ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de una aplicación de cálculo de mapas de visibilidad radioeléctricos para dispositivos móviles móvil con sistema operativo Android.

AUTOR: Jesús Navarro Moreno
DIRECTOR: Leandro Juan Llácer
Diciembre / 2013

Índice

Índice de figuras

Autor: Jesús Navarro Moreno
E-mail: jesusnm_15_1991@hotmail.com

1. ¿Qué es Android?

1.1. Reseña histórica

El sistema operativo para smartphones más usado actualmente en todo el mundo no es una idea surgida de un día y tuvo un recorrido fácil, sino que surge de diferentes etapas y consecuencias hasta que el primer Android ve la luz en un dispositivo móvil.

En Octubre del año 2003, Andy Rubin, licenciado en Ciencias de la Computación en 1986, en el Utica College de Utica (New York) , Ritch Miner, Nick Sears y Chris White dieron forma a Android Inc. una compañía software ubicada en Palo Alto (California) que en sus inicios se dedicó al desarrollo de software para teléfonos móviles. Android Inc. se dedicó al desarrollo de un sistema operativo basado en el kernel de Linux para teléfonos móviles y tablets, al cual denominaron Android.

En Agosto de 2005, una empresa llamada Google que se estaba dedicando a reclutar 'startup' (empresas nuevas a las que se le prevé un futuro prometedor) adquirió la empresa cuando sólo contaba con 22 meses de vida.

La fecha clave desde la que se entiende el éxito de Android es el 5 de Noviembre de 2007. Ese día se fundaba la OHA (Open Handset Alliance), una alianza comercial de treinta y cinco empresas entre ellas empresas dedicadas a la fabricación de microchips, fabricantes de teléfonos móviles, operadoras, etc. Esta alianza que estaba liderada por Google dio a conocer Android que se presentaba con muchísima garantía al ser una plataforma de código abierto. El 12 de Noviembre de 2007 se lanzó una beta del SDK para desarrolladores.

Hoy en día la OHA cuenta con ochenta y cinco empresas entre las que se encuentran empresas importantes como HTC, Dell, Motorola, Intel, Samsung, LG, Nvidia, etc.

Pero no sería hasta un año después, en Octubre del 2008 cuando se pudo ver por primera vez funcionando en un HTC Dream en su versión 1.0. Veía la luz en USA un móvil con la primera versión final de Android. El modelo G1 de HTC pasará a la historia como el iniciador de este gigante llamado Android. [1]

Se presenta ahora a "Andy". Andy es el logotipo del sistema operativo Android del cual no se esperaba la gran aceptación y la repercusión que tuvo el susodicho robot verde y es que se ha convertido en la imagen de Google en cuanto a smartphones y tablets se refiere. Aunque existen teorías que indican que Andy está inspirado en R2D2 (de la famosa saga Stars Wars), lo cierto es que existe una teoría diferente mucho más creíble que sitúa a un robotito muy parecido en un videojuego de los años 90. La verdad es que las similitudes entre ambos son asombrosas y lo más fascinante es que los dos se llaman igual.

Figura 1.1. Logo Android

En aquel entonces la diseñadora encargada del proyecto era Irina Block. En unas declaraciones señala el por qué un robot verde con una forma tan curiosa. Explica que no hay antecedentes y que se necesitaba una relación entre el logo y el nombre del sistema operativo. En cuanto al color verde fue seleccionado porque recordaba al color de la nostalgia que destaca sobre el fondo oscuro.

El sistema operativo de Google tiene apenas 6 años y en este periodo le hemos visto evolucionar de una forma realmente impresionante, han sido muchas las versiones de este sistema operativo que han salido a la luz. La primera versión se basa sólo en hitos y su nombre es “Apple Pie”. A los tres meses, surgió la siguiente versión 1.1 que sirvió para corregir errores y se denominó “Petit Four” dedicado al gusto del product manager por ese tipo de pastel. La primera versión en utilizar nombres de postres sería 1.5 Cupcake, a partir de dicha versión cada versión siguiente llevaría un nombre de postre en orden alfabético como clave. Tanto es así que a Android 1.6 se consideró llamarla Donut y a Android 2.0 se le denominaría Eclair.

La verdadera revolución llegaría a partir de la versión 2.2 Froyo que se comercializó a mediados de 2010 y el Nexus One fue el primer dispositivo móvil en actualizarse a esta versión. Esta explosión del sistema operativo llegó a causa de mejoras muy importantes como imágenes en 3D al iniciar el dispositivo o un soporte para hotspot móvil Wifi (compartir la conexión 3G) algo que algunas compañías solo permitían a cambio de un pago extra. Las siguientes versiones son las ya conocidas hasta llegar a la actualidad en la que se encuentra la versión Jelly Bean. Se convirtió así en el sistema operativo principal en los smartphones gracias a ser gratuito y multiplataforma, es decir, que el sistema operativo puede usarse en distintas versiones de hardware y software. [2]

1.2. Sistema Operativo Android

Como ya se ha mencionado, Android se basa en el kernel de Linux pero estos dos sistemas no son iguales. Android no cuenta con un sistema nativo de ventanas de Linux ni tiene soporte para glibc (soporte de librerías para C); tampoco se pueden utilizar la mayoría de las aplicaciones de GNU de Linux.

Sin embargo, además de todo lo implementado en el Kernel de Linux, Android agrega algunas cosas específicas para dispositivos móviles como la comunicación entre procesos, la forma de gestionar la memoria compartida y la administración de energía del dispositivo.

La estructura del sistema operativo se distribuye en cinco capas muy bien diferenciadas, lo que le permite a cada capa obtener información de la capa inferior y a su vez enviar información a la capa de nivel superior.

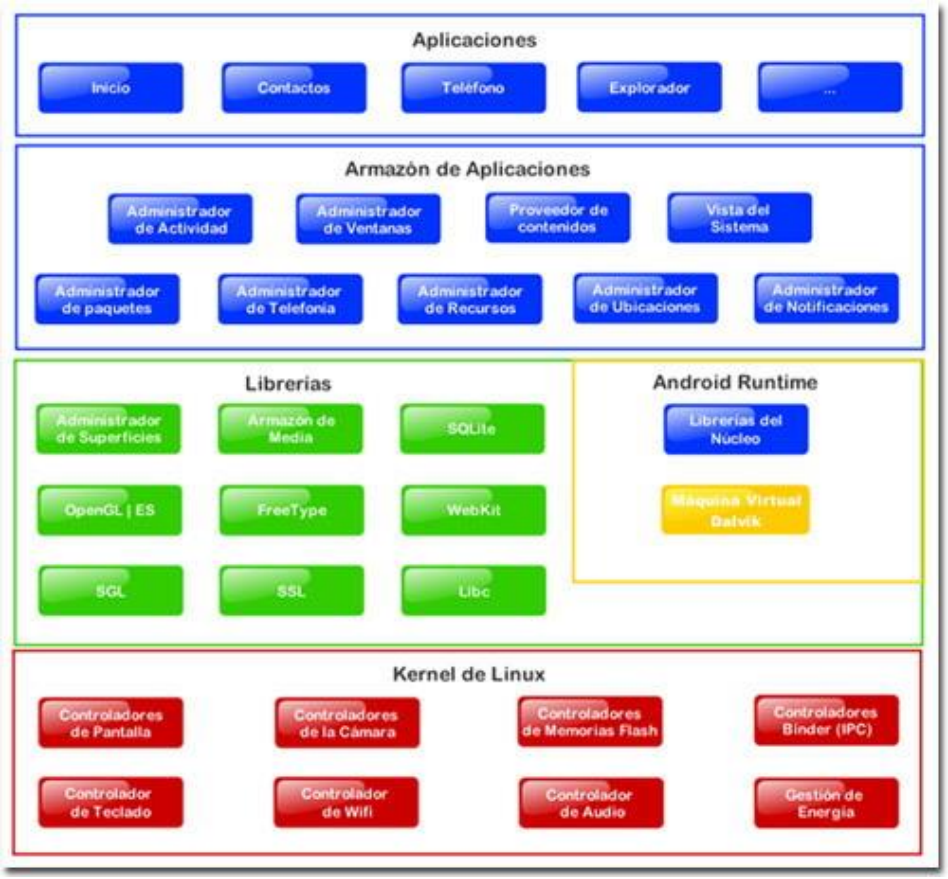

Figura 2. Estructura de Android

Se va a proceder a la explicación de las capas desde el nivel inferior hasta la capa de aplicaciones:

* **Kernel de Linux**. Es la capa de nivel más inferior, es decir, la más próxima al hardware del dispositivo. Se trata de una versión modificada del kernel de Linux 2.6, contiene los drivers necesarios para que cualquier elemento hardware pueda ser utilizado con las llamadas al sistema correspondientes.

* **Android Runtime** (Tiempo de ejecución de Android). Se encuentra al mismo nivel que las librerías de Android. Está constituido por las librerías del núcleo que son librerías donde se encuentran la mayoría de las funcionalidades disponibles por el lenguaje de programación Java y por la máquina virtual Dalvik, un intérprete de archivos ejecutables del tipo .dex que son óptimos para el almacenamiento eficiente de memoria. Cada aplicación corre en un proceso distinto de la máquina virtual Dalvik.

* **Librerías**. En esta capa se encuentran las librerías usadas por Android. Estas librerías han sido descritas en C/C++ y le proporcionan la mayor parte de sus características. Constituyen el corazón de Android junto con el Kernel de Linux. Entre las librerías más importantes se encuentran:

- LibC: Librería optimizada para usarla en dispositivos móviles. Incluye todas las funciones y cabeceras definidas según el lenguaje C.

- Librería Surface Manager: Es la encargada de componer los elementos de navegación de la pantalla del dispositivo. Gestiona las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.

- Librería OpenGL/SL: Son librerías gráficas, por tanto, sustentan la capacidad grafica de Android. Permiten incluso manejar gráficos en 3D y si el dispositivo lo permite se puede gestionar el hardware encargado de proporcionar dichos gráficos.

- Librería Media Libraries: Proporciona todos los códecs necesarios para el contenido multimedia soportado por Android.

- Librería SSL: Permite usar dicho protocolo para la creación de comunicaciones seguras.

- Librería WebKit: Motor para las aplicaciones de tipo navegador. Da forma al actual navegador incluido por defecto en la plataforma Android.

* **Armazón de aplicaciones**. Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para este sistema operativo utiliza el mismo conjunto de API y el mismo “framework” representado por esta capa.

* **Aplicaciones**. Es la última capa de Android e incluye todas las aplicaciones tanto las instaladas por defecto de Android como las que cualquier usuario vaya añadiendo posteriormente. Todas estas aplicaciones usas los servicios, las API y las librerías de todos los niveles anteriores.

