Universidad Politécnica de Cartagena

Proyecto Fin de Carrera

Hécate: Aplicación Android para notificación telefónica automática en caso de accidente de motocicleta.

Autor: José Juan Pedreño Manresa
Director: Juan Ángel Pastor Franco

Septiembre 2013

Autor	José Juan Pedreño Manresa
E-mail del autor	jjpedreno@gmail.com
Director	Juan Ángel Pastor Franco
E-mail del director	juanangel.pastor@upct.es

Título del pfc	Diseño e implementación de una aplicación Android para notificación telefónica automática en caso de accidente de motocicleta.

Resumen:

Proyecto consistente en el diseño e implementación de una aplicación para dispositivos con sistema operativo Android.

La aplicación una vez iniciada hace uso de los sensores (acelerómetro) para analizar las aceleraciones sufridas por el usuario durante travesías con motocicleta. Se detectará si se produce un accidente activándose una alarma sonora y visual que confirme que el usuario se encuentra consciente y a salvo. En caso contrario contacta vía llamada telefónica o mensaje de texto con un contacto de emergencia indicando las coordenadas geográficas del accidente (vía GPS).

Titulación	Ingeniería Técnica de Telecomunicación
Especialidad	Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	Octubre 2013

Índice

Introducción.

Planteamiento inicial.

Debido al carácter individual del transporte mediante motocicleta, se plantea un gran problema cuando surge un accidente. Si el conductor queda incapacitado y no existen testigos presenciales, no existe forma de avisar al servicio de emergencias.

En otro tipo de vehículos, como automóviles, existen sistemas integrados que en caso de accidente contactan automáticamente con una centralita de forma automática; pero estos son costosos y voluminosos[1].

Haciendo uso de las tecnologías integradas en dispositivos smartphones modernos, se pretende crear una aplicación compatible con dispositivos con sistema operativo Android para solventar este problema. Aprovechando el bajo coste y amplia disponibilidad de estos dispositivos, dicha aplicación detectará un posible accidente y avisará automáticamente por medios telefónicos.

Partiendo de un estudio teórico y posteriores pruebas empíricas, se analizará las aceleraciones y desaceleraciones sufridas por el usuario durante travesías en motocicleta. Se procederá entonces a programar una aplicación Android que haga uso del acelerómetro lineal (para medir dichas aceleraciones) y del GPS (para tener la posición del usuario actualizada) para detectar cuando se produce un accidente. El accidente se detectará detectando aceleraciones superiores a los umbrales calculados y comprobados. Una vez producido el accidente la aplicación hará saltar una alarma y se solicitará una acción explícita para desactivarla.

Objetivos.

Los objetivos planteados en el proyecto son los siguientes:

- Estudiar las aceleraciones típicas experimentadas durante una travesía en motocicleta, así como calcular aquellas sufridas en accidentes.
- Creación de una aplicación auxiliar para recogida de datos empíricos y su posterior contraste con el estudio teórico.
- Creación de la aplicación Hécate para notificación telefónica automática del accidente.
- Realización de pruebas de funcionamiento y depuración de posibles fallos.
- Redacción de la memoria.

Plan de trabajo.

Para la consecución del proyecto, este se ha dividido en una serie de fases bien definidas:

1. Estudio y aprendizaje de desarrollo de aplicaciones para dispositivos Android. Esto incluye la lectura y estudio de la sección 'Training' de la Guía para Desarrolladores de Android[2], realizando los tutoriales propuestos, así como de diversos libros versados en este tema. Se hará especial hincapié en las secciones de 'Sensores', y 'Localización'. También se

procederá a la instalación, configuración y aprendizaje del software necesario para realizar los tutoriales antes mencionados.

2. Desarrollo de la aplicación Hermes. Este punto conlleva un doble objetivo; primero la puesta en práctica de lo aprendido en el apartado anterior, desarrollando una aplicación completa y plenamente funcional. Segundo, dicha aplicación tendrá como función la toma de datos empíricos, lo cual lleva al siguiente punto.
3. Recogida de datos. Haciendo uso de la aplicación antes mencionada se procederá a recopilar datos relativos a las fuerzas sufridas por el conductor durante trayectos típicos en motocicleta. Para ello se intentarán realizar dichas pruebas con vehículos de diferentes cilindradas y potencia, así como en trayectos de ciudad e interurbanos. Dichas pruebas consistirán en la toma de muestras a intervalos regulares de las aceleraciones sufridas en cada uno de los tres ejes del teléfono (siguiendo un sistema de coordenadas cartesiano).
4. Análisis de datos. Una vez obtenidos los datos se procederá al análisis de los mismos. Esto incluye su visualización gráfica, su interpretación, manipulación y la aplicación de filtros adecuados.
5. Desarrollo de la aplicación Hécate. Esta es la parte central y más importante del proyecto. Se desarrollará la aplicación de aviso en caso de accidente vial haciendo uso de los datos previamente analizados. Ésta será plenamente funcional, con versión final estable y adecuada para su uso en situaciones reales.
6. Pruebas. Durante el desarrollo de la aplicación se irán realizando una serie de pruebas con la finalidad de comprobar su correcto funcionamiento así como de depurar posibles fallos o comportamientos no deseados.
7. Redacción de la memoria. En último lugar se procederá a redactar la memoria detallando todo el proceso de desarrollo del proyecto para su posterior defensa ante un tribunal.

Recursos y entornos de desarrollo.

Como entorno de desarrollo se ha utilizado el *Software Development Kit* de Android (incluyendo APIs, herramientas, emuladores, entorno de monitorización, etc.) integradas mediante el IDE Eclipse.

Se ha considerado el uso de Eclipse por su robustez, estabilidad, diversas funcionalidades, y por estar ya integrado con el kit de desarrollo de Android mediante el *plugin* ADT. Además es un entorno de desarrollo ya conocido debido a su uso a lo largo de la carrera.

Como plataforma de prueba, se ha optado por el Smartphone Samsung Galaxy SIII (GT-I9300). Aunque Android SDK ofrece un emulador de dispositivos tanto la aplicación Hermes como Hécate hacen uso de funcionalidades que no son emulables por software. También se han realizado pruebas de uso en dispositivos LG Google Nexus 4 y Samsung Galaxy Note II.

Estado de la técnica.

Sistema operativo Android.

Android es un sistema operativo inicialmente diseñado para controlar dispositivos móviles como pueden ser teléfonos (*smartphones*), tablets, o los recientes *smartwatches*[3]*;* aunque a día de hoy existen versiones utilizadas en otros dispositivos multimedia, como por ejemplo televisores (fenómeno *SmartTV*[4]).

Figura 1. ***Logotipo de Android, coloquialmente conocido como 'Andy'.***

Historia.

Desarrollada su primera versión por la compañía Android Inc. en 2003, el objetivo era crear "(...) dispositivos móviles más inteligentes, que presten atención a las preferencias y ubicación del usuario". Aunque el trabajo de la compañía permaneció en relativo secreto, ésta fue adquirida en 2005 por Google, que ya por entonces mostraba especial interés en el mercado de los dispositivos móviles inteligentes. Android Inc. pasó a convertirse en filial de Google, respetándose la mayoría de la plantilla original (programadores y directivos).

Se empieza en ese momento a buscar acuerdos comerciales con operadores telefónicos y fabricantes de dispositivos móviles, presentándose como una plataforma abierta, actualizable y personalizable. El objetivo es la inclusión de Android como sistema operativo por defecto en dispositivos de nueva manufactura. En noviembre de 2007 Android es presentado al público por la *Open Handset Alliance*[5], consorcio de reciente creación que incluye más de 70 empresas del sector de las telecomunicaciones (*Samsung, HTC, Qualcomm, Texas Instruments, Intel, Motorola*, etc.[6]) lideradas por Google. El objetivo del consorcio es el desarrollo de estándares abiertos para dispositivos móviles. Siguiendo esta filosofía Google libera la mayor parte del código fuente de Android[7] bajo licencia Apache[8], la cual es libre, gratuita y de código abierto.

Desde su primera versión estable (Android 1.0[9]) presentada en septiembre de 2008, la evolución de este sistema operativo ha sido tanto vertiginosa como espectacular, incorporándose funcionalidades como pantalla multi-táctil, reconocimiento facial, búsqueda y control del dispositivo mediante comandos de voz, soporte nativo para VoIP (*Voice Over Internet Protocol*), compatibilidad con HTML5, soporte NFC (*Near Filed Communications*) y *Bluetooth Low Energy* y un largo etc.

En 2010 Google adquiere *Motorola Mobility*[10] (escisión de la compañía original) marcando su inclusión en la fabricación hardware de dispositivos móviles.

En la actualidad, los últimos datos indican que Android acapara casi un 80% de cuota global de mercado[11], seguido por iOS con un 13%. Otros sistemas operativos como *Windows Phone*, *BlackBerry OS* o *Symbian* apenas alcanzan un 7% en conjunto.

Evolución de la tecnología móvil.

Smartphone.

El origen de Android está intrínsecamente ligado al concepto de 'smartphone'. El término aparece por primera vez en 1997, cuando la empresa Ericsson define a su dispositivo *GS88 Penelope* como 'teléfono inteligente. Al no existir una definición oficial, el término smartphone puede ser difícil de explicar. Una definición aproximada es aquel dispositivo telefónico móvil que ofrece una avanzada capacidad de cómputo (similar a la de un PC) y una conectividad superior a la del resto de teléfonos móviles convencionales.

Aquello en lo que sí coincide todo el mundo a la hora de definir un smartphone es que éste debe ejecutar un sistema operativo que permita la instalación y uso de aplicaciones desarrolladas por terceros. En el caso de Android, dichas aplicaciones puedan instalarse a través de su tienda oficial (Google Play) o directamente desde el almacenamiento interno del teléfono (las instrucciones específicas para ello se detallan en el apartado 'Instalación y manual de uso').

Sensores.

Como ya se ha mencionado anteriormente, los dispositivos modernos poseen multitud de sensores como pueden ser GPS, acelerómetro, giroscopio, magnetómetro, fotómetro, sensor de proximidad, etc. Se podría discutir acerca de ellos de manera extensa, sin embargo este proyecto se centra en dos concretamente, debido a la especial importancia que juegan en el desarrollo de las aplicaciones Hécate y Hermes.

