ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

ARRIVALERT: TOOL TO ANNOUNCE ARRIVAL IN ANDROID

Realizado por

JUAN LEÓN PADILLA

Dirigido por

JOSÉ RAMÓN PORTILLO FERNÁNDEZ

Departamento

MATEMÁTICA APLICADA I

Sevilla, Diciembre de 2012

"Sólo hay un camino para llegar, y mil para alejarse."

Jean De La Bruyere

AGRADECIMIENTOS

A mis padres y mi hermana, porque siempre han creído en mí

A Chari, porque hace que cada día sea mejor que el anterior

A Jose Manuel, Belén, Mari, Jose y Raúl, por todos esos buenos momentos

A Maruchita, por ser la testeadora oficial

A Bea, porque sin ella no tendría un logotipo decente

A Juanma, Pablo, Manolo y Carlos, por estos últimos años en la Escuela

A Jose Antonio, Ana Belén y Virginia, por aquellos primeros años en la Escuela

A Juanma, por ser el testeador oficial

A Antonio, por preocuparse de que no me faltara buen material bibliográfico

Y a Josera por haberme guiado a este momento

ÍNDICE

ÍNDICE DE FIGURAS

1.INTRODUCCIÓN

La aplicación a desarrollar, *ArrivAlert*, mantendrá informado a su usuario de cuándo debe llegar a cada uno de los eventos registrados en su calendario y avisará también a los invitados a dicho evento, en caso de no poder llegar a tiempo.

ArrivAlert dispone de su propio calendario de eventos, similar al de cualquier aplicación de calendario compatible con la tecnología *Android*. Para cada evento interesaría conocer su fecha de inicio y su localización para poder realizar las alertas al usuario y los avisos a los invitados.

Por ello, antes del inicio de cada evento se comprobará la localización del usuario, cotejándola con la localización de dicho evento, y mostrando por pantalla al usuario la distancia a la que se encuentra del lugar de reunión asignado al evento, así como una notificación indicando si llegará puntualmente o no a la ubicación de dicho evento.

En el caso que el usuario se encuentre a una distancia mayor, que supere el tiempo establecido para el inicio del evento, sería posible avisar a los invitados por cualquier servicio de mensajería instantánea que tenga instalado el dispositivo (correo electrónico, SMS, etc) mediante un mensaje que el usuario pueda haber escrito previamente.

1.1. DEFINICIÓN DE OBJETIVOS

El objetivo principal de *ArrivAlert* es avisar de la llegada de un usuario en un determinado lugar de reunión, tanto a él como a los demás invitados, a un evento registrado por el calendario de nuestra aplicación en el caso de no llegar a la hora prevista.

Otro objetivo a conseguir es ofrecer un funcionamiento simple e intuitivo, para que por parte del usuario sea lo más sencillo posible delegar la responsabilidad de aviso a los eventos activados.

1.2. ANÁLISIS DE ANTECEDENTES Y APORTACIÓN REALIZADA

El concepto que rodea a *ArrivAlert* es una mezcla de varias funcionalidades que recogen algunas aplicaciones de distintas plataformas. Algunas de estas aplicaciones son las siguientes:

1.2.1. GLYMPSE

Es una aplicación multiplataforma (disponible tanto para *Android* como para iOS) es capaz de compartir la posición del usuario de la aplicación en cada momento por SMS, *WhatsApp*, email, *Facebook* y/o *Twitter*, además de estimar a cuanto está en distancia y tiempo del lugar asignado como lugar de reunión, ofreciendo un mapa que se actualiza periódicamente. Se trata de una aplicación muy completa, vistosa y fácil de utilizar. Su principal inconveniente es que no se usa conjuntamente con ninguna aplicación de calendario, sino que es totalmente independiente.

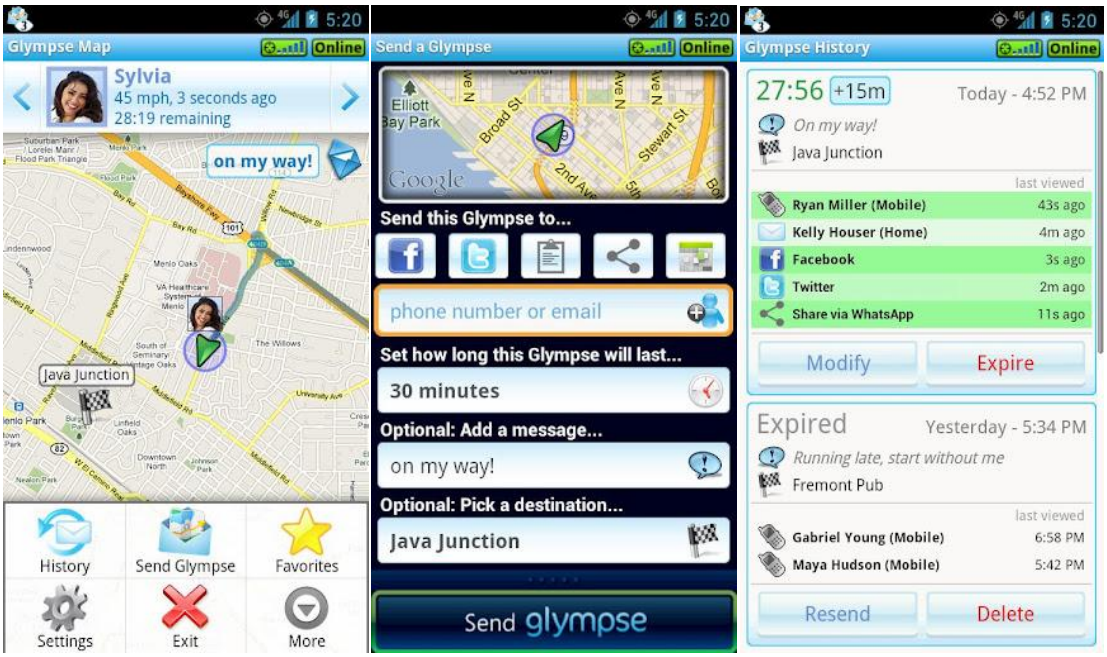

Figura 1 Captura de pantalla de Glympse

Tiene un tamaño de 1,9 MB y se ofrece de forma gratuita tanto en *Google Play* como en *App Store*:

http://itunes.apple.com/es/app/glympse-free-location-share/id330316698?mt=8

https://play.google.com/store/apps/details?id=com.glympse.android.glympse&hl=es

1.2.2. GOCAL

Se trata de una aplicación de *iOS* que se sincroniza con los calendarios instalados, actuando a la vez como un único calendario sobre los demás.

Es compatible con *Google Calendar*, ofreciendo una interfaz similar que permite gestionar los invitados a los eventos, agregándolos o eliminándolos en cualquier momento, además de poder compartirlos con otros usuarios.

Es capaz de generar emails y SMS como recordatorios para esos eventos.

Figura 2 Captura de pantalla de GoCal

Aunque actúa perfectamente como un calendario muy completo, carece de toda la funcionalidad necesaria para avisar de la llegada a un determinado lugar de reunión, ya que no usa una interfaz de mapas semejante a la que usaría *Google Maps*.

Tiene un tamaño de 4,6 MB y un precio de 2,39 €.

Está disponible exclusivamente para cualquier dispositivo de *Apple*. Se puede encontrar más información e incluso adquirir en el siguiente enlace:

http://itunes.apple.com/es/app/gocal-for-google-calendar/id346454140?mt=8

1.2.3. INAP

Se trata de una aplicación para dispositivos con *iOS* y *Android* que te avisa cuando estás próximo a una determinada localización mediante GPS.

Su nombre se debe a que está diseñado para ser utilizado como despertador, así el usuario aprovecha el trayecto del viaje para echarse una siesta hasta llegar al destino señalado.

Es capaz de alertar marcando una dirección concreta y un radio determinado desde la dirección de destino, así el usuario recibe el aviso con antelación, así se evitaría el riesgo de pasarse de parada en caso de viajar en tren o autobús.

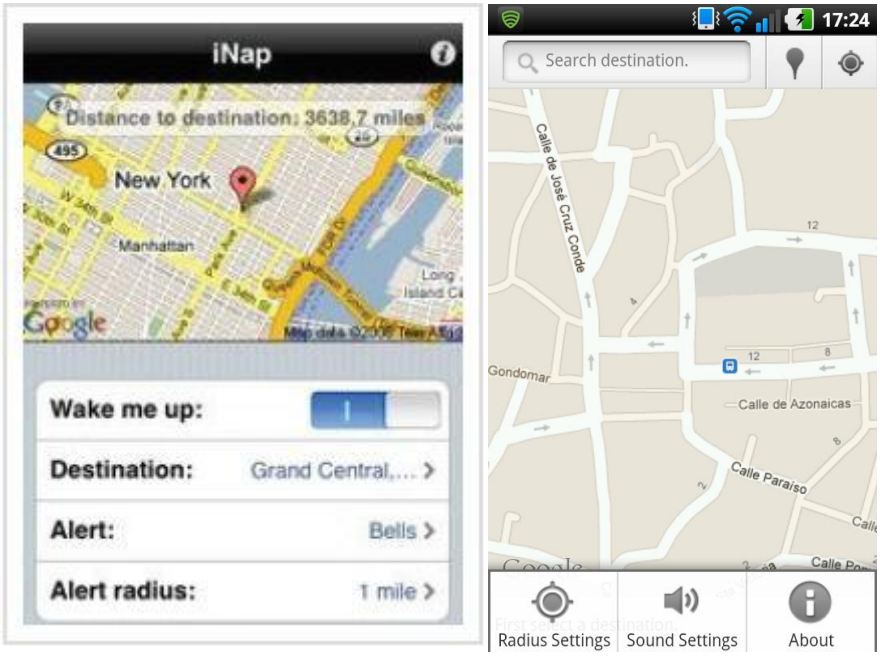

Figura 3 iNap en sus versiones para iOS y Android

Como hemos dicho anteriormente iNap está disponible para iOS a 0,99€ y gratis para Android.

1.2.4. LOCALE SEND POSITION PLUG-IN

Es una aplicación para dispositivos Android que es capaz de enviar por correo electrónico y/o por SMS la localización actual del usuario, además de un enlace de Google Maps del mismo.

Dicha posición es calculada por GPS (advirtiendo que esta opción es más lenta pero más certera) o a través de la red (que es más rápido de conseguir aunque más lento).

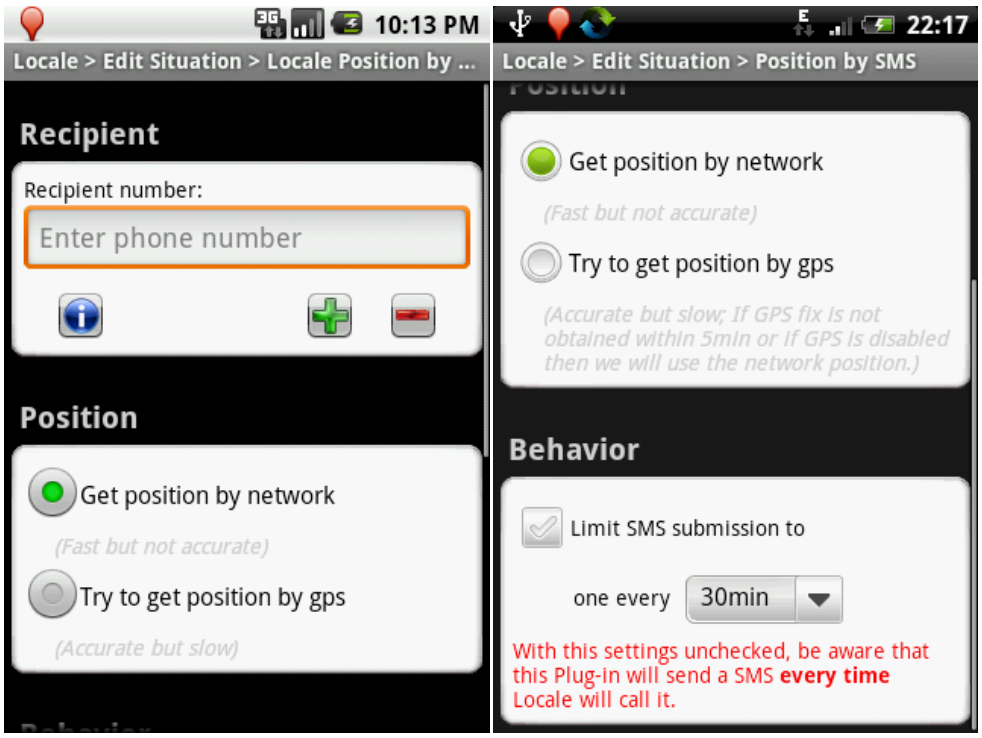

Figura 4 Capturas de Locale send position Plug-in

Está desarrollado por Androgone Development y puede descargarse en el siguiente enlace a un coste de 0,75€:

https://play.google.com/store/apps/details?id=com.androgone.locale.smsposition&hl=es

La aportación a realizar en *Arrivalert* ha sido la unificación de todas las funcionalidades básicas que reúnen las herramientas existentes en el mercado que acabamos de enumerar, de forma que se pueda reunir para el usuario una doble utilidad, por un lado le anuncia de su propia llegada y por otro se encarga de alertar a sus contactos en caso de no llegar a tiempo.

Es decir, crear en una única aplicación la posibilidad de notificar al usuario de la proximidad del inicio de un evento determinado, alertándolo de la distancia a la que se encuentra, así como de una estimación de la cercanía o lejanía a la que está el propio usuario de la ubicación al evento, avisando también a los invitados, en caso de lejanía.

2. ANÁLISIS DE REQUISITOS

La base principal en la que se apoya la aplicación son los eventos de calendario, los cuales resultan imprescindibles para la funcionalidad de *ArrivAlert*.

En un primer momento se pensó que *ArrivAlert* no generara su propio calendario, por lo que se concibió como una aplicación que actuara prácticamente sin ninguna interacción para el usuario.

Aunque más tarde se pensó en que nuestra aplicación tuviera una funcionalidad mucho más extendida, permitiendo al usuario tener una experiencia más personalizada y extendida de lo que normalmente ofrecen las aplicaciones basadas en el uso de eventos de calendario.

Por ello se pensó que los requisitos habituales que se encuentran asociados a un evento de calendario, como pueden ser una ubicación determinada y una fecha señalada, pudieran interactuar con el usuario, permitiéndole a éste en cualquier momento la configuración de los mismos dentro de la propia aplicación.

En los siguientes apartados se abordarán los pilares en los que se sustentan *ArrivAlert*, que marcan el principio de su funcionamiento.

2.1. REQUISITOS DE INFORMACIÓN

ArrivAlert se basa en aportar información al usuario acerca de todo lo que acontece acerca de sus eventos de calendario, por lo que resulta imprescindible asociar esa información con la referida al lugar donde sucede ese evento y al momento de inicio del mismo.

Por ello podemos definir los siguientes requisitos de información:

- **Información sobre eventos:** El sistema debe definir para cada evento un nombre, que identificará al mismo, una descripción, una fecha de inicio, una ubicación asociada, una fecha de alerta asignada y un radio de cobertura para estimar la cercanía a la que se encuentra el usuario de la ubicación asociada.

- **Información sobre ubicaciones:** El sistema debe definir para cada ubicación un nombre, que identificará al mismo, una descripción, una dirección de tipo postal entendible para el usuario y las coordinadas geográficas, latitud y longitud, entendibles para el sistema.

- **Información sobre alertas:** El sistema debe definir para cada alerta un nombre, una referencia para enviar esa alerta (la dirección de correo electrónico del invitado), un asunto y un contenido.

De forma complementaria a estos requisitos, podemos definir como ***restricciones de información*** la existencia de un identificador único para cada evento, ubicación o alerta que se produzca en el sistema y que la fecha de inicio de cada evento siempre sea posterior a la fecha de alerta asociada a dicho evento.

Así mismo, un evento se puede encontrar en varios estados, estado pendiente (si la fecha actual es anterior a la de inicio), estado finalizado (si la fecha actual es posterior a la de inicio) y estado en curso (si la fecha actual es igual a la de inicio).

2.2. REQUISITOS FUNCIONALES

En este apartado enumeraremos los requisitos encargados de especificar el comportamiento del sistema.

- **Conectividad:** El sistema debe ser capaz de detectar una conexión a internet válida.

- **Base de datos:** El sistema debe ser capaz de salvar, almacenar, editar y borrar toda la información referida a eventos, ubicaciones y alertas.

- **Localización:** El sistema debe ser capaz de conocer la última localización del usuario.

- **Aviso de proximidad:** El sistema debe ser capaz de avisar al usuario que la fecha de inicio de un evento está próxima.

- **Cálculo de distancia:** El sistema debe ser capaz de calcular la distancia entre la ubicación del usuario y la del evento.

- **Mensajería:** El sistema debe ser capaz de disponer un sistema encargado en el envío de mensajería instantánea.

- **Notificación:** El sistema debe ser capaz de notificar al usuario a qué distancia se encuentra de la ubicación del evento.

- **Interpretación de la distancia:** El sistema debe ser capaz de interpretar la proximidad geográfica entre la localización del usuario y la del evento.

- **Interpretación del evento:** El sistema debe ser capaz de establecer unas reglas de actuación, de acuerdo con el estado en el que se encuentre el evento.

- **Gestión de alertas:** El sistema debe ser capaz de acceder a información referida a contactos del usuario (como puede ser la dirección de correo electrónico).

2.3. REQUISITOS NO FUNCIONALES

A continuación enumeraremos los requisitos no funcionales de nuestra aplicación:

- **Usabilidad:** El sistema debe ofrecer toda la información posible y sus recursos de forma útil.

- **Compatibilidad:** El sistema debe ser compatible con la tecnología *Android*.

- **Portabilidad:** El sistema deberá ser portable para ser usado con total confianza en cualquier tipo de dispositivo móvil, de tecnología compatible.

- **Seguridad**: El sistema debe ser fiable en cuanto a seguridad y ofrecer robustez contra posibles ataques.

- **Calidad:** El sistema debe ofrecer un buen nivel de calidad, consiguiendo la atención e interés de los usuarios potenciales de la aplicación.

- **Uso de software libre:** El sistema debe desarrollarse y soportarse con tecnologías y herramientas de tipo software libre.

3. DISEÑO E IMPLEMENTACIÓN

Para la elaboración de *ArrivAlert* se ha necesitado una serie de componentes, la mayoría de ellos creados de manera propia y otros ya existentes por medio de la API que proporciona Google.

Ya que *Android* está basado en Java resulta cómodo usar sus bibliotecas disponibles, aprovechando esa compatibilidad se puede ahorrar tiempo a la hora de crear nuevas estructuras más complejas.

ArrivAlert utiliza la API 10 (*Android 2.3.3 – Gingerbread*) en conjunción con la API de Google compatible para ese nivel para disponer de las funciones propias de navegación GPS que ofrece *Google Maps*, ya que resultarán necesarias para definir la ubicación de un dispositivo.

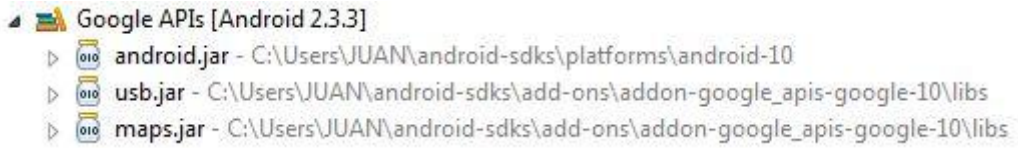

Figura 5 Detalle del paquete API en Eclipse

Actualmente *Android* ofrece la API 17 (Android 4.2 – *Jelly Bean*) como versión más reciente, sucesora de la famosa *Ice Cream Sandwich* que trajo la versión 4.0, con interesantes novedades a la plataforma.

Aún así, se prefirió desarrollar *ArrivAlert* con la API 10 por motivos de compatibilidad con la versión del sistema operativo del dispositivo móvil con el que se desarrollaban las pruebas, la cuál es la versión 2.3.4.

Figura 6 Detalle de la versión del sistema operativo

3.1. ESTRUCTURA DE CLASES

La estructura de clases de *ArrivAlert* está organizada en un único paquete para minimizar, en la medida de lo posible, errores de compilación. Dichos errores eran propensos a aparecer por la compartición de funcionalidad entre los distintos archivos *.java* que esperaban ser referenciados entre las distintas clases.

El nombre que recibe el paquete debe ser particular, para distinguirse de otros y así no generar conflicto. Por convenio se suele utilizar el nombre del dominio en orden inverso acompañado del nombre de la aplicación. De ahí viene que se denomine como *es.rediris.forja.jleon.arrivalert*.

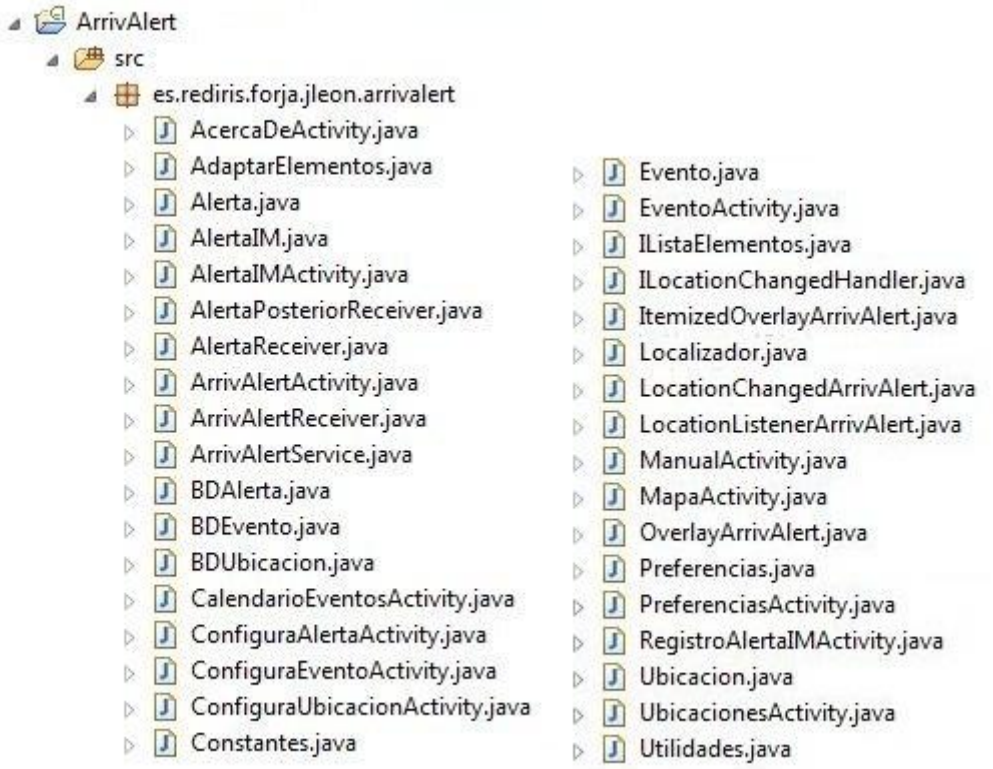

Figura 7 Estructura de clases de ArrivAlert

Como puede verse por los nombres de los archivos *.java* de la figura anterior, existen diversos tipos de archivos en *Android*.