2. Creación de un proyecto

2.1. Necesidades de instalación

Empezar a programar en Android es fácil y tiene los beneficios de código abierto. Para crear un entorno de desarrollo de aplicaciones para dicho sistema operativo debemos tener instalados en un ordenador tres herramientas:

* **JDK (Java Development Kit) de Java**. Es un software que provee herramientas de desarrollo para la creación de programas en el lenguaje de programación Java. Los programas más importantes que incluye son javac.exe y java.exe, compilador e intérprete de Java. Se puede descargar desde el enlace http://www.java.com/es/download/

* **Eclipse**. Es el famoso entorno de desarrollo integrado de código abierto multiplataforma para crear aplicaciones. Eclipse es usado como la principal herramienta de desarrollo para Java, pero también se puede instalar pluggins para otros lenguajes (como es el caso de Android). Se descarga desde su página oficial http://www.eclipse.org/downloads/

* **ADT (Android Development Tool)**. Es un pluggin especialmente diseñado para la IDE Eclipse. Al descargar este archivo se obtiene el SDK que son las API y las herramientas necesarias para crear, probar y depurar aplicaciones. Y por otro lado el ADT que servirá para poder crear los APKs a partir del /src de nuestro proyecto. El enlace de descarga es http://developer.android.com/sdk/index.html

Se deben descargar en instalar los archivos en el orden establecido porque cada uno necesita del anterior para su correcta instalación y funcionamiento. [3]

2.2. Estructura de carpetas y archivos

Se define en este apartado la estructura que tiene un proyecto de una aplicación Android y la funcionalidad de cada una de sus carpetas.

Cuando se crea un nuevo proyecto Android en Eclipse se genera automáticamente la estructura de carpetas necesaria para poder gestionar la aplicación. Esta estructura será común a cualquier aplicación, independientemente de su tamaño y complejidad. [4]

En la siguiente imagen se observa la estructura de un nuevo proyecto:

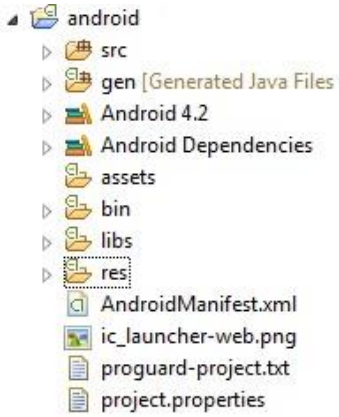

Figura 3. Estructura de carpetas

***Carpeta /src**. Esta carpeta contiene todo el código fuente de la aplicación, funcionalidad de cada una de las actividades y sus elementos y clases auxiliares necesarias para la ejecución de la aplicación, etc.

Figura 4. Carpeta /src

*__Carpeta /gen__. Esta carpeta es muy importante ya que contiene una serie de elementos de código generados automáticamente y destinados al control de los recursos de la aplicación. Lo más importante que contiene es la clase R, esta clase contiene una serie de constantes con los ID de todos los recursos de la aplicación (texto, imágenes, etc.) que se encuentran en la carpeta /res. Gracias a esta clase podemos acceder a nuestros recursos de forma muy sencilla en el código de la aplicación.

Figura 5. Carpeta /gen

* __Carpeta /res__. Contiene todos los recursos (imágenes, texto, etc.) que va a necesitar la aplicación. El contenido de esta carpeta se subdivide a su vez en otras que contendrán los elementos más específicamente. Algunas de ellas son:

- res/drawable: Contiene las imágenes que va a usar la aplicación. A su vez, se subdivide en varias carpetas dependiendo de la resolución del dispositivo en el que se encuentre dicha aplicación y es que Android permite crear aplicaciones versátiles según el móvil en el que funcione la aplicación:

  - drawable - ldpi (resolución baja).
  - drawable - mdpi (resolución media).
  - drawable – hdpi (resolución alta).
  - drawable – xhdpi (resolución muy alta).

- res/layout: Contiene los ficheros XML de las diferentes pantallas de la interfaz gráfica. Y es que las interfaces de usuario (GUI) se describen mediante este tipo de lenguaje de marcas y son llamadas a mostrarse en el código fuente de la aplicación. A su vez existe una subdivisión para los diseños de la pantalla en vertical y horizontal que son layout y layout-land respectivamente.

- res/menu: Aquí se sitúan los diferentes menus por los que pueda pasar la aplicación. También son ficheros que se describen en lenguaje XML.

- res/raw: Contiene recursos adicionales que no se incluyen el resto de carpetas del proyecto y vienen dadas en un lenguaje que no es XML.

- res/values: Otra carpeta donde se introducen otros ficheros XML con recursos de la aplicación, en este caso cadenas de texto (string.xml), diferentes estilos de la aplicación (styles.xml), colores (color.xml), etc.

Se debe mencionar que estas carpetas pueden no existir si estos recursos no son necesarios para la aplicación. Algunos de estos ficheros ni siquiera se crean inicialmente cuando se genera el proyecto y deben ser creados por los desarrolladores.

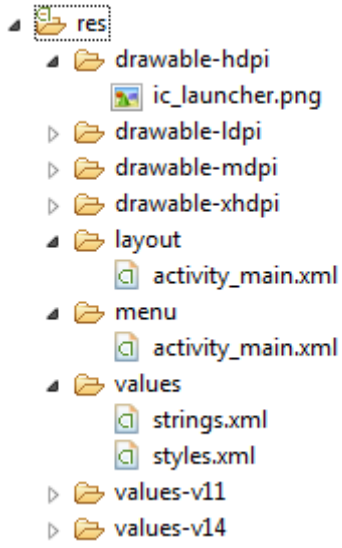

Figura 6. Carpeta /res

* **Carpeta /bin**. Esta carpeta, en principio, no se debe actualizar y debe dejarse tal y como esté. Contiene ficheros de compilación generados automáticamente y otros ficheros auxiliares. El fichero más importante que nos encontramos en esta carpeta es el .apk. Este fichero final que se crea al compilar la aplicación una vez terminada su realización, es el ejecutable de la aplicación que se instalará en los dispositivos móviles.

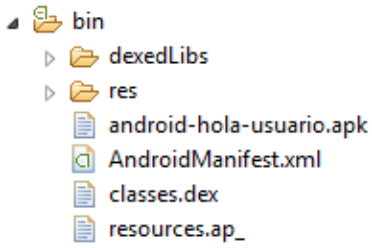

Figura 7. Carpeta /bin

* **Carpeta /assets**. Contiene otro tipo de recursos y ficheros auxiliares necesarios para la aplicación como, por ejemplo, ficheros de configuración, de datos, etc. Haciendo un poco de memoria, se observa que tiene la misma funcionalidad que la carpeta res/raw con la única salvedad de que los recursos guardados en esta carpeta no tienen un ID generado automáticamente por la clase R y, por tanto, debemos acceder a estos mediante su ruta como se haría con cualquier otro fichero del sistema.

* **Carpeta /libs**. Contiene las librerías auxiliares que se pueden usar en el desarrollo de la aplicación. Vienen dadas en formato .jar.

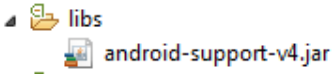

Figura 8. Carpeta /libs

* **AndroidMainfest.xml**. Es el fichero principal de configuración de la aplicación y viene descrito en lenguaje XML. Es muy importante que esté correcto, ordenado y actualizado para poder probar la aplicación y por supuesto para poder subirla a Google Play.

En primer lugar, indica el mínimo nivel de API que se requiere tener instalada en un dispositivo para que la aplicación se pueda descargar y ejecutar, el número de la versión de dicha aplicación y las librerías externas (como Google maps) que deben estar asociadas a la aplicación.

Lleva todos los aspectos más importantes de la aplicación, como la actividad principal que sería la primera que se muestra al usuario al iniciar la aplicación y la que primero se actualiza y se lee.

Incluye un listado de todas las actividades que se han desarrollado a lo largo del periodo de creación y componentes como contents providers, intent - filters, etc. Estas declaraciones indican al sistema Android bajo qué condiciones pueden ser cargadas las actividades.

Contiene el nombre de los paquetes Java que se encuentran en la aplicación, sirve de identificador único de cada uno de ellos.

También lleva otros detalles de configuración no menos importantes como son el icono de dicha aplicación y los permisos que necesita para poder funcionar, tanto los permisos para poder acceder a diferentes clases protegidas de una API o los permisos que tiene que tener para interactuar con otras aplicaciones o servicios.

A continuación se puede observar un pequeño ejemplo de un fichero AndroidMainfest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="es.tid.jala.android.ejemplos"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ActividadSMS"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".ReceptorSMS">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

Figura 9. AndroidMainfest.xml

Se aprecian las etiquetas que definen las funcionalidades como por ejemplo es el caso de:

- <uses-permission …/> que indican los permisos mencionados anteriormente.

- <application…/> que define la estructura de la aplicación.

- <acticity>…</activity> que definen cada una de las actividades.

- <intent-filter>…</intent-filter> que describen qué actividad es la principal y eventos que pueden ocurrir como la llegada de SMS en este ejemplo.

2.3. Elementos de una aplicación

Existen en Android una serie de elementos cuyo entendimiento resulta clave para poder desarrollar aplicaciones. Se va a realizar en este apartado una descripción sobre los elementos más importantes [5]:

* **Actividad (Activity).** Una aplicación Android estará formada por una serie de elementos básicos de visualización, estos elementos son las llamadas actividades y se conocen como pantallas de la aplicación. Se puede decir coloquialmente que una actividad es como un formulario o una pantalla.

Su principal función es la creación de la interfaz de usuario y puede haber tanto una única actividad como tantas se requieran pero todas deben descender de la clase Activity. Todas las actividades son individuales unas de otras, aunque podremos pasar de unas a otras con un elemento llamado “intent”.

* **Vistas (View)**. Las vistas estarían englobadas dentro de los elementos llamados actividades y son cada uno de los componentes que componen dicha actividad. Son, por tanto, los botones, las entradas de texto, checkbox, etc. que descienden de la clase View, lo que quiere decir que se pueden crear usando el lenguaje de programación Java y que su funcionalidad es descrita también por este lenguaje. Sin embargo, lo más usado en este tipo de programación es crear las vistas en XML dentro de los layout dejando que el sistema cree los objetos por nosotros.

* **Servicio (Service)**. Los servicios son componentes que se ejecutan en background, es decir, por detrás sin la necesidad de interactuar con el usuario. De hecho, este no tiene por qué tener conocimiento de su ejecución ya que no proporciona una interfaz para el usuario pudiendo lanzar algún mensaje o notificación en caso de que se precise de interacción con el usuario. Estos no dependen de una actividad así que pueden seguir ejecutándose aunque el usuario cambie de actividad o incluso puede acabar antes de que el usuario decida cambiarla. Los servicios deben estar definidos en el Mainfest.xml dentro de la actividad que lo vaya a lanzar. Tenemos dos tipos de servicios:

- Started: Este servicio se inicia cuando un componente de la aplicación lo llama haciendo uso de la función startService(). Una vez que este tipo de servicio ha comenzado puede durar en segundo plano de forma indefinida, incluso si el componente que lo creó es finalizado. Suele realizar una sola función y no suele devolver ningún resultado al componente que lo llamó; y cuando el servicio haya acabado este debería pararse.