Acelerómetro.

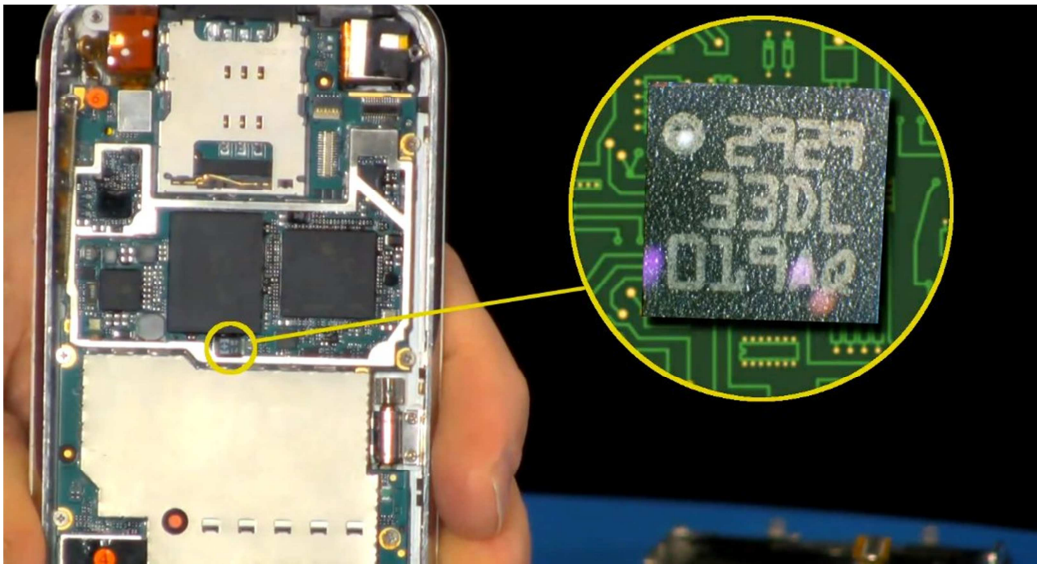

Figura 2. Ejemplo de *acelerómetro miniaturizado en un smartphone.*

Un acelerómetro, como su nombre indica, es un dispositivo usado para medir aceleraciones. Las aceleraciones medidas no solo corresponden a aquellas ocasionadas por cambios en la velocidad, sino también las provocadas por la gravedad. Un acelerómetro en reposo sobre una superficie plana experimenta una aceleración perpendicular a dicha superficie de 9.81m/s^2.

Los teléfonos móviles modernos incorporan acelerómetros en su hardware capaz de medir las aceleraciones sufridas por el dispositivo en cada uno de los 3 ejes ortogonales del teléfono (sistema cartesiano). Entre sus usos se encuentra el detectar cambios en la inclinación del dispositivo (para poder rotar la pantalla en consecuencia), el servir de interfaz de entrada para videojuegos mediante movimientos del usuario, estabilización de imágenes tomadas con la cámara, etc.

Dos ejemplos de modelos de acelerómetros más comunes específicamente diseñados para smartphones son *Bosch Sensortech SMB380*[12] e *InvenSense MPU-6050*[13], siendo este último el modelo en algunos de los teléfonos usados en la fase de pruebas a lo largo del proyecto (Galaxy S3 y Nexus 4).

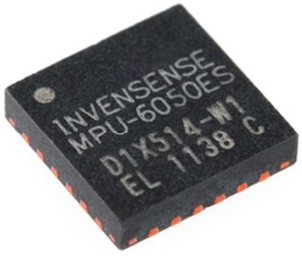

Figura 3. *Acelerómetro+Giroscopio InvenSenses MPU-6050*

Como puede observarse en los *datasheet* referenciados, estos acelerómetros de reducido tamaño (menos de medio centímetro cuadrado de superficie) presentan un reducido consumo (2.5V / 3.9mA), y permiten obtener lecturas de 'aceleraciones lineales' (sin incluir el factor gravedad) debido a que en el chip también está integrado un giroscopio.

El rango máximo de lectura en el caso concreto del modelo MPU-6050 es de +/-16G (≈ 157m/s^2), más que de sobra para el propósito de este proyecto, tal y como se verá más adelante. Sin embargo, este rango máximo es configurable pudiéndose escoger el límite en +/-2G/4G/8G. Esto puede suponer un problema si el límite escogido es inferior al requerido por la aplicación Hécate, ya que dicha configuración se establece de fábrica, a nivel de ensamblado, y no puede modificarse programáticamente.

En el caso concreto de los teléfonos probados, el smartphone Samsung Galaxy S3 permite lecturas de hasta +/-8G y el LG Nexus 4 hasta +/-4G.

GPS.

GPS (*Global Positioning System*, sistema de posicionamiento global[14]) es un sistema de navegación espacial basado en geo-posicionamiento por satélite. Este sistema, propiedad de los EE.UU. proporciona al usuario información precisa sobre su posición y hora actual en cualquier parte de la Tierra siempre que haya línea de visión directa con los satélites *GPS* en órbita. A pesar de ser un sistema de origen militar, se permite el uso civil casi sin restricciones en todo el mundo.

Aunque existen otros sistemas de radio-posicionamiento que preceden al *GPS*, se puede considerar su nacimiento en 1973, cuando el *Pentágono* decide la creación de DNSS (*Defense Navigation Satellite System*) como sistema militar de geo-localización. Más tarde ese año se renombró *Navstar-GPS* y posteriormente se acortó a simplemente *GPS*. En 1983 se decide permitir su uso al público, y más tarde, en el año 200 se eliminan las restricciones de 'precisión' (inicialmente 100 metros, pasando a menos de 20 metros de precisión en la actualidad).

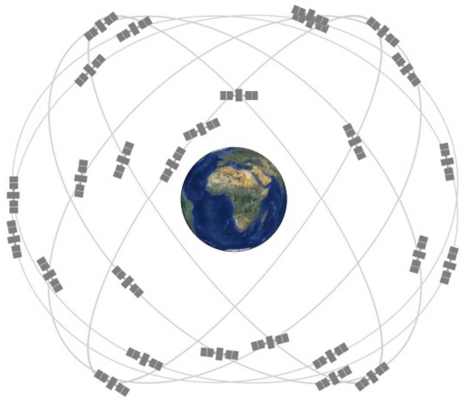

Figura 4. ***A día de hoy hay 31 satélites GPS en órbita constante alrededor de la Tierra.***

En los últimos años los dispositivos receptores de *GPS* han sufrido una asombrosa evolución; pasando de ser voluminosos, caros y lentos (tardando incluso varios minutos en proporcionar una lectura precisa de la posición) a ser diminutas piezas de hardware instaladas en prácticamente todos los dispositivos smartphone de nueva manufactura (incluso aquellos de gama baja) y capaces de obtener la posición del usuario en apenas unos segundos.

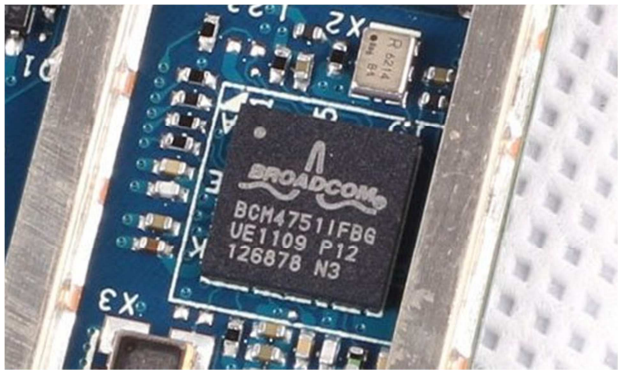

Figura 5. ***Receptor GNSS.***

Dos de los modelos presentes en los teléfonos usados durante el proyecto son *Broadcom BCM47511*[15] (instalado en *Samsung Galaxy S3*) y *Avago ALM-3012*[16] (presente en *Google Nexus 4*). Estos chips son compatibles no solo con *GPS* sino también con GLONASS (sistema equivalente desarrollado por el gobierno ruso) y otros sistemas satelitales GNSS (*Global Navigation Satellite System*[17]). Otras características que presentan estos dispositivos son un bajo consumo y el ser completamente transparentes al usuario (no requieren ningún tipo de configuración).

Requisitos de la aplicación.

Plataforma.

Los requisitos para una correcta instalación y uso de las aplicaciones Hécate y Hermes serán:

- Las aplicaciones desarrolladas debe ser compatibles con el mayor número posible de dispositivos. Por ello se ha seleccionado como plataforma un smartphone (o tablet) con sistema operativo Android 2.3 (*Gingerbread*[18]) o superior (API 9[19]) siguiéndose las recomendaciones de la guía para desarrolladores de Android[20] para que las aplicaciones sean compatibles con al menos el 90% de los dispositivos Android del mercado[21] (ver figura inferior).

Version	Codename	API	Distribution
2.2	Froyo	8	2.4%
2.3.3 - 2.3.7	Gingerbread	10	30.7%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	21.7%
4.1.x	Jelly Bean	16	36.6%
4.2.x		17	8.5%

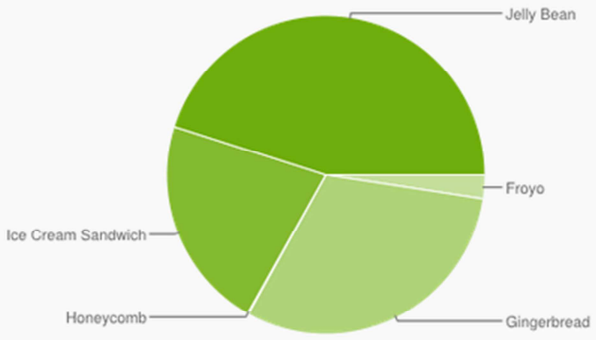

Figura 6. ***Fragmentación de las distintas versiones de Android a fecha de 4 de septiembre de 2013***

- Espacio libre disponible de al menos 0.94MB para la aplicación Hécate (ésta puede ser transferida al almacenamiento externo o SD). La aplicación Hermes necesita 700KB de espacio (esto se ha determinado a posteriori una vez desarrolladas las aplicaciones).
- Acelerómetro lineal. Este sensor debe estar implementado en el dispositivo. Está conformado por la 'fusión de sensores' normalmente acelerómetro y giroscopio. Además, el acelerómetro deberá permitir lecturas en un rango de al menos +/- 4G. Se profundizará más en el apartado *Sensores y acelerómetros*.

