Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de un sistema de tracking en tiempo real en entornos sin cobertura 3G integrado en una aplicación para dispositivos Android con implementación del protocolo 'Location Based Offline Services via SMS'

Autor/es

Ricardo Pallás Román

Director/es

Ramón Piedrafita Moreno

Escuela de Ingeniería y Arquitectura
2013/2014

Desarrollo de un sistema de tracking en tiempo real en entornos sin cobertura 3G integrado en una aplicación para dispositivos Android con implementación del protocolo 'Location Based Offline Services via SMS'
Resumen

El Grupo de Sistemas de Información Avanzados (IAAA) está desarrollando, junto a la empresa GeoSpatiumLab y el grupo HowLab, un sistema de tracking en entornos offline, basado en el protocolo "Location Based Offline Services via SMS" desarrollado por los grupos anteriores, para la monitorización en tiempo real de nodos móviles.
En un principio, el protocolo había sido pensado para ser utilizado con dispositivos específicos incorporados con GPS.

En este TFG se propone la ampliación del protocolo para soportar dispositivos móviles basados en Android en el sistema de tracking, así como el desarrollo en un framework Android que implemente dicho protocolo. De esta manera, se incorporan dispositivos Android al sistema de tracking, siendo capaces de operar en entornos offline (sin cobertura 3G) y online (con cobertura 3G), haciendo uso del framework desarrollado en este TFG.

Está ampliación se ha propuesto debido a las características de los nuevos teléfonos basados en Android. Éstos disponen de GPS y sensores (luminosidad, campo magnético, presión atmosférica, etc.). Por ello, son dispositivos válidos para funcionar en el sistema, ya que tienen las características de hardware que tienen los dispositivos específicos del sistema (ambos disponen de GPS), e incluso las superan (los dispositivos Android tienen sensores). Se han incorporado nuevas operaciones el protocolo debido a las nuevas características de los dispositivos Android. Dichas nuevas operaciones son de lectura de sensores.

En un principio, uno de los posibles usos del sistema era la monitorización de la posición en vehículos agrícolas en entornos offline, de tal manera que cada vehículo agrícola incorporaba uno de los dispositivos específicos. Pero, con el resultado de este TFG, los posibles casos de uso del sistema de tracking aumentan en gran medida, debido a que el uso de dispositivos Android en la actualidad está muy extendido. Por ejemplo, se podría realizar un sistema de tracking para personas mayores (puesto que no se necesitarían dispositivos específicos, sino solo sus teléfonos Android), entre otros muchos ejemplos.

Por último, como se pudo ver en la propuesta del proyecto, existía un objetivo doble, es decir, además del framework se pretendía la creación de una aplicación que debía ser un caso de uso, en el cual se refleje el potencial del framework implementado.

Índice

1. Introducción

En este capítulo se va a explicar el ámbito en el que se ha desarrollado el proyecto, tanto en el contexto profesional como en el tecnológico. Además se va a exponer la motivación del TFG junto con la estructura que va a seguir este documento.

1.1. Contexto profesional

1.1.1. Grupo de Sistemas de Información Avanzados (IAAA)

El grupo de Sistemas de Información Avanzados (IAAA[1]) es un grupo de I+D de carácter multidisciplinar, pero con un marcado perfil informático, adscrito al Instituto de Investigación en Ingeniería de Aragón de la Universidad de Zaragoza[2].

La actividad de investigación del grupo está enfocada en las tecnologías de software abierto, distribuido e interoperable, principalmente mediante servicios web y para sistemas de información geoespacial, abarcando áreas como Sistemas de Información Geográfica (SIG), Teledetección, Servicios Basados en la Localización (SBL) y, con una atención especial a las Infraestructuras de Datos Espaciales (IDEs).

1.2. Contexto tecnológico

1.2.1. Sistema de tracking y protocolo 'Location based offline services via SMS'

Consiste en un sistema de tracking compuesto por un servidor central, un cliente de dicho servidor y de 1 a N nodos (ver Figura 1).

El sistema de tracking produce información geoespacial en sus nodos, y con ella lleva a cabo las operaciones necesarias. Esa información geoespacial necesita ser transferida desde los nodos hasta el servidor, donde es procesada. Los nodos deben ser capaces de enviar la información desde cualquier parte. En ocasiones, la conexión a internet estará disponible, pero no siempre será así. El otros casos, la conexión 3G/UMTS no estará disponible, en cualquier caso, los nodos deberán ser capaces de mandar información. En entornos offline (sin cobertura 3G), se utilizarán SMS a través de la red GSM/GPRS, para comunicarse con el servidor.

[1] http://iaaa.cps.unizar.es/
[2] http://i3a.unizar.es/

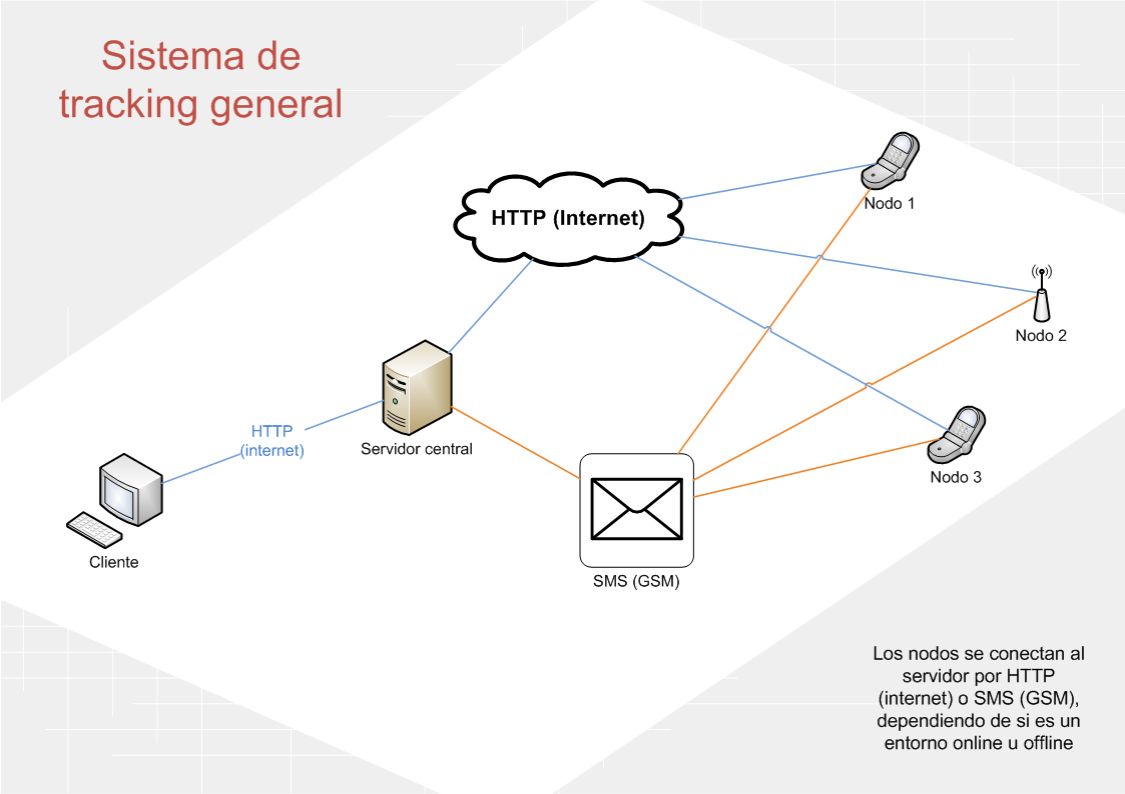


Figura 1. ***Visión general del sistema de tracking***

El principal objetivo es desarrollar sistemas capaces de trabajar en entornos tanto online como offline. En entornos online se utilizará Internet y HTTP como protocolo de comunicación, y en entornos offline se utilizará GSM y objetos JSON como codificación de texto.

Además, el servidor no necesita preocuparse por el tipo de dispositivo que le está mandando la información geoespacial. Los dispositivos que actúan como nodos pueden ser, sensores, vehículos, o en el caso del TFG, smartphones. Todos esos dispositivos se comunican con el servidor utilizando el mismo protocolo.

Ese protocolo es 'Location based offline services via SMS', el cual define los formatos de comunicación, reglas y operaciones que deben seguir tanto el servidor como los nodos del sistema. Que todos sigan el protocolo para comunicarse, permite que el sistema de tracking sea escalable y extensible, lo cual lo hace útil para muchas aplicaciones.

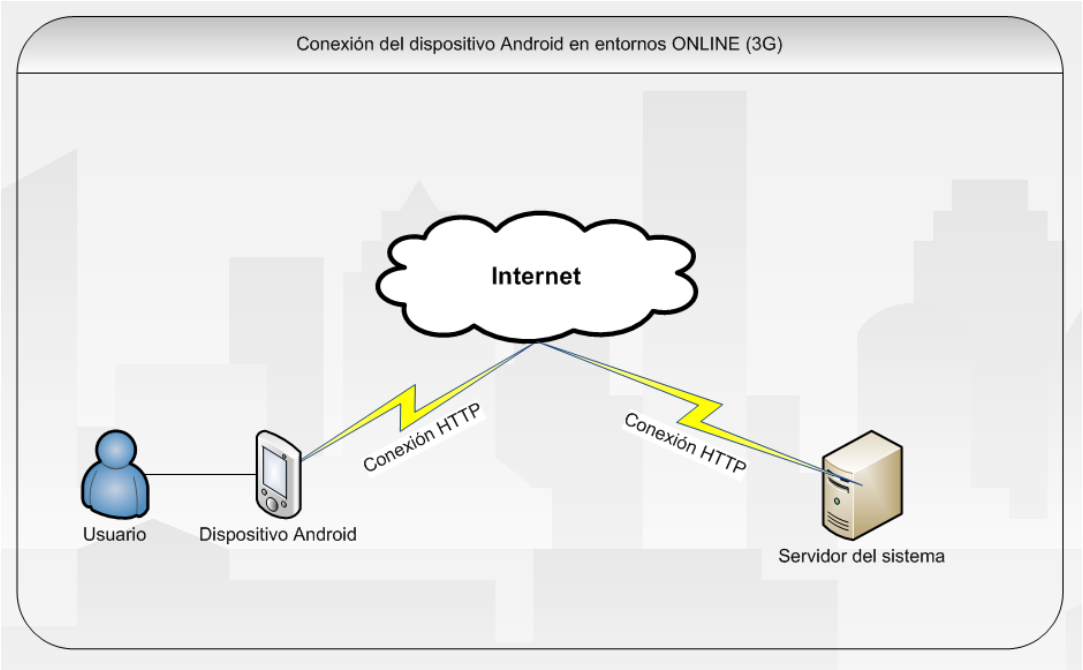


Figura 2. *Conexión en entornos online vía HTTP*

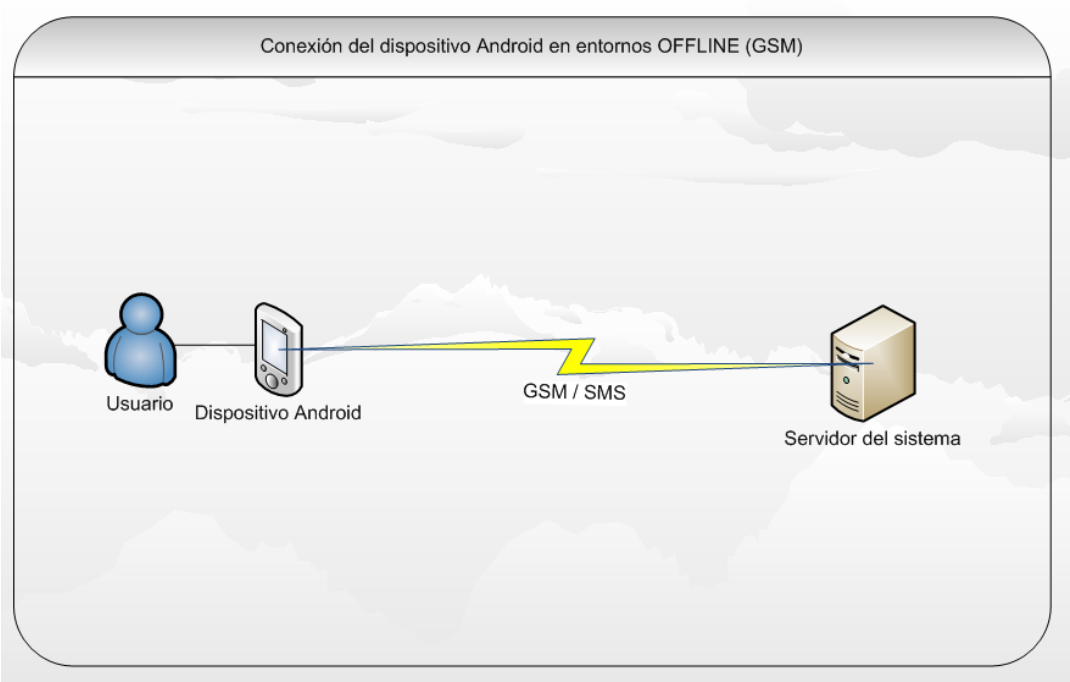


Figura 3. *Conexión en entornos offline vía SMS*

1.2.2. Android

Es un sistema operativo principalmente para móviles que ha cogido fuerza en los últimos tiempos. Actualmente Android supone en torno a un 70-80% de los dispositivos móviles a nivel mundial mientras que iOS su principal competidor supone en torno a un 20% del mercado mundial.