- Bound: Un servicio es de este tipo cuando la aplicación que lo inicia lo hace llamando a la función bindService(). Ofrece una interfaz cliente - servidor que permite a los

componentes interactuar con el servicio, por ejemplo, pidiendo datos. A diferencia de los servicios “started”, este tipo de servicios corre solamente cuando el componente que lo llamó esté activo o en caso de que tenga varios “clientes”, hasta que no tenga ninguno activo.

* **Intención (Intent)**. Este tipo de componente activa tres de los componentes principales de cualquier aplicación Android que son actividad, servicios y receptores de difusión (broadcast receivers). Una intención representa la voluntad de realizar alguna acción y se utiliza cada vez que se quiera cambiar a una actividad diferente, cuando se necesite un servicio, etc. Se pueden también usar para compartir información entre componentes de una aplicación.

Las intenciones se pueden inicializar desde el sistema operativo o desde la aplicación. Las que genera el sistema suelen ser del tipo visualizar una página web o las llamadas telefónicas, en el caso de que una aplicación necesite una intención se genera usando el objeto “Intent”. Para el caso de la aplicación, existen diferentes mecanismos que responden para tipo de intent a distintos componentes. El más usado es el que juega con las actividades que se pueden lanzar con el método startActivity() aunque también pueden ser lanzadas para que devuelvan información a la actividad que lo llamó con startActivityForResult().

* **Receptor de anuncios (broadcast receiver)**. No es más que un tipo de componente que reacciona ante eventos broadcast como pueden ser recibir un SMS, una llamada o incluso alertas de la batería. También permite lanzar anuncios broadcast y aunque no tienen interfaz de usuario, permite lanzar algún tipo de actividad que sí interactúe con él.

***Proveedores de contenido (content provider).** Uno de los temas más importantes es la compartición de información entre los dispositivos móviles. Android tiene un mecanismo estándar mediante el cual las aplicaciones pueden compartir archivos sin necesidad de poner en peligro la seguridad del sistema de ficheros. Gracias a los proveedores de contenido podemos acceder a datos de otras aplicaciones como, por ejemplo, la lista de contactos.

2.4. Interfaz de usuario

En este capítulo se va a detallar los elementos gráficos más importantes de una aplicación Android así como los elementos que se han añadido durante el desarrollo de este proyecto. La interfaz de usuario (GUI) es el mecanismo mediante el cual un usuario puede comunicarse con el teléfono móvil. Deben ser fáciles de entender y fáciles de accionar. [5]

En Android esta interfaz de usuario se describe mediante "layouts" que no son más que archivos XML con la definición de todos y cada uno de los elementos de dicho layout. Los layouts son elementos no visuales que nos ayudan a controlar la distribución, tamaño y posición de los elementos que se insertan en su interior.

Estos componentes se extienden de la clase ViewGroup, los que se heredan de esta clase con componentes capaces de obtener a otros componentes en su interior. A continuación se detallan algunos de los tipos de layout:

* **FrameLayout**. Este tipo es el más simple de todos ellos. Un frameLayout coloca el elemento que se le inserte en la esquina superior izquierda de la pantalla del dispositivo; si hay varios entonces va colocando debajo del anterior y salvo que el primero tenga transparencia es imposible que podamos ver el segundo elemento. Por todo esto, suelen usarse como contenedores de un único elemento.

* **LinearLayout**. Es el siguiente en cuanto a complicación se refiere. Este tipo de layout organiza los elementos ordenados uno a continuación del otro según la orientación que se le indique. La orientación puede ser vertical u horizontal y se le indica en la propiedad Android:orientation.

* **TableLayout**. Organizan su contenido en forma de tabla según las filas y columnas que se le indiquen. Además permite organizar cada elemento en la posición que se le pida.

* **RelativeLayout**. Este permite organizar la posición de un elemento con respecto a un elemento padre que ha sido añadido anteriormente. Podemos indicarle que se coloque a izquierda o derecha, arriba o abajo, incluso combinaciones de ambos.

Ahora toca definir los Views y los ViewGroups que son los elementos que le colocan a estos layout y que son los que verdaderamente permiten interactuar con el usuario insertando texto, pulsando botones, etc.

En cuanto a los ViewGroups, son conjuntos de elementos Views. Los más importantes son:

* **ListView**. Permite visualizar una lista de elementos (items) que puede deslizarse verticalmente si todos no caben en una sola pantalla. La utilización es algo compleja, pero es muy potente.

Figura 10. ListView

* **GridView**. Esta vista permite visualizar los elementos en forma de cuadricula bidimensional y además seleccionar alguno de ellos. Se puede elegir el número de columnas que tendrá esta vista, pero para las filas dependen del número de ítems que el adaptador que estemos usando indique que se deben mostrar.

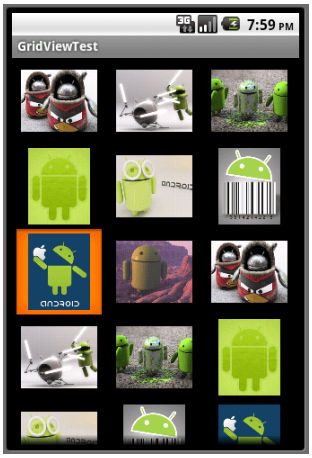

Figura 11. GridView

Por último, encontramos los Views (controles que permiten interactuar con el usuario) que se pueden combinar para darle el aspecto adecuado a la aplicación. Entre las más importantes se encuentran:

* **Botones (Button)**. Dibuja un botón en la pantalla para que el usuario lo pulse si quiere realizar alguna acción concreta. El SDK de Android pone a disposición tres tipos de botones que son el clásico botón (Button), el botón de encendido/apagado (ToogleButton) y el que contiene una imagen (ImageButton).

Figura 12. Botones

* **CheckBox**. Se suele utilizar para marcar o desmarcar opciones en una aplicación. Para hacer uso de ellos se usa la clase que tiene el mismo nombre, CheckBox.

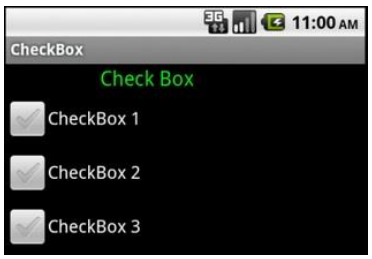

Figura 13. CheckBox

* **TextView**. Las etiquetas de texto se usan para mostrar un texto concreto al usuario. Se debe establecer el formato, el color del texto y del fondo, etc.

Figura 14. TextView

* **EditText**. Se encarga de la introducción y edición de texto, ya sea números o caracteres. También se le puede establecer el color de la letra y otros parámetros interesantes.

Figura 15. EditText

* **ImageView**. Este elemento permite mostrar imágenes en la pantalla de la aplicación. Dicha imagen debe estar incluida en el proyecto en la carpeta /res/drawable explicada anteriormente y se accede a ella mediante la propiedad android:src.

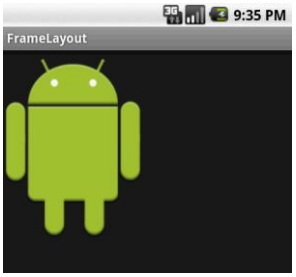

Figura 16. ImageView

2.5. Seguridad y firmado

En Android existen algunas acciones que pueden producir efectos negativos en el funcionamiento del dispositivo móvil, en las aplicaciones o incluso es archivos de datos. Es por ello que se dispone de mecanismos de control que determinan qué procesos pueden acceder a determinados recursos más restringidos como pueden ser obtener datos de localización, acceder a internet, llamar por teléfono, enviar SMS, etc. Se habla por tanto de dos mecanismos que ayudan a la seguridad que son los permisos y el firmado de las aplicaciones.