Los denominados con el sufijo *Activity* corresponden a las actividades en Android, que podrían definirse de manera

sencilla como las ventanas que contienen la interfaz de usuario de la aplicación, es decir, son la parte visible de *Android* que permite interactuar al usuario con la aplicación.

Cada *Activity* interactúa con otros elementos propios de *Android*, desde archivos XML complementarios a esas actividades (nos referiremos a ellos posteriormente en esta sección como layouts) hasta otras *actividades*. Cuando una actividad lanza a otra actividad suele enviarle información, esto se puede llevar a cabo mediante el objeto Intent (intención).

Dentro de ArrivAlert, podemos encontrar las siguientes actividades:

- **AcercaDeActivity**: Define información sobre la autoría de la aplicación.
- **AlertaIMActivity**: Define la gestión de las alertas de mensajería instantánea.
- **ArrivAlertActivity**: Es la actividad principal de la aplicación, recoge toda la actividad de la pantalla principal de *ArrivAlert*.
- **CalendarioEventosActivity**: Define la gestión de los eventos de calendario.
- **ConfiguraAlertaActivity**: Define la asociación de las alertas que se encuentran asociadas en la aplicación.
- **ConfiguraEventoActivity**: Define la configuración de los eventos de calendario.
- **ConfiguraUbicacionActivity**: Define la configuración de las ubicaciones de la aplicación.
- **EventoActivity**: Define los eventos de calendario.
- **ManualActivity**: Define información sobre el manual de usuario.

- **MapaActivity**: Define el mapa de la aplicación.

- **PreferenciasActivity**: Define las preferencias de la aplicación.

- **RegistroAlertaIMActivity**: Define la configuración de las alertas de la aplicación.

- **UbicacionesActivity**: Define las ubicaciones de la aplicación.

Además de los casos mencionados anteriormente, en *ArrivAlert* las actividades deben interactuar con elementos que actúan como servicios y otros que lo hacen como receptores.

El denominado con la extensión *Service* sirven para establecer un *servicio en Android*. Un servicio es una aplicación que se ejecutaran en segundo plano sin interactuación por parte del usuario, lo cual es muy útil para *ArrivAlert*, ya que resulta imprescindible a la hora que el sistema notifique al usuario su alerta de llegada.

Nuestro servicio se encuentra en ***ArrivAlertService***, actuando de forma conjunta con las clases que se ocupan de la notificación al usuario, sin contar con una interfaz de usuario.

Las denominadas con el sufijo Receiver, actuarán como receptores de mensajes, es decir serán componentes encargados en recibir y reaccionar frente a ciertos mensajes emitidos por el sistema. Es imprescindible que cada uno de estos receptores deba extender de la clase *BroadcastReceiver*.

Tampoco tienen una interfaz de usuario asignada, pero no la necesitan, ya que se lanzaría la actividad correspondiente como reacción al mensaje recibido. Ni siquiera es necesario que la propia aplicación se esté ejecutando para que ellos actúen, estando registrados el sistema se encargará de lanzar la aplicación si hiciera falta cuando se reciba el mensaje.

En *ArrivAlert* actuarán como receptores de mensajes los archivos ***ArrivAlertReceiver***, ***AlertaReceiver*** y ***AlertaPosteriorReceiver***. Estos ficheros serán los encargados de interactuar con el usuario a la hora de programar sus alertas de aviso e informar de la modificación del estado de sus eventos.

Para finalizar esta sección, mostraremos una figura en la que se recogen los principales recursos de la aplicación. Cabe destacar el uso de la carpeta **layout**, que contiene los ficheros XML con las vistas de la aplicación (layout-land hace referencia a las vistas horizontales). También se encuentra la carpeta **values** que contiene ficheros XML con los valores de los tipo string, arrays y styles (estilos), las carpetas con extensión **drawable** que contienen los iconos de la aplicación, la carpeta **menu** donde se alberga el fichero XML que define al menú contextual de la aplicación y la carpeta **xml** donde se define el fichero XML de preferencias, para la configuración de ArrivAlert.

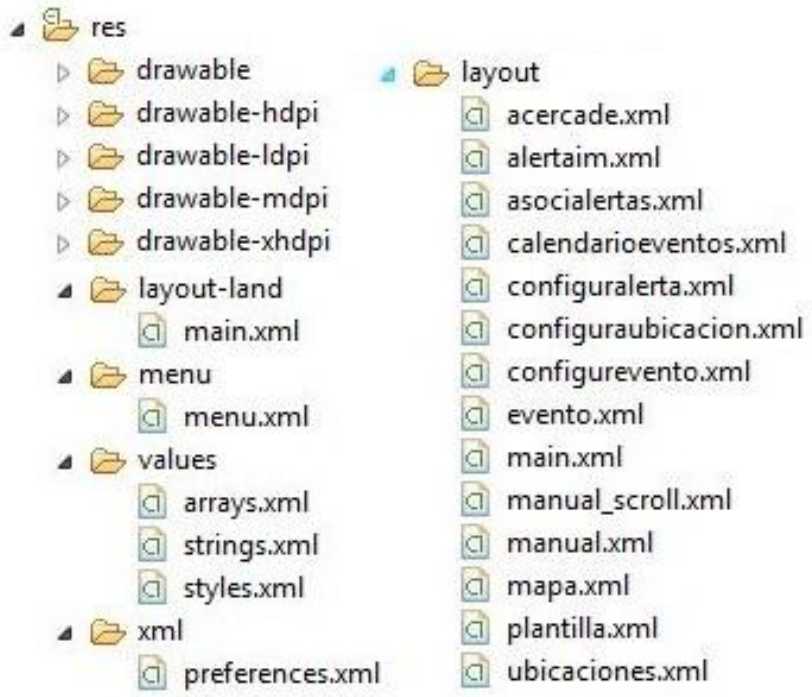

Figura 8 Carpeta de recursos de ArrivAlert

3.2. DISPOSICIÓN DE LA INFORMACIÓN

A la hora de mostrar por pantalla la información requerida para eventos, ubicaciones y alarmas se dispuso de una interfaz que recopilara de manera sencilla cada uno de los elementos que definen estos requisitos.

Esta interfaz es **IListaElementos**, que define para cada uno de los requisitos de información definidos en la sección anterior, un identificador único, una imagen asociada, un elemento principal y uno secundario.

```
public interface IListaElementos {

        int    getId();
        int    getElementoImagenID();
        String getElementoPrincipal();
        String getElementoSecundario();
}
```

Figura 9 Interfaz IListaElementos

En el caso de los **Eventos**, el elemento principal es el propio nombre del evento, destacando sobre los demás para actuar como identificador único. Como elemento secundario figura la fecha del inicio del evento y el estado en el que se encuentra el evento, que como comentamos anteriormente, dependerá de la fecha actual con respecto a la de inicio.

La imagen que acompaña refleja el estado en el que se encuentra el evento.

Figura 10 Detalle de un evento de calendario

Con respecto a las **Alertas**, el elemento principal es el nombre que se le puso a la alerta en cuestión, también destacando sobre los demás para actuar como identificador único. Como elemento secundario se encuentra la dirección de correo electrónico del invitado, actuando como destinatario de la alerta. También se incluye la descripción y el contenido para que el usuario visualice en todo momento el mensaje que se enviará posteriormente. Aquí la imagen es meramente un elemento enumerativo.

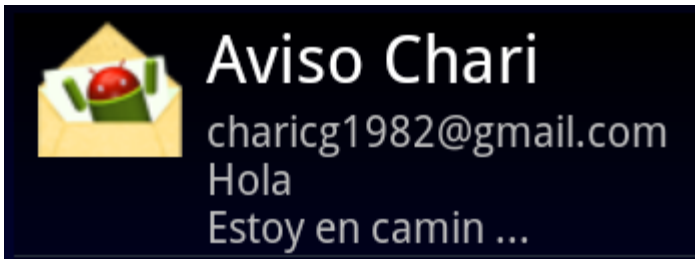

Figura 11 Detalle alarma

Para las **Ubicaciones**, el elemento principal es nuevamente el nombre con el que hemos querido definir a la ubicación y el elemento secundario es la dirección de esa ubicación, generada por *geocódigos inversos* mediante la API de Google. También aquí la imagen es meramente un elemento enumerativo.

Figura 12 Detalle ubicación

3.3. DESARROLLO DE ALERTAS

Otra de las herramientas imprescindibles para el buen funcionamiento de *ArrivAlert* son las alertas. Como hemos comentado anteriormente, son utilizadas para avisar tanto al usuario de nuestra aplicación como a nuestros contactos.

A lo largo de nuestra aplicación hemos usado varios tipos de alertas, en los siguientes apartados explicaremos el uso de cada una de ellas y su utilidad en *ArrivAlert*.

3.3.1. TOAST

La clase Toast permite visualizar un mensaje. Un objeto Toast se define como un mensaje simple, no persistente, diseñado para indicar al usuario que se ha producido un evento o acción concreta por parte del sistema o por interactuación de ambos.

Se muestra de forma flotante sobre la aplicación, lo más discreto posible, ya que aunque sea de carácter informativo, pretende que moleste lo menos posible.

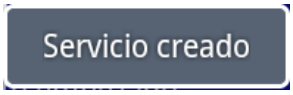

Figura 13 Ejemplo de Toast ejecutado

Los objetos Toast aparecen con mucha frecuencia en ArrivAlert con roles diversos (avisar de una alerta, crear un servicio, mensaje de error, etc) y se ha preferido implementarlos para que permanezcan en pantalla lo máximo posible, por el carácter informativo de este proyecto. Esto se ha llevado a cabo usando la constante **LENGTH_LONG** en lugar de **LENGTH_SHORT**.

```
Toast.makeText(this,"Servicio creado", Toast.LENGTH_LONG).show();
```

Figura 14 Ejemplo de Toast implementado

3.3.2. NOTIFICACIONES

Notification y NotificationManager son las clases encargadas en Android de realizar notificaciones, las cuales engloban un amplio

abanico, desde un mensaje emergente hasta un LED que parpadea, pasando por una vibración.

En ArrivAlert se han utilizado estas clases en conjunción con Service para avisar al usuario de la proximidad en el tiempo de un evento de calendario.

En este caso un Toast no resulta útil ya que no se mantienen mucho tiempo en pantalla y el usuario podría estar distraído y no verlos. La clase NotificationManager muestra un mensaje persistente en la parte superior del dispositivo (en su barra de estado).

Figura 15 Detalle de Notificación ejecutada

A la hora de implementarlo se necesita la clase Intent para conectar con la actividad CalendarioEventosActivity mientras que la clase PendingIntent especificará otro elemento Activity para crear una instancia.

```
Intent i = new Intent(this, CalendarioEventosActivity.class);
i.putExtra(Constantes.CTE_EVENTO_ID, ev.getId());
PendingIntent pi = PendingIntent.getActivity(this, 7777, i, 0);
n.setLatestEventInfo(this, "Evento " + ev.getNombre(),
 ev.getStrEstado() + " de " + ev.getUbicacion().getNombre(), pi);
nm.notify(ev.getId(), n);
```

Figura 16 Detalle de Notificación implementada

También se muestra un mensaje en la parte superior de la pantalla, conocido como tickerText.

Figura 17 TickerText de una notificación

Además de implementar este mensaje de texto, se ha dispuesto para *ArrivAlert* otros campos que complementan la notificación, como el sonido y la vibración, siendo éste último configurado personalmente mediante el método vibrate.

```
String tickerText = "ArrivAlert informa sobre " + ev.getNombre();
NotificationManager nm = (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
Notification n = new Notification(idIcon, tickerText, System.currentTimeMillis());
n.defaults |= Notification.DEFAULT_SOUND;
n.vibrate = new long[] {100,250,100,250,100,450};
n.defaults |= Notification.DEFAULT_LIGHTS;
n.flags    |= Notification.FLAG_AUTO_CANCEL;
```

Figura 18 Detalle de la implementación de la notificación

3.3.3. ALARMAS

En Android la clase encargada de las alarmas es Alarm. Las alarmas permiten programar la ejecución de la aplicación en el futuro sin interacción por parte del usuario.

En *ArrivAlert* se usan principalmente para avisar al usuario de un evento de calendario. Se presentarán al usuario en forma de Toast.

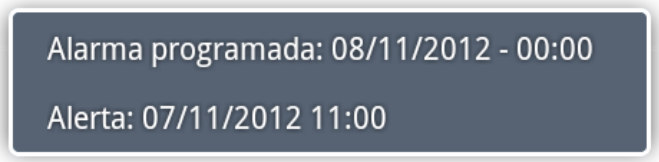

Figura 19 Alarma ejecutada

Su funcionamiento se basa en que la alarma registra un *Intent*, emitiéndolo en el momento programado.

Su gestión se realiza a través de la clase *AlarmManager* mediante instancias de esta clase y de forma indirecta, como sucedía con *NotificationManager* a través de *Context.getSystemService(Context.ALARM_SERVICE).*

La clase BroadcastReceiver actúa de receptor de la alarma. Dicho funcionamiento se recoge implementado en la clase ArrivAlertReceiver.

```
    public static void ProgramarAlertas(Context c, Evento ev) {
        if(ev.getEstado() == 2) {
            AlarmManager alarma = (AlarmManager)c.getSystemService(Context.ALARM_SERVICE);
            Integer i = Integer.valueOf(ev.getId());
            if(alarmasAlertas.containsKey(i)) {
                alarma.cancel((PendingIntent)alarmasAlertas.get(i));
                Toast.makeText(c, "Alerta reprogramada (1)", 0).show();
        }

        Intent i2 = new Intent(c, AlertaReceiver.class);
        i2.putExtra(Constantes.CTE_EVENTO_ID, ev.getId());
        PendingIntent pi = PendingIntent.getBroadcast(c, 7777, i2, PendingIntent.FLAG_ONE_SHOT);
        alarma.set(1, ev.getFechaAlerta().getTime(), pi);
        alarmasAlertas.put(i, pi);
        Toast.makeText(c, "Alerta programada (1): " + ev.toString(), 1).show();
    }

}
```

Figura 20 Detalle de ArrivAlertReceiver

3.4. LOCALIZACIÓN DE UBICACIONES

ArrivAlert permite conocer la ubicación de un dispositivo en tiempo real, por ello se debe implementar utilizando las herramientas propias que usa *Android* para la geolocalización.

No es la única plataforma que ofrece estas prestaciones, pero se distingue del resto por una API de estructura de ubicación cuyo manejo es sencillo y con naturaleza de código abierto.

En este apartado comentaremos la utilidad de dichas herramientas al haber sido empleadas en *ArrivAlert*.

Un proveedor de ubicación se encarga de proporcionar al sistema un conjunto de mediciones relacionadas con la localización, como puede ser la latitud y la longitud.

LocationProvider es una clase abstracta que permite definir las prestaciones de un determinado proveedor. En cada dispositivo, y en función de las circunstancias, pueden existir diferentes implementaciones de proveedor, encargados de devolver información sobre la ubicación.

LocationManager será la clase principal que utilizaremos para interactuar con datos relacionados con la ubicación. A través de ella se obtiene el administrador del servicio, así como los proveedores disponibles.

Dados los requerimientos necesarios para el funcionamiento de *ArrivAlert*, y teniendo en cuenta que se puede usar tanto en exteriores como en interiores, sería necesario que abarcara al proveedor *LocationManager.GPS_PROVIDER*, aprovechando que la mayoría de los dispositivos de Android disponen de receptor GPS, y a *LocationManager.NETWORK_PROVIDER*, aprovechando los puntos de acceso Wi-Fi.

```
private LocationManager locManager;

public void activar() {
   this.locManager.requestLocationUpdates("gps", 5000L, 1.0F, this);
   this.locManager.requestLocationUpdates("network", 5000L, 1.0F, this);
}
```

Figura 21 Detalle de la funcionalidad de los proveedores

LocationListener es una interfaz completa y muy flexible que nos permite filtrar distintos eventos de ubicación en función de diversas propiedades. Para *ArrivAlert* se ha creado la clase Localizador y LocalizadorListenerArrivAlert que extienden de *LocationListener*.

```
public class LocationListenerArrivAlert implements LocationListener {

   private Context ctxt;
   private GeoPoint gploc;
   private ArrayList<ILocationChangedHandler> listListen;
   private LocationManager locManager;
```

Figura 22 Detalle de LocalizadorListenerArrivAlert

Desde ella se reciben actualizaciones de cualquier ubicación. Para obtener la actualización de la ubicación del dispositivo, se debe trabajar en conjunción con *LocationManager*.

```
public void onLocationChanged(Location loc) {
   this.gploc = new GeoPoint((int)(1000000.0D * loc.getLatitude()),
                             (int)(1000000.0D * loc.getLongitude()));
   if(this.gploc != null && this.listListen != null) {
      Iterator<ILocationChangedHandler> it = this.listListen.iterator();

      while(it.hasNext()) {
         ((ILocationChangedHandler)it.next()).LocationChanged(this, this.gploc);
      }
   }

}

public void onProviderDisabled(String s) {}

public void onProviderEnabled(String s) {}

public void onStatusChanged(String s, int i, Bundle b) {}

public void removeLocationChangedListener(ILocationChangedHandler lch) {
   if(this.listListen != null) {
      this.listListen.remove(lch);
   }

}
```

Figura 23 Actualizar ubicación mediante LocationListener

En el método de la figura anterior se obtiene la latitud y longitud de la ubicación correspondiente a la última vez que se localizó el dispositivo.

Estos datos se usan para crear un objeto *GeoPoint*, compuesto por estos valores de latitud y longitud, siendo necesario multiplicar

por un millón esas coordenadas geográficas, ya que este objeto opera en micrograd os, y es así como es interpretado correctamente por un objeto *MapController*.

La clase *MapController*, junto a *MapView* y *MapActivity*, son las empleadas para la creación y representación de mapas.

La clase *MapView* permite mostrar un mapa como un objeto View en el layout *mapa*. Un *MapView* solo puede usarse en una actividad del tipo *MapActivity*. Ambas clases pertenecen al paquete de *Google Maps* **com.google.android.maps**.

```
public class MapaActivity extends MapActivity
{
    private Localizador mLocationListener;
    private OverlayArrivAlert overlay;
    private int mode;

    public MapView mu;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        this.setContentView(R.layout.mapa);

        //Mapa de Ubicación de ArrivAlert
        mu = (MapView)findViewById(R.id.mapaUbicacion);

        mu.setSatellite(false);
        mu.setTraffic(true);
```

Figura 24 Detalle de MapaActivity

Para utilizar este paquete en *ArrivAlert* se solicitó previamente una clave del API de *Google Maps* (ver el apéndice B para más información). Una vez obtenida, debemos incluirla en nuestro objeto *MapView* asignando la propiedad apiKey.

```
<com.google.android.maps.MapView
   android:id="@+id/mapaUbicacion"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:clickable="true"
   android:apiKey="096uaizC7wlWxe6Q9bSEev4STB-duPA0Azu9KCw"
       />
```

Figura 25 API Key de ArrivAlert

Una vez que hemos creado un mapa visual, mediante un objeto MapView, es la hora de interactuar con él mediante la clase Overlay.

Overlay permite dibujar en la pantalla, lo cual es de gran utilidad para marcar una posición concreta en el mapa que ofrece nuestra aplicación. Cabe destacar el método **onTouchEvent** que gestiona las acciones de tocar de pantalla. ArrivAlert dispone de su propia clase que extiende de Overlay.

```
public class OverlayArrivAlert extends com.google.android.maps.Overlay {
```

Figura 26 Detalle de OverlayArrivAlert

Para completar la funcionalidad que ofrece Overlay, se ha creado una clase que extiende de ItemizedOverlay, para incluir puntos de interés (como marcadores de boya) mediante objetos OverlayItem.

ArrivAlert utiliza la clase ItemizedOverlayArrivAlert con esta finalidad, agregando como atributos objetos de la clase Bitmap y Canvas, a los que ya habían sido mencionados anteriormente.

```
public class ItemizedOverlayArrivAlert extends ItemizedOverlay<OverlayItem> {

private ArrayList<OverlayItem> overlayItemList = new ArrayList<OverlayItem>();
private MapView mv;
private Bitmap pica;
private Paint innerPaint, borderPaint, textPaint;
private Bitmap marca;
private GeoPoint gpoint;
private OverlayItem overItem;
```

Figura 27 Atributos de ItemizedOverlayArrivAlert

Los objetos de Bitmap y Canvas funcionan de forma conjunta para crear la superficie de dibujo, delegando en la función dibujaBitMap

```
public static Bitmap dibujaBitmap (Drawable dibuja) {

    if (dibuja instanceof BitmapDrawable) {
        return ((BitmapDrawable)dibuja).getBitmap();
    }

    Bitmap bitmap = Bitmap.createBitmap(dibuja.getIntrinsicWidth(),
            dibuja.getIntrinsicHeight(), Config.ARGB_8888);
    Canvas canvas = new Canvas(bitmap);
    dibuja.setBounds(0, 0, canvas.getWidth(), canvas.getHeight());
    dibuja.draw(canvas);

    return bitmap;
}
```

Figura 28 Método dibujaBitmap

Sin embargo para el cálculo de la distancia en kilómetros (método getDistanciaKm) se ha empleado el método **distanceBetween** de la clase Location en conjunción con GeoPoint. Este método se aleja algo de la distancia real, ya que la calcularía en línea recta.