  Otro requisito importante es que dicho sensor siga funcionando con la pantalla apagada. Se ha detectado que algunos dispositivos dejan de enviar las medidas tomadas por los sensores cuando la pantalla se apaga. Esta peculiaridad de funcionamiento depende exclusivamente del fabricante de hardware y no suele ir documentada en las especificaciones del teléfono. La única forma de comprobar si un teléfono incurre en dicho comportamiento es haciendo una comprobación empírica con una aplicación que haga uso del acelerómetro (por ejemplo Hécate y Hermes). No obstante existen páginas web dedicadas a hacer listados clasificando dichos dispositivos[22].

- *GPS* integrado en el dispositivo. Para la correcta detección de la posición del usuario durante el uso de la aplicación Hécate, el dispositivo debe contar con un *GPS* plenamente funcional.
- Línea telefónica de voz (sólo para Hécate). Para un correcto funcionamiento de la aplicación, y que ésta pueda avisar mediante voz o SMS, es necesario que el dispositivo tenga acceso a una línea de voz plenamente funcional.

En un primer momento se pensó en incluir como requisito que el dispositivo disponga de la programa de síntesis de voz (*TTS*) con su correspondiente paquete para lengua Española; pero debido a que esta funcionalidad viene instalada por defecto en todos los teléfonos Android de nueva manufactura, se ha obviado este punto por ser redundante.

Por último, debido a la heterogeneidad de los dispositivos compatibles con Android, el correcto funcionamiento de la aplicación dependerá también del hardware específico y la implementación individual de cada fabricante. Aunque se ha procurado que el desarrollo de las aplicaciones Hécate y Hermes sean compatibles con la mayor cantidad de dispositivos existentes, no se puede garantizar en última instancia un correcto funcionamiento en el cien por cien de los dispositivos a disposición del público. Para garantizar un correcto funcionamiento de la aplicación Hécate, el fabricante del teléfono debe haber configurado el acelerómetro para permitir un rango de valores de al menos +/-4G. De lo contrario no puede asegurarse que en caso de accidente la aplicación funcione como es debido.

Casos de uso de Hécate.

Puede observarse en la figura inferior los casos de uso de la aplicación y posteriormente su descripción detallada.

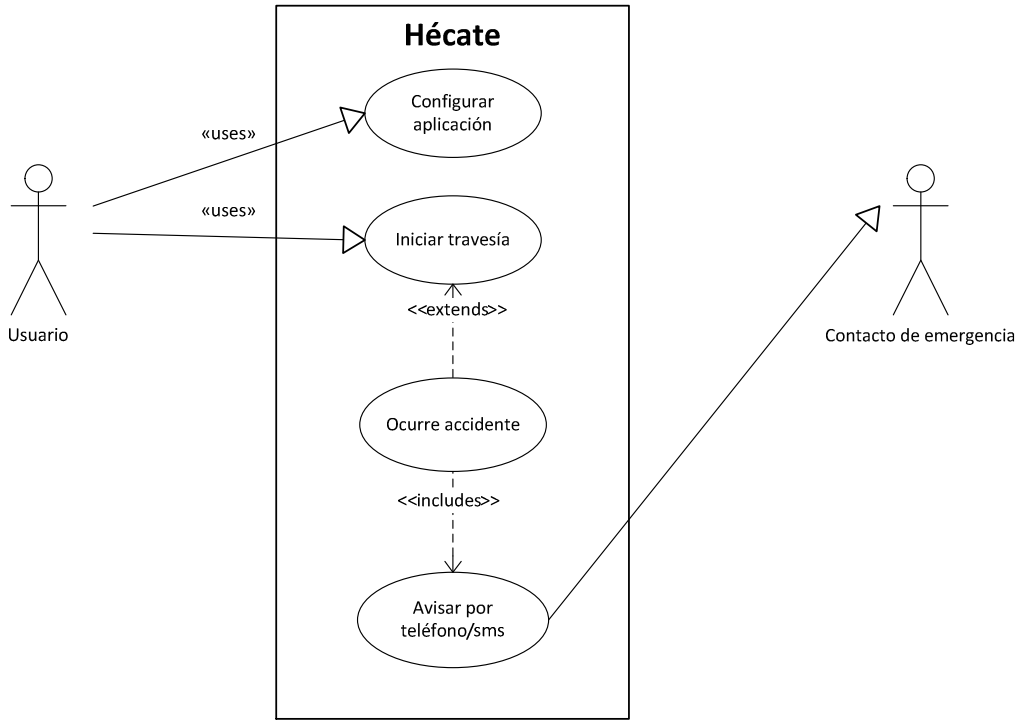

Figura 7. ***Diagrama UML de casos de uso de la aplicación Hécate.***

Caso de uso 1: Configuración inicial.

Descripción
Muestra los pasos seguidos por el usuario durante la configuración inicial o posterior de la aplicación.
Flujo principal
1. El usuario arranca la aplicación. 1.1. El sistema muestra la actividad principal, la cual contiene instrucciones básicas de funcionamiento y el botón 'Start'. 2. El usuario pulsa el botón de opciones (icono con forma de llave inglesa). 2.1. El sistema muestra la pantalla de preferencias. 3. El usuario pulsa la opción 'Teléfono'. 3.1. El usuario introduce un número de teléfono en el campo de texto. 4. El usuario marca/desmarca la opción 'SMS'. 4.1. El sistema informará mediante SMS en caso de accidente. 5. El usuario marca/desmarca la opción Debug. 5.1. El sistema se configura en modo de pruebas.

Caso de uso 2: Travesía no accidentada.

Descripción
Muestra los pasos seguidos por el usuario que hace uso de la aplicación durante una travesía.
Flujo principal
6. El usuario arranca la aplicación. 6.1. El sistema muestra la actividad principal, la cual contiene instrucciones básicas de funcionamiento y el botón 'Start'. 7. El usuario pulsa el botón 'Start'. 7.1. El sistema muestra un mensaje informativo e inicia los sensores y el *GPS*. 8. El usuario apaga la pantalla. 8.1. La aplicación sigue funcionando en segundo plano (servicio) mientras se realiza la travesía. 9. El usuario enciende la pantalla y pulsa 'Stop'. 9.1. La aplicación desactiva los sensores y el *GPS*.
Flujos alternativos
3. El usuario cierra la aplicación. 3.1. La aplicación sigue funcionando en segundo plano (servicio) mientras se realiza la travesía.

Caso de uso 3: Travesía accidentada.

Descripción
Muestra los pasos seguidos por el usuario que hace uso de la aplicación durante una travesía, produciéndose un accidente en el transcurso de ésta.
Flujo principal
1. El usuario arranca la aplicación. 1.1. El sistema muestra la actividad principal, la cual contiene instrucciones básicas de funcionamiento y el botón 'Start'. 2. El usuario pulsa el botón 'Start'. 2.1. El sistema muestra un mensaje informativo e inicia los sensores y el *GPS*. 3. El usuario apaga la pantalla. 3.1. La aplicación sigue funcionando en segundo plano (servicio) mientras se realiza la travesía. 4. Se produce una aceleración brusca. 4.1. La aplicación lo interpreta como posible accidente. 4.2. Sensores y *GPS* son desactivados. 4.3. El sistema inicia la actividad 'Alarma'. El dispositivo reproduce una alarma al máximo volumen permitido a la vez que vibra, alertando al usuario. Se muestra el botón 'Desactivar'. 5. El usuario pulsa el botón 'Desactivar' antes de 90 segundos. 5.1. La aplicación interpreta que ha ocurrido un falso positivo o que el usuario se encuentra sano y salvo. Desactiva la alarma sonora y vibratoria.
Flujos alternativos
Flujo alternativo 1 (SMS)
5. El usuario no pulsa el botón 'Desactivar' antes de 90 segundos. 5.1. La aplicación interpreta que el usuario se encuentra inconsciente. 5.2. Procede a enviarse un mensaje de texto (SMS) al contacto de emergencia (configurado previamente) indicando latitud y longitud del accidente.
Flujo alternativo 2(Llamada telefónica)
5. El usuario no pulsa el botón 'Desactivar' antes de 90 segundos. 5.1. La aplicación interpreta que el usuario se encuentra inconsciente. 5.2. Se realiza una llamada telefónica al contacto de emergencia (configurado previamente). 5.3. A la vez que la llamada, haciendo uso del sintetizador de voz, se reproduce un mensaje automático en bucle que indicando latitud y longitud del accidente. Dicho mensaje será escuchado por el contacto de emergencia.

Casos de uso de Hermes.

Puede observarse en la figura inferior los casos de uso de la aplicación y posteriormente su descripción detallada.

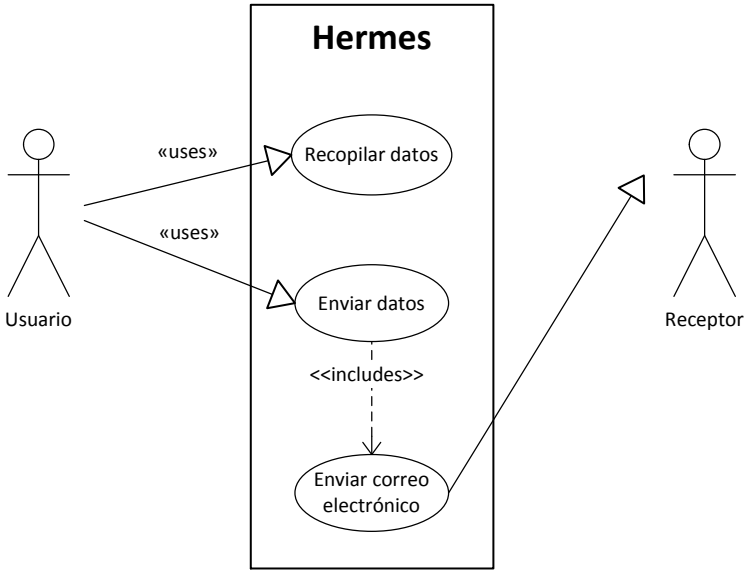

Figura 8. ***Diagrama UML de casos de uso de la aplicación Hermes.***

Caso de uso 1: Recopilación de datos.