1.2.3. Java

Es un lenguaje de programación orientado a objetos que puede ser ejecutado en diversas plataformas sobre una máquina virtual. De esta manera el mismo código puede ser ejecutado en cualquier dispositivo que disponga de una máquina virtual de Java. En el caso de Android, el código compilado se empaqueta, junto con otra información, en un fichero APK, el cual al ser instalado en un dispositivo Android, se puede ejecutar en la máquina virtual Java que posee Android, llamada Dalvik.

1.2.4. JSON

Se trata del acrónimo *de JavaScript Object Notation*. Es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos, en este contexto, es que es mucho más sencillo escribir un analizador sintáctico y es más ligero.
En el contexto del TFG se ha utilizado, puesto que en el sistema de tracking, los nodos se comunican con el servidor utilizando este formato, y viceversa. Además se ha hecho uso de una biblioteca llamada GSON para facilitar la serialización de objetos Java a formato JSON.

1.2.5. SMS

Short Message Service o SMS es un servicio de mensajería de texto para teléfonos, web o sistemas de comunicación móvil. Utiliza protocolos de comunicación estándar para permitir a dispositivos móviles el intercambio de mensajes de texto cortos.
En el TFG se utiliza para la comunicación entre el servidor y los nodos del sistema de tracking en los escenarios en los que no hay cobertura 3G disponible (se utiliza la tecnología GSM).

1.2.6. HTTP

Hypertext Transfer Protocol o HTTP es un protocolo estándar del nivel de aplicación, utilizado en sistemas distribuidos, colaborativos o de hipermedia.
Es utilizado dentro del contexto del TFG para la comunicación bidireccional entre servidor y nodos del sistema de tracking, en los casos en los que la cobertura 3G está disponible.

1.2.7. Google Cloud Messaging y notificaciones PUSH

Al extender el protocolo a dispositivos Android, la forma de comunicación en entornos 3G entre el servidor y los nodos, debe ser adaptada para nodos Android. El problema es que los teléfonos Android no tienen una IP fija para que el servidor se pueda comunicar con ellos (los dispositivos específicos utilizados en el sistema si la tienen). Es por ello que la manera de comunicarse en entornos online es a través de las notificaciones PUSH, y para ello se ha utilizado el *Google Cloud Messaging* o GCM.

GCM es un servicio gratuito de Google que permite mandar datos desde un servidor a los usuarios de una aplicación ejecutada en un dispositivo Android y viceversa. El servicio se encarga de controlar las colas de mensajes y la entrega de éstos a la aplicación del dispositivo Android.

Las notificaciones PUSH son los datos que reciben las aplicaciones de parte del servidor a través de GCM. La arquitectura del servicio se puede ver en la Figura 4.

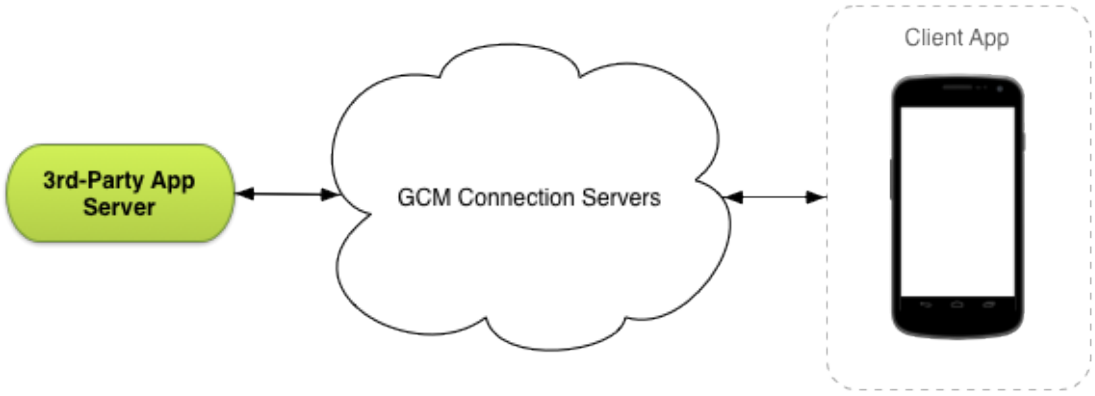


Figura 4. *Arquitectura Google Cloud Messaging*

1.2.8. SQLite

Es un sistema gestor base de datos relacional. El entorno de desarrollo de Android ofrece clases para trabajar con él. La principal ventaja es que es un sistema ligero, y muy utilizado en tecnologías móviles. Se ha utilizado en el TFG para almacenar datos persistentes del sistema.

1.3. Objetivos y motivación del proyecto

Actualmente el uso de dispositivos móviles se ha extendido por todo el mundo y ha crecido en los últimos años. Tener un Smartphone se ha convertido en algo cotidiano entre las personas. El sistema operativo que lidera la cuota de mercado actualmente es Android, con una gran ventaja sobre su competidor iOS.

Así pues, el grupo IAAA y *GeoSpatiumLab,* poseen como una de sus líneas de trabajo el desarrollo de un sistema de tracking capaz de funcionar en sistemas offline. Por lo que la principal motivación del proyecto ha sido el extender dicho sistema, para que cualquier dispositivo Android pueda ser capaz de funcionar y trabajar dentro de él, así como de dotar de mayor funcionalidad al mismo, aprovechando los sensores que incorporan los teléfonos Android. De esta manera, en el sistema se aumenta el número de usos que se le puede dar a este sistema de tracking.

En consecuencia, el principal objetivo de este trabajo ha sido el desarrollar una framework para dispositivos Android, que implemente el protocolo del sistema de tracking, tanto como las reglas de comunicación con el servidor en entornos offline (SMS) y entornos online (HTTP), así como las operaciones descritas en dicho protocolo. Debido a que en este trabajo también se propone una incorporación de nuevas operaciones de lectura de sensores, otro objetivo es la implementación de otro framework para la lectura de sensores y GPS en Android. Por lo tanto el framework que implementa el protocolo hará uso del otro, para poder ejecutar las operaciones relacionadas con la lectura de sensores y GPS.

Con el fin de mostrar el framework en uso, se ha desarrollado una aplicación, que lo utiliza, a modo de caso de uso que simula el sistema de tracking dentro del dispositivo. Se ha incorporado una interfaz gráfica con controles para que el usuario pueda ejecutar operaciones del protocolo, ver el formato de los mensajes, ajustar las configuraciones, guardar estadísticas de mensajes enviados tanto en SMS como en HTTP, etc.

Una primera visión a la arquitectura de los frameworks desarrollados se puede ver en la siguiente imagen:

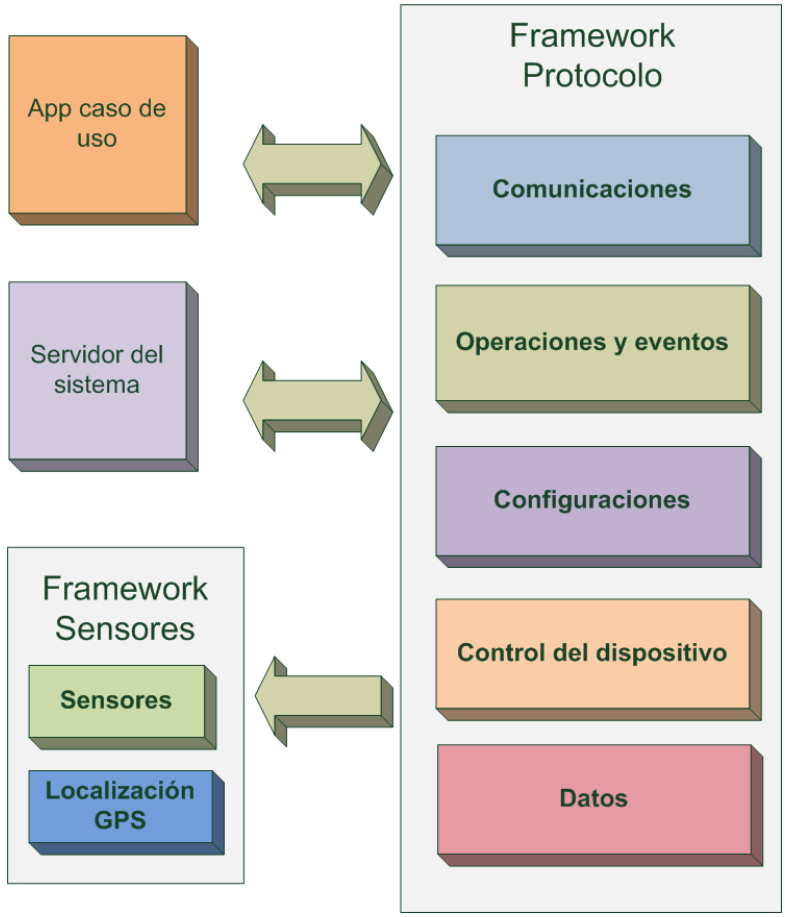


Figura 5. *Visión general de los frameworks desarrollados en el TFG*

1.4. Estructura del documento

Este documento ha sido redactado y estructurado siguiendo la estructura recomendada por el grupo IAAA. Se diferencian tres partes claras. La primera es la memoria como tal del proyecto, en la que se muestra, sin gran nivel de detalle, el trabajo hecho. Después, se muestran los anexos, donde se detalla más el trabajo y sus resultados. Por último, una tercera parte con la bibliografía, un listado de acrónimos y un índice de figuras. Seguidamente, se muestra el esquema de la estructura:

- Parte I: Memoria:
  - Introducción
  - Descripción del trabajo realizado
  - Conclusiones

- Parte II: Anexos:
  - (A) Análisis de requisitos
  - (B) Casos de uso
  - (C) Documentación de la arquitectura
  - (D) Modelo dinámico
  - (E) Adaptación y extensión del protocolo
  - (F) Diseño y documentación del módulo de comunicaciones
  - (G) Diseño módulo operaciones
  - (H) Diseño del módulo ModeController
  - (I) Diseño módulo Data y modelo base de datos
  - (J) Diseño framework sensores
  - (K) Pruebas
  - (L) Manual utilización frameworks
  - (M) Aplicación caso de uso
  - (N) Gestión del proyecto
- Parte III:
  - Bibliografía
  - Acrónimos
  - Figuras

2. Trabajo realizado

En este capítulo se detalla el trabajo realizado en este TFG, destacando los aspectos más importantes. En el siguiente capítulo se encuentran los anexos donde se amplía la información del trabajo.

2.1. Formación

Uno de los primeros trabajos que se llevó a cabo al principio del TFG fue la formación en la tecnología Android y un estudio detallado del protocolo 'Location Bases Services via SMS". Esto fue así porque la tecnología Android era prácticamente nueva para el proyectando, y tampoco había trabajado con protocolos de comunicación. De esta manera, la formación se dividió en dos partes:

2.1.1. Formación Android

Con el fin de familiarizarse con la plataforma, lo primero que se hahecho fue un estudio de bibliotecas Android desarrolladas por la empresa GeoSpatiumLab, de visión de mapas.

Una vez introducido en Android, analizando a alto nivel los frameworks que se iban a desarrollar, se vio que la formación en Android debería centrarse en:

- **Comunicación SMS:** como enviar y recibir los SMS en Android; saber si ha sido recibido, si ha sido enviado. Además de análisis del formato de mensajes PDU.
- **Localización GPS:** lectura de datos de GPS y modos de ahorrar energía.
- **Sensores:** lectura de sensores en Android y modos de ahorrar energía.
- **Testing:** testing en Android, utilizando jUnit para pruebas unitarias.
- **SQLite:** utilización de bases de datos en Android
- **Preferencias Android:** como almacenar preferencias en Android, para los parámetros de configuración del protocolo.
- **JSON:** estudio del formato de datos JSON y cómo trabajar con él en Android.
- **Crear bibliotecas:** Crear bibliotecas Android que permitan ser utilizadas por Aplicaciones.
- **Notificaciones PUSH:** utilización de notificaciones PUSH y el servicio Google Cloud Messaging.

