ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Mejora de la aplicación social en Android por Bluetooth Meet Me (versión 2.0)

AUTOR: Francisco José Martínez García
DIRECTOR(ES): Juan Carlos Sánchez Aarnoutse

Septiembre / **2013**

Autor	Francisco José Martínez García
E-mail del Autor	fcojosemartinez@gmail.com
Director(es)	Juan Carlos Sánchez Aarnoutse
E-mail del Director	juanc.sanchez@upct.es
Título del PFC	Mejora de la aplicación social en Android por Bluetooth Meet Me (versión 2.0)
Descriptores	Android, SQLite, Bluetooth, Intent, Social

Resumen

En este proyecto se ha desarrollado una aplicación en Android para interacción social de corto alcance por bluetooth.

La aplicación recoge los datos obtenidos de tu usuario para su interacción con otros usuarios que se comuniquen con él por bluetooth. Depende de los datos de cada usuario saca compatibilidades y las muestra en porcentajes.

El usuario se crea mediante unos test y se almacenan en una base de datos en SQLite para que estos sean persistentes.

Se han implementado algunas líneas futuras de la anterior aplicación, como los puntos:

3.4. Capturando a los gestores de archivos

3.5. Capturando imágenes con la cámara

El punto ***3.6. Localización*** interna es un extra que no estaba previsto en las líneas futuras, pero se ha implementado debido a lo interesante de la geolocalización interna.

Titulación	Grado en Ingeniería Telemática
Departamento	Tecnologías de la Información y Comunicaciones
Fecha de Presentación	Julio - 2013

Agradecimientos

Me gustaría mostrar mi agradecimiento, en primer lugar, a **mis padres** que siempre han estado ahí dándome facilidades y apoyándome. Ellos me criaron y educaron y gran parte de todo se lo debo a ellos.

A **mi novia**, por su eterna ayuda; siempre apoyando, animando y, sobretodo, confiando ciegamente en mí. Tiene una gran fe en mí y espero no defraudarla.

A **mi familia**, que ha contribuido con mis padres a cuidar de mí y su alegría siempre es aire fresco que revitaliza cuerpo y mente. Gracias por todo el cariño y amor dados.

A **mis compañeros de carrera**, haciendo este camino más ameno y fácil. Entre ellos están Francisco José Hernández, Jose Antonio Manrubia, Pascual Madrid… me dejo algunos nombres, pero esto sería eterno, y a los ya ingenieros Ángel Peñaranda, Jordi Carrasco, José Asensio, Rosa Liarte, Iván Pablo Avilés, Juan Agüera, Andrés Dolón y algunos más.

A **mis profesores** por todo lo enseñado, pero, en lo referente a programar, me gustaría dar las gracias a Cristina Vicente, por enseñarme bien Java, a Javier Vales, porque gracias a él hice un servidor web desde cero con el que aprendí infinidad de cosas, gracias por tu exigencia y flexibilidad y, sobretodo, a Juan Carlos Sánchez Aarnoutse por sus ideas, su apoyo, su ayuda, su disposición y algunas cosas que se escapan a las palabras.

Por último, pero no menos importante, una mención especial a todos **mis amigos** de la infancia, del colegio, del barrio, del instituto… he compartido infinidad de experiencias que me han ido puliendo en la vida. El camino es más importante que la meta, y ellos me han convertido en quien soy ahora. Gracias de corazón.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

1 Introducción

En esta introducción hablaremos de las motivaciones que han llevado a hacer este proyecto.

Se ha preferido hacer este proyecto en **Android** por su emergente mercado y su rápida extensión por todos los teléfonos móviles de última generación, además de la potencia que este sistema tiene, lo cual veremos más adelante.

Debido al éxito de la integración de las relaciones sociales en los ordenadores y móviles **Twitter**, **Facebook**, **Tuenti**, **Wassup**… se ha querido hacer una red social de corto alcance, por *bluetooth*, pero totalmente innovadora.

1.1 ¿Por qué bluetooth?

La tecnología bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Consumo y simplicidad

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, con una cobertura baja y basados en transceptores de bajo costo.

Gracias a este protocolo, los dispositivos que lo implementan pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión lo permite.

El objetivo de este proyecto es crear redes ad hoc de *bajo consumo* y la tecnología bluetooth es la única que puede proporcionar ambas cosas, ya que en los teléfonos móviles no se puede configurar la tarjeta wifi para modo ad hoc, además de su elevado consumo frente a bluetooth.

Usando una red wifi habría que hacerlo en modo Infraestructura y tener un router centralizador que simplemente reenvíe los mensajes a los distintos usuarios conectados o registrados en el router. Esta variante tiene varios inconvenientes indeseables: 1) centralizar las comunicaciones puede presentar un riesgo para la protección de los datos de los usuarios. 2) Requiere de una infraestructura adicional (router Wifi) que debe ser configurado por alguien. Además, este router limita la libertad geográfica de la aplicación puesto que únicamente podrá ser empleada cerca de uno de estos routers.

Por otro lado, otra posibilidad sería emplear la tecnología 3G con un servidor centralizador, en este caso cada usuario se tendría que identificar y agregar una serie de *"amigos"* y una serie de opciones que harían del proyecto un nuevo *"Facebook"*. El consumo de batería sería disparatado y el gasto económico también ya que los usuarios tendrían que utilizar la tarifa de datos. La idea es totalmente diferente. Se quiere buscar encuentros y compatibilidades con personas que estén en el entorno de cada uno, no se quiere buscar la compatibilidad entre una persona de Huesca y otra de Berlín sino entre una persona de Cartagena y otra que se encuentre con ella en un bar, una biblioteca…

Seguridad

Bluetooth es una tecnología muy segura hoy en día. Anteriormente había muchos ataques por bluetooth, pero eso era en las primeras versiones. A partir de Bluetooth 2.0 hay una seguridad casi infranqueable que permite que el bluetooth pueda estar conectado sin riesgo de que haya intrusión. Para que dos dispositivos bluetooth puedan interactuar hay que pasar por tres fases: **emparejamiento**, **autenticación** y **autorización**. También se cuenta con el **cifrado de datos**.

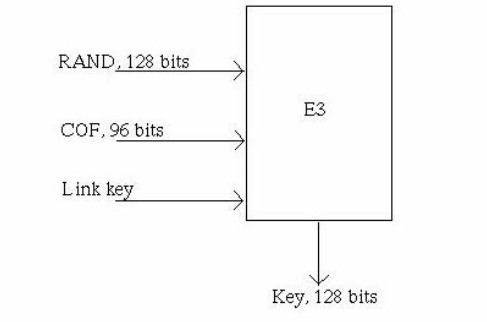


Figura 1. Cifrado de datos de bluetooth

El cifrado de datos protege la información que se transmite en un enlace entre dispositivos Bluetooth. Garantiza la confidencialidad del mensaje transmitido, de forma que si el paquete es capturado por un usuario que no posea la clave de descifrado, el mensaje le resultará ininteligible.

El maestro genera una clave de cifrado KC de 128 bits usando el **algoritmo E3**, el cual requiere como parámetros de entrada un número aleatorio de 128 bits, la clave de enlace ***Kab*** generada durante el procedimiento de emparejamiento y número ***COF*** (*Ciphering Offset*) de 96 bits basado en el valor temporal ***ACO*** (*Authenticated Ciphering Offset*) calculado durante el procedimiento de autenticación.

1.2 Bases de Datos

Android incorpora clases y librerías ***SQLite*** para manejar *Bases de Datos*.

SQLite es un proyecto de dominio público creado por **D. Richard Hipp**.

SQLite es una versión reducida de **SQL** (*Structured Query Language*), lo que significa que las bases tendrán un tamaño reducido, pero la potencia del manejo de la *query* se conserva. Es muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y, por supuesto, ser de código libre.

En Android, la forma típica para *crear*, *actualizar*, y *conectar* con una base de datos SQLite es a través de una clase auxiliar llamada **SQLiteOpenHelper** o, para ser más exactos, de una clase propia que derive (herede) de ella y que se debe personalizar para adaptarla a las necesidades concretas de la aplicación.

La clase **SQLiteOpenHelper** tiene tan sólo un constructor, que normalmente no será preciso sobrescribir, y dos métodos abstractos: ***onCreate()*** y ***onUpgrade()***. Estos métodos se deben personalizar con el código necesario para crear la base de datos propia y para actualizar su estructura respectivamente.

Las principales estructuras para inserción, actualización y eliminación de registros de una base de datos son, respectivamente, ***INSERT***, ***UPDATE*** y ***DELETE*** con el resto de la query detrás.

Para aprender el manejo de BBDD mediante código **SQL** (utilización de las *queries*) es recomendable la visita a *http://www.conclase.net/mysql/curso/index.php*.

Una de las características de SQL es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre ella. Es un lenguaje declarativo de "*alto nivel*" o "*de no procedimiento*", que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos. De esta forma una sola sentencia puede equivaler a uno o más programas que se utilizarían en

un lenguaje de bajo nivel orientado a registros. Así pues, con una *Query* simple (no muy extensa) se pueden modificar miles de registros en pocos segundos.