Descripción
Muestra los pasos seguidos por el usuario para tomar muestras de las aceleraciones sufridas durante una travesía.
Flujo principal
1. El usuario arranca la aplicación. 1.1. El sistema muestra la actividad principal, la cual contiene instrucciones básicas de funcionamiento y los botones 'Start' y 'Enviar'. 2. El usuario pulsa el botón 'Start'. 2.1. El sistema muestra un mensaje informativo y empieza a recopilar datos (estos son guardados en la memoria externa del teléfono). 3. El usuario apaga la pantalla. 3.1. La aplicación sigue funcionando en segundo plano (servicio) mientras se realiza la travesía. 4. El usuario enciende la pantalla y pulsa 'Stop'. 4.1. La aplicación deja de recopilar datos. 4.2. La aplicación muestra por pantalla las aceleraciones máximas observadas.

Caso de uso 2: Enviar datos recopilados.

Descripción
Muestra los pasos seguidos por el usuario para tomar muestras de las aceleraciones sufridas durante una travesía.
Flujo principal
5. El usuario arranca la aplicación. 5.1. El sistema muestra la actividad principal, la cual contiene instrucciones básicas de funcionamiento y los botones 'Start' y 'Enviar'. 6. El usuario pulsa el botón 'Start'. 6.1. El sistema inicia la aplicación predeterminada de gestión de correo electrónico. Automáticamente se adjunta el fichero de datos. El destinatario será jjpedreno@gmail.com (autor de este PFC). 7. El usuario envía el correo electrónico.

Requisitos no funcionales.

Los requisitos no funcionales de la aplicación Hécate pueden englobarse en tres puntos bien definidos:

- Posibilidad de expansión. Dentro del ámbito del proyecto es requisito fundamental que la aplicación sea capaz de avisar mediante llamada telefónica y SMS en caso de que accidente. Sin embargo se planteará el diseño de Hécate de forma que dichos mecanismos de aviso puedan ser fácilmente ampliados en caso de ser necesario (por ejemplo envío de email, o el uso de una API de un servicio privado) sin que por ello suponga alterar sustancialmente el código ni la estructura del programa. En concreto se hará uso del patrón *Estrategia*, el cual será explicado en detalle en el apartado 'Aplicación Hécate'. Facilidad de uso por parte del usuario. Es un requisito prioritario que la aplicación pueda ser instalada, configurada y usada por cualquier usuario con conocimientos mínimos de manejo de Android. Para ello se diseñará una interfaz minimalista y auto-explicativa, proporcionando las instrucciones necesarias para el correcto uso desde la propia aplicación.
- Portabilidad. Por último, se pretende que la aplicación sea lo más universal posible. Como ya se ha mencionado en el apartado 'Plataforma', es recomendable que la aplicación pueda ser ejecutada por al menos el 90% de los dispositivos Android del mercado. Aunque inicialmente no forma parte del ámbito del proyecto, se estudiará su posible portabilidad a la plataforma iOS[23] si los requisitos de software y hardware lo permiten.

Entorno de desarrollo.

Durante la consecución del proyecto se ha intentado en todo momento usar el software y hardware más adecuado y avanzado para cada una de las fases del mismo. De las herramientas utilizadas, muchas de ellas su uso ya es familiar al haber sido ampliamente usadas a lo largo de la carrera. Se detallan a continuación:

- Android SDK (*Software Development Kit*, kit de desarrollo). Es el conjunto de herramientas básicas, bibliotecas y APIs necesarias para para compilar y depurar aplicaciones para el sistema operativo Android, así como su posterior empaquetamiento, firma electrónica y distribución (por ejemplo en Google Play[24]).

  Algunas de las herramientas a destacar:

  - Bibliotecas y APIs, con distintas versiones según la versión del sistema operativo Android donde será ejecutada la aplicación.
  - Herramientas de configuración, depurado, *profiling*, empaquetado, ofuscación etc. a través de líneas de comandos.
  - Emulador dispositivos Android. Permite ejecutar la aplicación en un entorno virtual, simulando pantalla, teclado y demás elementos hardware.
  - Controlador USB y herramientas necesarias para ejecutar y depurar aplicaciones en dispositivos reales. Útil para aquellas aplicaciones que no pueden depurarse correctamente en un emulador, como es el caso de este proyecto.
  - DDMS (*Dalvik Debug Monitor Server*). Herramienta que permite monitorizar y controlar aspectos del dispositivo que normalmente se encuentran ocultos al usuario. Pila de procesos e hilos, simulación de datos de ubicación *GPS* arbitrarios, control de latencia en la conexión, simulación de llamadas entrantes, logs del sistema y otras funcionalidades.
  - Documentación, códigos de ejemplo y aplicaciones sencillas para ilustrar los diversos tutoriales proporcionados en la web.
  - Librerías de soporte. Permiten la *retrocompatibilidad* proporcionando funcionalidades que no están presentes en versiones anteriores de Android.
  - Librerías externas adicionales, como la API de Google Maps o Google Play Services[25]. No se hace uso de éstas en el proyecto.

  Las herramientas antes mencionadas permiten compilar código escrito en Java, ejecutarlo en un dispositivo Android (virtual o real), y crear los ficheros binarios de la aplicación. Todo ello puede hacerse con tan solo un editor simple de texto y una línea de comandos. Sin embargo para una mayor eficiencia, sobre todo en aplicaciones complejas, es recomendable el uso de un entorno de desarrollo integrado, como el que se describe a continuación.

- Eclipse (*Eclipse Project*, proyecto Eclipse[26]). Es un de IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado) abierto, libre y extensible mediante *plug-ins*. Inicialmente orientado al desarrollo de aplicaciones *Java*, debido a su código abierto, modularidad y posibilidad de crear aplicaciones complejas sin demasiada dificultad,

Eclipse goza de gran popularidad y una amplia comunidad a su alrededor. Ésta proporciona compatibilidad con múltiples lenguajes (*Ada, Python, C++, Scheme, Perl*, etc.), módulos y extensiones para cualquier tarea que se presente a lo largo del ciclo de vida de la aplicación (por ejemplo integración con repositorios como *Subversion*), así como tutoriales para entender su uso.

Eclipse surgió como un proyecto de la división canadiense de la empresa IBM destinado a sustituir la gama de productos *VisualAge* (desarrollados en *SmallTalk*) por una plataforma más moderna basada en *Java*. En 2001 se liberó el código fuente, creándose un consorcio de más de 50 empresas que apoyó el desarrollo abierto de la plataforma. La Fundación Eclipse se creó a partir del consorcio original y a actualmente está compuesta por más de 180 miembros[27].

A día de hoy Eclipse es una plataforma ampliamente usada por desarrolladores independientes, en entornos empresariales y académicos dada la gran cantidad de herramientas, *plug-ins* y tutoriales disponibles; por no hablar de su coste (completamente gratuito).

Se ha considerado el uso de Eclipse durante el desarrollo del proyecto por diversas razones:

- Robustez, funcionalidad y estabilidad. Tras más de 12 años de desarrollo Eclipse presenta una serie de funcionalidades y una estabilidad general que se han considerado adecuadas: organización del código, herramientas de control y depuración, etc.

  A su vez, permite gestionar grandes proyectos de software sin por ello perder estabilidad o rapidez, esto hace que Eclipse sea perfectamente escalable en caso de continuar el desarrollo de la aplicación Hécate más allá del ámbito de este proyecto.

- Integración con Android mediante ADT (Android Development Tools, herramientas de desarrollo de Android[28]). ADT es un *plug-in* diseñado específicamente para Eclipse por los desarrolladores de Android. Añade opciones, menús y otros *front-ends* que facilitan en gran manera la creación y depuración de aplicaciones para entornos Android. Compilación automática de clases, instalación de la aplicación en diferentes dispositivos (virtuales o no), previsualización de la interfaz gráfica escrita en XML, conexión con los dispositivos en modo *debug* etc. son algunas de las funcionalidades que proporciona ADT.
- Familiaridad con el entorno. Debido a que Eclipse ha sido usado con asiduidad a lo largo de la carrera en diversas asignaturas, no existe periodo de aprendizaje y adaptación, lo cual favorece una mayor eficiencia y rapidez en la fase de codificación y depuración de código.

Por último, se considera adecuado mencionar la existencia del IDE *Android Studio*[29] (publicado en Mayo de 2013). Esta plataforma desarrollada por Google y basada en *IntelliJ*

IDEA[30], solventa algunos de los problemas de integración de Eclipse con ADT. Algunas de las novedades que aparta son: editor completo *WYSIWYG* para la interfaz gráfica, integración con *Gradle*[31] para la compilación y gestión de dependencias, refactorizado y *quickfixes* específicos para Android, etc.

Android Studio permite crear aplicaciones Android fácilmente conservando todas las funcionalidades comentadas en el apartado anterior, con una mayor estabilidad y soporte por parte de Google.

Puesto que en el momento de publicarse, las fases de codificación y depuración ya estaban finalizadas, no se creyó adecuado realizar el cambio a esta nueva plataforma. No obstante se considerará su uso en un futuro en caso de ampliarse el proyecto o la continuación del desarrollo de la aplicación Hécate.

- Matlab (Matrix laboratory, laboratorio de matrices[32]). Matlab es una herramienta de cálculo numérico y algebraico que además incorpora su propio lenguaje de programación. Entre sus diferentes funcionalidades, se incluye la manipulación de matrices, representación gráfica de datos y funciones, implementación de algoritmos, creación de interfaces de usuario y la integración de éstas con programas desarrollados en otros lenguajes de programación (*C, C++, Java, FORTRAN*).