2.1.2. Estudio del protocolo

Se ha estudiado el protocolo para ver cómo adaptarlo e implementarlo en dispositivos Android. El proyectando no tenía experiencias con implementación de protocolos de comunicación, por lo cual, su estudio ha servido como aprendizaje.

Primero se analizaron las operaciones y formatos de datos que debía ofrecer el dispositivo. Se vieron que todas eran factibles, excepto la de recibir mensajes por HTTP. La solución alternativa propuesta, para mitigarlo, fue utilizar notificaciones PUSH.

En cuanto a otras exigencias del protocolo, como los modos de energía, se pudieron implementar adaptando el protocolo al dispositivo móvil.

Por último se hizo un estudio de sensores en Android (ver Anexo 3.3.E2) para ver si era posible añadir operaciones de lectura de sensores en el protocolo. Los resultados demostraron que sí, debido que la lectura de todos los sensores en Android devuelven el mismo tipo de datos, fácil de serializar.

2.2. Análisis de requisitos

Tras el estudio de las versiones previas del protocolo de comunicación, y análisis de cómo implementar dicho protocolo en sistemas Android, se han especificado los requisitos, que especifican la funcionalidad que deberán soportar los frameworks desarrollados. A continuación, se listan los requisitos a un alto nivel de abstracción. Más adelante, se detallan en el anexo de Análisis de requisitos.

Stakeholders

- **Usuario**: es el usuario que está utilizando el sistema en su dispositivo Android.
- **App caso de uso:** es la aplicación caso de uso desarrollada en este TFG que utiliza los frameworks desarrollados.
- **Servidor**: es el servidor del sistema de tracking que especifica el protocolo.

Requisitos funcionales Framework Protocolo

- **RFFP-1:** El sistema deberá ser capaz de enviar y recibir SMS.
- **RFFP-2**: El sistema deberá implementar las operaciones de un nodo detalladas en el protocolo “Location Based Services via SMS”.
- **RFFP-3:** El sistema deberá ser capaz de enviar eventos de nodo al servidor, detallados en el protocolo “Location Based Services via SMS”.
- **RFFP-4:** El sistema deberá ser capaz de enviar peticiones vía HTTP.
- **RFFP-5:** El sistema deberá ser capaz de recibir notificaciones PUSH
- **RFFP-6:** El sistema deberá permitir configurar los parámetros del protocolo
- **RFFP-7:** El sistema deberá permitir iniciar la ejecución del protocolo
- **RFFP-8:** El sistema deberá permitir detener la ejecución del protocolo

Requisitos funcionales Framework Sensores

- **RFFS-1:** El sistema deberá ser capaz de leer los valores de los sensores del dispositivo donde se ejecuta.
- **RFFS-2**: El sistema deberá ser capaz de obtener las coordenadas, a través de GPS, en las que se encuentra el dispositivo donde se ejecuta.

Requisitos funcionales Aplicación caso de uso

- **RFACU-1:** La aplicación deberá ser capaz de iniciar y finalizar la ejecución del protocolo.
- **RFACU-2:** La aplicación deberá ser capaz de mostrar las respuestas a las operaciones del protocolo al usuario.

- **RFACU-3:** La aplicación deberá ser capaz de permitir que el usuario configure los parámetros de ejecución del protocolo.
- **RFACU-4:** La aplicación deberá ser capaz de mostrar estadísticas de peticiones y respuestas de operaciones por SMS y por HTTP.
- **RFACU-5:** La aplicación deberá ser capaz de mostrar y refrescar los últimos valores leídos de los sensores del dispositivo incluyendo GPS.
- **RFACU-6:** La aplicación deberá ser capaz de mostrar y refrescar un mapa de Zaragoza, incluyendo las localizaciones obtenidas del dispositivo durante su funcionamiento en el sistema de tracking, y una ruta seguida por el dispositivo.
- **RFACU-7:** La aplicación deberá ser capaz de mostrar y refrescar gráficos interactivos con los valores leídos de los sensores del dispositivo.

Requisitos no funcionales

- **RNF-1:** El sistema debe tener acceso a los permisos del dispositivo: enviar y recibir SMS, internet, acceder a la localización, leer el estado del teléfono y de la red, obtener cuentas de usuario, impedir que el teléfono entre en modo inactivo.

Requisitos no funcionales Framework Protocolo

- **RNFFP-1:** El framework deberá implementar el hash de seguridad especificado en el protocolo.

- **RNFFP-2:** El framework deberá permitir ejecutarse en varios modos de energía.

- **RNFFP-3:** El dispositivo que ejecute el framework deberá tener sistema operativo Android 2.1 o superior.

- **RNFFP-4:** El dispositivo que ejecute el framework deberá tener instalados los Google Play Services, para permitir el funcionamiento de las notificaciones PUSH.

- **RNFFP-5:** El framework debe almacenar estadísticas de SMS enviados y recibidos, así como comandos HTTP enviados y recibidos.

- **RNFFP-6**: El framework debe poder acceder al dispositivo con permisos de súper-usuario o root.

Requisitos no funcionales Framework Sensores

- **RNFFS-1:** El dispositivo que ejecute el framework deberá disponer de GPS.

- **RNFFS-2:** El dispositivo que ejecute el framework deberá tener sistema operativo Android 2.1 o superior.

Requisitos no funcionales Aplicación Caso de Uso

- **RNFACU-1:** El dispositivo que ejecute la aplicación caso de uso deberá tener sistema operativo Android 4.0 o superior.

2.3. Arquitectura general del sistema

En este apartado se presenta una visión general de la arquitectura (ya se había adelantado en la Figura 5), de los frameworks desarrollados, mediante un diagrama de componentes. En dicho diagrama se puede ver los principales componentes y sus relaciones del framework del protocolo y del framework de lectura de sensores. Para una visión más completa de la arquitectura del sistema desarrollado, ver el anexo 3.3.C) .

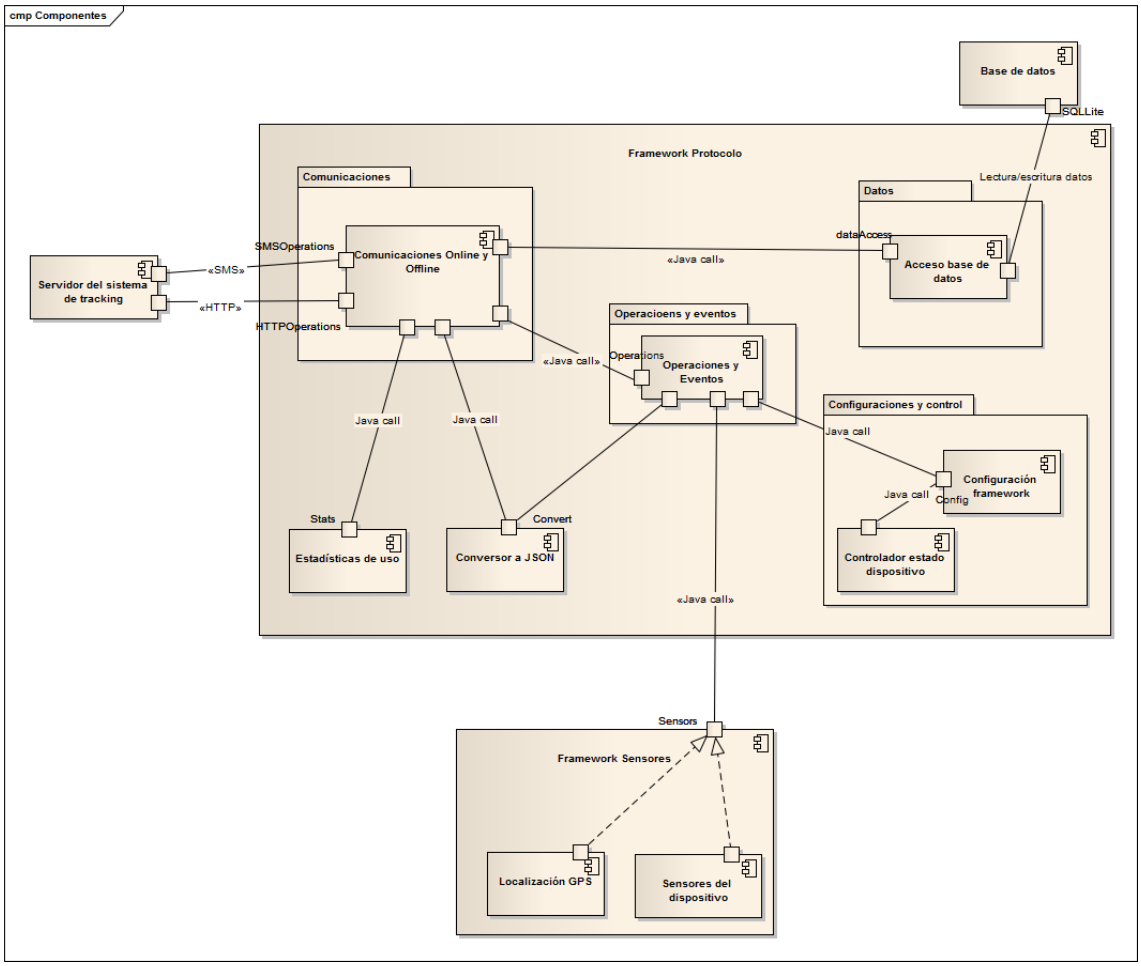


Figura 6. Diagrama de componentes de los frameworks

En el diagrama se pueden ver 3 componentes principales: el servidor del sistema de tracking, el framework protocolo y el framework sensores.

El servidor del sistema de tracking controla todos los nodos del mismo a través del protocolo de comunicación. Aunque el servidor no se implementa en este TGF, se ha incluido en el diagrama para ver la comunicación del framework protocolo con dicho servidor.

El framework de sensores, hay dos componentes, Localización GPS y Sensores del dispositivo:

- **Localización GPS:** es el encargado de obtener la localización del dispositivo mediante su GPS, además debe permitir elegir un tiempo de refresco de dicha localización para poder ahorrar energía.
- **Sensores del dispositivo:** es el componente encargado de leer los sensores disponibles en el dispositivo.

En el framework protocolo, que es el encargado de implementar las comunicaciones y operaciones del protocolo, podemos diferenciar 7 componentes claros:

- **Comunicaciones online y offline:** es el componente encargado de comunicarse con el servidor del sistema de tracking para recibir sus órdenes y enviarle resultados y eventos. Este componente implementa los dos tipos de comunicaciones, tanto la comunicación entornos online (HTTP) y en entornos offline (SMS).
- **Operaciones y eventos:** este componente es el encargado de ejecutar las operaciones definidas por el protocolo para un nodo, así como los eventos que se envían regularmente al servidor.
- **Estadísticas de uso:** registra el número de mensajes enviados y recibidos, tanto por SMS como por HTTP.
- **Conversor JSON:** componente encargado de codificar/decodificar JSON, para facilitar la comunicación con el servidor acorde a los formatos de datos descritos por el protocolo.
- **Acceso base de datos:** este componente se encarga de facilitar la inserción y borrado de datos utilizados por el framework en la base de datos SQLite.
- **Configuración framework:** su función es la de almacenar y dar acceso a los parámetros de configuración del framework, como por ejemplo el modo de energía, o la dirección IP del servidor.
- **Controlador de estado del dispositivo:** su función principal es controlar el estado de las redes del dispositivo (conectarlas, desconectarlas, saber si se dispone de conexión 3G…).

A continuación se muestra un diagrama de módulos, en el cual se pueden ver los principales módulos en los que se han dividido los frameworks y sus relaciones. Más información detallada está disponible en el anexo 3.3.C) .