2 Android

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que *iOS*, *Symbian*, *Windows Mobile* y *Blackberry OS*. Un sistema operativo más se podría decir, pero ¿qué es lo que realmente lo diferencia de los demás? La respuesta es clara: Android está basado en ***Linux***, un núcleo de sistema operativo **libre**, **gratuito** y **multiplataforma**.

El sistema permite programar aplicaciones en una *variación de Java* llamada ***Dalvik***. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es **Java**.

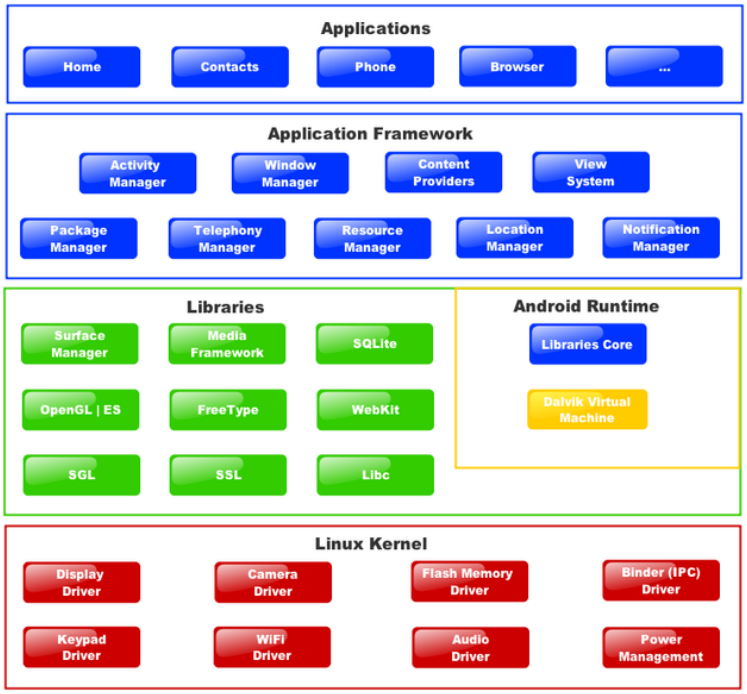


Figura 2. Arquitectura de Android

Como hemos dicho anteriormente, la *libertad* lo desmarca del resto de sistemas operativos. Ni para programar en este sistema ni para incluirlo en un teléfono hay que pagar nada. Y esto lo hace muy popular entre fabricantes y desarrolladores, ya que los costes para lanzar un teléfono o una aplicación son muy bajos.

Cualquiera puede bajarse el ***código fuente***, **inspeccionarlo**, **compilarlo** e incluso **cambiarlo**. Esto da seguridad a los usuarios, ya que algo que es abierto permite detectar fallos más rápidamente. Y también a los fabricantes, pues pueden adaptar mejor el sistema operativo a los terminales.

2.1 Un poco de historia

Android era un sistema operativo totalmente desconocido hasta que en 2005 Google lo compró. Hasta noviembre de 2007 sólo hubo rumores, pero en esa fecha se lanzó la **Open Handset Alliance**, que agrupaba a muchos fabricantes de teléfonos móviles, chipsets y Google y se proporcionó la primera versión de Android, junto con el SDK para que los programadores empezaran a crear sus aplicaciones para este sistema.

Aunque los inicios fueran un poco lentos, debido a que se lanzó antes el sistema operativo que el primer móvil, rápidamente se ha colocado como el sistema operativo de móviles más vendido del mundo, situación que se alcanzó en el último trimestre de 2010.

Figura 3. Android Cupcake

A mediados de mayo 2009, Google lanza la **versión 1.5** (versión de **Linux 2.6.27**) de Android OS (***Cupcake***) con su respectivo SDK que incluía nuevas características como: grabación de vídeo, soporte para estéreo Bluetooth, sistema de teclado personalizable en pantalla, reconocimiento de voz y el AppWidget framework que permitió que los desarrolladores puedan crear sus propios widgets para la página principal. Fue la versión que más personas utilizaron para iniciarse en Android y supuso el inicio de la implantación de Android OS.

Figura 4. Android Donut

Luego apareció **Android 1.6** (versión de **Linux 2.6.29**), llamada ***Donut***, con mejoras en las búsquedas, indicador de uso de batería y hasta el VPN control applet. De hecho, esta

versión fue tan buena que todos los Android que no tienen una interfaz personalizada como *HTC Sense* o *Motoblur* ahora corren 1.6.

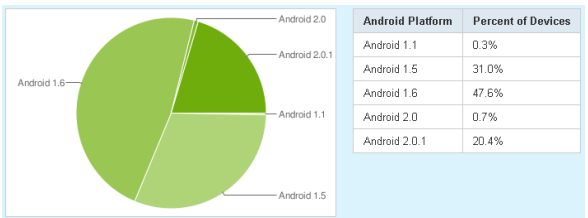
Figura 5. Android Eclair

Para llevar las cosas más allá, el Motorola Droid (Motorola Milestone para nosotros) fue lanzado con **Android 2.0** (versión de **Linux 2.6.29**), llamada ***Eclair***, que incluía varias características nuevas y hasta aplicaciones precargadas que requerían un hardware mucho más rápido que la generación anterior de móviles con **Android**.

Poco después, llegó con **Android 2.1** (el cual algunos llamaron "***Flan***" pero Google sigue considerándolo parte de "***Eclair***") con nuevas *capacidades 3D*, live wallpapers y lo que significó la gran mejora de la plataforma desde 1.6. De hecho, todos están pidiendo que les *actualicen* a **2.1** sus propios dispositivos con **Android** pero es probable que en algunos no funcionen bien todas las características de la versión.

Si miramos al futuro de la plataforma, por un lado **Android Market** es la tienda de aplicaciones que más crece ya llegando a las 40.000 aplicaciones, **Android** es el sistema operativo que más está creciendo en Estados Unidos casi por superar a iPhone, Motorola junto con algunos otros fabricantes están propulsando el desembarco en América Latina de **Android** con equipos económicos, y por el otro algunos se quejan de la fragmentación de la plataforma debido a las diferentes versiones pero lo cierto es que ya se está empezando a desarrollar el ***know how*** para brindar las actualizaciones a los usuarios 2.1 y en el futuro a las siguientes además de que la fragmentación es algo que en el mundo móvil es casi inevitable.

Veamos un cuadro que representa la utilización de las distintas versiones con datos obtenidos hasta Enero de 2010.



Android Platform	Percent of Devices
Android 1.1	0.3%
Android 1.5	31.0%
Android 1.6	47.6%
Android 2.0	0.7%
Android 2.0.1	20.4%

Figura 6. Porcentaje de versiones de Android

Después han salido dos versiones más: **Froyo** y **Gingerbread**.

Figura 7. Android Froyo

El flamante sistema operativo **Android 2.2** (versión de **Linux 2.6.32**)**,** denominado **"*Froyo*"** (**fro**zen **yo**ghourt) posiciona claramente a **Google** como competencia directa de **Apple**, junto con su nueva *tienda de música*.

Además, con ***Froyo***, podemos crear un *punto de acceso WiFi* a partir de nuestra *conexión 3G*, y como demostración, se utilizó para conectar a Internet un *iPad*… Esta funcionalidad se denomina ***Tethering***.

Pero no es ni mucho menos la mejora más destacada de **Android**. En el centro del sistema operativo ahora corre un **compilador Just-in-time** (*JIT*), que optimiza la ejecución de código mediante pre-compilación y cacheo de las instrucciones compiladas. El resultado es demoledor: un **incremento** brutal de la **velocidad de ejecución**, que ahora es de dos a cinco veces más rápida.

Mientras que en un Nexus One con Android 2.1 se obtienen de 6.5 a 7 MFLOPS (millones de operaciones de punto flotante por segundo), al actualizar a ***Froyo*** se alcanzan **37.6 MFLOPS**. Esta optimización es sorprendente.

El navegador de **Android 2.2** cuenta con la potencia del **motor JavaScript V8**, el mismo que trae integrado *Chrome*. Esto consigue una mejora de dos a tres veces en la velocidad de ejecución de código JavaScript.

Con una colaboración estrecha entre los desarrolladores de Adobe y los desarrolladores de Google, han llevado **Flash 10.1 a Froyo**. De nuevo el mensaje es claro: da igual como sea la web, tú no tendrás que preocuparte si estás con nosotros. Aparentemente, parece que en **Android 2.2** el soporte para vídeo es mejor con HTML5 que con Flash, pero desde Adobe avisan que el Flash 10.1 implementado por ahora es una beta y que carece de aceleración por hardware.

Figura 8. Android Gingerbread

La versión más actual es **Android 2.3** (versión de **Linux 2.6.33**), llamada ***Gingerbread***, y tiene varias mejoras respecto a ***Froyo***, pero solamente haré mención a dos de éstas. **Llamadas VoIP**, es necesario tener una cuenta SIP y esta función estará disponible dependiendo del fabricante de dispositivos y de la compañía de telecomunicaciones que suministre la señal. **NFC (Comunicaciones de Campo Cercano)**, la compatibilidad con ***NFC*** está incluida en el sistema operativo, aunque su uso *dependerá del hardware* del dispositivo.