  Matlab fue desarrollado a finales de la década de los años setenta por el Departamento de Ciencia Computacional de la Universidad de Nuevo México. Fue creado para que los alumnos pudiesen hacer uso de las herramientas LINPACK y EISPACK (para cálculo numérico y algebra lineal) sin necesidad de conocer *FORTRAN* (lenguaje con el que fueron escritas las dos herramientas antes mencionadas).

  Debido a su potencia y facilidad de uso fue prontamente adoptado por otras universidades y equipos de investigación, sobre todo en el campo de las matemáticas aplicadas. Previendo su potencial, en 1984 se fundó Mathworks, Matlab fue reescrito en C y se procedió a distribución comercial.

  A día de hoy Matlab es una herramienta ampliamente usada tanto en el ámbito académico como en el profesional debido a su facilidad de uso, potencia y versatilidad, ya que posee módulos y extensiones para todo tipo de tareas (tratamiento de señales, matemáticas, química, ingeniería etc.).

  Se ha optado por su uso en el proyecto principalmente por dos razones.

  - Agilidad para la lectura y tratamiento de grandes volúmenes de datos. Debido a que los archivos de datos generados por la aplicación Hermes contienen fácilmente entre 35.000 y 50.000 líneas, es necesario un software que pueda analizar y mostrar por pantalla de forma gráfica estos datos con la menor demora posible. Matlab cumple este cometido de forma altamente eficiente.
  - Scripting. Para cada archivo de pruebas se van a realizar las siguientes operaciones:
    - Lectura y conversión a variables numéricas (matrices).

- Representación gráfica de los datos.
- Aplicación de un filtro paso-bajo.
- Representación gráfica de los datos filtrados.

Para la realización de estas tareas de forma repetitiva Matlab dispone de un sencillo, pero potente, lenguaje de programación que permite la automatización de las mismas. Esto facilita en gran medida la realización de la fase de análisis de datos.

- Smartphone Samsung Galaxy SIII (GT-I9300). Android SDK ofrece un emulador de dispositivos Android perfectamente funcional y que permite emular con gran fidelidad la mayoría de características de un dispositivo con dicho sistema operativo. Sin embargo tanto la aplicación Hermes como Hécate hacen uso de funcionalidades que no son emulables por software.

  El uso del acelerómetro lineal y la realización de llamadas externas de voz hacen necesario un dispositivo físico para el correcto desarrollo y depuración de las aplicaciones antes mencionadas. Se ha optado por tanto por realizar el desarrollo sobre un smartphone Samsung Galaxy modelo SIII. Este dispositivo en el momento de redactarse la memoria es uno de los más representativos del mercado actual, y dispone del hardware y software requeridos por las aplicaciones desarrolladas. Todas las capturas de pantalla incluidas a lo largo de la memoria han sido tomadas con dicho teléfono, 'tal se ve' en la pantalla del mismo.

  Características más representativas:

  - CPU: 1.4GHz Quad-core ARM Cortex-A9
  - GPU: ARM Mali-400 MP4
  - Pantalla: 4.8" Super AMOLED (1280x720 pixels)
  - RAM: 1GB
  - Almacenamiento: 16GB
  - 3G: 850, 900, 1800, 2100 MHz UMTS/HSPA+
  - Sensores: Acelerómetro lineal, *GPS*, magnetómetro, giroscopio, etc.

También se han realizado pruebas de uso en dispositivos LG Google Nexus 4 y Samsung Galaxy Note II.

Instalación y manual de uso.

Instalación.

La aplicación puede ser instalada en cualquier dispositivo que cumpla los siguientes requisitos:

- Sistema operativo Android 2.3 (Gingerbread) o superior.
- Conexión telefónica.
- Sensor de aceleración lineal.
- *GPS*.

Antes de proceder debe asegurarse que se permite la instalación de aplicaciones no provenientes de 'Play Store'. Para ello debe marcarse la casilla situada en Ajustes→Seguridad→Fuentes Desconocidas (u Orígenes Desconocidos). En algunos dispositivos dicha opción puede encontrarse en Ajustes→Aplicaciones.

Mediante un explorador de archivos debe ejecutarse el fichero instalable (suministrado con el nombre Hecate.apk) y aceptar los permisos requeridos para su instalación.

Figura 9. Instalación, pantalla de permisos.

La aplicación está firmada mediante un certificado digital de pruebas válido hasta el año 2037, pasado este tiempo deberá recompilarse la aplicación a través del código fuente suministrado junto a la memoria.

Una vez instalada, puede abrirse la aplicación.

Manual de uso.

Una vez iniciada la aplicación, se muestra la actividad principal:

Figura 10. Pantalla principal de la aplicación.

La actividad queda divida en tres zonas diferenciadas:

- Barra de acción (ActionBar) donde se muestra el icono y el nombre de la aplicación. Además consta de un botón (representado por una llave inglesa) mediante el cual acceder al menú de opciones.
- La parte central de la pantalla muestra un texto dando instrucciones para iniciar la monitorización.
- Botón principal situado en la parte inferior de la pantalla. Con él se inicia o para la monitorización de los sensores.

Si es la primera vez que se accede a la aplicación, es recomendable configurar la misma antes de proceder a su uso. Pulsando en el botón situado en la esquina superior derecha se accede al menú de opciones y preferencias.

Figura 11. Menú de configuración de Hécate.

Las opciones que pueden modificarse son:

- **Teléfono:** Número de teléfono que será contactado por la aplicación en caso de emergencia e incapacidad del usuario.
- **SMS:** Si está opción está marcada, el aviso se realizará por mensaje de texto en vez de una llamada telefónica.
- **Modo de pruebas:** Marcando esta opción el umbral de detección se establece en 1G. Solo debe ser usado cuando se deseen realizar pruebas y debugging de la aplicación (una simple sacudida brusca del dispositivo hará saltar la alarma).

Una vez pulsado el botón 'Start' la aplicación empieza a monitorizar cambios en las aceleraciones sufridas por el dispositivo, detectando si se produce un posible accidente. También se activa el *GPS* y se solicita una actualización de la posición a intervalos de tiempos regulares. Queda constancia de ello en forma de aviso en la barra de notificaciones; este aviso permanecerá activo y no podrá eliminarse mientras la aplicación esté monitorizando.

Cuando la aplicación está activa y monitorizando, puede cerrarse si se desea la actividad principal pulsando el botón 'Atrás' del dispositivo. La aplicación no se detendrá y seguirá ejecutándose en segundo plano (incluso con la pantalla apagada).

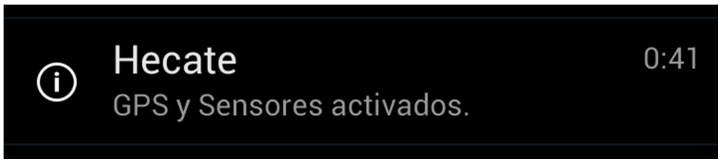

Figura 12. Aviso 'permanente' en la barra de

En caso de sufrir una desaceleración brusca que pueda ser interpretada como un posible accidente, la aplicación mostrará inmediatamente la actividad ‘Alarma’. Al inicio de esta actividad la pantalla del dispositivo se activará con el máximo brillo posible, y sonará por los altavoces externos un distintivo sonido de alarma para alertar al usuario.

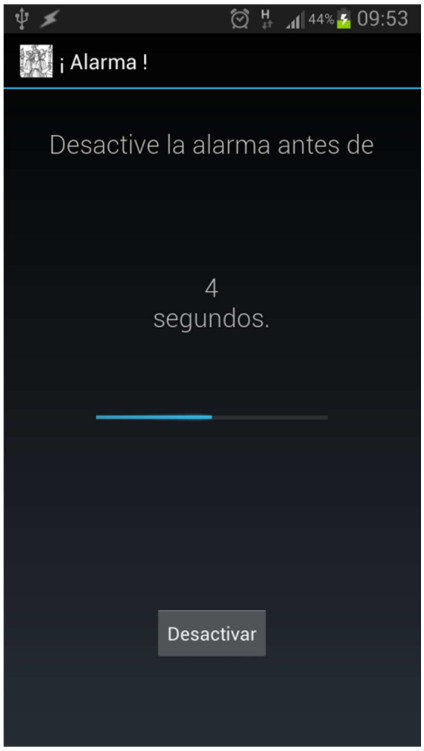

Figura 13. Pantalla de Alarma. Incluye sonido y vibración.

Llegado a este punto el usuario dispone de 90 segundos (10 si la opción Debugging está activada) para pulsar el botón ‘Desactivar’ para parar la alarma, dando así a entender que se ha producido un falso positivo o que el usuario se encuentra consciente a pesar del accidente.

Una vez pasado este tiempo si no se ha pulsado el botón, la aplicación avisará mediante mensaje de texto o llamada telefónica al teléfono escogido en las opciones del accidente ocurrido, incluyendo la localización del mismo en forma de coordenadas geográficas.

Aplicación Hécate.

A pesar de que el código adjunto a la memoria correspondiente a la aplicación Hécate está comentado, en esta sección se procederá a explicar con cierto detalle cada una de las partes de programa que necesiten una explicación más detallada. Se presupone al lector un cierto dominio de Java y conocimientos básicos de programación en Android.

Diseño del código.

Estructura de clases.

Debido a que Hécate no presenta gran complejidad en cuanto al número de clases, se ha optado porque todas estén incluidas en su solo paquete. Para el nombre de éste último se ha seguido la convención de nominación de paquetes de Java[33].

Puede observarse la estructura de clases en la figura de la página siguiente.

Dinámica.

Aunque cada una de las clases está suficientemente comentada, a continuación se realizará un resumen de cada una de ellas; explicándose su función, descripción de sus métodos y una explicación más exhaustiva de aquellas porciones de código que lo necesiten.

Antes de proseguir, como recordatorio se incluye en la figura inferior un esquema con el ciclo de vida interno de las aplicaciones Android.