En cuanto a los permisos, Android define una serie de esquemas para poder proteger ciertos recursos y características especiales del hardware de cualquier acción negativa que ocurra en el dispositivo. Toda aplicación que necesite acceder a estos recursos está obligada a declarar esa intención en el fichero AndroidMainfest.xml de forma que al instalar la aplicación en un móvil, el usuario es consciente en todo momento de los permisos que le va a asignar a esa aplicación. En caso de que una aplicación intente acceder a un recurso del cual no ha solicitado el permiso, generará una excepción de permiso y la aplicación se cerrará inmediatamente. En la siguiente figura se observa un ejemplo de declaración de permisos en el fichero AndroidMainfest.xml.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="com.android.vending.CHECK_LICENSE"/>
```

Figura 17. Permisos

En la parte del firmado de las aplicaciones, todas las aplicaciones ya sea instaladas o subidas a Google Play están firmadas por el desarrollador con una clave de privada que posee. Esta clave existe para autenticar al creador y establecer las relaciones de confianza entre las aplicaciones.

La firma evita que un usuario cualquiera pueda hacerse pasar por el desarrollador e intentar modificar el funcionamiento de la aplicación. Se realiza con un certificado en el que van incluidos los datos del autor, un código del país del desarrollador y la clave privada para firmar la aplicación.

2.6. API Google Maps

Sin duda, un complemento ideal para los desarrolladores a la hora de realizar una aplicación Android es la utilización de mapas haciendo uso de la ya conocida API de Google Maps. [6]

Google Maps es el dominador absoluto a la hora de buscar localizaciones geográficas y tiene desde hace bastante tiempo una API que se puede integrar en las aplicaciones de manera bastante sencilla.

Entre todas las cosas que permite hacer esta API, lo más destacado es crear aplicaciones basadas en la localización, crear mapas para aplicaciones móviles y personalizarlos de forma que destaquen imágenes, datos o marcas y visualizar datos espaciales como imágenes en 3D.

Antes de empezar a usar esta API cualquier desarrollador debe hacer una serie de tareas previas como la descarga e instalación de dicha API.

En primer lugar, se debe tener instalado el paquete con la versión de Android para la que desarrollamos y a su vez complementado con la API de Google. Estos paquetes suelen llamarse "Google APIs by Google, Android API x revisión y" y para descargarlo se hace uso de Eclipse, en concreto, en la pestaña Android SDK Manager.

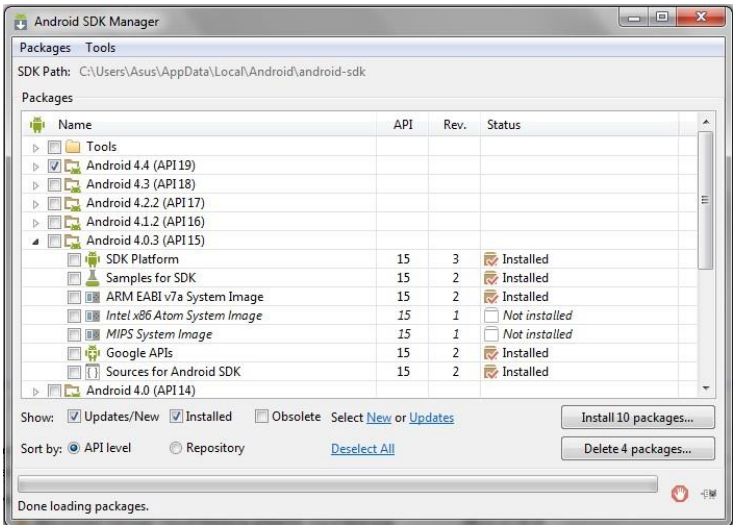

Figura 18. Android SDK Manager

El siguiente paso, no es más que incluir en el proyecto de la aplicación dicha API. Para ello, se debe poner como target en las propiedades a las cuales se accede haciendo click derecho en la carpeta principal del proyecto. Una vez dentro en la pestaña Android se selecciona el target que se desea además de los complementos que se le quiera insertar a dicha API, en este caso, la API de Google Maps.

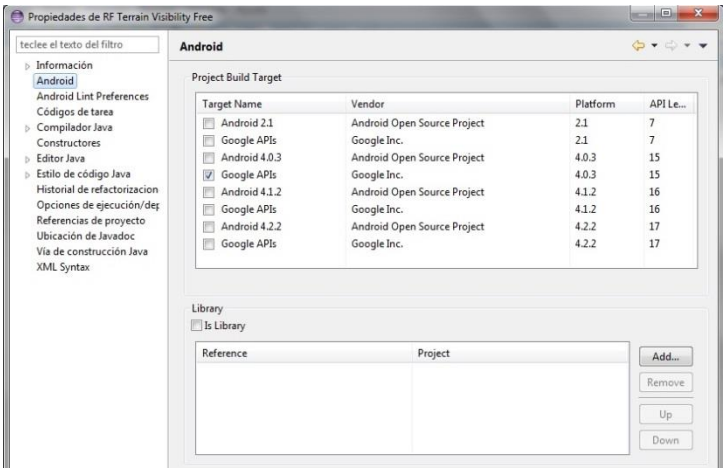

Figura 19. Propiedades del Proyecto

Con todo esto ya estaría creado el proyecto en Eclipse con la API instalada, pero para poder pasar utilizar la API de Google Maps se requiere una obtención previa de una clave de uso (API key) que debe de estar asociada al certificado con el que se firma la aplicación. Esto implica que si se cambia el certificado con el que se firma la aplicación y esta hace uso de la API de Google Maps, se debe cambiar también la clave de uso de la API.

Para obtener la clave de uso, se debe localizar el fichero donde se almacenan los datos del certificado de duración llamado “debug.keystore”. Después, haciendo uso de la herramienta keytool.exe, se accede a él y se obtiene el hash MD5 del certificado. Esto se hace mediante la línea de comandos (cmd) e insertando el comando < keytool -list -alias androiddebugkey -keystore “ruta del fichero debud.keystore” -storepass Android -keypass android> de la que se obtiene como resultado la huella digital del certificado (hash MD5).

Por último, con el hash MD5 se debe acceder a la web http://code.google.com/android/maps-apisignup.html para obtener la clave de uso (API key) de la API de Google Maps para poder publicar la aplicación en el Google Play.

Se acaba así todo el proceso de instalación y verificación de la API y ya se puede pasar directamente a la programación y uso de ella.

Para incluir el mapa en una ventana de la aplicación se debe usar el control MapView . Este control se puede incluir en el layout de igual forma que todos los controles vistos anteriormente (botones, etc.), sólo se debe tener en cuenta que dentro de este control tiene que haber una propiedad que indique la clave de uso de la API de Google Maps como se ilustra en la imagen.