Todo lo mencionado anteriormente se recoge en nuestra clase Localizador (que recordemos extendía de *LocationListener*) para actualizar nuestro mapa.

```
public class Localizador implements LocationListener {

    private MapController mapController;
    private MapView mapView;
    private LocationManager manejador;
    private GeoPoint lastLocation;
    private Context context;
    private ItemizedOverlayArrivAlert itemizedOverlay;
```

Figura 29 Detalle de Localizador

En ella se reciben las actualizaciones necesarias para refrescar la posición actual en el mapa de *ArrivAlert*.

```
@Override
public void onLocationChanged(Location location) {
        int lat = (int) (location.getLatitude()  * 1E6);
        int lng = (int) (location.getLongitude() * 1E6);
        this.lastLocation = new GeoPoint(lat, lng);

        // Refrescar posicion actual
        if(this.lastLocation != null) {
            if(this.itemizedOverlay.getCurrentLocation() == null) {
                this.mapController.setCenter(this.lastLocation);
            }


             this.itemizedOverlay.setCurrentLocation(this.lastLocation);
             this.mapView.invalidate();
        }
}

@Override
public void onProviderDisabled(String provider) {}

@Override
public void onProviderEnabled(String provider) {}

@Override
public void onStatusChanged(String provider, int status, Bundle extras){}
```

Figura 30 Refrescar posición actual en el mapa

Para terminar esta sección comentaremos el uso que se hace en ArrivAlert de la geocodificación inversa, que consiste en la obtención de una dirección a partir de unas coordenadas. Se realiza mediante la clase Geocoder que permite acceder a una lista de objetos de tipo Address.

```
Geocoder geoCoder = new Geocoder(mapView.getContext(), Locale.getDefault());
List<Address> addresses = geoCoder.getFromLocation(
                        p.getLatitudeE6()/1E6, p.getLongitudeE6() / 1E6, 1);
```

Figura 31 Geocodificación inversa

Resulta de gran utilidad para registrar las ubicaciones de ArrivAlert mediante direcciones postales, que son más informativas de cara al usuario que las coordenadas geográficas.

3.5. INTERFAZ DE USUARIO

Android permite la realización de interfaces de usuario mediante el uso de archivos de diseño en XML combinados con las acciones y actividades que pueden recogerse en los archivos de tipo Java.

Es una ventaja porque de esta manera se pueden realizar pequeños cambios en la programación de esa interfaz sin tener que hacer grandes cambios de código.

ArrivAlert funciona bajo una interfaz lo más intuitiva y atractiva posible para el usuario. Para ello, dispone de una pantalla de inicio donde se muestra su logotipo en la parte superior.

Figura 32 Pantalla principal de ArrivAlert

En la zona inferior se encuentra la parte en la que usuario puede interactuar con la aplicación, mediante cuatro botones que definen la funcionalidad de *ArrivAlert*, dichos botones son *Calendario, Alertas, Ubicaciones* y *Configuración*.

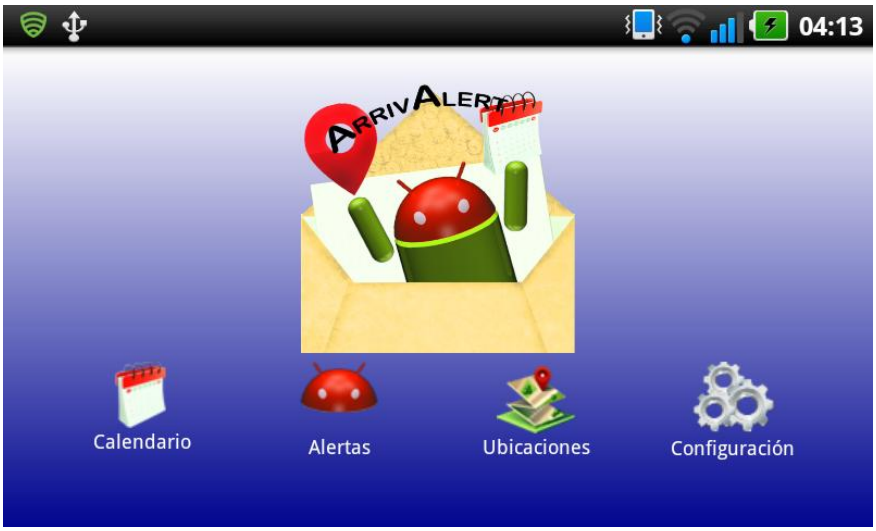

Figura 33 Vista Horizontal de la pantalla principal de ArrivAlert

También dispone de un menú contextual en el que se puede acceder a las opciones de Manual, Acerca De y Salir.

Figura 34 Menú contextual de ArrivAlert

3.6. DEFINIENDO PREFERENCIAS

Los parámetros con los que trabajaremos en la aplicación serán definidos como preferencias para ofrecer mayor variedad de casos al usuario.

A la hora de confeccionar el archivo de configuración creamos un archivo de preferencias al que llamaremos *preferencias.xml*

La etiqueta ***PreferenceScreen*** resulta imprescindible para que nuestro menú de ajustes se visualice como un menú de preferencias:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
```

Figura 35 Fragmento de PreferenceScreen

La etiqueta ***PreferenceCategory*** determinará cada uno de los apartados de nuestro menú de configuración (para distinguirlos podemos usar el atributo ***title***)

```
<PreferenceCategory android:title="Configuración básica">
```

Figura 36 Fragmento de PreferenceCategory

Dentro de cada etiqueta de *PreferenceCategory* podemos definir otras etiquetas que servirán a modo de submenús, dentro de cada apartado. Las utilizadas en esta aplicación son las siguientes:

- ***ListPreferences***: Determina una serie de preferencias a modo de lista (proporciona múltiples opciones de elección)

```
<PreferenceCategory android:title="Avisos">

<ListPreference
  android:entries="@array/radio"
  android:key="kms"
  android:title="@string/radio"
  android:summary="@string/radiexplain"
  android:defaultValue="2000"
  android:entryValues="@array/radioValues"
  />

</PreferenceCategory>
```

Figura 37 Fragmento de ListPreference

- ***CheckBoxPreference***: Determina una serie de preferencias a modo de checkbox (la opción sólo se cumple como verdadera o falsa)

```
<PreferenceCategory android:title="Configuración básica">

<CheckBoxPreference
 android:key = "enable"
 android:title = "@string/enable"
 android:summary = "@string/enablexplain"
 android:defaultValue = "true"/>

</PreferenceCategory>
```

Figura 38 Fragmento de CheckBoxPreference

Estas preferencias están relacionadas con los métodos de la clase **Preferencias**, siendo un modo básico de almacenamiento de datos que usa Android.

```
public class Preferencias {

    public static String PREFERENCIAS ="es.rediris.forja.jleon.arrivalert_preferences";
    public static String RADIO = "radio";
    public static String HABILITAR = "enable";


    public static float getRadio(Context c) {
         SharedPreferences pref = c.getSharedPreferences(
         PREFERENCIAS, android.content.Context.MODE_PRIVATE);

         return pref.getFloat(RADIO, 2.0f);
    }
    public static boolean getEnable(Context c) {
         SharedPreferences pref = c.getSharedPreferences(
         PREFERENCIAS, android.content.Context.MODE_PRIVATE);

         return pref.getBoolean(HABILITAR, false);
    }
}
```

Figura 39 Clase Preferencias

3.7. ASIGNANDO PERMISOS

Para proteger ciertos recursos y características especiales propias del hardware de Android, hay que crear un determinado esquema de permisos.

Como información al usuario, el sistema avisa de los permisos que lleva integrada una determinada aplicación en el momento previo de la instalación, para que sea el propio usuario el que tenga la última palabra a la hora de instalarla.

De cara a la aplicación, siempre se debe declarar la intención de usar estos recursos porque, en caso contrario, se generaría una excepción de permiso que obligaría a cerrar la aplicación de forma brusca.

Este esquema de permisos debe generarse en el archivo ***AndroidManifest.xml***, siguiendo una estructura definida para cada tipo de recurso a utilizar.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<user-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
```

Figura 40 Esquema de permisos de ArrivAlert

A continuación enunciaremos cada uno permisos que se han definido para el funcionamiento de *ArrivAlert*, así como la descripción de sus principales características:

- **INTERNET**: Pertenece al grupo de las comunicaciones de red, encargándose del acceso íntegro a Internet, convirtiéndolo por ello en el posiblemente más destacado de los permisos que se otorgan. Resulta imprescindible en nuestra aplicación ya que se necesita para automatizar el proceso de envío de correo electrónico**.**

- **ACCESS_COARSE_LOCATION**: Se corresponde con el proveedor NETWORK_PROVIDER, siendo capaz de devolver una ubicación basada en conexión a la red (Cel-ID y Wi-Fi).

- **ACCESS_FINE_LOCATION**: Sirve para determinar una ubicación basada en GPS. Se corresponde con el proveedor GPS_PROVIDER.

- **ACCESS_NETWORK_STATE**: Comunica el estado de la red en una conexión mediante torres y puntos de acceso.

- **ACCESS_WIFI_STATE**: Pertenece al grupo de las comunicación de red. Este permiso informa si existen redes Wi-Fi disponibles y cuál es su estado, ya que si no hay encuentra un estado óptimo, el dispositivo trabajaría con su red de tarificación correspondiente, en caso de poseerla.

- **VIBRATE**: Controla parte del hardware del dispositivo, permitiéndole hacer vibrar. Esta opción también hace que se pueda configurar permitiendo que se defina una gama de vibraciones distintas.

- **RECEIVE_BOOT_COMPLETED**: Es capaz de actuar como receptor de la información, accionado incluso si el dispositivo se encuentra apagado.

- **WAKE_LOCK**: Permite atender a una petición cuando el dispositivo se encuentra en suspensión. Es perfecto para recibir notificaciones en *ArrivAlert* cuando el sistema no se encuentra en activo.

- **GET_ACCOUNTS**: Permite el acceso a las cuentas de correo del dispositivo, para obtener datos como nombre de la dirección de correo electrónico.

- **SEND_SMS**: Permite a la aplicación mandar mensajes del tipo SMS bajo pago posterior.

En conclusión, me gustaría comentar que buena parte de estos permisos que utiliza *ArrivAlert* son considerados dentro de los muy elevados en riesgo de seguridad, según algunas fuentes expertas en Informática, aunque resultan imprescindibles para el buen funcionamiento de la aplicación. Por lo que recomiendo desde aquí desconfiar, en principio, de toda aplicación de la que se desconozca su autoría y que utilice permisos que no use la aplicación, ya que podría aprovechar estas vulnerabilidades en beneficio propio para, por ejemplo, la obtención de datos personales mediante el acceso a Internet.

3.8. BASE DE DATOS

ArrivAlert ha utilizado un sistema de base de datos procedente de las clases que dispone *Android* para el caso, que proceden de la plataforma *SQLite*.

SQLite es un sistema de gestión de base de datos que utiliza el lenguaje *SQL* (Structure Query Language) siendo éste un lenguaje de consulta y acceso a bases de datos ampliamente utilizado en muchos sistemas.

Hay una base de datos para cada uno de los elementos principales de *ArrivAlert*, es decir, para los eventos, para las ubicaciones y para las alertas.

Como ya comentamos en el apartado de *Disposición de la información*, estos elementos constituyen una parte vital de la aplicación, siendo necesario almacenar de forma permanente sus contenidos para posteriores consultas.

Para cada uno de ellos se ha creado una base de datos con un nombre identificativo (**bd_eventos**, **bd_ubicaciones** y **bd_alertas**) y en cada archivo que define estas bases de datos se ha creado una serie de funciones para la creación, consulta, modificación y eliminación de los elementos que las constituyen.

Figura 41 Detalle de la vista DDMS de Eclipse

Para la creación de estos archivos es necesario extender de la clase **SQLiteOpenHelper** e importar la declaración de **SQLiteDatabase**, todas ellas del paquete **android.database.sqlite**.

4. PRUEBAS

Las pruebas realizadas a ArrivAlert han sido efectuadas desde distintas ubicaciones y desde varios dispositivos móviles.

Como la aplicación depende de las ubicaciones y del envío de mensajería instantánea, se han dispuesto unas pruebas donde interviene la cercanía que existe hasta el lugar del evento y la selección de tipo de mensajería u otro.

4.1. PRUEBA DE CERCANÍA

Para esta prueba se ha escogido como ubicación de partida mi domicilio familiar en Córdoba y como ubicación del evento la estación de tren Córdoba-Central, que se encuentra a poca distancia.

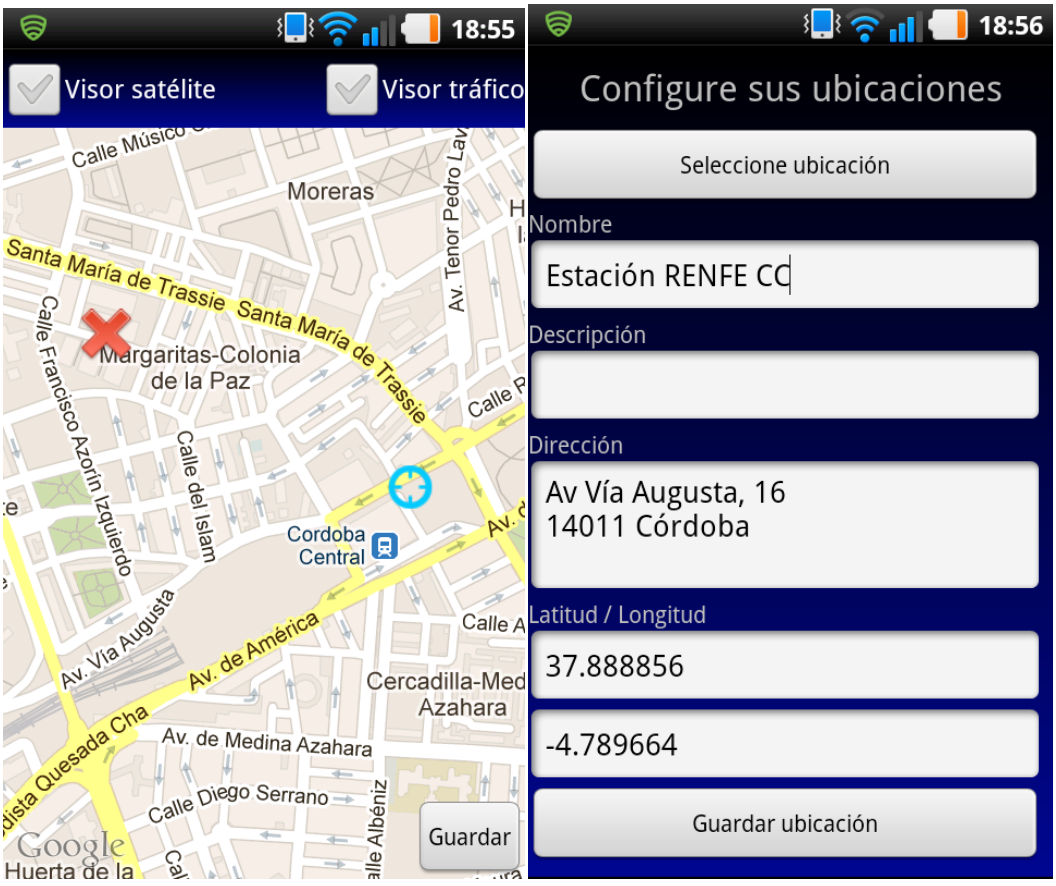

Figura 42 Ubicaciones de partida y de destino

En el calendario de eventos configuramos nuestra prueba de cercanía, seleccionando la ubicación del evento y eligiendo un radio de alerta superior a la distancia que hay entre los dos lugares, para que ArrivAlert determine que estamos cerca de nuestro destino.

También es conveniente, aunque dado el caso no es necesario ya que estamos cerca, asociar una alerta de aviso a uno de los contactos del dispositivo, que actuará como invitado al evento.

Figura 43 Editando y asociando alerta al evento Prueba Cercanía

Una vez que se llega a la fecha para la alerta, se muestra por pantalla que la alarma ha sido activada, por lo que el servicio que ofrece ArrivAlert ya está en funcionamiento y calculando la distancia entre las dos ubicaciones.

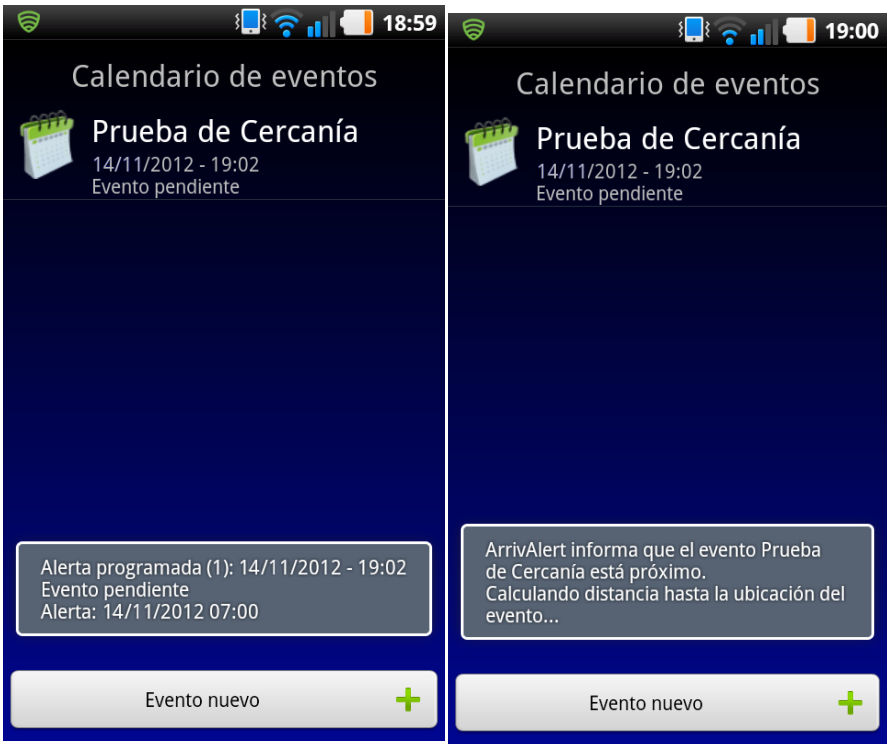

Figura 44 Alarma y Servicio activos en Prueba de Cercanía

A continuación se activa la alerta, mostrando la distancia entre las dos ubicaciones, y cambiando por tanto el estado asociado al evento.

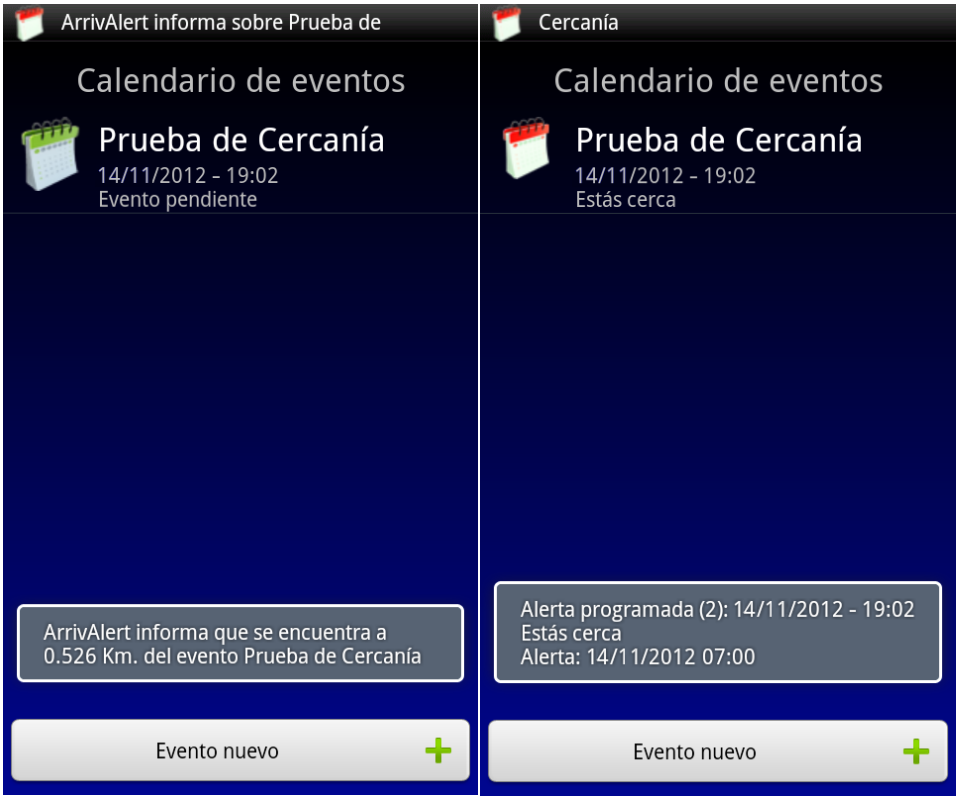

Figura 45 Cálculo de la distancia y cambio del estado del evento

Esto verifica que la prueba de cercanía funciona, también lo podemos comprobar mediante el notificador de la barra de tareas.

Figura 46 Notificación de prueba de Cercanía

Una vez que se ha alcanzado la fecha de inicio del evento, ArrivAlert avisará que el evento ya ha comenzado y volverá a cambiar el estado del evento.