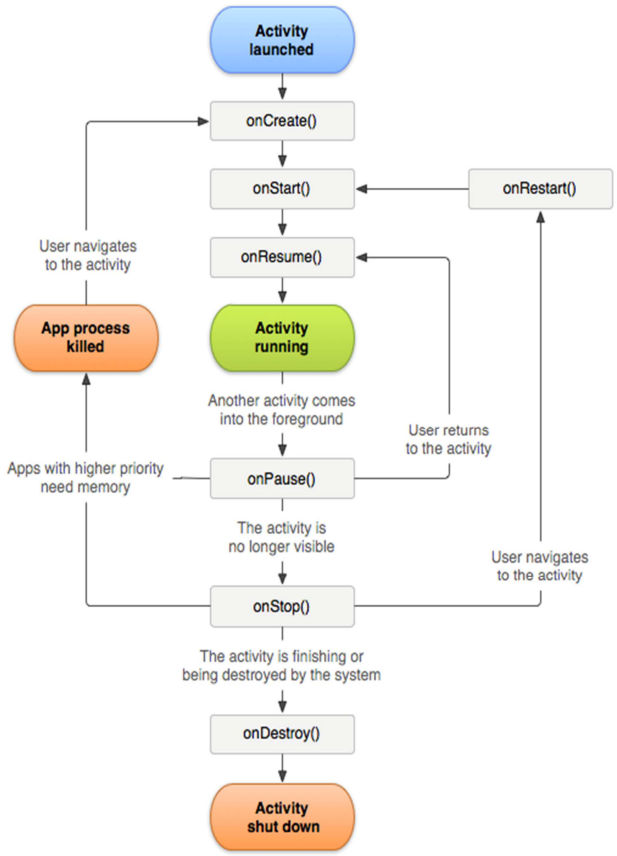

Figura 14. Ciclo de vida de una Actividad dentro de una aplicación.

MainActivity.java

Como su nombre indica, esta es la actividad principal de la aplicación, y por tanto aquella que se muestra cuando Hécate es iniciada. Tiene como función servir de interfaz gráfica así como de elemento de entrada de las acciones del usuario.

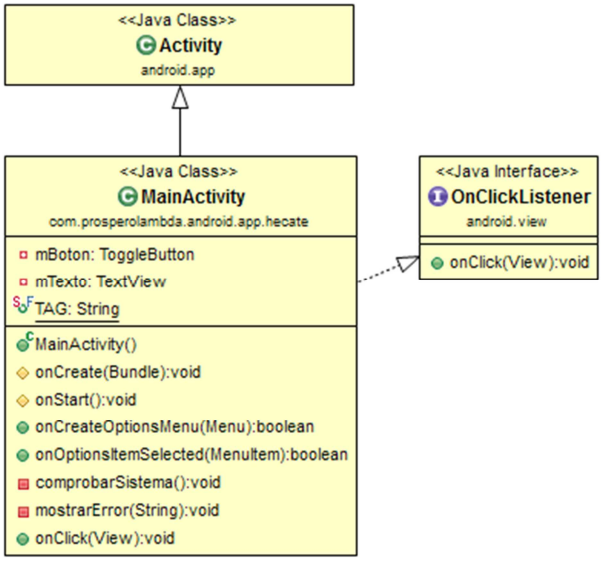

Figura 15. Clase MainActivity.java y sus relaciones.

Método	Observaciones
onCreate	-Personalización de la actividad y comprobaciones iniciales del sistema
onStart	-Inicio de la actividad.
onCreateOptionsMenu	-Establece el icono y menú de 'Opciones'.
comprobarSistema	-Comprueba la existencia de *GPS* y Acelerómetro en el dispositivo.
mostrarError	-Crea y muestra un dialogo de error.
Manejadores de eventos.	
Método	**Observaciones**
onClick	-Inicia el servicio de monitorización.
onOptionsItemSelected	-Abre la actividad de 'Opciones'.

`protected void onCreate(Bundle savedInstanceState)`

En primer lugar, realiza las comprobaciones iniciales del sistema. Acto seguido establece la interfaz gráfica definida en `/res/layout/activity_main.xml` y se asocia el único botón de la actividad con el manejador de eventos. No se profundizará más en este punto ya que dicha interfaz es muy sencilla y no presenta elementos que no puedan encontrarse en el tutorial correspondiente en la guía de desarrollo[34].

Por último, se establece la tipografía 'Roboto' para el texto mostrado en la actividad. Esto no supone ninguna funcionalidad adicional, pero otorga un estilo visual más acorde con las recomendaciones de diseño del equipo de Android[35] (estilo Holo).

```
mTexto= (TextView) findViewById(R.id.principal_texto);
Typeface tf= Typeface.createFromAsset( getAssets(), "fonts/Roboto-Light.ttf");
mTexto.setTypeface(tf);
```

Figura 16. Roboto-Light (TrueType Font) incluída en /assets/fonts

protected void onStart()

Poco hay que destacar en este método, inicia la actividad propiamente dicha. Establece el texto de la interfaz dependiendo de si se está 'monitorizando' en ese momento o no.

public boolean onCreateOptionsMenu(Menu menu)

'Incrusta' el menú de opciones en la *ActionBar* en forma de icono con llave inglesa.

public boolean onOptionsItemSelected(MenuItem item)

Manejador de eventos que se ejecuta en el momento en que un elemento de la ActionBar es pulsado. Puesto que la aplicación solo dispone de un solo icono, se abrirá la actividad de 'Opciones' (`OpcionesActivity.java`).

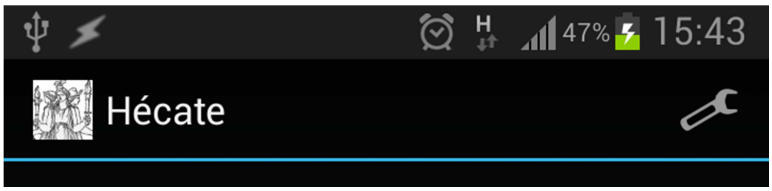

Figura 17. 'ActionBar' incluye el nombre e icono de la aplicación, así como el botón de 'Opciones'

private void comprobarSistema()

Este método privado, el cual se ejecuta nada más empezar la actividad tiene como objetivo comprobar la existencia y disponibilidad de los sensores *GPS* y *Acelerómetro Lineal*, ya que son indispensables para el correcto funcionamiento de la aplicación. En caso de no detectarse alguno de los dos procederá a mostrarse un mensaje de error informando al usuario.

```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
if (sm.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION) == null) {
      Log.e(TAG, "Acelerometro lineal NO detectado");
      mostrarError(getString(R.string.error_acelerometro));
      }
      Log.d(TAG, "Rango máximo del Acelerometro Lineal:
"+sm.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION).getMaximumRange());
      Log.d(TAG, "Rango máximo del Acelerometro Raw:
"+sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER).getMaximumRange());
LocationManager lm= (LocationManager) getSystemService(LOCATION_SERVICE);
if(lm.getProvider(LocationManager.GPS_PROVIDER)==null){
      Log.e(TAG, "Gps NO detectado");
      mostrarError(getString(R.string.error_gps));
```

Figura 18. La referencia a null implica la no existencia o disponibilidad de un sensor.

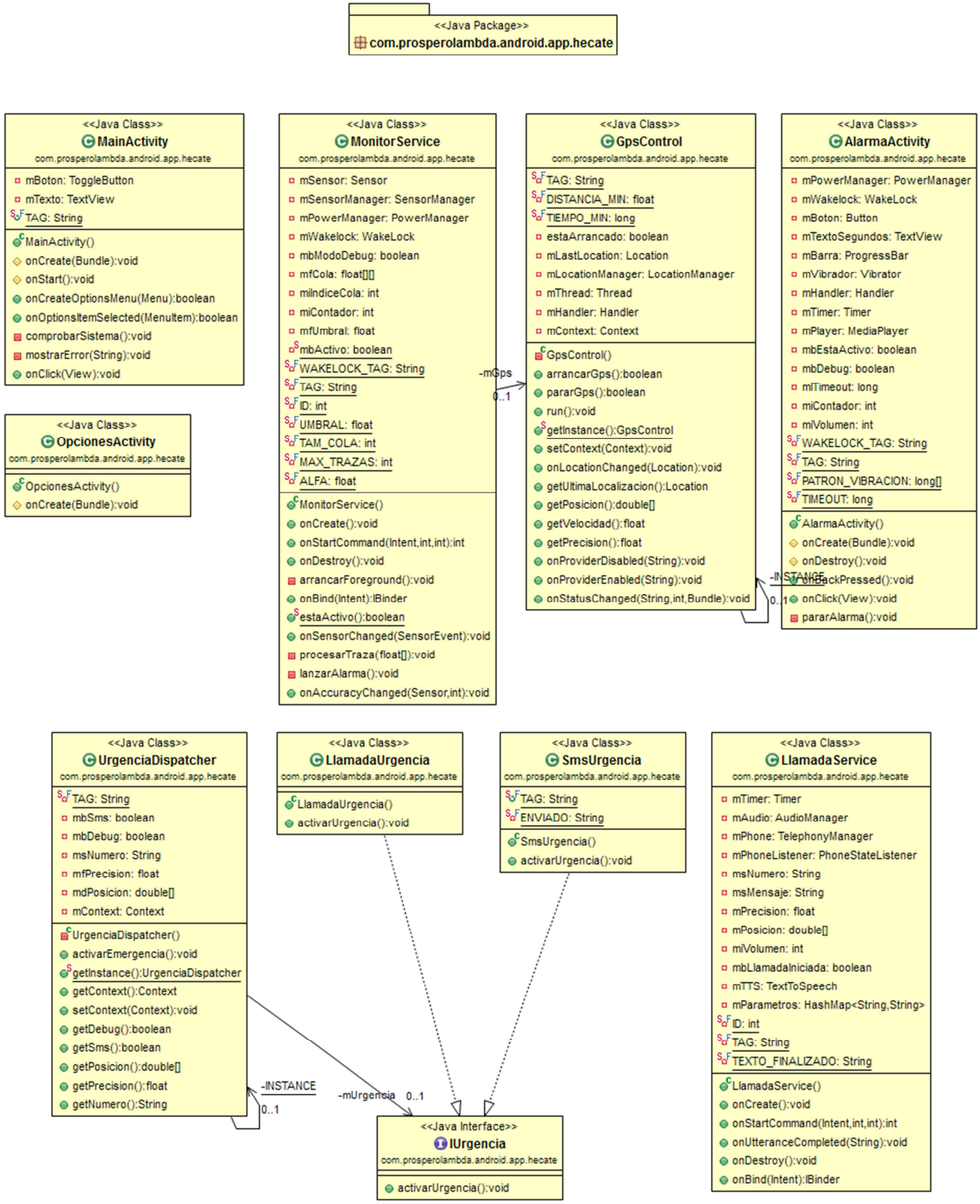

Figura 19. Diagrama de clases de la aplicación Hécate.