2.2 Arquitectura

La arquitectura de **Android** se puede observar en la Figura 1 y a continuación se explicará cada uno de los grupos principales:

Aplicaciones

Todas las aplicaciones creadas para Android incluirán como base un cliente de email, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

Framework de Aplicaciones

Todos los desarrolladores de aplicaciones Android tienen acceso total al código fuente usado en las aplicaciones base. Esto ha sido diseñado de esta forma para que no se generen cientos de componentes de aplicaciones distintas, que respondan a la misma acción, dando la posibilidad de que los programas sean modificados o reemplazados por cualquier usuario sin tener que empezar a programar sus aplicaciones desde el principio.

Bibliotecas

Android incluye en su base de datos un set de bibliotecas C/C++ usadas por varios componentes del sistema Android. Estas características se exponen a los desarrolladores a través del Framework de Aplicaciones de Android, como System C Library, bibliotecas de medios, de gráficos, 3D, SQLite…

Runtime de Android

Cada aplicación Android corre su propio proceso, con su propia instancia de la Máquina Virtual Dalvik.

Núcleo Linux

Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, stack de red y modelo de controladores. El núcleo también actúa como capa de abstracción entre el hardware y el resto del stack de software.

2.3 Partes de una Aplicación Android

En este apartado se explicará un poco más a fondo la principal funcionalidad usada para el desarrollo de aplicaciones Android, que es la capa Framework.

En la creación de las aplicaciones tenemos cuatro clases que se pueden utilizar dependiendo del objetivo de éstas. La arquitectura de Android define cuatro elementos de los que podemos heredar:

Activity: Toda clase que como consecuencia de instanciarla implique una impresión por pantalla. Es comúnmente conocida como "*formulario*" y en una **Activity** se colocan los elementos de la interfaz gráfica. En un programa lector de ***RSS*** sería la pantalla donde lista los elementos nuevos por ejemplo.

Service: Un servicio sería todo proceso que corre sin necesidad de usar una interfaz gráfica. Es comúnmente conocido como "proceso" y seguirá corriendo aunque no haya ninguna interfaz gráfica que mostrar. Siguiendo con el ejemplo de antes, un servicio sería el proceso que se encarga de ir comprobando cada X tiempo si hay o no algo nuevo en el ***RSS***.

Intent: Es la interpretación abstracta de una acción. Es un mecanismo para comunicar las distintas aplicaciones y *Activities*. Semejante a un evento o interrupción. (Un click, pulsación en pantalla o pulsación de un botón).

Content Providers & Broadcast Receivers: Este es un aspecto novedoso y jugoso de Android. Está enfocado a la reutilización de código en una aplicación, por tanto una aplicación puede tener ciertos elementos que sean llamados por cuales quieran otras aplicaciones para que realice una acción.

La diferencia entre "***Content providers***" y "***Broadcast receivers***" es que los primero trabajan sobre ***URI***, es decir sobre tipos de datos *MIME*, y los segundos trabajan a nivel de "**Intent**". ***Content providers*** es el encargado de administrar la información que se pretende

que perdure. En el ejemplo, uno sería llamado cuando se encontrara el valor "*application/rss+xml*" y otro cuando se lanza un "**Intent**".

En el apartado 2.5 se verá la relación entre Activity y Service. Ya que con la combinación de ambas se puede enviar aplicaciones a "*correr en segundo plano*".

2.4 Ventajas de Android

Al estar basado en Linux, el Sistema Operativo de Android da muchas ventajas a favor si se compara con los competidores de telefonía. Algunas de las ventajas son las siguientes:

Sistema de Última Generación: Gracias a que Android puede instalarse teóricamente en todo tipo de dispositivos, sean teléfonos móviles, portátiles e incluso microondas, hace que Android siempre esté presente en los teléfonos más potentes del mundo, siendo una apuesta importante por fabricantes y operadoras por la posibilidad de que independientemente del potencial o prestaciones del dispositivo, Android podrá adaptarse a la perfección a todo tipo de necesidades. Por otra parte, otro tipo de sistemas operativos se ven obligados a estar rezagados a teléfonos más obsoletos o estar limitados a una determinada marca de fabricante.

Software Libre: Android está liberado con licencia Apache y es software libre, lo que lo convierte en un sistema operativo totalmente libre para que cualquier desarrollador no sólo pueda modificar su código, sino también mejorarlo. Se pueden publicar las nuevas mejoras y el nuevo código, ayudar a mejorar el sistema para futuras versiones sin depender de fabricantes o distribuidores. Al ser código abierto garantiza que, en caso de haber un error de programación, sea detectado y reparado con rapidez, al no existir ninguna traba legal para destripar el código interior, ni depender de alguien para pedir autorización.

Libertad: Android da completa libertad al propietario de un teléfono a instalar lo que desee, sea desde Android Market o un ejecutable aparte; no limitando la libertad del usuario, ni imponiendo software propietario para poder instalar música, archivos, documentos directamente desde el cable USB como si de un disco externo se tratara. La misma libertad tienen los desarrolladores o empresas pudiendo realizar aplicaciones o complementos como Flash, Opera o cualquier otro software sin tener que pedir permiso a nadie para ofrecerlo a los usuarios que libremente podrán instalarlo.

Sin fronteras: El desarrollo de Android no está apadrinado por fabricantes o proveedores. Android es libertad en todos los aspectos, y permite que todos puedan disfrutar de él siendo del operador que sea. Android no se reservará nunca el derecho a escoger una determinada operadora para imponer al usuario el hecho de contratarla para poder disfrutar de él así como sistema operativo que es, permite meterse en su código a través del SDK o desde el propio teléfono así como modificar su Firmware de manera extraoficial.

La Comunidad: Android no sólo cuenta con la comunidad más grande a nivel mundial de desarrolladores, sino también el mayor movimiento de éstos con multitud de

eventos, concursos, competencias y reuniones así como múltiples vías de comunicación como foros y chats oficiales para fomentar la participación y la colaboración para encontrar mejoras e ideas para futuras versiones. Por otro lado, las modificaciones o mejoras no dependerán de un limitado equipo de desarrolladores de una empresa sino que contarán con el apoyo, respaldo y participación de todos los desarrolladores del mundo.

Costos: Precisamente por el hecho de que Android puede ser instalado en teléfonos de cualquier fabricante o incluso en otros dispositivos, esto permite poder disfrutar de un número infinito de teléfonos de diferentes precios y tipos de precio sin tener que forzar o limitar un sistema operativo o teléfono a determinadas capacidades, dando la opción de que toda persona pueda adquirir el teléfono que más le guste.

Ahorro de Batería: Todos los teléfonos con Android instalado, deben tener siempre una batería extraíble dando la opción al usuario para poder sustituirla o llevar sistema de recarga en caso de que su batería se agote. De este modo se evita que el usuario tenga problemas en cuanto a su autonomía permitiendo que, aunque la batería haya acabado con su vida útil o considere que su batería es excesivamente pequeña pueda adquirir otra más potente o de reserva.

Personalizar: Al ser Software Abierto y Libre, Android es completamente "personalizable" tanto por usuarios instalando fondos de pantalla, animaciones, widgets y temas como para fabricantes con la posibilidad de crear sus propias capas como *MotoBlur* o *HTC Sense* permitiendo a unos y a otros poder cambiar o personalizar sus teléfonos de la mejor manera posible y dando a elegir al usuario la interfaz más adecuada para su gusto evitando imponer un determinado estilo o interfaz, al estilo Linux, por supuesto.

Multitarea: Android con su sistema de multitarea inteligente es capaz de gestionar varias aplicaciones abiertas a la vez dejando en suspensión aquellas que no se utilicen y cerrarlas en caso de resultar ya no necesarias, para la buena administración de la memoria.

No sólo en teléfonos móviles: Android ha hecho que en poco tiempo se implante en multitud de dispositivos electrónicos, desde teléfonos móviles hasta *notebooks*, *netbooks*, *microondas*, *lavadoras*, *marcos digitales*, *navegadores GPS*, *relojes* e incluso en *navegadores de abordo de coches*. Esto convierte a Android en un sistema operativo multifuncional, que garantizará su crecimiento y expansión así como ayudará a fabricantes a tener un sistema operativo inteligente para sus creaciones.

2.5 Ciclo de Vida de Android

Toda ***Activity*** sigue un ciclo, el paso entre estos estados se pueden deber a la ejecución de código o a la intervención del usuario. A continuación se verá que hay estados destinados a realizar algunas acciones y algunas que, simplemente, no usaremos nunca. Cabe decir que, pese a que no se introduzca nada de código en estos estados, la ***Activity*** sigue pasando por ellos. Los cambios de estado se pueden ver en las figuras 8 y 9, una mostrada a modo de diagrama de bloques y otra a modo de máquina de estados.