Framework protocolo

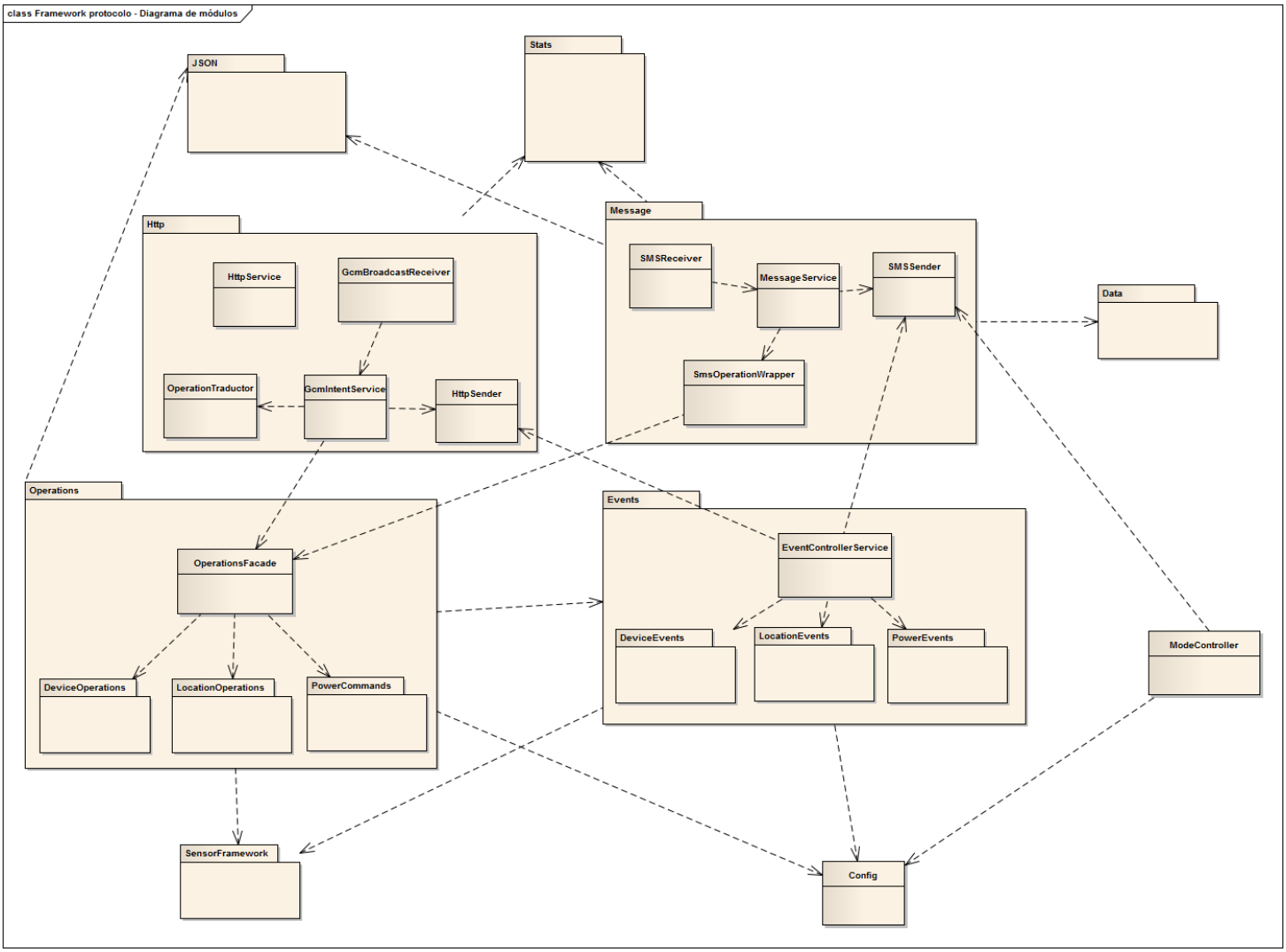


Figura 7. *Diagrama de módulos del framework Protocolo*

Framework sensores

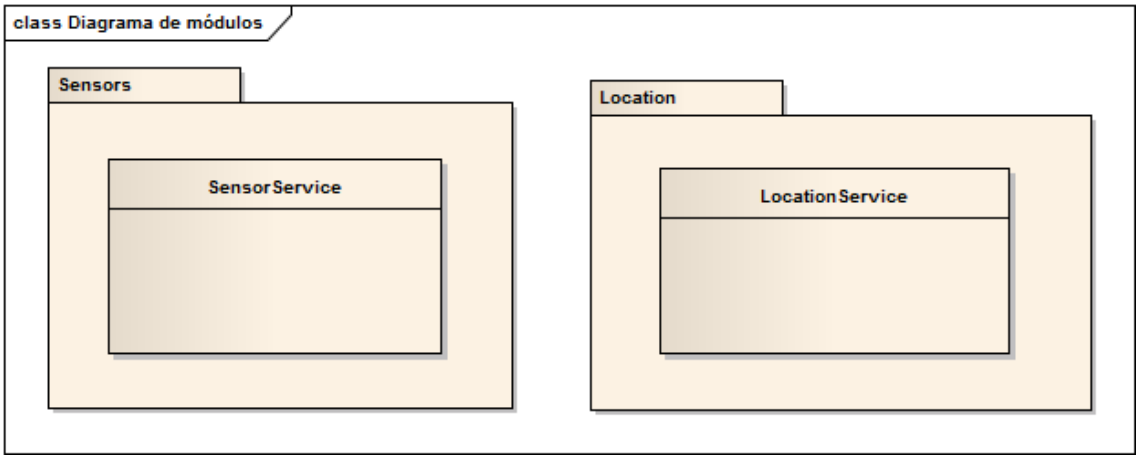


Figura 8. *Diagrama de módulos del framework Sensores*

Seguidamente se explica la responsabilidad de cada módulo:

Framework protocolo

- **Message:** es el módulo encargado de recibir, enviar y leer mensajes. Una vez que le ha llegado un mensaje debe analizarlo para saber qué operación del módulo "Operations" se debe ejecutar. Tiene 4 clases:
    - **SMSReceiver:** extiende la clase BroadcastReceiver, para recibir los SMS. Una vez recibido el SMS, se lo pasa a la clase MessageService, mediante un Intent de Android. Es decir, esta clase es un filtro, consume un evento, lo procesa y genera otro evento.
    - **SMSSender:** es la clase encargada de enviar mensajes SMS.
    - **MessageService:** Obtiene los mensajes recibidos de la clase SMSReceiver e invoca a la clase SmsOperationWrapper, pasándole como argumento el SMS recibido. La clase SmsOperationWrapper le devolverá el resultado de la ejecución de la operación correspondiente al SMS, entonces, MessageService, enviará el resultado al servidor a través de la clase SMSSender.
    - **SmsOperationWrapper:** esta clase, a través del SMS recibido que le pasan como argumento en su constructor, es capaz de invocar a la operación correspondiente del módulo "Operations" y devolver el resultado de dicha operación a la clase MessageService.
- **JSON:** este módulo ofrece operaciones para transformar objetos/cadenas a objetos JSON y viceversa.
- **Stats:** es un módulo encargado de recoger las estadísticas de uso de la aplicación.
- **Context:** módulo utilizado para almacenar los parámetros de configuración del framework. Estos parámetros son utilizados en el protocolo.
- **Data:** este módulo es el encargado de ofrecer operaciones de acceso a la base de datos.
- **Operations:** este módulo implementa las operaciones propias de un nodo, tal como especifica el protocolo. Tiene una clase que actúa como fachada e invoca a las operaciones. Cada operación es una clase, cada clase tiene un método *execute* que realiza su operación y devuelve un String codificado en JSON con el resultado.
- **Events:** este es el módulo que implementa y controla los eventos de un nodo especificados en el protocolo. Tiene una clase EventController que es un servicio Android que ofrece operaciones para invocar a los eventos, y que controla ejecución periódica de eventos. Cada evento está representado por una clase separada.
- **ModeController:** este módulo es el encargado conectar y desconectar la red móvil periódicamente dependiendo de la configuración de energía almacenada en la clase SharedPreferences, obtenida en el módulo Context.

Framework sensores

- **SensorFramework:** este módulo ofrece operaciones de lectura de sensores y localización. Está compuesto por dos módulos:

- **Sensors**: es el módulo encargado de leer los sensores del dispositivo.
- **Location**: es el módulo encargado de obtener la localización del dispositivo.

2.4. Gestión y desarrollo del proyecto

Primeramente, al ser las tecnologías utilizadas nuevas para el proyectando, se partió de una fase previa de formación. Debido al desconocimiento de estas tecnologías y con el fin de desarrollar un trabajo de mayor calidad, se evitó el uso de ciclos de vida en cascada, los cuales son mucho más rígidos y no tan adecuados para entornos y proyectos nuevos para el desarrollador. En vez de eso, se llevó a cabo un desarrollo iterativo, siguiendo la filosofía Scrum, en la cual, el trabajo se dividió en requisitos y cada requisito en tareas. A estas tareas se les asignó una prioridad, con el fin de obtener el mínimo producto viable lo antes posible, y en cada ciclo añadir más funcionalidades.

Todo ello está vinculado con la búsqueda de la excelencia y la calidad de los productos, hecho por el que *GeosptiumLab* ha promovido un Sistema Integrado de Gestión de la Calidad y el Medioambiente sobre la base de los requisitos de las Normas UNE-EN-ISO 9001:2008 (noviembre de 2008) y UNE-EN-ISO 14001:2004 (diciembre de 2004), para todas las actividades que desarrolla, que se complementa con los requisitos de la norma ISO/IEC 15504 "Software Process Improvement Capability Determination" (SPICE), nivel de capacidad 2, con el objetivo primordial de satisfacer a sus clientes mediante la prestación de servicios de la máxima calidad y respeto al entorno. Este Sistema Integrado de Gestión obtuvo su certificación formal en el año 2011.

Los incrementos en los que se dividió el proyecto por orden de prioridad son los siguientes:

Primera iteración:

- Framework protocolo:
  - Capaz de enviar y recibir SMS
  - Capaz de ejecutar operaciones (excepto las localización)
  - Almacenamiento de preferencias y configuración de la aplicación

Segunda iteración:

- Framework sensores:
  - Capaz de leer la localización
  - Capaz de leer los sensores
- Framework protocolo:
  - Se añade el envío de eventos
  - Se añaden las operaciones de localización
  - Se añaden estadísticas

Tercera iteración:

- Framework protocolo:
  - Se añade el funcionamiento en varios modos de energía. (El dispositivo tiene intervalos en los cuales se conecta y desconecta el modo avión para ahorrar batería).

- Se mejora el componente de comunicaciones SMS (si se envía algún mensaje cuando no hay red SMS disponible, se guarda en base de datos, y cuando vuelve a haber red disponible se mandan todos los mensajes pendientes.)
- Se implementa el módulo de comunicación por HTTP y notificaciones PUSH.
- Por último, se crea una aplicación caso de uso

2.5. Pruebas

Se han realizado pruebas unitarias, de aceptación y de integración con el servidor. Se explican por separado, en los siguientes puntos.

2.5.1. Pruebas unitarias

Las pruebas unitarias de los frameworks se han hecho utilizando el framework de testing JUnit. Se han escrito tests automáticos y se ejecutaban en JUnit para probar los frameworks. Esto se ha podido hacer ya que Android provee un framework de testing basado en JUnit. Este framework ofrece una API para testing, que está basada en la API JUnit y extendida con un framework de instrumentación y clases de test específicas para Android (ver Figura 9).

Sin embargo, no todas las pruebas se pueden automatizar con JUnit en Android, puesto que no ofrece instrumentalización para probar clases que extienden BroadcastReceivers, por ejemplo.

Para este tipo de pruebas no automatizables, se ha utilizado una plantilla para hacer pruebas unitarias. Dicha plantilla se utilizó en la asignatura de Verificación y Validación. Consta de una columna con pruebas y otra con resultados. Cada fila representa una prueba, que tiene un título, unas precondiciones, unas acciones para ejecutarla y unos resultados esperados. En la columna de resultados se apunta si se ha pasado cada prueba, y se puede añadir un comentario opcional.

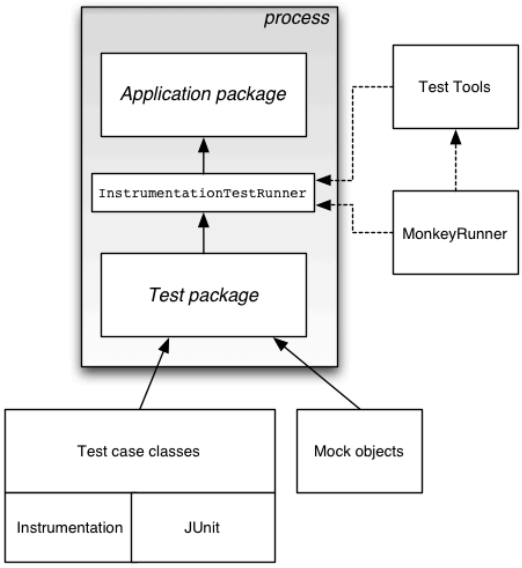


Figura 9. *Framework de testing de Android*

2.5.2. Pruebas de aceptación

Además de las pruebas unitarias, al final de caca ciclo de desarrollo se han realizado pruebas de aceptación de los requisitos implementados en ese ciclo.

Se han realizado 3 pruebas de aceptación, porque el trabajo se ha dividido en 3 iteraciones. En las pruebas de aceptación se comprobaba la definición de hecho y los criterios de aceptación para cada requisito.

2.5.3. Pruebas de integración

Puesto que el servidor del sistema de tracking está siendo desarrollado en paralelo, se han hecho pruebas de comunicaciones entre el servidor y el nodo Android. Para ello, el servidor ha invocado operaciones del nodo y ha leído sus respuestas. Estas invocaciones han sido a través de SMS y de HTTP con notificaciones PUSH.

Todas las pruebas se detallan más adelante en el anexo J: Pruebas.