```
<com.google.android.maps.MapView
    android:id="@+id/mapview"
    android:apiKey="
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"/> <!-- Nos permite interactuar con el mapa -->
```

Figura 20. Control MapView

Se debe decir que a este tipo de controles deben añadirse a una actividad de tipo MapActivity por lo que la actividad (o pantalla de la aplicación) en la que se vaya a incluir este mapa debe estar heredando dicha clase. Además, se debe incluir en el fichero AndroidMainfest.xml una etiqueta de la forma <uses-library>.

Se enumeran ahora algunos métodos básicos a modo de ejemplo con los que se manipula el mapa de una aplicación.

Lo más lógico es empezar por algunos métodos sobre la apariencia. Se puede cambiar la apariencia de un mapa a las tres vistas tan conocidas como satélite, tráfico y streetView, llamando al método que los activa los cuales son setSatelite(), setTraffic() y setStreetView() respectivamente. Existen también modos como Hybrid que es tipo satélite con vistas normales o relieve. En la siguiente imagen se puede apreciar los dos tipos más conocidos; satélite y mapa.

Figura 21. Tipos de mapa

En algunas de las propiedades del mapa, entran conceptos como centrar un mapa o ampliar/reducir el zoom. Para obtener el centro del mapa se usa el método getMapCenter() y para ver el nivel de zoom getZoomLevel(). También se puede mostrar controles sobre el mapa, por ejemplo, los de zoom con el método setBuiltInZoomControls().

Existen otros métodos para obtener información acerca del mapa pero la cuestión es, además de ajustar el mapa o cambiar sus características, ¿Cómo se puede interactuar con el mapa y cambiar valores?; la respuesta es una clase que nos permite controlar todos los parámetros del mapa y cambiarlos a la necesidad del desarrollador, la clase MapController que se genera tras llamar al método getController(). Este método nos devolverá el objeto con el que podremos interactuar con el mapa y modificar los parámetros a necesidad de la aplicación.

Se debe hablar de una serie de clases muy interesantes que están ligadas al uso del mapa, estas clases son GeoPoint y la clase Projection.

La clase GeoPoint representa un punto en el mapa con su respectiva latitud y longitud. Ha servido de mucho en la realización del proyecto para poder colocar emplazamientos y obtener sus coordenadas gracias a dos clases getLatitudeE6() y getLongitudeE6().

La clase Projection permite pasar desde coordenadas x/y de los píxeles de la pantalla del dispositivo móvil a coordenadas latitud/longitud que representan un punto sobre la superficie terrestre. Se crea con getProjection() y sus principales métodos son los que convierten de píxeles a coordenadas y viceversa, que son fromPixel() y toPixel() respectivamente.

Por último, se trata la forma de añadir información propia al mapa de Google y como responder a eventos de pulsación sobre el control MapView.

La información a un mapa se introduce añadiendo al mapa nuevas capas (Overlays) donde se incorpora la información personalizada que se quiera incluir sobre el mapa. Se puede incluir cualquier tipo de información sobre el mapa ya sean rutas, notas, imágenes, etc. Por supuesto, se puede añadir todas las capas que se desee sobre el mapa de Google. Para añadir capas existe el método add() y llamando después a postInvalidate() se redibuja el mapa y todas sus capas.

El primer paso para definir una capa es crear una clase que extienda a Overlay y sobreescribir su método Draw(), es este donde debemos dibujar toda la información a representar sobre el mapa. El método Draw() debe recibir como parámetro un objeto Canvas sobre el que apoyarse para dibujar toda la información necesaria, utilizando los métodos de dibujo que trae este objeto como son drawLine(), drawText(), drawCircle(), drawBitmap(), etc. Pero surge una pequeña cuestión, y es que a estos métodos debemos pasarle coordenadas y tamaños en pixeles, por tanto, entra en escena la clase Projection explicada anteriormente mediante la cual se puede pasar de pixeles a coordenadas y viceversa.

Figura 22. Overlay

3. RF Terrain Visibility

3.1. Zona de Fresnel, curvatura de la Tierra y despejamiento

La radiación electromagnética se traduce en una combinación de campo eléctrico y magnético que se propaga por el espacio en forma de ondas que portan energía. Las ondas al propagarse por el espacio libre, en este caso aire, sufren un proceso de expansión lo que permite equiparar la onda con un elipsoide entre los emplazamientos transmisor y receptor.

Las zonas de Fresnel son el volumen de espacio entre el emisor y el receptor de una onda electromagnética de modo que el desfase de las ondas en este volumen no supere un cierto valor en grados. En el caso de la primera zona de Fresnel son 180 grados y en el caso de la segunda 360.

En un factor a tener en cuenta en comunicaciones inalámbricas o por radio además de que haya visibilidad directa entre los dos emplazamientos ya que la onda, en su recorrido desde el transmisor al receptor, puede atravesar obstáculos que infieren en cambios de fase y reflexiones. Todo esto implica una reducción del nivel de señal que llega al receptor, es decir, pérdidas.

La consideración en cuanto al umbral es que la obstrucción no se recomienda que sea superior al 20% en la primera zonal de Fresnel y a partir del 40% se considera que no se puede hacer el radioenlace porque hay demasiada obstrucción.

La zona más interesante es la primera, ya que encierra la mayor parte de energía de la señal. Por eso la aplicación se centra en el estudio de esta primera región.

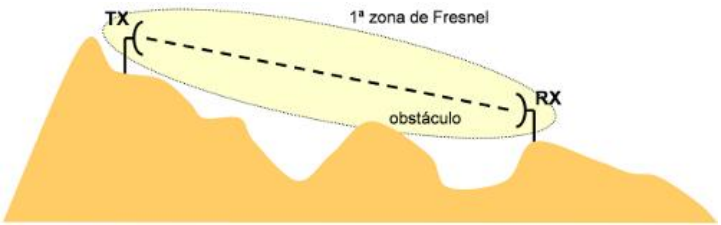

Figura 23. Primera zona de Fresnel

Para calcular la primera zona o región de Fresnel lo primero es trazar la línea de visibilidad directa entre las dos estaciones. A partir de ahí se puede calcular con la fórmula de la izquierda para la primera región y la derecha para la región enésima. [7]

$$r_1 = 548\sqrt{\frac{d1 \cdot d2}{f \cdot d}} \qquad r_n = 548\sqrt{\frac{n \cdot d1 \cdot d2}{f \cdot d}}$$

Siendo,

- r_1 y r_n el radio de la primera y enésima zona de Fresnel.
- d1 es la distancia desde el emplazamiento origen hasta el punto de cálculo (en km).
- d2 es la distancia desde el punto de cálculo hasta el emplazamiento destino (en km).
- f es la frecuencia del radioenlace, debe venir en MHz.
- d es la distancia total entre los dos emplazamientos.

Otro factor muy importante a tener en cuenta es la curvatura de la Tierra. Este factor da lugar a muchos errores porque en radioenlaces cortos es insignificante y no se tiene por qué tener en cuenta, pero a medida que se aumenta la distancia entre las estaciones va tomando importancia hasta el punto de tener que tenerlo muy en cuenta para el cálculo de la visibilidad ya que reduce la distancia entre la zona de Fresnel y los obstáculos.

Es evidente que la curvatura de la Tierra provoca una mayor influencia de los obstáculos sobre la zona de Fresnel, hasta tal punto que llega a definirse un horizonte radioeléctrico a partir del cual se crea una zona de sombra donde no hay visión directa entre los emplazamientos. Por tanto, contribuye a aumentar la altura de los obstáculos.

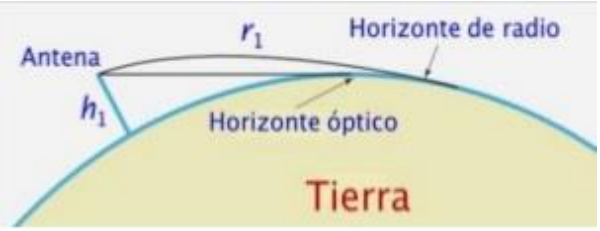

Figura 24. Horizonte radioeléctrico

La fórmula que permite calcular la curvatura de la Tierra en un punto determinado es la siguiente [8]:

$$c_n = \frac{4}{51} \cdot \frac{d1 \cdot d2}{k}$$

Siendo,

- d1 es la distancia desde el emplazamiento origen hasta el punto de cálculo (en km).
- d2 es la distancia desde el punto de cálculo hasta el emplazamiento destino (en km).
- k es el factor de radio efectivo terrestre.

Como se ha visto en la fórmula, existe un factor que describe el grado de curvatura de la Tierra que se conoce como factor K o factor de radio efectivo terrestre. Este factor depende del gradiente de refractividad por kilómetro con respecto de la altura (dN/dh). Se calcula mediante la fórmula:

$$k = \frac{1}{1 + a \cdot (dN/dh) \cdot 10^{-6}}$$

Al depender de la atmósfera tiene diferentes valores según el lugar en el que se esté calculando el radioenlace. Así pues, en condiciones normales o atmosfera estándar el valor de este factor K es de 4/3. Si el factor supera el valor estándar se dice que es “superrefractiva”, en cambio, si el valor de este factor está por debajo de 1 entonces se dice que es “subrefractiva”.

En la siguiente imagen podemos ver cómo influye la curvatura de la Tierra en función del valor de este factor.

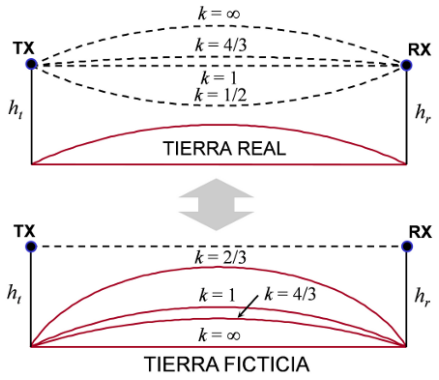

Figura 25. Influencia del factor k

El concepto de despejamiento es el resto de zona de Fresnel que no queda obstruida por un obstáculo. En líneas generales, sería el concepto opuesto a obstrucción y, por tanto, sabiendo que a partir del 40% de obstrucción el enlace no es viable, es análogo decir que con menos del 60% de despejamiento en un radioenlace no es posible establecer comunicación.

En el cálculo de dicho parámetro entran en escena la línea de visión directa entre los emplazamientos, su primera zona de Fresnel y la curvatura de la Tierra. Son con ellos con los que la aplicación se encarga de sacar el porcentaje de despejamiento para cada una de las celdas y es ese valor el que se encarga de decir qué celdas entran en la matriz de visibilidad y también en el rango de colores que se ha introducido, esto se verá posteriormente.

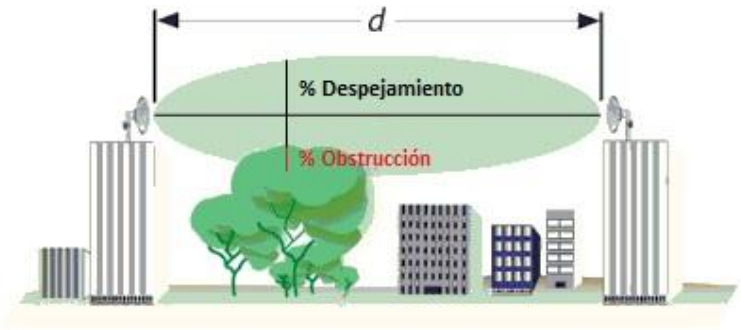

Figura 26. Despejamiento y obstrucción

3.2. Modelo Digital del Terreno (MDT)

Un modelo digital del terreno (MDT) no es más que una representación de la topografía de una zona de la superficie terrestre en una forma adaptada mediante un ordenador. En este caso el modelo digital del terreno será una nube de puntos que llevarán la altura de la topografía y que servirán para conocer la altura y los obstáculos que pueda haber en dicha topografía.

En cartografía, las altitudes suelen representarse mediante curvas de nivel y cotas. Según la zona cubierta existen diferentes tipos de modelo digital del terreno, siendo un modelo rectangular para zonas relativamente pequeñas hasta un modelo pseudocuadrado cuyos lados son meridianos y paralelos para zonas muy grandes.

Existen varios tipos de modelos digitales del terreno como pueden ser cuadrado, rectangular, hexagonal o incluso triangular. En el caso de los modelos cuadrado y rectangular se pueden usar cuadrados, pero en los otros modelos las estructuras son mucho más complejas y no son objetivo de la aplicación que lo usa. En este caso, interesa el modelo cuadrado ya que se rellena una matriz con el valor de las alturas que llega a resoluciones de 100x100m para mapas de visibilidad de hasta 10 km de radio, 200x200m para mapas de hasta 25 km, 300x300m para mapas de hasta 40 km y hasta 50 km de radio la resolución es de 400x400m.