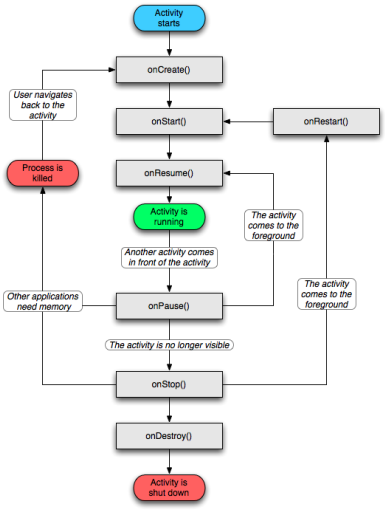


Figura 9. Ciclo de Vida (Flujograma)

Una ***Activity*** es una única y focalizada cosa que el usuario puede hacer. Casi todas las *Activity* interactúan con el usuario así que ellas mismas se encargan de crear una ventana para colocar la UI. Android maneja las *Activity* creadas como una pila, cuando una nueva *Activity* se crea, se coloca en lo más alto de la pila y se convierte en la *Activity* en curso. La *Activity* anterior permanece justo debajo y no volverá al frente hasta que la nueva *Activity* acabe. Existe una función que envía una *Activity* al final de la pila y la deja ahí para que el sistema la elimine en cuanto lo necesite, aunque la *Activity* sigue funcionando mientras no se destruya. Se puede devolver la prioridad a la *Activity* mediante el botón **HOME** (*pulsación larga*) y seleccionando la aplicación que está en "background". Esta función es ***moveTaskToBack****(boolean nonRoot)* y envía al final de la pila la tarea dependiendo de lo que se elija como argumento de entrada. Si es **true** se enviará cualquier *Activity*, si es **false** solamente se enviará si la *Activity* es el ROOT de una tarea.

A continuación se muetra una breve explicación de cada estado:

onCreate(): Se ejecuta cuando se crea la *Activity* por primera vez. Aquí es donde se deberían crear *views*, enlazar datos a listas, en definitiva el proceso de inicialización de nuestra aplicación.

onRestart(): Se ejecuta cuando la aplicación se ha cerrado y se va a ejecutar nuevamente.

onStart(): Se ejecuta cuando al aplicación aparece visible para el usuario. Si la aplicación es un proceso en segundo plano *(background)* el siguiente estado es ***onStop()***, si la aplicación se ejecuta en primer plano *(foreground)* el siguiente método es ***onResume()***.

onResume(): Se ejecuta cuando la *Activity* interactúa con el usuario. En este punto la *Activity* está en la cima de la pila.

onPause(): Se ejecuta cuando el sistema está a punto de continuar una *Activity* anterior. Se utiliza típicamente para guardar datos que no se han grabado anteriormente, parar animaciones y otras acciones que consuman CPU. Seguida por ***onResume()*** si la *Activity* vuelve a primer plano o por ***onStop()*** si es invisible para el usuario.

onStop(): Se ejecuta cuando la *Activity* deja de ser visible al usuario, porque otra *Activity* ha continuado y pasa a un lugar más prioritario de la pila. Puede ocurrir porque una nueva *Activity* ha sido creada, una *Activity* ya creada pasa a primer plano o ésta está siendo destruida. El siguiente método sería ***onRestart()*** si la *Activity* vuelve a interaccionar con el usuario o sería ***onDestroy()*** si la *Activity* es destruida.

onDestroy(): Última llamada antes de destruir la *Activity*. Puede ocurrir porque la *Activity* está acabando (llamada a ***finish()***) , o porque el sistema destruirá la instancia para guardar espacio. Se puede distinguir esos escenarios con el método ***isFinishing()***.

Debido a esto, la mejor manera de tener una aplicación en segundo plano es mediante la utilización de un **Service**. Un *proceso* servicio es un proceso que hospeda a un "***Service***" que ha sido inicializado con el método "***startService()***". Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son muy importantes (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos, entre otras), por lo tanto el sistema siempre tratará de mantener esos procesos corriendo a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano. En este proyecto se verá cómo se ha solventado el tema del funcionamiento en segundo plano recurriendo al ***Service***. A continuación se puede ver el esquema del ciclo de vida mediante máquinas de estados:

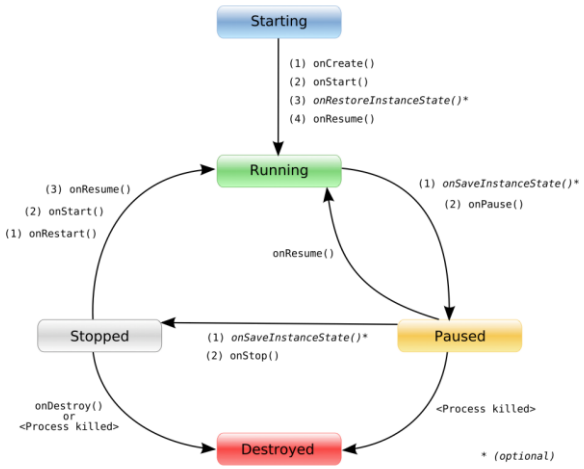


Figura 10. Ciclo de vida (máquina de estados)

2.6 Sistema de Archivos Android

Los proyectos de Android tienen un árbol de directorios y archivos estructurado y cada elemento tiene que estar en el lugar adecuado. Eclipse genera automáticamente esta estructura y será común a cualquier aplicación, sea cual sea su tamaño y complejidad. En la siguiente imagen se muestra esta estructura.

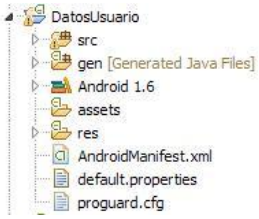


Figura 11. Estructura aplicación Android

A continuación se muestra una breve descripción de los elementos principales. Conociendo estos elementos se puede empezar a aprender Android.

/src/

Esta carpeta contiene todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc. Inicialmente, Eclipse creará por nosotros el código básico de la pantalla (*Activity*) principal de la aplicación, siempre bajo la estructura del paquete java definido. Aquí se añadirán todas las clases necesarias tales como más *Activities*, *Services*, *Threads*, *Objetos*…

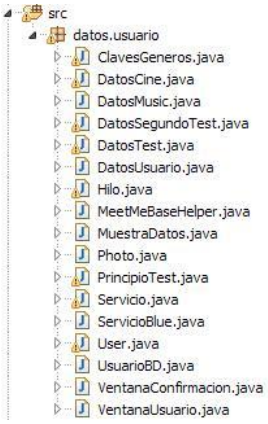


Figura 12. Estructura del directorio /src/

/res/

Esta carpeta contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, strings, etc. Los diferentes tipos de recursos de deberán distribuir entre las siguientes carpetas:

- **/res/drawable**: Contiene las imágenes de la aplicación. Se puede dividir en /drawable-ldpi/, /drawable-mdpi/ y /drawable-hdpi/ para utilizar diferentes recursos dependiendo de la resolución del dispositivo.
- **/res/layout**: Contiene los ficheros de definición de las diferentes pantallas de la interfaz gráfica. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación del dispositivo.
- **/res/anim**: Contiene la definición de las animaciones utilizadas por la aplicación.
- **/res/menu**: Contiene la definición de los menús de la aplicación.
- **/res/values**: Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
- **/res/xml**: Contiene los ficheros XML utilizados por la aplicación que no se incluyan en el resto de carpetas de recrusos, tal como se muestra en la **figura 12**.
- **/res/raw**: Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos (ver **figura 12**).

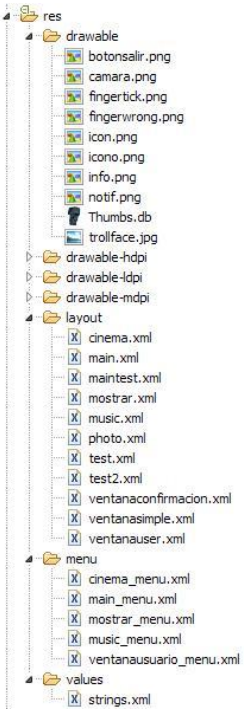


Figura 13. Estructura del directorio /res/

/gen/

Esta carpeta contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que se genera el proyecto, la maquinaria de compilación de Android genera automáticamente una serie de archivos fuente dirigidos al control de los recursos de la aplicación. Estos archivos van en **Java**. El más importante es el archivo *R.java* con su clase **R**.

Esta **clase R** contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta */res/*, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato. Así, por ejemplo, la constante *R.drawable.icon* contendrá el ID de la imagen "*icon.png*" contenida en la carpeta */res/drawable/*. Su gran utilidad es a la hora de acceder a las *Views* de las *activities*, si queremos obtener el control del botón btnSalir lo crearíamos referenciando desde el **código Java** el botón del **XML**:

```
Button bSalir = (Button) findViewById(R.id.btnSalir);
```

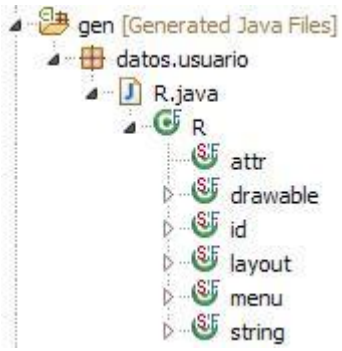


Figura 14. Estructura del directorio /gen/

/assets/

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

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

AndroidManifest.xml

Este archivo contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono…), sus componentes (pantallas, mensajes…), o los permisos necesarios para su ejecución. Android lleva unos permisos por defecto, pero en este archivo se deberán añadir los permisos necesarios para que el instalador le pida confirmación al usuario (como manipulación de la Wifi, Bluetooth, acceso al GPS, a la cámara…).