2.6. Aplicación caso de uso

Se ha creado una aplicación caso de uso que utiliza los frameworks desarrollados en el TFG. La aplicación tiene dos pantallas principales, una llamada Inicio, y otra llamada Simulador. Estas pantallas se pueden ver en la Figura 11. *Pantalla Inicio* y en la Figura 10.

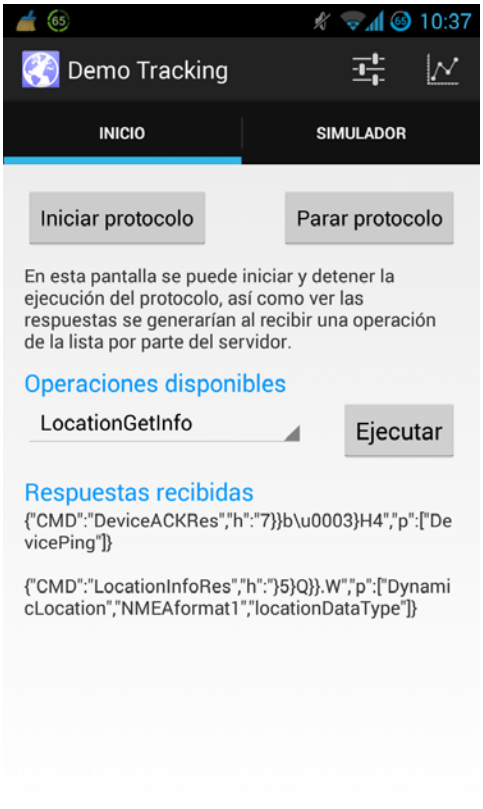


Figura 11. *Pantalla Inicio*

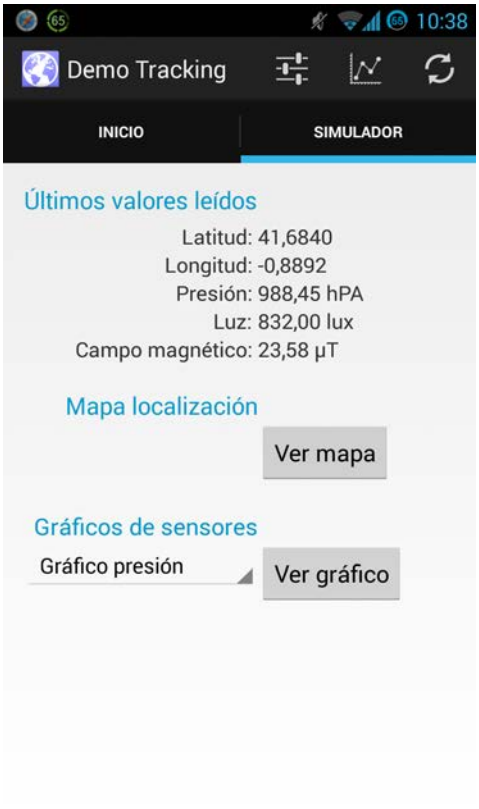


Figura 10. Pantalla Simulador

En la pantalla Inicio hay dos botones, uno para iniciar y otro para parar la ejecución del protocolo, es decir, para que se lancen o paren los servicios creados en los frameworks, y el teléfono empiece a actuar como un nodo dentro del sistema de tracking. También se pueden ver los mensajes que el nodo generaría como respuesta a una petición de operación por parte del servidor.

En la pantalla de Simulador se pueden ver los últimos valores leídos del GPS y de los sensores. Éstos se pueden actualizar pulsando al botón de actualizar de la barra de acción. Pulsando sobre el botón "Ver mapa" se puede ver un mapa con la ruta que ha hecho el dispositivo marcada por puntos enlazados con líneas. Por último se puede ver una gráfica con el histórico de valores leídos de los diferentes sensores del dispositivo.

Además, en la barra de acción de la aplicación se puede acceder a una pantalla de configuraciones y otra de estadísticas. En la pantalla de configuraciones el usuario puede ver y cambiar todos los parámetros del nodo (por ejemplo, su modo de energía o la tasa de obtención de localizaciones).

En la pantalla de estadísticas se muestran el número de mensajes recibidos y enviados, así como el número de operaciones enviadas y recibidas a través de HTTP.

3. Conclusiones

En este apartado se explican las conclusiones obtenidas, algunas posibles líneas futuras del proyecto y una valoración personal del mismo.

3.1. Valoración del trabajo realizado

El trabajo ha partido de una formación en el desarrollo en la plataforma Android, así como un análisis del protocolo de comunicación del sistema de tracking, para poder obtener una buena solución en la integración de dispositivos Android al sistema.

Tras concluir este trabajo, es posible afirmar que se ha conseguido una integración completa de smartphones Android al sistema, sin tener que cambiar el protocolo (sólo adaptarlo e incluso, extenderlo para darle más funcionalidad.)

A pesar de que se ha desarrollado una aplicación caso de uso que simula la interacción del dispositivo dentro del sistema, se ha tratado de ser en todo momento proactivo y se han desarrollado mecanismos para que las soluciones resultantes en el trabajo puedan ser utilizadas en proyectos futuros. Por este motivo se han implementado dos frameworks, el framework protocolo y el framework sensores.

El framework protocolo se encarga de gestionar todas las comunicaciones con el servidor del sistema, tanto online como offline. Además, implementa las operaciones y eventos necesarios, así como una configuración del modo de operación del mismo. Es por ello, que cualquier aplicación o caso de uso que se quiera realizar en el contexto del sistema de tracking, podrá incluir como biblioteca el framework protocolo, por tanto se favorece la reusabilidad en aplicaciones futuras.

De la misma manera, la funcionalidad de lectura de sensores y de acceso a la localización del dispositivo, se ha encapsulado en otro framework. Esto se ha hecho para poder reutilizar las funcionalidades que ofrece la biblioteca. La principal diferencia con el código libre accesible en internet e funciona as es que, esta biblioteca, permite obtener los valores de los sensores y la localización en segundo plano y por intervalos, minimizando el gasto de energía. Cualquier aplicación (incluso ajena al sistema de tracking) puede utilizar la biblioteca con diversos fines.

Por último, a modo de conclusión, se puede afirmar que los objetivos han sido satisfechos y los requisitos se han cumplido.

3.2. Líneas futuras

El resultado de este TFG da una base para seguir trabajando y crear aplicaciones que hagan uso de ambos frameworks. Por ello, las líneas futuras son amplias, algunas abordables a corto y medio plazo.

Al incorporar dispositivos Android al sistema de tracking, sus posibles aplicaciones crecen. Por ejemplo, inicialmente se había pensado tener en tractores y demás maquinaria agrícola dispositivos específicos del sistema de tracking que se comunicaran en el servidor. Con dispositivos Android, se podría utilizar los frameworks para desarrollar un sistema de control de personas mayores; cada una llevaría la aplicación en su teléfono Android y se podría saber dónde están en cualquier momento ya que aunque no haya cobertura 3G es capaz de funcionar y comunicarse con el servidor.

Otra opción sería, aprovechando la extensión de la funcionalidad del protocolo, el desarrollo de un sistema que captara los valores de los sensores (magnetómetro, barómetro, sensor de luz, temperatura…) de varios smartphones Android para realizar estudios de factores climatológicos.

Una posibilidad, podría ser incluir los frameworks en una aplicación en un dispositivo Android para niños. De esta manera los padres podrían acceder a una interfaz gráfica del servidor y ver la posición de sus hijos.

También, se podría utilizar para controlar flotas de camiones de una empresa. Es decir, cada conductor llevaría en su teléfono una aplicación con los frameworks, y la empresa podría ver la localización de los camiones en tiempo real, ajustando el modo de energía para ahorrar batería. No se necesitaría invertir en hardware específico de localización, con los teléfonos móviles bastaría.

Otras posibles líneas futuras, fuera del contexto del sistema de tracking, serían utilizar como base del protocolo sensores. El cual provee un acceso de los sensores y localización GPS del móvil minimizando el gasto de energía y en segundo plano. Las posibilidades podrían ser:

- Aplicación de deporte: guarda la ruta hecha por del deportista y muestra medidas como tiempo y velocidad.
- Aplicación de información genérica del móvil: se podrían mostrar con el framework los valores de todos los sensores disponibles en un dispositivo Android y su localización a modo informativo.
- Aplicación de recordatorios: se podría ofrecer recordatorios basados en la ubicación del usuario. Es posible debido al bajo consumo energético del framework.

Como se ha visto, las líneas futuras y las posibilidades son amplias y variadas, existiendo otras muchas, debido al gran éxito de los smartphones Android.

3.3. Valoración personal

La valoración personal de este trabajo ha sido positiva, ya que se han utilizado tecnologías actuales, cada vez más en uso, que gracias al TFG he podido aprender.

Por un lado, la plataforma Android, utilizada cada vez más a nivel mundial y que sigue creciendo, por lo que es un objetivo interesante para el desarrollo de aplicaciones por el mercado que abarca. Además, mi experiencia con Android era mínima, y después del trabajo he profundizado y aprendido en ella, siendo ahora capaz de desarrollar aplicaciones complejas.

Por otro lado, el contribuir en un proyecto más amplio, que está siendo desarrollado por IAAA, GeoSpatiumLab y HOWLab, me ha aportado una motivación extra para la realización del proyecto. Aunque ha podido haber bloqueos en el desarrollo, debido a que el servidor y los nodos se han ido programando en paralelo. Sin embargo, dichos problemas, se han solventado bien gracias realizar el trabajo en los laboratorios de la Universidad, donde podía comunicar mis problemas con los demás desarrolladores en tiempo real.

Me ha servido de gran apoyo contar con la ayuda de los integrantes del IAAA y GeoSpatiumLab, ya que me han resuelto dudas y problemas durante todo el proyecto.

Otro aspecto nuevo para mí ha sido la implementación de un protocolo acordado entre varias partes. Al utilizar todos, el mismo protocolo (que especifica el formato de datos, las operaciones, interfaces…) se ha podido integrar las partes del sistema de tracking con menor complejidad. Aunque, al querer extender el protocolo, añadiendo operaciones de lectura de sensores, el proceso ha sido más lento al tener que aceptarlo todas las partes.

Personalmente, considero que este trabajo ha supuesto una aportación muy positiva a mi formación, por los temas tratados y las tecnologías utilizadas. He puesto en práctica lo que me han enseñado los profesores durante todo el grado. He aprendido a realizar un proyecto desde cero, más amplio que los de clase. También he aprendido a sincronizarme y trabajar en paralelo con las otras partes implicadas en el sistema de tracking, y todo esto en un entorno real. Por último he conocido como es implementar un framework el cual tiene que cumplir unas reglas fijadas por un protocolo, porque, en caso contrario no se podría integrar en el sistema global.

En general, estoy satisfecho con lo aprendido durante el desarrollo y creo que podré darle uso en el futuro.

Anexos

A) Análisis de requisitos

Anteriormente, se ha visto en la sección 2.2, un análisis de requisitos del sistema. En este anexo se detallan más los requisitos y stakeholders.

A1) Actores / Stakeholders

- **Usuario:** usuario que utiliza el framework es su dispositivo a través de una aplicación de caso de uso.
- **App caso de uso:** es la aplicación caso de uso desarrollada en este TFG que utiliza los frameworks desarrollados.
- **Servidor:** servidor del sistema de tracking que interactúa con los frameworks a través de conexiones 3G (HTTP) y GSM (SMS).

A2) Requisitos funcionales formalizados

Requisitos funcionales Framework Protocolo

[RFFP-1] El sistema deberá permitir
(A) <enviar SMS>
(B) <recibir SMS>
(C) <almacenar SMS> siempre que <se hayan intentado enviar> y <no hayan podido ser enviados>
(D) <enviar SMS> que <habían sido almacenados previamente>

[RFFP-2] El sistema deberá ser capaz de <ejecutar las operaciones especificadas en el protocolo para nodos del sistema de tracking> en <entornos online> y <entornos offline>.

[RFFP-3] El sistema deberá permitir
(A) <enviar peticiones HTTP>
(B) <recibir peticiones PUSH>

[RFFP-4] El sistema deberá permitir <configurar los parámetros del protocolo>

[RFFP-5] El sistema deberá permitir
(A) <iniciar la ejecución del protocolo>
(B) <detener la ejecución del protocolo>

Requisitos funcionales Framework Sensores

[RFFS-1] El sistema deberá ser capaz de <leer los valores de los sensores disponibles en el dispositivo>

[RFFS-2] La aplicación deberá permitir <obtener la localización del dispositivo> mediante <GPS> en <intervalos de tiempo fijados como parámetro>

Requisitos funcionales Aplicación caso de uso

[RFACU-1]: La aplicación deberá ser capaz de <iniciar> y <finalizar> la >ejecución del protocolo>.