El modelo digital del terreno tiene tres características principales que permiten hacerse una idea rápida de él y conocer su adecuación para según cuales sean las necesidades, su resolución, es decir, la distancia que hay entre dos puntos consecutivos del modelo; la calidad de los puntos, debido a que hay zonas que los puntos no son muy exactos y ampliando este último detalle, se debe tener en cuenta su cobertura geográfica, ya que hay zonas donde ni siquiera existen datos sobre las alturas del terreno.

El modelo digital del terreno puede obtenerse de organismos cartográficos que ponen a disposición pública las bases de datos a las que se puede acceder. Los principales organismos que ofrecen este servicio son la NASA, la NIMA y el USGS. [9]

La aplicación RF Terrain Visibility usa SRTM3 (The Shuttle Radar Topography Mission). La misión topográfica es un proyecto internacional de la NGA (Agencia Nacional de Inteligencia-Geoespacial) y la NASA. Consistió en un sistema radar que voló a bordo del trasbordador espacial “Endeavour” para adquirir los datos de elevación topográfica.

Los modelos digitales del terreno proporcionados por SRTM pueden accederse desde Internet y además vienen en ficheros .hgt que son soportados por infinidad de software. El enlace para acceder a este tipo de datos es http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/ [10]

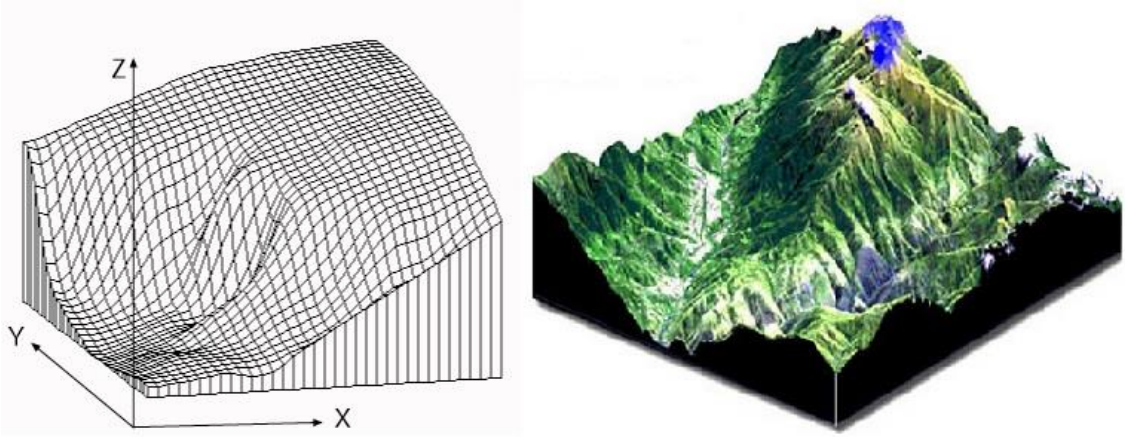

Figura 27. Modelo Digital del Terreno

Una vez conocido el modelo digital del terreno que usa esta aplicación, se debe introducir el concepto de qué es un perfil radioeléctrico. Un perfil radioeléctrico es una representación donde se encuentra un perfil terrestre sacado en este caso de SRTM3, junto con la representación de la zona de Fresnel. Con este perfil se puede calcular los puntos que producen obstrucción y también cuál de ellos es el que más obstrucción produce.

Dicho de otra forma es un corte transversal por la línea que une los dos emplazamientos y que permite ver la situación de un radioenlace. En la imagen se observa la definición grafica de un perfil radioeléctrico.

Figura 28. Perfil radioeléctrico

3.3. Descripción de la aplicación

Se describe en este apartado la aplicación a la que se le han añadido las funcionalidades. Antes de empezar a comentar el objetivo y la forma que tiene la aplicación, se deben tener en cuenta una serie de conceptos para el mayor entendimiento.

Primero debe hablarse de qué es un mapa de visibilidad. Un mapa de visibilidad no es más que una matriz de celdas en la que cada celda, que representa una zona en la que hay situado un emplazamiento receptor, tiene asociado un valor de porcentaje de visibilidad radioeléctrica; siendo este porcentaje el despejamiento de la primera zona de Fresnel en un perfil trazado entre la estación base y cada una de las celdas de dicha matriz.

El otro concepto es un emplazamiento. Un emplazamiento es un punto que puede contener una o varias antenas y que se coloca en un punto del mapa con una latitud y una longitud. A cada emplazamiento debemos indicarle su altura ya que influye en el cálculo del despejamiento. Sobre los emplazamientos se realizan los mapas de visibilidad y es aquí donde se ve todo el potencial de la aplicación.

Por último, los archivos KML no son más que ficheros con datos geográficos que permiten situar en un mapa puntos que estén relacionados.

La aplicación se denomina RF Terrain Visibility y sus funcionalidades son:

- Bases de datos de los emplazamientos. Se encarga de la creación y gestión de los emplazamientos que se almacenan en ficheros XML que contienen su nombre, su posición y altura.

- Cálculo de mapas de visibilidad. Representa las celdas que tienen el porcentaje de despejamiento superior a un umbral especificado por el usuario. Permite ver los puntos que tienen visión radioeléctrica sobre un radioenlace.

- Representación de varios mapas de visibilidad. Muy útil para ver en caso de que entre dos emplazamientos no haya visión directa el lugar donde se podría situar un punto intermedio para realizar la comunicación que tenga visibilidad con ambos.

- Orientación de múltiples antenas. Es capaz de orientar múltiples antenas situadas en un emplazamiento con respecto a los demás emplazamientos que se le indiquen. Para ello hace uso de una brújula e indica la orientación en azimut y elevación.

- Importar/Exportar archivos KML. Es capaz de importar emplazamientos y exportar los creados por la aplicación en formato KML para luego poder ser usados por Google Earth. También es capaz de exportar los mapas de visibilidad en dicho formato.

Figura 29. Iconos de la aplicación

Al iniciar la aplicación lo primero que se muestra es el menú principal que está formado por tres pestañas, la primera para los emplazamientos, la segunda para los mapas de visibilidad y la tercera para la orientación de los emplazamientos. Este tipo de menú se crea usando la clase TabActivity. La actividad que desarrolle este menú debe heredar dicha clase y crear por separado cada una de las pestañas añadiéndolas en el método onCreate(). Esta clase es la principal con la que se inicia la aplicación y así debe indicarse en el AndroidMainfest.xml.

La pestaña Emplazamientos es la que permite crear, editar y exportar como KML los emplazamientos. También permite sincronizar y mandar los archivos a Dropbox a la vez que se puede compartir la aplicación por medio de otras aplicaciones. La apariencia es la que aparece en la siguiente imagen.

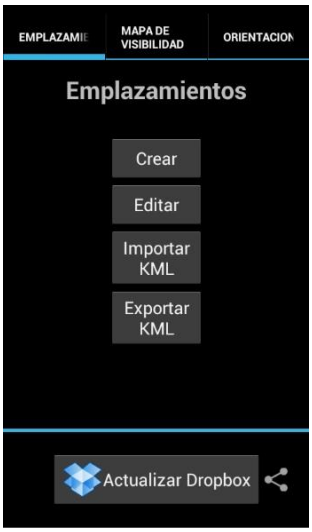

Figura 30. Emplazamientos

Al pulsar el botón Crear se pasa a introducir los datos del emplazamiento. Se debe introducir el nombre, la posición y la altura de este.

Figura 31. Datos emplazamiento

Para introducir la ubicación es posible hacerlo de tres formas. La primera de ellas es introducir directamente la latitud y longitud en los cuadros, la segunda es pulsando el botón Mi ubicación que saca las coordenadas en la que se encuentra el dispositivo por GPS y la ultima es girando el móvil de forma que se puede colocar el emplazamiento pulsando sobre el mapa de Google como aparece en la siguiente imagen.

Figura 32. Emplazamiento sobre mapa

Volviendo a la pantalla emplazamientos, si se pulsa el botón editar aparece una lista con los emplazamientos existentes y pinchando en uno de ellos aparece la misma ventana que al crear un emplazamiento y se podrá modificar completamente. Si se mantiene pulsado un emplazamiento saldrá un cuadro de dialogo que nos pregunta si se quiere borrar el emplazamiento.

Figura 33. Lista emplazamientos

La pestaña Mapa de visibilidad es la que permite crear mapas nuevos y abrir mapas antiguos. Y por supuesto, como se ha dicho anteriormente, exportar dicho mapa como archivo KML.

Figura 34. Mapa de visibilidad

De nuevo, al pulsar sobre el botón Nuevo se pasará primero a recoger los datos del emplazamiento que se tomará como referencia en el mapa. Y en esa misma actividad se recogerá también el radio del mapa y la altura de la antena receptora.

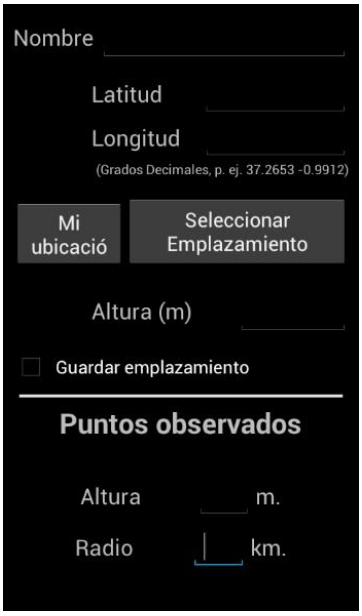

Figura 35. Emplazamiento referencia

De nuevo se puede introducir el emplazamiento de tres formas. Creando uno nuevo introduciendo los datos y marcando el checkbox, seleccionando la ubicación o seleccionando un emplazamiento de los existentes anteriormente.

En la siguiente ventana se introducen los datos relativos al mapa. En ella se pone el nombre del mapa, la frecuencia que viene con un spinner para introducir las unidades, el factor k que por defecto se pone a 1.33 (atmósfera en condiciones normales) y el umbral de visibilidad (el valor a partir del cual se tiene que pintar un cuadrado del mapa de visibilidad), también por defecto a 75%.

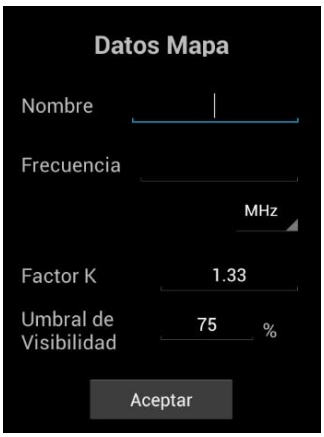

Figura 36. Datos mapa

Una vez que se pulsa el botón Aceptar automáticamente empieza a generarse el mapa y una vez generado se muestra en la pantalla sobre el mapa de Google. Se aprecia en la imagen un ejemplo de mapa de visibilidad ubicado uno en Cartagena y otro sobre Murcia.

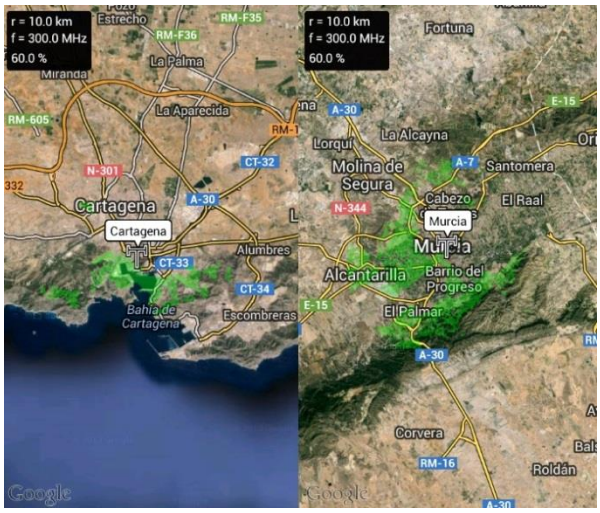

Figura 37. Ejemplo mapas de visibilidad

Y volviendo a la ventana mapa de visibilidad, si se pulsa el botón Abrir se pasa a una lista con todos los mapas que se hayan creado anteriormente. En esta actividad, se pueden seleccionar varios mapas de visibilidad y ser mostrados a la vez sobre el mapa de Google.

Figura 38. Lista mapas

Al seleccionar dos o más emplazamientos, se le asigna a cada uno un color para poder diferenciarlos. Es aquí donde se observa todo el potencial de la aplicación ya que permite visualizar puntos intermedios donde se podrían colocar emplazamientos en caso de que dos puntos no tengan una visibilidad aceptable.

En la imagen se puede ver la representación de los dos mapas. El mapa realizado en Murcia se pinta en rojo para diferenciarlo del verde y se representan los dos el mismo mapa.

Figura 39. Varios mapas

La última pestaña de la aplicación es la orientación. Esta pestaña nos da opción a un botón llamado Múltiple. En esta pestaña se acude cuando se quiere conocer la orientación a la que se encuentran los otros emplazamientos.

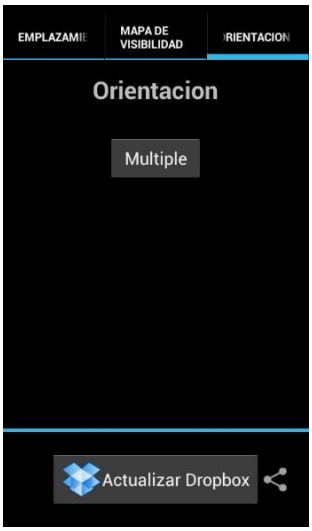

Figura 40. Orientación

Una vez se pincha en este botón. La primera actividad mostraría una lista con todos los emplazamientos existentes y solamente permitiría elegir uno de ellos. En cuanto se selecciona uno de ellos entonces da paso a otra actividad que nos permite elegir uno o varios emplazamientos destino, es decir, hacia los que vamos a orientar el emplazamiento seleccionado en la actividad anterior.

Lo siguiente que se muestra cuando se ha realizado todo lo anterior es la imagen de una brújula junto con una flecha que señala hacia qué dirección está el emplazamiento/s destino.

Figura 41. Brújula

Debajo se tiene un menú deslizante escondido que se muestra/esconde cada vez que se toca en la flecha. Es un objeto SlidingDrawer y al tocar en él se despliega mostrando los valores de orientación de los emplazamientos, el color que representa cada emplazamiento y el nombre del emplazamiento al que se refiere.

Figura 42. SlidingDrawer

4. Mejoras introducidas

4.1. El caso de Google

La API de elevación de Google proporciona una sencilla interfaz con la que permite a los usuarios realizar consultas sobre la elevación en cualquier punto de la Tierra. Permite tanto la consulta individual de un punto de la Tierra como de una ruta completa (esta parte es muy interesante porque ayuda a conseguir todos los puntos intermedios entre dos emplazamientos para poder levantar un perfil terrestre).

La API posee datos de elevación para todas las ubicaciones de la superficie terrestre, tanto los de la superficie terrestre (valores positivos) como los valores de las profundidades del océano (valores negativos). Puede ocurrir que no exista una medida de elevación exacta sobre un punto de los solicitados, en ese caso, se interpolan los 4 puntos más cercanos y se devuelve un valor medio.

Para acceder a la API de elevación se hace uso de una interfaz HTTP, pasándole una ruta o URL que debe tener el siguiente formato:

```
http://maps.googleapis.com/maps/api/elevation/outputFormat?parameters
```

El formato de salida se indica en la URL sustituyendo outFormat por una de las dos posibilidades que ofrece:

- /json. Devuelve los resultados en notación de objetos de JavaScrpit.

- /xml. Devuelve los resultados en formato XML, dentro de un nodo <elevationRespone>. Es el tipo que se ha decidido usar ya que es el más claro y el que permite usar el XmlPullParser (por comodidad) para sacar todos los valores de las elevaciones.

Se puede especificar la forma en la que la API devuelve los datos; como una serie de localizaciones (location) o como una serie de puntos conectados a lo largo de una ruta (path).

Con el campo Sensor se especifica si la aplicación está solicitando los datos de la posición mediante un sensor como el GPS. Este campo es obligatorio. [11]

Por tanto, la URL que se ha definido en este proyecto para obtener las alturas de cada perfil es:

```
http://maps.googleapis.com/maps/api/elevation/xml?locations=
39.7391536,-104.9847034|36.455556,-116.866667&sensor=true_or_false
```

En cuanto a la respuesta, cada una incluirá los siguientes elementos:

- Un código Status. Indica el estado de la devolución. Si ha llegado bien indica OK y si ha superado la cuota de peticiones diaria entonces indica OVER_QUERY_LIMIT.

- Un conjunto results. Contiene un elemento con la localización, otro con la altura y otro con la resolución.

En la siguiente figura se puede observar la estructura XML de una devolución del fichero de alturas que se obtiene de la API de elevaciones de Google.