3 Proyecto Meet Me

En este capítulo se mostrarán aspectos importantes referentes al proyecto **Meet Me**.

Como hemos dicho anteriormente el proyecto se basa en una aplicación social de corto alcance por bluetooth. La aplicación hace cálculos de compatibilidad según la información obtenida tras una serie de preguntas en la creación del perfil.

Para el desarrollo de este proyecto se ha combinado la utilización de ***Activity*** con la de ***Service*** y para lanzarlo todo ha sido necesario utilizar ***Intent***. En la creación del usuario se ha utilizado una variable ***Bundle*** que ha ido recogiendo todo tipo de información (Activity tras Activity) que al final se ha volcado en una base de datos para tener los datos de forma persistente.

3.1 Bundle

El tipo de variable **Bundle** puede contener en su interior infinidad de variables (llamadas *Extras*) que se crean mediante una *KEY*. Se muestra un extracto de código para su mejor entendimiento:

```
//Creamos la información a pasar entre actividades
Bundle b = new Bundle();
b.putString("NOMBRE", nombre.getText().toString());
b.putInt("EDAD", age);
if(radioHombre.isChecked())
      b.putBoolean("HOMBRE", true);
else
      b.putBoolean("HOMBRE", false);
```

Código 1. Recopilación de datos con Bundle.

Como se puede ver se puede introducir cualquier tipo de variable (putInt, putString, putBoolean, putDouble…) donde el primer argumento es un String con el nombre de la variable (*KEY*) y el segundo es el valor. Tras la recopilación de datos con el Bundle ya se puede volcar la información en el Intent para pasarlo de una Activity a otra y así llevar la información ventana tras ventana. A continuación se puede ver un extracto para enviar dicha información:

```
//Creamos el Intent
Intent intent = new Intent(PrincipioTest.this, DatosMusic.class);

//Añadimos la información al intent
intent.putExtras(b);
```

```
//Iniciamos la nueva actividad
startActivity(intent);
```

Código 2. Envío del Bundle a otra Activity.

En la siguiente Activity se debe recoger el Bundle y continuar añadiéndole datos (si procede). Se puede observar en el siguiente código:

```
//Creamos una variable donde volcamos el Bundle anterior
Bundle b = this.getIntent().getExtras();

//Creamos el Intent
Intent intent = new Intent(DatosTest.this, DatosSegundoTest.class);

//Añadimos nueva información
b.putInt("EDADMIN", Integer.valueOf(minAge.getText().toString()));
b.putInt("EDADMAX", Integer.valueOf(maxAge.getText().toString()));
b.putInt("LOVE", barraLove.getProgress());

//Añadimos la información al intent
intent.putExtras(b);

//Iniciamos la nueva actividad
startActivity(intent);
```

Código 3. Manejo de un Bundle anterior.

De esta manera se puede recopilar información a lo largo de una serie de Activities, siempre y cuando se tenga claro que *KEYS* se han utilizado para no repetirlas.

3.2 Algunas Características Gráficas

A continuación se muestra el diagrama de bloques de la aplicación. La aplicación comienza comprobando si existe algún usuario ya creado, comprobando si existe el archivo de la base de datos **SQLite**. En caso de que exista carga toda la información (incluida la imagen) en la ventana principal del usuario, desde la cual ya existen todas las opciones para cambiar la foto, salir de la aplicación, activar el servicio bluetooth para el intercambio de información y dejar la aplicación corriendo en segundo plano.

A continuación, se puede apreciar cómo la aplicación se lanza al background con su servicio.

Figura 15. Meet Me en segundo plano

Aquí se puede ver la creación de las notificaciones que nos muestran los cambios de estado y es la opción para volver a la ventana principal al pulsar en la notificación, carga la última ventana mediante el uso de una variable del tipo **PendingIntent**.

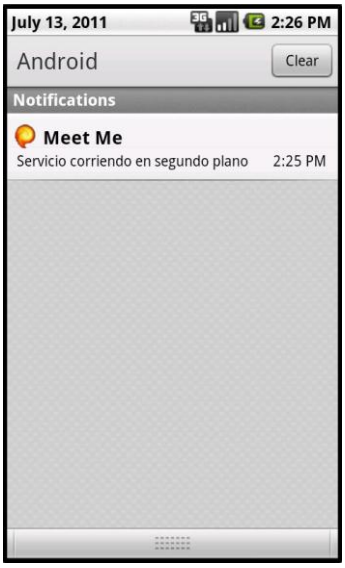


Figura 16. Notificaciones Meet Me

El primer test general presenta una interfaz bastante intuitiva debido a la inclusión de varios botones tipos **ImageButton** con pulgares hacia arriba o hacia abajo. Además, los valores de las preguntas se muestran con una **ProgressBar** que se decanta hacia un lado o hacia el otro.

Figura 17. ImageButton

Al principio de cada ventana (Activity) de creación del usuario se muestra una barra de progreso que indica el avance del test, para estimar el número de pasos que quedan por realizar. Aunque no tiene un número excesivo de pasos, en ocasiones el usuario puede abandonar el proceso antes de finalizarlo, bien porque tiene poca paciencia o bien porque tiene otros asuntos que atender, en esos casos la aplicación almacena la información que ha ido introduciendo.

Se muestra también la segunda parte de la ventana puesto que hay unos botones normales con texto (Button) con su barra de progreso que indica el nivel de "la balanza", pero que cambian de color según se decante hacia un lado o hacia otro. Cada extremo tiene la posibilidad de estar en 5 niveles: *equilibrio*, *a favor*, *muy a favor*, *en contra*, *muy en contra*. El nivel "*equilibrio*" carece de color, pero según avancemos "*a favor*" o "*en contra*" entramos en niveles de rojo y verde, apagado para "*a favor*" y "*en contra*" y vivo para "*muy a favor*" y "*muy en contra*". En la siguiente imagen se puede apreciar:

Figura 18. Color de los botones

La ventana principal del usuario muestra toda la información del perfil y la foto (escalada para mantener que no se salga de la pantalla y mantenga un buen aspecto respecto a la ventana). A continuación se muestra un fragmento de código del escalado de la imagen (donde *f* es el argumento de entrada de la función, es del tipo **ImageView**):

```
//REDIMENSIONAMOS LA IMAGEN
int width = myBitmap.getWidth();
int height = myBitmap.getHeight();

float ratio = (float) width / (float) height;

//Tamaño deseado
float newWidth = 130 * ratio;
float newHeight = 130;

// Calcular la escala
float scaleWidth = ((float) newWidth) / width;
float scaleHeight = ((float) newHeight) / height;

// Crear una matriz para la transformación
matrix = new Matrix();
Establecer la escala para el redimensionado
matrix.postScale(scaleWidth, scaleHeight);

// Crear el nuevo Bitmap
Bitmap resizedBitmap = Bitmap.createBitmap(myBitmap, 0, 0, width, height,
matrix, true);

f.setImageBitmap(resizedBitmap);
```

Código 4. Escalado de la imagen para ImageView.

Una vez cambiada la imagen se tiene que actualizar en la **base de datos**. La ventana donde se especifica la ruta de la foto nos devuelve la ruta en el extra "*PHOTO*" del **Bundle**. Para su actualización se utiliza el método execSQL(String query), cuyo argumento de entrada es la *query SQL* que se quiere ejecutar. Se ha utilizado este método puesto que no es una query "con resultado". Para hacer un **SELECT** se utilizaría otra función que devuelve una variable del tipo **Cursor**, donde con los índices se irían recorriendo los campos. Para su mayor comprensión se muestra el siguiente fragmento de código, donde *db* es una variable del tipo **SQLiteDataBase**:

```
addPhoto(foto);
dataBase = new File(PATH);
basehelper = new MeetMeBaseHelper(this, dbName, null, 1);
db = basehelper.getWritableDatabase();

db.execSQL("UPDATE " + basehelper.getNombreTabla() + " SET
photo='"+b.getString("PHOTO")+"'");

db.close();
db = null;
```

Código 5. Update de SQLite.

Si se pulsa la tecla "Menu" se mostrará el menú desplegable, con las opciones: **cambiar la foto**, **salir de la aplicación** (*detendrá el servicio* en caso de estar activo para así cerrar la aplicación totalmente), **enviar la aplicación a segundo plano** (*iniciará el servicio bluetooth* en caso de estar detenido) y **detener el servicio bluetooth** (útil para ahorro de batería y para hacer cambios en el perfil).

El uso del menú es muy útil para limpiar la ventana de botones y organizarlo todo de una manera más eficiente. Las ventanas de creación son activities largas, que se ayudan de un *scroll vertical*, y al final se centraliza todo desde una ventana que engloba mucha información, una imagen y botones para borrar el perfil. Puede resultar incómodo tener que estár utilizando el scroll para subir y bajar para moverte por los botones, por eso queda más elegante, cómodo y práctico establecer en algunas ventanas un menú emergente con el que puedas acceder de manera rápida a funciones importantes.

