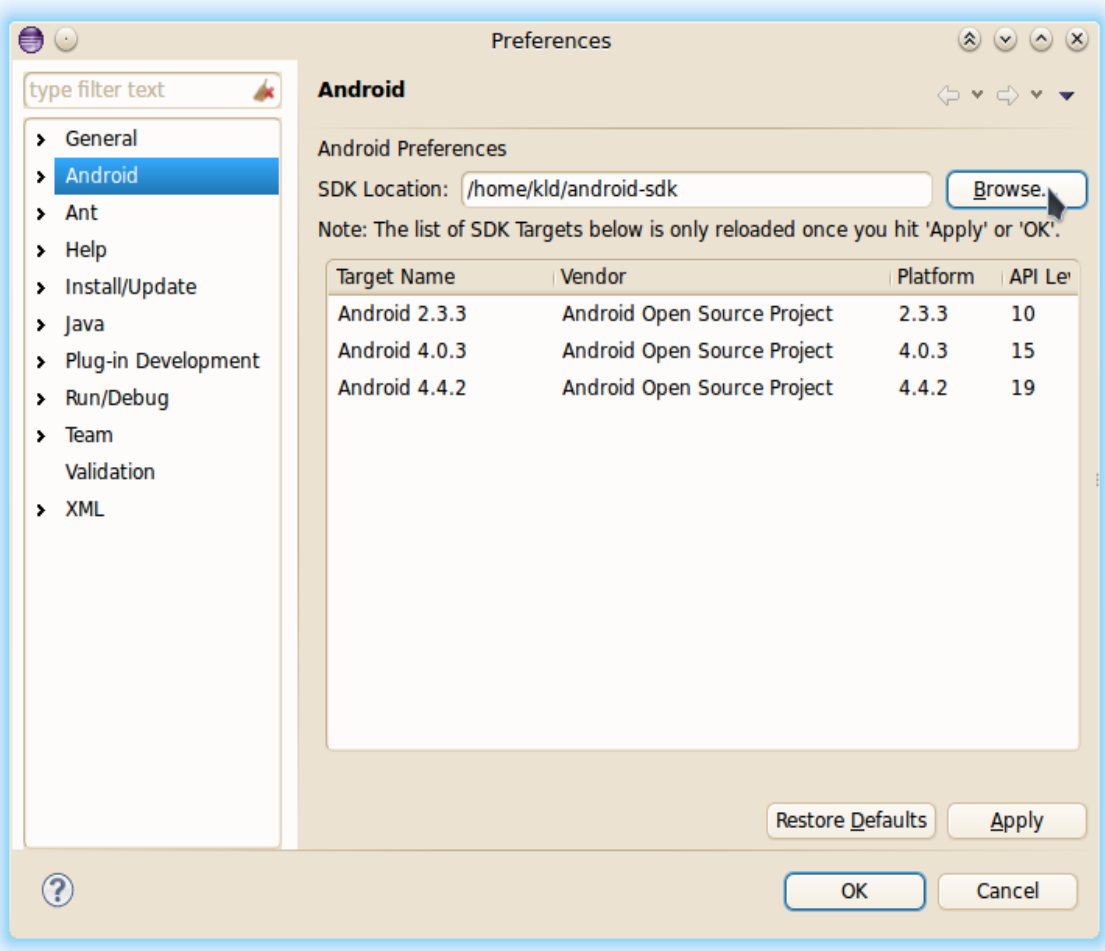


Figura 3.2. Definiendo la ubicación del Android Standard Kit en Eclipse.

Una vez instalado el ADT plugin para Eclipse y configurada nuestra carpeta con el Android SDK accedemos al *SDK Manager* y al *Virtual Device Manager* desde Eclipse simplemente haciendo clic en la barra de menú superior en la sección *Window → Android SDK Manager* y *Window → Virtual Device Manager* respectivamente.

- Ahora para poder continuar con el *setup* simplemente accedemos al SDK Manager para instalar los paquetes básicos (Figura 3.3).

- Ahora instalamos los siguientes paquetes desde el SDK Manager: *Android SDK Tools, Android SDK Platform-tools, Android SDK Build-tools, Android API (última versión)* y en la categoría de Extras, la biblioteca de soporte *Android Support Library.* (Figura 3.4 y Figura 3.5)

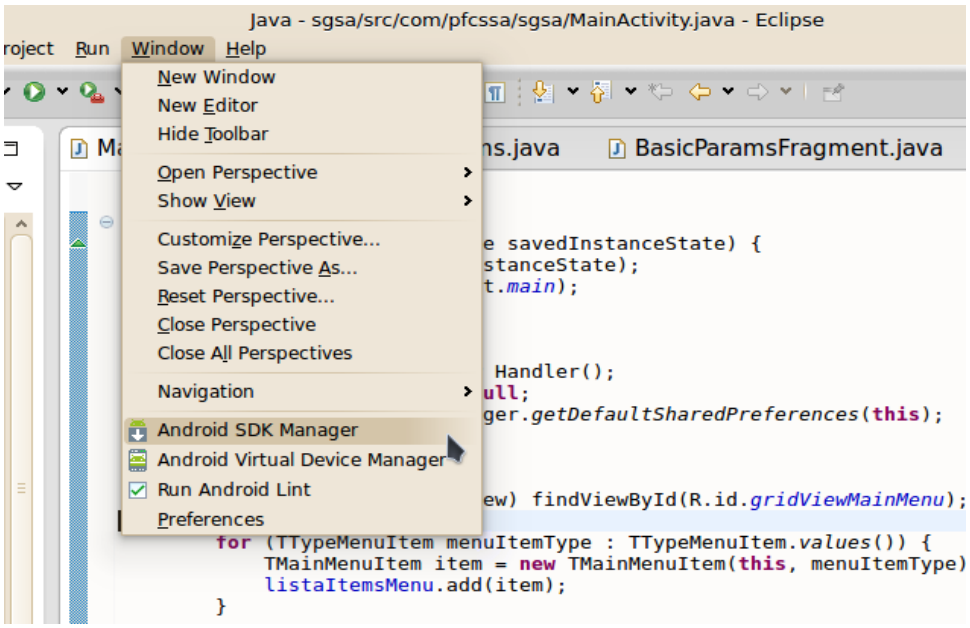


Figura 3.3. Menú del ADT en Eclipse con acceso al SDK y VDM.

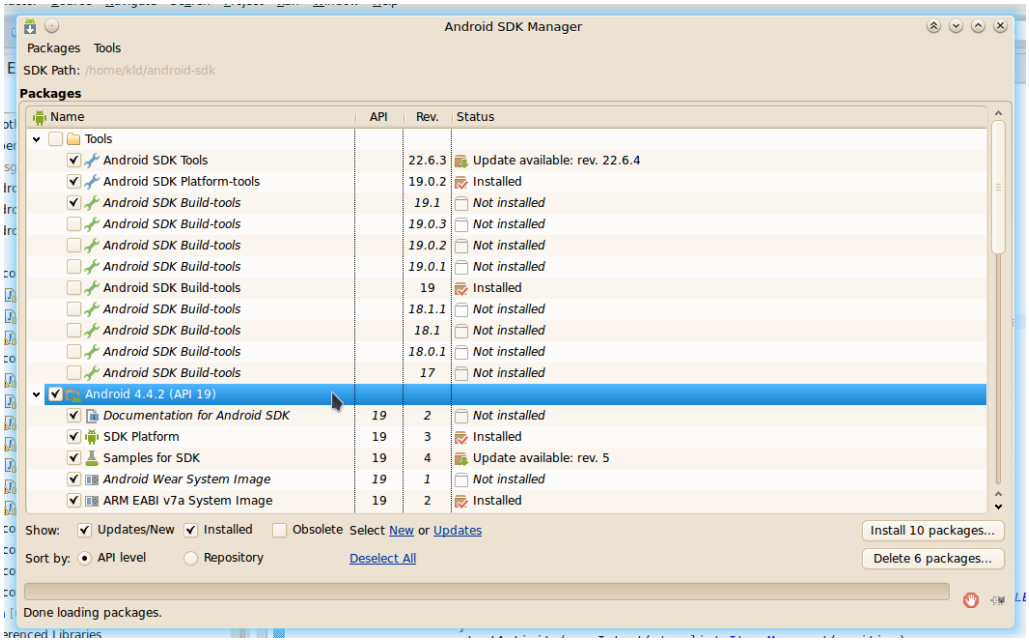


Figura 3.4. Android SDK, Instalación de componentes.

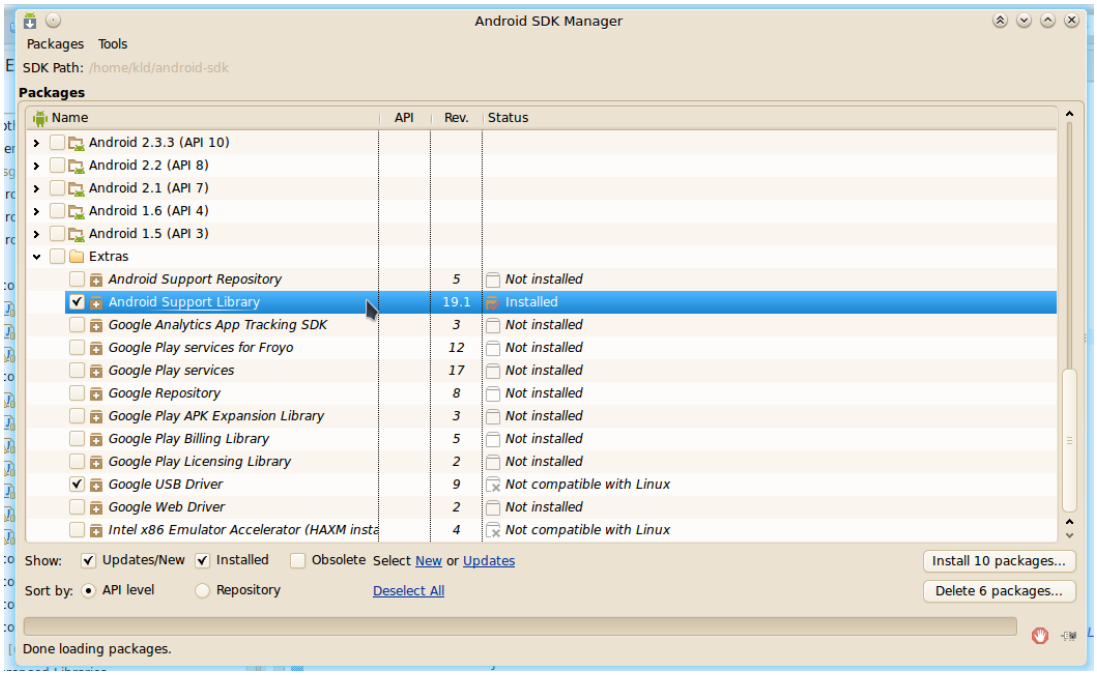


Figura 3.5. Android SDK, Instalación de componentes.

- Finalmente simplemente le daríamos a *Instalar*, aceptamos la licencia y volvemos a hacer clic en *Instalar*. (Figura 3.6)

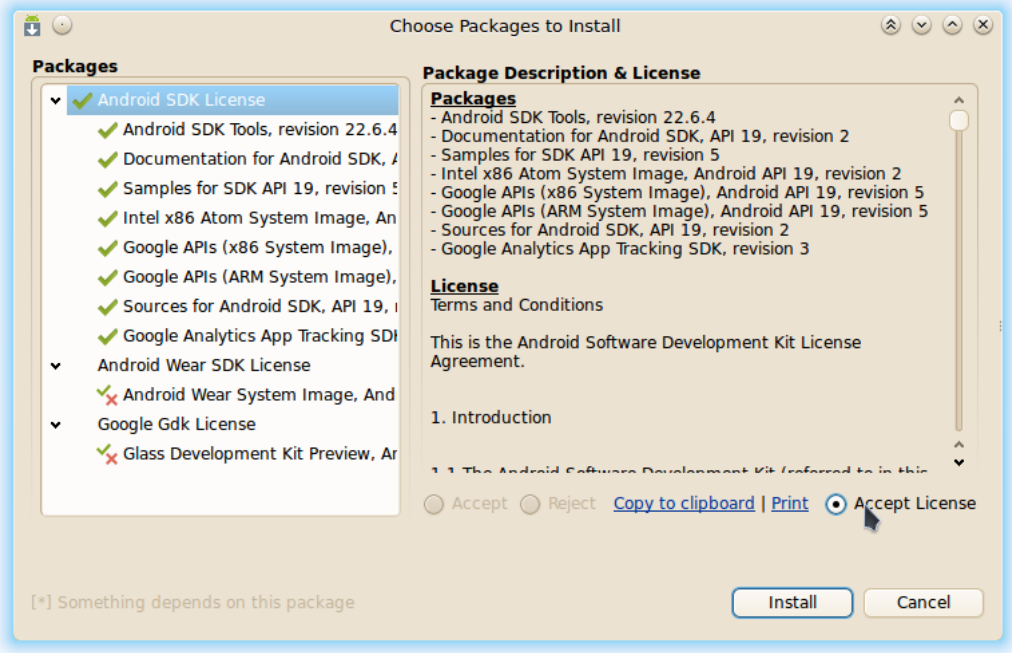


Figura 3.6. Aceptando la licencia e instalando.

3.4 Android Virtual Device Manager

Con el *Android Virtual Device Manager* (*AVDM*) que provee el SDK creamos y gestionamos diferentes teléfonos móviles Android virtuales de una forma fácil y potente. Además podemos crear y ejecutar tantos como queramos o podamos (Figura 3.7).

Dichos teléfonos móviles virtuales permiten probar nuestras aplicaciones en diferentes entornos con diferentes características como diferente resolución, hardware... Esto es muy importante para estudiar cómo se comporta nuestra aplicación en diferentes situaciones. (Figura 3.8)

Además de la alternativa oficial de Google existen otros emuladores Android que trabajan incluso mejor que éste como es el caso del emulador *Genymotion* [13] el cual destaca sobre todo por su velocidad y un uso mínimo de recursos. *Genymotion* suele ser la primera opción de emulador de muchos desarrolladores.

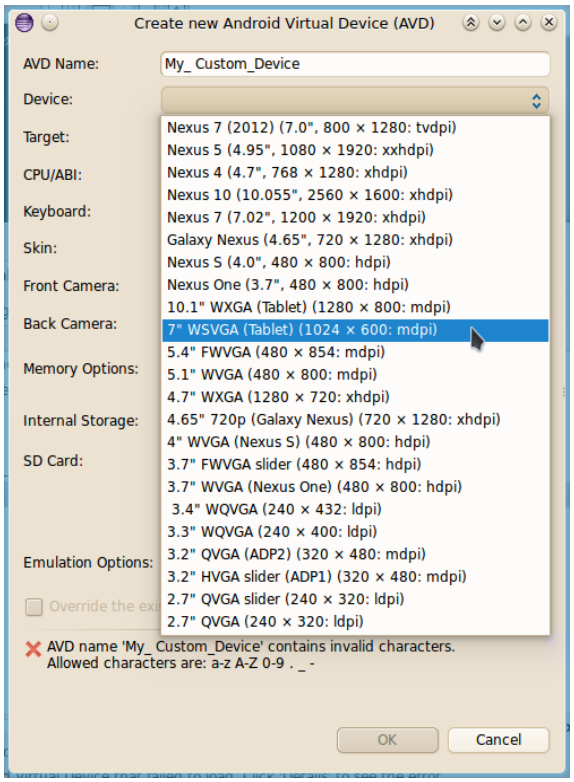


Figura 3.7. Creando un nuevo emulador con el ADVM.

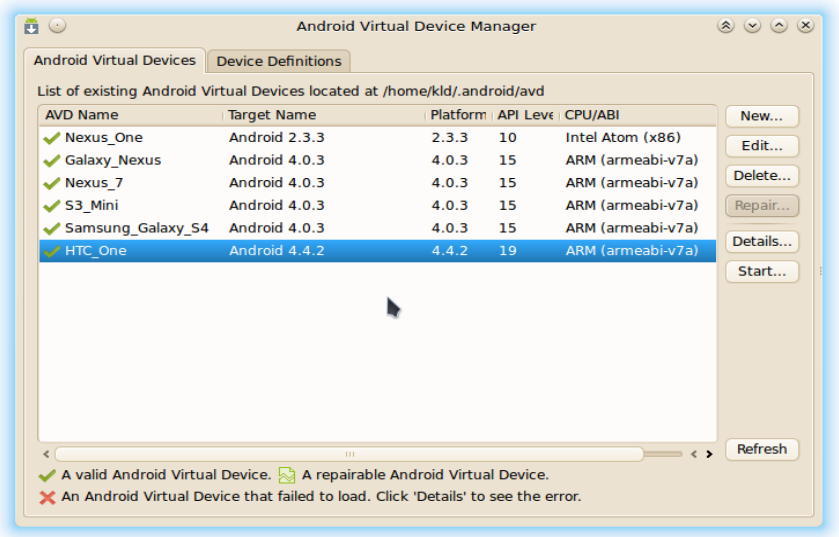


Figura 3.8. Listado de teléfonos móviles en el AVDM.

3.5 Patrón Modelo Vista Controlador

Android hace uso extensivo y profundo del patrón de diseño *denominado Modelo Vista Controlador* (*MVC*) porque permite ser modulares, limpios y poder reutilizar código. Este patrón pretende dividir el desarrollo de una aplicación en estas tres partes. La principal característica de MVC consiste en *separar los datos de la aplicación, de la interfaz de usuario y de la lógica interna en estos tres componentes*. Finalmente la relación de estos tres componentes tendría como resultado nuestra aplicación (Figura 3.9):

- **Modelo:** todas aquellas clases (métodos, servicios, receivers...) que se encargan de gestionar los datos propiamente dichos de la aplicación. Estos datos pueden ser generados por la propia aplicación y salvados en archivos o base de datos, obtenidos como datos de sensores o datos que traemos de Internet a través de *web services*. En cualquier caso todas las clases que gestionan y conformen una API que pueda leer y guardar dichos datos forman lo que se conoce como modelo.

- **Vista:** toda la parte visual o interfaces gráficas (GUI) que permite representar nuestra aplicación en pantalla. En Android toda la definición de gráfica de las ventanas (*activities* y/o *fragments*) se hace a través de unos archivos XML llamados normalmente *layouts.* Además de estos archivos XML que definen la interfaz gráfica también contamos con todos los otros *resources* (Anexo I) a los que podemos hacer referencia en estos archivos XML.

- **Controlador:** todas las clases que ayudan a darle vida a las interfaces gráficas definidas en esos archivos XML. Esas clases ayudan a introducir los datos, manejar las decisiones del usuario y conectar con nuestro modelo de datos. En Android como parte Controladora podemos entender a las diferentes clases de *Activities y Fragments* que tengamos definidas (Anexo I).

En definitiva si conseguimos separar nuestra aplicación en estos tres elementos además de tener el código fuente mucho más ordenado y claro vamos a conseguir reutilizar muchísimo código y de esta manera conseguimos evitar grandes quebraderos de cabeza debido a la repetición de código fuente no eficiente o donde no se ha hecho de la misma forma en un sitio que en otro.

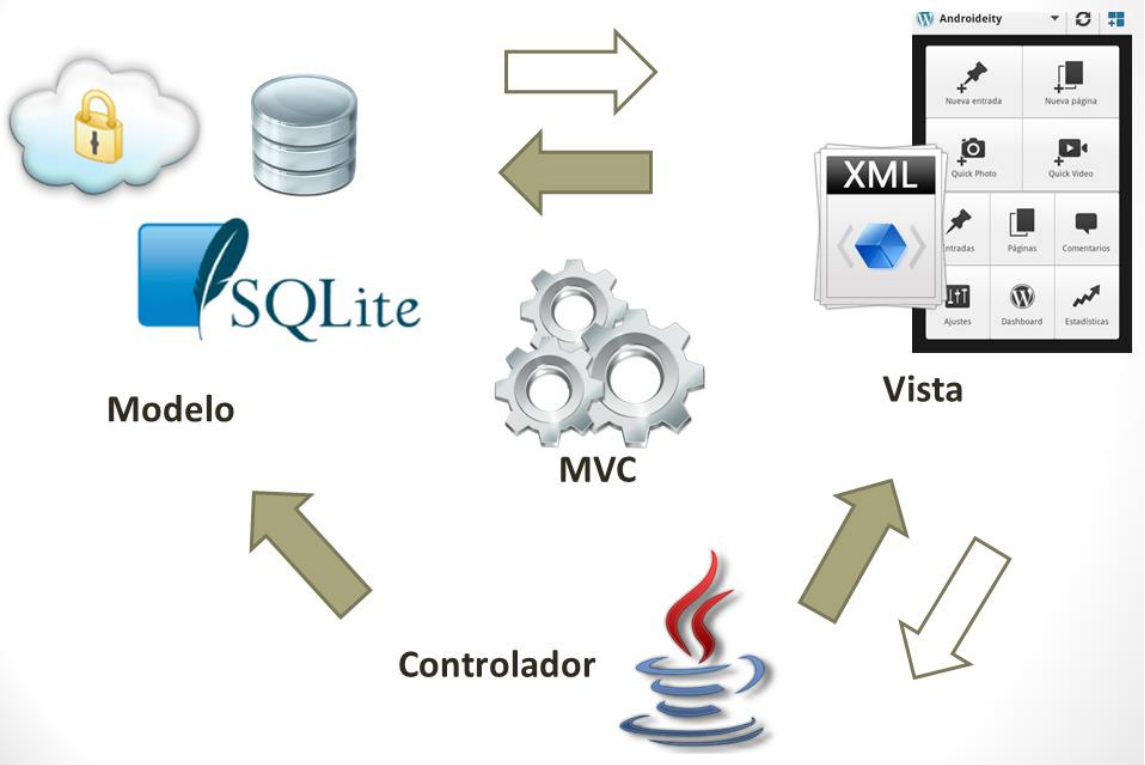


Figura 3.9. Patrón Modelo Vista Controlador.

3.6 Bibliotecas externas

Además de usar las bibliotecas propias de Android en el desarrollo de esta aplicación móvil se necesita hacer uso de una serie de bibliotecas externas las cuales nos van a permitir trabajar sobre seguro en varias áreas críticas.

- *Android Java Mail:* esta biblioteca vamos a poder manejar una parte del envío de correo electrónico así como garantizar la transparencia en esta operación. Esta biblioteca nos provee acceso 100% a una adaptación de la API JavaMail. Dicha versión está basado en una versión libre de esta biblioteca bajo licencia GPL 2.0.

- *Apache Commons Lang:* esta biblioteca es bastante amplia y nos provee un completo set de utilidades (utilidades para java.lang.Api, manejo de strings, métodos para trabajar con números, reflexión, concurrencia, serialización…) de carácter general que usamos en todo el proyecto.

- *Android v4 Support:* esta biblioteca es probablemente la más usada en proyectos Android que desean usar retro compatibilidad ya que es la encargada de implementar y dar soporte de las nuevas funcionalidades de Android 4.x en versiones anteriores.

3.7 Repositorio de código y control de versiones con Git

Un sistema de control de versiones es un sistema que registra todos los cambios de los archivos del proyecto o conjunto de archivos registrando la evolución del proyecto, revertir cambios en el proyecto y/o archivos a estados anteriores, comparar los cambios de archivos, saber quién y cuándo ha modificado el qué, saber qué y quién ha introducido un fallo en el proyecto, corregirlo… En definitiva con un sistema de control de versiones podemos tener un control total de toda la vida del proyecto y si lo sabemos gestionar bien además es una herramienta de trabajo fundamental y básica para la seguridad e integridad del proyecto, sea cual sea éste.

Hemos usado *Git* [16] ya que desde hace varios años su uso está creciendo mucho en el desarrollo de pequeños y medianos proyectos software hasta convertirse en una moda-referente (para grandes proyectos distribuidos ya era un gran referente ya que de hecho nació para gestionar mejor el Kernel de Linux). Otra de las principales razones a la hora de elegir Git ha sido que hoy en día en el Mercado existen infinidad de herramientas gratuitas para trabajar con servidores Git así como clientes para Windows, MacOS y obviamente Linux.

La característica principal de Git es que es un *Sistema de Control de Versiones Distribuido* (*DVCS*) por lo que cada programador tiene en local una copia de todo el proyecto y trabaja de forma local y rápida. Obviamente en cualquier momento puede subir los cambios al servidor o servidores (Figura 3.10). Entre otras características están:

- Sistema de control distribuido.
- Rápido.
- Eficiente.
- Simple.
- Potente.
- Alto soporte para el desarrollo no lineal.
- Capaz de trabajar con proyectos enormes de forma eficiente.
- Compatible con múltiples de los protocolos existentes: Hypertext Trasfer Protocl (HTTP), File Transfer Protocol (FTP), Secure Shell (SSH)...
- Seguro.

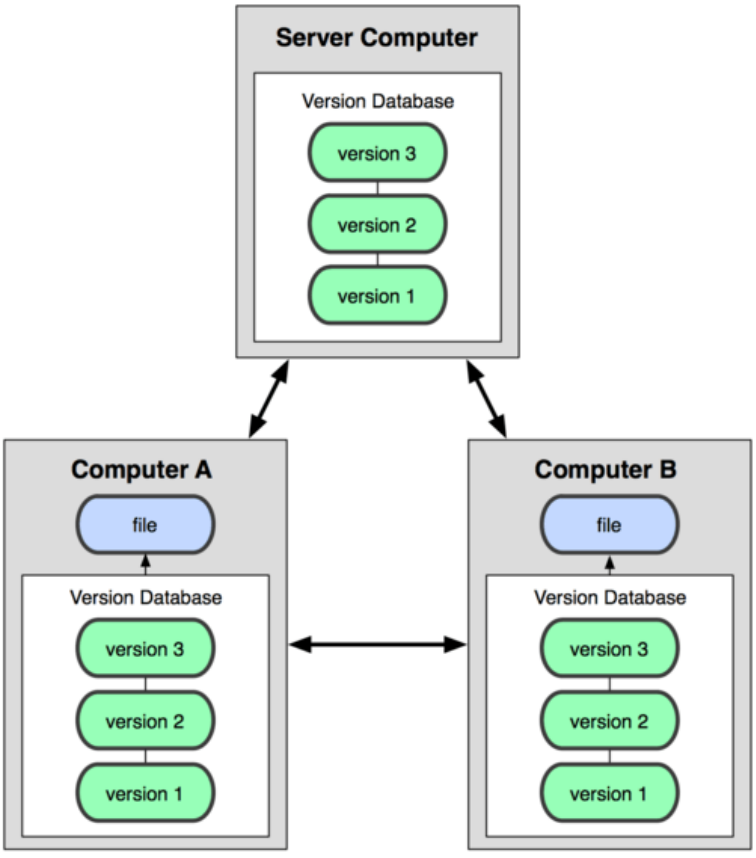


Figura 3.10. Funcionamiento de Git, donde cada usuario tiene una copia del proyecto completo.

Los cambios en los archivos son a nivel de línea, no de bytes o caracteres. Cuando se cambia una palabra o carácter en una línea equivale a eliminar dicha línea y añadir la nueva en la misma posición. Estos cambios no son vistos como diferencias a nivel de archivo sino como *snapshots* (capturas del estado completo del proyecto o rama) de todo el sistema de archivos en conjunto. Es decir, a diferencia de otros sistemas de control de versiones donde se tiene una lista de cambios por archivo en Git cada vez que se realiza un cambio y se hace un *commit* es como si Git tomara una foto del estado de todos los archivos del proyecto (aunque sólo haya variado un único archivo), obviamente para los archivos que no han cambiado se crea un enlace al estado anterior del mismo. A continuación una representación gráfica de los cambios por archivo de un sistema tradicional y el esquema correspondiente en Git (Figura 3.11).

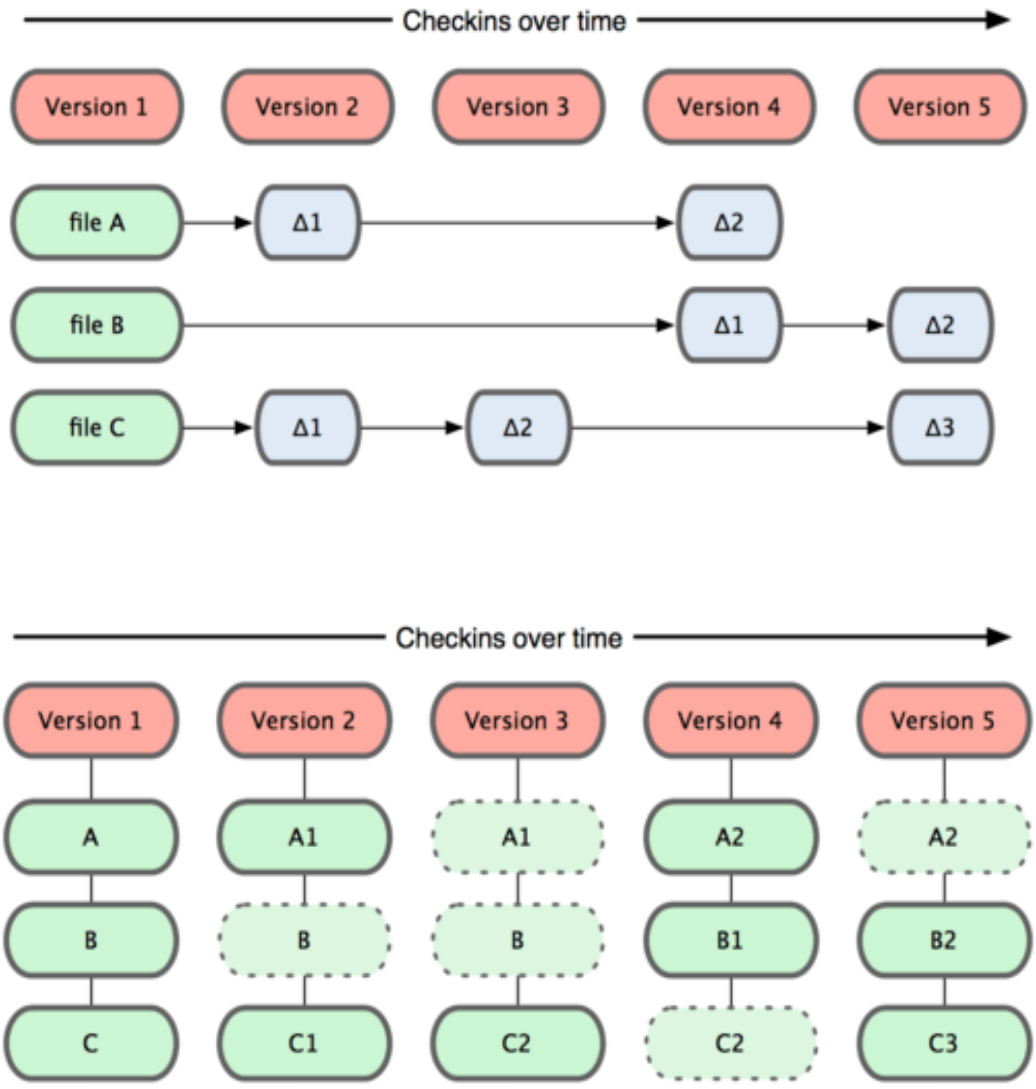


Figura 3.11. Evolución de un proyecto y correspondiente snapshot en Git

El uso de *branchs* (ramas) ayuda al desarrollo no lineal de un proyecto. Las ramas permiten gestionar las diferentes versiones del proyecto, además de usarlas para su principal cometido que es el de solucionar bugs, implantar funcionalidades y su testeo. Se puede crear ramas para cualquier cometido y después optar por fusionarlas con otras (por versión, estado…) o con la rama principal de cada proyecto. En el tema 4.6 seguiremos estudiando el uso de Git en nuestro proyecto y podremos ver las ramas que se han creado para la implementación de éste.