[RFACU-2]: La aplicación deberá ser capaz de <mostrar> <las respuestas a las operaciones del protocolo> al <usuario>.

[RFACU-3]: La aplicación deberá ser capaz de <permitir> que <el usuario> <configure los parámetros de ejecución del protocolo>.

[RFACU-4]: La aplicación deberá ser capaz de <mostrar estadísticas de peticiones y respuestas de operaciones por SMS y por HTTP>.

[RFACU-5]: La aplicación deberá ser capaz de <mostrar y refrescar> <los últimos valores leídos de los sensores del dispositivo incluyendo GPS>.

[RFACU-6]: La aplicación deberá ser capaz de <mostrar y refrescar> <un mapa de Zaragoza>, incluyendo <las localizaciones obtenidas del dispositivo durante su funcionamiento en el sistema de tracking> y <una ruta seguida por el dispositivo>.

[RFACU-7]: La aplicación deberá ser capaz de <mostrar y refrescar> <gráficos interactivos> con <los valores leídos de los sensores> del <dispositivo>.

Requisitos no funcionales

[RNF-1]: El sistema debe <tener acceso> a <los permisos del dispositivo: enviar y recibir SMS, internet, acceder a la localización, leer el estado del teléfono y de la red, obtener cuentas de usuario, impedir que el teléfono entre en modo inactivo>.

Requisitos no funcionales Framework Protocolo

[RNFFP-1]: El framework deberá <implementar> <el hash de seguridad especificado en el protocolo>.

[RNFFP-2]: El framework deberá <permitir> <ejecutarse en varios modos de energía>.

RNFFP-3: El dispositivo que <ejecute> <el framework> deberá <tener> <sistema operativo Android 2.1 o superior>.

RNFFP-4: El dispositivo que <ejecute> <el framework> deberá <tener> <instalados los Google Play Services>, para <permitir> <el funcionamiento de las notificaciones PUSH>.

RNFFP-5: El framework debe <almacenar estadísticas> de <SMS enviados> y <recibidos>, así como <comandos HTTP enviados y recibidos>.

RNFFP-6: El framework debe <poder acceder> al <dispositivo> con <permisos de súper-usuario o root>.

Requisitos no funcionales Framework Sensores

RNFFS-1: El dispositivo que <ejecute> el <framework> deberá <disponer> de <GPS>.

RNFFS-2: El dispositivo que <ejecute> el <framework> deberá <tener> <sistema operativo Android 2.1 o superior>.

Requisitos no funcionales Aplicación Caso de Uso

RNFACU-1: El dispositivo que <ejecute> <la aplicación caso de uso> deberá <tener> <sistema operativo Android 4.0 o superior>.

B) Casos de uso

En este apartado se muestran dos diagramas de caso de uso; uno para los frameworks desarrollados y otro para la aplicación caso de uso. Posteriormente, se detallan casos de uso extendidos.

B1) Diagramas de caso de uso

Seguidamente, se detallan y muestran el diagrama de casos de uso de Frameworks y el de la aplicación caso de uso.

En el **diagrama de casos de uso de Frameworks** aparecen dos actores, la App caso de uso y el Servidor.

La App caso de uso, es la aplicación caso de uso que importa los frameworks como bibliotecas Android y aprovecha la funcionalidad de éstos. Las principales acciones que tiene este actor son iniciar y detener el protocolo (es decir, que el dispositivo forme parte del sistema de tracking o no), configurar el protocolo (mediante una pantalla de ajustes) y ver estadísticas de peticiones, además de leer los valores de los sensores y GPS. Toda la lógica del protocolo (ejecución de operaciones y eventos, así como tareas de control de energía, etc.) no se incluyen como un caso de uso para este actor, ya que una vez iniciado el protocolo, la biblioteca framework protocolo se encarga de todo.

El servidor es el actor que representa al servidor central del sistema de tracking. El cual tiene como caso de uso el ejecutar operaciones (mediante SMS o HTTP). Para ejecutar operaciones, se utilizan los casos de uso de recibir y enviar SMS y HTTP, así como leer sensores y obtener localización. El servidor, a su vez, es capaz de configurar el protocolo mediante la ejecución de operaciones de configuración.

El diagrama de casos de uso explicado en los dos párrafos anteriores, se puede ver en la Figura 12. *Diagrama casos de uso Frameworks*

Por último, en el **diagrama de casos de uso de la App caso de uso**, aparece un actor usuario, que es el usuario de la aplicación en su dispositivo Android. Dicho actor puede iniciar y finalizar la ejecución del protocolo, mostrar por pantalla las respuestas generadas al procesar una operaciones a modo de ejemplo, configurar el protocolo directamente desde la aplicación, mostrar estadísticas de uso de la red por el protocolo (SMS y comandos HTTP enviados y recibidos), mostrar y refrescar los últimos valores de localización y de los sensores, mostrar y refrescar un mapa con la ruta seguida por el dispositivo, y finalmente, mostrar y refrescar gráficos en el tiempo de cada sensor con sus valores leídos.

El diagrama de casos de uso, anteriormente descrito, se puede ver en la Figura 13.

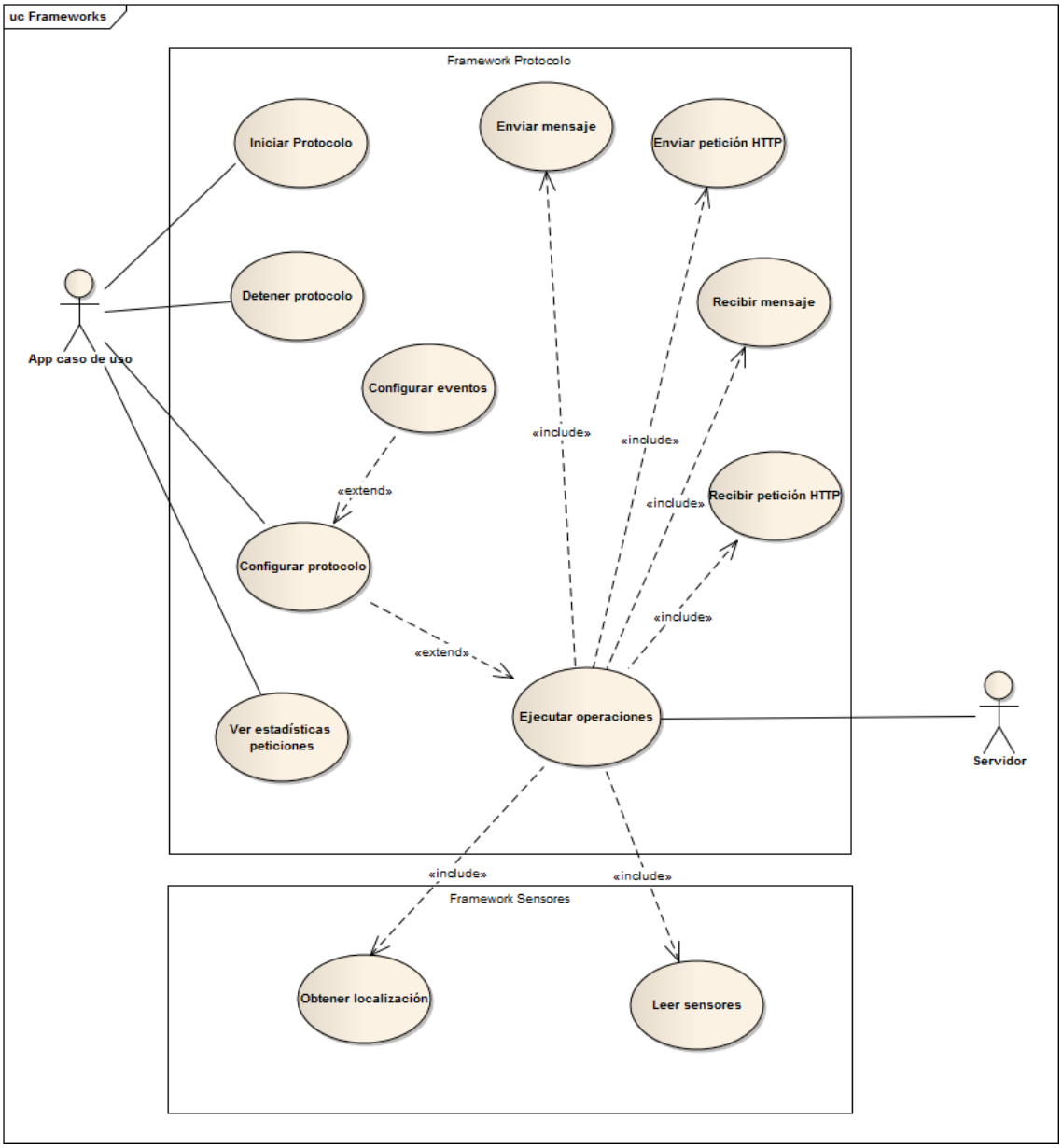


Figura 12. *Diagrama casos de uso Frameworks*

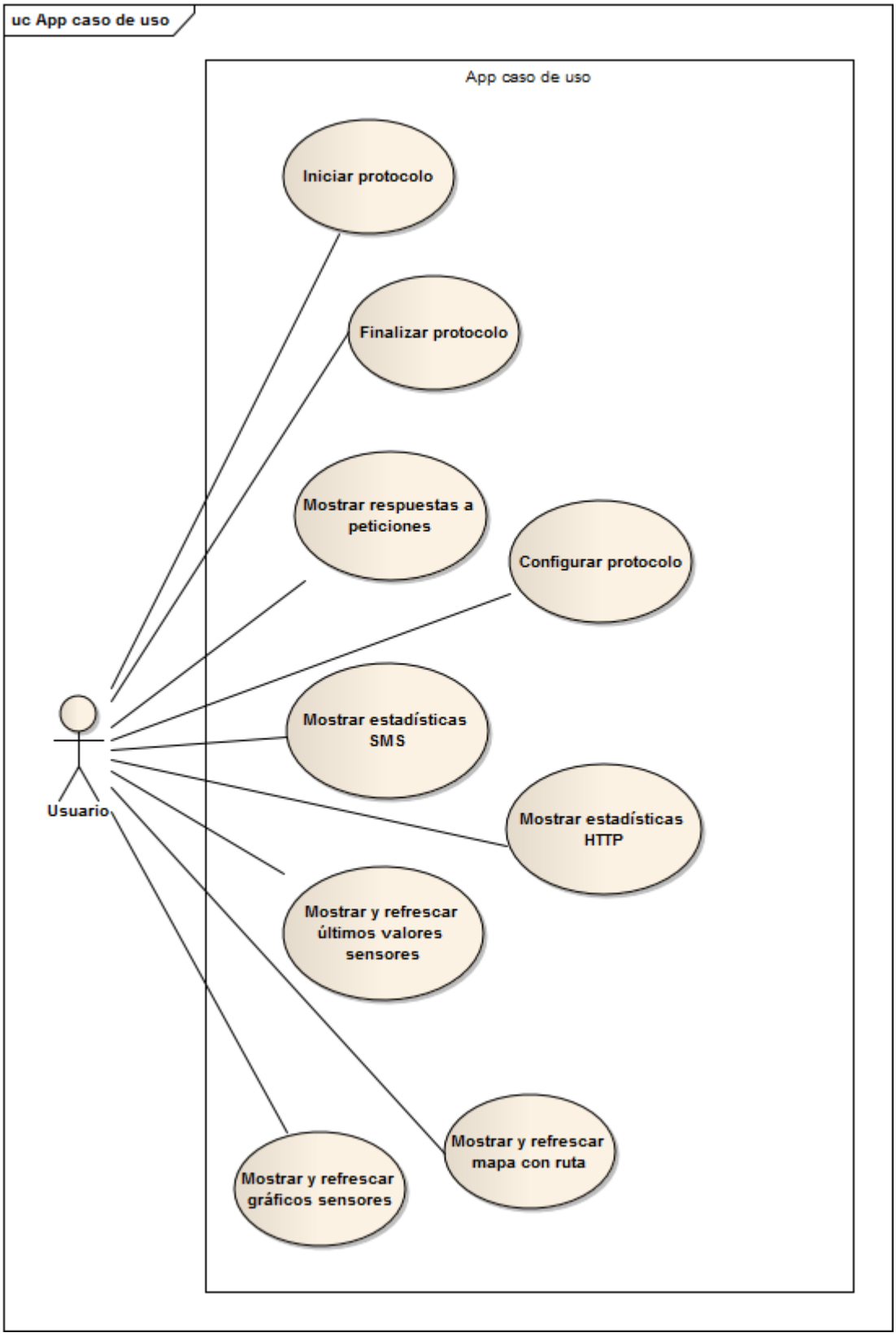


Figura 13. *Diagrama casos de uso de la App caso de uso*

C) Documentación de la arquitectura

En este anexo, se va documentar la arquitectura del sistema mediante varias vistas. Primero se explicará cómo se documenta cada vista y se dará una explicación del funcionamiento general del sistema. Por último se detallará cada vista arquitectural.