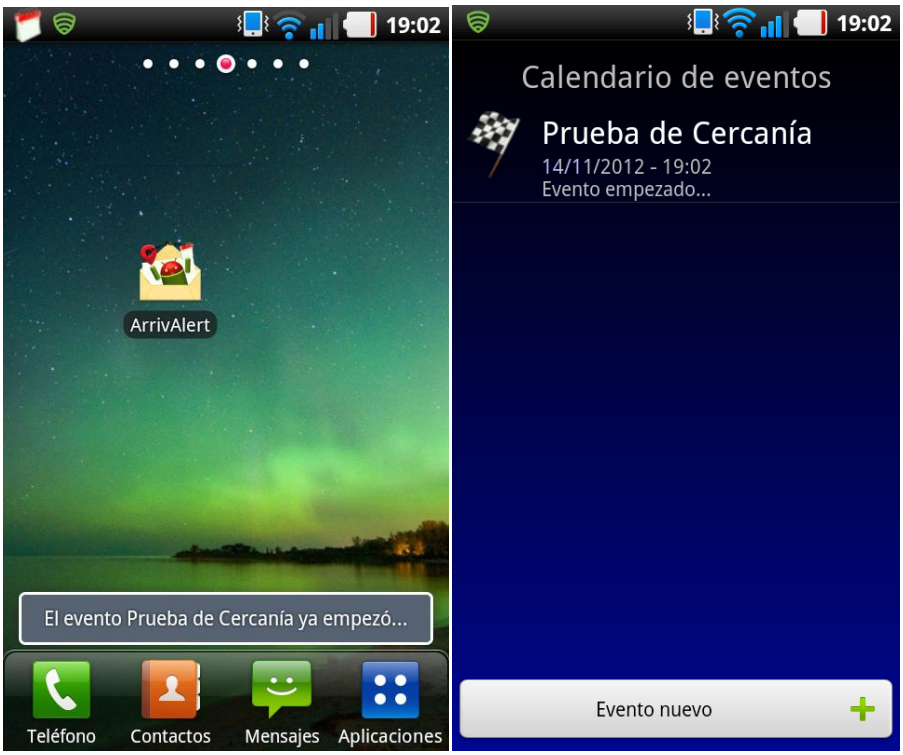

Figura 47 Estado empezado para Prueba de Cercanía

Obsérvese que *ArrivAlert* no ha llegado a mandar ningún mensaje instantáneo porque la distancia entre las dos ubicaciones (algo más de medio kilómetro) no superaba el radio de alerta establecido para este evento (dos kilómetros). Lo cual es también una prueba de buen funcionamiento.

4.2. PRUEBA DE LEJANÍA

Para la prueba de lejanía dispondremos del mismo punto de partida de la prueba anterior, pero en esta ocasión elegiremos como ubicación de destino a la Escuela Técnico Superior de Ingeniería Informática en Sevilla, para asegurarnos que no va a ser posible llegar a tiempo y entonces se tenga que enviar un mensaje de aviso al contacto que elijamos como invitado en la prueba.

Figura 48 ETSII como ubicación de Prueba de Lejanía

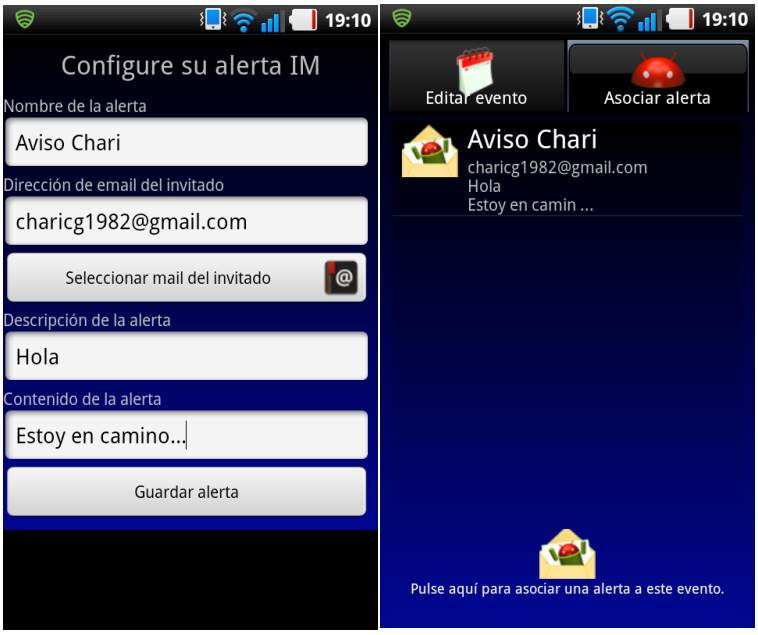

Figura 49 Invitado al evento Prueba de Lejanía

La prueba se desarrolla de manera análoga a la anterior en el momento en el que se alcanza la fecha de alerta (se reciben las alarmas, se crean e inician los servicios, ...)

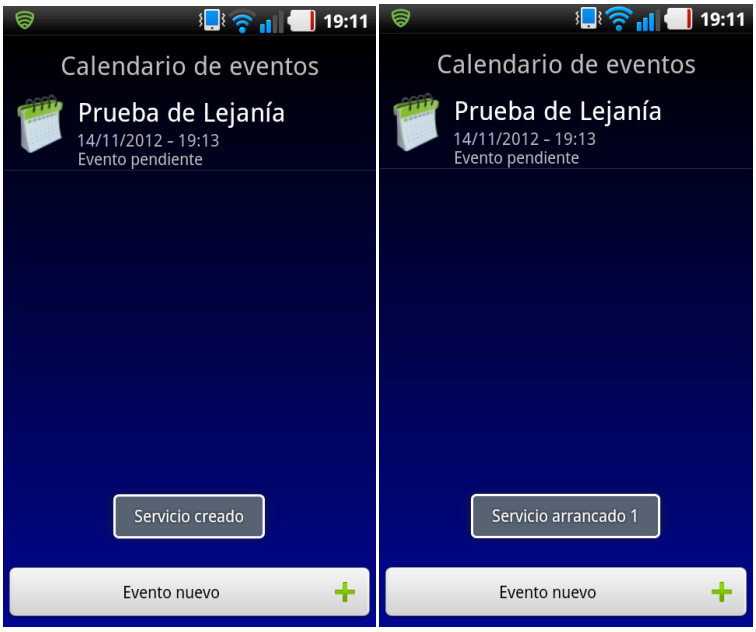

Figura 50 Servicio creado e iniciado en Prueba de Lejanía

La diferencia se produce tras el cálculo de la distancia, en la que es claramente mayor al radio de alerta elegido, ya que ArrivAlert advierte de la lejanía al evento mediante ese icono.

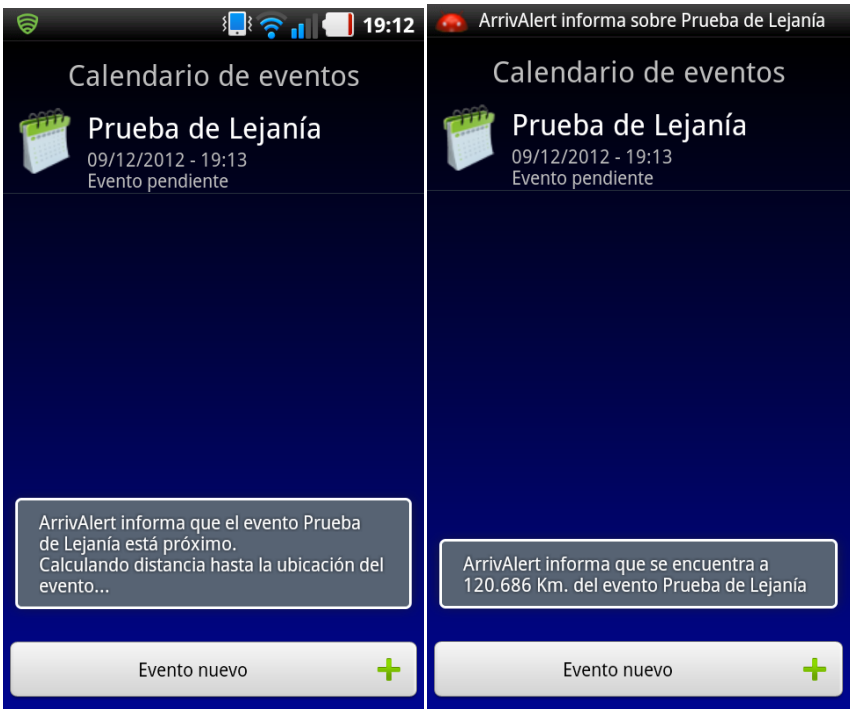

Figura 51 Cálculo de la distancia en Prueba de Lejanía

La notificación nos confirma que nos encontramos lejos de la ubicación destino y por tanto nos vamos a poder llegar a tiempo.

Figura 52 Notificación de Prueba de Lejanía

Por lo tanto, *ArrivAlert* nos dará la oportunidad de enviar un mensaje instantáneo al invitado que hemos elegido. Se abrirá de forma automática un menú para poder elegir con que medio enviaremos ese aviso. Para el ejemplo se ha elegido **Gmail**.

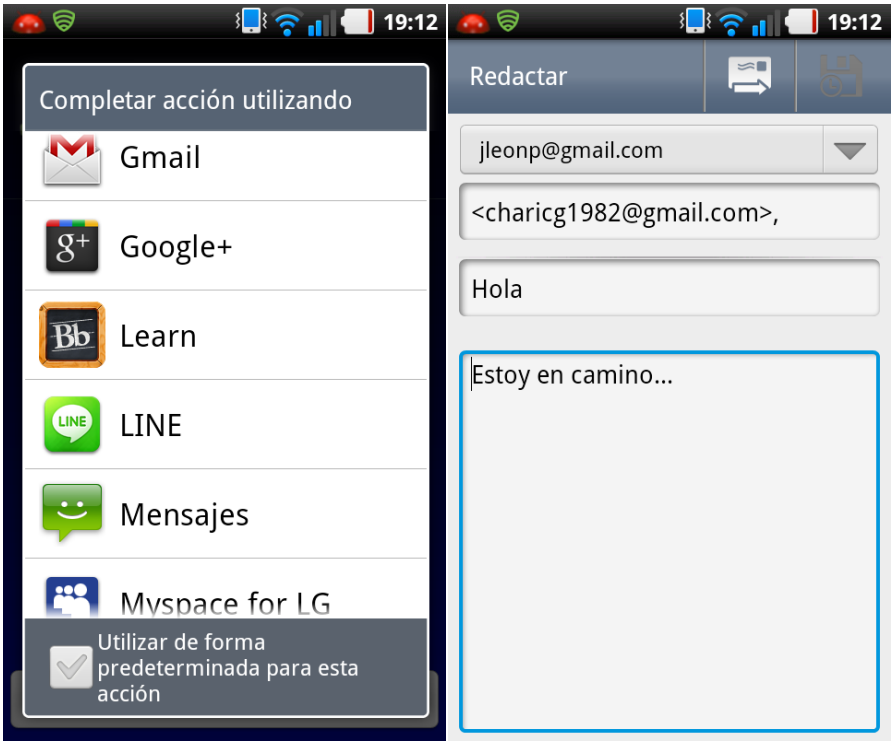

Figura 53 Ejecución de aviso IM

Este tipo de prueba también se ha completado con éxito. Es conveniente no usar un radio de alerta muy elevado, ya que alteraría la interpretación de los datos. En este ejemplo si se hubiera dispuesto un radio mayor de 120 Km, el resultado sería que estaríamos cerca del lugar del evento, y en consecuencia no se enviaría ningún aviso al invitado.

4.3. PRUEBA MÚLTIPLE

Es habitual a la hora de utilizar una aplicación que gestiona eventos de calendarios, tener varios eventos pendientes de ser ejecutados. Una prueba múltiple hace referencia a que haya varios eventos pendientes a la vez.

Figura 54 Múltiples eventos pendientes

ArrivAlert funciona con total normalidad aunque, realizando pruebas múltiples al respecto, se ha observado que algunos eventos se quedan en estado pendiente cuando sus fechas de alerta se solapan en el tiempo con las fechas de otros eventos, por lo que nunca llegan a ser atendidos.

Esto se debe a que se llama más de una vez al método ***onStartCommand()*** que es el encargado de arrancar el servicio, por lo que una solución posible podría ser crear y gestionar *una cola de servicios*. Aunque esta posibilidad es tan sólo una idea, habría que dejar su implementación para desarrollos futuros.

En cualquier caso, para evitar posibles conflictos, lo más conveniente es dejar como mínimo dos minutos de separación entre la fecha de inicio al evento y la fecha de alerta del evento siguiente, para dejar al menos un margen de un minuto de error para el evento que finaliza y el que empieza a continuación.

4.4. PRUEBA DE MENSAJERÍA INSTANTÁNEA

Recordemos que *ArrivAlert* ofrece la posibilidad de enviar mensajes de alerta, avisando de la imposibilidad del usuario de llegar a la ubicación del evento antes de la fecha de inicio del mismo.

Dichos mensajes son guardados en el sistema como alertas y son enviados por el usuario después que éste reciba una notificación informándole que se encuentra lejos de su sitio de reunión.

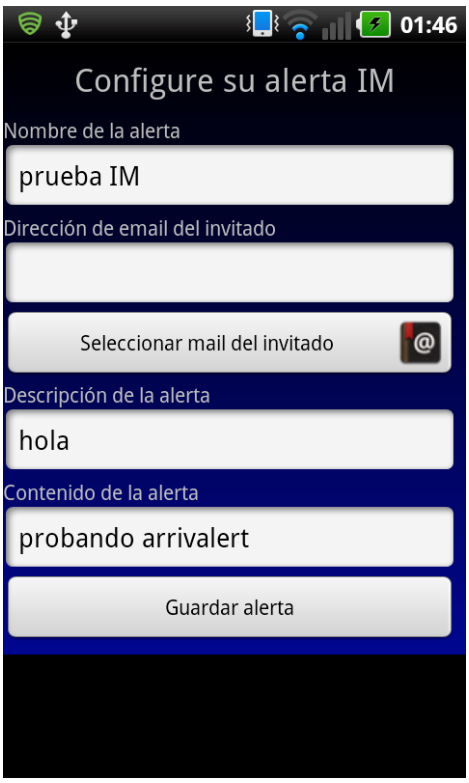

Figura 55 Ejemplo de aviso IM

Llegado el momento, el usuario podrá enviar estos avisos de alertas decidiéndose por un sistema de mensajería en particular de entre los que tenga instalados en el dispositivo móvil.

En esta sección se dará a conocer el resultado de implantación de esos mensajes dentro de las aplicaciones más relevantes de mensajería que ofrece actualmente el mercado de los dispositivos Android.

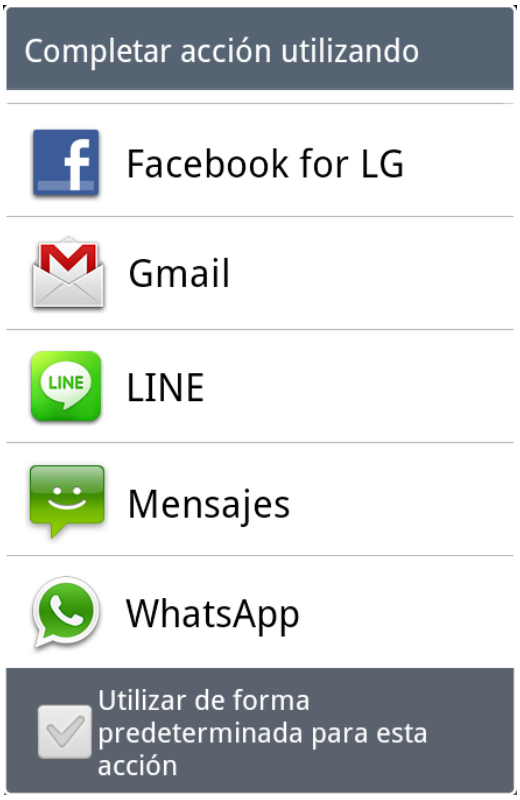

Figura 56 Elección de sistemas de mensajería

El que da mejor resultado es la mensajería de correo electrónico, ya que la configuración de avisos de mensajería instantánea de *ArrivAlert* estaba pensado para albergarlos desde un principio ya que el invitado está asignado mediante una dirección de correo, la descripción del aviso actúa de asunto y su contenido como cuerpo del correo electrónico. Esta opción da un resultado completo.

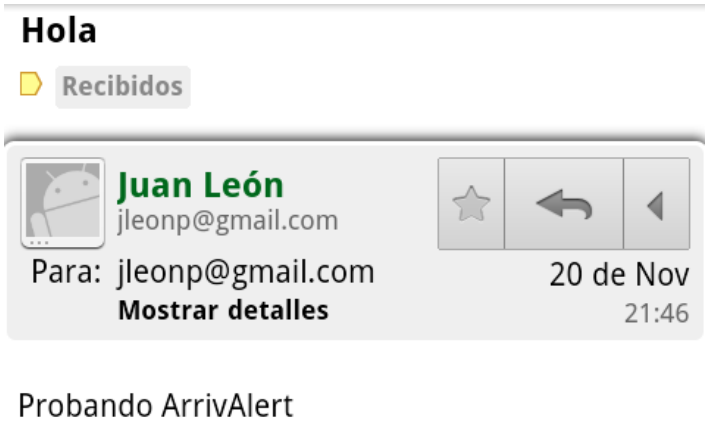

Figura 57 Ejemplo de aviso de IM ejecutado en Gmail

En segundo lugar encontramos las aplicaciones de mensajería SMS, WhatsApp y Line. Ofrecen un resultado a medio camino, ya que aún siendo capaces de enviar el contenido del aviso se debe seleccionar previamente la persona a quién enviárselo.

Figura 58 Ejemplo de aviso de IM ejecutado en SMS, WhatsApp y Line

Por último lugar se probó en la aplicación en Facebook (como representante de aplicaciones que gestionaran una red social) con resultados nulos, no aparecía el contenido del aviso ni tampoco un listado que sugiriera a quién enviarlo.

Por lo tanto, *ArrivAlert* funciona mejor mandando correos electrónicos que cualquier otro tipo de mensajería instantánea, aunque este hecho no excluye que se pueda usar con otras aplicaciones que no aparecen en esta sección. Se recomienda usarlo conjuntamente con correos electrónicos, y en su defecto con las aplicaciones mencionadas en segundo lugar.

Aparte de estas pruebas y aunque *ArrivAlert* funciona plenamente, se han encontrado en algunos casos que la aplicación no reconoce la posición actual del usuario en el mapa, es decir, que al cargarse la interfaz de Google Maps no aparece la cruz roja que señala la ubicación en la que nos encontramos. Esto puede deberse a algún tipo de problema con el proveedor encargado en la localización, a la conectividad del dispositivo, a la versión de núcleo de sistema operativo instalado en el mismo o a la reciente declaración de obsolescencia de la versión 1 de la API de Google Maps (fechada el 3 de diciembre), y de la cual dispone esta aplicación.

5. ANÁLISIS TEMPORAL Y DE COSTES DE DESARROLLO

5.1. MATERIAL EMPLEADO

Para poder llevar a cabo la aplicación se ha dispuesto de una serie de recursos (software y hardware) que han facilitado la consecución del proyecto.

Algunos de ellos han resultado imprescindibles, ya que para simular la aplicación se necesita un equipo informático con bastante cantidad de memoria RAM para que su emulación sea rápida y eficaz.

Además se ha dispuesto de algunos dispositivos en *Android* para probar directamente la aplicación.

En cuanto al software se ha apostado principalmente por software libre, ya que *Android* es una plataforma *OpenSource* y todo lo que implique su desarrollo también lo es.

A continuación se enumera dicho material y se recogen las características más importantes de los mismos.

5.1.1. DISPOSITIVOS HARDWARE

Como hemos mencionado anteriormente, se necesita un equipo lo bastante potente como para simular la aplicación que se está programando.

Es una de las desventajas del simulador que se instala junto al SDK de Android, que requiere de al menos de 2 GB de RAM para empezar a trabajar mediamente.

Por ello se dispuso de un equipo que tratará de minimizar los tiempos de ejecución. Dicho equipo, junto a sus principales características técnicas, es el siguiente:

Ordenador portátil TOSHIBA Satellite L750

- **Procesador**: Intel Core i5-2430M CPU @ 2.40 GHz
- **Memoria RAM**: 4 GB
- **Sistema Operativo**: Windows 7 (64 bits)

Este dispositivo ha sido empleado para la elaboración total del proyecto, ya que se ha empleado tanto para el desarrollo de la aplicación como para la elaboración de su memoria.

Con respecto a lo mencionado con anterioridad sobre la simulación de pruebas, hay que mencionar que también resulto totalmente necesario la utilización de dispositivos móviles para probar la aplicación durante su desarrollo, ya que a pesar de que el simulador de SDK funcionaba con total rapidez, fallaba en algunos testeos simples relacionados con la depuración de la aplicación.

En otras ocasiones resultaba imprescindible el uso de estos dispositivos móviles porque el simulador no contaba con la flexibilidad de éstos a la hora de simular envíos de alertas, como podía hacer con el posicionamiento GPS, para poder recoger el lugar de ubicación del usuario.

Por ello se ha empleado estos dispositivos constantemente, ya que cualquier modificación de carácter puntual realizada en el código era mejor tratada que con el simulador de ofrecía *Android*, debido al carácter especial de la aplicación.

También se han empleado para realizar pruebas de funcionamiento que han servido como depuración para conseguir una aplicación que sea lo más universal posible para todas los dispositivos *Android* disponibles en el mercado.

Describimos a continuación los dispositivos móviles utilizados, siendo el primero el que se ha utilizado para la depuración de la aplicación cuando el simulador resultaba limitado por sus características.

Smartphone LG-P970 (LG Optimus Black)

Figura 59 LG Optimus Black

- Sistema Operativo
  - Android 2.3 Gingerbread (Actualizable)
- Dimensiones alto x Ancho x Profundidad
  - 122 x 64 x 69.2mm
- Peso
  - 109 gr.
- Vibrador: Sí
- Pantalla
  - Táctil: Sí
  - Tamaño de pantalla: 4 pulgadas
- Mensajería
  - SMS: Sí
  - EMS: Sí
  - MMS: Sí
  - E Mail: Sí
- Video llamada: Sí
- Juegos: Sí
- Altavoz manos libres: Sí
- Cámara: Sí
- Sensor de movimiento: Sí
- GPS: Sí
- MP3: Sí
- Comunicaciones
- Internet
- WiFi
- Google
- Bluetooth
- Batería (mAh)

Li-Ion 1500mAh

Smartphone Samsung Galaxy Mini S5570

Figura 60 Samsung Galaxy Mini S5570

- Sistema Operativo
  - Android 2.2 Froyo (Actualizable)
- Dimensiones (Altura x Anchura x Profundidad)
  - 110.4 x 60.6 x 12.1mm
- Peso
  - 105 gr.
- Vibrador: Sí
- Pantalla
  - Táctil: Sí
  - Tamaño de pantalla: 240 x 320 pixeles, 3.14 pulgadas
- Mensajería
  - SMS: Sí
  - IM: Sí
  - MMS: Sí
  - EMail: Sí
- Video llamada: No
- Juegos: Sí
- Altavoz manos libres: Sí
- Cámara: Sí
- Sensor de movimiento: Sí
- GPS: Sí
- MP3 / MP4: Sí
- Comunicaciones
- Internet
- WiFi
- Google
- Bluetooth
- Batería (mAh)

Li-Ion 1200mAh

5.1.2. HERRAMIENTAS SOFTWARE

- **ECLIPSE + Android SDK**

Este famoso entorno de desarrollo, completamente gratuito y de código abierto, supone la mejor herramienta para programar en Android, debido a su fuerte orientación a JAVA.