4. Análisis previo, funcional y orgánico

Una vez tenemos el Eclipse configurado con el *plugin Android Developer Tools* (ADT), el *Standard Development Kit* (SDK) de Android descargado con todos los paquetes necesarios instalados a través del SDK Manager ya podemos comenzar a desarrollar la aplicación móvil y en este capítulo estudiamos a fondo dicho proceso realizando un análisis previo, funcional y orgánico de los elementos más importantes que constituyen la aplicación móvil.

4.1 Análisis previo

Los criterios de diseño impuestos para la aplicación móvil fueron presentados en el capítulo 2: funcionamiento sigiloso, control remoto, opciones multimedia, notificaciones por varios canales, alta disponibilidad… Además debe ser: fácil de usar y de activar con alta disponibilidad, exhibir diferentes modos de funcionamiento, modular… Profundizando un poco más en este análisis inicial establecemos los requisitos de funcionamiento que analizamos a continuación.

El SMS (*SMS activador*) enviado debe contener una contraseña definida previamente. Y se debería desarrollar tanto la opción sencilla de activación (simplemente recibir la palabra clave), así como una completa línea de órdenes vía SMS que permitiera un mejor control remoto del teléfono móvil perdido o robado.

El sistema de control de arranque debe detectar un encendido del teléfono móvil no autorizado y debe poderse poner en marcha, su método de control será mediante forma gestual sigilosamente, y en caso de fallo activar alguno de los modos de funcionamiento previamente configurados.

Para implantar la interfaz de usuario se debe diseñar un *grid* de opciones a configurar y/o activar, así como el *registro del sistema* y las otras opciones. Todas las opciones deben acompañarse de una pequeña descripción de funcionamiento, además de que por su posición en pantalla es el orden en que debemos configurar y/o activar dichas opciones. Desde el mismo grid del menú principal también se podrán activar o desactivar las opciones que lo permitan.

Se debe monitorizar el estado de la batería si se activa el servicio de geolocalización abortando el proceso si se estima necesario. Además en cada mensaje de información que se envía al usuario afectado se le debería informar del nivel batería del teléfono móvil remoto. También en el sistema de geolocalización se tienen en cuenta varios parámetros para realizar la operación lo más eficiente posible.

Sería deseable disponer de un histórico de operaciones implantando un sistema completo de registro (log) que estaría encargado de registrar todas las operaciones (realizadas de forma sigilosa o no) que se hagan en el teléfono móvil. Obviamente a dicho sistema sólo se debería acceder mediante la aplicación móvil.

Debido a que se usa el SDK de Android para implantar la aplicación móvil, no existen muchas alternativas de implantación que merezcan la pena discutir. Se usa los componentes

más adecuados. En cambio sí es interesante analizar los casos de uso de la aplicación que se han realizado utilizando el *Unified Modeling Language* (*UML*).

En la Figura 4.1 se muestra el diagrama de casos de uso de la aplicación móvil. En este diagrama se muestra todas las operaciones básicas que puede realizar el usuario con la aplicación móvil. Las cuatro primeras opciones de configuración dependen de tener realizada la operación de *Configuración básica*. Todas las operaciones dependen del módulo de control de seguridad que en el diagrama se representa con la acción *Hacer login si existe.*

En la Figura 4.2 se muestra el diagrama de casos de uso cuando se ha robado o perdido el teléfono móvil. Básicamente, lo único que tiene que hacer el usuario si tiene nuestro sistema instalado es enviar un SMS *activador.* Esto lo puede hacer haciendo uso de Internet o usando el teléfono de alguna persona conocida.

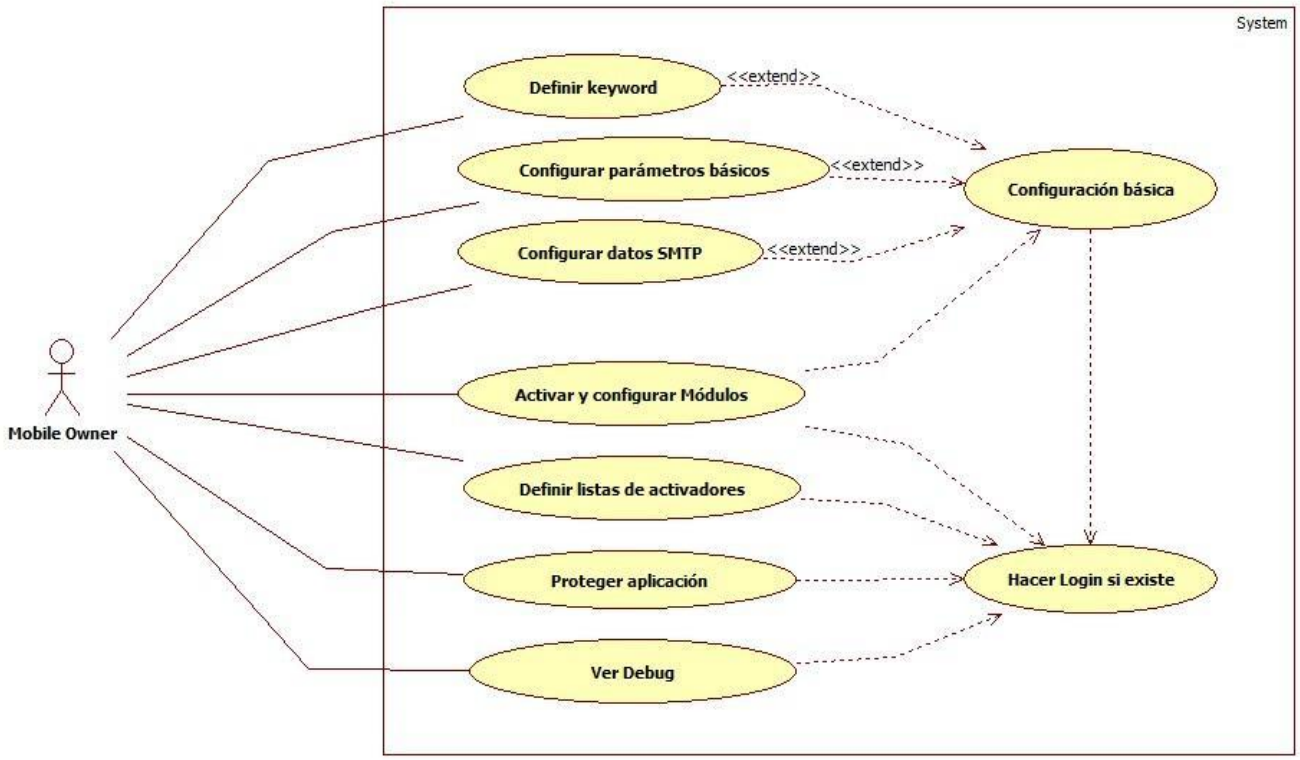


Figura 4.1. Diagrama de casos de uso de la aplicación móvil

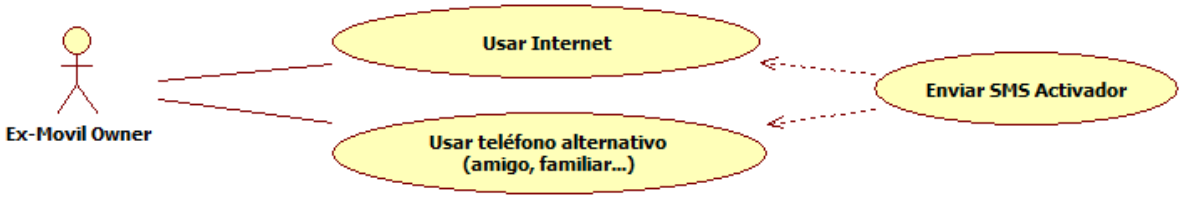


Figura 4.2. Diagrama de casos de uso en situación de pérdida o robo

4.2 Análisis funcional

A partir del análisis previo realizado es posible identificar la funcionalidad de la aplicación móvil. Para ello, identificamos un conjunto de módulos funcionales y su funcionalidad principal.

Los módulos principales de la aplicación móvil son los siguientes:

- *Modulo Activador*: decide cuándo se ha recibido correctamente la palabra clave configurada (*keyword)* y qué módulos se deben activar en consecuencia de los parámetros recibidos en el mensaje activador.

  Cuando se recibe un SMS decide si debe analizarlo o no dependiendo de si está activo algún tipo de seguridad de usuarios. Si el usuario remitente es válido entonces se comprueba si existe la palabra clave en el mensaje recibido. Si la comprobación *bruta* es verdadera entonces se activa un servicio que entre otras funciones es el encargado de analizar el mensaje mediante una batería de expresiones regulares que determinan si efectivamente el mensaje es válido así como los parámetros que contiene. Una vez determinados los parámetros dicho servicio activa los módulos correspondientes, por ejemplo: *captura de audio, geolocalización…*

- *Módulo de Geolocalización*: proporcionar información lo más precisa posible (sin redundancia de datos) y debe de ser capaz de:

- *Activarse cuando el usuario infractor no esté haciendo uso del teléfono móvil*.
- *Cuidar la calidad de la señal de red*.
- *Controlar la calidad y precisión de la señal GPS*.
- *Controlar el nivel de batería del teléfono móvil*.

El sistema debe ser capaz de abortar el proceso en caso de que alguno de estos parámetros no sean los adecuados, siendo capaz de reactivarse desde que sea posible para dar al usuario la mejor información posible.

- *Módulo Alarma*: implementa un sistema de alarma sonora muy estridente con el que se pretende poder localizar el teléfono móvil en pequeños espacios (por ejemplo una discoteca, bar/cafetería...) o cuando se crea que el infractor está lo bastante cerca de nosotros como para oír la alarma del teléfono móvil.

  La característica fundamental de este modo es que independientemente de que el teléfono móvil esté en silencio, o del nivel de volumen que tenga configurado cuando se active este modo el teléfono móvil suena y vibra con el nivel de volumen indicado y durante el tiempo configurado. No se puede hacer nada para detener o bajar el volumen de dicha alarma salvo o esperar el tiempo configurado o bien enviar un SMS activador con el parámetro *stop* que detiene la alarma así como otros módulos si estuviesen activos.

- *Módulo de control de arranque*: proteger el teléfono de los inicios no autorizados. Para ello cada vez que el teléfono móvil se inicie solicita de forma lo más sigilosa posible la introducción de un patrón libre (y opcionalmente invisible).

  Es útil ya que en caso real de hurto es probable que justo después de que el teléfono móvil fuese robado el infractor lo apague (para cambiarle o quitarle la tarjeta SIM) para después volverlo a encender en un momento más cómodo para éste. Pues es precisamente en estas situaciones cuando el módulo de control de arranque entraría en acción detectando que nuestro teléfono móvil ha sido robado y llevando a cabo las acciones previamente configuradas.

  Para que este módulo funcione correctamente se necesita configurarlo y activarlo previamente. Lo primero que tenemos que hacer es definir al menos uno de los *gestos* (patrones libres) de seguridad de los tres que permite el

sistema. A la hora de definir estos tres gestos disponibles de control de arranque lo lógico sería definir aproximadamente el mismo gesto pero de 3 formas diferentes (*uno más pequeño, otro empezando en sentido contrario...*) con el fin de evitar que no se active el sistema de seguridad accidentalmente.

Una vez definido el (o los gestos) de seguridad lo siguiente sería indicar que módulos ejecutamos si falla el control de arranque y finalmente indicar el tiempo que tenemos para introducir correctamente el patrón desde que se solicita.

Cuando todo el sistema de control de arranque está configurado y activado éste se ejecuta automáticamente al iniciar el teléfono móvil. Cuando se encienda el teléfono móvil el sistema solicita el gesto de desbloqueo de la forma más transparente posible, y, sólo se sabe que está solicitando el gesto de desbloqueo porque el teléfono ha emitido un beep durante 2 segundos (independientemente de que el teléfono esté en silencio) pero en cambio en la pantalla principal no vemos nada (excepto nuestro fondo habitual y el *launcher* de Android que tengamos configurado por defecto) sin embargo debemos tener precaución sobre todo si hemos indicado que el gesto sea *transparente* ya que aunque no veamos nada realmente se está solicitando que lo próximo que dibujemos en la pantalla sea el control de desbloqueo, si lo hacemos correctamente el sistema indica que el acceso ha sido permitido, en cambio, si no introducimos el gesto correcto o se alcanza el *timeout* previamente definido, el sistema ejecuta las acciones configuradas previamente de forma transparente.

- *Módulo de audio:* graba audio de forma totalmente transparente al usuario. El sistema utiliza tanto el formato de salida como el *encoder* por defecto del sistema para no tener problemas dependiendo del teléfono móvil.

  Una vez ha finalizado el proceso de grabación el sistema genera el archivo de audio que se envía al usuario. Si se ha solicitado también realizar una fotografía el sistema antes de enviar el archivo de audio realiza la fotografía antes de enviar nada al usuario.

  Cuando el sistema dispone del archivo de audio y de la fotografía realizada, si es que también se solicitó, el módulo compone el e-mail donde además de los datos de los elementos capturados se informa del nivel de batería del teléfono móvil remoto. Finalmente adjunta los archivos al email y los envía usando el módulo de envío de correo electrónico.

- *Módulo de fotografía:* realiza fotografías del infractor o la zona donde se puede encontrar.

  Al igual que en el resto de módulos se ha trabajado para que éste sea capaz de capturar fotografías de la forma sigilosa para el usuario que maneja el teléfono. El sistema es capaz de realizar las fotografías en menos de 500ms e incluso es capaz de realizar la fotografía teniendo la pantalla totalmente apagada.

  Lo primero que hace es iniciar la configuración del teléfono móvil de hardware que va a usar. Para ello primero analiza todos los dispositivos de captura de imágenes de que dispone el teléfono móvil. Si dentro de la lista de teléfonos móviles disponibles existe alguna cámara frontal ésta tendría prioridad sobre las cámaras principales (posterior). Una vez selecciona el teléfono móvil de captura de imágenes que va usar ajusta la orientación de la imagen dependiendo de si está usando una cámara frontal o posterior para después terminar de configurar los siguientes parámetros; formato de salida de imagen a JPEG, desactiva el flash de la cámara si lo hubiese, establece el modo de enfoque en automático y desactiva cualquier efecto de pos-procesado.

- *Módulo de Protección*: protege el acceso a la aplicación de configuración del sistema para evitar que el usuario infractor pueda desactivar el sistema o percatarse de que aplicación es. Para ello realiza una protección típica mediante un acceso con contraseña definida.

  Tiene la particularidad de que aunque el usuario haya abierto y obtenido un acceso positivo a la aplicación, una vez se apague la pantalla, el usuario cambie de aplicación o se cumpla un *timeout* definido, cuando el usuario recupere dicha aplicación el sistema le vuelve a solicitar la contraseña.

  Aunque este comportamiento es un poco molesto es el más seguro de cara a evitar que la aplicación se quede en memoria con un acceso permitido y después cualquiera pueda acceder a su configuración. Además se entiende que la configuración de una aplicación de este tipo no se hace de forma asidua.

4.3 Análisis orgánico

A partir del análisis funcional realizado es posible identificar el comportamiento de los distintos módulos que lleven a cabo las acciones necesarias para cumplir con el análisis de requisitos efectuado para la aplicación móvil. Para ello, identificamos un conjunto de módulos orgánicos y su comportamiento principal expresado en base a patrones de diseño conocidos o diagramas de clases y secuencias de acciones entre ellas.

4.3.1 Módulo Activador

La forma de decidir si ha llegado un mensaje activador válido, de forma bruta pero rápida es usar el método *contains* de la clase String de Java. Esto permite decidir de forma rápida si debemos analizar el mensaje de forma estricta o no. El posterior análisis de forma estricta se hará mediante expresiones regulares. En esta análisis inicial también se comprueba el usuario remitente y el tipo de lista de seguridad si es que hay alguno activo.

Clases que lo componen

SmsReceiver.java: en esta clase tenemos definido el *receiver* principal que escucha los SMS recibidos y decide si es o no un mensaje activador válido.

MainService.java: el servicio principal el cual después de recibir un SMS activador válido analiza el mensaje para ver que parámetros lleva y qué módulos solicita el usuario que activemos remotamente.

Activadores.java: es una *activity* encargada de permitir al usuario configurar seleccionar el tipo de usuarios activadores que van a poder enviar mensajes activadores. Para esto el usuario crea una lista de usuarios de tipo *blacklist* o *whitelist* (o ninguna de éstas).

4.3.2 Módulo de Geolocalización

Este módulo (también llamado internamente *Modo Sigilo*) es uno de los más importantes y cuando se inicia se configura con los parámetros que se han establecido en la aplicación o bien con los parámetros de entrada recibidos si los hubiera.

Clases que lo componen

MainService.java: el servicio principal también es el encargado de registrar, inicializar y gestionar los servicios de geolocalización.

ModoSigilo.java: actividad donde podemos activar y ver la configuración por actual de este modo.

ConfigSigilo.java: p*referenceActivity* donde establecemos la configuración de todo el modo.

Parámetros del modo

Tiempo de apagado: número de minutos tras los cuales el servicio se detiene. Este parámetro se configura en la aplicación pero también se puede enviar como parámetro en el SMS activador y es éste último el que prevalece sobre el configurado en la aplicación.

Variación en metros: variación de metros que el sistema interpreta como una nueva señal. Es decir si la nueva señal GPS ha variado *x* metros respecto a la anterior informada al usuario se reenvía esta nueva señal.

Responder vía SMS y teléfono por defecto: indica si se debe responder automáticamente mediante SMS al remitente y/o a un número configurado por defecto.

Responder siempre al remitente: indica si el sistema debe responder siempre al remitente que ha solicitado activar el servicio. El sistema compueoba que el remitente y el número por defecto no sean el mismo para no repetir la información.

Responder vía e-mail y dirección por defecto: indica si se debe responder automáticamente mediante e-mail a la dirección configurada por defecto.

Dirección de e-mail parametrizada: en el mismo SMS activador también se puede enviar una dirección de e-mail alternativa donde enviar la información de posición del teléfono móvil.

Precisión mínima de la señal: este parámetro es interno y sólo configurable a través del código fuente. Actualmente tiene un valor de 15 m y es el valor que usa el sistema para dar por válida una señal GPS.

Una vez se inicia, dicho servicio trata de activar los servicios de localización nativos del sistema Android (localización mediante sistemas GPS, Glonass…) y redes móviles/Wireless Fidelity), si dichos servicios nativos no estuviesen disponibles el sistema envía un mensaje al remitente indicándole que no ha sido posible obtener la localización del teléfono móvil.

Si los servicios de localización estuviesen disponibles se inicia el protocolo normal de funcionamiento; el sistema comienza a buscar una señal de geolocalización válida (con una precisión igual a *MIN_GPS_SIGNAL_ACCURACY*). Cuando el sistema tenga una señal válida obtiene los datos de dirección real de esa posición (usando los servicios de Google), genera una dirección Uniform Resource Locator (URL) corta con la posición del teléfono en Google Maps e informa al usuario mediante SMS y/o e-mail de la ubicación detectada, parámetros de la señal (precisión de la señal y velocidad si tuviese), la URL previamente generada y el nivel de batería del teléfono móvil.

Si el sistema llegase a alcanzar el tiempo de apagado sin haber logrado informar de una señal GPS válida, antes de apagarse informa al usuario con la señal más precisa que hubiese encontrado.

4.3.3 Módulo de Alarma

Cuando se activa se ejecuta la alarma con los parámetros configurados por defecto en la aplicación. Este módulo no permite parámetros en el SMS activador.

Clases que lo componen

MainService.java: en este caso el servicio principal sólo actua como llamador al módulo de alarma.

Alarma.java: en esta clase se implementa internamente la alarma, el control de volumen, el tiempo de apagado...

ModoAlarma.java, *ConfigAlarma.java*: estas dos actividades son las encargadas de la parte visual y de configuración del módulo.

Parámetros del modo

Nivel de sonido: se indica el nivel de volumen en el sonido de la alarma. El sistema da cuatro opciones: *bajo, medio, alto y máximo.* Siendo éste último el recomendado.

Duración del sonido: duración de la alarma en segundos y minutos. El tiempo máximo permitido es de 5 minutos y el mínimo de 5 segundos.

Vibración: indica si se desea vibración o no.

4.3.4 Módulo de control de arranque

Una vez configurado y activado el módulo, cuando el sistema detecta que el terminal acaba de encenderse a través del *Broadcast Receiver* implementado y la acción *ON_BOOT_COMPLETED* del sistema Android entrará en funcionamiento un servicio especial (ServiceGestureLogin) que se encargará de lanzar el proceso de control de arranque mediante el proceso gestual en el momento adecuado.

Clases que lo componen

SmsReceiver.java: usa el SmsReceiver para detectar el encendido del teléfono móvil e inicializar el servicio de control *ServiceGestureLogin.java*

ServiceGestureLogin.java: se encarga de controlar la activación y cierre de la actividad de *StartupGestureLogin*.

StartupGestureLogin.java: actividad que implementa propiamente dicho el control de login mediante el gesto en la pantalla. Es una actividad un tanto especial ya que se ha hecho totalmente transparente.

ConfigGestures.java, *DefineGesture.java*, *Gesture Tester.java*: encargadas de permitir configurar, definir y probar el modo y los gestos creados.

4.3.5 Módulo de grabación de audio

Una vez concluida la grabación genera un archivo con extensión *.mp4.* Si no se especifica ningún parámetro de duración, se utiliza la duración por defecto *DEFAULT_RECORD_AUDIO_TIME*. Se usan los valores de *encoder* y formato de salida estándar del teléfono móvil para evitar problemas de formatos.

Clases que lo componen

MainService.java: encargada de llamar a la clase correspondiente y recibir el resultado para gestionarlo.

AudioRecorder.java: realiza la grabación de audio propiamente dicho y avisar al servicio principal una vez haya finalizado.

Parámetros del modo

Duración del audio: se especifica en el SMS activador en minutos. El valor se normaliza para que no sea inferior de 5 segundos y no sea mayor de 3600 segundos (5 minutos).

Ejemplos

keyword g 3: solicita sólo servicios de grabación durante 3 minutos.
keyword g: solicita sólo servicios de grabación durante el tiempo por defecto (30s).
keyword m 45: solicita servicios multimedia (foto + grabación). Graba durante 45s.

4.3.6 Módulo de captura de imagen

Establece un tamaño máximo de ancho de imagen que viene dado por la variable de sistema *MAX_WIDTH_PICTURE*, la altura es proporcional a la resolución del teléfono móvil usado. Una vez ha realizado la captura genera el archivo JPEG cuyo nombre tendría el formato *picture_datetime.jpg* que guarda de forma temporal en el subdirectorio *.pics/* dentro de la carpeta de la aplicación hasta que sea enviado. Para terminar avisa al servicio del sistema de que la fotografía solicitada ha sido realizada y le pasa la ubicación de ésta.

Clases que lo componen

MainService.java: el servicio principal vuelve a ser el encargado de ejecutar los módulos necesarios en este caso realiza una llamada a la clase *TakePicture*.

TakePicture.java: esta es la clase encargada de realizar la fotografía. Es bastante compleja ya que selecciona la cámara frontal si existe, desactiva el flash y una vez ha realizado la fotografía sin dejar rastro avisa al receiver con la ubicación del archivo resultado.

PhotoTakenReceiver.java: recibe el resultado de la fotografía realizada para volver a avisar al servicio principal.

CameraPreview.java: se ha usado para pruebas con la cámara.

4.3.7 Módulo de protección de la aplicación

Al principio no hay contraseña definida por lo que este es el primer paso, definir la contraseña. Para ello el sistema pide que introduzcamos una contraseña para proteger el sistema. A continuación el sistema genera una clave hash para la contraseña proporcionada con el algoritmo de cifrado especificado en *HASH_ALGORYTHM*. Este hash se convierte a hexadecimal y se guarda en el archivo de preferencias por defecto de la aplicación (getDefaultSharedPreferences). Esto se hace obviamente por establecer un punto básico de seguridad y no guardas las contraseñas de forma explícita en los archivos de configuración XML de la aplicación. Una vez tenemos la contraseña definida y la protección activa, en el momento de acceder a cualquier módulo de la aplicación se solicita dicha contraseña mediante un diálogo, impidiendo el acceso a la aplicación hasta que no se haya hecho un acceso correcto.

Si la contraseña introducida es correcta daría acceso al sistema durante el tiempo establecido en *MAX_SECS_RELOGIN* o hasta que se cierre la aplicación. Una vez transcurrido este timeout se volvería a solicitar la contraseña y en caso de introducirla adecuadamente volvería a proporcionar el mismo tiempo de login (*MAX_SECS_RELOGIN* = 120). En caso de introducir la contraseña erróneamente se vuelve a solicitar hasta un máximo de 2 veces, la 3ª vez se cierra la aplicación completa y queda bloqueada durante 1 hora. Este bloqueo no detendría los módulos que estuviesen activos ya que solo afecta al acceso a la configuración del sistema (*MAX_LOGIN_TIMES* = 2 y *MINUTES_LOCKED_STATE* = 60).

Clases que lo componen

SetPassword.java: actividad encargada de configurar la password y activar la seguridad.

Login.java: actividad de bloqueo propiamente dicha que no deja realizar otra acción hasta no introducir la contraseña definida correctamente.

4.3.8 Módulo de correo electrónico interno

Envía los correos electrónicos de forma totalmente independiente y transparente, es decir, sin hacer uso de un Intent de Android con una ACTION_SEND y el mimeType correspondiente ya que esto generaría el envío a través de una de las aplicaciones de e-mail instaladas en el teléfono móvil y el acto de enviar tendría que ser manual. En definitiva, este módulo permite pasarle la información que enviemos por e-mail junto con los datos del destinatario y él se encarga de enviarlo de forma totalmente autónoma y sin dejar rastro.

Funcionamiento

Este módulo para enviar e-mails se basa en una versión libre de la API de JavaMail adaptada para Android. Se usa el protocolo SMTP con lo que uno de los pasos necesarios para que el módulo funcione correctamente es configurar los datos de acceso al servidor *Simple Mail Transfer Protocol* (*SMTP*). Estos datos son: *servidor smtp, puerto, nombre de usuario y contraseña* y se configuran en el apartado de configuración de la aplicación. Si estos datos no se configuran correctamente no funciona el envío de correo electrónico.

Se pueden usar los mismos datos de login de la cuenta de gmail que tengas configurada en el teléfono móvil sabiendo que el servidor SMTP de Gmail es smtp.gmail.com y el puerto es el 465, de hecho desde que la aplicación detecta que es una cuenta de Gmail escribe automáticamente estos valores. Aunque se use el servidor SMTP de Gmail con las mismas credenciales de la misma cuenta configurada en el teléfono móvil no queda registro de los correos electrónicos enviados ya que el proceso se ha realizado fuera de la aplicación de Gmail y por lo tanto sin hacer uso de su API.

Como comentamos anteriormente se hace uso de una API que es una adaptación de las primeras versiones de JavaMail. Internamente se usa la API en una clase estática que

implementa una cola mediante una lista que va gestionando el módulo. Finalmente el proceso de envío se hace usando un AsyncTask (thread) ya que es una operación lenta y bloqueante.

Con este método se tiene la versatilidad de poder llamar al módulo enviarle el objeto tipo email que enviémos y dejar que él gestione su lista interna y envíe el mensaje cuando sea posible sin bloquear el servicio principal y thread principal de la aplicación.

Clases que lo componen

ConfigMail.java: actividad que se encarga de permitir al usuario configurar los datos del servidor de correo *smtp* para poder enviar los correos con dichos datos.

Correo.java: envía los correos electrónicos vía protocolo SMTP con los datos de usuario definidos previamente. Se basa en una modificación de *java mail* para Android.

4.3.9 Módulo de Registro

Para el desarrollo de este proyecto se ha creado un sistema propio e interno de registro (logging) con persistencia de datos. Este sistema permite crear mensajes de diferente categoría: debug, information, warning, error y critical; estos mensajes son mostrados tanto en el sistema de log nativo de Android (*android.util.Log*) como enviados a un archivo interno que gestiona el propio sistema de seguridad y que puedan ser consultados con posterioridad en cualquier momento en la sección de *Consola de Debug.*

Esto permite poder consultar siempre el estado de las últimas operaciones que ha realizado el sistema. Todos estos mensajes van organizados y ordenados por su tipo y un *timestamp* (fecha y hora).

Funcionamiento

Su funcionamiento se basa en una clase estática llamada *Logeo* que implementa todos los métodos necesarios para realizar las funciones de registro con persistencia.

Su funcionamiento básico consiste en el manejo de un *ArrayList* estático donde a través de los métodos de escritura va añadiendo los mensajes solicitados a la lista en memoria, dichos

listado en cualquier puede ser volcado a disco a través del método *commit().* También se disponen de otros métodos para generar un archivo en formato HTML para que se pueda leer de forma más cómoda en la aplicación y así poder exportar/adjuntar al correo electrónico que es precisamente una de las opciones que permite la aplicación móvil mediante la cual veremos más adelante.