El menú se añade implementando las funciones ***onCreateOptionsMenu***(Menu menu) y ***onOptionsItemSelected***(MenuItem item).

A continuación se muestra una captura del menú emergente:

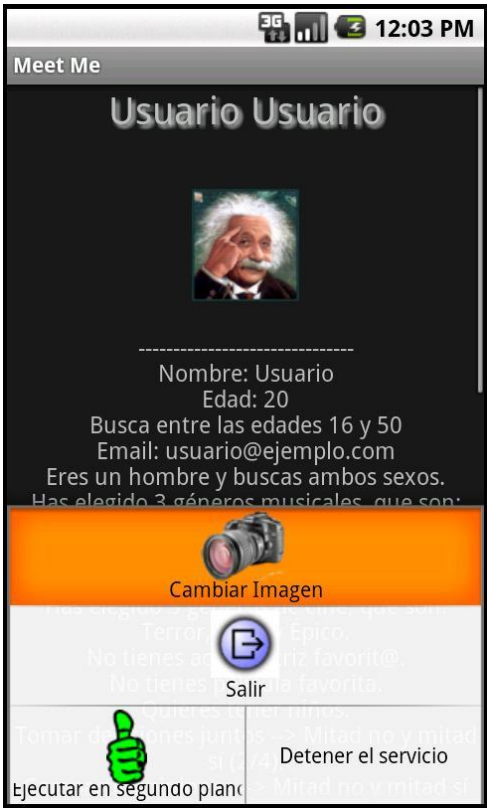


Figura 19. Menú desplegable

El botón ***Cambiar Imagen*** lanzará una activity rediseñada. Previamente se ofrecía un cuadro de texto para escribir el nombre de la imagen, exigiéndole estar en una carpeta concreta, para cambiar la foto de perfil. Actualmente está rediseñada según el punto **3.3**.

Bluetooth

No es gráfico, de hecho es un servicio en background transparente al usuario, pero merece la pena hacer alguna mención.

Se utilizan las clases **BluetoothSocket** y **BluetoothServerSocket** con sus canales de entrada y de salida **InputStream** (***getInputStream()***) y **OutputStream** (***getOutputStream()***). Estos canales se utilizan como cualquier canal de socket **TCP** o **UDP**, con sus métodos *write()*, *read()*, *flush()* … Las clases de socket de bluetooth también tienen sus métodos *accept()*, *connect()*, *close()* … En definitiva, se utilizan como si se utilizara un socket TCP o UDP, como se ha dicho anteriormente.

Quizá, lo más destacable sea las funciones que se han tenido que incorporar para enviar los objetos por dichos canales, puesto que la información se intercambia mediante los campos del objeto **User**. Este objeto es el utilizado para volcar toda la información de la Base de Datos SQLite. Una vez los datos se quedan de forma persistente, se cargan al comienzo de la aplicación y se vuelcan a un objeto User que tiene absolutamente todos los datos obtenidos de los tests. Para la comparación y su obtención de compatibilidad los dispositivos tienen que intercambiar estos datos por un canal de ***bytes []*** y sería excesivamente tedioso hacer un envío por cada campo, y son 65 campos. Así pues, el extremo que envía el objeto **serializa** el objeto y obtiene el array de bytes necesario para enviarlo por el *OutputStream*. El array se obtiene de la siguiente forma:

```
public byte [] serializar(Object o) {

	ByteArrayOutputStream bos = new ByteArrayOutputStream();

	try {
		ObjectOutput out = new ObjectOutputStream(bos);
		out.writeObject(o);
		out.close();
		byte[] buf = bos.toByteArray();
		return buf;
	}catch(IOException ioe) {
		Log.e("serializeObject", "error", ioe);
		return null;
	}

}
```

Código 6. Serialización del objeto.

En el otro extremo se obtiene el array de bytes a través del *InputStream*, y se **deserializa** para obtener el objeto y poder trabajar con él:

```
public Object deserializar(byte [] b) {

     ObjectInputStream in = new ObjectInputStream(new
     ByteArrayInputStream(b));

      try {
            Object object = in.readObject();
            in.close();
            return object;
      }catch(ClassNotFoundException cnfe) {
            Log.e("deserializeObject", "class not found error", cnfe);
            return null;
      }catch(IOException ioe) {
            Log.e("deserializeObject", "io error", ioe);
            return null;
      }

}
```

Código 7. Deserializar el objeto.

3.3 Diagrama de Bloques

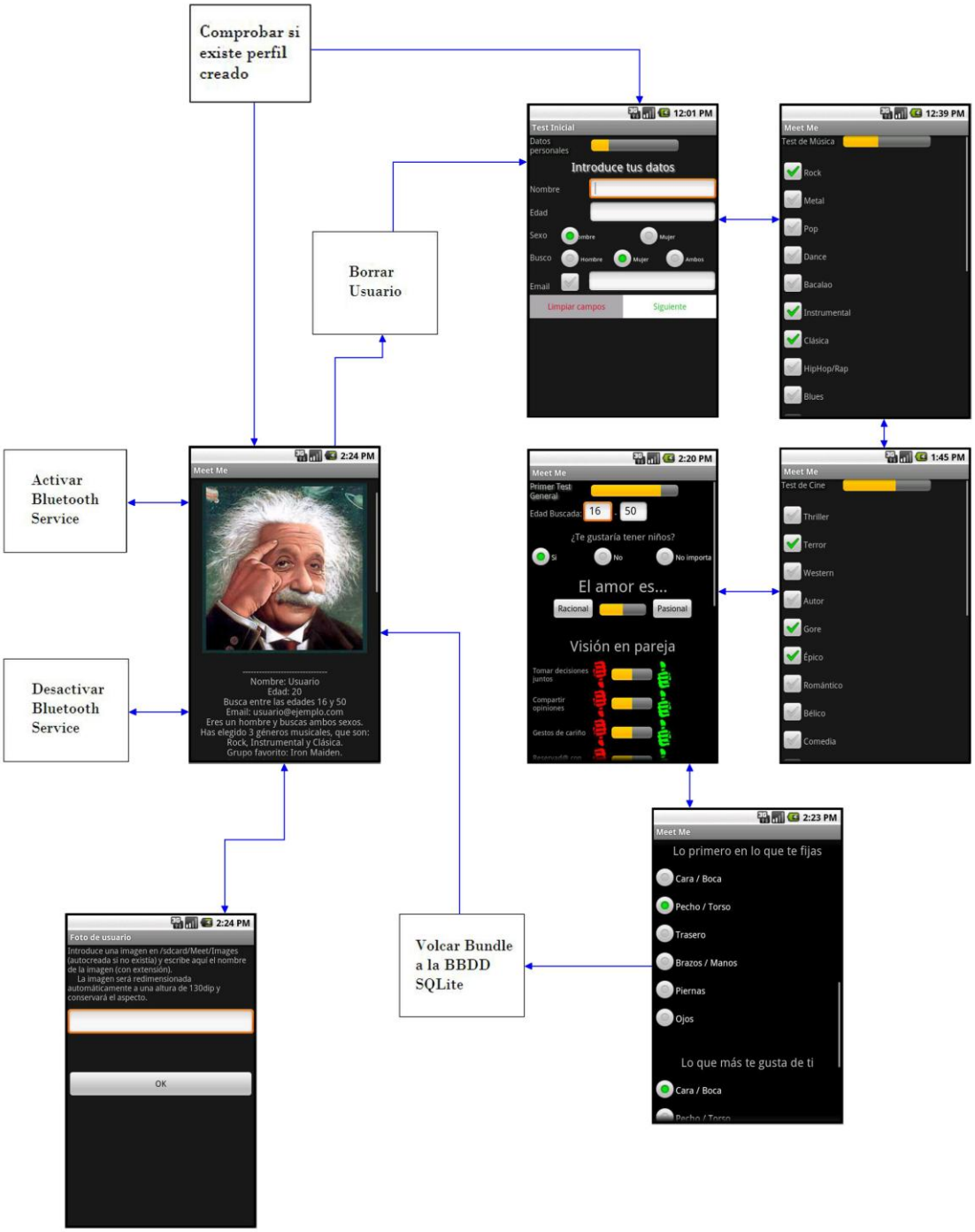


Figura 20. Diagrama de bloques

3.4 Capturando a los gestores de archivos

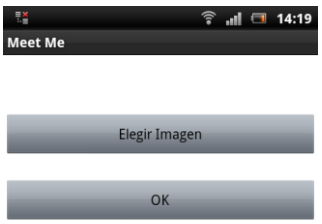


Figura 21. Ventana Cambiar Imagen

El botón ***Elegir Imagen*** lanzará un Intent que capturarán los gestores de archivos que tengamos instalados en el dispositivo. Si no hay ningún gestor de archivos instalado en nuestro dispositivo la función startActivityForResult(Intent intent) devolverá una excepción que será capturada y abrirá la galería para que procedamos a la elección de la imagen.