Incluye todas las librerías necesarias para el desarrollo de aplicaciones, así como una máquina virtual configurable para probarla como simulador. Además, como se ha menciona antes, sobre el empleo de dispositivos físicos, el propio Eclipse puede instalar y ejecutar la aplicación mediante una simple conexión USB, ofreciendo además todo tipo de herramientas para facilitar la programación, depuración de errores, etc.

Para más información consultar la sección de anexo, donde se expone una pequeña guía de instalación, así como los siguientes enlaces:

http://www.eclipse.org

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

- **FORJA REDIRIS**

La Forja de la Comunidad RedIRIS ofrece un amplio repositorio de proyectos de *software libre* con material muy interesante en el que apoyarse. También dispone de un SVN para tener sincronizado el proyecto desde distintos equipos y para tener un registro tanto de los cambios como de las versiones anteriores.

Ha resultado muy útil desde el punto de vista didáctico ya que se aprende bastante a la hora de elaborar la documentación resultante, consultando como guía los proyectos finalizados anteriormente.

Se puede acceder a la Forja mediante el siguiente enlace: http://forja.rediris.es

- **SQLITE**

SQLite es un motor de bases de datos de uso extendido que ofrece características muy interesantes como su reducido tamaño y ser de código libre. También, a diferencia de los sistema de gestión de bases de datos clásicos basados en un modelo cliente-servidor, el motor de SQLite no es un proceso independiente, sino que es la biblioteca SQLite la que se enlaza directamente con el programa pasando a ser parte del mismo.

Como ventaja destaca que se reduzca la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes al no tener comunicación entre procesos.

Para más información consultar el siguiente enlace:

http://www.sqlite.org/

- **ANDROID ASSET STUDIO**

Se trata de una herramienta online con la que puedes crear tus propios iconos.

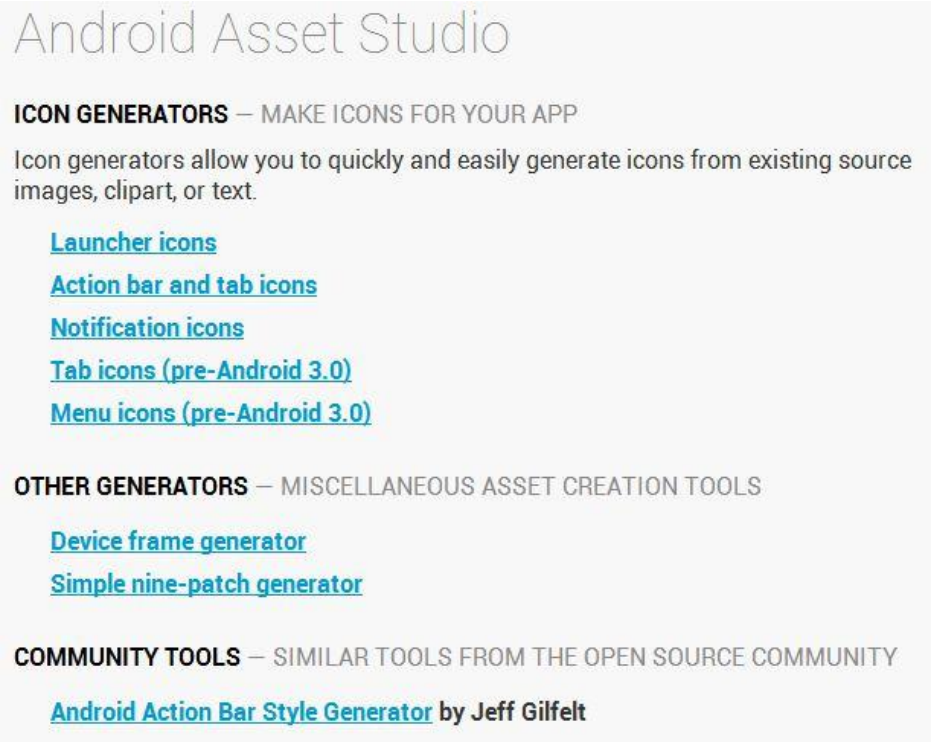

Figura 61 Menú principal de Android Asset Studio

Es muy sencilla de utilizar y contiene una base de datos con algunos diseños predeterminados. Tanto si se empieza desde cero o con una imagen ya diseñada da la posibilidad de descargarla en los tamaños idóneos para la interfaz *drawable* que usa Android en cada proyecto.

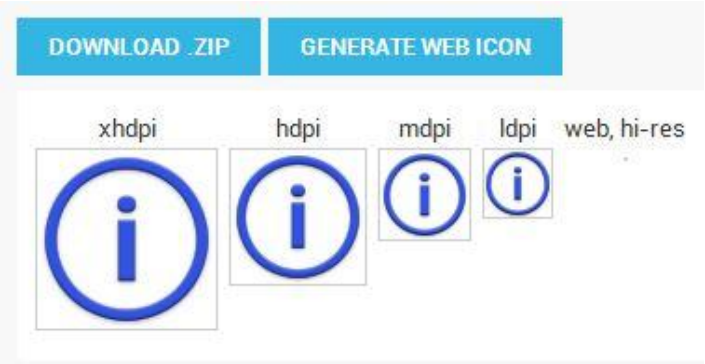

Figura 62 Contenido del paquete de iconos

Esta herramienta se encuentra en el siguiente enlace:

http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

- **ADOBE PHOTOSHOP CS3**

  Programa con el que se diseñó el logotipo y alguno de los iconos de la aplicación.

- **MICROSOFT WORD 2007**

  Programa de procesador de textos con el que se elaboró la presente memoria.

5.2. TIEMPO DEDICADO

En este apartado se expondrá el número de horas dedicadas a este proyecto, para calcular en el siguiente el coste de desarrollo que ha supuesto el mismo, sumándole el coste de las herramientas expuestas en la sección anterior.

El tiempo dedicado a desarrollar ArrivAlert se puede dividir en varias actividades diversas. A continuación las enumeraremos, detallando que se realizó en cada una de ellas.

- **Investigación**

  - Aprendizaje del lenguaje de programación Android mediante material bibliográfico (libros de texto y páginas web).
  - Aprendizaje de las herramientas de desarrollo para Android (instalación de Eclipse + SDK + AVD + ADT).
  - Aprendizaje y análisis de las aplicaciones de calendario y antecedentes a ésta que existen en el mercado.
  - Búsqueda y análisis de los requerimientos técnicos para la implementación de la aplicación (APIs, clases externas, frameworks, etc).

  Tiempo empleado: **154 horas**

- **Implementación**

  - Desarrollo de la idea general de la aplicación
  - Desarrollo de las estructuras de clases
  - Desarrollo de la interfaz de usuario
  - Depuraciones finales

  Tiempo empleado: **242 horas**

- **Tutorías**

  - Consultas con el tutor de proyecto

  Tiempo empleado: **6 horas**

- **Documentación**

  - Recopilación del seguimiento de trabajo
  - Recopilación de elementos bibliográficos
  - Redacción de la memoria del proyecto
  - Elaboración de la presentación

  Tiempo empleado: **182 horas**

- **Pruebas**

  - Test de eficacia con distintos dispositivos
  - Búsqueda de fallos

  Tiempo empleado: **28 horas**

  Tiempo total empleado: **612 horas**

Figura 63 Gráfico de tiempo empleado

5.3. COSTES DE DESARROLLO

En esta sección haremos un cálculo aproximado del coste del proyecto, sumando recursos tecnológicos y humanos.

- **Dispositivos Hardware**
  - Ordenador portátil Toshiba: 650€
  - Smartphone LG Optimus Black : 240 €
  - Smartphone Samsung Galaxy Mini: 120 €

Total: 1010 €

- **Dispositivos Software**
  - Eclipse + Android: 0 €

Total: 0 €

- **Coste programador**
  - Horas trabajadas * sueldo/mes

Suponiendo un sueldo de 1200€/mes y habiendo trabajado 612 horas, siendo una jornada laboral de 8 horas, sin contar los fines de semana, sería entonces un mes de 22 días.

Por lo que el tiempo trabajado es aproximadamente 1,16 meses, es decir 1392 €

Total: 1392 €

Entonces, sumando todas las cantidades anteriores, el coste de desarrollo de ArrivAlert sería de 2402 €.

6. MANUAL DE USUARIO

En esta sección se describirá una breve guía para hacer un buen uso de ArrivAlert.

6.1. INSTALACIÓN

Normalmente las aplicaciones de Android se instalan directamente desde *Google Play Store*, mediante una cuenta de correo electrónico de Gmail. Ya que ArrivAlert no está publicado se puede instalar directamente mediante el archivo APK de Android que lo define.

Por ello, se debe asegurar que el dispositivo permite la instalación de aplicaciones ajenas a la tienda de Google.

La comprobación es tan simple como consultar si está marcada la casilla de *Orígenes desconocidos* en la ubicación *Aplicaciones (Mis programas) > Ajustes > Aplicaciones*

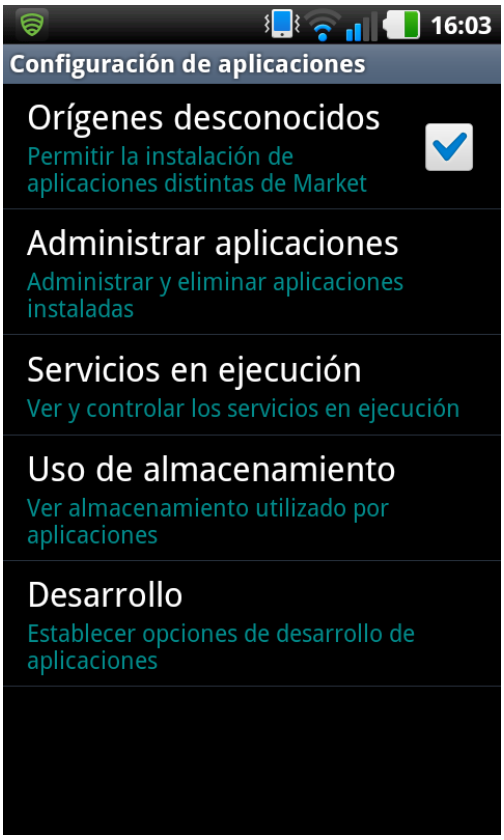

Figura 64 Habilitación de orígenes desconocidos

Una vez realizada la comprobación de la habilitación de la casilla de Orígenes desconocidos, se puede proceder a la instalación del archivo APK.

Normalmente se hace guardándolo dentro de alguna tarjeta externa de memoria compatible con el dispositivo y con la ayuda de otra aplicación que permita la exploración entre los directorios del sistema (*AndroXplorer* y *Root Browser* fueron las aplicaciones utilizadas en este proyecto para ese cometido).

Pulsando sobre él aparecerá el siguiente diálogo, que informa de los permisos que ofrece ArrivAlert (se han hablado de ellos con anterioridad).

Para instalar ArrivAlert sólo hay que pulsar *Instalar*.

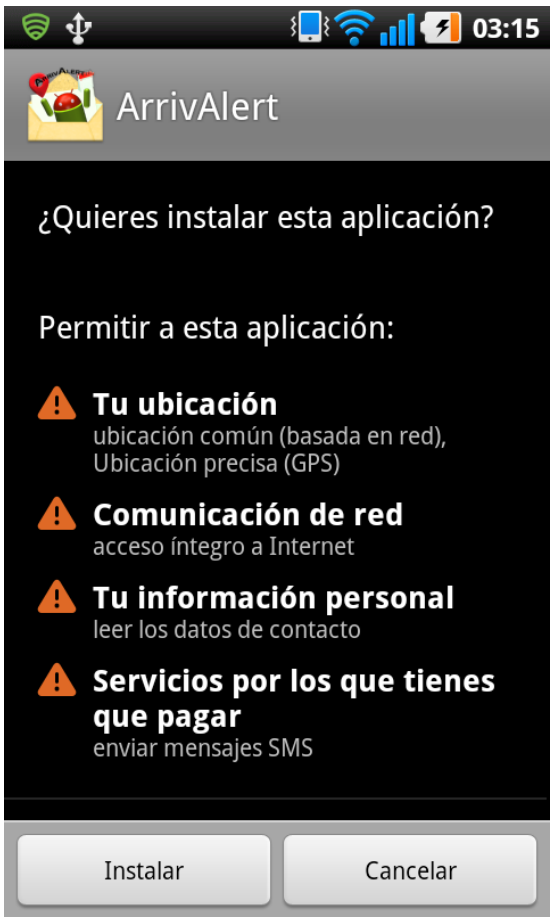

Figura 65 Diálogo de instalación de ArrivAlert

La instalación sólo dura unos segundos, mientras dure el diálogo de *Instalando…*

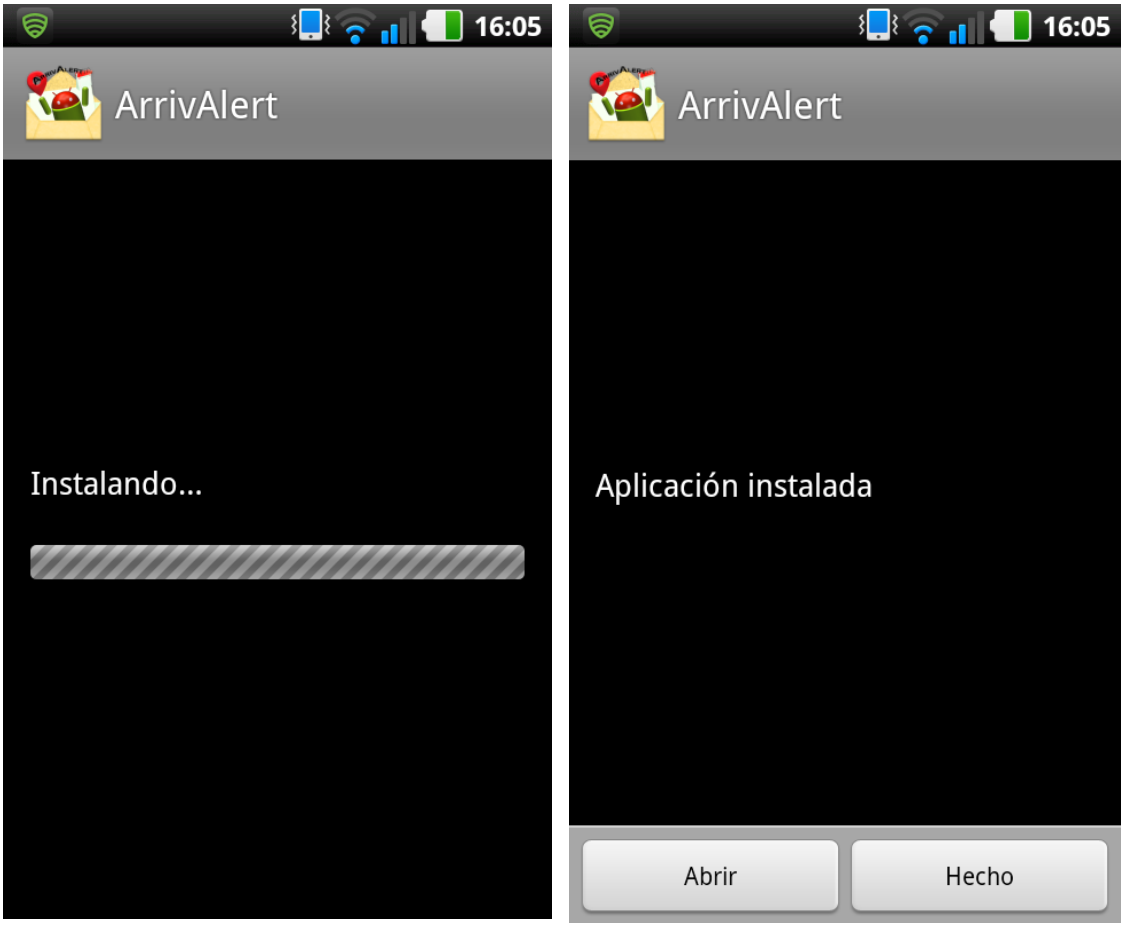

Figura 66 Instalación de ArrivAlert

Una vez que se indique que la aplicación esté instalada, se puede abrir directamente desde ahí mediante el botón *Abrir* o hacerlo posteriormente desde el menú de aplicaciones del dispositivo, pulsando en *Hecho*.

6.2. DESINSTALACIÓN

Para desinstalar ArrivAlert de nuestro dispositivo se debe recurrir al menú de *Ajustes* de Android, ya que se instaló de manera manual.

Siguiendo la ruta Aplicaciones (Mis programas) > Ajustes > Aplicaciones > Administrar aplicaciones > Todos > ArrivAlert llegamos al siguiente diálogo:

Figura 67 Propiedades de ArrivAlert

Para desinstalar ArrivAlert simplemente hay que pulsar el botón *Desinstalar*.

La desinstalación sólo tarda unos segundos, mientras dure el diálogo de *Desinstalando…*

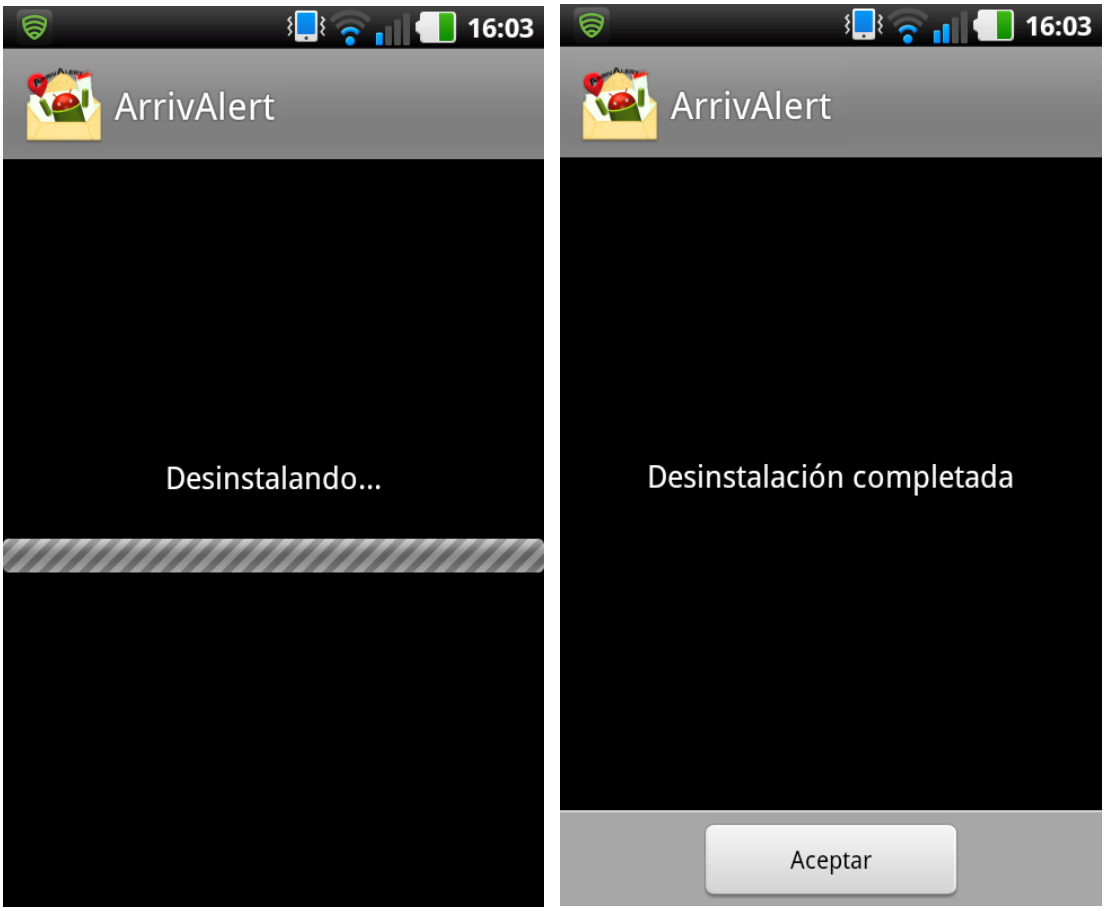

Figura 68 Desinstalación de ArrivAlert

Una vez que se indique que la aplicación esté desinstalada de forma completa, se pulsa *Aceptar*.

6.3. USO DE LA APLICACIÓN

ArrivAlert es sencillo e intuitivo de utilizar. A continuación se detalla una guía de uso.

Lo primero que encontramos al arrancar la aplicación es la vista principal en la que aparece el logotipo en la parte superior y cuatro botones en la inferior.

Figura 69 Vista principal de ArrivAlert

Cada botón registra una funcionalidad distinta de la aplicación, estando cada una de ellas estrechamente relacionadas con las otras.

Con el botón **Calendario** se accede al ***Calendario de eventos***, en el que se recogen todos los eventos de *ArrivAlert*.

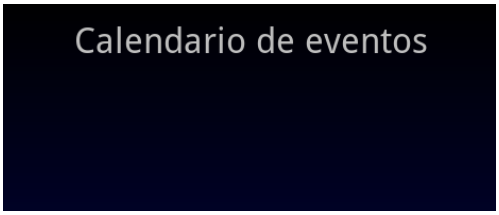

Figura 70 Calendario de eventos vacio

Para crear un nuevo evento tan solo hay que pulsar en el botón que se encuentra en la parte inferior llamado **Evento nuevo**. Una vez pulsado se puede acceder al menú de edición para un nuevo evento.