4.3.10 Servicio acortador de URLs

Su función es devolver de forma inmediata la forma corta de cualquier URI que le pasemos como parámetro. Se usa principalmente para poder devolver la dirección URI de la ubicación exacta en Google Maps ya que esta URI generalmente es bastante larga y no cabría correctamente en un único SMS de 160 caracteres. Esta función se basa en el servicio de Google llamado *Google URL Shortener* el cual nos brinda a través de la consola de Google acceso a su API la cual mediante un método POST nos devolverá las URLs de entrada en formato corto.

Lo constituye una única clase llamada *UrlShortener*, la cual implementa un único método que ejecuta el servicio REST *urlshortener* de la API de Google.

Funcionamiento

Este método realiza una conexión HTTP y llama al método REST definido en *GOOGL_URL* para ejecutar la POST ACTION pasándole los datos de la URL a acortar en el JSON de entrada. Inmediatamente se queda esperando la respuesta en la misma conexión y una vez reciba un 200 OK, analiza manualmente los datos y obtener la URL acortada la cual devuelve inmediatamente. La clave de esta llamada es realizar una conexión HTTP y llamar al método REST lo más rápido posible y de forma bloqueante (síncrona) pero con un control de timeout. Si hubiese cualquier problema en la red o si cumplimos el time-out el sistema se vuelve la misma URL sin acortar. Una de las claves está en no utilizar ningún método de parseo de Jsons automático como *Jackson* o *Gson* ya que añadirían complejidad al proceso y lo harían más lento por lo que no sería tan inmediato y el timeout se podría cumplir. En definitiva, la principal ventaja de éste método es que en la misma llamada síncrona recibimos el resultado deseado.

4.4 Proyección de los módulos orgánicos en un proyecto Eclipse-Android

Para entender mejor un proyecto Android es fundamental conocer de forma general cual es la estructura de un proyecto en Android y en especial usando Eclipse. Conociendo su organización presentamos cómo se estructura el proyecto, así como los diferentes elementos que lo componen y cómo se relacionan estos entre sí. En la Figura 4.3 se muestra la estructura completa de nuestra aplicación móvil.

Lo primero que se observa es la carpeta *Sistema General de Seguridad en Android* (*SGSA*) que se refiere a la carpeta raíz y principal de nuestro proyecto y entre paréntesis podemos leer [*sgsa develop*] que hace referencia a la rama del Git (sección 4.6) en la que se encuentra en este momento: *develop.* A partir de aquí empezamos a analizar los puntos y la estructura del proyecto en sí haciendo referencia a los puntos marcados en la Figura 4.3:

- *Punto 1:* se hace referencia al target usado y por lo tanto a las bibliotecas propias de Android necesarias así como las bibliotecas dependencias de nuestro proyecto. Este apartado lo genera automáticamente el propio SDK de Android una vez definimos nuestro target y añadimos dependencias al proyecto.

- *Punto 2:* se observa toda la carpeta *src* del proyecto que es donde están todas las clases java con el código fuente de éste. Podemos ver el nombre de los *packages* que hemos creado para tener mejor ordenado el código fuente. Podemos crear tantos *packages* como queramos y con la estructura que queramos.

- *Punto 3:* en este punto seguimos dentro de la carpeta *src* y simplemente observamos el contenido del package *com.pfcssa.sgsa* con las 3 clases que representan el splash screen, la actividad principal y una clase de apoyo para los elementos del menú principal.

- *Punto 4:* por orden, encontramos con las bibliotecas referenciadas del propio SDK de Android (se autogenera). Después con la carpeta de *assets* que es para recursos libres que en este caso no usamos y por lo tanto está vacía. La carpeta *bin* donde se generan los archivos ejecutables .apk generados por Eclipse. La carpeta *externals* la hemos creado nosotros para guardar algunas versiones de otras bibliotecas pero no afecta al proyecto en sí y no forma parte de la estructura tradicional. Y finalmente la carpeta *libs* que es en la que debemos depositar los archivos .jar de las bibliotecas externas que necesitemos usar en nuestro proyecto; según los .jar depositados en esta carpeta se genera la sección *Android Private Libraries* del primer punto.

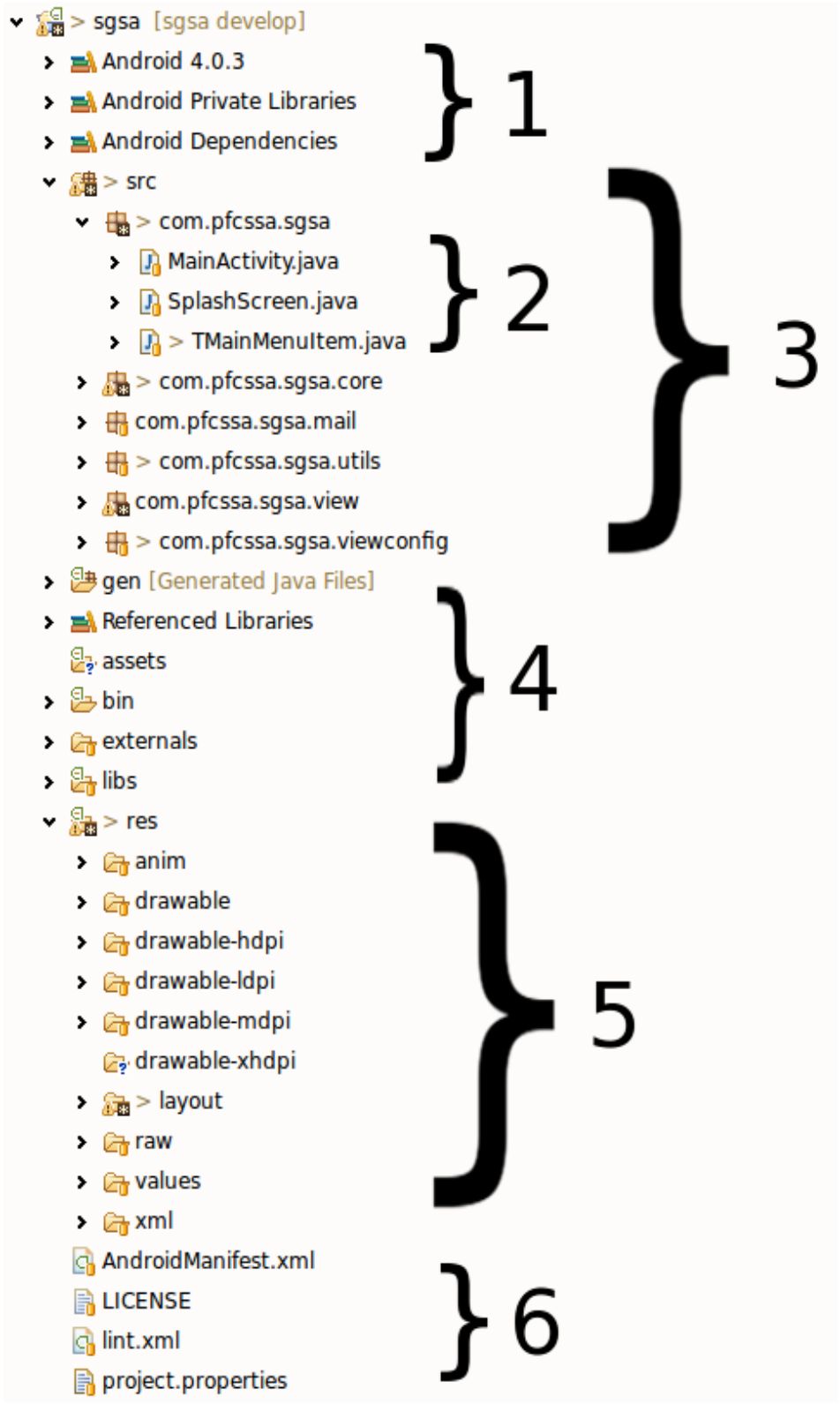


Figura 4.3. Estructura de la aplicación móvil desarrollada en Eclipse.

- *Punto 5:* carpeta de recursos del proyecto. Aquí guardamos todas las imágenes, sonidos, animaciones, textos, diseño de layouts... Se corresponde perfectamente con la parte de la Vista del patrón *Modelo Vista Controlador* (*MVC*). Como explicamos en el capítulo de *recursos* a diferencia de la carpeta *assets* en esta carpeta los recursos se organizan por las categorías soportadas en Android y además podemos crear criterios para diferentes resoluciones o idiomas como se observa por ejemplo con las carpetas *drawable-xxx*.

- *Punto 6:* en este punto encontramos con el archivo más importante dentro de la estructura de un proyecto Android. El archivo *AndroidManifest.xml (*conocido como archivo *manifest).* En este archivo como presentamos en el próximo capítulo se detallan todos los componentes de nuestro proyecto, así como permisos y requerimientos, target mínimo... Después encontramos con los archivos LICENSE que simplemente es un texto con la licencia de nuestro proyecto, obviamente no es un archivo obligatorio. A continuación el archivo lint.xml de la herramienta lint de android que ayuda a evitar bugs a la hora de escribir el código fuente. Y finalmente el archivo *project.properties* que pertenece al propio Eclipse y que lo va a autogenerar éste, a diferencia del archivo *manifest* éste nunca deberíamos editarlo manualmente

A continuación presentamos el archivo *manifest* del proyecto dividido por secciones para poder ir explicando un poco cada parte:

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

En esta primera parte simplemente está definido el *namespace* del archivo XML, la versión y codificación que son comunes a cualquier archivo de tipo XML.

Después ya encontramos con una sección propia de Android donde se especifica el nombre del paquete, la versión del código de la aplicación que es siempre un entero y el *versionName* que podría ser un string. Se entiende que cada vez que actualicemos nuestra aplicación debemos actualizar estos dos valores.

```
<uses-sdk
  android:minSdkVersion="10"
  android:targetSdkVersion="16" />
```

En esta sección se especifica lo que ya explicamos en apartados anteriores, la versión mínima de Android para la cual nuestra aplicación funciona y el target API que se usó para su desarrollo y pruebas.

Se puede observar que nuestra aplicación funciona en teléfonos móviles con una versión de API 10 o superior, es decir igual o mayor que una versión Android 2.3.3. Y el sistema de compatibilidad así como el desarrollo y pruebas de la aplicación se hizo en un teléfono móvil con API 16 (Android 4.1.2).

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.Internet" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission
android:name="com.google.android.gm.permission.READ_CONTENT_PROVIDER"/>
```

En esta parte observamos todos los permisos asociados a funcionalidades que la aplicación usa. Esta lista de permisos son los que en el momento de instalar la aplicación el usuario debe aceptarlos. No hace falta explicarlos ya que se entienden perfectamente y en cualquier caso su uso está definido en la documentación de Android.

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

El tag *uses-feature* se usa para especificar aquellas funcionalidades específicas va a usar nuestra aplicación y va a ser totalmente dependiente de ellas. En este caso nuestra aplicación requiere de teléfonos móviles con cámara con opción de enfoque automático.

```
<application
  android:allowBackup="true"
  android:icon="@drawable/find"
  android:label="@string/app_name" >
  <activity
   android:name=".SplashScreen"
   android:screenOrientation="portrait"
   android:theme="@style/AppThemeFakeMain" >
   <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
  </activity>
```

Aquí se empieza a definir el objeto aplicación así como todas las actividades y diferentes elementos que componen la aplicación en sí misma.

Como podemos observar se especifica un icono y un título para la aplicación. Cabe destacar que el título se puede especificar de forma literal aunque lo recomendado es hacer como se ha hecho en el proyecto y es especificarlo a través de una variable string definida en los *resources* del proyecto (lo estudiamos más adelante).

Además se especifica la actividad principal, que es en nuestro caso el *SplashScreen*. Lo que define que es la actividad principal es el *intent-filter* a través de su *ACTION* y su *CATEGORY*. También podemos especificar el *THEME* por defecto para dicha actividad así como la orientación de la pantalla si la queremos forzar.

```
<activity android:name=".MainActivity" android:label="@string/app_name" />
<activity android:name=".ConfigSigilo" android:label="Configuración Modo Sigilo" />
<activity android:name=".ConfigAlarma" android:label="Configuración Modo Alarma" />
<activity android:name=".Activadores" android:label="Configuración de Activadores" />
<activity android:name=".ModoAlarma" android:label="Modo Alarma" />
<activity android:name=".ModoSigilo" android:label="Modo Sigilo" />
<activity android:name=".ConfigMail" android:label="Configuración correo" />
<activity android:name=".ConfigKeyword" android:label="Configura tu palabra clave" />

<activity android:name=".LogReader" android:screenOrientation="landscape"
android:label="Log del sistema" />

<activity android:name=".SetPassword" android:label="Proteger Sistema"
android:theme="@android:style/Theme.Dialog" />

<activity android:name=".Login" android:label="Acceso"
android:theme="@android:style/Theme.Dialog" />

<activity android:name=".TakePicture" android:label=""
android:screenOrientation="portrait"
```

```
android:theme="@android:style/Theme.Translucent.NoTitleBar"
android:finishOnTaskLaunch="true" android:launchMode="singleInstance" />

<activity android:name=".GestureTester" android:label="Gesture Tester"
android:theme="@android:style/Theme.Dialog" />

<activity android:name=".ConfigGestures" android:label="Seguridad en arranque" />
<activity android:name=".DefineGesture" android:label="Gesture Definer"
android:theme="@android:style/Theme.Dialog" />

<activity android:name=".StartupGestureLogin" android:label="Gesture Login"
android:theme="@android:style/Theme.Translucent.NoTitleBar" />
```

Se puede ver la definición de todas las actividades que conforman la aplicación. En el campo *android:name* simplemente especificamos el nombre del archivo de la clase que representa dicha Actividad. Hay que tener claro que si dicho archivo se encuentra en la misma ruta del paquete definido al principio del archivo XML simplemente vale con poner *.NombreClase* donde el punto no significa que el archivo sea oculto sino que dicho archivo se encuentra en la misma ruta que el paquete que especifica toda la aplicación.

El segundo campo importante es *android:label* donde especificamos el título de la actividad y después tenemos varios campos importantes como *android:screenOrientation* y *android:Theme* que permiten seleccionar una orientación fija por defecto para esa actividad así como un diseño base respectivamente.

Quizá la actividad más compleja en cuanto a su definición en el archivo *manifest* sea la actividad *TakePicture* debido a los siguiente parámetros: *finishOnTaskLaunch* y *launchMode=singleInstance*

Estableciendo el parámetro *launchMode* a *singleInstance* le especificamos al sistema que dicha actividad va a ser la única de su *Task* y que sólo podría haber una instancia de dicha actividad. Y con el parámetro *finishOnTaskLaunch* a *true* le decimos al sistema que en el momento de instanciar nuestra Actividad si existiera una previamente la finalice. Esto último parece redundante si nuestra actividad es de tipo *singleInstance* pero realmente es necesario para liberar correctamente los servicios que hacían uso de la cámara del teléfono móvil.

```
<receiver android:name=".SMSReceiver">
      <intent-filter android:priority="9999">
            <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
      </intent-filter>
</receiver>
```

Esta es la definición del *BroadcastReceiver* de la aplicación el cual se encarga tanto de recibir los mensajes como de comprobar que el teléfono móvil acaba de encenderse. Estas dos acciones se configuran al igual que en la Actividad por defecto con un *Intent-Filter* y las *Actions* correspondientes.

Es importante ver como se establece un campo nuevo de *priority* a un valor de *9999* con la finalidad de que dicho *Receiver* tenga prioridad para estos eventos ante cualquier otro *Receiver* del mismo tipo de sistema.

```
            <service android:enabled="true" android:name=".Servicio"/>
            <service android:enabled="true"
                   android:name=".ServiceGestureLogin"/>
 </application>
</manifest>
```

Finalmente observamos la definición de los dos servicios principales del proyecto donde la única particularidad es que se fuerzan a activarse desde el archivo *manifest*.

Aunque hemos terminado de estudiar el archivo *manifest* del proyecto es recomendable indicar que otros tags podemos encontrarnos en un archivo *manifest:*

- Podemos definir constantes globales al proyecto:

```
<meta-data android:value="valor1" android:name="constante1" />
```

- Proveedores de contenidos de Android:

```
<provider>
   <grant-uri-permission />
   <meta-data />
</provider>
```

- Pantallas soportadas y compatibles por nuestra aplicación:

```
<supports-screens />
<compatible-screens />
```

- Conjunto de características y software que requiere la aplicación (no se suele usar):

```
<uses-configuration />
```

- Permisos y grupos de permisos de la aplicación:

```
<permission />
<permission-tree />
<permission-group />
```

- Instrumentación para testing:

```
<instrumentation />
```

4.4.1 Organización del código en *packages*

Los *packages* permiten organizar el código de una forma libre para el programador (su semejante en C++ son los *namespaces*), tener cierto nivel de seguridad y reutilizar código. En pocas palabras, como su nombre indica un *package* es un paquete que permite agrupar y organizar diferentes elementos con cierta relación entre sí bajo un mismo nombre.

Las ventajas del uso de *packages* se pueden resumir en:

- Agrupar diferentes elementos con característica comunes.
- Organización y limpieza del código.
- Mejor estructura.
- Reutilización del código.
- Cierto grado de seguridad básica.

Un *package* puede contener:

- Clases.
- Interfaces.
- Enumerados.
- Anotaciones.

Para este proyecto de desarrollo hemos decidido crear los *package*s de la Figura 4.4:

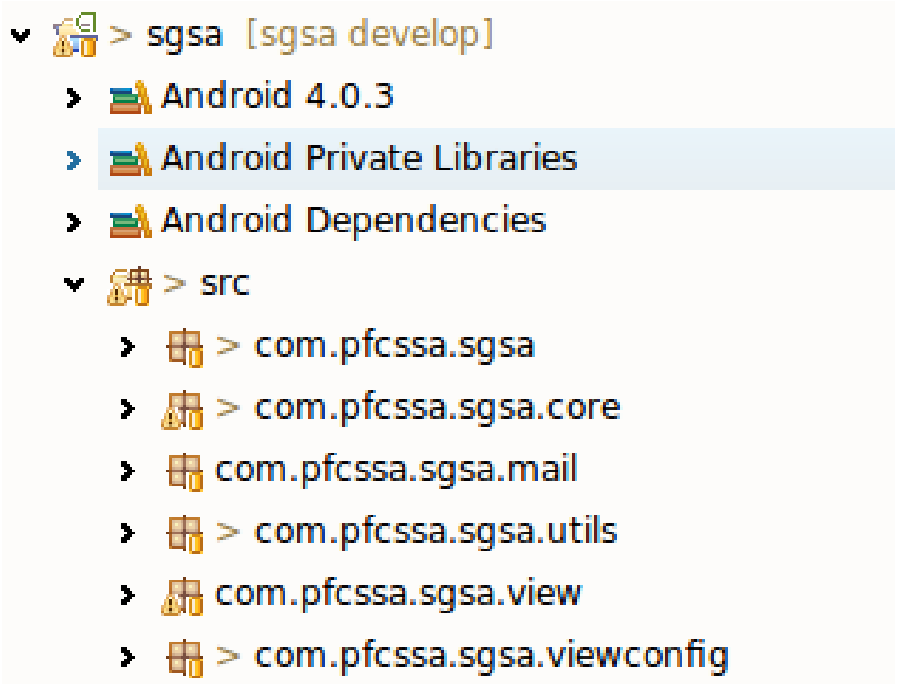


Figura 4.4. Estructura en *packages* de la aplicación móvil comprimida.

- *com.pfcssa.sgsa*: paquete principal que contiene la pantalla de bienvenida y el menú principal de la aplicación.
- *com.pfcssa.sgsa.core*: en este paquete se encuentra todas aquella clases e interfaces relacionadas con el *core* entre ellas el servicio principal.
- *com.pfcssa.sgsa.mail*: las clases relativas al envío de e-mails.
- *com.pfcssa.sgsa.utils*: utilidades comunes a todo el proyecto como el sistema de Log.
- *com.pfcssa.sgsa.view*: todas las actividades y clases relativas a la vista.
- *com.pfcssa.sgsa.viewconfig*: las actividades de configuración especiales.

En la Figura 4.5 presentamos en detalle cada *package* con sus clases y elementos.

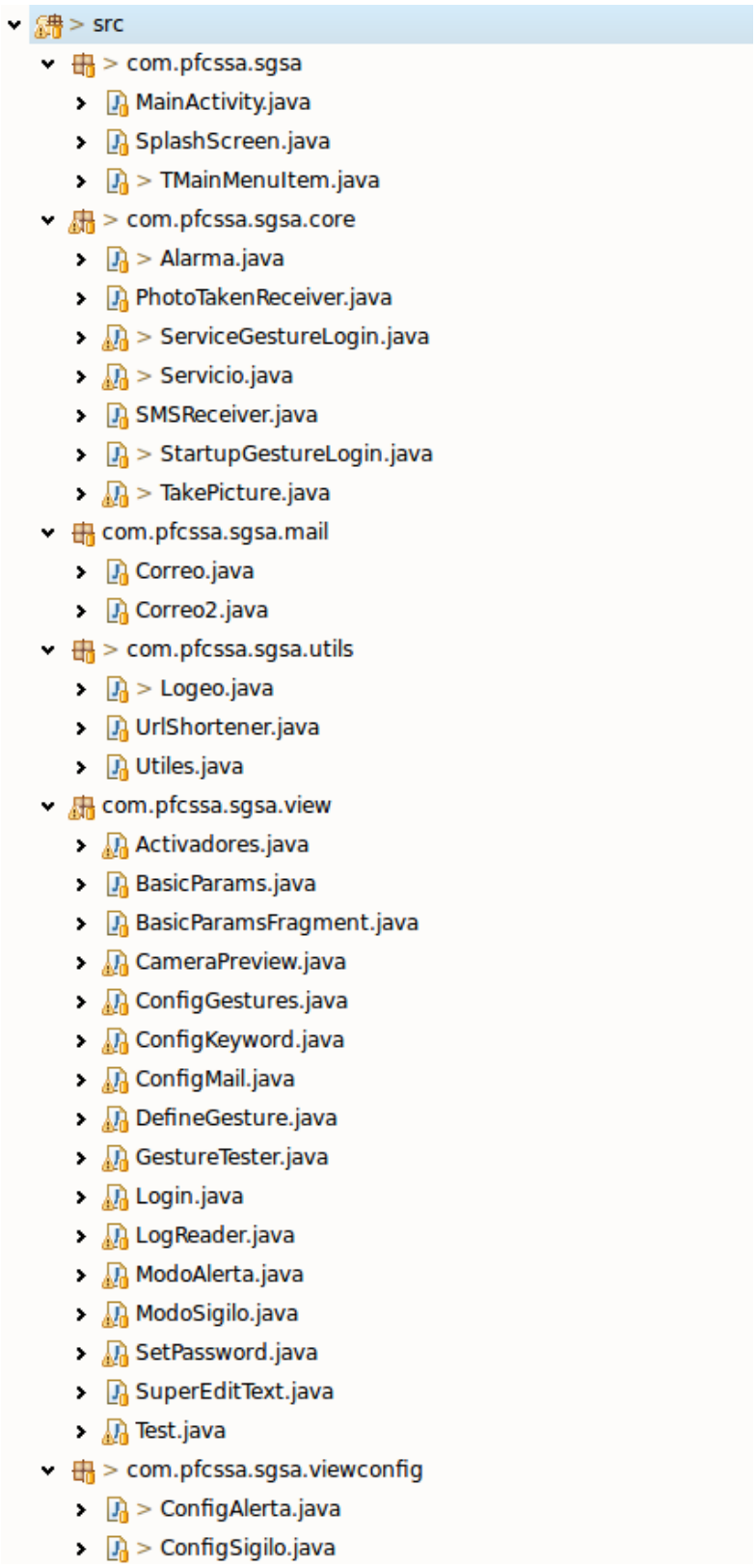


Figura 4.5. Estructura en *packages* de la aplicación móvil expandida.

4.5 Proyección del patrón MVC: el *Núcleo* y la *Vista*

En este proyecto hemos utilizado el patrón MVC aunque con algunas variantes propias del desarrollo en Android. En nuestro proyecto las clases del Servicio Principal, Broadcast Receivers, Módulo de Alarma, Cámara Audio y Email formarían la parte del Modelo, los archivos XML formarían la parte de la vista y finalmente el resto de clases de Vista y Core componen el controlador. Además, el código fuente del proyecto se ha dividido a su vez en dos partes bien diferencias; el *Core* y *la Vista.* Donde simplemente los elementos del *Controlador* pertenecientes más a la parte visual se han dejado en la *Vista* (como son *activities* y *fragments)* y las clases del controlador más relativas al modelo se han dejado dentro del *core.*

A continuación presentamos un análisis usando UML de las clases que representan tanto la vista como el núcleo que fueron explicadas en la sección 4.3.

El *Núcleo*

Podemos observar en la parte superior de la *Figura 4.6,* la cual representa el diagrama en *Unified Modeling Language* (UML) de clases del Núcleo (Core), los tres módulos principales del *Core* que son: *SmsReceiver, Analizador y MainService*. Estos módulos son los principales del *Core* ya que son los que entran en funcionamiento una vez se recibe un *SMS activador* válido.

El primero de ellos es el encargado de escuchar la entrada de mensajes SMS y decidir si es un *SMS activador* válido o no. El segundo como su propio nombre indica analiza todo el mensaje en busca de los posibles parámetros extras recibidos y por lo tanto de los módulos a activar, finalmente llama al servicio principal con los parámetros internos para que ejecute las órdenes recibidas de forma correcta, como puede ser por ejemplo activar los servicios de geolocalización si fuese necesario, hacer sonar la alarma, tomar una fotografía para enviarla al remitente o teléfono especificado…

Como parte del *Core* también se ha incluido el proceso de control de arranque ya que cuando está activado y en caso de fallo de *login* también llamaría directamente al *MainService* del *Core*. Finalmente queda claro que prácticamente la totalidad de las clases del *Core* hacen uso de las clases comunes estáticas de *Logeo* y *Utiles*. (Figura 4.6)

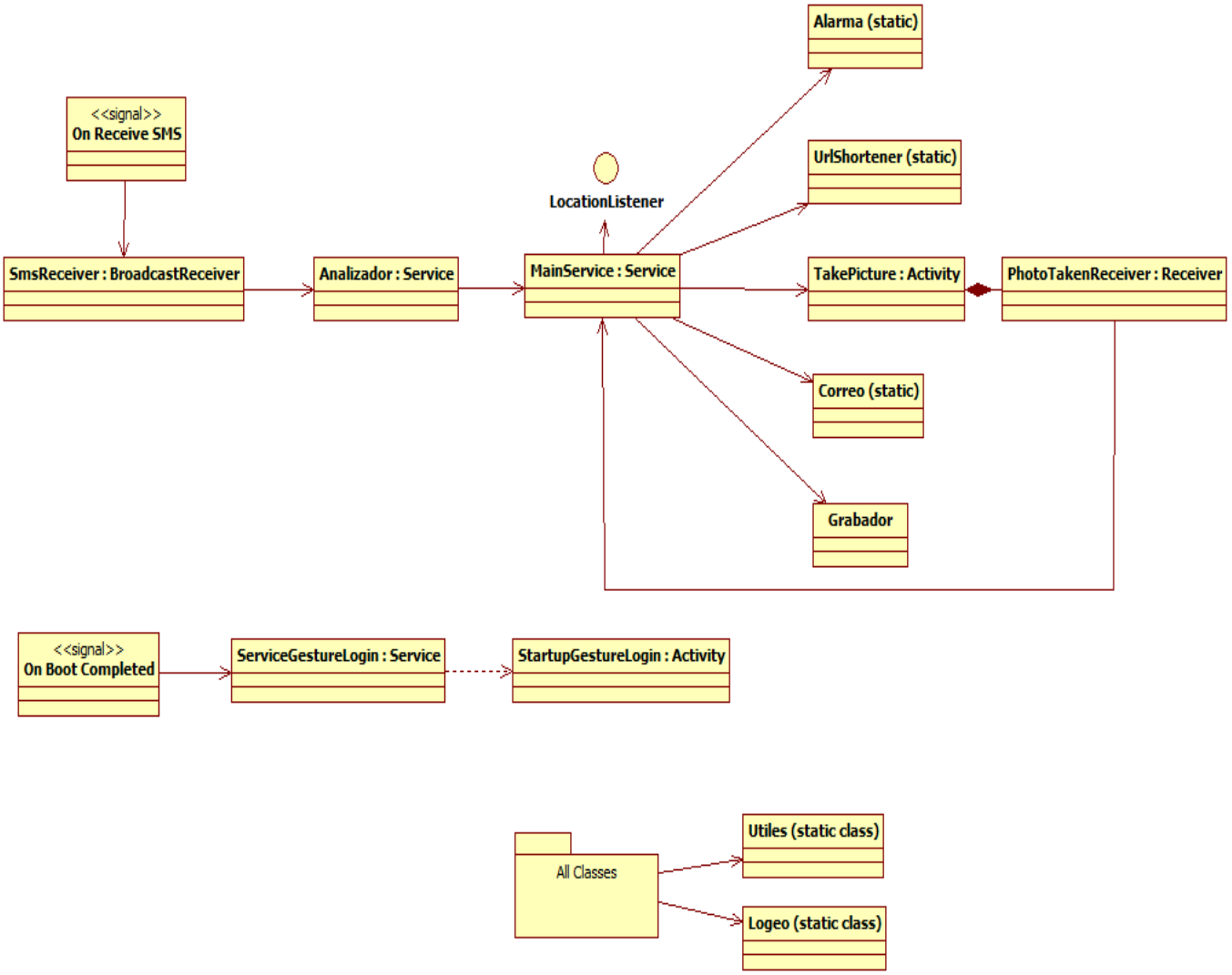


Figura 4.6. Diagrama de clases del Core (interconexión de clases)

Para seguir analizando el comportamiento del *Core* así como ver en detalle cada una de sus clases a continuación presentaremos el resto de diagramas en *Unified Modeling Language* (UML) del Núcleo de la aplicación móvil. En la Figura 4.7 podemos ver el diagrama de estados del Core y en la Figura 4.8 el diagrama de secuencias, ambos en para el caso de recibir un mensaje activador.

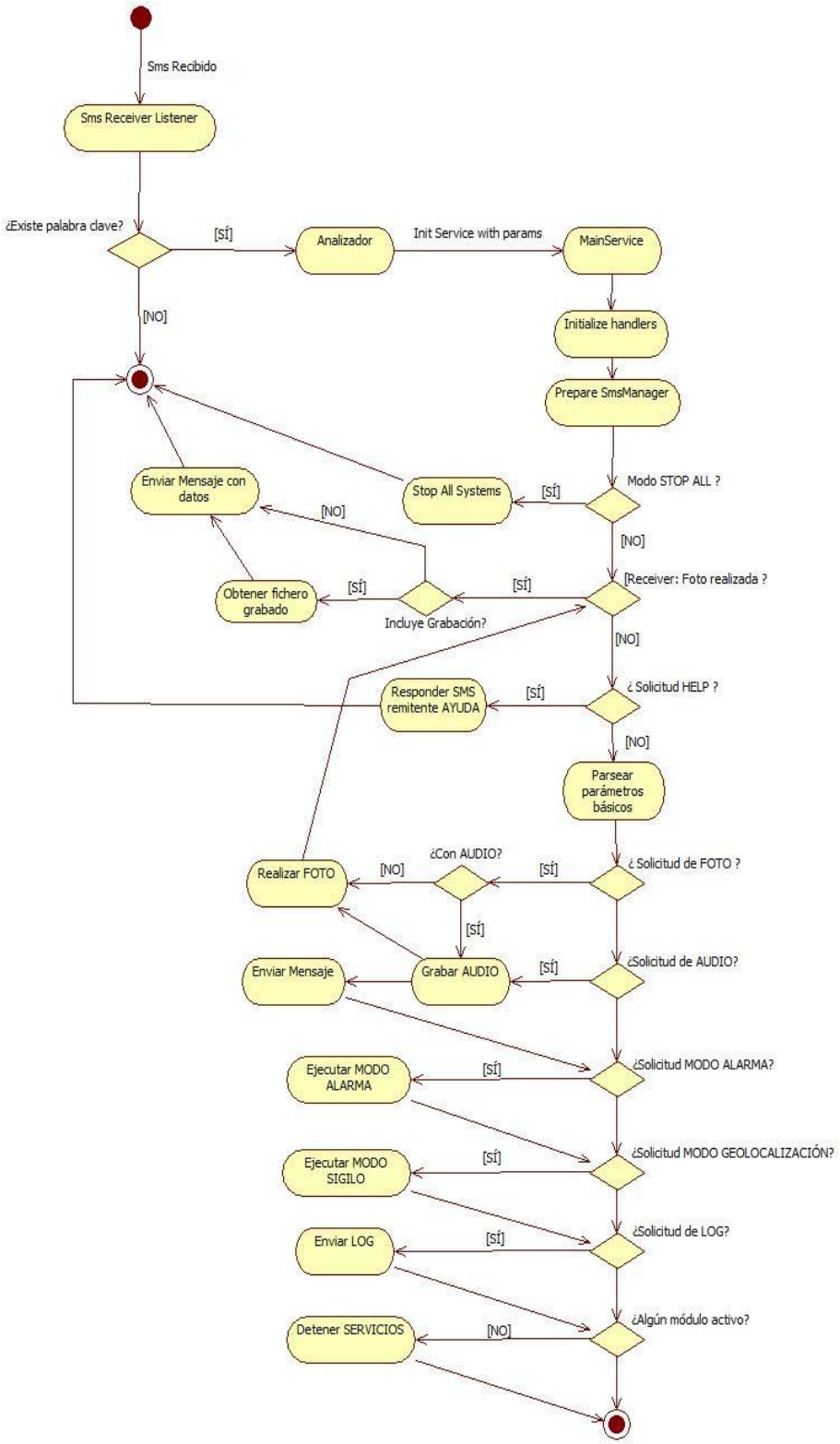


Figura 4.7. Diagrama de Estados del Núcleo (al recibir mensaje activador)

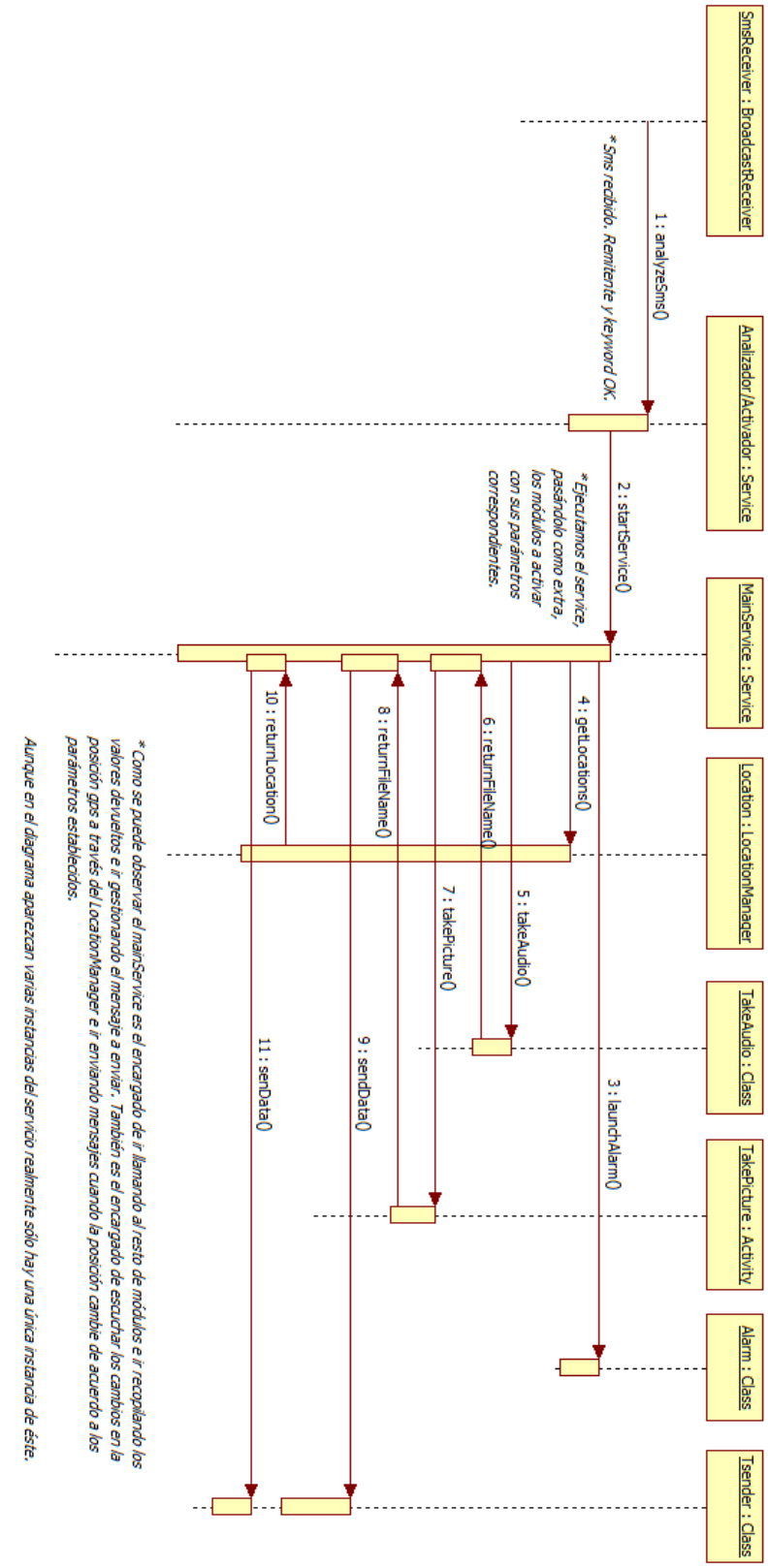


Figura 4.8. Diagrama de Secuencias del Núcleo (al recibir mensaje activador)

A continuación en la Figura 4.9 y en la Figura 4.10 se observan las clases del núcleo con total detalle.

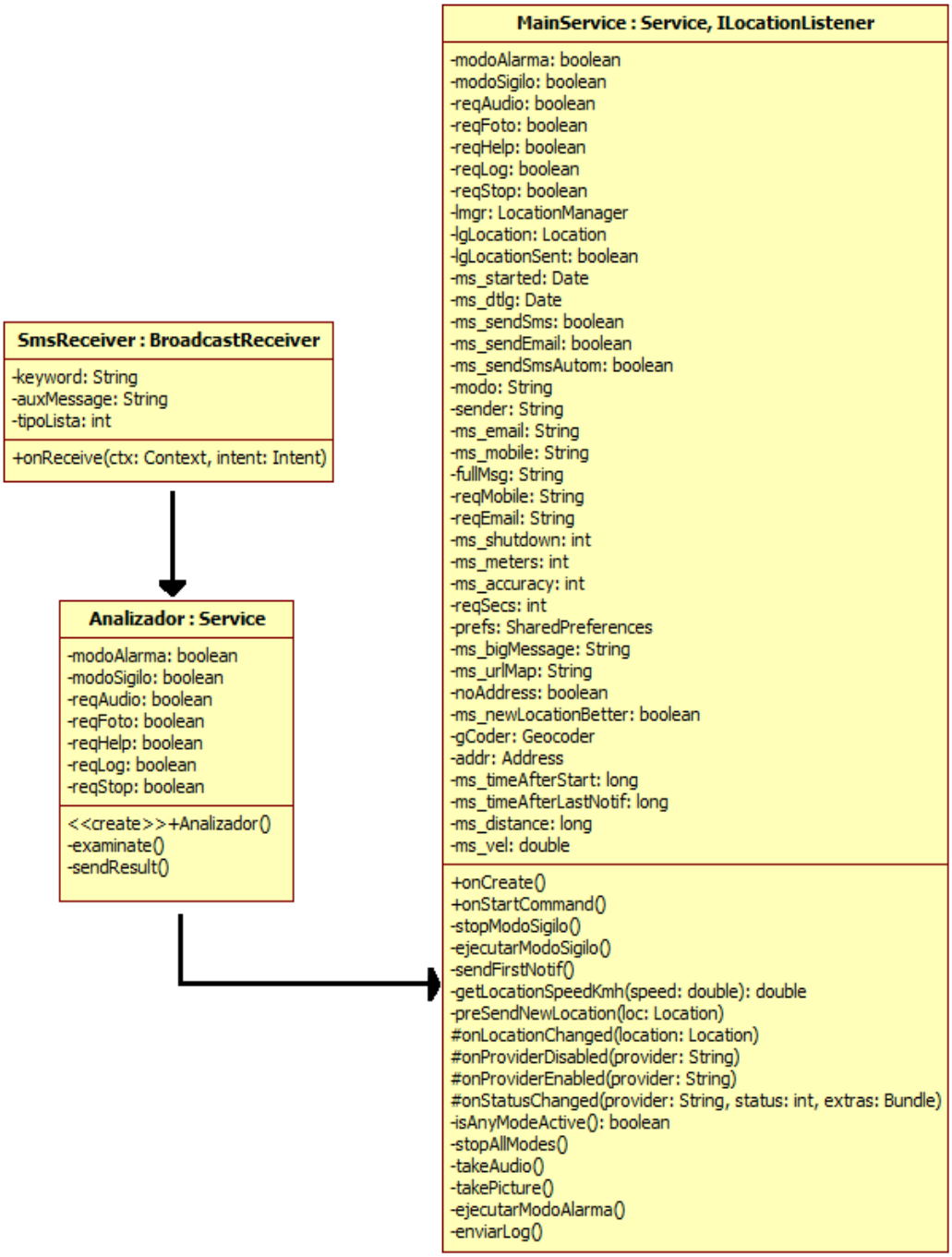


Figura 4.9. Diagrama de clases del Núcleo en detalle.

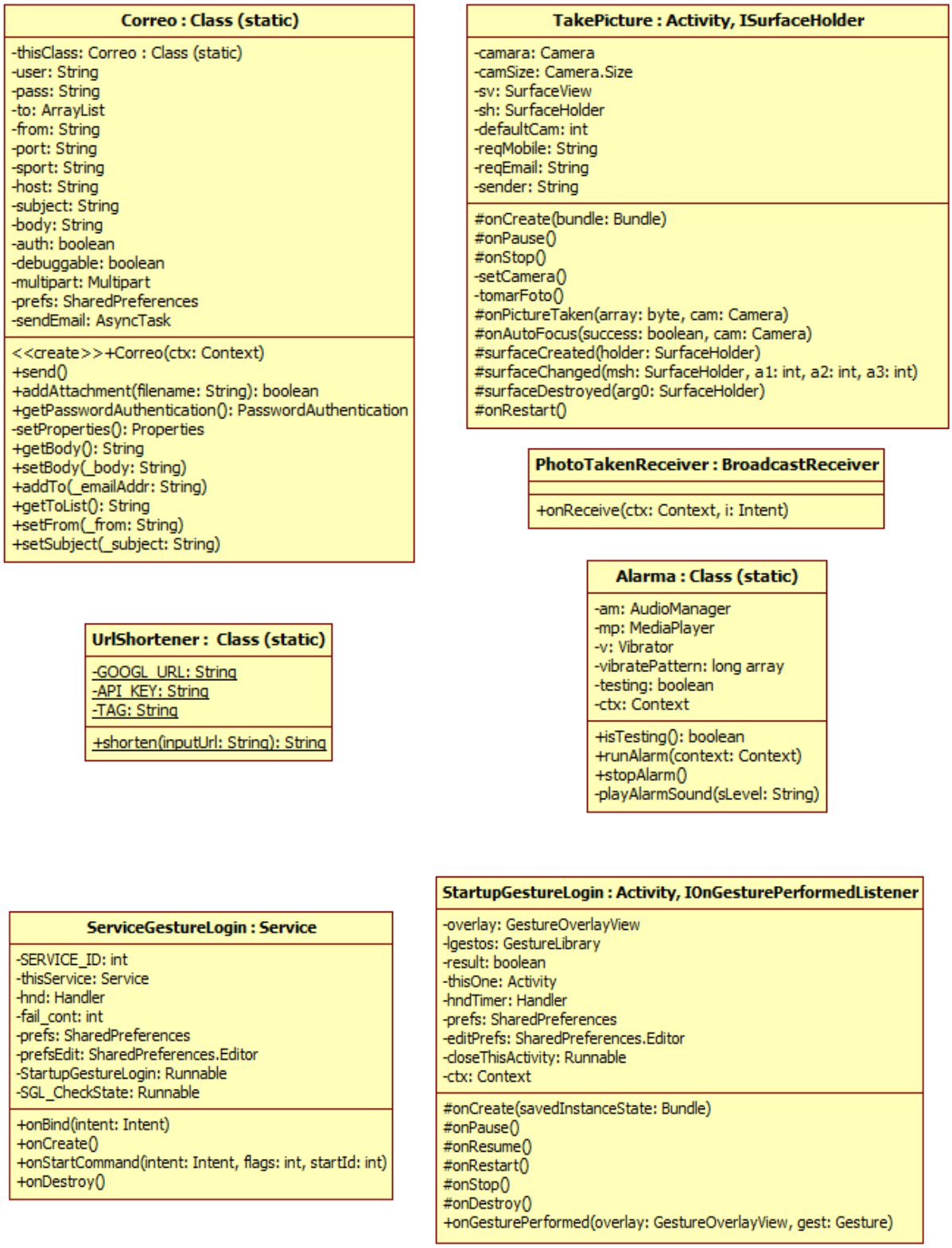


Figura 4.10. Diagrama de clases del Núcleo en detalle.

Antes de pasar a analizar la *Vista* es importante estudiar dos componentes principales de Android que forman parte del Núcleo.

- **Broadcast Receivers:** son un componente Android que permite recibir eventos del sistema o eventos propios. El proceso es bastante sencillo, se crea una clase que extienda de la clase *BroadcastReceiver* y se registra dicho receiver en el archivo *manifest* de la aplicación o en alguna parte del código fuente que queramos (mediante *IntentFilters)* especificando los eventos que queremos escuchar. En el Núcleo de nuestra aplicación tenemos varios eventos *BroadcastReceivers*, los más importante son el *SmsReceiver* y el *OnBootCompletion.* Estos son los encargados de escuchar los mensajes de texto recibidos y avisarnos cuando el terminal se acaba de encender.

  Nuestro *receiver* se registra en el archivo *manifest* a la acción *android.provider.Telephony.SMS_RECEIVED* aunque esto lo presentamos mejor en el código extraído del archivo *manifest* encargado de registrar nuestro *receiver:*