`private void mostrarError(String error)`

Este método tiene como función la creación de un `AlertDialog` con el título "Error" y el mensaje incluido en el parámetro. Dispondrá de un solo botón que cerrará la aplicación, ya que no se cumplen los requisitos mínimos para el funcionamiento de la misma.

Para ello se hace uso de AlertDialog.Builder[36], que al igual que muchas otras clases del SDK de Android, hace uso del patrón Builder [DRAFT, incluir referencia].

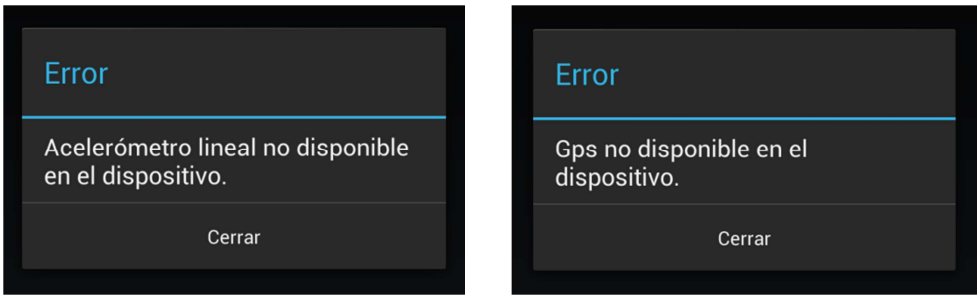

Figura 20. Se informa al usuario de la causa del error.

`public void onClick(View v)`

Manejador de eventos asociado a 'pulsaciones sencillas', este método está asociado al único botón presente en la interfaz. Su función es la de activar o desactivar la monitorización de los sensores según se inicie o finalice una travesía.

¿Cómo distingue si el servicio ya está activo o no? Por defecto la API de Android no proporciona un método para comprobar si un servicio está activo. Para solventar este problema se hace uso del método estático `MonitorService.estaActivo()`que devuelve un booleano indicando el estado del servicio (ver análisis de la clase `MonitorService.java`).

Si el servicio ya estaba activo, lo desactiva y reestablece el texto de bienvenida inicial. En caso contrario, inicia el servicio e informa al usuario que ya puede apagar la pantalla o salir de la aplicación si lo desea.

Justo antes de iniciar el servicio propiamente dicho, comprueba si el usuario ha activado el *GPS*. Si no es así, se da la opción de activarlo para poder proseguir (abriendo las opciones de 'Servicios de localización' mediante la construcción de un `AlertDialog`). El servicio no se iniciará a menos que el *GPS* esté encendido.

```
if (lm.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
      startService(intent);
      mTexto.setText( getString(R.string.principal_texto_encendido) );
}
```

Figura 21. Es requisito indispensable que el GPS esté activo para iniciar el servicio.

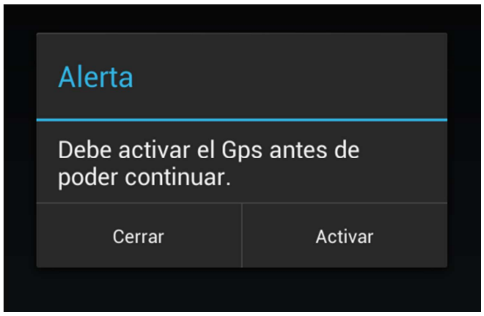

Figura 22. No se iniciará el servicio hasta activar el GPS.

MonitorService.java

Puede considerarse esta clase como el núcleo de la aplicación, ya que Hécate está diseñada para funcionar en segundo plano mientras el móvil se encuentra con la pantalla apagada. Su función es la de ir monitorizando el acelerómetro lineal, tomando lecturas actualizadas a intervalos regulares. Si después de filtrar y analizar dichas muestras se detecta un posible accidente, el servicio arrancará la actividad `AlarmaActivity.java`, y por último se detendrá.

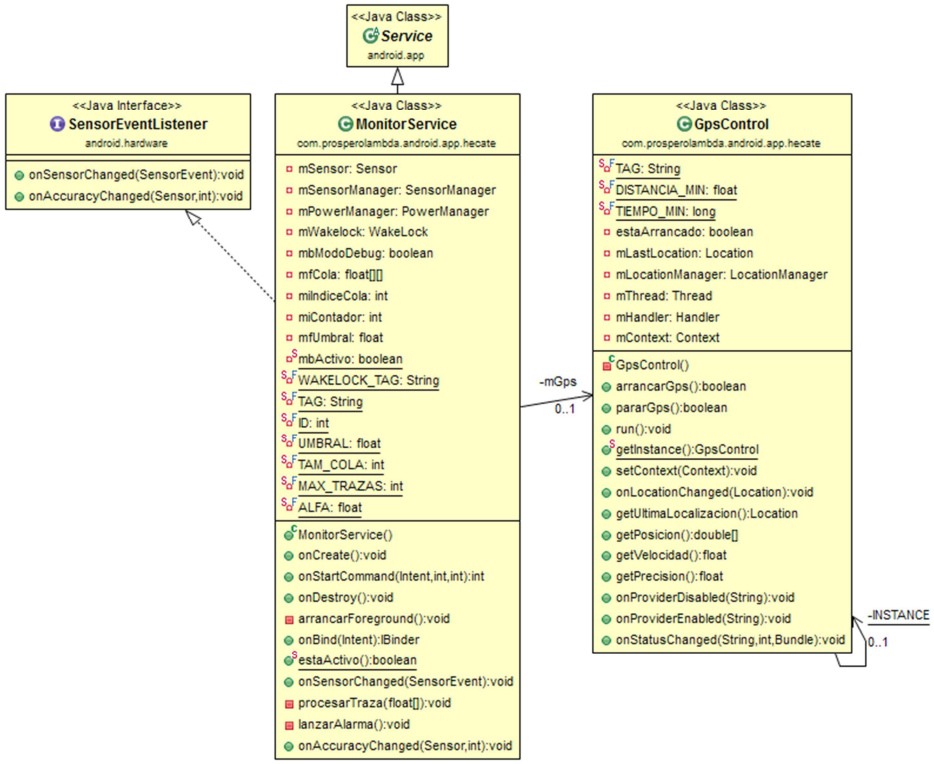

Figura 23. Clase MonitorService.java y sus relaciones.

Método	Observaciones
onCreate	-Configuración inicial de servicio, inicialización de variables.
onStartCommand	-Inicio del servicio.
onDestroy	-Liberación de recursos.
arrancarForeground	-Establece icono en la barra de notificación.
estaActivo	-Devuelve true/false dependiendo de si el servicio está iniciado.
procesarTraza	-Filtra y analiza las muestras tomadas por el acelerómetro.
lanzarAlarma	-Arranca la actividad AlarmaActivity y para el servicio.
Manejadores de eventos.	
Método	**Observaciones**
onSensorChanged	-Recoge cada nueva lectura del acelerómetro.

`public void onCreate()`

Al igual que en una actividad, este método tiene como función realizar las configuraciones iniciales pertinentes. Según si se ha configurado la aplicación para modo 'Debug' o no, se tomará como umbral máximo de aceleración permitida un valor u otro (1G y 4G respectivamente).

También se activa un '*Wakelock* parcial[37]'; esto evita que el dispositivo una vez se apague la pantalla entre en modo 'suspensión' y por tanto cese toda actividad de la *CPU*. Una vez activado dicho mecanismo, el servicio funcionará con normalidad hasta la destrucción del mismo. El uso de *wakelocks* debe hacerse con extremo cuidado, ya que sin un debido control una aplicación podría hacer un uso demasiado intensivo de la *CPU* y drenar la batería en un corto periodo de tiempo.

```
mPowerManager= (PowerManager) getSystemService(Context.POWER_SERVICE);
mWakelock= mPowerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, WAKELOCK_TAG);
mWakelock.acquire();
```

Figura 24. Una vez adquirido el testigo, este se mantiene hasta que se libera explícitamente.

Se activa el GPS, y por último se crea una cola circular usando un array bidimensional donde se guardarán las últimas 10 muestras leídas por el acelerómetro para cada uno de los tres ejes.

`public int onStartCommand(Intent intent, int flags, int startId)`

Inicio del servicio. Se activa el acelerómetro con una latencia aproximada de 70ms[38] entre cada muestra.

Por último, devuelve la constante `START_REDELIVER_INTENT`. Eso significa que si en algún momento el sistema operativo Android requiere 'matar' el proceso por necesidades de memoria, volverá a reanudar este servicio en cuanto sea posible.

`public void onDestroy()`

No hay mucho que destacar acerca de este método. Una vez se finaliza el servicio se detienen las lecturas del acelerómetro y se devuelve el testigo del '*wakelock*'.

`private void arrancarForeground()`

Este servicio hace uso de algunas características que pueden suponer un alto gasto de batería (como por ejemplo el GPS) si se mantiene activo por largo periodos de tiempo, se hace por tanto recomendable recordar al usuario de la presencia activa de dicho servicio.

La función de este método es la de colocar en la barra de notificaciones un icono y un mensaje distintivo que sirva como recordatorio de que el servicio está activo y monitorizando. Puede verse en el capítulo 'Instalación y manual de uso' su aspecto.