C1) Cómo se documentan las vistas

El estándar para la documentación de una vista es el siguiente:

- **Presentación primaria**: diagrama asociado a la vista.
- **Catálogo de la vista**
  - **Elementos y sus propiedades**: definición de los elementos que aparecen en el diagrama y sus propiedades.
  - **Relaciones y sus propiedades:** definición de las relaciones entre los elementos que aparecen en el diagrama y las propiedades.
  - **Interfaces de los elementos:** definición de las interfaces que ofrece cada elemento de la vista y sus propiedades.
- **Diagrama de contexto:** diagrama que define la frontera del sistema y su entorno.
- **Guía de variabilidad:** es una guía que contiene los puntos de variación de una vista, es decir, elementos específicos de flexibilidad.
- **Exposición de razones:** argumentación de las decisiones arquitecturales de la vista.

C2) Visión general del sistema

Se trata de un sistema de tracking capaz de operar en entornos online y offline. El sistema está compuesto por un servidor central, un cliente y de 1 a n nodos (ver Figura 1). El cliente se conecta con el servidor a través de internet y puede invocar operaciones sobre él. Esas operaciones van a estar relacionadas con los nodos. El servidor se conecta con los nodos y viceversa. El servidor invoca operaciones de los nodos. Éstos le pueden enviar eventos al servidor, por ejemplo eventos de localización, batería baja, etc.

En una aplicación final, el cliente podrá trackear los nodos a través del servidor, es decir, el servidor le informará al cliente de la situación de los nodos. El servidor es un intermediario entre el cliente y los nodos.

Sin embargo, en el contexto de este TFG el cliente no es relevante en el sistema, puesto que los nodos desconocen la existencia de éste. El TFG se centra en la implementación del protocolo en sistemas Android, de manera que cualquier dispositivo Android se convierta en un nodo y se pueda integrar en el sistema de tracking.

C3) Vista de componente y conector

C3.1) Presentación primaria

Se pueden diferenciar dos claros componentes, el framework Protocolo y el framework Sensores:

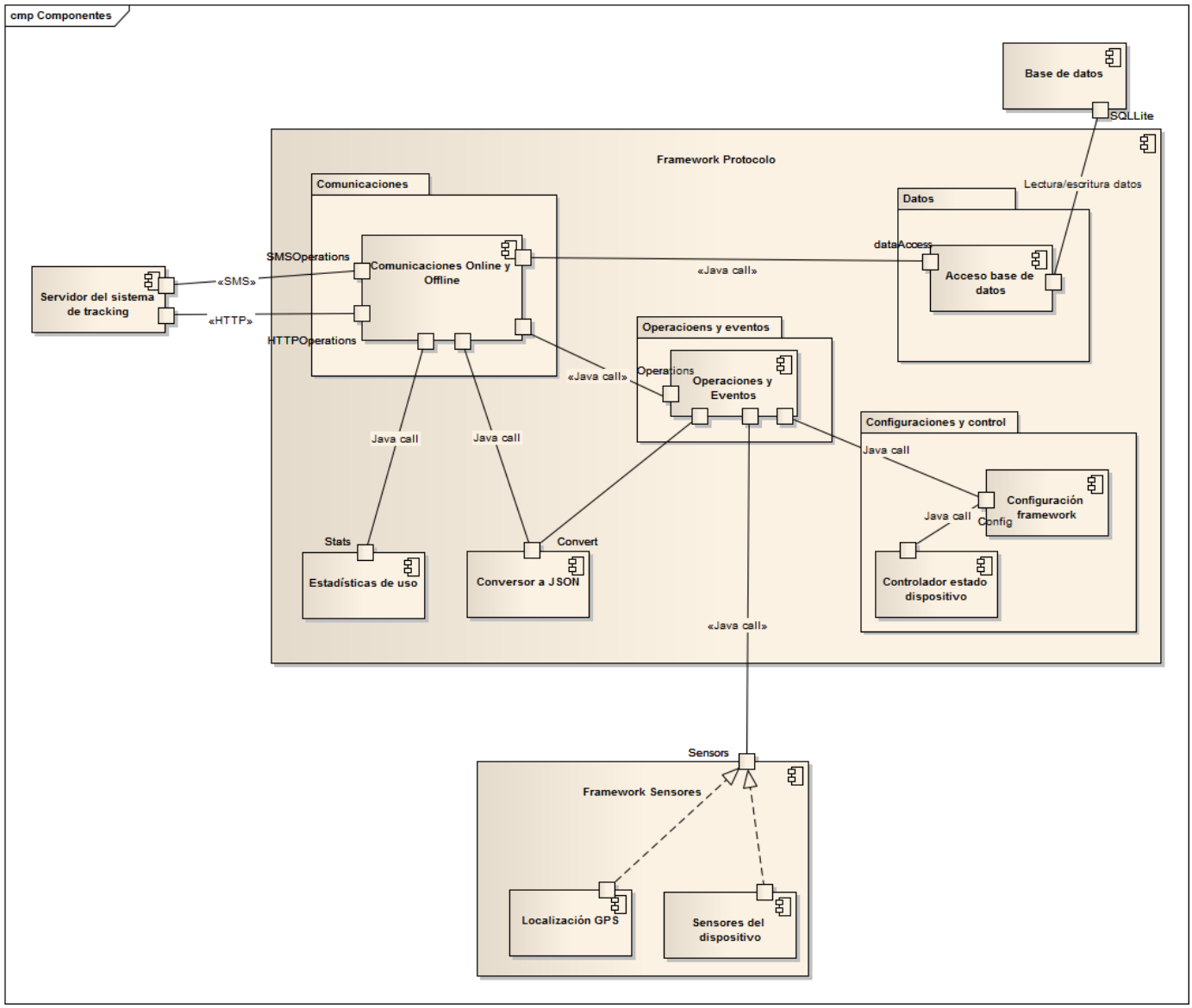


Figura 14. *Presentación primaria de la vista componente y conector*

C3.2) Catálogo de la vista

Elementos y sus propiedades
El framework de sensores, hay dos componentes, Localización GPS y Sensores del dispositivo:

- **Localización GPS:** es el encargado de obtener la localización del dispositivo mediante su GPS, además debe permitir elegir un tiempo de refresco de dicha localización para poder ahorrar energía.
- **Sensores del dispositivo:** es el componente encargado de leer los sensores disponibles en el dispositivo.

En el framework protocolo, que es el encargado de implementar las comunicaciones y operaciones del protocolo, podemos diferenciar 7 componentes claros:

- **Comunicaciones online y offline:** es el componente encargado de comunicarse con el servidor del sistema de tracking para recibir sus órdenes y enviarle resultados y eventos. Este componente implementa los dos tipos de comunicaciones, tanto la comunicación entornos online (HTTP) y en entornos offline (SMS).
- **Operaciones y eventos:** este componente es el encargado de ejecutar las operaciones definidas por el protocolo para un nodo, así como los eventos que se envían regularmente al servidor.
- **Estadísticas de uso:** registra el número de mensajes enviados y recibidos, tanto por SMS como por HTTP.
- **Conversor JSON:** componente encargado de codificar/decodificar JSON, para facilitar la comunicación con el servidor acorde a los formatos de datos descritos por el protocolo.
- **Acceso base de datos:** este componente se encarga de facilitar la inserción y borrado de datos utilizados por el framework en la base de datos SQLite.
- **Configuración framework:** su función es la de almacenar y dar acceso a los parámetros de configuración del framework, como por ejemplo el modo de energía, o la dirección IP del servidor.
- **Controlador de estado del dispositivo:** su función principal es controlar el estado de las redes del dispositivo (conectarlas, desconectarlas, saber si se dispone de conexión 3G…).

Relaciones y sus propiedades

Los conectores “Java call” que se pueden observar en la presentación primaria, se refieren a llamadas entre objetos Java.

El conector etiquetado como “Lectura/Escritura datos”, es el ofrecido por Android para trabajar con bases de datos SQL Lite.

Los conectores que se pueden ver dentro del componente framework Sensores son delegaciones de interfaz.

El conector etiquetado como “HTTP” hace referencia a conexiones HTTP.

El conector etiquetado como “SMS” hace referencia a la comunicación mediante SMS.

Interfaces de los elementos

Se muestran una tabla por cada componente con los métodos de sus interfaces externas

Componente Operaciones y Eventos
public String deviceGetHTTPConfig (JSONObject jsonObject, Context context) *Ejecuta la operación de DeviceGetHTTPConfig* **Parameters:** *context - Contexto de la aplicación* *jsonObject - Petición del servidor* **Returns:** *Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye información de la configuración de comunicaciones HTTP del dispositivo.*
public String deviceGetInfo (JSONObject jsonObject, Context context) *Ejecuta la operación de DeviceGetInfo* **Parameters:** *jsonObject – Petición del servidor* *context - Contexto de la aplicación* **Returns:** *Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye la ID y versión del dispositivo.*
public String deviceGetMode (JSONObject jsonObject, Context context) *Ejecuta la operación de DeviceGetMode* **Parameters:** *jsonObject - Petición del servidor* *context - Contexto de la aplicación* **Returns:** *Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye el modo de energía en el cual se está ejecutando el dispositivo.*
public String deviceGetSMSCOnfig(JSONObject jsonObject, Context context) *Ejecuta la operación de DeviceGetSMSConfig* **Parameters:** *context - Contexto de la aplicación* *jsonObject - Petición del servidor* **Returns:** *Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye información de la configuración de comunicaciones SMS del dispositivo*

public String devicePing(JSONObject jsonObject, Context context)

Ejecuta la operación de DevicePing

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK confirmando que el dispositivo está vivo.

public String deviceGetHTTPConfig (JSONObject jsonObject, Context context)

Ejecuta la operación de DeviceReset, que resetea las configuraciones del dispositivo a las por defecto

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK confirmando que la operación se ha ejecutado

public String deviceGetHTTPConfig (JSONObject jsonObject, Context context)

Ejecuta la operación de DeviceSetHTTPConfig, que cambia las configuraciones de comunicación HTTP del dispositivo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK confirmando que la operación se ha ejecutado.

public String deviceSetMode (JSONObject jsonObject, Context context)

Ejecuta la operación de DeviceSetMode, que cambia el modo de energía del dispositivo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye

un ACK confirmando que la operación se ha ejecutado.

public String deviceSetSMSConfig (JSONObject jsonObject, Context context)

Ejecuta la operación de DeviceSetSMSConfig, que cambia las configuraciones de comunicación SMS del dispositivo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK confirmando que la operación se ha ejecutado.

public String locationGet (JSONObject jsonObject, Context context)

Ejecuta la operación de LocationGet

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye la localización del dispositivo.

public String locationGetInfo (JSONObject jsonObject, Context context)

Ejecuta la operación de LocationGetInfo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye los parámetros de configuración de localización del dispositivo

public String locationGetRefreshRate (JSONObject jsonObject, Context context)

Ejecuta la operación de LocationGetRefreshRate

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye la tasa de refresco de la localización

public String locationSet (JSONObject jsonObject, Context context)

Ejecuta la operación de LocationSet, que fija la localización del dispositivo.

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK para que el servidor sepa que se ha ejecutado la operación.

public String locationSetRefreshRate (JSONObject jsonObject, Context context)

Ejecuta la operación de LocationSetRefreshRate, que fija la tasa de refresco de la localización del dispositivo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye un ACK para que el servidor sepa que se ha ejecutado la operación.

public String powerGetInfo (JSONObject jsonObject, Context context)

Ejecuta la operación de PowerGetInfo

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye la información de energía y batería del dispositivo.

public String powerGetLevel (JSONObject jsonObject, Context context)

Ejecuta la operación de PowerGetLevel

Parameters:

context - Contexto de la aplicación
jsonObject - Petición del servidor

Returns:

Devuelve en formato JSON la respuesta para ser enviada al servidor, que incluye el nivel de batería del dispositivo.

Componente Conversor a JSON

public JSONObject convertToJSONObject(String command)

Transforma un String JSON a un objeto JSON. En caso de que el parámetro command no sea un string JSON válido, devuelve null.

Parameters:

command - String JSON a convertir

Returns:

El JSONObject convertido, o null si el parámetro command no es un String con formato JSON válido

public String getCommandName(JSONObject obj)

Método que recibe un objeto JSON correspondiente a una petición de una operación del protocolo SMS y devuelve el nombre de la operación, correspondiente a la clave CMD del JSONObject.