```
<receiver android:name=".core.SMSReceiver" >
 <intent-filter android:priority="9999" >
  <action android:name="android.provider.Telephony.SMS_RECEIVED" />
  <action android:name="android.intent.action.BOOT_COMPLETED" />
 </intent-filter>
</receiver>
```

  Es importante recalcar varios puntos importantes del código anterior; Lo primero es que estamos registrando tanto a la acción de recibir mensajes de texto como a la acción *Boot Completed* que hace referencia al arranque del teléfono móvil, que es usada para decidir si debemos lanzar o no el sistema *Control de Arranque*. Usamos el mismo receiver ya que ahorramos una clase y ambos eventos los registramos en el archivo *manifest.* El otro punto importante a recalcar es el hecho de la prioridad ya que permite escuchar el evento antes que otros *listeners* y poder abortar la propagación del evento si lo deseamos. *Esto último es crucial para ocultar el SMS recibido* al resto de aplicaciones.

- **Services:** son un componente de aplicación Android importante en el Núcleo de nuestro proyecto, probablemente el más importante. Los servicios permiten ejecutar operaciones en segundo plano. Los servicios son perfectos y están

diseñados para realizar operaciones largas o pesadas en segundo plano, aunque se pueden utilizar con cualquier propósito. En nuestro proyecto la clase *MainService* implementa nuestro servicio principal, dicho servicio como ya sabemos es el encargado de analizar el resultado de un *SMS Activador*, y activar/ejecutar los diferentes módulos necesarias además de gestionar los servicios de geolocalización.

La *Vista*

En el diagrama de interconexión de clases de la Vista *(Figura 4.11)* podemos observar el proceso para cargar la *MainActivity* después del SplashScreen así como la *MainActivity* es la encargada de lanzar el resto de formularios y/o ventanas (*Activities* en Android).

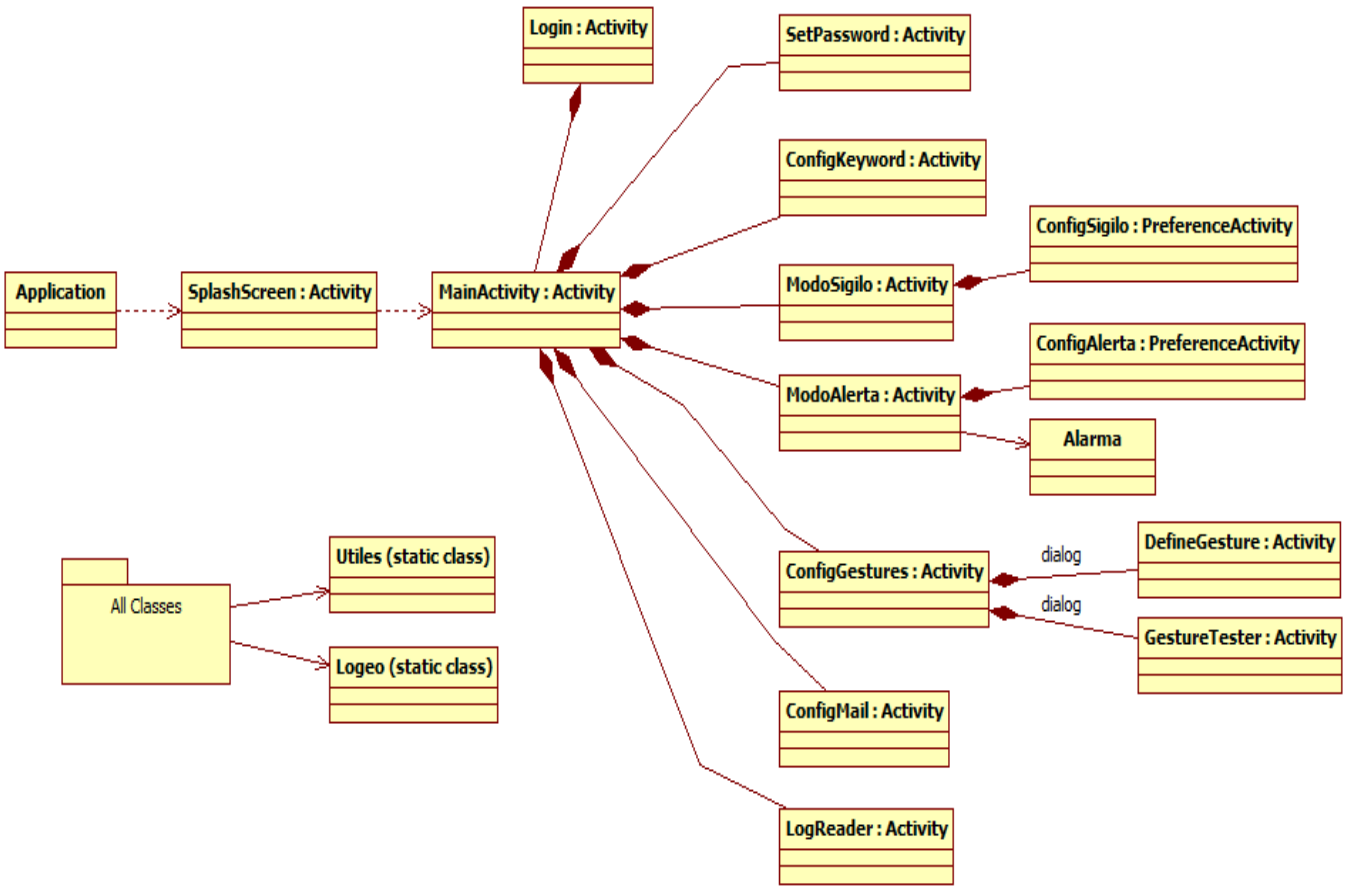


Figura 4.11. Diagrama de clases de la Vista (interconexión de clases)

Una parte importante es darse cuenta que todas las clases que representan una ventana o formulario en Android van a descender de la super clase de Android *Activity* o *Fragment* ya que éstas son las encargadas en el sistema Android de representar tales recursos. Estos dos componentes Android son muy importantes y podemos conocer su funcionamiento en nuestro proyecto en profundidad en el Anexo I.5

Después de ver el diagrama de Clases de la Vista ahora vamos a ver en la Figura 4.12 el diagrama UML de clases pero con total detalle de cada clase.

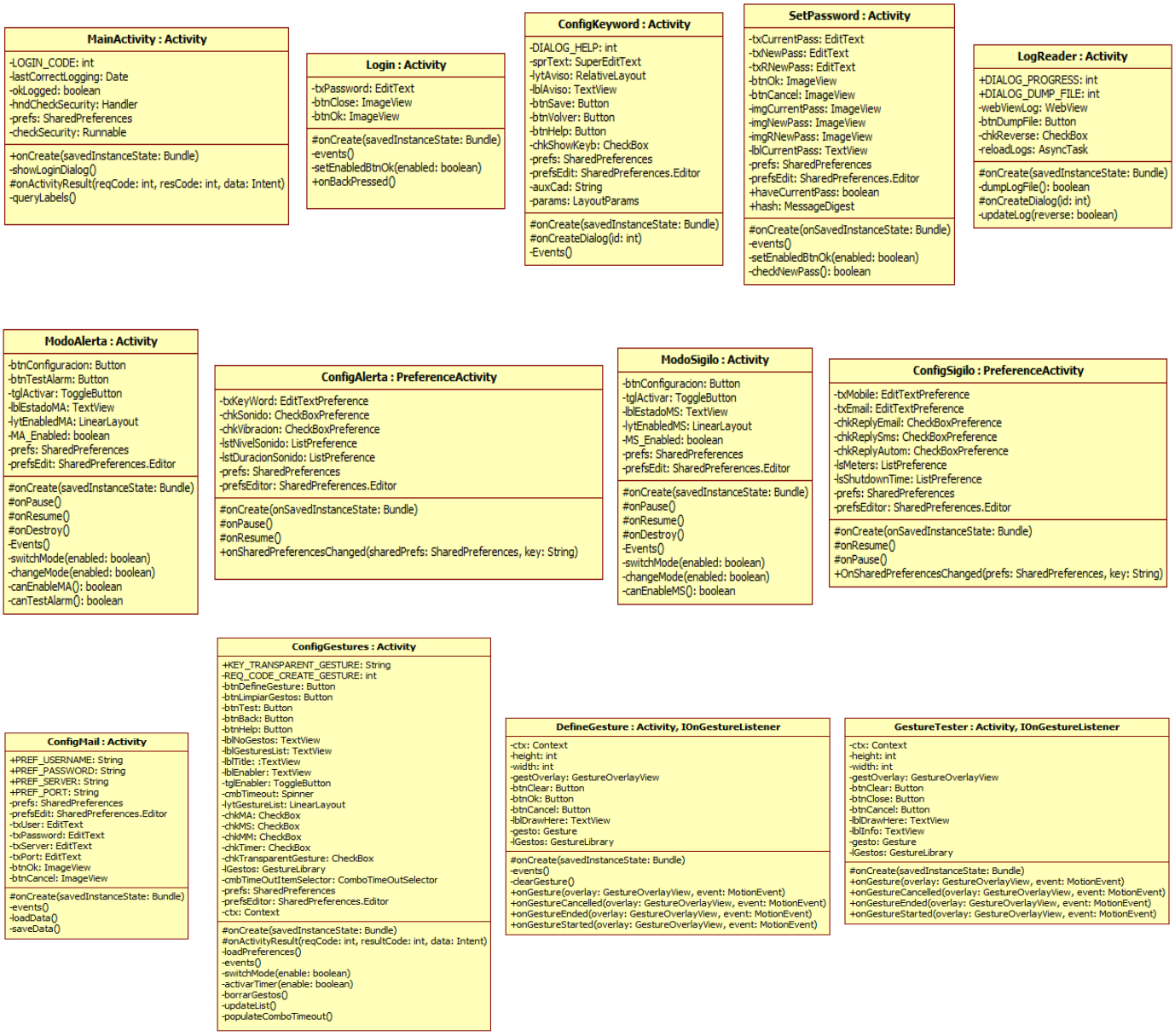


Figura 4.12. Diagrama de clases de la Vista en detalle.

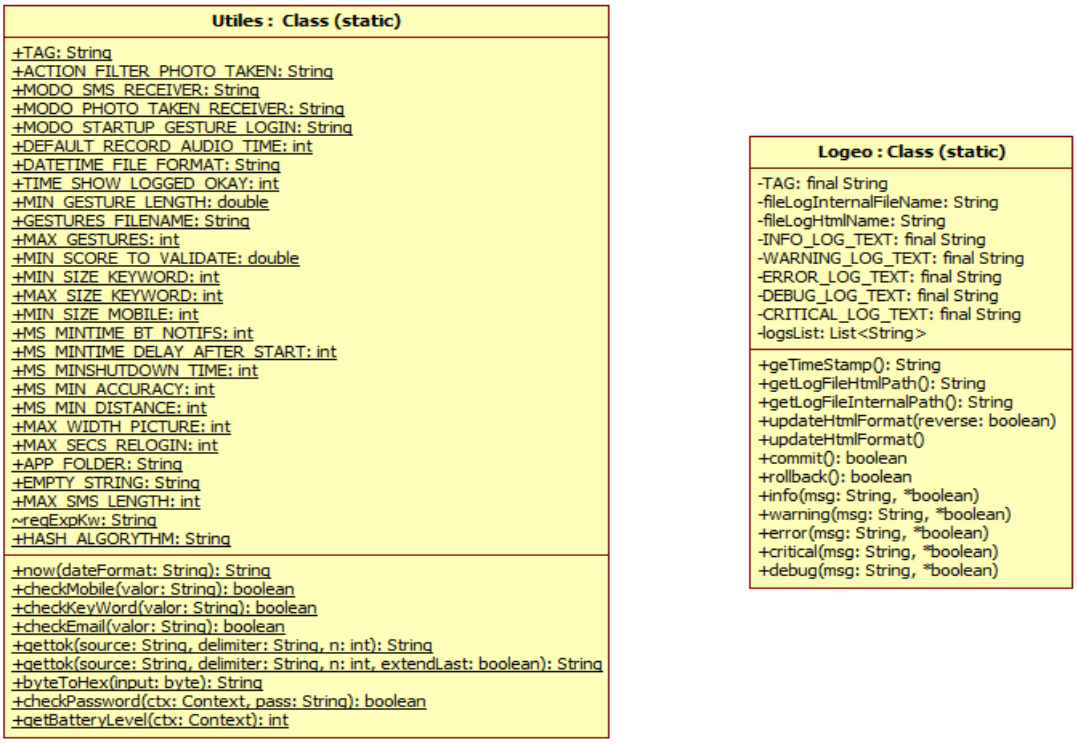


Figura 4.13. Diagrama de las clases comunes *Utiles* y *Logeo*.

Al igual que en las clases del Core prácticamente la totalidad de clases de la vista hacen uso de las clases comunes estáticas *Utiles* y *Logeo* (Figura 4.13).

4.6 Estructura Git del proyecto

Una vez conocido el comportamiento de las clases y los patrones de diseño que hemos empleado, en este apartado presentamos algunos detalles adicionales sobre aspectos concretos de implantación del software en Eclipse adaptado por Google para Android y sobre su control de versiones usando Git.

Como repositorio (servidor) remoto Git hemos usado el que brinda el servicio de Bitbucket [12] ya que permite tener de forma gratuita varios proyectos privados siempre que no superen un mínimo de colaboradores en el proyecto y en este caso al sólo haber un único colaborador (desarrollador) no hemos tenido ningún problema. (Figura 4.14)

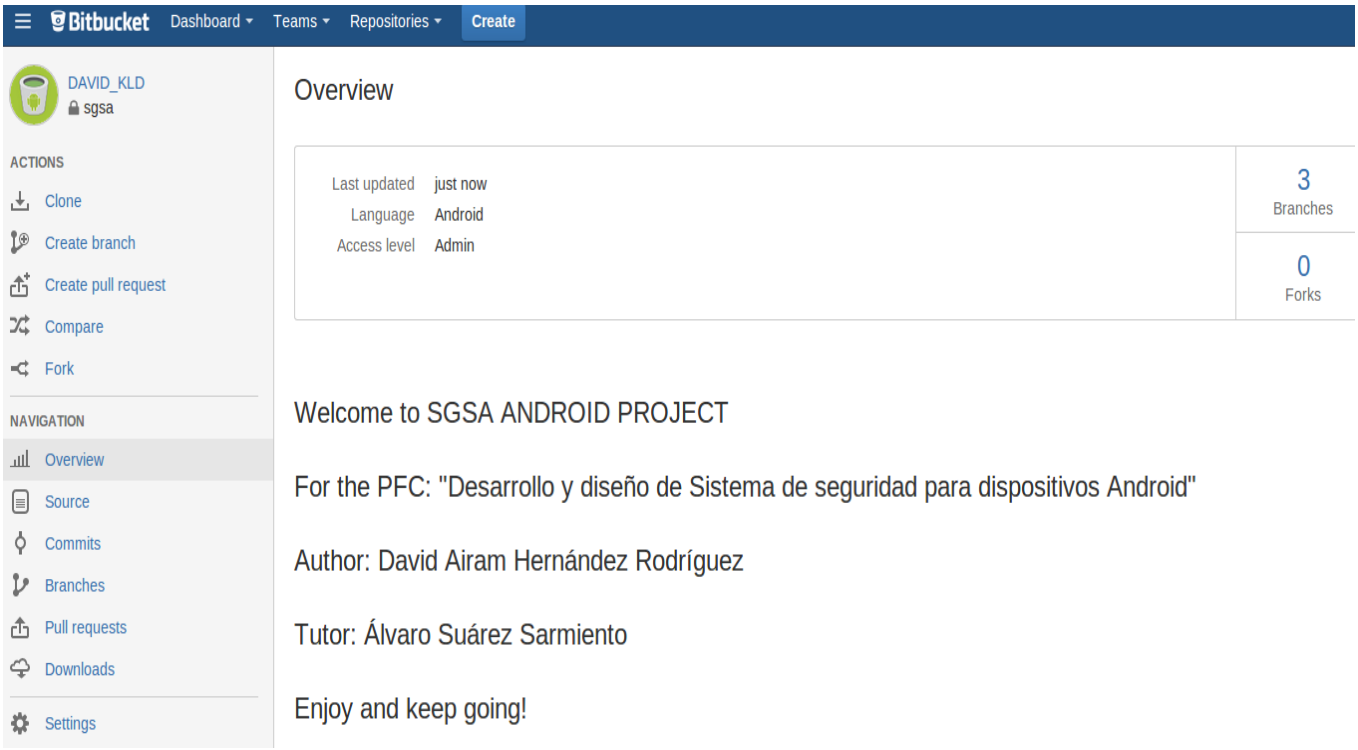


Figura 4.14. Bitbucket, página de inicio al PFC.

Para nuestro proyecto hemos creado 3 ramas básicas: ***master, develop y prepro.*** Obviamente durante el proceso de creación del proyecto hemos ido creando otras ramas que normalmente partían y parten de *develop* para funcionalidades nuevas, solucionar errores...

La rama master la usamos como rama final de *producción* ya que es la rama por defecto en Git. *Develop*, como su nombre indica la usábamos como rama principal de trabajo y desarrollo continuo. Es esta rama la que recibía principalmente todos los cambios y *commits* del desarrollo del proyecto.

Finalmente cuando había una versión que se podía cerrar se pasaba a *prepro* para probarla. Si había algunos fallos se solucionaban y se volvía a probar hasta que finalmente se pasaba a producción en *master*... Cuando se iba a pasar (*mergear)* una nueva versión a la rama master se creaba una nueva rama con el nombre de la versión anterior que partía de master, para así poder tener el histórico de todas las versiones realizadas. (Figura 4.15)

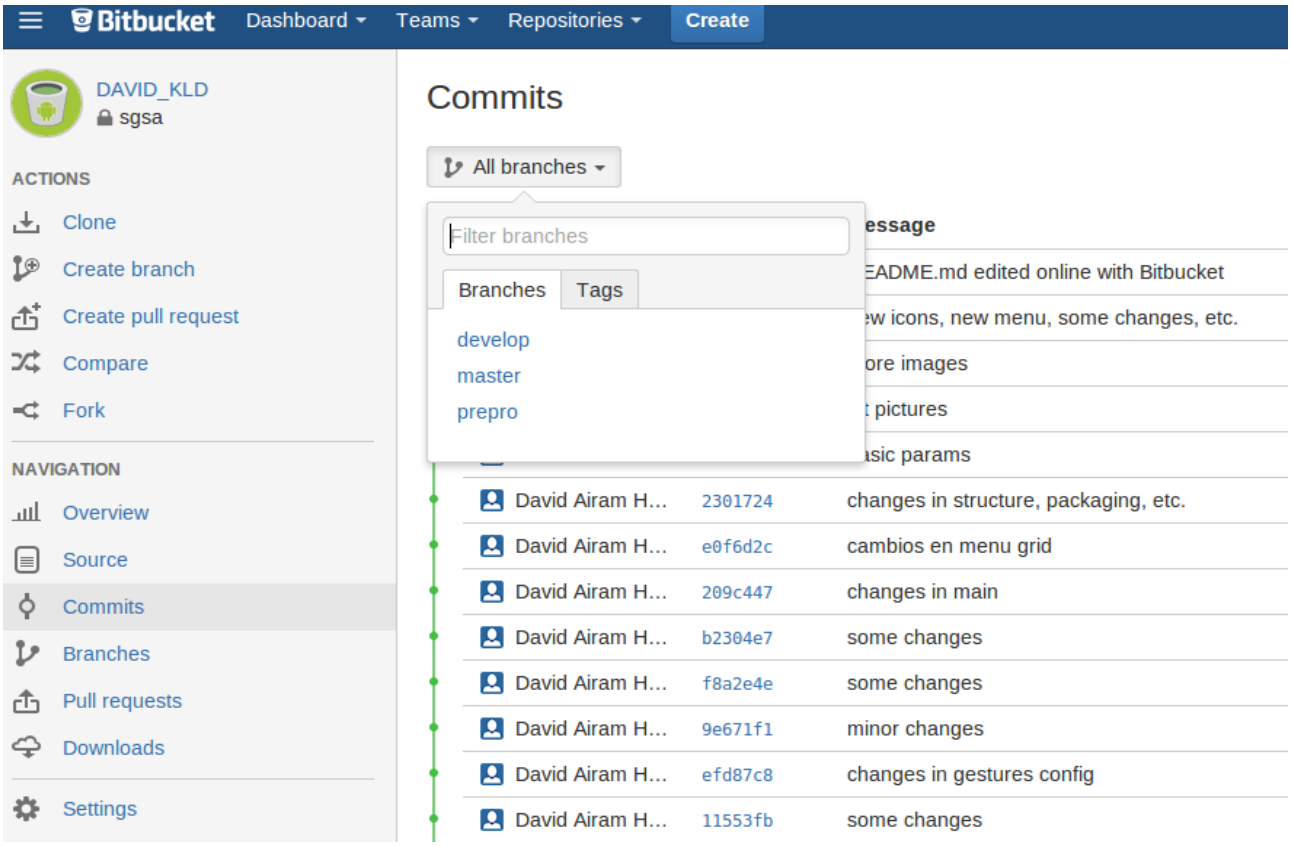


Figura 4.15. Bitbucket, vista de los últimos *commits* y selección de ramas.

5. Análisis de la interfaz de usuario

En este capítulo finalmente analizamos el funcionamiento de la aplicación de cara al usuario final. Mostramos la mayoría de las pantallas por las que está formada la aplicación. Además centramos principalmente en su facilidad de uso y lo hacemos de una forma ordenada para que el usuario pueda apreciar y entender poco a poco las diferentes opciones a configurar y probablemente hacerse una idea mejor y más visual de lo que es este PFC.

Y para finalizar en el último apartado de este capítulo presentamos todos los parámetros posibles del sistema, realizamos una batería de pruebas y presentamos los resultados obtenidos en dichas pruebas.

5.1 Descripción de la aplicación móvil

Figura 5.1. Icono de la aplicación instalada en Android.

Una vez instalada la aplicación encontramos el acceso a la aplicación (*SGSA*) con el icono que representa el logo de la *Universidad de Las Palmas de Gran Canaria* (*ULPGC*) en blanco sobre un fondo azul marino con forma circular (Figura 5.1).

Cuando se ejecuta la aplicación observamos de forma breve una pantalla de carga o *Splash Screen* (el archivo de código fuente que hace referencia a esta pantalla se llama *SplashScreen.java*). En esta pantalla simplemente observamos el nombre de la aplicación así como los autores del PFC y el logo de la universidad en la parte inferior. (Figura 5.2)

Figura 5.2. *SplashScreen* de la aplicación.

Pantalla Principal

Una vez se termina de ejecutar la aplicación encontramos con la pantalla principal (*MainActivity.java* en el código fuente). Esta pantalla principal muestra todas las opciones disponibles en una cuadrícula invisible (Figura 5.3).

Esta forma de presentación de opciones no es la nativa de Android pero se ha usado ya que es mucho más fácil e intuitiva. Además como podemos observar en la *Figura 5.3*, de esta manera podemos ver el icono del módulo a configurar, su nombre y una breve descripción que serviría de ayuda al usuario.

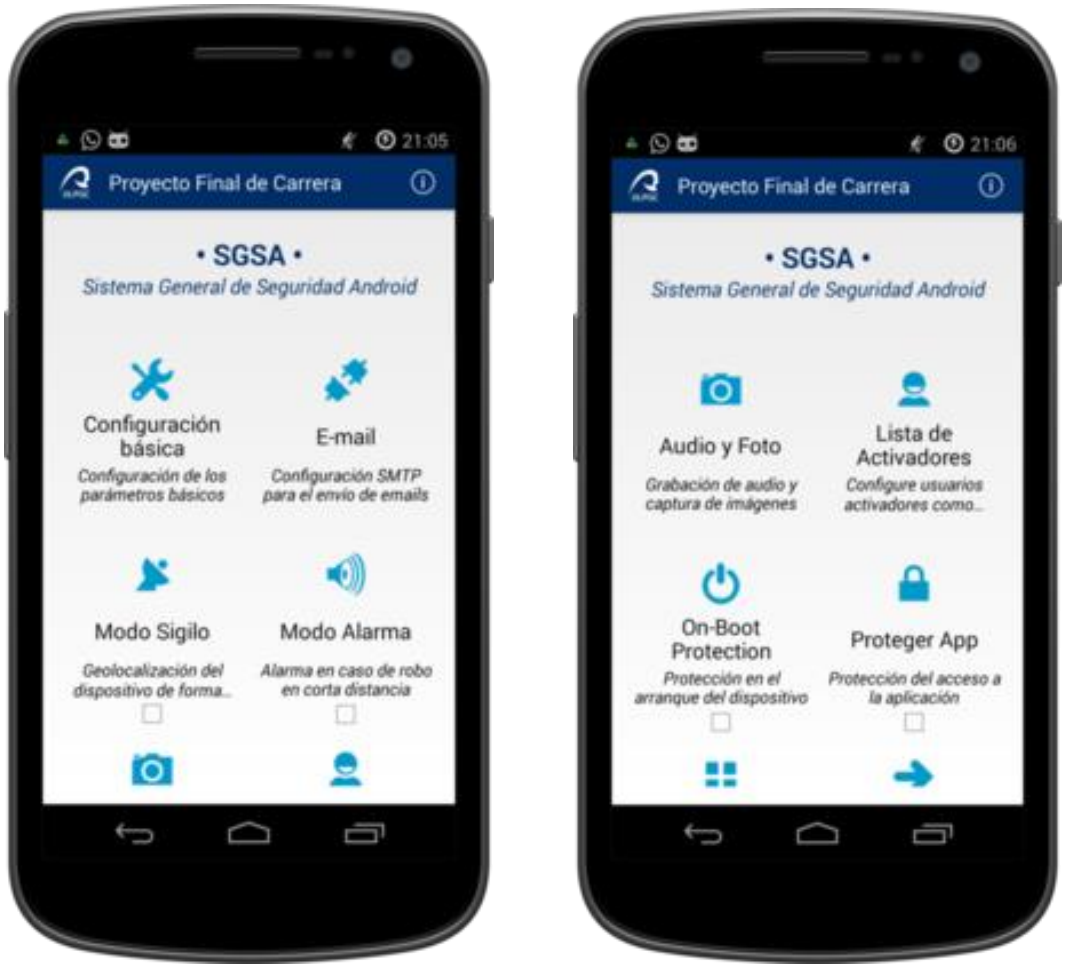


Figura 5.3. Menú principal de la aplicación.

Configuración Básica

Esta opción es la primera y probablemente la más importante ya que permite definir los tres parámetros básicos para que el sistema empiece a funcionar:

- *Palabra clave o Keyword*: esta palabra clave va a ser la que una vez recibida mediante SMS en el teléfono móvil active los sistemas solicitados con los parámetros extra o si no se especifica ninguno active por defecto el *Modo Alarma.*
- *Teléfono alternativo*: el número de teléfono que especifiquemos aquí va a servir como número de respuesta alternativo en caso de que sea diferente al número de teléfono del remitente del mensaje *activador.* Aquí podemos configurar por ejemplo el teléfono de nuestra pareja o persona de confianza de forma que en caso de robo o pérdida poder notificarle también a esta persona la ubicación o los datos capturados con el teléfono robado.
- *E-mail alternativo*: al igual que el teléfono por defecto este parámetro permite añadir de forma opcional un e-mail alternativo que va a servir para enviar los datos recogidos por el sistema. El hecho de que en el mismo SMS activador se puede pasar como parámetro una dirección de correo electrónico no anula esta dirección que si fuera diferente a la pasada por parámetro se notifica también a ésta dirección.

A diferencia del primer parámetro, cuyo uso es obligatorio, los dos siguientes son opcionales pero obviamente es altamente recomendable tenerlos configurados por si olvidamos de pasarle el parámetro en el SMS Activador ya que en ese caso sólo tendríamos una respuesta vía SMS al móvil remitente si es que hemos marcado esta opción en el módulo correspondiente el cual presentamos más adelante. Además si solicitamos servicios donde la respuesta sólo funciona por correo electrónico y además de no pasar ningún parámetro extra de correo electrónico tampoco tenemos ninguno configurado en esta sección de configuración básica el sistema no podrá enviar los datos que haya capturado al usuario. En definitiva aunque los parámetros alternativos no son bloqueantes sí que es muy recomendable configurarlos correctamente ya sea en este módulo en el módulo correspondiente (Figura 5.4).

Figura 5.4. Configuración de parámetros básicos.

Configuración E-mail

Este apartado obviamente también es muy importante ya que aunque la configuración es totalmente opcional sí que es muy recomendable activarlo ya que sin esta sección configurada el sistema *no va a ser capaz de enviar correos electrónicos de forma transparente al usuario afectado* (Figura 5.5).

Como observamos en la *Figura 5.5* los parámetros a configurar en este apartado son los básicos para la configuración de una cuenta de correo usando el protocolo SMTP: nombre de usuario, contraseña, servidor SMTP y puerto por defecto. Aunque pueda parecer algo complejo realmente no lo es y la aplicación trae los parámetros de servidor y puerto configurados con los valores de Google, así que si usamos una cuenta Google sólo le tendríamos que dar un nombre de usuario@gmail.com con su contraseña.

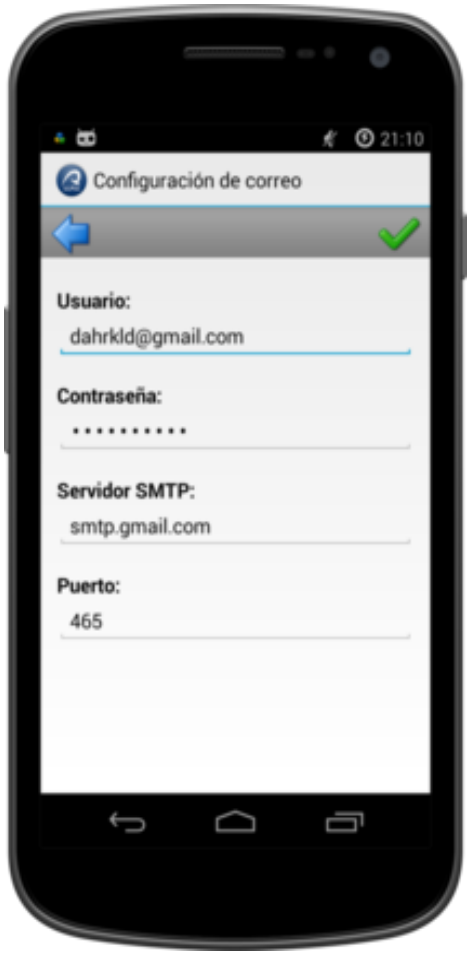
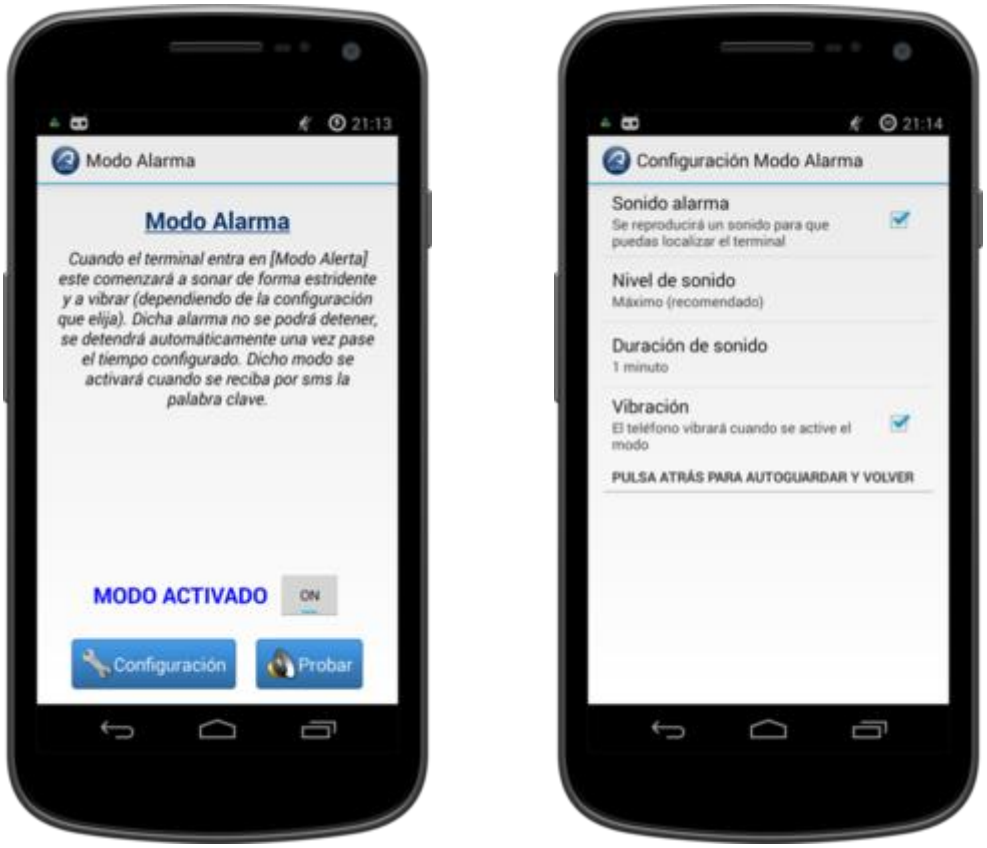


Figura 5.5. Configuración del servidor SMTP para el envío de correo electrónico.

Finalmente para guardar los datos introducidos tocamos en la flecha verde que se encuentra en la esquina superior derecha (Figura 5.5) ya que si tocamos la flecha azul de la esquina superior izquierda los datos introducidos no quedan registrados.

Modo Alarma

Este modo implanta una alarma acústica estridente que el usuario del teléfono no podría detener ni bajar de volumen hasta que termine la duración previamente configurada en esta misma sección.



(a) (b)

Figura 5.6. (a) Modo Alarma, pantalla principal. (b) Pantalla de configuración.

Como observamos en la *Figura 5.6.a* en esta pantalla presentamos una breve descripción del modo en la parte superior y en la parte inferior activamos y desactivamos el módulo así como configurarlo y probarlo. Cabe destacar que no lo probamos o activamos hasta que no lo hayamos configurado correctamente.

Cuando tocamos el botón *Configuración* de la parte inferior se presentan las opciones que podemos configurar las cuales podemos observarlas en la *Figura 5.6.b.* En la pantalla de configuración del modo propiamente dicho (Figura *5.6.b*) podemos observar cuatro parámetros muy fáciles de configurar. Aunque ningún parámetro es obligatorio obviamente si no activamos la opción *Alarma Sonora* ni la opción *Vibración* no podemos activar ni probar el modo.

Los otros dos parámetros que figuran en el medio son referentes como se puede observar a la *Alama Sonora* y permiten configurar tanto el nivel de sonido entre cuatro

posibles así como la duración de éste (Figura *5.6.b*). Recordar que una vez se active la alarma vía *SMS activador* no es posible detener la alarma hasta que se cumpla el tiempo configurado o bien se envíe otro *SMS activador* con el parámetro *stop.*

Modo de Geolocalización

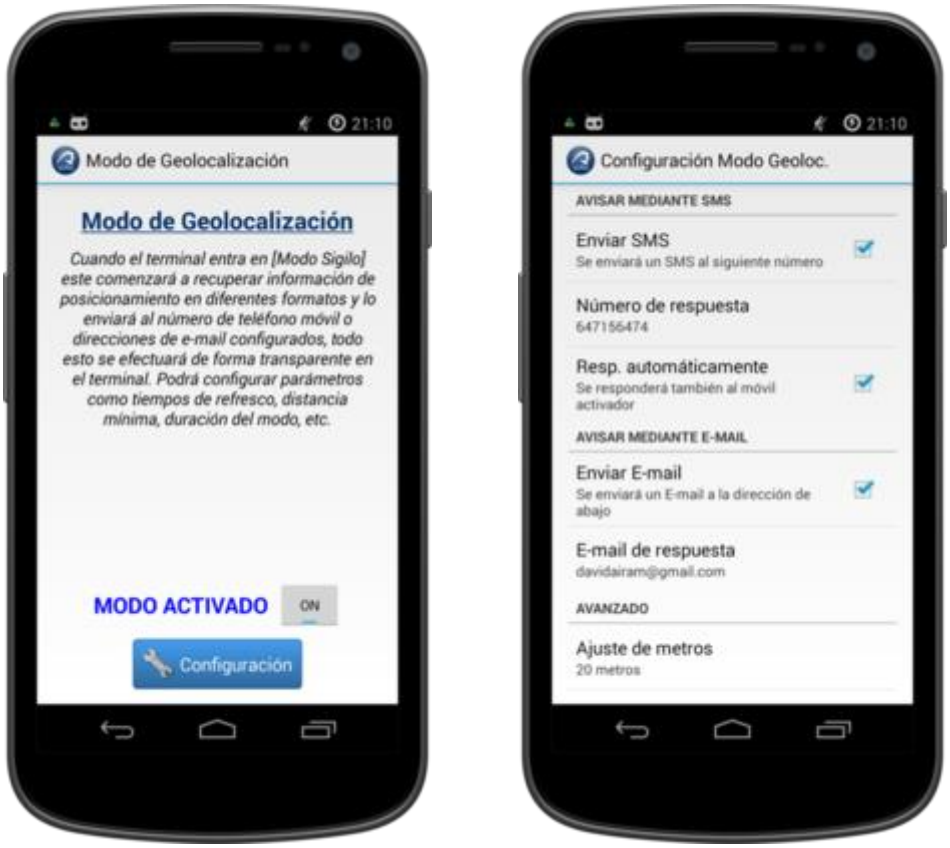
Al igual que en el modo de *Alarma* cuando accedemos a este modo lo primero que observamos es una pantalla donde se presenta un resumen de su funcionamiento así como los posibles parámetros que acepta el modo. En la parte inferior en este caso simplemente observamos las opciones para activar/desactivar el modo y el botón de *Configuración* ya que este módulo no acepta una prueba de su funcionamiento como el módulo anterior(Figura 5.7.a).

Una vez accedemos a las opciones de configuración de este modo (Figura 5.7.b) encontramos con varias opciones donde las más importantes son:

- *Responder automáticamente*: se responde de forma automática al teléfono del remitente (el que envió el SMS Activador).

- *Ajuste de metros*: hace referencia a la distancia mínima que el sujeto (*smartphone* perdido o robado) se tiene que desplazar sobre la última posición notificada al usuario para que el sistema decida notificarnos nuevamente. En otras palabras, el sistema considera una nueva ubicación cuando la posición varíe la cantidad de metros indicada y por lo tanto el sistema notifica de nuevo al usuario con dicha ubicación.

- *Tiempo de apagado*: hace referencia al tiempo que el sistema está activo esperando y escuchando la posición del teléfono móvil desde que se notificó por primera vez al usuario en la sesión. Es recomendable no configurar esta opción con un tiempo muy elevado ya que podríamos mermar rápidamente la batería del teléfono móvil perdido o robado lo que podría derivar en su auto-apagado.



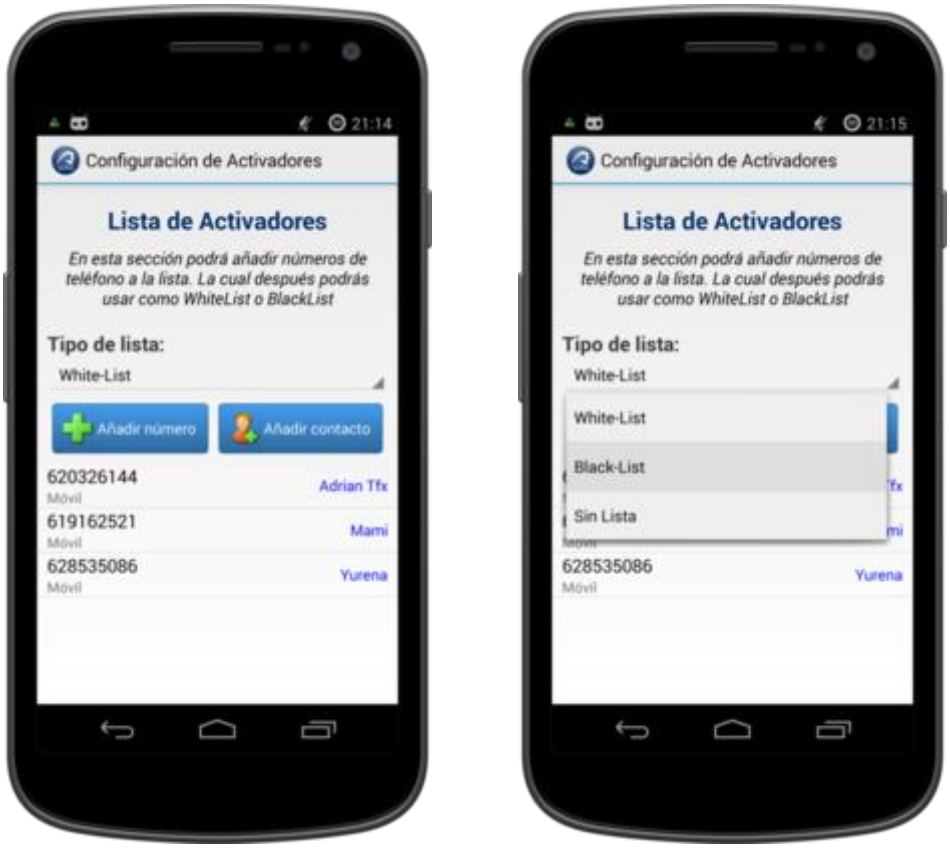
(a) (b)

Figura 5.7. (a) Modo Geolocalización pantalla principal. (b) Pantalla de configuración.

Lista de Activadores

La *Lista de Activadores* es una característica de la aplicación de la cual no se ha hablado mucho hasta ahora pero realmente es una funcionalidad bastante especial ya que va a ayudar a evitar activaciones involuntarias (Figura 5.8.a).

Para ello permite definir una lista de números de teléfonos tanto como *Whitelist* o bien como *Blacklist* (Figura 5.8.b).



(a) (b)

Figura 5.8. (a) Lista de Activadores. (b) Opción de tipo de lista.

- *Blacklist*: esta configuración va a *bloquear* el acceso como usuarios activadores a los números de teléfono que se encuentran en la lista. Esta función es útil para evitar que ciertos contactos (números de teléfono) activen una alarma o el sistema de geolocalización de forma remota indebidamente.

- *Whitelist*: en contraposición con la *Blacklist,* la *Whitelist* sólo y exclusivamente permite que envíen mensajes de activación los usuarios que se encuentran en la lista. Esta opción aunque es verdaderamente útil hay que usarla con mucho cuidado ya que en caso de hurto o pérdida solo los usuarios con el de esta lista van a poder activar el sistema remotamente, con lo que puede ser un problema si no se configura correctamente.

- *Sin lista*: finalmente, esta configuración permite acceso al control remoto a cualquier teléfono que conozco nuestra *keyword o palabra clave.*

Probablemente sea la opción más recomendada al comenzar a utilizar el sistema, donde la seguridad del sistema frente a activaciones no deseadas recae en la complejidad de la *keyword.*

Para terminar y como se observa en la Figura 5.8 comentar que podemos confeccionar la lista de usuarios seleccionando los contactos desde nuestro teléfono o bien añadiendo directamente el número de teléfono manualmente. Tanto si lo añadimos desde los contactos como si no, si dicho número de teléfono figura entre nuestros contactos en la lista aparece el nombre de este para ayudarnos a identificar el número.

Control de Arranque

El *Control de Arranque* es otro de los módulos de protección más importantes del sistema ya que gracias a este módulo evitamos encendidos del teléfono móvil (cambios de SIM) no autorizados (Figura 5.9).

Para ello y como se observa en la Figura 5.9.a el módulo permite definir hasta un máximo de tres gestos en pantalla. Estos tres gestos pueden ser parecidos o muy diferentes, pero lo ideal y por temas de seguridad es que los tres gestos sean el mismo pero dibujados de diferente forma con el fin de garantizar la seguridad del teléfono móvil (Figura 5.9.b).

Aunque sólo es obligatorio definir como mínimo un gesto de los tres que permite el sistema es muy recomendable definir los tres gestos (de forma parecida). Es importante porque cuando definimos un gesto en el momento de hacer el *login gráfico* el gesto sólo se da por válido si se realiza en el mismo sentido y dirección que se realizó el original. Es decir, si creamos como gestor un triángulo y lo definimos de izquierda a derecha empezando por el vértice inferior izquierdo, cuando lo reproduzcamos lo tenemos que dibujar de la misma forma ya que si realizamos la misma figura pero en sentido contrario, aunque el resultado visual sea el mismo el gesto no es válido y obtenemos un error de autenticación interno. En definitiva aunque queda a decisión del usuario como definir los tres gestos, la recomendación es definir el mismo gesto dibujado de diferentes formas o en diferentes tamaños.

(a) (b)

Figura 5.9. Control de Arranque. (a) Gestos parecidos. (b) Gestos diferentes.

En cualquier caso y como observamos en la Figura 5.10 siempre probamos el sistema de autenticación desde el botón *Probar* situado a la izquierda debajo de la lista de gestos, de esta forma podemos probar libremente y todas las veces que deseamos el conjunto de gestos definido.

Una vez definidos y probados los gestos encontramos con varias opciones de cara al sistema de autenticación real, que se ejecuta después del arranque del teléfono móvil si este módulo está activo.

(a) (b)

Figura 5.10. Control de Arranque. (a) Creando un gesto. (b) Probando el gesto creado.

Destacar que cuando se solicita al usuario la autenticación real lo hace de forma totalmente transparente y el usuario sólo sabe que se le está pidiendo autenticación por un *beep* de 1,5 segundos de duración que suena después del arranque del teléfono móvil (tanto si el teléfono está o no en silencio) con lo que en el momento del arranque el usuario debe de estar atento para introducir el gesto correcto y no generar una activación no deseada.

Siguiendo con las opciones disponibles en este modo, ahora encontramos con la opción de poder definir el gesto totalmente transparente o no en el momento de realizar la autenticación real. Puede ser conveniente ponerlo transparente para que un amigo o usuario que esté cerca no vea nuestro gesto en pantalla, dando la sensación de que simplemente hemos intentado realizar un movimiento por la pantalla.

A continuación encontramos con los diferentes modos a activar si falla el proceso de autenticación. Podemos activar tanto el *Modo Alarma, Modo de Geolocalización* o *realizar una fotografía*. Podemos combinarlos como queramos. Destacar que la fotografía puede ser una buena opción ya que es un proceso transparente y es muy probable que precisamente en este momento el usuario infractor esté mirando a la pantalla del teléfono siendo un momento ideal para que la cámara frontal lo capte y el sistema envíe dicha captura.

Debemos saber que en caso de que el sistema de autenticación falle no se notifica de ninguna forma al usuario y simplemente se ejecutan los modos seleccionados. Sólo habría notificación en caso de una autenticación positiva.

Finalmente encontramos con la opción de poder definir un *tiempo máximo de respuesta* para el sistema de autenticación. Esta opción es bastante útil ya que muchos usuarios después de introducir el *Personal Identification Number* (PIN) de la tarjeta SIM simplemente apagan la pantalla o no hacen nada con el teléfono móvil, con lo que si no definimos un tiempo máximo de respuesta no recibimos la notificación hasta que el usuario falle el gesto cuando desbloquee de nuevo el teléfono móvil. Por esto es bastante recomendable definir un tiempo máximo de respuesta, la opción recomendada sería de 3 minutos ya que daría a nosotros como poseedores del teléfono móvil un margen de respuesta por si olvidamos de introducir el gesto y escuchamos el *beep* en la distancia.

A continuación en la Figura 5.11 puede observar el módulo de Control de Arranque en funcionamiento. En este caso no hemos activado la opción de "gesto transparente" para poder ver nuestro *gesture login* en pantalla. Una vez el gesto es detectado correctamente vemos el mensaje de *Acceso Permitido*. Recordar que en caso de fallo no hay aviso, simplemente se desencadenan los procesos previamente configurados en el módulo.

(a) (b)

Figura 5.11. Control de Arranque. (a) Haciendo *login* con gesto visible. (b) Acceso concedido.

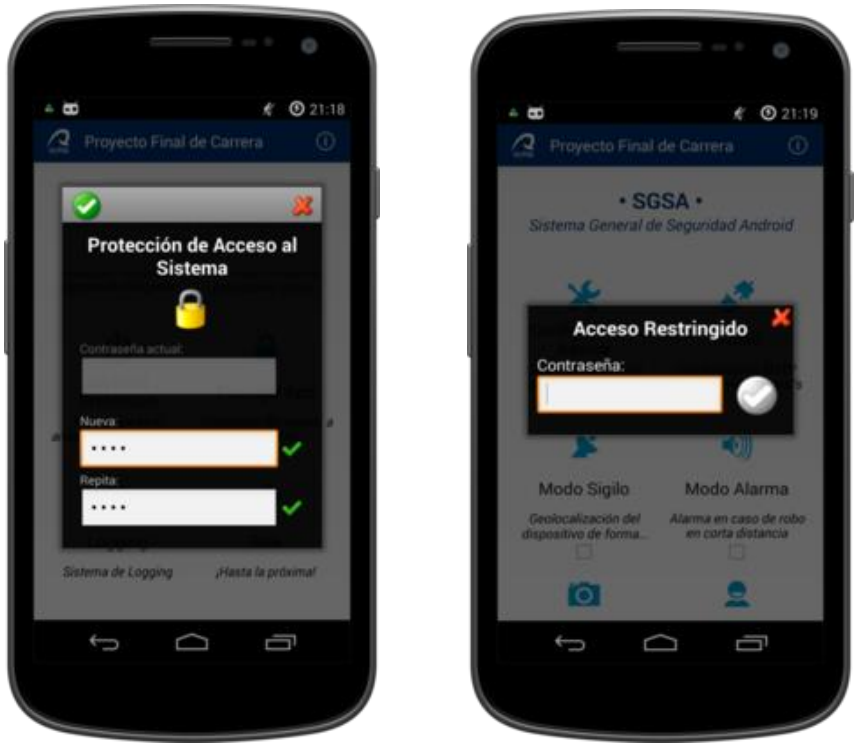
Protección de la Aplicación

Este módulo es el encargado de proteger el acceso indebido a la aplicación en sí misma. Para ello es tan sencillo como definir una contraseña y confirmarla (Figura 5.12.a).