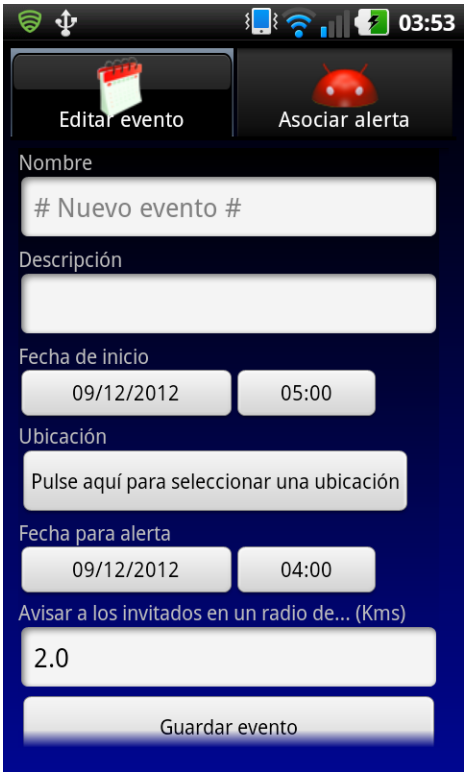

Figura 71 Editar evento

Aquí podemos configurar nuestro evento indicando nombre, descripción y fecha de inicio del mismo, así como su ubicación, fecha de alerta y radio de aviso.

La fecha de inicio indicará el momento en el que empieza el evento, siendo siempre ésta posterior a la fecha de alerta, en la que se notificará al usuario que un evento se encuentra próximo, siendo también el momento en el que se avise a los invitados al evento en caso de no llegar a tiempo.

El radio de aviso indicará el límite por el cual se determinará si se debe enviar el aviso a los invitados, es decir, si la distancia que existe entre la ubicación del usuario y la del evento es superior a ese

radio se consideraría que el usuario no llega puntualmente a la fecha de inicio del evento, por lo que se avisaría a los invitados.

Los avisos se pueden seleccionar pulsando en la pestaña **Asociar alertas** que se encuentra en la parte superior derecha.

Figura 72 Detalle de pestañas del evento

Una vez que se haya pulsado se accede a un menú en el que se nos da la oportunidad de agregar nuevas alertas a nuestro evento. Si no existen alertas asociadas, se indicará en un mensaje en la parte central de la pantalla. Para asociar una nueva alerta sólo hay que pulsar el botón de la parte inferior de la pantalla.

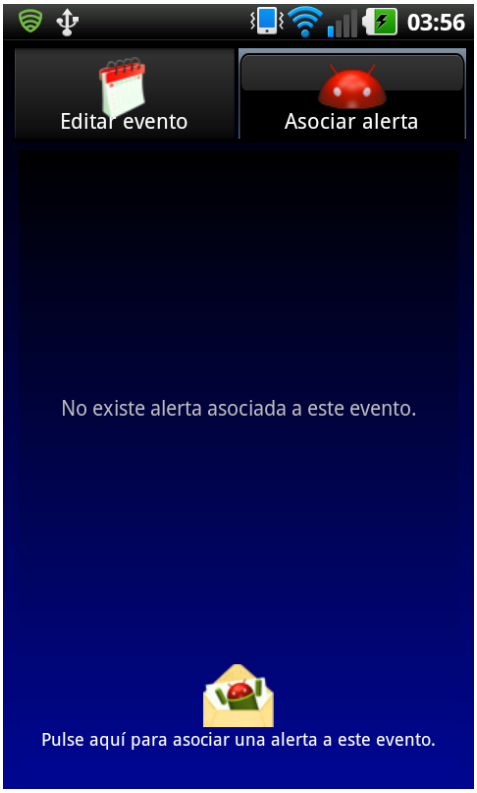

Figura 73 Menú asociar nueva alerta

Cuando se pulse se accede al menú de **Alertas** (también accesible desde el menú principal). Aquí podemos seleccionar una alerta creada anteriormente o configurar una nueva, pulsando el botón de la parte inferior denominado **Nueva alerta**.

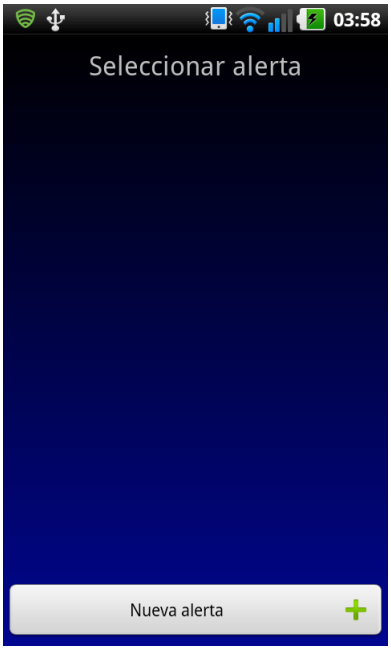

Figura 74 Menú seleccionar alerta

Aquí podemos configurar nuestra alerta indicando nombre, descripción, contenido y dirección de correo electrónico del invitado, pudiéndose también seleccionar ésta desde la agenda de contactos del dispositivo.

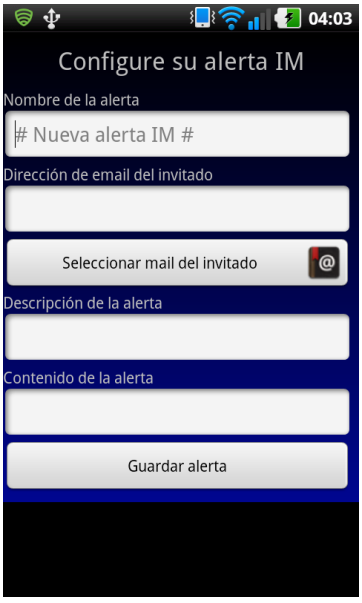

Figura 75 Menú de configuración de alerta IM

Por defecto, la alerta se puede enviar como correo electrónico usando la descripción y el contenido como asunto y cuerpo, aunque *ArrivAlert* permite seleccionar otras vías de mensajería, tantas como aplicaciones relacionadas con la mensajería tenga instalado el dispositivo.

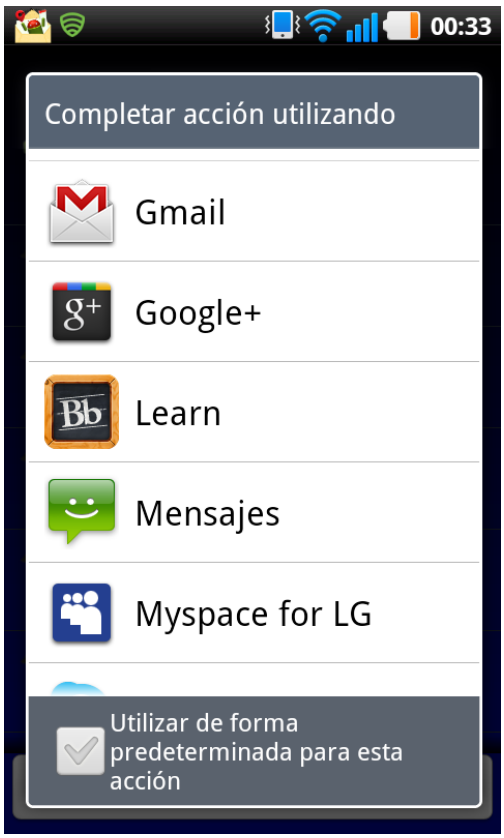

Figura 76 Elección de mensajería para la alerta

Una vez seleccionada la alerta, volvemos a la pestaña de **Asociar alertas** por si deseamos volver a seleccionar otra. En caso de estar conformes con la elección realizada podemos volver al menú de configuración de eventos pulsando en la pestaña de **Editar evento**.

Al elegir ubicación accedemos directamente al menú de **Seleccionar ubicación** (también accesible desde el botón **Ubicaciones** del menú principal) donde podemos seleccionar una ubicación ya creada con anterioridad o elegir una nueva.

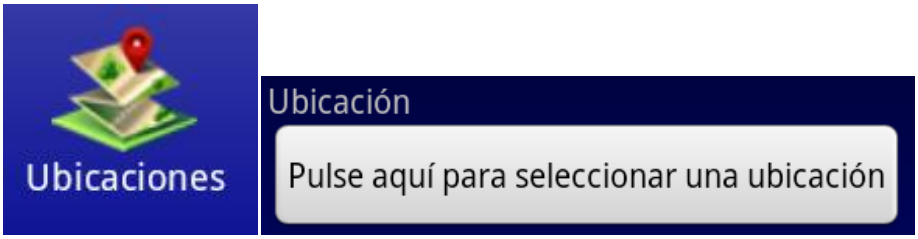

Figura 77 Botones *Ubicaciones*

Dentro de este menú podemos seleccionar una ubicación ya existente o una nueva, pulsando sobre el botón **Ubicación nueva**, situado en la parte inferior.

Dicho botón nos conduce al menú **Configure sus ubicaciones**, donde podemos elegir una nueva ubicación pulsando el botón **Seleccione ubicación**.

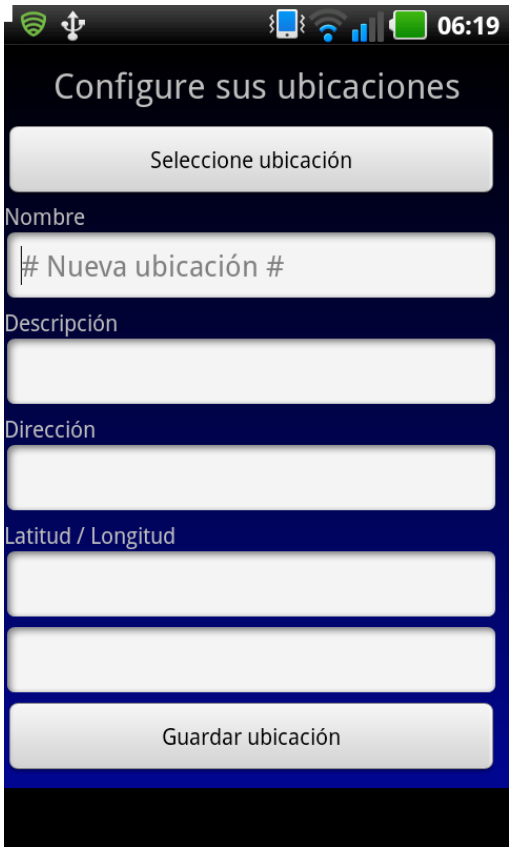

Figura 78 Menú de Configure sus ubicaciones

Al pulsarlo accedemos a un mapa de Google donde la cruz roja señala la ubicación en el que nos encontramos actualmente.

Deslizando el dedo sobre la pantalla nos podemos mover sobre él, controlando también el efecto de zoom en la parte inferior.

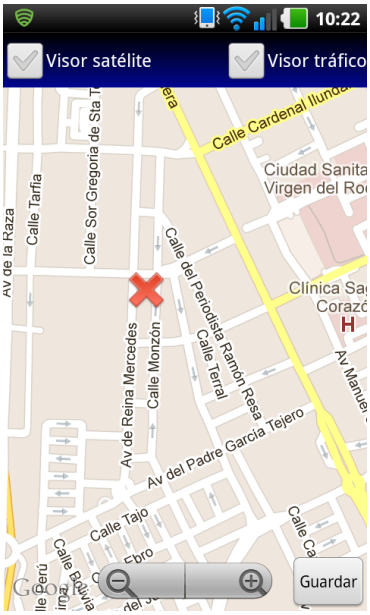

Figura 79 Posición actual del mapa

Pulsando los controles de la parte superior de la pantalla (**Visor satélite** y **Visor tráfico**) podemos controlar la vista por satélite y la vista del tráfico.

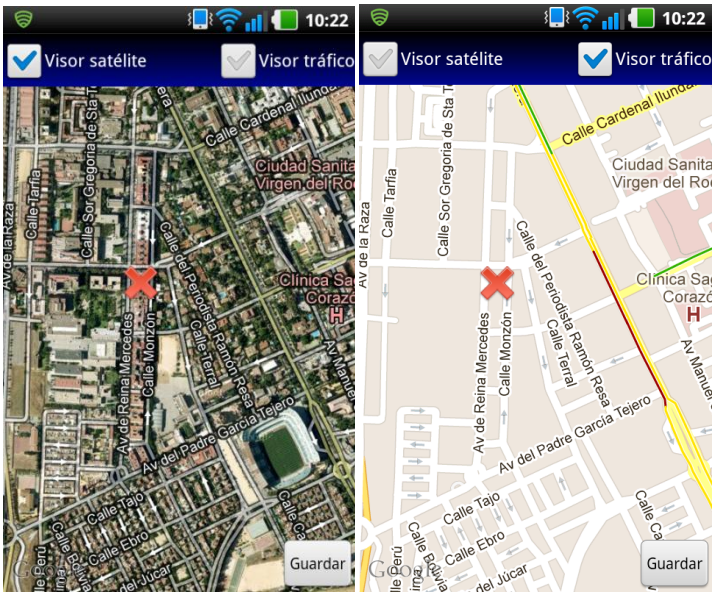

Figura 80 Vistas satélite y tráfico

Para elegir una ubicación basta con pulsar el punto deseado del mapa, aparecerá un símbolo azul. Para guardarlo sólo hay que pulsar el botón guardar que se encuentra en la parte inferior derecha de la pantalla.

Figura 81 Elección de una ubicación

Al guardar la ubicación se rellenan de manera automática los campos de dirección, latitud y longitud. Sólo faltaría rellenar el resto de campos para su configuración.

Figura 82 Edición de una ubicación

Al guardar la ubicación podemos volver a visualizarla seleccionando el botón de **Ver ubicaciones**. También podemos visualizar el nombre con el hemos llamado a nuestra ubicación pulsando sobre el marcador.

Figura 83 Detalle de las marcas

Para añadir una ubicación al evento, sólo hay que seleccionar esa ubicación y se agregará automáticamente al evento.

Figura 84 Detalle de ubicación configurado y en evento

Entonces, una vez que se haya completado la configuración del evento, pulsamos **Guardar evento**, sumando nuestro evento al calendario.

Figura 85 Detalle de calendario con evento agregado

Cada evento muestra una parte de información del mismo (nombre, fecha de inicio y estado) así como un icono relacionado que también indica el estado en el que se encuentra dicho evento.

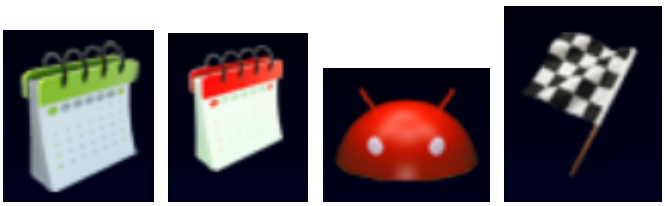

Figura 86 Estado pendiente, puntual, impuntual y empezado

ArrivAlert está activado por defecto, aunque se puede comprobar que esto es así accediendo al menú Configuración.

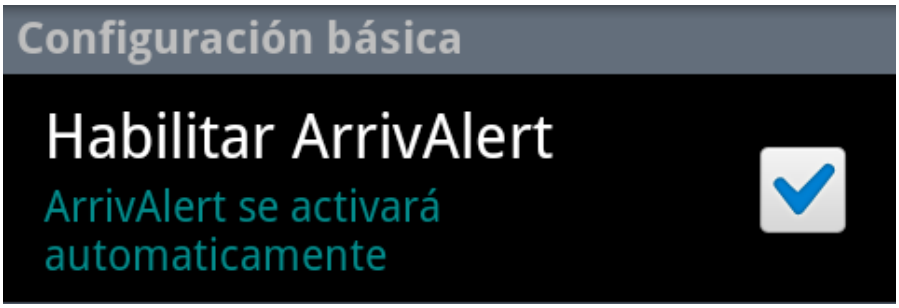

Figura 87 ArrivAlert Habilitado

En el menú Configuración tenemos varios apartados:

En **Configuración básica**, podemos habilitar o deshabilitar nuestra aplicación. Como ya hemos dicho anteriormente, ArrivAlert se encuentra habilitado por defecto, y es conveniente que lo esté para recibir nuestras alertas y notificaciones.

En **Avisos** podemos configurar por defecto nuestro *radio de alerta*. Es conveniente que el valor no sea relativamente alto, para tener mejor estimación en la alerta que le llega al usuario.

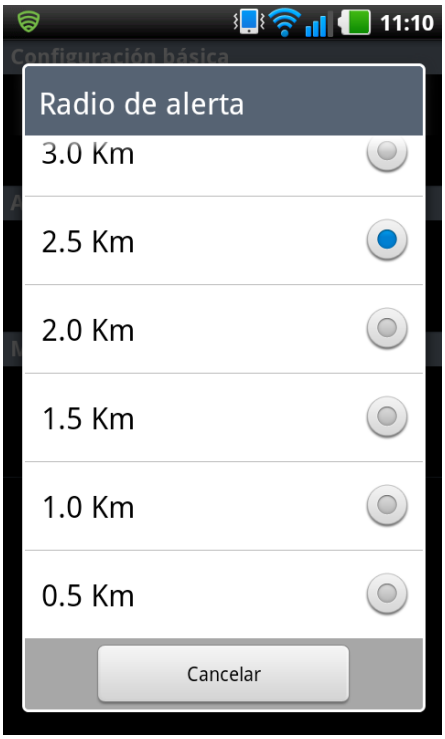

Figura 88 Radio de alerta

En **Mapas**, podemos determinar nuestro *filtro de visión* para visualizar nuestras ubicaciones, por defecto no se muestra ninguno.

Figura 89 Filtro de visión

Otras opciones se pueden visualizar en el menú contextual de la pantalla principal.

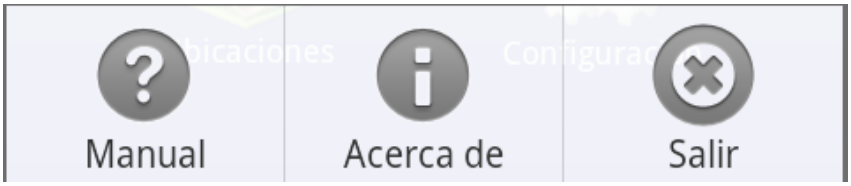

Figura 90 Detalle menú contextual

El menú **Manual** ofrece una guía de instrucciones dirigidas al usuario para que aprenda el manejo de *ArrivAlert*

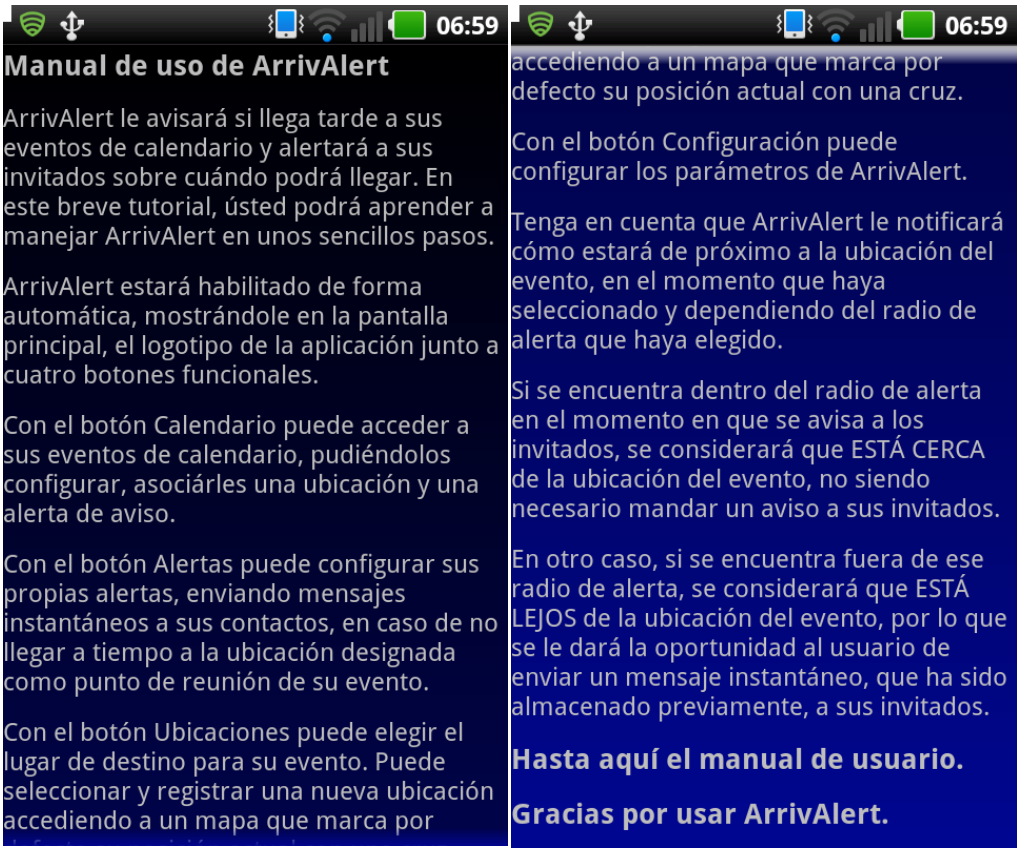

Figura 91 Menú *Manual*

El menú **Acerca De** es de carácter informativo, devuelve una breve información sobre la autoría de *ArrivAlert*.