```
<?xml version="1.0" encoding="UTF-8"?>
<ElevationResponse>
 <status>OK</status>
 <result>
  <location>
   <lat>39.7391536</lat>
   <lng>-104.9847034</lng>
  </location>
  <elevation>1608.6379395</elevation>
  <resolution>4.7719760</resolution>
 </result>
 <result>
  <location>
   <lat>36.4555560</lat>
   <lng>-116.8666670</lng>
  </location>
  <elevation>-50.7890358</elevation>
  <resolution>19.0879040</resolution>
 </result>
</ElevationResponse>
```

Figura 43. Respuesta XML

La idea principal de esta mejora es cambiar el servidor SRTM3 que supone muchísimo código de gestión de la matriz de alturas y no es un proyecto seguro por la API de elevaciones de Google que le da mayor estabilidad a RF Terrain Visibility y hay menos posibilidades de que la aplicación pueda quedar inservible.

La forma de crear esta nueva matriz de visibilidad es sabiendo un punto central donde está el emplazamiento de referencia y el radio del mapa de visibilidad que ha introducido el usuario.

Se puede calcular en primer lugar el número de filas y columnas que tendrá la matriz y saber la posición en coordenadas de cada una de esas celdas sumando la latitud y la longitud de la celda a cada posición, es decir, cuando vamos recorriendo una fila se va cambiando la longitud y cuando se cambia a la siguiente bajamos en latitud volviendo a recorrer la fila en longitud.

Es posible hacer un barrido de toda una matriz (estableciendo un tamaño de celda) y levantando un perfil radioeléctrico para cada celda. Para cada perfil radioeléctrico entre una

celda y el emplazamiento referencia se hace una petición a la API de elevaciones de Google y se extraen en un array con todas las alturas. Para cada petición se calcula el porcentaje de despejamiento y se borra el fichero utilizado para evitar el abuso de memoria.

Para obtener su correspondiente despejamiento, se obtiene primero la distancia de cada uno de los puntos obtenidos a la antena de referencia, después se calcula la curvatura de la Tierra y se suman las alturas de los emplazamientos. Una vez conseguido el array de alturas correcto entonces se pasa a calcular la línea de visión directa y la región o zona de Fresnel. Por último, se obtiene el porcentaje de despejamiento y se devuelve el porcentaje menor ya que es el que más obstrucción produce.

El proceso se repite para todas las celdas, se va rellenando así la matriz y cuando se ha terminado se pasa a crear la imagen de visibilidad que será la overlay que se mostrará sobre el mapa de Google. En la imagen de visibilidad se pone un pixel a verde si el despejamiento supera el umbral establecido por el usuario, y si no es así entonces se deja transparente.

El tamaño de celda que se propone es de 100x100m y el número de puntos intermedios entre el emplazamiento referencia y la celda es de 50 puntos.

Haciendo un poco de cálculo, para una matriz que tenga 1 km de radio se necesitan 20.000 puntos. Para una matriz de 2 km de radio, la cifra asciende hasta un total de 80.000. Esta cifra va aumentando hasta límites que no se pueden considerar como es el caso de los mapas objetivo de esta aplicación y es que para un mapa de 25 km de radio, la cifra de elevaciones llega a 3.125.000 puntos, esto es inviable pensando en las limitaciones que pone Google para evitar un abuso o uso indebido de la API de elevación.

La API de elevaciones de Google está sujeta a un límite de 2.500 solicitudes al día o de 25.000 ubicaciones totales en un día. En caso de ser usuario “for Business” permite 100.000 solicitudes al día y 1.000.000 de ubicaciones. Cuando se sobrepasa este límite en un día, el servicio deja de funcionar durante 24 horas, y si se sobrepasa durante repetidas ocasiones se puede llegar a bloquear el acceso de ese usuario a la API de elevaciones de Google.

Por tanto, mientras exista esta restricción no es viable hacer el traslado y habrá que conformarse con SRTM3. No obstante, el trabajo no ha sido en vano ya que el algoritmo realizado es correcto y sólo falla una vez se ha llegado al límite establecido.

Además se ha dado a conocer esta limitación de la tecnología y para próximos estudios se parte de esa base salvo que las limitaciones de Google puedan volver a cambiar y dar la posibilidad de que se permita hacer el traslado a la API de elevaciones de Google.

4.2. Rango de colores

La siguiente funcionalidad introducida propone otra visión del mapa de visibilidad o dicho de otra forma un tipo distinto de mapa en el cual ya no se observa sólo los lugares donde se supera el umbral.

El tipo de mapa se le ha denominado “Rango de colores” y se crea a la vez que se genera la imagen de cobertura. Aprovecha la matriz de visibilidad que se genera y asigna un color dentro de una escala predefinida que le da al usuario una visión más concreta de en qué puntos se obtiene mejor visibilidad con respecto a otros aunque ambos superen el umbral establecido.

La escala que se ha definido es la que se ve en la siguiente figura.

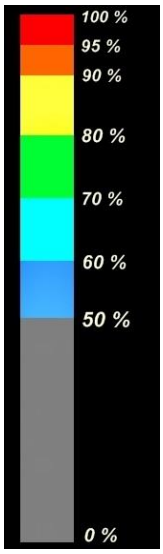

Figura 44. Escala de colores

Como se puede observar, se ha definido la escala de tal forma que los colores que transmiten más sensación de calor son a los que mayor porcentaje de visibilidad se les asigna y a los que transmiten una sensación más fría se les da menor porcentaje. Así se ha decidido empezar con el color rojo para un porcentaje de 95-100%, el naranja para 90-95%, amarillo para un valor comprendido entre 80-90%, verde para 70-80%, cyan para 60-70%, azul para 50-60% y de ahí para abajo 0-50% se puede apreciar que se le ha colocado el color gris. Se tiene en cuenta que a partir de un nivel inferior al 60% de despejamiento el radioenlace no es aconsejable y que el usuario rara vez seleccionará un porcentaje de tal cantidad. A pesar de ello se le asigna el color gris para cubrir algún caso en particular.

Se ha modificado también de la pestaña del menú principal “mapa de visibilidad” dándole un aspecto más adecuado para introducir la nueva situación en la que se encuentra la aplicación. Se ha considerado el tipo de botón que se encargue de abrir los antiguos mapas de visibilidad con el texto “Abrir”, dejando “Rango de colores” para el nuevo tipo de mapa introducido. Como es normal debajo se ha insertado un nuevo botón “Abrir rango de colores” que lleva a la selección de uno o varios mapas de rango de color, de la misma forma que se hace con el mapa de visibilidad.

Figura 45. Nuevo menú mapa visibilidad

Así cuando se genera un mapa de visibilidad, aparece una ventana de diálogo que pregunta por el tipo de mapa a visualizar. Esta opción se creó para que desde que el usuario genera un mapa, ya sea capaz de elegir qué mapa quiere ver para así poder visualizar más rápidamente su mapa objetivo.

Se ha dispuesto las mismas opciones a la hora de seleccionar cuál es el tipo de mapa que se quiere abrir. Por tanto, se elige la opción de “Abrir” si se quiere elegir el mapa de visibilidad normal y si, por el contrario, se prefiere visualizar el rango de colores se elige la opción “Abrir rango de colores”.

Figura 46. Dialogo

Esta opción cambia al abrir un mapa antiguo ya que, como se ha dicho antes, se ha dispuesto un nuevo menú para mejorar la apertura de un tipo de mapa.

Cuando se pulsa el botón de “Abrir mapa de calor” entonces se lleva a la lista de mapas existentes igual que se hacía con los mapas de visibilidad. Una vez abierto el mapa se puede observar el nuevo tipo de mapa con sus respectivos colores.

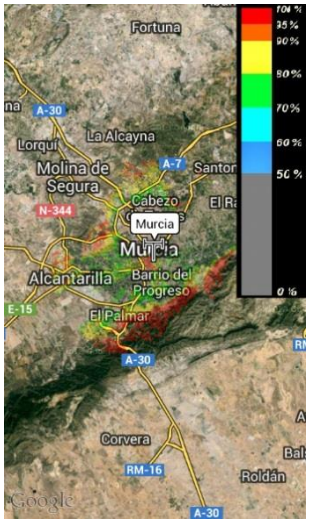

Figura 47. Mapa rango de colores

Como se puede observar se ha introducido una imagen con la escala en un imageView colocado en la esquina superior derecha de la pantalla del dispositivo móvil. Esta imagen sólo se muestra cuando se ha abierto un mapa de calor, por tanto, en las demás vistas permanecerá escondida ya que no es necesaria cuando se abre un mapa del tipo bicolor. Se puede observar esto en la comparativa entre el mismo mapa con los dos tipos existentes.

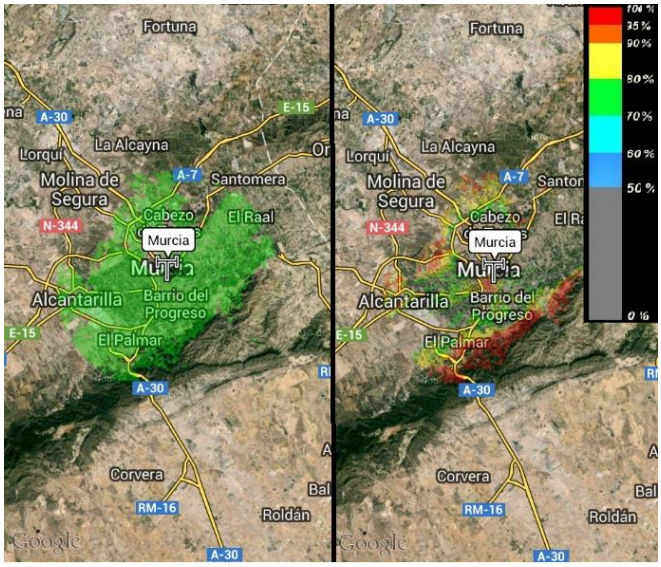

Figura 48. Comparativa

Como es normal, se pueden incluso abrir varios mapas. Esta opción permite ver mucho más concretamente dónde colocar un emplazamiento auxiliar que pueda hacer de repetidor en caso de que no exista una visibilidad aceptable entre dos emplazamientos.

De nuevo aparece la escala que acompaña a cualquier actividad que represente el mapa de calor ya sea al crear uno nuevo o al abrir uno o varios mapas.

En la siguiente imagen se puede observar la representación de varios mapas de calor sobre un mismo mapa de Google.

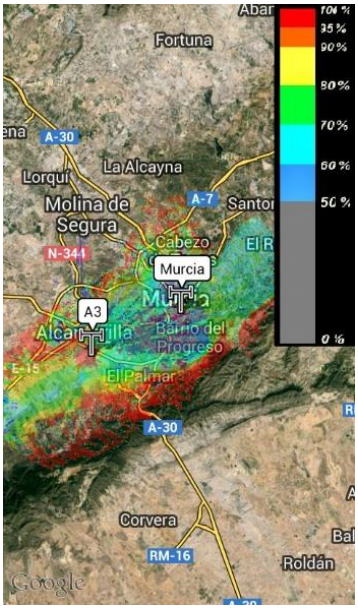

Figura 49. Varios mapas rango de color

En este caso no es necesario cambiar el color de cada uno de ellos ya que entonces perdería sentido este tipo de mapas. De nuevo se hará una comparativa para observar que ya no es tan necesario cambiar el color del mapa. Esto se puede apreciar en la siguiente imagen.

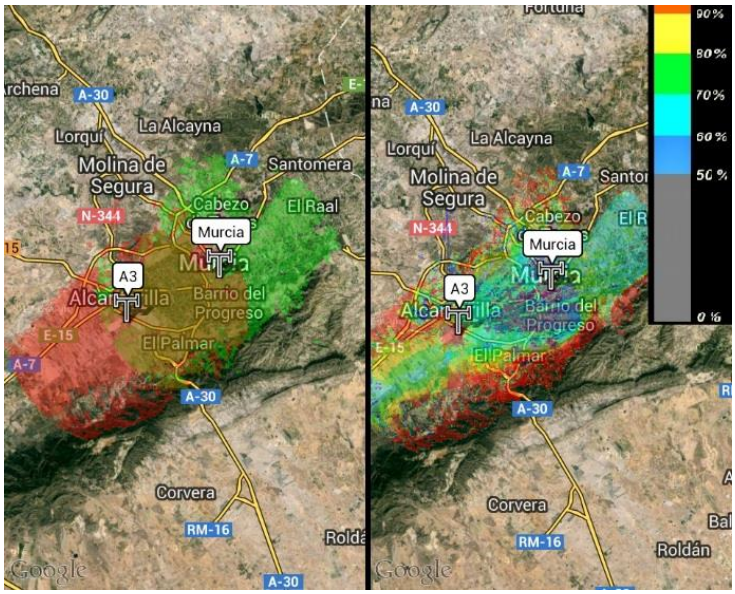

Figura 50. Comparativa varios mapas

4.3. Emplazamiento complementario

Antes se ha comentado que uno de los potenciales de RF Terrain Visibility era poder mostrar varios mapas. Esto hacía que en caso de que no hubiera una visibilidad aceptable entre dos emplazamientos, se podía observar si existe algún punto concreto para situar un emplazamiento intermedio que hiciera de repetidor y permitiera hacer el radioenlace.

Pues bien, ¿por qué no poder dar esa opción al usuario? Precisamente en esto se centra esta funcionalidad. El objetivo es que siempre que se esté mostrando un mapa recién generado, uno o varios mapas antiguos ya sea de cualquier tipo; el usuario pueda colocar el emplazamiento auxiliar pulsando sobre el mapa de Google y además se pueda guardar como un nuevo emplazamiento con las coordenadas donde se colocó.

Se incluye una línea que une el emplazamiento complementario con el emplazamiento que se ha abierto para saber el perfil que los une. Como motivación especial, se intuía conveniente conocer la distancia que hay entre los emplazamientos y el punto intermedio o auxiliar. Se calcula la distancia para después mostrarla pero con una cierta observación, cuando se crea un mapa nuevo era más lógico colocar la distancia debajo del emplazamiento complementario para así entender que esa es la distancia entre ambos. Pero cuando hay varios, se decide poner esa distancia en cada uno de los emplazamientos que se habían abierto para así poder diferenciar qué distancia supone para cada uno. Además se crea un botón que se muestra en la esquina inferior derecha y que aparece cuando el usuario ha colocado el emplazamiento auxiliar con el fin de no molestar a la pantalla donde se están mostrando los emplazamientos.

Se ha predispuesto un mensaje de la clase Toast que muestra al usuario una notificación indicándole de esta nueva funcionalidad. Aparece cada vez que se representa un mapa ya sea nuevo o antiguo. En la imagen se puede ver el mensaje descrito justo cuando se ha representado el mapa.

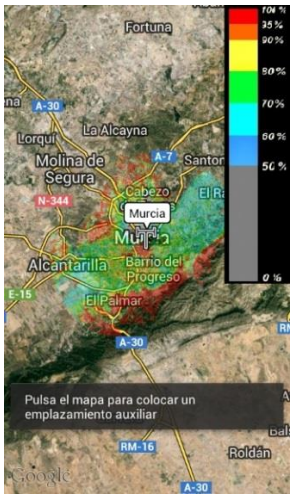

Figura 51. Toast

En la siguiente imagen se puede observar que cuando se ha generado un nuevo mapa al pulsar sobre un punto aparece el emplazamiento y la línea que une ambos. También observar que a distancia se representa bajo el emplazamiento complementario.

Figura 52. Emplazamiento complementario

Se debe decir que también funciona para el caso de que al generar el mapa se haya decidido ver el mapa rango de colores. Es análogo a esto simplemente se vería la escala colocada en la esquina superior derecha.

Cuando se representan varios mapas ocurre lo mismo. Se puede hacer cuando se abre tanto un solo mapa o varios de ellos. También funciona para los dos tipos como se ve en la siguiente imagen.

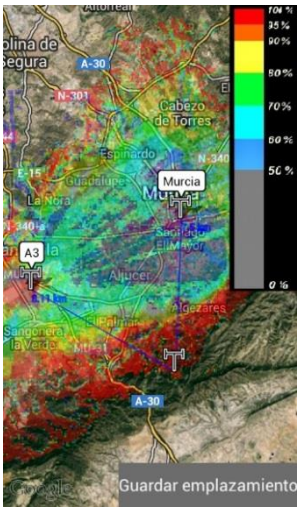

Figura 53. Emplazamiento complementario varios

De nuevo, observar que ahora se representan las líneas a varios emplazamientos y que, en este caso, la distancia aparece sobre los emplazamientos que se abrieron para no confundir cada distancia con su emplazamiento.

Por último, el botón “Guardar emplazamiento” que se activa solamente cuando se ha colocado un emplazamiento y que cuando se pulsa, se encarga de coger las coordenadas donde el usuario puso el emplazamiento complementario y pasarlas mediante un intent a la actividad que crea un emplazamiento. Se le da así al usuario la oportunidad de incluir el nombre y la altura que hace falta para tener un emplazamiento guardado correctamente.

Figura 54. Guardar emplazamiento complementario

Se carga el valor de la latitud y la longitud ya que cuando se pone un emplazamiento no se conoce el valor donde se puso, es decir, el usuario no tiene que saber la latitud y la longitud donde lo ha colocado para guardar el emplazamiento.

Una vez se pulsa el botón “Aceptar” se guarda el emplazamiento y se vuelve a la pantalla donde se visualizaban los mapas.

5. Conclusiones

El sistema operativo Android posee, hoy en día, una amplia gama de dispositivos móviles que son capaces de soportarlo. Es por ello que es el sistema que ha tenido mayor crecimiento en estos últimos años y tanto sus aplicaciones como dicho sistema operativo amplían de forma exponencial el número de usuarios que lo utilizan en cualquier momento.

Es por ello que se ha decidido hacer RF Terrain Visibility para Android a la espera de que sean muchos los usuarios que puedan disfrutar y trabajar con esta aplicación.

El Grupo de investigación de Sistemas de Comunicaciones Móviles (SiCoMo) posee experiencia en el desarrollo de herramientas para la planificación de sistemas de radiocomunicaciones basadas en sistemas de información geográfica. Gracias a ello se ha podido hacer esta nueva e importante versión de RF Terrain Visibility que incorpora dos nuevas funcionalidades que dotan a la aplicación un campo mucho más amplio de posibilidades y a la que se le abre un futuro prometedor.

Como propuesta para posibles líneas de trabajo futuras y de ampliación de esta aplicación se puede hacer que funcione para otros sistemas operativos, como por ejemplo iOS, y que pueda ser comercializada para otros sistemas operativos.

Además como futuras funcionalidades se podría incluir que cuando haya varios mapas mostrándose, se pueda incluir información sobre uno de los emplazamientos al tocarlo (visibilidad respecto a las demás, orientación, etc.). Incluso puede hacerse que se pueda usar la cámara para mostrar la orientación, es decir, que al usuario que se encuentre en un emplazamiento le aparezca un punto con el lugar hacia donde se encuentran los otros, pudiendo así saber hacia dónde tiene que apuntar la antena.

Para finalizar, se han cumplido los objetivos propuestos para este proyecto ya que se ha conseguido poner en funcionamiento la nueva versión de la aplicación. Elegir este proyecto, para mí, ha sido un acierto ya que me abre un amplio abanico de posibilidades laborales ya que hay multitud de empresas dedicadas al desarrollo de estas aplicaciones sabiendo que mi aprendizaje no ha hecho nada más que comenzar. Empezar a trabajar con Android supuso algunas dificultades que pudieron ser superadas satisfactoriamente con la motivación de que el lenguaje de programación “Java” visto en la carrera me ha ayudado bastante a hacer más ameno el aprendizaje de todas las librerías y funcionalidades que trae la programación de aplicaciones para Android.

6. Bibliografía

* RF Terrain Profiles, descripción y descarga.

https://play.google.com/store/apps/details?id=com.version1

* Historia y comienzos de Android.

http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html [1]

http://histinf.blogs.upv.es/2012/12/14/android/ [2]

http://www.openhandsetalliance.com/index.html

*Necesidades de instalación de entorno de desarrollo.

http://www.cristalab.com/tutoriales/herramientas-para-comenzar-a-programar-para-android-c105668l/ [3]

* Estructura de un proyecto

http://www.androidcurso.com/index.php/tutoriales-android/31-unidad-1-vision-general-y-entorno-de-desarrollo/148-elementos-de-un-proyecto-android [4]

* Información sobre componentes de una aplicación, elementos gráficos de Android.

http://developer.android.com/guide/components/index.html [5]

* Documentación de la API de Google Maps.

http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/tutorial.html [6]

* JM Hernando Rábanos, “Transmisión por radio”, Ed. Centro de estudios Ramón Areces, Tercera edición, 1998. [7]

* Curvatura de la Tierra.

http://www.radioenlaces.es/articulos/correccion-de-la-altura-de-los-obstaculos/ [8]

* Modelo Digital del Terreno.

http://es.wikipedia.org/wiki/Modelo_digital_del_terreno [9]

* SRTM3.

http://es.wikipedia.org/wiki/Misi%C3%B3n_topogr%C3%A1fica_Radar_Shuttle

http://www2.jpl.nasa.gov/srtm/index.html [10]

* Documentación de la API de elevaciones de Google.

https://developers.google.com/maps/documentation/elevation/?hl=es [11]