Una vez definida la contraseña y por lo tanto activada la protección de la aplicación cada vez que ejecutemos la aplicación se solicita la contraseña. También se vuelve a solicitar en caso de que sigamos dentro de la aplicación y hayan pasado más de 2 minutos con el fin de evitar que un simple acceso correcto deje la aplicación desbloqueada hasta su cierre (Figura 5.12.b).

Si introducimos mal la contraseña o pretendemos cerrar la ventana de *login* sin haberlo realizado correctamente la aplicación se cierra inmediatamente

Para desactivar el sistema de protección de la aplicación simplemente volvemos a activar este módulo, se solicita la contraseña actual y después simplemente dejamos en blanco los campos de definición de la nueva contraseña y le damos de nuevo al botón de confirmar, en este momento se informa de que se ha desactivado la protección.



(a) (b)

Figura 5.12. Protección del sistema. (a) Definiendo una clave de protección y activando el módulo. (b) Solicitud de contraseña.

Sistema de Registro

Finalmente encontramos con el sistema de registro total del sistema. Este módulo es el encargado de registrar todo los procesos que realice el sistema. Con lo que podemos ver cuando se recibe un mensaje activador, quien lo envía, los procesos que se desencadenan posteriormente, y en definitiva todo un registro de lo que acontece en el sistema (Figura 5.13).

El registro es bastante útil ya que como usuario permite conocer si algo ha fallado y el motivo del fallo. Es importante saber que en caso de pérdida o hurto podemos enviar un mensaje activador con el parámetro *L* lo que indica al sistema que envíe el archivo de log al correo electrónico definido en el mismo mensaje y/o al correo alternativo configurado en los parámetros básicos de la aplicación.

Como se puede observar en la Figura 5.13 la pantalla se habría configurado en horizontal y en la parte superior tenemos tres opciones: activar/desactivar orden cronológico, actualizar la lista y finalmente borrar todo el registro (Figura 5.14).

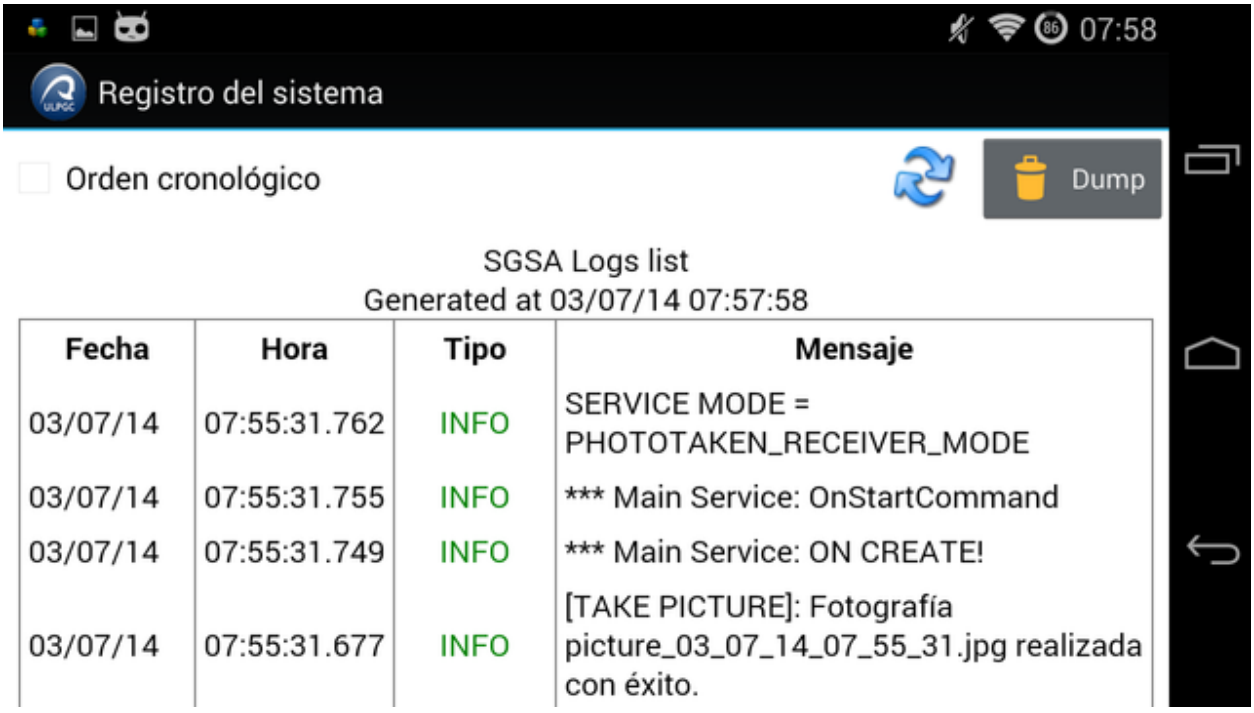


Figura 5.13. Definiendo una clave y activando el módulo. Solicitud de contraseña.

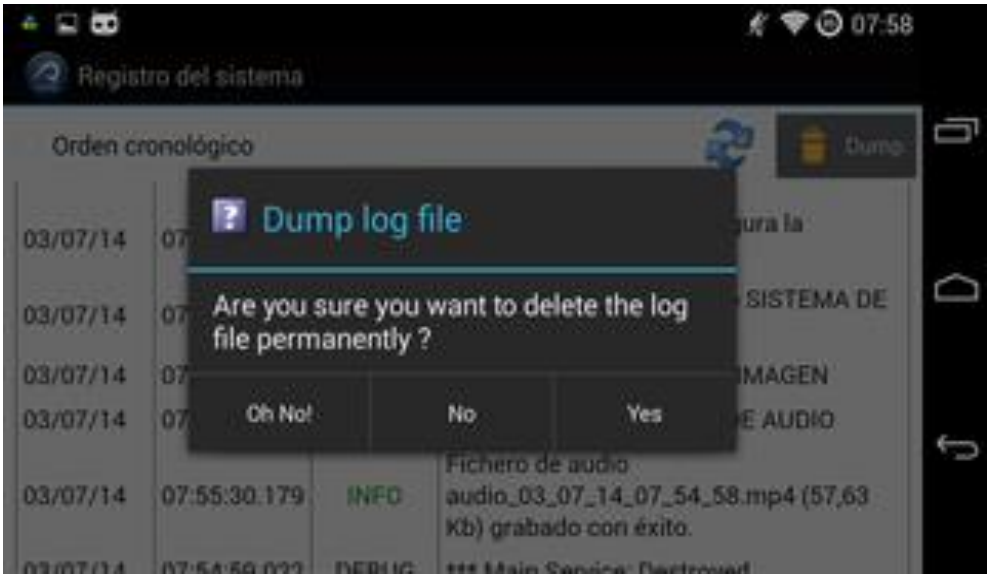


Figura 5.14. Registro del sistema. Borrar todo el fichero de registro.

Como se muestra en la Figura 5.13 el registro se divide en cuatro columnas: fecha, hora, tipo y mensaje. En cuanto al tipo, el sistema de registro cuenta con cuatro tipos definidos: DEBUG, *INFORMACIÓN (INFO), ADVERTENCIA (WARNING) y ERROR.* Cada tipo lleva un color asociado: negro, verde, azul y rojo respectivamente, con lo que es más fácil detectar los diferentes tipos de mensaje de un simple vistazo. Cabe decir que cuando recibimos el registro por correo electrónico el formato y el código de colores se mantiene ayudándonos así a leerlo de la misma forma que en la aplicación del teléfono móvil.

Acerca de…

Finalmente y para terminar con el análisis de la interfaz gráfica de la aplicación, si tocamos en el icono de información en la esquina superior derecha de la pantalla principal, accedemos a todos los datos de este Proyecto Final de Carrera. Además tenemos la opción de poder desplazarnos verticalmente haciendo un desplazamiento (*scrolling)* para poder ver más información tanto del proyecto como de la aplicación móvil que se ha desarrollado. (Figura 5.15).

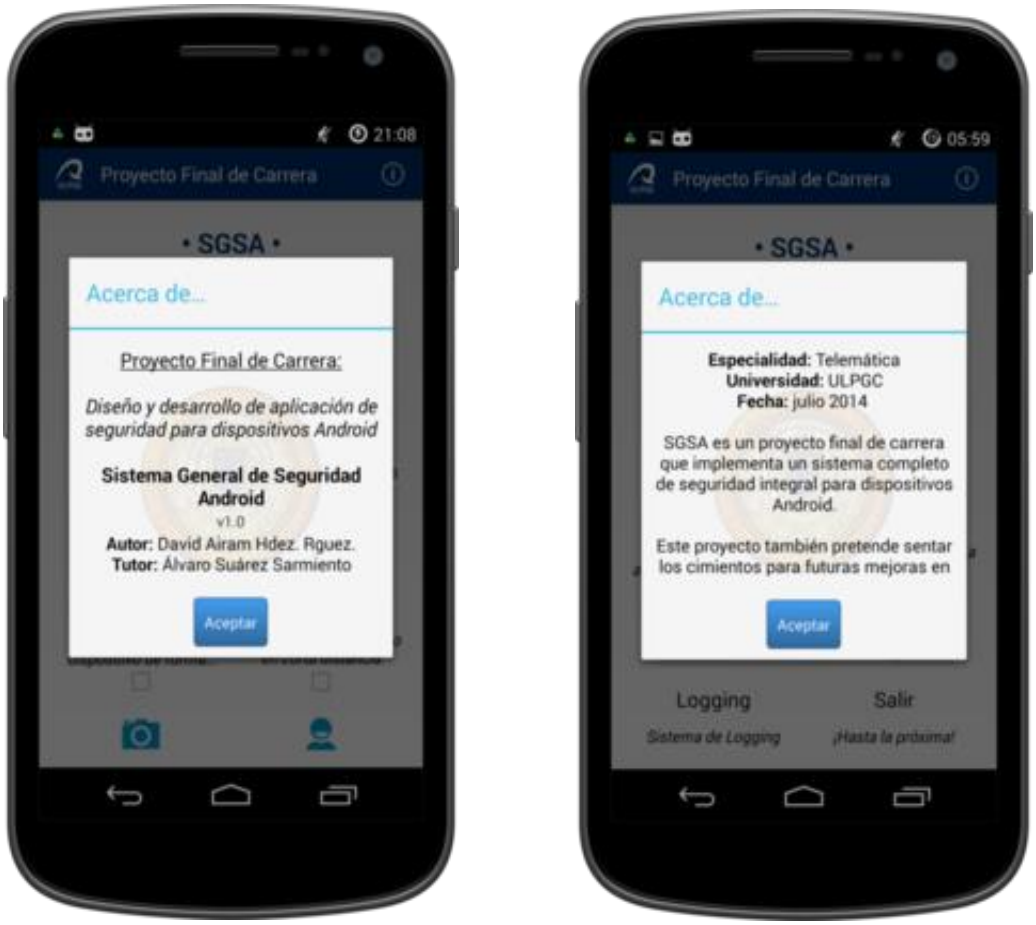


Figura 5.15. - Acerca de… Con la información del PFC y la aplicación desarrollada.

5.2 Ejemplos de funcionamiento y resultados

Finalmente en esta sección presentamos varios ejemplos de mensajes de activación y estudiamos los resultados obtenidos para cada caso, además podemos ver los mensajes de respuesta del sistema tanto en el envío de e-mails como en los mensajes de texto.

Pero antes de estudiar cada escenario en particular veamos todos los posibles parámetros que puede recibir el sistema mediante el mensaje de texto corto. Dichos parámetros acompañarán a la contraseña definida (*keyword)* en el mensaje de texto activador. En la Tabla Tabla 5.1 se pueden observar una lista de todos los parámetros disponibles así como una breve descripción para cada uno de ellos

Parámetro	Descripción
A	Activa el Modo Alarma. También se activará el Modo Alarma si no se pasa ningún parámetro después de la *keyword.*
S	Activa el Módulo de Geolocalización (Sigilo). Con este parámetro podemos acompañar una dirección de email de respuesta así como el tiempo en minutos de duración de este Módulo.
F	Solicita que se realice la captura de fotografía. Debemos acompañar una dirección de email para la respuesta.
G	Solicita que se realice una grabación de audio. Debemos acompañar una dirección de email para la respuesta y también podemos especificar la duración de la grabación en minutos. Si no se especifica la grabación durará por defecto 30 segundos.
M	Solicita con un único parámetro la captura de fotografía y grabación de audio (Modo Multimedia). Se debe pasar también la dirección de email y de manera opcional la duración del audio en minutos.
L	Solicita al sistema el registro completo de éste. Debemos acompañar también la dirección de email de respuesta.
E	Modo Especial, equivalente a SFG con un único parámetro.
STOP	Exige la detención de forma remota de todos los servicios del sistema.

Tabla 5.1. Resumen completo de los parámetros que puede recibir el sistema.

Ahora sí vamos a realizar diferentes pruebas al sistema y para los diferentes ejemplos y situaciones que pretendemos probar el estado de configuración del sistema es el siguiente (Tabla 5.2):

Palabra clave o *keyword*	aloha123
Mail alternativo	dahrkld@gmail.com
Teléfono alternativo	644600700
Módulo de Alarma	Sí
Módulo de Geolocalización	Sí
Responder remitente	Sí
Lista de Activadores	No
Control de Arranque	Sí

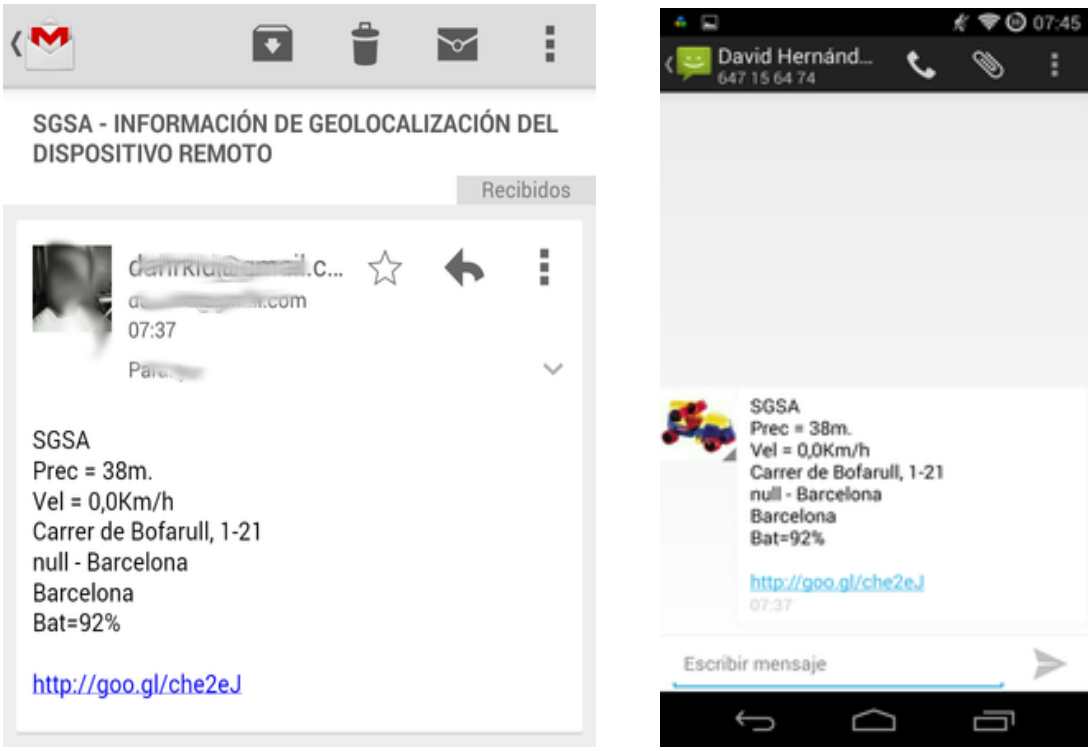
Tabla 5.2. Estado de configuración del sistema y teléfono móvil para las pruebas.

Escenario 1

En este caso enviamos un mensaje sencillo para activar el módulo de geolocalización pasándole también como parámetro una dirección de correo electrónico donde responder.

Mensaje activador: aloha123 s pepito@gmail.com
Parámetros:
s: es la solicitud de activación de módulo de geolocalización (sigilo).
pepito@gmail.com: es la dirección para la respuesta.

En la Figura 5.16 se muestran las respuestas de correo electrónico y la respuesta vía SMS.



(a) (b)

Figura 5.16. Respuestas al primer ejemplo. (a) Respuesta vía e-mail. (b) Respuesta vía SMS.

Escenario 2

En esta otra situación solicitamos únicamente la grabación de audio y la captura de imagen. También le pasamos un email de respuesta y le indicamos la duración de la captura de audio.

Mensaje activador: aloha123 fg 0.5 pepito@gmail.com
Parámetros:
f: realizar catpura de fotografía.
g: realizar grabación de audio
0.5: duración de la grabación de audio en minutos.
pepito@gmail.com: dirección para la respuesta.

En la Figura 5.17 se muestran las respuestas de correos recibidos para la imagen capturada (a) y para el audio grabado (b).

(a) (b)

Figura 5.17. Respuestas del segundo ejemplo. (a) Captura de imagen. (b) Captura de audio.

Escenario 3

En esta situación solicitamos el envío del archivo de registro, activar el modo de geolocalización y la alarma. Como en los casos anteriores también le pasamos un email de respuesta.

Mensaje activador: aloha123 las 20 pepito@gmail.com
Parámetros:
L: Petición del *log* de registro del sistema.
a: Solicitud de activación del módulo de alarma.
s: Solicitud de activación de módulo de geolocalización (sigilo)
20: tiempo de autoapagado en minutos del módulo de geolocalización.
pepito@gmail.com: dirección para la respuesta.

En la Figura 5.18 se muestran la respuesta del correo recibido.

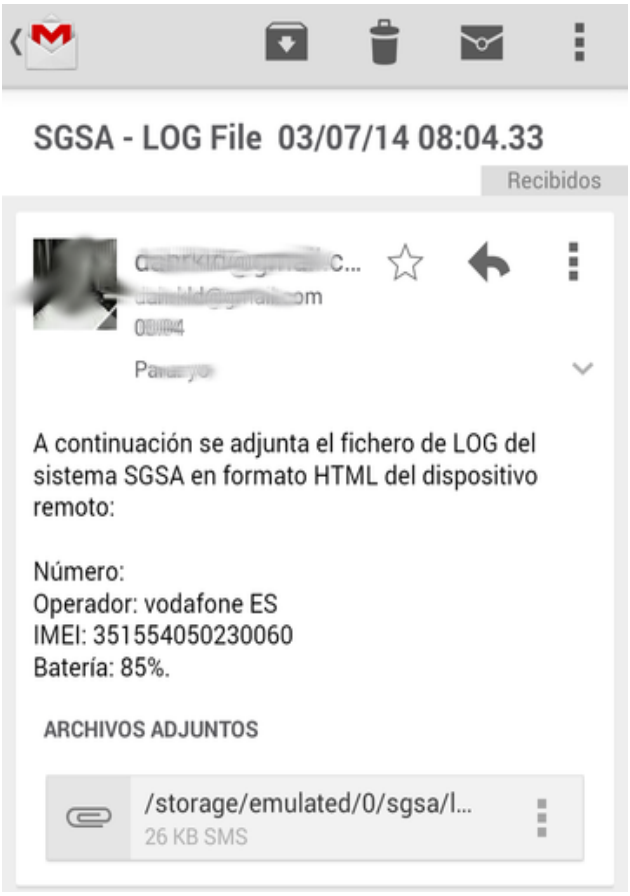


Figura 5.18. Respuestas al tercer ejemplo. El sistema nos envía el *log* del sistema.

Escenario 4

Para finalizar, en este último ejemplo vamos a forzar de forma remota una detención manual de todos los modos que hubieran activos en el terminal remoto.

Mensaje activador: aloha123 stop
Parámetros:
stop: forzar detención manual de todos los modos activos.

En este caso no hay una respuesta para el usuario que envía el mensaje activador ya que al solicitar este modo de detención de los servicios lo que se pretende es no recibir ninguna notificación más y esto implica parar la alarma, parar los servicios de geolocalización, detener la captura de imágenes, detener captura de audio si la hubiera… En definitiva el único registro lo vemos precisamente analizando el módulo de registro desde la misma aplicación (Figura 5.19).

Figura 5.19. El registro del sistema. Respuesta al cuarto ejemplo ejecutada con éxito.

6. Conclusiones y posibles ampliaciones

En este apartado final analizamos objetivamente el PFC desarrollado así como la aplicación implementada, su funcionamiento y los resultados obtenidos.

Esto da una visión resumida y global de lo que ha sido todo el desarrollo de este PFC y ayuda a comprender mejor el enfoque general del proyecto y las posibilidades que quedan abiertas en este campo y especialmente con el desarrollo de este proyecto.

6.1 Conclusiones

Sin duda la realización y finalización de este PFC ha sido bastante duro debido sobre todo a la cantidad de trabajo y materias nuevas que ha sido necesario estudiar y aprender, también ha tenido bastante culpa lo exigente y cambiante que es el Mundo de la programación, pero a la vez, también ha sido bastante satisfactorio y gratificante poder aprender a trabajar con todas estas tecnologías y especialmente en el desarrollo de aplicaciones móviles en el sistema operativo Android, ya que esto hace un poco más competente al ingeniero puesto que amplía su margen de acción como profesionales del sector.

El desarrollo de la aplicación en sí mismo y el hecho de verla en funcionamiento deja claro la gran base de desarrollo y trabajo que hay debajo. La conclusión en este aspecto es totalmente positiva ya que la aplicación es totalmente funcional y cumple con los objetivos planteados en el anteproyecto inicial de forma holgada.

Además la aplicación cuenta con características de funcionamiento muy interesantes como por ejemplo:

- La sencillez de uso con la configuración únicamente en local sin requerir darse de alta en servicios por Internet.
- Envío de e-mails de forma transparente para el usuario.
- La robustez de la alarma implementada.
- Poder capturar una imagen de forma transparente, siendo capaz automáticamente de detectar las cámaras del teléfono móvil y elegir la frontal si la tuviese disponible.
- El sistema de parámetros en el mensaje de texto activador.
- La Lista de Activadores para poder filtrar y darnos un extra de seguridad.
- La lógica interna del sistema de geolocalización.
- El módulo de Control de Arranque mediante gestos para solucionar los problemas de apagado y encendido.
- El completo sistema de registro de la aplicación.

Obviamente hay bastantes partes mejorables pero la mayoría de partes que no se han cerrado ha sido precisamente con la idea de dejarlas abiertas (semi-implementadas) a futuras aportaciones e ideas de otros alumnos.

Además sabemos que el campo de la seguridad que toca este PFC es bastante amplio, atractivo y cada vez es más importante aplicarlo correctamente en todos nuestros teléfonos móviles. Y es ahí precisamente donde este proyecto entra en acción brindándoles la oportunidad a otros alumnos de la universidad para que retomen este trabajo y así poder conseguir cada vez una mejor solución adaptada a los nuevos cambios de hardware así como a las nuevas ideas que surgen día tras día.

En resumen este proyecto es una base fundamental para que otros alumnos estudien el problema de la seguridad en los teléfonos móviles, así como la solución implementada en este proyecto pudiendo basarse o mejorar ésta con sus propias aportaciones e ideas.

6.2 Posibles ampliaciones

El campo de la seguridad en los *smartphones* es bastante amplio y si a ello le unimos las posibilidades que brindan hoy en día tanto los teléfonos móvil como el sistema operativo Android las opciones que tenemos son muchísimas.

En cualquier caso enumeramos las posibles ampliaciones que podríamos desarrollar para mejorar esta aplicación y este proyecto.

- Detección de cambio de SIM en caliente.
- Poder recibir los *mensajes activadores* a través de aplicaciones de mensajería instantánea tipo: Whatsapp, Telegram, Hangouts...
- Ocultar la activación de los servicios de localización en el teléfono móvil.
- Utilizar la tecnología bluetooth de emparejamiento para mejorar el sistema con un sistema tipo pulsera o smartwatch.
- Utilizar la tecnología *Near Field Communication* (*NFC*) para activar o desactivar ciertas funciones de la aplicación.
- Enmascarar la aplicación como otra aplicación de sistema.
- Permitir la captura de vídeos cortos.
- Mejorar la entrada de parámetros y hacerla más potente y sencilla.
- Detectar el patrón de movimiento del teléfono móvil para detectar cuando el usuario infractor está mirando hacia el teléfono móvil y en ese momento realizar la captura de imagen.
- Crear un servidor HTTP interno que permita un control total.
- implantar el backup de datos con algún servicio de terceros: Google Drive, OneDrive, Dropbox...

- Borrado remoto completo con confirmación doble.

Podríamos seguir enumerando posibles mejoras y adaptaciones al proyecto ya que las opciones son prácticamente infinitas y el tema de la seguridad en los teléfonos móviles es bastante amplio y atractivo pero dejemos que sean los futuros alumnos los que aporten con sus contribuciones e ideas.

Pliego de Condiciones y Presupuesto

En este capítulo desarrollamos un presupuesto completo suponiendo que dicha aplicación de seguridad la haya pedido un cliente real. Para ello tenemos en cuenta diferentes supuestos como que sólo habría un desarrollador y que la duración del proyecto sería de dos meses. A continuación presentamos el estudio y el presupuesto para el desarrollo de esta aplicación.

Estudio previo

Se ha estimado que toda la aplicación podría ser desarrollada en 8 semanas contando desde la planificación inicial hasta la fecha de la *entrega* final al cliente o publicación en mercado de aplicaciones.

También se ha estimado que el equipo técnico completo de desarrollo para llevar a cabo el proyecto en condiciones óptimas debería constar de:

- Un Gestor de proyectos.
- Un desarrollador Android Senior.
- Un diseñador de interfaces.
- Una persona de Q&A (tester).
- Ordenador de desarrollo con capacidades mínimas para mover holgadamente un proyecto Eclipse.
- Teléfono de la gama Nexus de desarrollo (mínimo Galaxy Nexus).
- Teléfonos Samsung y HTC de pruebas.

Pliego de condiciones

1) Se establece un margen de seguridad del 20% sobre el tiempo estimado. Este margen en caso de hacerse efectivo siempre será a costa del cliente. Si el margen superase dicho 20% del 20% en adelante sería a costo de la empresa desarrolladora.

2) Cualquier cambio por parte del cliente durante el proceso de desarrollo se factura al 1,8 de la tarifa habitual de cada profesional. Para aceptar dicho cambio se entrega un documento de *change-request*s al cliente el cual debe aceptar el sobrecargo y firmarlo.

3) El tiempo extra añadido debido a los diferentes cambios solicitados por el cliente no suma como parte del 20% de margen de seguridad que ha establecido el equipo técnico.

4) Cualquier retraso no contemplado donde la culpa recaiga en el equipo técnico lo afronta la empresa desarrolladora.

5) Se ha acordado que la aplicación sólo tiene retro compatibilidad hasta la versión 4.0.1 Ice Cream Sandwich de Android no asegurando su comportamientos para versiones anteriores.

6) Las modificaciones para la versión 4.4 de Android referente a la gestión de mensajes cortos de texto no entran en este primer proyecto, para versiones anteriores la aplicación cumple todo lo acordado previamente.

7) No se comenzará con el proceso de desarrollo propiamente dicho hasta que el cliente acepte el documento de funcionalidades completas y los esquemas de diseño de la aplicación a implementar.

Presupuesto

Con el equipo técnico seleccionado y el presupuesto a continuación presentado se estima que el proyecto duraría aproximadamente unas 8 semanas (con un margen de seguridad del 20%).

Concepto	**Importe**
Estudio previo con *preview* no funcional	168,00 €
Diseño de solución	
Preview no funcional	
Análisis de la solución con el cliente	
Documento de especificaciones técnicas	149,00 €
Documento funcional	149,00 €
Listado de *wireframes*	249,00 €
Reunión y firma del proyecto por ambas partes	0,00 €
Explicación del documento de especificaciones y firmado	
Explicación del documento funcional y firmado	
Revisión del listado de *wireframes* y firmado de cada uno	
Acuerdo de fecha de release final (con margen del 20% si todo OK)	
Project Management (8 semanas)	6.750,00 €

Planificación de recursos	
Definición del método de trabajo con el equipo	
Definición de sprints y del número de releases	
Gestión de cambios (change-requests)	
Comunicación con el cliente	
Diseño completo (4 semanas)	3.860,00 €
Creación de logo e imagen corporativa	
Diseño de *wireframes*	
Diseño y esquema de colores global	
Crear diseños finales de cada pantalla partiendo (*visuals)*	
Iconografía	
Desarrollo de la aplicación (8 semanas)	5.940,00 €
Preparación del entorno	
Esqueleto de la aplicación	
Desarrollo completo (core, adaptador y vista)	
Test coverage del 40% (no en vista)	
Preparación de versiones beta por sprint	
Quality & Assurance (Q&A) (6 semanas)	2.160,00 €
Testeo continuo de cada versión liberada	
Control de bugs	
Informes exhaustivos de cada versión	
Pruebas de carga, pruebas *monkey...*	
teléfonos móvil de prueba y coste de las pruebas	
Proceso de release final	1.200,00 €
Pruebas finales	

Publicación en Google Play	
Subtotal	**20.625,00 €**
Impuestos (5%)	**1.031,25 €**
TOTAL	**21.656,25 €**

El presupuesto y coste total del proyecto asciende a VEINTIUN MIL SEISCIENTOS CINCUENTA Y SEIS COMA VEINTICINCO (21.656,25) euros.

El proyectando:

David Airam Hernández Rodríguez.

ANEXO I. Conceptos Android de proyección de los módulos orgánicos

En este anexo se exponen una serie de conceptos que ayudan a entender la proyección de los módulos orgánicos al desarrollar un proyecto Android con Eclipse.

I.1 Configuración inicial del proyecto

Lo principal a la hora de desarrollar un proyecto de software es haber hecho un *documento de funcionalidades* acompañado de unos *wireframes* (esquemas de diseño) y finalmente de unos *visuales* acordes con los *wireframes* previamente diseñados.

En este caso explicamos partes importantes del desarrollo de un proyecto Android usando éste proyecto como ejemplo.

La elección del *target* al desarrollar una aplicación en Android es bastante importante ya que va a definir a partir de qué versión de Android nuestra aplicación es compatible y por lo tanto se podría instalar. Es decir, se instala en aquellos teléfonos móviles cuya versión sea igual o superior a dicho target y excluye totalmente al resto de teléfonos móviles. Para ello debemos de configurar dos parámetros básicos. Dicho parámetros son dos números enteros que representan valores de la API de Android: *minSdkVersion y targetSdkVersion*

Como comentamos anteriormente dichos parámetros van a definir los límites de versiones de la API de Android (teléfonos móviles) donde nuestra aplicación se va a poder instalar y funciona correctamente:

- *minSdkVersion*: este parámetro es el mínimo nivel de API requerido para ejecutar tu aplicación, de hecho el sistema Android no deja instalar tu aplicación en teléfonos móviles cuya versión sea inferior al *minSdkVersion.*

- *targetSdkVersion*: este valor es otro entero que representa el valor para el que la aplicación está diseñada. Este parámetro no significa que no se vaya a permitir instalar la aplicación en teléfonos móviles con una API superior al *targetSdkVersion* ya que realmente nuestra aplicación siempre se va a poder instalar desde el *minSdkVersion* hasta la última versión del sistema operativo. Este parámetro lo que representa es que la aplicación ha sido testeada y probada para funcionar con esta versión de sistema y que además el sistema deshabilita cualquier versión de compatibilidad que no sea

necesaria para la versión *targeted.* Con lo cual desde nuestra version target hasta nuestro versión mínima se utilizan las bibliotecas de compatibilidad para aquellas características de la target version o de versiones superiores si es que están disponibles.

Estos parámetros normalmente se configuran en el archivo *AndroidManifest.xml* dentro de la sección *uses-sdk.* Y en proyectos Android Studio aunque también se puede configurar en el también en el archivo *manifest* la tendencia es que se haría a través del archivo *build.gradle* y dicho valor sobrescribiría el del *manifest*.

En nuestra aplicación dicha configuración aparece de la siguiente manera:

```
<uses-sdk
  <!-- Ice Cream Sandwich 4.0.1 -->
  android:minSdkVersion="14"

  <!-- Ice Cream Sandwich 4.1.x -->
  android:targetSdkVersion="16"
/>
```

I.2 El archivo *Manifest*

El archivo *manifest* de un proyecto Android es probablemente uno de los archivos más importantes del proyecto. En este archivo se especifican la mayoría de los parámetros básicos y especificaciones de la aplicación en formato XML.

Este archivo se genera automáticamente cuando creas un proyecto Android con Eclipse (o con Android Studio). Dicho archivo lo puedes editar a través de un editor gráfico aunque lo mejor es aprender a editar el código XML directamente ya que no es para nada difícil y te da muchísimas más libertades además de conocer mejor la estructura de dicho archivo.

En este archivo *manifest* se van a especificar prácticamente todas las características que la aplicación requiera así como los distintos elementos que la conforman. Intentar resumir de forma general los elementos más importantes que pueden aparecer en el archivo *manifest* de un proyecto Android:

- *Target mínimo de la API que la aplicación requiere*.
- *Versión de la API para la cual se desarrolla la aplicación.*

- *Nombre de la aplicación*.
- *Descripción*.
- *Icono*.
- *Componentes Hardware del teléfono móvil que la aplicación requiere*.
- *Permisos que la aplicación requiere*.
- *Operaciones que la aplicación podría ejecutar*.
- *Claves para bibliotecas*.
- *Todas las Actividades (ventanas) que conforman la aplicación*.
- *Diseño (Themes) y Estilos por defecto*.
- *Orientación de la pantalla para cada ventana*.
- *Servicios*.
- *BrodcastReceivers públicos*.
- *Intents & IntentFilters...*

I.3 El diseño de *Layouts*

En Android el diseño de *layouts* está basado en archivos XML y es bastante potente además se puede usar el diseñador integrado que proporciona el *plugin* ADT el cual aunque no llega a los niveles del proporcionado por Android Studio sí que es bastante potente.

En la Figura I.1 mostramos con números los elementos principales de estos layouts que explicamos a continuación:

Punto 1: caja de controles donde podemos ir eligiendo a través de cada pestaña el control a añadir al área de diseño (6).

Punto 2: diferentes pestañas de la caja de controles (1).

Punto 3: en esta sección podemos cambiar entre la vista gráfica y la vista de XML puro. La vista gráfica es la que se ve actualmente.

Punto 4: estructura en forma de árbol donde podemos ver todos los controles que tenemos en el diseño. Los podemos seleccionar borrar, cambiar propiedades...

Punto 5: sección donde podemos ver y cambiar todas las propiedades del objeto seleccionado.

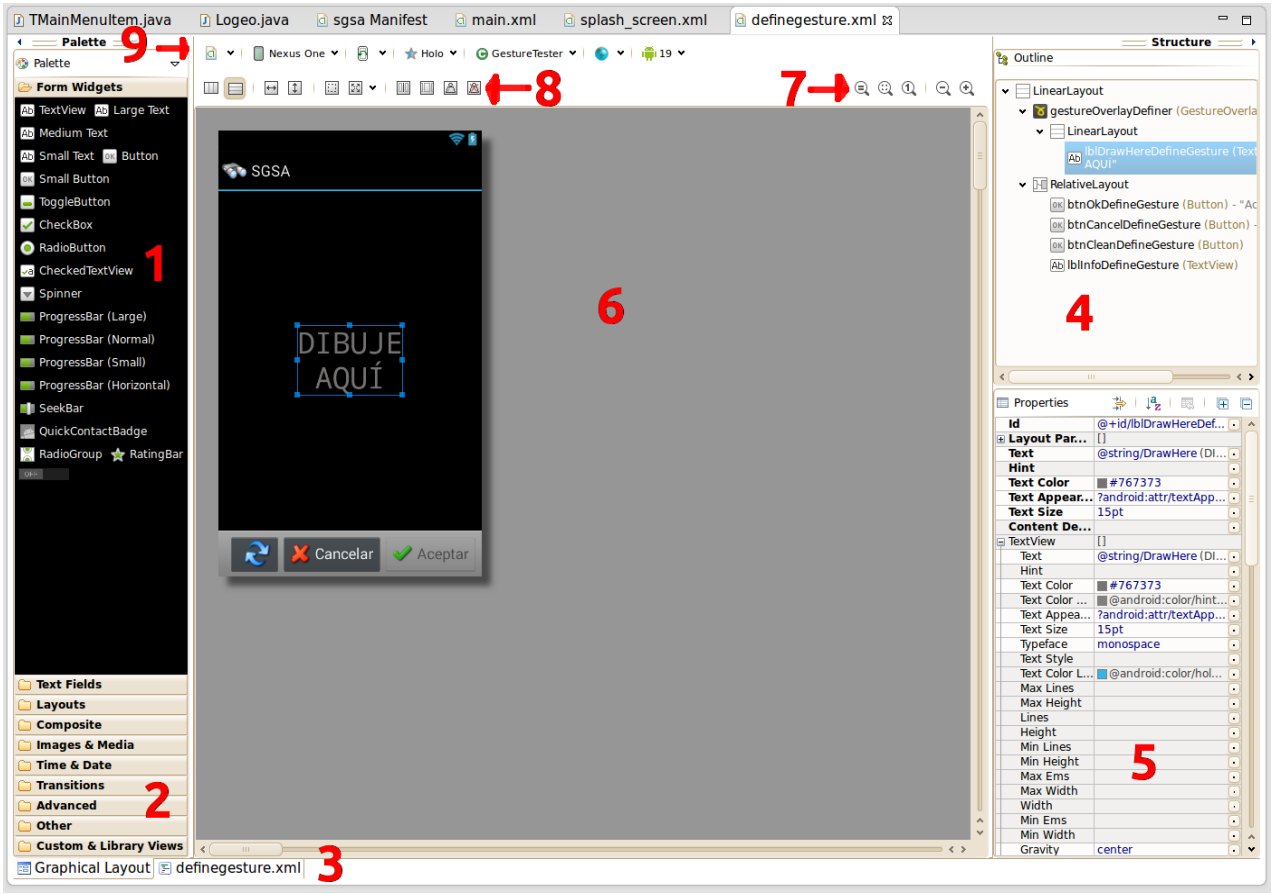


Figura I.1. Diseño de *layouts* en Eclipse

Punto 6: zona de diseño del *layout* (vista) en sí mismo.

Punto 7: controles de zoom que actúan sobre el área de diseño (6).

Punto 8: sección donde podemos cambiar de forma rápida el tamaño, márgenes, alineación, pesos... del objeto seleccionado.

Punto 9: con estos botones podemos cambiar el área de diseño acorde a otro tamaño de pantalla sólo de cara a ver cómo quedaría en tiempo de diseño. También podemos cambiar el *theme* por defecto, cambiar a otros layouts, idiomas así como la versión de renderizado de la API si tenemos varias instaladas.

Figura I.2. Editando archivo XML de *layouts* en Eclipse.

Aunque siempre es recomendable trabajar directamente con el archivo XML en vez de con el diseñador ya que a la larga con la práctica somos más rápidos, productivos y precisos. (Figura I.2)

A la hora de diseñar una pantalla (*layouts)* quizá la parte más importante y compleja sea el manejar correctamente los diferentes tipos de *ViewGroups* o *Layouts.* Estos elementos realmente son contenedores para los otros elementos gráficos o *views* y permiten organizar las *views* a nuestro antojo. Los *layouts* más importantes y los que más usamos son:

- *LinearLayout:* es probablemente el tipo de layout más utilizado. permite agrupar las vistas una tras otra ya sea en una orientación *vertical* u *horizontal.* También podemos dejar diferentes márgenes entre cada elemento y darle un alineado (*gravity)* global para todas las *views* que contendría dicho layout. Finalmente también podemos establecer el tamaño de las vistas en base a *pesos* dejando el parámetro de cada view que quedamos definir mediante pesos a *0dp.*

- *RelativeLayout:* es de los más versátiles y más usados en el diseño de pantallas en Android ya que permite colocar cada view en relación a los parámetros del mismo

layout (*anclarlo a cualquier margen, centradas, con márgenes...)* o bien en base a la posición de cualquier otra de las *views* que contenga el layout. Esto permite organizar las vistas prácticamente con cualquier nivel de complejidad y de la forma que queramos.

- *FrameLayout:* permite agrupar las vistas una encima de la otra. Obviamente al igual que con cualquier otra view podemos dejar márgenes con relación a los bordes del layout pero no en base a las otras *views* como si ocurre con el *Relative Layout.* Es útil para agrupar imágenes o poner textos encima de imágenes o viceversa.

- *TableLayout:* permite no tener que introducir varios *LinearLayouts* para organizar las *views* en un formato de tabla. Este layout como su nombre indica permite organizar las *views* como si fuese una tabla y se apoya en otro objeto que es el *TableRow.* Además, podemos ocultar columnas o permitir redimensión de las columnas. También a través de las propiedades a*ndroid:layout_rowSpan* y *android:layout_columnSpan* podemos conseguir que una celda ocupe el lugar de varias filas o columnas.

  android:stretchColumns: Indica las columnas que pueden expandir para ocupar el espacio libre dejado por el resto de columnas.

  android:shrinkColumns: Indica las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la parte derecha de la pantalla.

  android:collapseColumns: Indica las columnas de la tabla que se quieren ocultar completamente.

- *GridLayout:* fue incluido a partir de la API 14 (Android 4.0) y sus características son similares al TableLayout, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el GridLayout de colocar y distribuir sus elementos hijos en el espacio disponible. En este caso, a diferencia del TableLayout indicamos el número de filas y columnas como propiedades del layout, mediante *android:rowCount* y *android:columnCoun*t. Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento *TableRow* del *TableLayout*, sino que los diferentes elementos hijos se ian colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation) hasta completar el número de filas o columnas indicadas en los atributos anteriores. Adicionalmente, igual que en el caso anterior, también tenemos disponibles las

propiedades *android:layout_rowSpan* y *android:layout_columnSpan* para conseguir que una celda ocupe el lugar de varias filas o columnas.

Cabe decir que el tamaño de cada layout lo podemos establecer a nuestro gusto tanto a un tamaño fijo, siempre en density pixels (dp) y para fuentes en scaled pixels (sp) o usando los parámetros *wrap_content* (que se ajuste al tamaño de su contenido) o *match_parent* para que ocupe el tamaño máximo posible de su contenedor. Obviamente es posible y además muy común tener layouts dentro de otros.

En definitiva Android a día de hoy ofrece una herramienta para el diseño de interfaces visuales muy potente y cómoda. Además desde la versión 4.0 hay mucha unificación y retrocompatibilidad con las bibliotecas de soporte.

I.4 Los *resources*

En Android la gestión de recursos es bastante interesante y también muy potente. Entendemos por recursos de un proyecto todo lo que no sea código fuente, es decir: *imágenes, iconos, fuentes, sonidos, texturas, fondos, shapes, tipografías, datos localizados como cadenas de texto, valores, arrays de valores, códigos de colores...*

El hecho de externalizar este tipo de recursos permite poder gestionarlos y actualizarlos sin tener que cambiar el código fuente. Además el sistema de gestión de recursos de Android también permite cosas realmente interesantes como tener diferentes recursos dependiendo del idioma de la aplicación o la orientación de la pantalla y un sinfín de variables posibles que presentamos a continuación.

Dentro de la estructura de un proyecto Android tenemos dos directorios básicos que hacen referencia precisamente a la gestión de recursos y que se encargan de gestionar y contener dichos recursos:

res/
assets/

Por lo que a partir de ahora hablamos de dos tipos de recursos; los *resources* que se encuentran dentro del directorio *res/* y los *assets* que se encontran del directorio *assets/.*

Pues bien vayamos por partes y veamos que representa cada uno y cuáles son sus diferencias.

Resources

Los recursos dentro del directorio *res/* tienen muchas particularidades especiales que los convierte en precisamente eso, especiales, pero quizá la más importante es que para todos los recursos que organicemos dentro de este directorio tenemos un identificador único en tiempo de diseño. En otras palabras, para cada cambio que realicemos en este directorio se auto-genera una clase especial (no editable manualmente) que es (una clase muy famosa en proyectos android) clase *R* del proyecto. A través de esta clase tenemos identificadores a cada uno de los archivos que tengamos en dicha carpeta.

Por ejemplo, si tenemos el siguiente archivo *res/drawable/icono_button_ok.png* accedemos a su identificador en código fuente a través de la expresión *R.drawable.icono_button_ok* y podemos obtener dicha imagen con la siguiente línea de código:

*getResources().getDrawable(**R.drawable.icono_button_ok**);*

Tener el identificador disponible en tiempo de diseño permite evitar problemas de aperturas de archivos, equivocaciones en el nombre del recurso, evita errores de tipo I/O... Ya que si no existe el recurso el mismo editor va a indicar que dicho tipo (identificador) no existe o no es válido y como no podemos editar dicha clase a mano si no que es auto-generada y mantenida por el sistema Android (plugin ADB) no hay posibilidad de cometer errores en este punto.

Hay que tener en cuenta que los nombres de los recursos en la carpeta de res/ no pueden tener espacios ni guiones medios.

Como ya comentamos anteriormente, dentro de esta carpeta guardamos muchísimos tipos de recursos: *animaciones, imágenes, cadenas, xmls, colores...* Y de forma similar a como hicimos con el ejemplo de la imagen anterior podemos obtener dichos valores de una forma sencilla a través de la clase R de nuestro proyecto Android.

A continuación mostramos las carpetas por defecto que el sistema Android permite crear para gestionar y guardar nuestros recursos, y en resumen los diferentes tipos de recursos que guarda y gestiona de esta forma (Tabla I.1):

Directorio	Tipo de recurso
animator/	archivos XML para definir animadores.
anim/	archivos XML para definir animaciones.
color/	archivos XML para definir Colores.
drawable/	archivos de imagen*: .bmp, .png, .jpg, .gif, .9.png, android shapes, y otros android drawables etc.*
layout/	archivos XML que definen layout (pantallas).
menu/	archivos XML que definen menús: *OptionsMenu, ContextMenu, SubMenu...*
raw/	archivos de otros tipos sin orden alguno.
values/	archivos XML para valores: *strings, arrays, dimens, styles. etc.*
xml/	archivos XML que pueden ser leídos con *Resources.getXML().* Se suelen guardar archivos de configuración de settings.

Tabla I.1. Directorio de recursos soportados en Android. Todos están dentro de la carpeta de proyecto res/.

Para cada tipo y subtipo de recurso tenemos disponible su método para obtener el contenido de dicho recurso en la clase *Resources* de Android:

```
getResources().getColor(...)
getResources().getString(...)
getResources().getAnimation(...)
getResources().getInteger(...)
getResources().getMovie(...)
getResources().getDrawable(...)
...
```

Pero la verdadera utilidad de los *resources* en Android no es sólo el hecho de contar con las clases de apoyo que permiten acceder de forma fácil y sencilla a estos recursos sin riesgo

alguno, la verdadera potencia radica en que el sistema permite (a través de la estructura que vimos anteriormente que representa todos los tipos de recursos básicos que podemos tener) crear recursos alternativos basados en especificaciones determinadas de los teléfonos móvil u otras características e incluso combinarlas

Estas especificaciones determinadas pueden ser características del hardware de los teléfonos móvil, orientación de la pantalla, modos UI, densidades de pantalla diferentes, versión de API diferente, idioma o región... Para conocer todas las posibilidades que permite el sistema Android podemos visitar en el link [9] de las referencias.

Android consigue esto de una forma muy elegante y es simplemente añadiendo el sufijo (*calificador de configuración*) correspondiente al nombre de la carpeta de recursos que queramos. La lista completa de los calificadores ordenados por tipo la podemos encontrar en la dirección web anterior. La estructura a crear sería:

/res/<tipo de recursos>-<calificador>

Donde *tipo de recursos* son los vistos anteriormente en la estructura dentro de */res/* y el calificador sería un sufijo de texto que representa la particularidad del sistema o teléfono móvil.

Cabe decir que podemos anidar varios tipos de calificadores:

/res/<tipo de recursos>-<calificador1>-<calificador2>

Por ejemplo, si queremos tener los iconos de la aplicación para diferentes densidades de pantalla podríamos tener la siguiente carpeta con los iconos de la aplicación por defecto:

/res/drawable/

main_icon.png
delete_icon.png
add_icon.png
background.png

Y después tener ésta otra estructura sólo para pantallas de alta resolución:

/res/drawable-xhdpi/

main_icon.png
delete_icon.png
add_icon.png
background.png

En este caso el calificador *xhdpi* significa: *densidad de pantalla extra alta de aproximadamente 320dpi*. Con lo que, en dicha carpeta deberíamos meter los mismos iconos pero con una resolución mayor para que se vieran bien en una pantalla con esas especificaciones.

Otro ejemplo básico es el de los textos localizados, ya que en la carpeta por defecto tendríamos los textos en inglés (por ejemplo) y después creamos carpetas para cada uno de los diferentes idiomas que queramos soportar en nuestra aplicación:

/res/values/strings.xml	para valores por defecto (inglés por ej.)
/res/values-es/strings.xml	para valores en español
/res/values-fr/strings.xml	para valores en francés
…	
/res/values-de/strings.xml	para valores en alemán

Cuando no especificamos un calificador a una carpeta de tipos de recursos decimos que esta es la *carpeta por defecto*, por ejemplo */res/drawable/* es la carpeta por defecto para guardar todos los recursos gráficos. La carpeta por defecto es muy importante ya que se usan los recursos de la carpeta por defecto siempre que no exista el mismo recurso (mismo nombre) en una carpeta con calificador y que el sistema y el teléfono móvil cumplan con ese calificador o calificadores. En otras palabras, se usan los recursos de la carpeta por defecto cuando no exista el recurso en alguna de las carpetas calificadas que coincida con las especificaciones del sistema o teléfono móvil.

Es muy importante que las carpetas por defecto tengan siempre el recurso por defecto y después creamos carpetas para configuraciones independientes ya que si por casualidad no existiera un recurso que solicitamos en la carpeta por defecto y el teléfono móvil no cumpliese con los requisitos de ninguna de las carpetas calificadas encontramos con un error en tiempo de ejecución al no poder seleccionar el recurso válido. Obviamente el compilador no informa de este error ya que realmente si tienes este recurso en una carpeta calificada por lo que a la hora

de programar no existe ningún problema, el problema viene cuando el sistema Android va a elegir el recurso que mejor se adapte al teléfono móvil y elimine las carpetas calificadas por incompatibilidad con el teléfono móvil en el momento de ejecución.

Ejemplo:

res/drawable/boton_comprar.png
res/drawable-xxhdpi/boton_comprar.png
res/drawable-xxxhdpi/boton_comprar.png
res/drawable-de/boton_comprar.png
res/drawable-de-xxxhdpi/boton_comprar.png
res/drawable-de-xxhdpi/boton_comprar.png

Supongamos que el teléfono móvil de ejemplo está en idioma *español* y su resolución es *hdpi*. Al estar en idioma español ya el sistema elimina (opciones tachadas) de su decisión para elegir el *resource* las carpetas de otros idiomas:

res/drawable/boton_comprar.png
res/drawable-xxhdpi/boton_comprar.png
res/drawable-xxxhdpi/boton_comprar.png
~~*res/drawable-de/boton_comprar.png*~~
~~*res/drawable-de-xxxhdpi/boton_comprar.png*~~
~~*res/drawable-de-xxhdpi/boton_comprar.png*~~

Y al no cumplir con la densidad de pantalla queda sólo la carpeta por defecto*:*

res/drawable/boton_comprar.png
~~*res/drawable-xxhdpi/boton_comprar.png*~~
~~*res/drawable-xxxhdpi/boton_comprar.png*~~
~~*res/drawable-de/boton_comprar.png*~~
~~*res/drawable-de-xxxhdpi/boton_comprar.png*~~
~~*res/drawable-de-xxhdpi/boton_comprar.png*~~

En general los calificadores de recursos más utilizados son los de la densidad de pantalla: *hdpi, xhdpi, xxhdpi...* ya que es la forma más cómoda y correcta de crear recursos para diferentes resoluciones independientemente del tamaño de la pantalla. Otro tipo de calificador muy utilizado en los recursos es obviamente el de lenguaje. Y finalmente el último sería el de

orientación de la pantalla (éste se usa sobre todo en layouts para tener diseños para vista vertical y horizontal). El resto también son usados pero en menor medida y depende de la complejidad de nuestra aplicación y del *minSdk* que usemos.

En la sección de anexos al final de la memoria en la sección X.X se podría ver la estructura del directorio *res/* del proyecto desarrollado.

Assets

A diferencia de los recursos contenidos en el directorio *res/* los contenidos en el directorio *assets/* y llamados a partir de ahora de esta manera, *assets* pierden toda la ventaja de tener las clases de apoyo para su gestión así como obviamente el hecho de tener un identificador autogenerado en tiempo de de diseño para su acceso y por supuesto no pueden ser ordenados y/o cualificados por características del teléfono móvil o por idioma salvo que lo haga el programador de forma manual.

En la carpeta de *assets* ponemos todos aquellos archivos que no son capaz de ser manejados a través de la clase *getResources()* de Android, es decir que no son de los tipos básicos que pueden ser gestionados en la carpeta *res/*.

En esta carpeta se suelen guardar *texturas de juegos, archivos de tipografías, archivos de diccionarios, otros tipos de archivos que no sean de ningún tipo básico que pueda ser guardado en los recursos estándar, o archivos que no necesitan ser calificados de ninguna forma.*

Dentro de la carpeta de assets podemos organizar los *assets* a nuestro antojo y tenemos total libertad. Los archivos aquí contenidos se leen de forma normal como cualquier otro archivo en Java aunque tenemos una pequeña clase de apoyo llamada *AssetManager* con su método *getAssets()* perteneciente al contexto de una actividad Android.

Normalmente si no estamos desarrollando un juego, una aplicación compleja que haga uso de (Open Graphics Library) OpenGL, no tenemos tipografías personalizadas, archivos criptográficos o ficheros complejos que no nos interese tener calificados en */res/raw* es normal que no usemos los recursos de tipo *assets* en nuestro proyecto.

I.5 *Activities* y *Fragments*

En un proyecto Android existen diferentes componentes que permiten desarrollar nuestra aplicación de una manera más eficiente y siempre dentro de los estándares Android. Ejmplos de estos componentes son: *Activities, Fragments, Services, BroadcastReceivers, ContentProviders...*

Los dos componentes más importantes que componen la parte visual son los dos primeros: *Activities y Fragments.* Por lo que es muy importante conocer y estudiar el ciclo de vida de dichos componentes el cual es muy parecido ya que un fragmento siempre está incrustado dentro de una *Activity* que es el componente principal para crear ventanas (la parte visual) en una aplicación Android.

Cuando no existían los *Fragments* en la API todos los *layouts* se cargaban en un objeto Activity con lo que si el diseño del *layout* era muy complejo la carga de dicha Actividad requería/requiere un tiempo considerable. Además el hecho de tener que crear e instanciar una actividad totalmente nueva cuando queremos cambiar el diseño de la actividad de forma parcial hacía que el sistema no fuese muy eficiente.

En cualquier caso hay que saber cuando debemos elegir una navegación mediante *fragments* y cuando elegir una actividad nueva.

A continuación veremos una imagen de los tres conceptos más básicos resumidos de como crear una Actividad y/o Fragmentos con su diseño (*layouts).* (Figura I.3)

Después de ver los tres casos básicos cabe recordar que cuando hablemos de un *Fragment Container* lo que realmente tenemos es la posibilidad de ir cambiando de Fragments (con diferentes layouts) dentro de la misma Actividad, tenemos incluso si lo deseamos acceso a la pila de Fragments que hemos ido añadiendo y obviamente podemos pasar parámetros entre Fragments.

Siempre hay que tener claro que en Android cambiar de *Fragment* es un proceso mucho más ligero que cambiar de *Activity* que es un proceso realmente pesado.

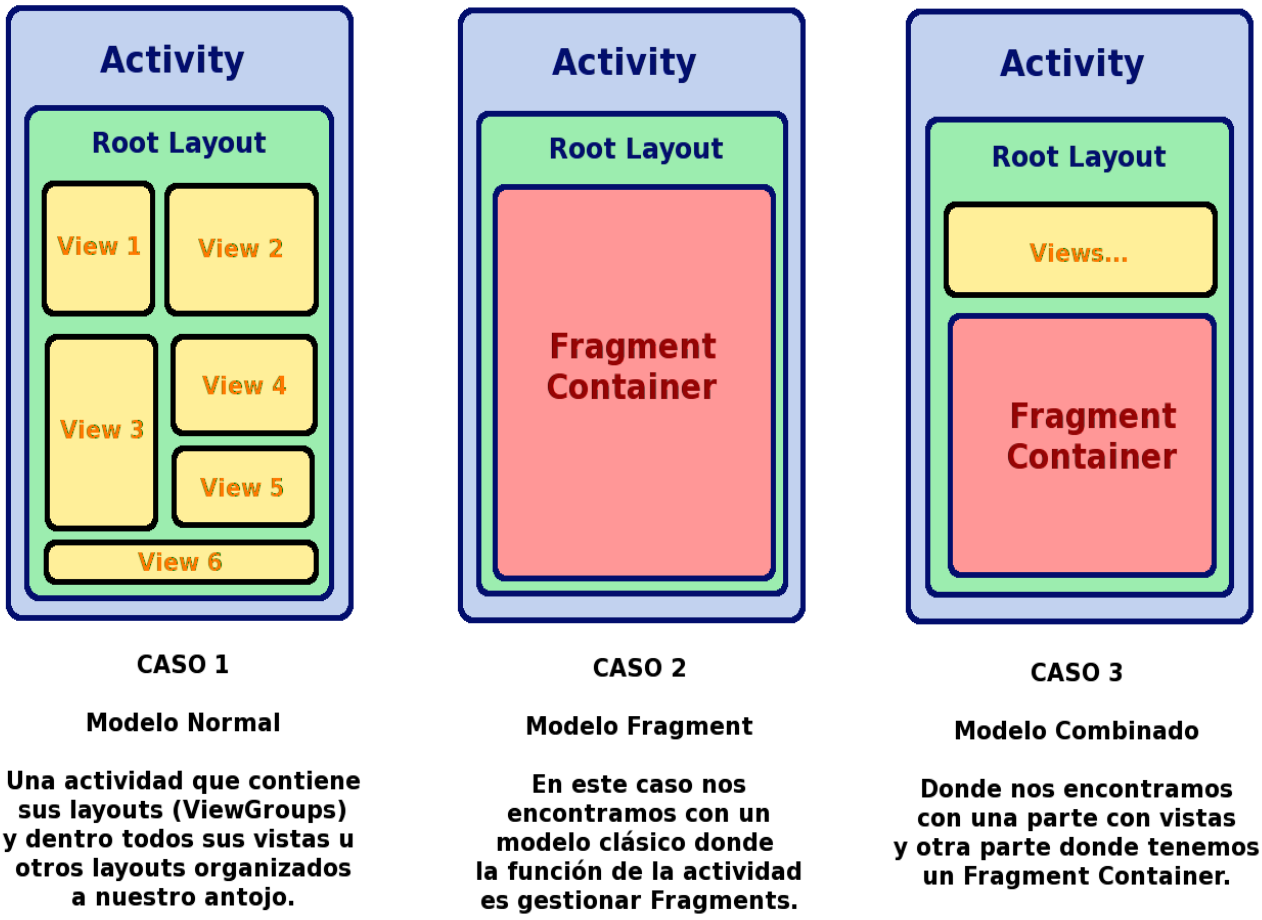


Figura I.3. Los tres tipos de combinaciones básicos entre *Activities, Fragments* y *Views.*

El ejemplo más clásico de cuando hacer uso de *Fragments* puede ser en el clásico *Wizard* o cuando tenemos una parte fija en una Actividad (menú) y queremos ir variando la otra parte. La parte variable sería el *Fragment* (Fragment Container) y la parte fija sería la *Activity*. Esto se suele utilizar por ejemplo en las típicas aplicaciones que presentan opciones a la izquierda y dependiendo de la opción seleccionada nos mostraría un layout totalmente diferente en la derecha pero manteniendo el resto de objetos (Barras de Android…) tal como están. Esto que acabamos de comentar lo podemos observar en la siguiente Figura I.4. También en la Figura I.5 vemos definido el proceso de búsqueda y cambio (gestión) de *fragments* que se lleva a cabo en la *Activity.*

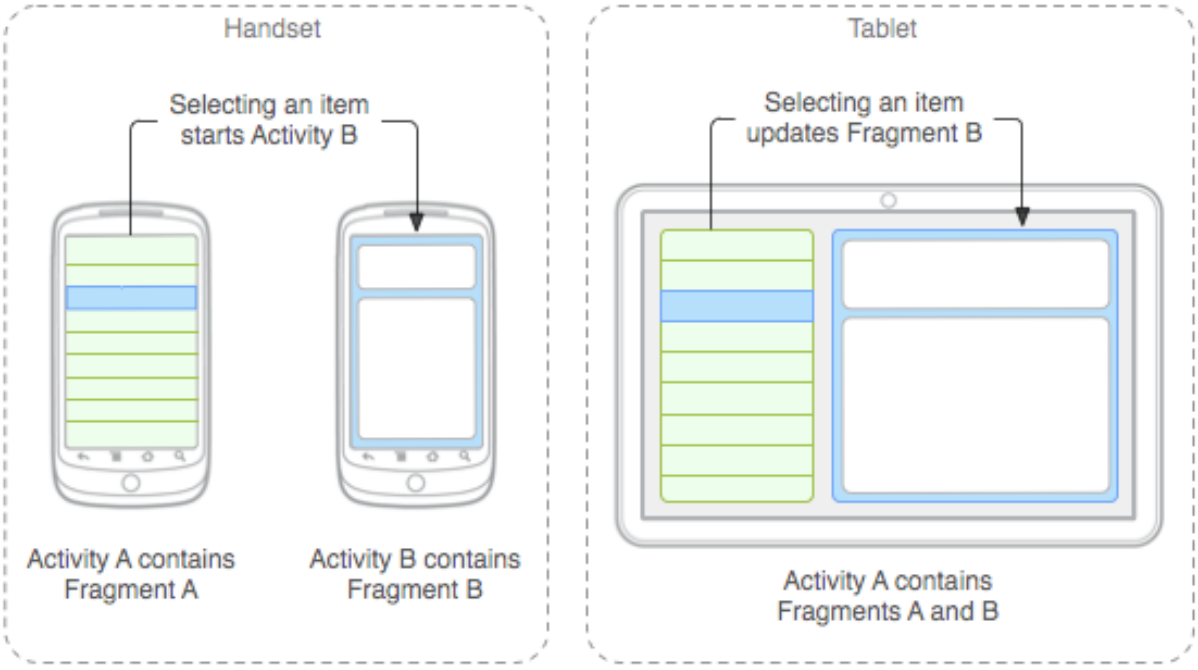


Figura I.4. Uso y navegación de *fragments.*

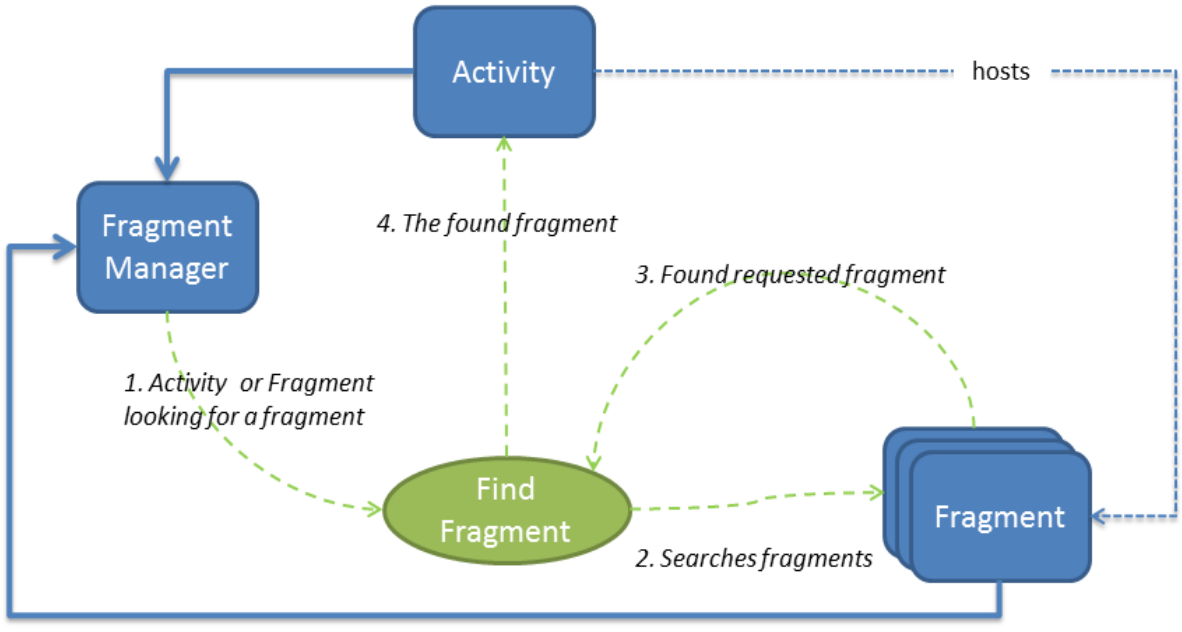


Figura I.5. Gestión de Fragments en una *Activity.*

En el momento de desarrollar una aplicación es muy importante controlar los *ciclos de vida* de cada elemento. En este momento centramos en los ciclos de vida de la *Activity* y del *Fragment*. Para controlar los ciclos de vida de cada elemento simplemente tenemos que sobreescribir los métodos del ciclo de vida de cada elemento que presentamos a continuación. (Figura I.6 y Figura I.7)

Ciclo de vida de una Activity

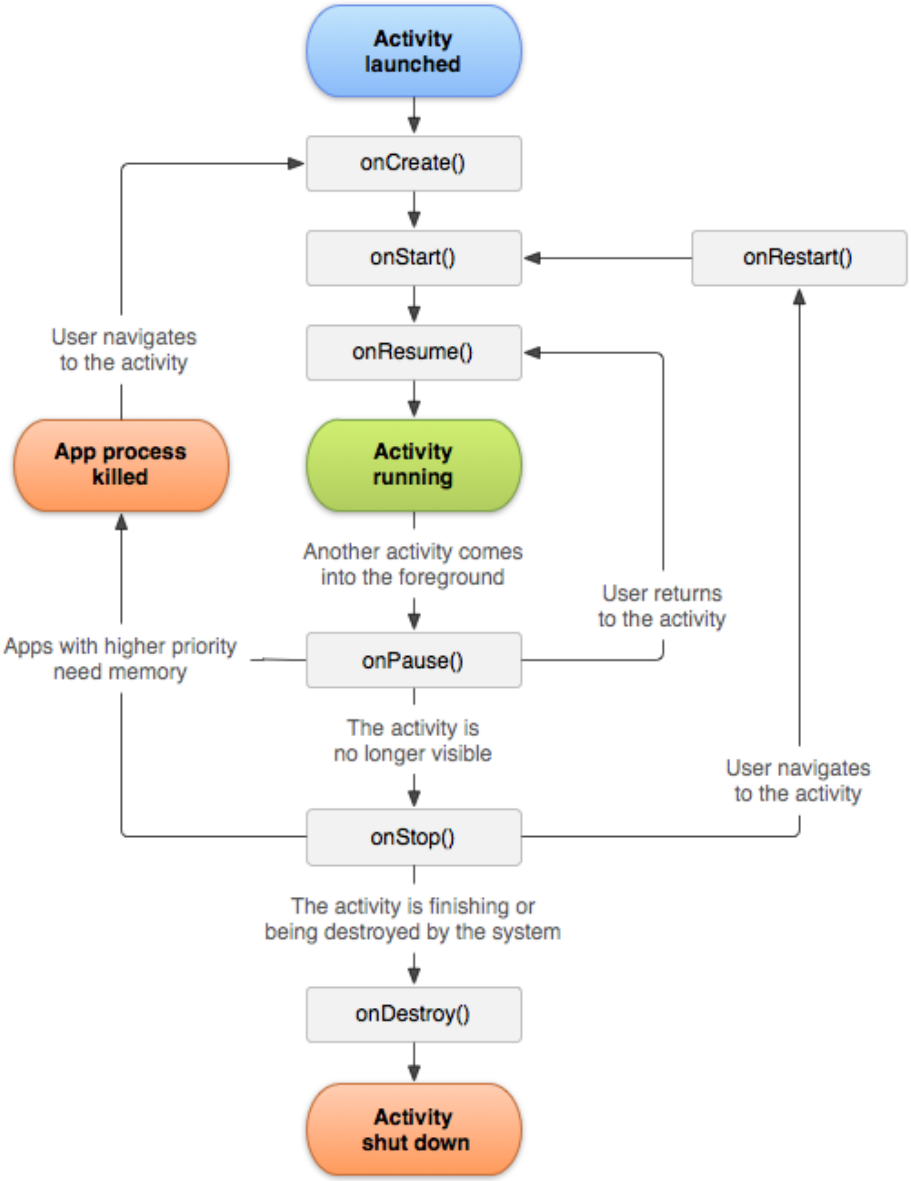


Figura I.6. Ciclo de vida de una *Activity* en Android.

Los métodos más importantes que debemos sobreescribir del *lifecycle* de la *Activity* son:

- *onCreate():* es el método que se sobreescribe para inflar los componentes visuales y realizar operaciones iniciales.

- *onResume():* este método ocurre cada vez que se termina de pintar (o reaparece) la actividad en pantalla. Por ejemplo al apagar y encender la pantalla ocurre este evento de la actividad que estuviese activa.

- *onPause():* es justo el opuesto del método *onResume.*

- *onDestroy()*: este método ocurre cuando el sistema destruye totalmente la actividad.

Ciclo de vida de un Fragment

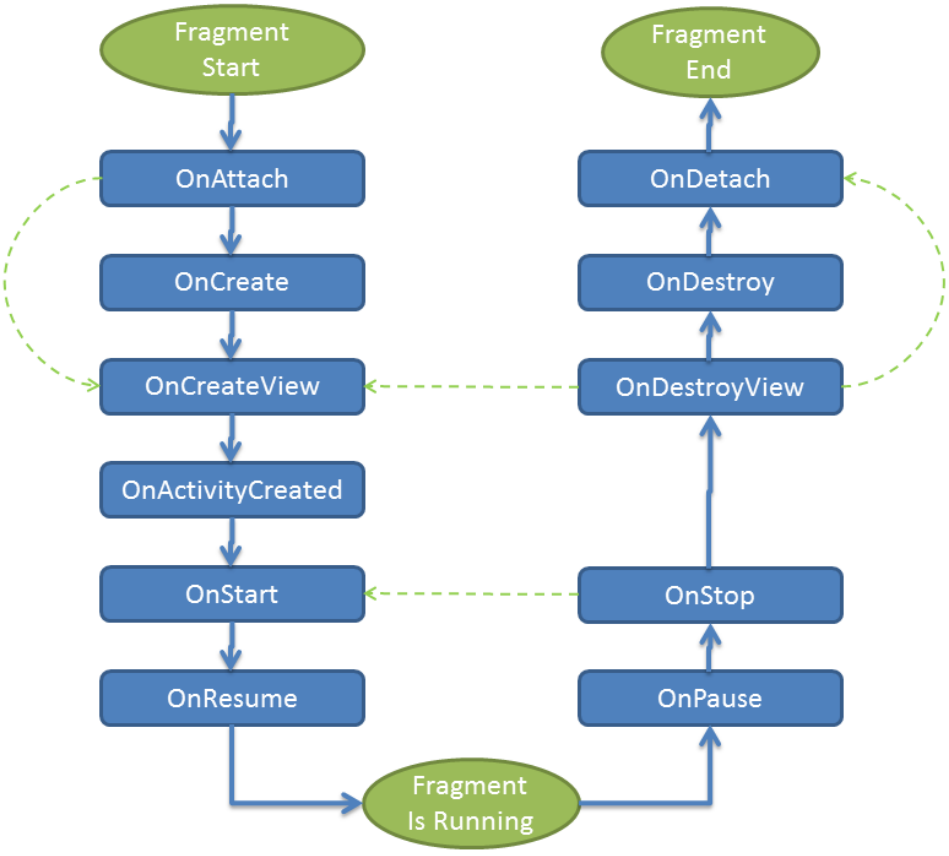


Figura I.7. Fragment Life cycle en Android.

Los métodos más importantes que debemos sobreescribir del *lifecycle* del *Fragment* son:

- onAttach(): ocurre cuando el *fragment* es vinculado (añadido) a la actividad.
- *onCreateView():* es el método que se sobreescribe para inflar los componentes visuales y realizar operaciones iniciales.
- *onResume():* este método ocurre cada vez que se termina de pintar (o reaparece) el fragment en pantalla. Por ejemplo al apagar y encender la pantalla ocurre este evento de la actividad que estuviese activa.
- *onPause():* es justo el opuesto del método *onResume.*
- *onDestroy()*: este método ocurre cuando el sistema destruye totalmente el fragment.

Finalmente en la Figura I.8 se muestra la correspondencia de los ciclos de vida de una *Activity* con el ciclo de vida del *Fragment*.

I.6 *BroadcastReceivers* y *Services*

Como ya hemos explicado anteriormente en *Android existen diferentes tipos componentes* que van a ayudar a realizar nuestra aplicación Android.

De estos componentes Android que usamos en el PFC ya hemos visto en este anexo las *activities* y los *fragments* que se usan principalmente para representar la parte visual. Ahora presentamos otros dos componentes muy importantes los *services* y los *broadcast receivers* dichos componentes forman una parte importante del *Core* en nuestra aplicación y dentro del modelo MVC formarían parte del Controlador y el Modelo.

Broadcast Receivers

Los *Broadcast Receiver* son un componente Android que permite recibir eventos del sistema o eventos propios. En otras palabras los *receivers* despiertan a nuestra aplicación cuando ocurra el evento solicitado, tiene un funcionamiento similar a las interrupciones en sistemas operativos pero con la ventaja de que no hay interrupción (Figura I.9).

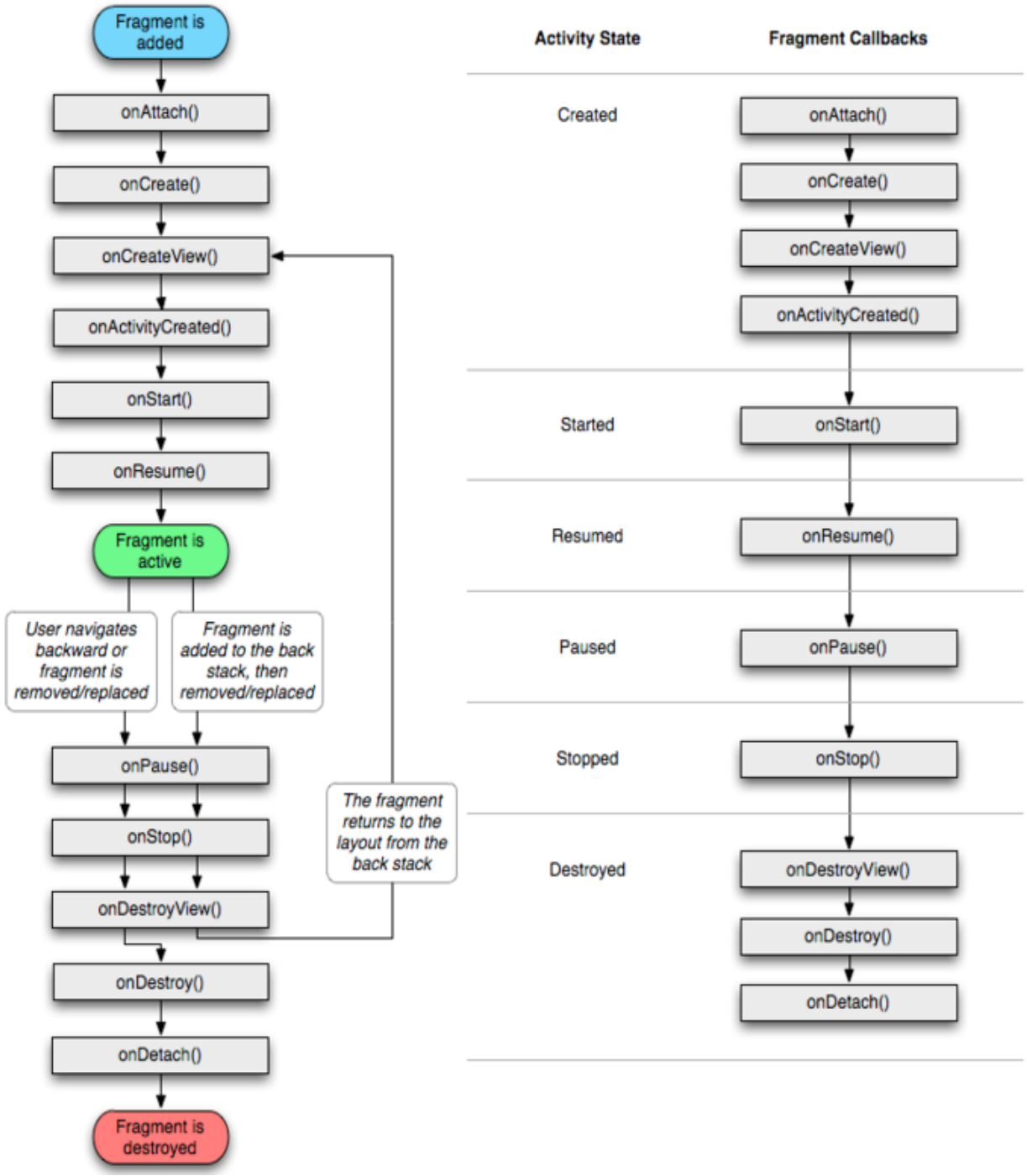


Figura I.8. Correspondencia de los ciclos de vida de *Activity* y *Fragment*.

El proceso es bastante sencillo, se crea una clase que extienda de la clase *BroadcastReceiver* y se registra dicho receiver en el archivo *manifest* de la aplicación o en alguna parte del código fuente que queramos (mediante *IntentFilters)* especificando los eventos que queremos escuchar.

Podemos escuchar prácticamente cualquier evento de sistema, desde el apagado de la pantalla, apagado del teléfono móvil, pasando por la recepción de una llamada o la recepción de un SMS.

Aunque en nuestra aplicación tenemos varios eventos *BroadcastReceivers* probablemente el más importante es el que se define en la clase *SMSReceiver* y obviamente es el encargado de escuchar los mensajes de texto recibidos y decidir si es o no un posible *SMS Activador* válido.

Services

Los servicios como ya comentamos anteriormente también son un componente de aplicación Android bastante importante, éstos permiten ejecutar operaciones en segundo plano. Los servicios son perfectos y están diseñados para realizar operaciones largas o pesadas en segundo plano, aunque se pueden utilizar con cualquier propósito.

La principal ventaja de los servicios es que aunque no haya ningún componente visual activo de nuestra aplicación los servicios se podan seguir ejecutando en segundo plano sin ningún problema y sin ningún indicador visual de su ejecución. Podemos observar el ciclo de vida de los servicios en Android en la Figura I.10.

En nuestro proyecto la clase *MainService* implementa nuestro servicio principal, dicho servicio como ya sabemos es el encargado de analizar el resultado de un *SMS Activador*, y activar/ejecutar los diferentes módulos necesarias además de gestionar los servicios de geolocalización.

Cabe decir que no confundir un *Service* con un *Thread*. Un *thread* es otro hilo de ejecución que no afecta al hilo principal o *UI Thread*, en cambio un *Service* es un componente Android que permite ejecutar código en segundo plano, pero dicho código salgo que se haga lo contrario se ejecuta en el UI Thread principal de nuestra aplicación independientemente de que hayan o no componentes visuales activos de nuestra aplicación.

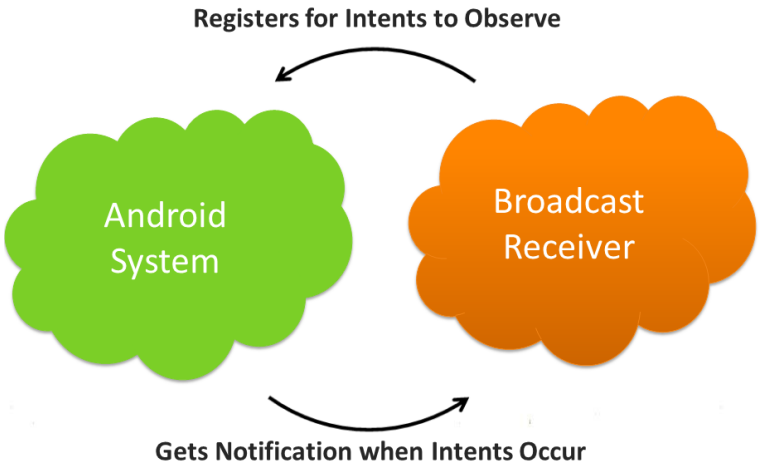


Figura I.9. Proceso de comunicación de un *BroadcastReceiver* con el sistema Android.

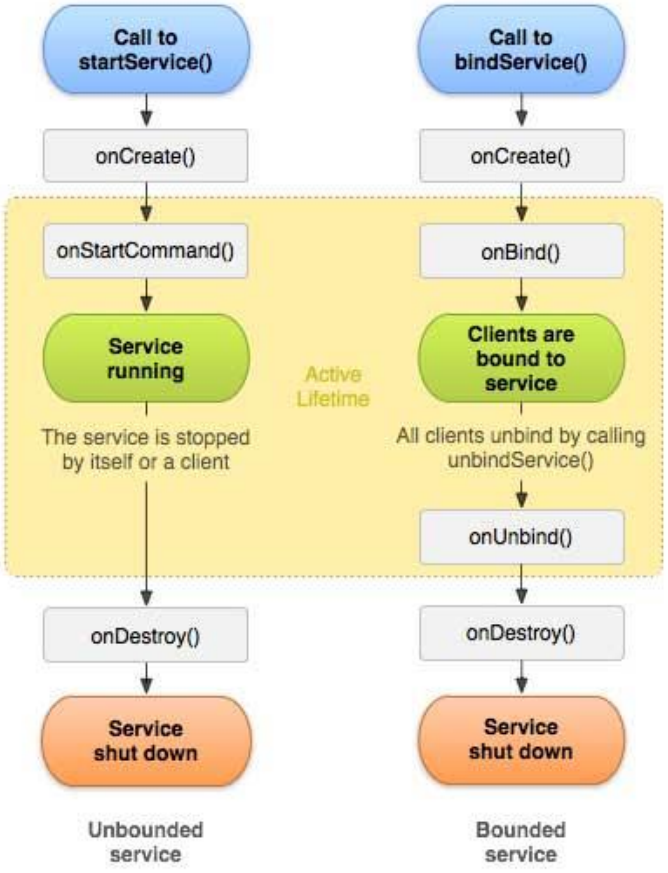


Figura I.10. Ciclo de vida de los *Services* en Android.

Bibliografía

[1] StackOverflow - www.stackoverflow.com

[2] Android-SDK - http://developer.android.com/sdk/

[3] Android-Developers - https://groups.google.com/forum/#!forum/Android-developers

[4] Androideity - http://androideity.com/

[5] Wikipedia - http://www.wikipedia.org/

[6] Libro, ProAndroid 2 - *by Sayed Hashimi, S. Komatineni and Dave MacLean (Marzo 2010, Apress)*

[7] Libro, Profesional Android 4, Application Development - *by Reto Meier (Mayo 2012, Wrox)*

[8] Libro, Pro Git - *by Scott Chacon (Agosto 2009, Apress)*

[9] Recursos Android - http://developer.android.com/guide/topics/resources/providing-resources.html

[10] Eclipse - www.eclipse.org/downloads

[11] *Google Device Manager* - https://www.google.com/android/devicemanager

[12] Bitbucket – www.bitbucket.org

[13] Genymotion – www.genymotion.com

[14] Android Studio – https://developer.android.com/sdk/installing/studio.html

[15] Eclipse Google Edition – http://developer.android.com/sdk/index.html

[16] Git – http://git-scm.com/

Las páginas webs anteriores están activas a fecha de julio de 2014.