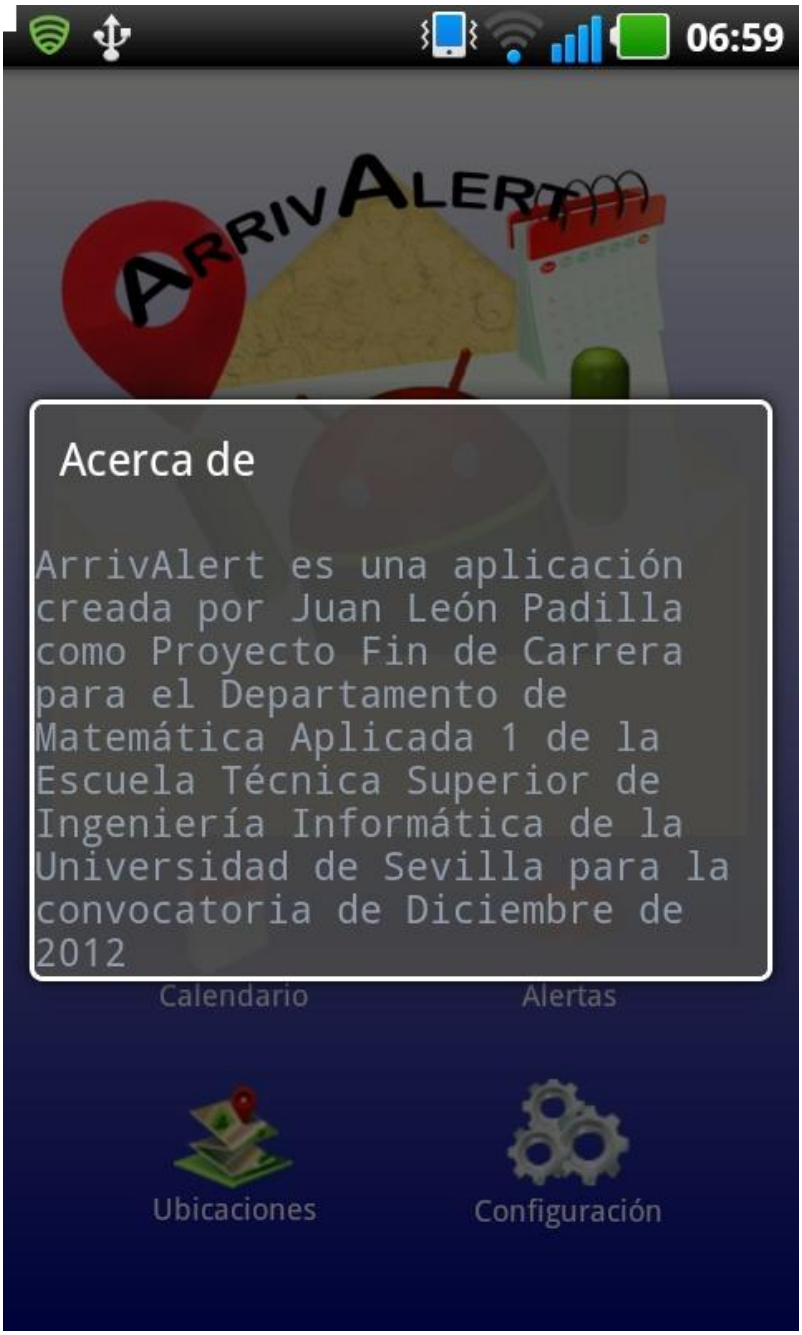

Figura 92 Menú *Acerca de*

El menú **Salir** es el que permite abandonar *ArrivAlert*. Normalmente no se incluye dicha opción en las aplicaciones de *Android* porque se puede utilizar el botón Volver que ofrece el dispositivo, aunque de esta forma se ofrece una funcionalidad completa para el usuario.

7. COMPARACIÓN CON OTRAS ALTERNATIVAS

En esta sección analizaremos algunas de las aplicaciones que existen en el mercado que presentan un mayor grado de similitud con respecto a *ArrivAlert* y se expondrán las ventajas e inconvenientes de ellas.

7.1. TIME TO GO

Es una aplicación de *Android* que nos ofrece una interfaz muy simple en la que simplemente habilitamos esta aplicación y llegado el momento nos avisará de la proximidad del evento así como una estimación del tiempo que nos tomará llegar al punto de reunión de dicho evento.

La principal ventaja es que es capaz de trabajar con cualquiera de los calendarios instalados en el sistema.

El principal inconveniente es que no existe la posibilidad de cambiar las opciones de configuración y carece de la posibilidad de alertar a los invitados al evento.

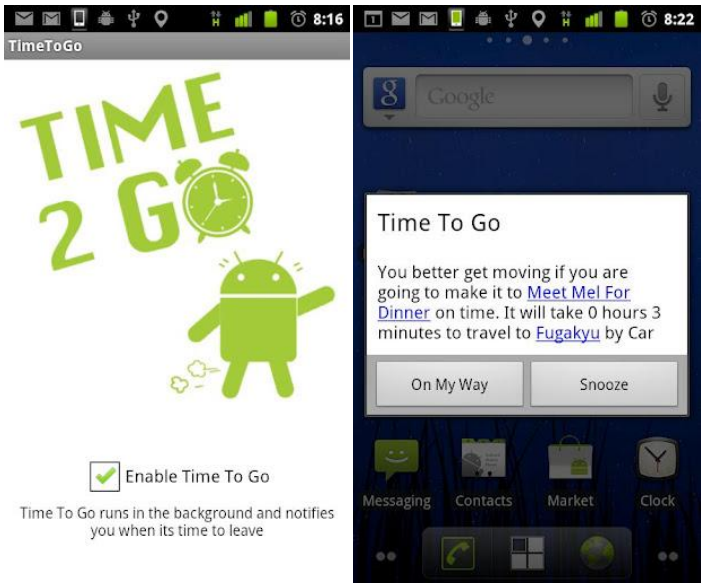

Figura 93 Capturas de pantalla de *Time To Go*

La aplicación tiene un tamaño de 88 KB y se puede descargar de forma gratuita en el siguiente enlace:

https://play.google.com/store/apps/details?id=com.devsmart.timetogo&hl=es

7.2. NEVER BE LATE

Es una aplicación de *Android* que trabaja conjuntamente con cualquier aplicación de calendario.

Es capaz de avisar al usuario de cuánto tiempo le queda para llegar al destino, incluso le ofrece una ruta mediante *Google Maps*, informándole también si encontrará peajes en su ruta.

Se puede seleccionar el modo de viaje aprovechando la API de *Google Maps* (coche, bicicleta o andando), la forma de aviso (pantalla de diálogo o barra de notificación), sonidos de la alerta y el tiempo previo al aviso en cuestión.

Su principal inconveniente es que sólo es informativo para el usuario, carece de la funcionalidad de avisar por email, SMS o cualquier otro tipo de IM a los invitados al evento.

Incluye un manual de instrucciones y la posibilidad de ofertar la aplicación a los contactos de agenda del usuario.

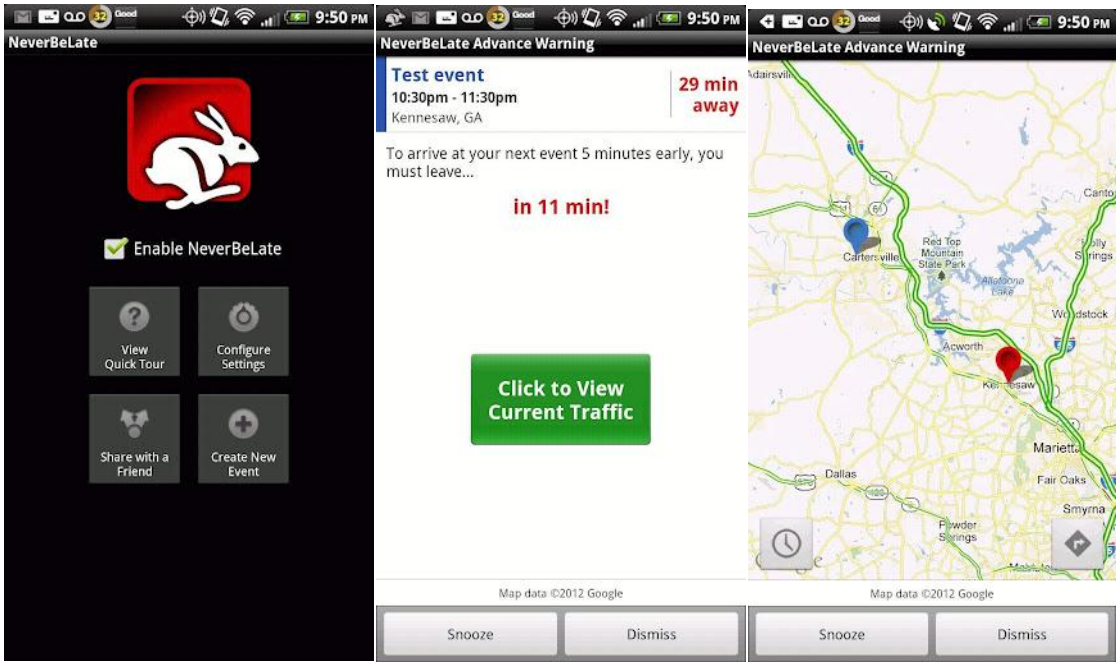

Figura 94 Capturas de pantalla de *NeverbeLate*

Tiene un tamaño de 333 KB y se ofrece de forma gratuita en Play Store:
https://play.google.com/store/apps/details?id=com.madhackerdesigns.neverbelate&hl=es

7.3. LOCATION ALARM

Se trata de una aplicación para dispositivos de Apple que informa mediante una alarma de la llegada del usuario a una determinada localización, haciendo uso del GPS.

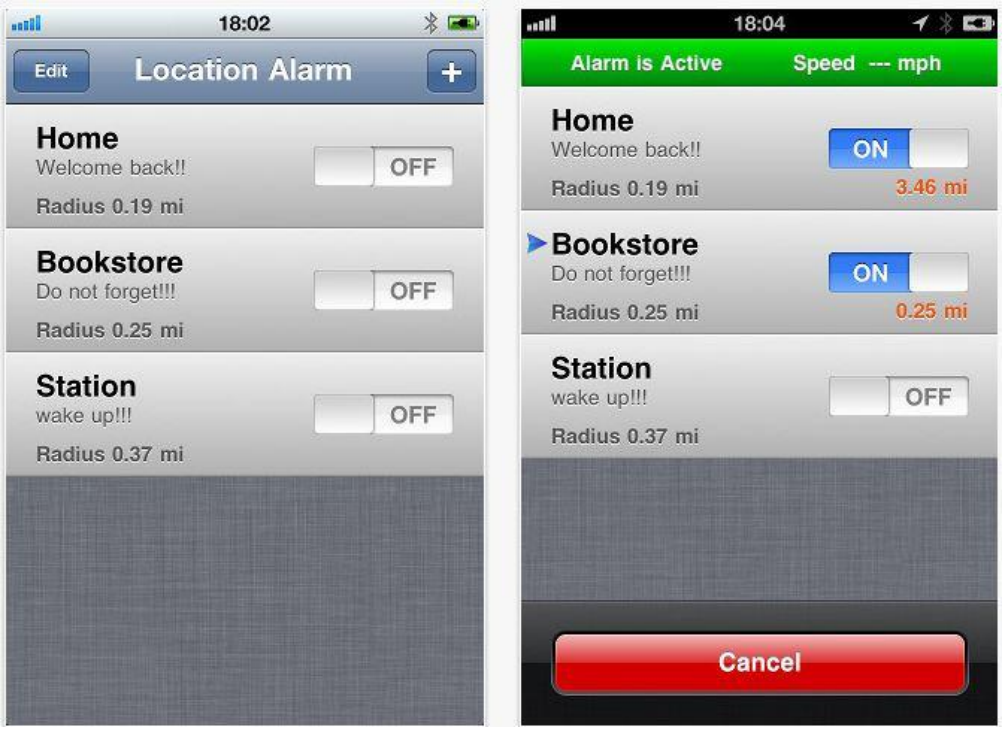

Figura 95 Vistas de *Location Alarm*

El funcionamiento de Location Alarm es similar al de *ArrivAlert*, aunque más limitado, ya que sólo avisa al usuario de eventos programados por él y de su proximidad geográfica al destino, sin advertir a los invitados al evento.

El principal inconveniente de esta aplicación es la imposibilidad de ahorro de batería por parte del uso del GPS en versiones anteriores al sistema operativo iOS4, ya que hasta esa versión no es posible que la aplicación trabaje de fondo, como hace *ArrivAlert*.

Location Alarm está desarrollado por *MAYY Projects* y puede descargarse del siguiente enlace a una coste de 0,79€:

http://itunes.apple.com/es/app/locationalarm-gps-arrival/id413028125?mt=8

7.4. CALENDAR EVENT REMINDER

Es una aplicación para ser usada en Android, que avisa de los eventos de calendario sincronizando para ello con la correspondiente cuenta de *Gmail* del usuario.

El aviso puede personalizarse, con lo que el usuario puede seleccionar su sonido, su vibración, la duración de ambas y su posible repetición.

La información que le llega al usuario del evento es mediante una ventana emergente, recordándole al usuario, el asunto, la fecha y la ubicación.

Figura 96 Aspecto de *CalendarEventReminder*

Se trata de una aplicación muy similar a *ArrivAlert*, ya que alerta de los eventos que se encuentran en la aplicación de calendario, pero se diferencia en que no avisa a los invitados a dichos eventos de la localización del usuario, por lo que no usa GPS ni proveedor de correo electrónico.

Entre los permisos que dispone la aplicación cabe destacar el que impide que el dispositivo entre en suspensión (WAKE_LOCK) y el que inicia la aplicación a la vez que se produce el encendido del dispositivo (BOOT) ya que esto denota que no trabajará de fondo, lo cual es bueno para impedir la pérdida de avisos pero contraproducente para el consumo moderado de batería.

La aplicación está desarrollada por *invalidobject.com* y puede descargarse por el precio de 1,59 € en el siguiente enlace:

https://play.google.com/store/apps/details?id=de.foobarsoft.calendareventreminder

8. CONCLUSIONES Y DESARROLLOS FUTUROS

Arrivalert consigue unificar una serie de utilidades que hacen posible administrar mejor el tiempo, lo cual es de agradecer en los tiempos que corren.

La vida moderna en la que estamos sumergidos hace que nos sea imprescindible el uso de los dispositivos móviles, ya que ahora tenemos la necesidad de estar comunicados e informados constantemente, y cada vez más sin que exista alguien pendiente al otro lado de forma directa.

Así que me siento orgulloso de aportar mi granito de arena, habiendo desarrollado una herramienta que pueda mejorar, aunque sea mínimamente nuestra calidad de vida, ahorrando en preocupaciones y disculpas el hecho de que alguien no pueda llegar puntualmente a una cita.

En el plano profesional, me alegro de haber adquirido conocimientos de una plataforma desconocida para mí antes de la elaboración de este proyecto, como ha sido Android.

Como desarrollos futuros, animo desde aquí, a la elaboración de esta aplicación como multiplataforma, para que sea conjuntamente compatible también con los usuarios de iOS y Windows Mobile.

Además de la interactuación automatizada de mensajería por sistemas de IM, como pueden ser SMS, Whatsapp, Skype, Viber, Line, etc. Siendo también accesibles mediante las redes sociales (Facebook, Twitter, Tuenti, etc).

También a solucionar el problema de los eventos múltiples que nunca llegan a ejecutarse porque se solapan en el tiempo, que comento en la sección de *Pruebas*.

Otra posibilidad de ampliación sería localizar a los usuarios por *Google Maps*, con opción de seleccionar las mejores rutas por GPS para llegar al destino fijado, fijando así una distancia más próxima a la realidad.

APÉNDICE A: INSTALACIÓN DEL ENTORNO DE DESARROLLO

Para desarrollar nuestra aplicación en Android es necesario utilizar un moderno y potente entorno de desarrollo.

Existen varias alternativas a la hora de elegir un entorno de desarrollo, para este proyecto, todas las herramientas que se han usado están basadas en *software libre*, recurriendo al lenguaje Java mediante el Android SDK, aunque también es posible escribir aplicaciones Android en C/C++ con el Android NDK.

A continuación se expondrá los pasos que se han seguido para su instalación.

A.1. INSTALACIÓN DE LA MÁQUINA VIRTUAL JAVA

Resulta imprescindible para poder ejecutar código en Java. La máquina virtual Java (JVM, Java Virtual Machine) se le conoce también como entorno de ejecución Java (JRE, Java Runtime Environment).

En este proyecto se optó por la versión más moderna hasta la fecha de JRE, concretamente la JRE 6.30.

Para instalar la máquina virtual Java se puede hacer desde su página oficial http://java.com/es/download/ y desde allí descargar el archivo *jre-6u30-windows-i586-s.exe*, que fue el que se eligió por la arquitectura y sistema operativo del equipo utilizado.

A.1.1. INSTALACIÓN DE ECLIPSE

Existen varias opciones a la hora de desarrollar Android (NetBeans, InteliJIdea, …) pero Eclipse es el entorno más recomendado, ya que además de ser libre, fue el que usaron los desarrolladores de Google para crear el propio Android.

Eclipse puede descargarse de su página oficial, también de forma gratuita, http://www.eclipse.org/downloads/ eligiendo para este caso la versión *Eclipse IDE for Java Developers*, concretamente

la versión Indigo para Windows de 32 bits, fichero *eclipse-java-indigo-SR1-win32.zip*, quedando instalado con sólo descomprimir los archivos del fichero en una de las carpeta del sistema.

La primera vez que se inicie Eclipse, se pide al usuario una localización para el *workspace*, donde situarán por defecto los proyectos desarrollados.

A.1.2. INSTALACIÓN DE ANDROID SDK DE GOOGLE

Android SKD de Google es un paquete de desarrollo con el que se puede elaborar aplicaciones y ejecutarlas en un emulador del sistema Android con una determinada versión.
Para instalar el paquete se puede descargar de la página web http://developer.android.com/sdk

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r17-windows.zip	37417953 bytes	3af1baeb39707e54df068e939aea5a79
	installer_r17-windows.exe (Recommended)	37410775 bytes	5afaf6511ebaa52bd6d1dba4afc61e41
Mac OS X (intel)	android-sdk_r17-macosx.zip	33867836 bytes	52639aae036b7c2e47cf291696b23236
Linux (i386)	android-sdk_r17-linux.tgz	29706368 bytes	14e99dfa8eb1a8fadd2f3557322245c4

Figura 97 Detalles del Android SDK en varias plataformas

El paquete se descomprime y se ejecuta el archivo *SDK Setup* seleccionando *Settings* en la parte izquierda de la ventana *Android SDK and AVD Manager*, marcando el *checkbox* etiquetado como *Force https://... Sources to be fetched using http://...*

A continuación se pueden elegir los paquetes a instalar eligiendo la versión correspondiente con los que se quieren desarrollar los paquetes en Android, seleccionando la opción *Accept All* y pulsando el botón *Install Accepted*.

El proceso de instalación puede tardar unos minutos, dependiendo del volumen de paquetes a instalar.

Posteriormente se puede gestionar la instalación y desinstalación de paquetes desde Eclipse, mediante *Android SDK*

Manager. Se puede seleccionar en el menú *Windows* > *Android SDK Manager* o directamente en la barra de herramientas de Eclipse.

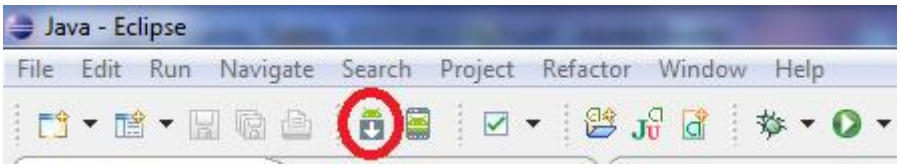

Figura 98 Icono de SDK Manager

A.1.3. CREACIÓN DEL DISPOSITIVO VIRTUAL ANDROID (AVD)

Para ejecutar las aplicaciones en Android sin necesidad de un dispositivo físico, hay que tener un dispositivo simulado que puede ser creado dentro de Eclipse, mediante *Android Virtual Device Manager*.

Se puede seleccionar en el menú *Windows* > *Android Virtual Device Manager* o directamente en la barra de herramientas de Eclipse.

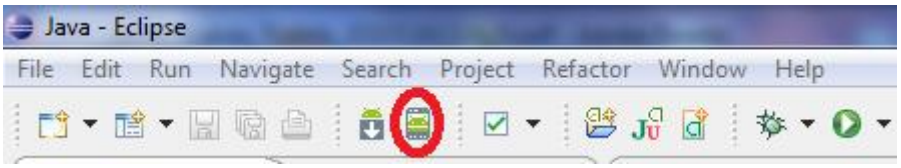

Figura 99 Icono de AVD Manager

Pulsando el botón *New* aparecerá la siguiente ventana de creación:

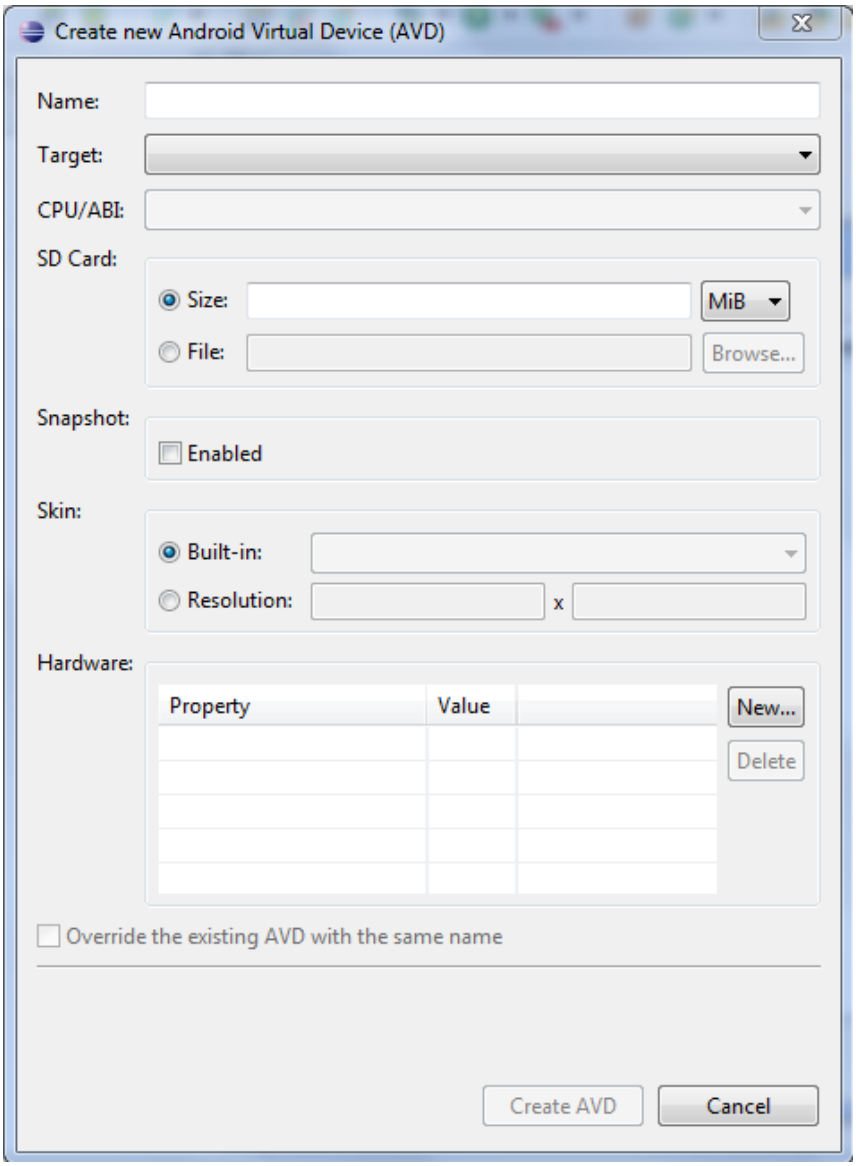

Figura 100 Configuración para la nueva creación de un AVD

Se pueden introducir las siguientes características:

- **Name**: Nombre para el dispositivo virtual.
- **Target**: Indica la version SDK del dispositivo virtual.
- **SD Card**: Crea el tamaño de la tarjeta SD usada en el emulador.
- **Skin**: Selecciona las características y apariencia del dispositivo.
- **Hardware**: Agrega características específicas para el dispositivo (caché, particiones, acelerómetro, *GPS*, audio, batería, cámara, etc).