`public static boolean estaActivo()`

Un servicio por implementación solo puede tener en activo una sola instancia del mismo. Android no proporciona en su API mecanismos para saber si un servicio se encuentra iniciado o no. Esto está hecho de forma deliberada para evitar el problema de '*race conditions*[39]'.

Cuando se hace imperativo conocer de manera local (dentro de la misma aplicación) el estado de un servicio, la solución más eficiente y elegante es establecer una variable privada estática[40], y conocer su valor mediante este método.

`public void onSensorChanged(SensorEvent event)`

Invocado cada vez que se toma una lectura del acelerómetro, la variable `event.values` (vector unidimensional) contiene en formato `float` los valores de las aceleraciones medidas (de cada uno de los tres ejes del teléfono) en m/s^2.

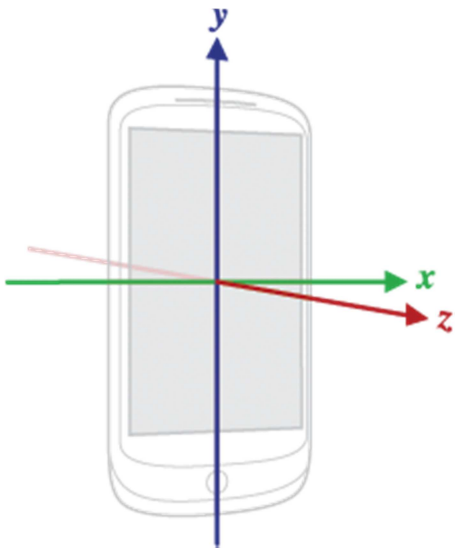

Figura 25. Representación visual de los tres ejes con respecto al teléfono.

private void procesarTraza(float[] traza)

Cada muestra leída por el acelerómetro será encolada tras haber sido filtrada. La aplicación de un sencillo filtro paso-bajo lineal[41] se realiza para descartar picos aberrantes que puedan ser producidos por el usuario al sacudir involuntariamente el teléfono, por la vibración del vehículo,

```
mfCola[0][actual]= mfCola[0][anterior]+(Math.abs(traza[0])-mfCola[0][anterior])*ALFA;
mfCola[1][actual]= mfCola[1][anterior]+(Math.abs(traza[1])-mfCola[1][anterior])*ALFA;
mfCola[2][actual]= mfCola[2][anterior]+(Math.abs(traza[2])-mfCola[2][anterior])*ALFA;
```

Figura 26. Filtro paso bajo lineal aplicado a cada valor antes de ser encolado.

un bache en el camino etc.

Se empieza una cuenta cada vez que una traza supera el umbral permitido, si 6 muestras consecutivas superan el umbral (equivalente a medio segundo aproximadamente) se considerará que existe un accidente o situación de peligro, y procederá a invocarse la función `lanzarAlarma`.

private void lanzarAlarma()

Lanza la actividad `AlarmaActivity`. El flag `FLAG_ACTIVITY_NEW_TASK`[42] es necesario ya que dicha actividad se está arrancando desde un servicio.

GpsControl.java

Esta clase, de la cual solo hay una instancia en todo momento, es la encargada de mantener actualizada en todo momento la localización del usuario mediante el *GPS*. A través de sus métodos puede iniciarse o pararse el *GPS* y obtener la última localización recibida.

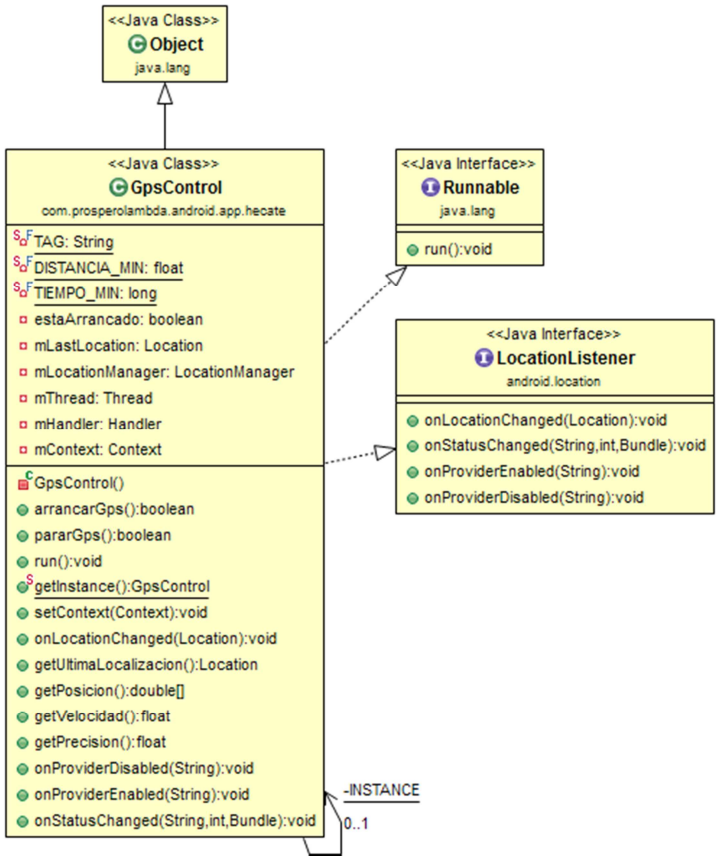

Figura 27. Clase GpsControl.java y sus relaciones.

Método	Observaciones
GpsControl	-Constructor privado de la clase.
arrancarGps	-Inicia las actualizaciones de la posición por *GPS*
pararGps	-Detiene las actualizaciones de la posición por *GPS*
run	-Implementación de 'Runnable'.
getInstance	-Devuelve la instancia del objeto.
setContext	-Establece el contexto de la aplicación.
getUltimaLocalizacion	-Devuelve el último objeto '*Location*'
getPosicion	-Devuelve la última posición.
getVelocidad	-Devuelve la velocidad del usuario.
getPrecision	-Devuelve el radio de precisión de la localización.
Manejadores de eventos.	
Método	**Observaciones**
onLocationChanged	-Recoge cada nueva lectura del acelerómetro.

Referencias.

[1] Phantom Tracking - http://www.phantomtracking.com/product.aspx
[2] Android Developers Guide: Training - http://developer.android.com/training/index.html
[3] Wikipedia: Smartwatch - http://en.wikipedia.org/wiki/Smartwatch
[4] Wikipedia: SmartTV - http://en.wikipedia.org/wiki/SmartTV
[5] Open Handset Alliance - http://www.openhandsetalliance.com/
[6] Open Handset Alliance Members - http://www.openhandsetalliance.com/oha_members.html
[7] Android Open Source Project - http://source.android.com/
[8] Apache License 2.0 - http://www.apache.org/licenses/LICENSE-2.0
[9] Android Developers Blog - http://android-developers.blogspot.com.es/2008/09/announcing-android-10-sdk-release-1.html
[10] Motorola Mobility – http://www.motorola.com
[11] IDC Mobile Phone Market Share 2Q2013 - http://www.idc.com/getdoc.jsp?containerId=prUS24257413
[12] Sensortech SMB 380 Datasheet - https://www.olimex.com/Products/Modules/Sensors/MOD-SMB380/resources/SMB380_Preliminary_Datasheet_Rev13_20070918.pdf
[13] InveSense MP-6050 Product Specification - http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf
[14] GPS Standard Positioning Service Performance Standard - http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf
[15] BCM47511 Product Highlight – http://files.shareholder.com/downloads/BRCM/1500168922x0x440049/7883c1a7-f330-4521-ba0d-e5b96e095b5e/BRCM_News_2011_2_9_Mobile_and_Wireless_News.pdf
[16] AvagoTech 3012 - http://www.avagotech.com/docs/AV02-3026EN
[17] GNSS Inside Mobile Phones - http://www.insidegnss.com/auto/marapr11-VanDiggelenREV.pdf
[18] Android 2.3 Gingerbread - http://developer.android.com/about/versions/android-2.3-highlights.html
[19] Android SDK API 9 Differences Report - http://developer.android.com/sdk/api_diff/9/changes.html
[20] Android Developers Guide, Supporting platform versions - http://developer.android.com/training/basics/supporting-devices/platforms.html
[21] Android Dashboard - http://developer.android.com/about/dashboards/index.html
[22] Salt Websites - http://www.saltwebsites.com/2012/android-accelerometers-screen-off
[23] Apple iOS - http://www.apple.com/ios/what-is/
[24] Google Play Store - http://play.google.com/
[25] Google Play Services - http://developer.android.com/google/play-services/index.html
[26] Eclipse Project - http://eclipse.org/eclipse/
[27] Eclipse Foundation Members - http://www.eclipse.org/membership/showAllMembers.php
[28] ADT Plugin - http://developer.android.com/tools/sdk/eclipse-adt.html
[29] Android Studio - http://developer.android.com/sdk/installing/studio.html
[30] IntelliJ IDEA - http://www.jetbrains.com/idea/
[31] Gradle - http://www.gradle.org/
[32] Mathworks Matlab - http://www.mathworks.es/products/matlab/
[33] Java Documentation (Oracle), Naming a Package – http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html
[34] Android Developers Guide: Building a Simple user Interface – http://developer.android.com/training/basics/firstapp/building-ui.html
[35] Roboto Tipography - http://developer.android.com/design/style/typography.html
[36] Android Developers Guide: AlertDialog.Builder – http://developer.android.com/reference/android/app/AlertDialog.Builder.html
[37] Android Developers Guide : Wakelocks - http://developer.android.com/reference/android/os/PowerManager.WakeLock.html
[38] Android Developers Guide: SensorManager – http://developer.android.com/reference/android/hardware/SensorManager.html#SENSOR_DELAY_GAME
[39] What is a race condition? - http://www.celticwolf.com/blog/2010/04/27/what-is-a-race-condition/
[40] Google Groups: Android Developers - https://groups.google.com/forum/#!topic/android-developers/jEvXMWgbgzE

[41] Frame Rate-Independent Low-Pass Filter - http://phrogz.net/js/framerate-independent-low-pass-filter.html

[42] Android Developers Guide: Intent – http://developer.android.com/reference/android/content/Intent.html#FLAG_ACTIVITY_NEW_TASK