En la siguiente figura podemos ver como el Intent es capturado por los gestores del dispositivo. En este caso, hay dos instalados.

Figura 22. Elegir gestor de archivos

Una vez que abrimos con una aplicación (en este caso ES Explorador) podemos movernos por el sistema y elegir una imagen. La aplicación le devolverá a la activity la ruta de la imagen para configurarla como imagen de perfil.

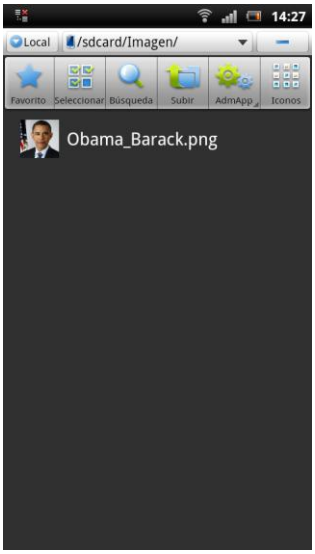


Figura 23. Elegir archivo

3.5 Capturando imágenes con la cámara

Ofrece también la posibilidad de capturar la imagen de perfil con la cámara del dispositivo móvil. Es un código muy simple, así que lo voy a añadir para que se vea el manejo del Intent, ya que ofrecerlo como figura es muy pobre al ver, simplemente, la cámara del dispositivo móvil, lo cual se puede ver abriendo la propia cámara desde el S.O.

```
//Al principio de la clase metemos estos campos finales
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private static final String CARPETA = "MeetMe/";

//Al pulsar el botón ejecutamos el siguiente código
try {
	Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
	fileUri;
	File photo = new File(Environment.getExternalStorageDirectory(),  CARPETA
	+ "PictureMeetMe.jpg");
	fileUri = Uri.fromFile(photo);
	intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
	startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}catch(Exception e) {
	Log.e("CAPTURAR", e.getMessage(), e);
}
```

Código 8. Captura con cámara.

3.6 Localización interna

Mediante cuatro dispositivos Bluetooth colocados en las esquinas del local, se podría representar un mapa de la localización de los diferentes usuarios (dispositivos) siguiendo la siguiente tabla de visibilidad.

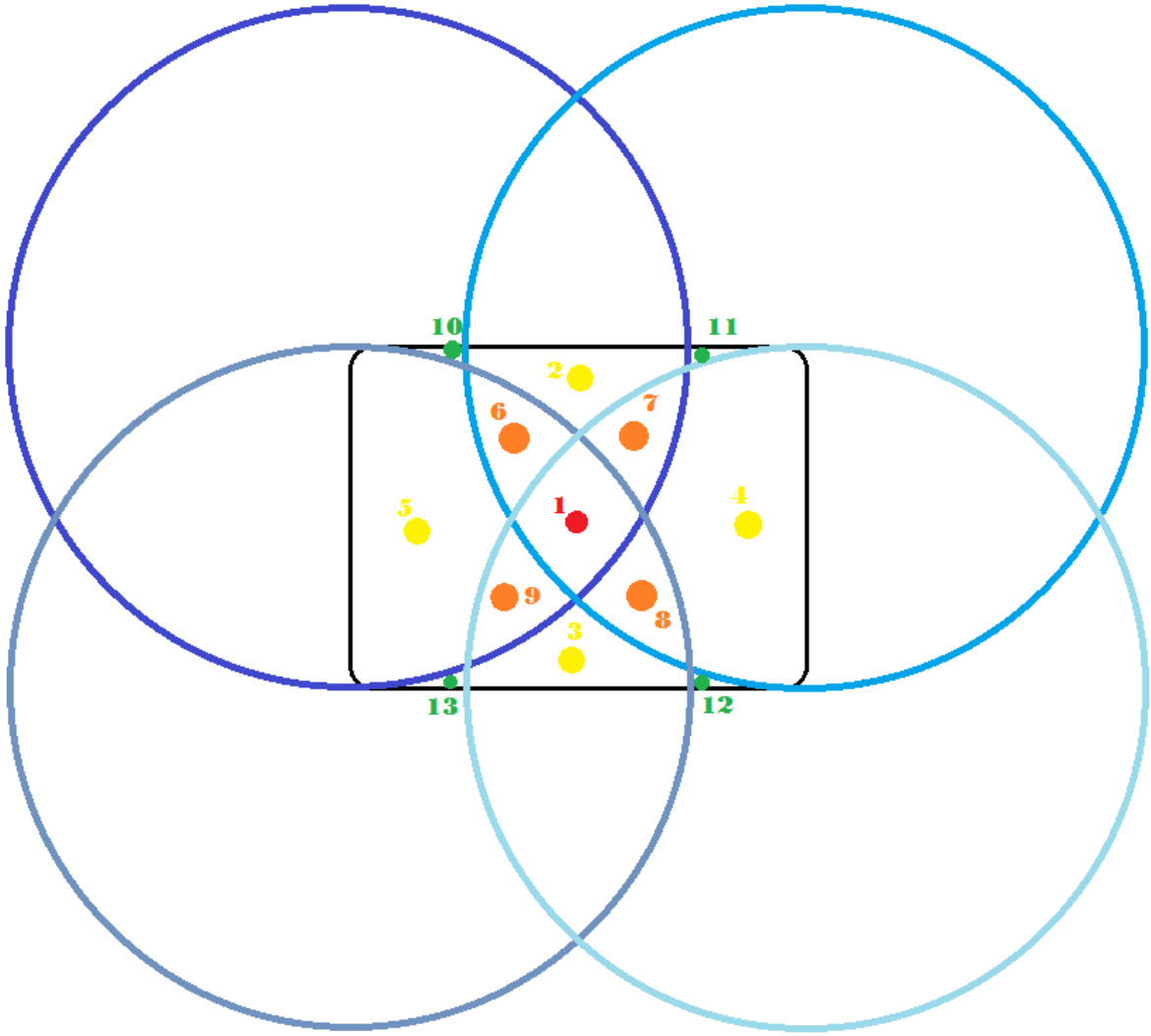


Figura 24. Repartición de puntos

Aquí se muestra la tabla con la repartición de los puntos, podemos alcanzar hasta 13 posiciones según la visibilidad, lo cual da una información bastante preciosa dentro de un local.

BT1	BT2	BT3	BT4	POSICIÓN
S	S	S	S	1
S	S	N	N	2
N	N	S	S	3
N	S	S	N	4
S	N	N	S	5
S	S	N	S	6
S	S	S	N	7
N	S	S	S	8
S	N	S	S	9
S	N	N	N	10
N	S	N	N	11
N	N	S	N	12
N	N	N	S	13