Después de introducir la configuración se pulsa el botón *Create AVD*.

A continuación podemos ponerlo en funcionamiento seleccionando de la lista un dispositivo Android.

En la siguiente figura se puede elegir entre un emulador compatible con la API 10 (*Android 2.3.3*) y un dispositivo físico móvil (concetamente un *LG Optimus Black* bajo *Android 2.3.4*) siendo también posible lanzar otros emuladores con la API 15 (*Android 4.0.3*)

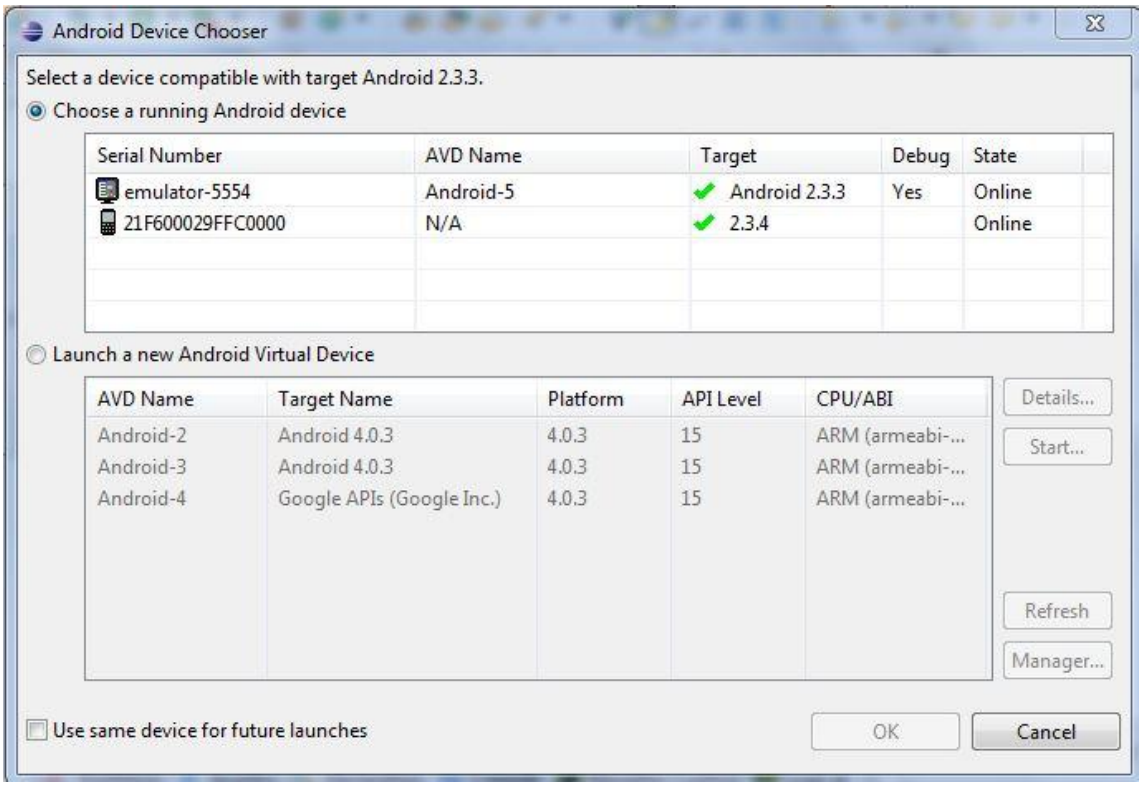

Figura 101 Seleccionador de dispositivos Android

Pulsando el botón Launch arrancara el AVD, es posible que tarde algunos minutos

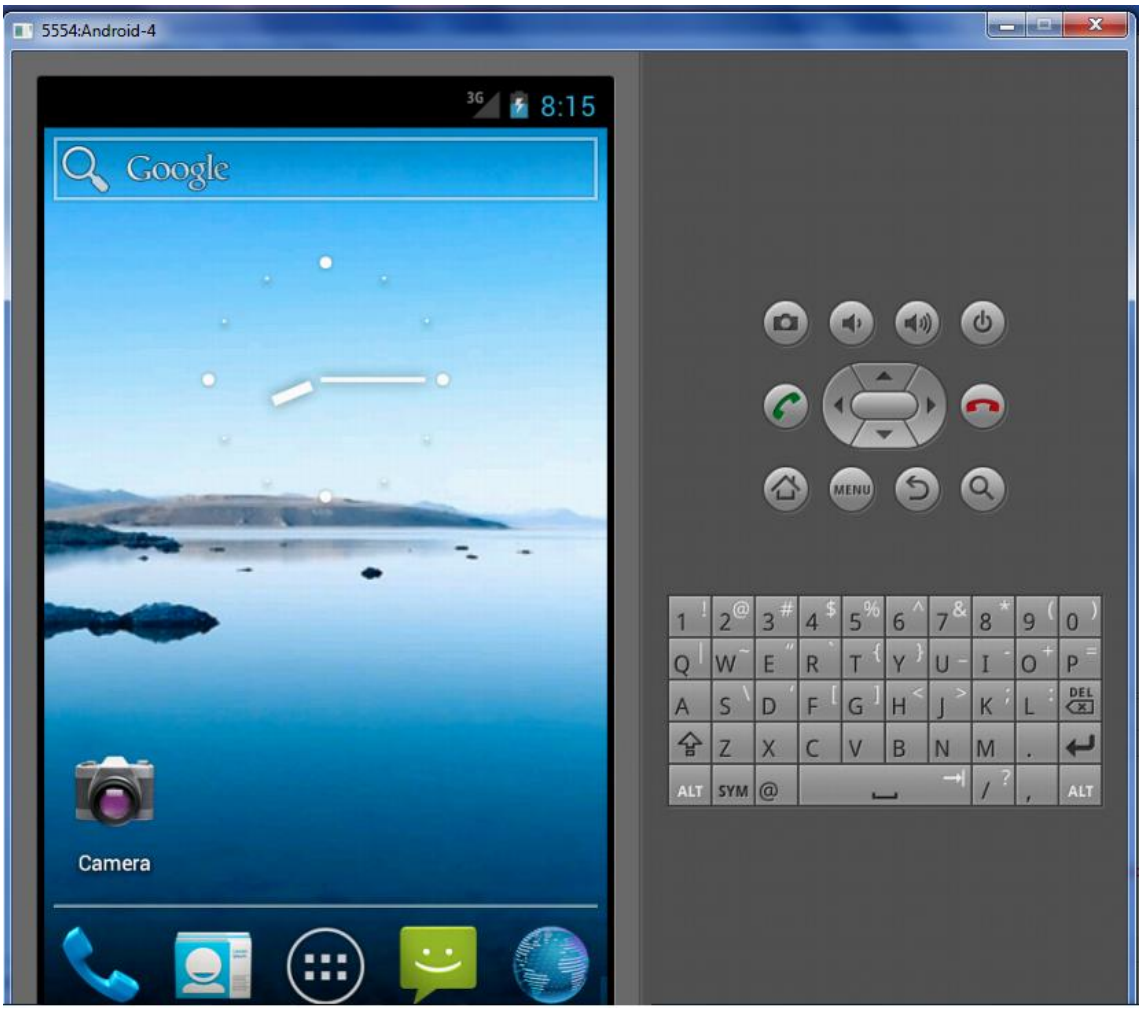

Figura 102 Emulador de Android ya arrancado

A.1.4. INSTALACIÓN DEL COMPLEMENTO PARA ECLIPSE (ADT)

Para facilitar el desarrollo de aplicaciones en Android, Google ha dispuesto un complemento para Eclipse llamado *Android Development Toolkit* (ADT).

Dicho plug-in puede instalarse de forma sencilla en Eclipse desde el menú *Help > Install New Software*, haciendo clic en el enlace *Available Software Sites* y añadiendo la ubicación (botón *Add…*) del sitio web de actualización de *Android Development Toolkit*, http://dl-ssl.google.com/android/eclipse.

El cuadro de diálogo debe aparecer de la forma siguiente:

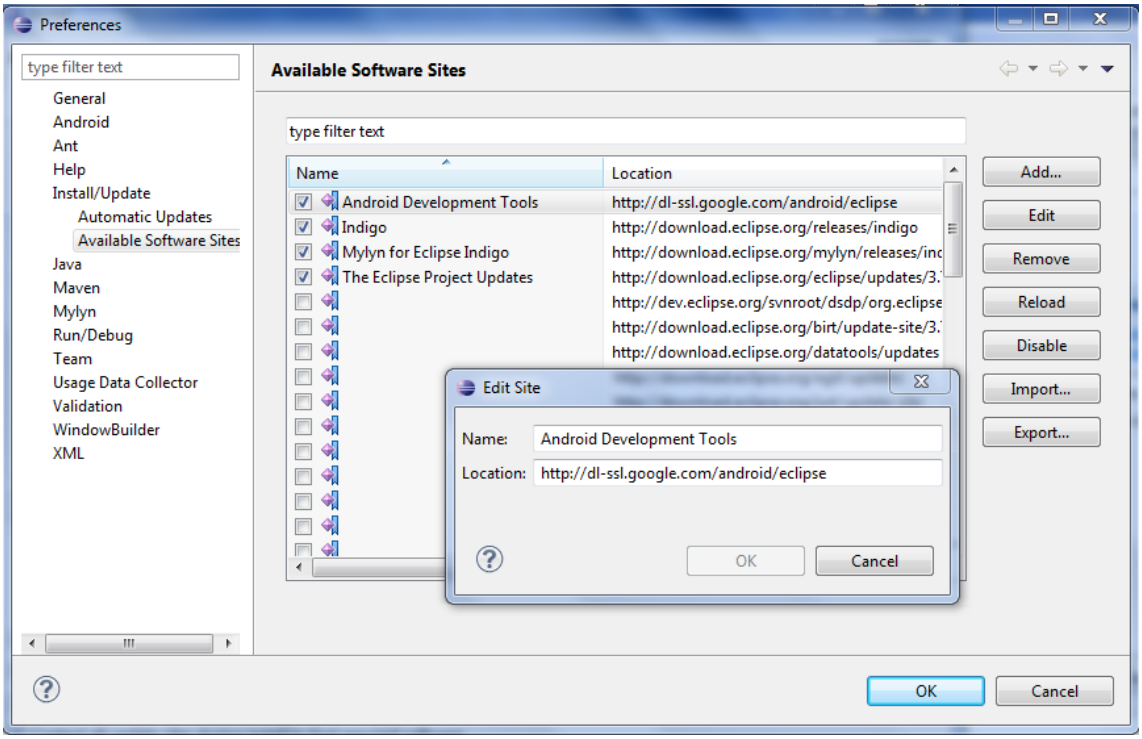

Figura 103 Detalle de la instalación de ADT

APÉNDICE B: OBTENCIÓN DE LA CLAVE PARA EL USO DE GOOGLE MAPS

Toda aplicación de Android que haga uso de mapas de Google necesita una clave que le permita hacer un empleo de los mismos, durante el desarrollo de la aplicación (para su distribución se necesitaría otra clave distinta).

Para ello se deben aceptar los términos y condiciones que solicita Google para el uso de este servicio, además de inscribir nuestra huella digital MD5 de certificado, accediendo al sitio web https://developers.google.com/android/maps-api-signup

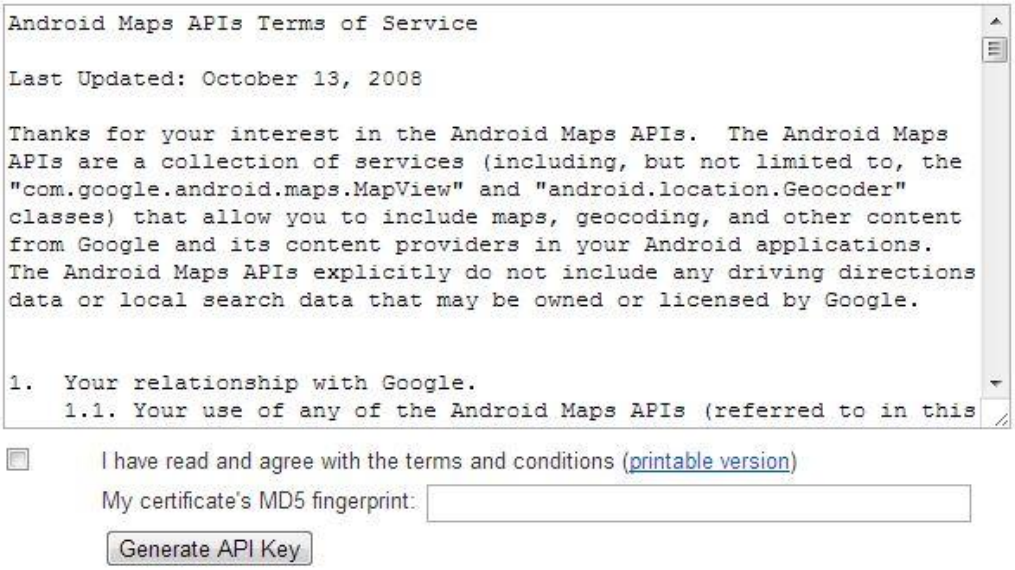

Figura 104 Solicitud de clave para uso de Google Maps

Para saber cuál es nuestra huella digital, primero se accede al menú de Eclipse, **Window > Preferences > Android > Build**.

Entonces en la opción *Default debug keystore* aparecerá la ruta de acceso al archivo donde se almacena el certificado digital que se usa en la depuración.

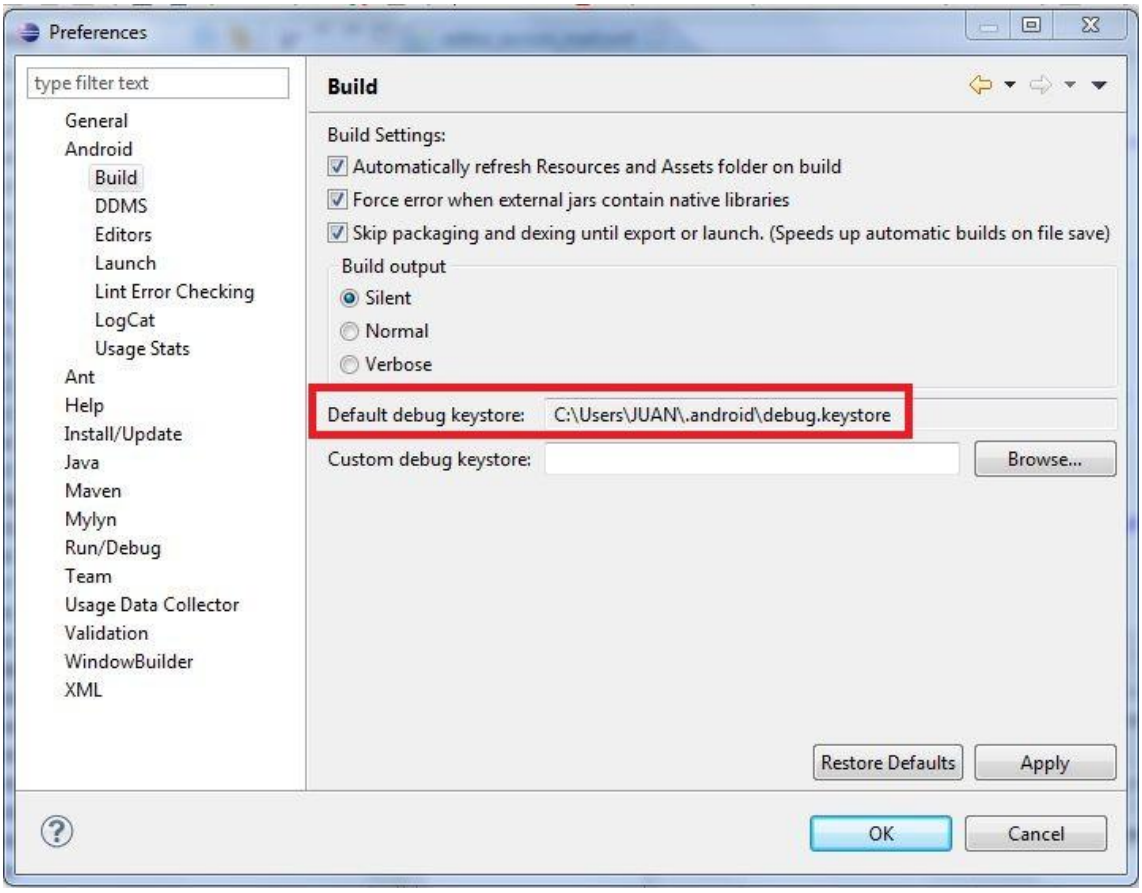

Figura 105 Detalle de la ruta de acceso al certificado digital

En una ventana de *símbolo del sistema* se ejecuta el siguiente comando, que devuelve nuestra huella digital de certificado:

keytool.exe –list –alias androiddebugkey –keystore "ruta de acceso" –storepass android –keypass android

Fíjese que, por defecto, la clave predeterminada para el almacén de claves y su contraseña es **android**.

Figura 106 Ejecución de keytool

Una vez que conocemos cuál es nuestra huella digital de certificado podemos completar la solicitud de clave que nos requería Google para poder obtener una.

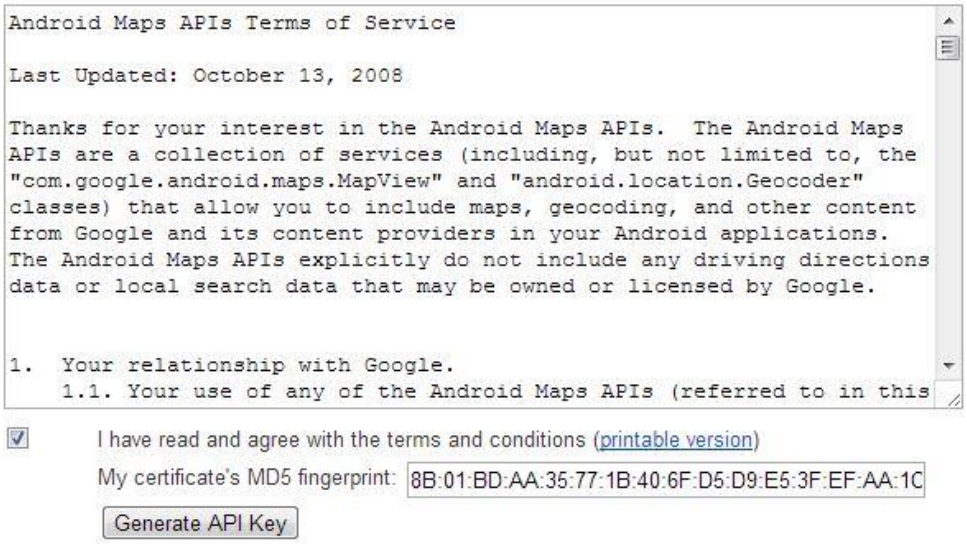

Figura 107 Solicitud de clave para uso de Google Maps rellena

Con tan sólo pulsar en el botón *Generate API Key*, ya dispondremos de nuestra propia clave para el uso de Google Maps.

Figura 108 Detalle de la obtención de la clave de Google Maps

BIBLIOGRAFÍA

- Tomás Gironés, Jesús. El gran libro de Android. Editorial Marcombo (2011).
- Tomás Gironés, Jesús. El gran libro de Android. 2ª Edición. Editorial Marcombo (2012).
- Meier, Reto. Professional Android 2. Application Development. Editorial Wiley Publishing (2010).
- Ableson, W. Frank y otros. Android: Guía para desarrolladores. 2ª Edición (2011).
- Burnette, Ed. Android. Ediciones Anaya Multimedia (2011).
- Lee, Wei-Meng. Android 4. Desarrollos de aplicaciones. Ediciones Anaya Multimedia (2012).
- McCracken, Scott. Android: Curso de desarrollo de aplicaciones. Editorial INFORBOOK'S S.L. (2012).
- Ribas Lequerica, Joan. Desarrollo de Aplicaciones para Android. Editorial Anaya Multimedia (2011).
- Amaro Soriano, José Enrique. El gran libro de programación avanzada con Android. Editorial Marcombo (2012).
- Amaro Soriano, José Enrique. Android: programación de dispositivos móviles a través de ejemplos. Editorial Marcombo (2011).
- Montero Miguel, Roberto. Desarrollo de aplicaciones para Android. Editorial Ra-Ma (2012).

- Web oficial de Android:
  http://developer.android.com/index.html

- Página web de Stackflow:
  http://stackoverflow.com/questions/tagged/android

- Página oficial de LG Optimus Black:
  http://www.lg.com/es/telefoniamovil/moviles-lg/LG-touch-P970-Optimus-Black.jsp

- Tutorial de Android:
  http://comunidadcodificada.com/portal/index.php/2012/01/android-anadiendopreferencias-a-nuestras-apps/

- Tutorial de Android:
  http://osl.ulpgc.es/files/docs/cursos/android/D2-1.pdf

- Web de iconos para aplicaciones:
  http://www.iconspedia.com/