Tabla. Listado de puntos según la visibilidad de los nodos bluetooth

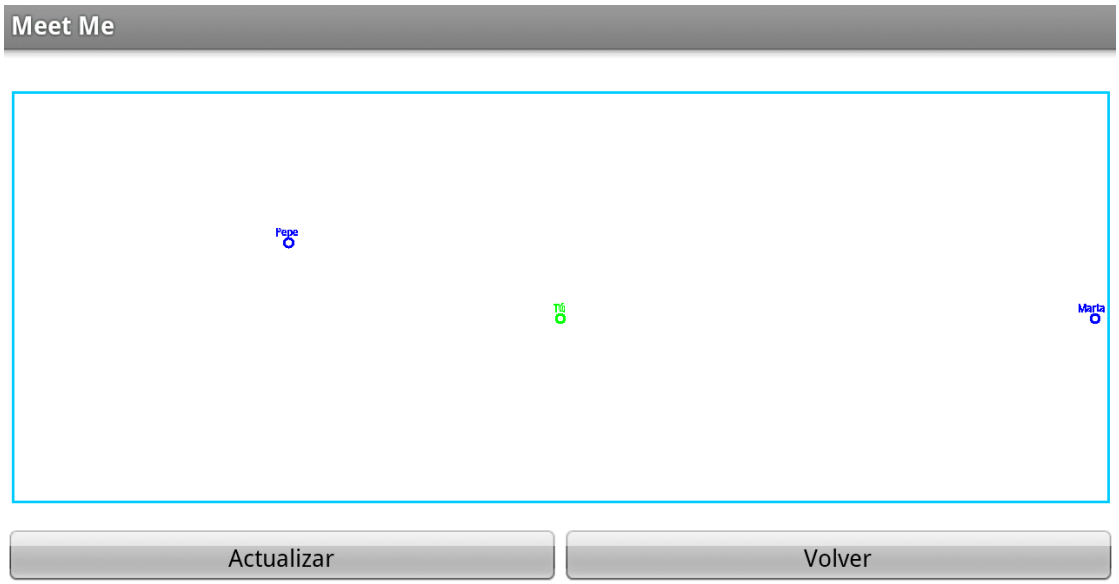


Figura 25. Dispositivos localizados

3.7 Diagrama UML

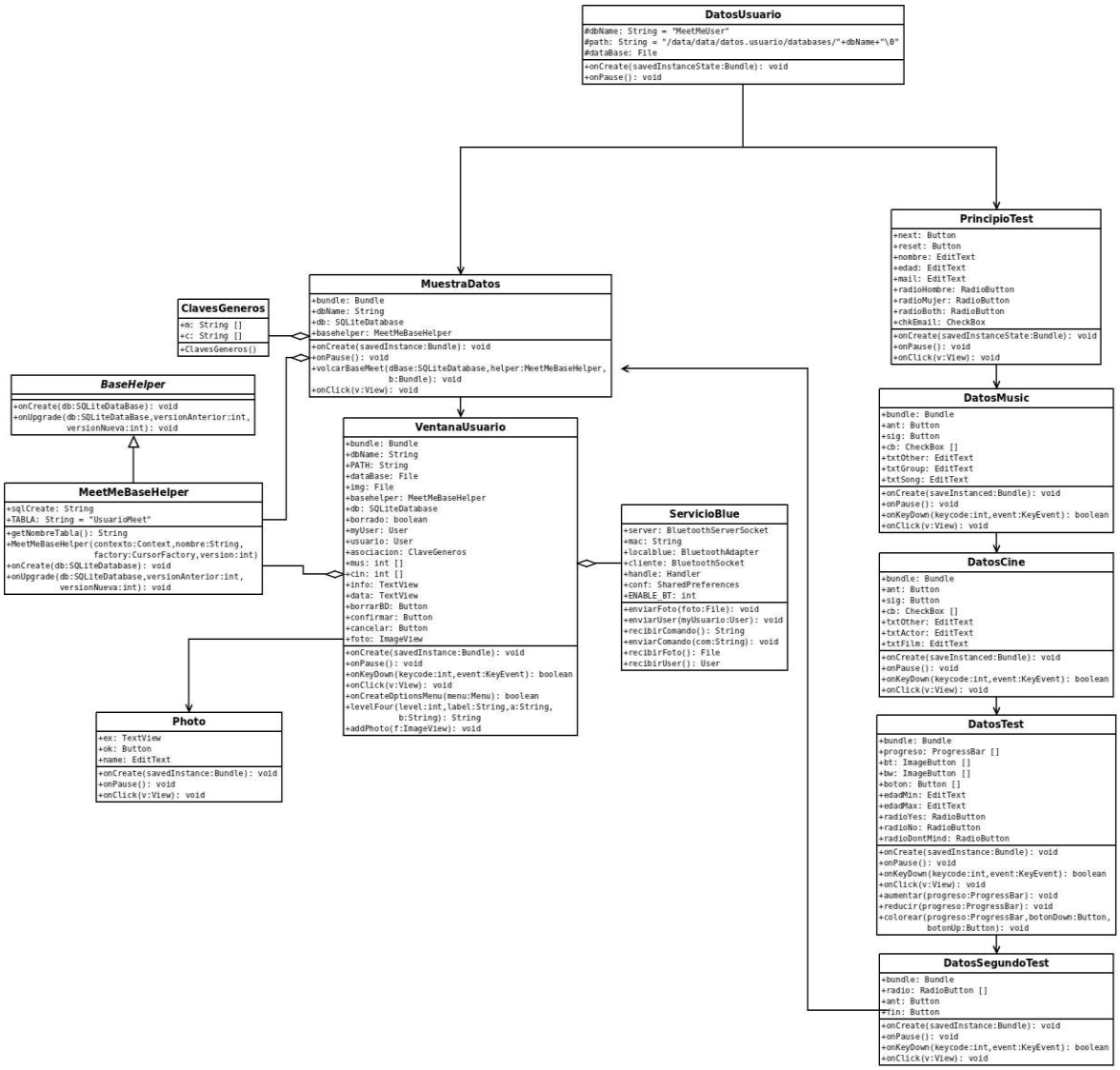


Figura 26. Diagrama UML

4 Conclusiones y Líneas Futuras

En este capítulo se obtendrán valoraciones actuales y potenciales mejoras para el futuro. Es un proyecto mejorable a largo plazo y, como red social, en caso de tener uso debería estar en continuo progreso.

4.1 Conclusiones

Este proyecto trata sobre la mejora de un innovador modelo de red social mucho más cercana, en todos los aspectos. Se aleja de la frialdad de tener "*amigos*" que nunca se han visto y que son reales desconocidos; en su lugar acerca a las personas que se tienen a su alcance.

Se ha utilizado la interconexión entre ventanas recopilando datos y luego volcándolos a una **base de datos**, la mejor manera de manejar datos persistentes. Estas ventanas se han ido cerrando (destruyendo la *Activity*) a medida que se ha avanzado para que no quedasen ventanas sueltas que estropeasen el funcionamiento de la aplicación, ya que volver a un estado anterior e innecesario puede interferir en el correcto funcionamiento de la aplicación.

Se ha creado una clase propia que hereda de la clase necesaria para el manejo de las bases de datos en Android. Esta clase se ha personalizado para su adaptación a este proyecto concreto, sobretodo el método ***onCreate()*** que mediante varios bucles forma una *query* de creación de base de datos que consta de 65 campos de diversos tipos. Se han utilizado adecuadamente las 3 sentencias claves de SQL: **INSERT**, **UPDATE** y **DELETE**.

Se ha utilizado la tecnología bluetooth para la comunicación debido a su bajo consumo, a su economía y por ser la única manera de conseguir una conexión ad hoc, ya que las tarjetas wifi de los móviles no permiten hacerlo, ya que para esta función disponen de bluetooth. También se puede explotar esta tecnología mediante una localización interna como se ha explicado en el punto 3.6.

Se ha utilizado la clase **Service** para lanzar un servicio, prioritario para el sistema a la hora de eliminar tareas, para que trabaje en *segundo plano* y tener la comunicación siempre disponible.

Se ha utilizado el mecanismo **Intent** que ofrece Android para utilizar la cámara desde el programa y para utilizar los gestores instalados para acceder a los archivos de nuestro teléfono.

4.2 Valoración personal

Ha sido costoso el comienzo en Android, ya que surgen pequeños problemas en la simple configuración del entorno y en la creación de proyectos que no se explican en los

sitios. Tardé mucho en hacer mi primer programa ya que tuve varios problemas con la **variable R** (la que se autogenera), que solventé suprimiendo los espacios de cualquier campo en la creación de un nuevo proyecto.

En la realización de este proyecto he detectado que el entorno de programación y simulación está muy "*verde*" y necesita mejorar mucho, aunque es bastante eficiente y fiable en aplicaciones locales. Sin embargo, es incapaz de simular conexiones bluetooth o wifi. A priori, herramientas como Netbeans y RealJ son más sencillas e intuitivas que Eclipse pero una vez te acostumbras se puede decir que Eclipse es una gran herramienta y muy útil para la programación de diversos lenguajes en general y de Android en particular. Quizás una de sus mayores ventajas es la ayuda con la importación de librerías y la forma en la que ayuda a resolver los típicos despistes de programación.

Me ha gustado iniciarme en la programación de Android puesto que es un nuevo lenguaje que está en alza y que, a priori, garantiza un largo futuro profesional ya que impera sobre los móviles del mercado actual y tiene una gran proyección en los del futuro (y no solamente móviles, como hemos visto en el apartado 2).

En mi opinión, Android adapta muy bien Java a las unidades móviles y es como debería haber sido J2ME (Java Micro Edition). Cierto es que los móviles de ahora son mucho más potentes que los de antes, pero obviando las limitaciones de arquitectura la forma de programar y usar el lenguaje es mucho más parecida a Java que lo que es J2ME.

La mayor parte de la ayuda la he encontrado en Internet, especialmente en www.android-spa.com y http://www.sgoliver.net. Hoy en día los lenguajes se exprimen más gracias a las comunidades de la red.

4.3 Líneas Futuras

En esta mejora se han solventado algunos puntos que quedaron para mejorar, como es el manejo de la cámara y la opción de recorrer el teléfono en busca de fotos ya realizadas con anterioridad para seleccionarlas como foto de perfil.

El tema de la localización interna no estaba planteado como mejora interna pero es un interesante añadido.

Debido a esto, puede surgir el planteamiento de alguna línea futura nueva, que se uniría a las que aún no han sido implementadas.

Chat

Un gran avance sería implementarle la función chat por bluetooth, así podrías contactar libremente con una persona tras ver la foto y el resultado de la compatibilidad. Esto daría mucho juego de cara a la interacción social, que al fin y al cabo es el objetivo de la aplicación**.**

Modificación máxima del perfil

Ahora mismo solamente actualiza la foto (ya que es lo que más se cambia). Pero sería necesario añadir un menú para modificar cualquier valor y así no tener que borrar el usuario y volverlo a hacer por cambiar un par de valores.

Edición de foto

Sería interesante que, una vez que se eligiera la imagen de perfil, se pudiese seleccionar el área que se quiere utilizar, y así poder elegir una foto de grupo y centrar la imagen en el área elegida, eliminando el resto.

5 Bibliografía

- http://es.wikipedia.org
- http://www.somoslibres.org
- http://www.sgoliver.net
- http://www.celularis.com
- http://rollanwar.net
- http://bicefalia.com
- http://www.lgblog.es
- http://developer.android.com
- http://mobile.davidocs.com
- http://celutron.blogspot.com
- http://liteapps.blogspot.com

En resumen:

- http://www.google.es