Parameters:

obj - Objeto JSON que representa una petición de operación

Returns:

Devuelve el nombre de la operación que se invoca dentro el objeto JSON.

public JSONObject httpRequestToJSON(String httpRequest)

Método que devuelve un objeto JSON que representa la petición de una operación Esta cadena es obtenida a partir de una petición de operación en formato HTTP

Parameters:

httpRequest - Petición de operación codificada en HTTP (formato del protocolo)

Returns:

Petición de operación codificada en un objeto JSON.

public boolean isJSONValid(String jsonString)

Método que recibe una cadena y validad si está correctamente formada en JSON

Parameters:

jsonString - Cadena JSON que se quiere validar

Returns:

Verdadero si es un String valido y falso en caso contrario.

public String operationToJSONString(Object obj)

Transforma un objeto Java a un String JSON

Parameters:

obj - Objeto Java para transformar a String JSON

Returns:

Devuelve el String JSON correspondiente al objeto (sus atributos no nulos y sus valores se codifican en JSON).

Componente Estadísticas de uso

public int[] getStats()

Devuelve un vector de enteros con las estadísticas de uso de la aplicación hasta el momento. El formato del array es: [sms enviados, sms recibidos, comandos http enviados, comandos http recibidos]

Returns:

Vector de enteros con las estadística de uso de la aplicación hasta el momento.

public void receivedHttp()

Incrementa las estadísticas por una unidad de comandos http recibidos

public void receivedMessage()

Incrementa las estadísticas por una unidad de comandos mensajes recibidos

public void resetStats()

Resetea a 0 los contadores de estadísticas

public void sentHttp()

Incrementa las estadísticas por una unidad de comandos http enviados

public void sentMessage()

Incrementa las estadísticas por una unidad de mensajes enviados

Componente Acceso base de datos
public void close() *Cierra la conexión con la base de datos*
public boolean deleteAllMessage() *Borra el mensaje con Id == rowId de la base de datos* **Returns:** *False si no se ha borrado ningún mensaje, true en caso contrario.*
public boolean deleteMessage(long rowId) *Borra el mensaje con Id == rowId de la base de datos* **Parameters:** *rowId - Id del mensaje a borrar* **Returns:** *True si se ha borrado, false en caso contrario.*
public boolean deleteSentMessage() *Borra todos los mensajes en la base de datos que tengan el flag isSent a true, es decir los mensajes que ya han sido enviados.* **Returns:** *False si no se ha borrado ningún mensaje, true en caso contrario.*
public android.database.Cursor getAllMessages() *Recupera todos los mensajes almacenados en la base de datos* **Returns:** *Un objeto Cursor con todos los mensajes almacenados en la base de datos*
public android.database.Cursor getLastValue() throws android.database.SQLException *Devuelve el último mensaje almacenado en la base de datos*

Returns:

Cursor apuntando al último mensaje insertado en la base de datos

Throws:

android.database.SQLException – Si no hay mensajes

public android.database.Cursor getUnsentMessages()

Devuelve todos los mensajes almacenados que no han sido enviados, es decir que su flag sent es false

Returns:

Mensajes almacenados que no han sido enviados.

public long insertMessage(String phoneNumber, String body, boolean sent)

Almacena un mensaje en la base de datos

Parameters:

phoneNumber - Número de teléfono
del mensaje
body - Cuerpo del mensaje
sent - Flag que indica si el mensaje se ha enviado (true si se ha enviado)

Returns:

El ID del mensaje insertado

public DBMessageAdapter open() throws android.database.SQLException

Abre la base de datos

Returns:

Una instancia de la clase DBMessageAdapter para invocar a sus métodos.

Throws:

android.database.SQLException - Ha ocurrido un problema al abrir la base de datos

public boolean updateMessage(int rowId, boolean isSent)

Actualiza el valor del flag que indica si el mensaje ha sido enviado del mensaje con id == rowId

Parameters:

rowId - Id del mensaje a actualizar

isSent - Valor del flag a actualizar

Returns:

True si se ha actualizado, false en caso contrario.

Componente Configuración framework

public static int getCustomRefreshRate(android.content.Context context)

Devuelve el locationRefreshRate fijado por el usuario en las configuraciones. Sólo debe ser usado si el modo de energía actual es el 5 (Custom Mode)

Parameters:

context - Contexto de la aplicación

Returns:

locationRefreshRate en segundos (tasa de refresco de la localización)

public static int getLocationEventRefreshRate(android.content.Context context)

Devuelve la tasa de refresco de la localización (LocationRefreshRate), dependiendo del modo de energía. En caso de que, el modo de energía sea Custom Mode (código 5), se busca la tasa introducida por el usuario.

Parameters:

context - Contexto de la aplicación

Returns:

locationRefreshRate en segundos (tasa de refresco de la localización)

public static int getPowerMode(android.content.Context context)

Devuelve el modo de ejecución almacenado en las preferencias de la aplicación

Parameters:

context - Contexto de la aplicación

Returns:

El id del modo ID's del modo:

0 - The device is waiting to be configured

1 - Extreme Loew Power Mode

2 - Very Low Power Mode

3 - Low Power Mode

4 - Always Locate mode

5 – Custom

public static int getSmsPollTime(android.content.Context context)

Devuelve, en segundos, el periodo de escucha de SMS. Sólo utilizar en power mode 5 (custom)

Parameters:

context - Contexto de la aplicación

Returns:

Periodo de escucha de SMS, en segundos

public static int getSmsTransmitPeriod(android.content.Context context)

Devuelve, en segundos, el periodo de transmisión de SMS Sólo utilizar en power mode 5 (custom)

Parameters:

context - Contexto de la aplicación

Returns:

Periodo de transmisión de SMS en segundos

public static void setPowerMode(android.content.Context context, int powerMode)

Fija el modo actual de ejecución a [poderMode], guardándolo en las preferencias. Si el valor no está entre 0 y 5 (incluidos) no se hace nada.

Parameters:

context - Contexto de la aplicación

powerMode - Valor del 0 al 5 (incluidos) que indican el modo a fijar.

Returns:

powerMode - Valor del 0 al 5 (incluidos) que indican el modo a fijar.

public static void setSmsPollTime(int value, android.content.Context context)

Fija en las preferencias, el valor de pref_smsPollTimeType a [value], que es el intervalo en el cual se mira si han entrado nuevos mensajes SMS.

Parameters:

context - Contexto de la aplicación

value - Valor en segundos que indican el intervalo a fijar.

Returns:

powerMode - Valor del 0 al 5 (incluidos) que indican el modo a fijar.

Componente Base de datos

La interfaz que ofrece este componente es que da Android en su documentación oficial.

http://developer.android.com/reference/android/database/sqlite/package-summary.html

Componente Localización GPS

public void changeRefreshRate(int refreshRate)

Cambia la tasa de refresco para obtener la localización.(se ha tenido que llamar a startLocationListening antes de utilizar este método).

Parameters:

refreshRate - Tasa de refresco para obtener la localización

public android.location.Location getLocation()

Devuelve la última localización obtenida del GPS del dispositivo

Returns:

Última localización obtenida del GPS del dispositivo o null si no la hay

public void startLocationListening(int refreshRate)

Construye el objeto LocationReader que empieza a escuchar la localización del dispositivo a través del GPS.

Parameters:

refreshRate - Tasa de refresco para obtener la localización

public void stopLocationListening()

Para de escuchar las actualizaciones de la localización del dispositivo.

Componente Sensores del dispositivo

public void setSensorListening(int interval)

Lanza o actualiza la lectura de los sensores del sensorReader cada [interval] segundos

Parameters:

interval - Intervalo de actualización de la lectura de los sensores en segundos.

public void stopSensorListening()

Para la lectura periódica de sensores

public SensorValue getLight()

Devuelve un objeto SensorValue con el último valor leído del sensor de luz.

Returns:

Última valor leído del sensor de luz

public SensorValue getPressure()

Devuelve un objeto SensorValue con el último valor leído del barómetro.

Returns:

Última valor leído del barómetro

public SensorValue getX() (uno para cada tipo de sensor que se quiera leer, disponible en Android)

Devuelve un objeto SensorValue con el último valor leído del del sensor X.

Returns:

Última valor leído del sensor de X

Otros componentes como Controlador Estado Dispositivo, no ofrecen ninguna interfaz ya que son Servicios independientes que corren en otro hilo de ejecución.

C3.3) Diagrama de contexto

Se muestra en la Figura 15 un diagrama los frameworks en su contexto de utilización:

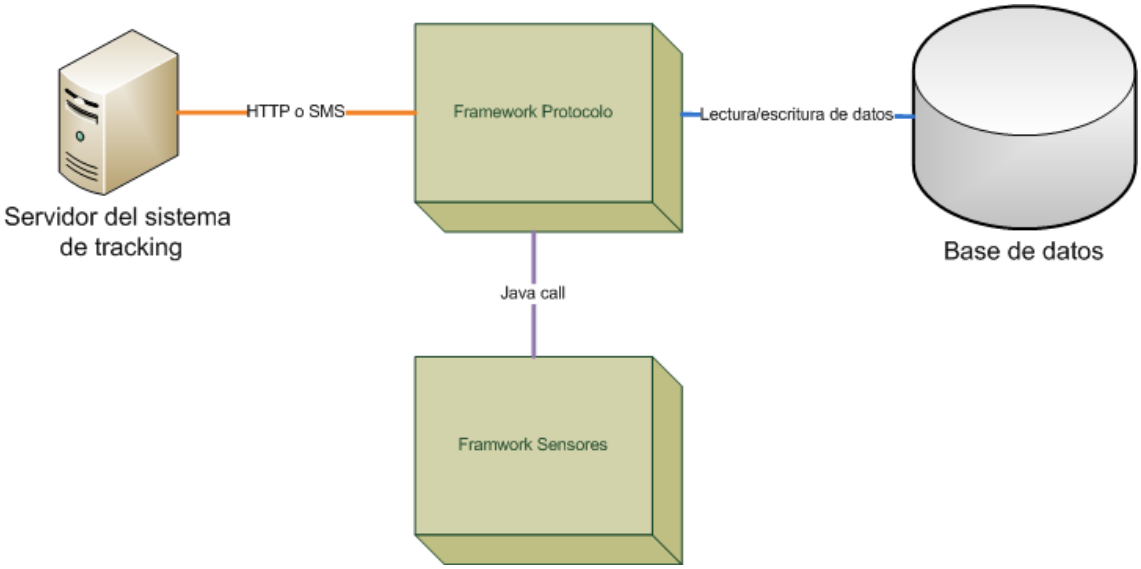


Figura 15. *Diagrama de contexto de la vista componente y conector*

C3.4) Guía de variabilidad

Se podría añadir nueva funcionalidad al componente sensores del dispositivo, para que pudiera leer sensores externos, conectados al dispositivo Android mediante Bluetooth.

C3.5) Exposición de razones

Aquí se exponen algunas razones del diseño arquitectural de la vista de Componentes y Conectores en las cuales la arquitectura ha podido cambiar a lo largo del TFG.

1. **Decisión separar en dos componentes la lógica del protocolo con la lectura de sensores**

    a. **Asunto**: Se separa en dos componentes la lógica del protocolo (Framework protocolo) y lectura de sensores (Framework sensores).
    b. **Decisión y estado**: Aceptado
    c. **Etiquetas**: frameworks
    d. **Alternativas**: Que las dos estén en el mismo componente.
    e. **Argumento**: Se toma la decisión de separarlo es dos componentes porque son dos responsabilidades diferentes en las cuales existe poco acoplamiento. Además el framework de lectura de sensores podrá ser reutilizado por otras aplicaciones Android.
    f. **Implicaciones**: Se modificó el diagrama de componentes. Se crean dos librerías Android separadas.

2. **Modelizar SMS y HTTP como componentes y no como conectores**

    a. **Asunto**: Se modelizan SMS y HTTP como componentes y no como conectores.
    b. **Decisión y estado:** Aceptado
    c. **Etiquetas:** conectores
    d. **Alternativas**: Que sean conectores entre el Servidor y el Framework Protocolo.
    e. **Argumento**: Se ha decidido hacerlo así porque tienen una responsabilidad más que compleja que sólo conectar componentes.
    f. **Implicaciones**: Se modificó el diagrama de componentes.

3. **Borrar base de datos**

    a. **Asunto**: Se elimina la base de datos
    b. **Decisión y estado**: Rechazado
    c. **Etiquetas**: base de datos
    d. **Alternativas**: No tener base de datos, almacenar en ficheros.
    e. **Argumento**: Una vez se empezó a programar se vio que las preferencias o configuraciones del framework no se necesitaban almacenar en base de datos, porque Android ofrece un protocolo para almacenar este tipo de preferencias en ficheros. Por lo tanto se pretendía borrar la base de datos. Sin embargo, más adelante se vio que era necesaria porque se deben almacenar SMS que no han podido ser enviados (para intentar volverlos a enviar cuando sea posible).
    f. **Implicaciones**: Se modificó el diagrama de componentes. Se borró la base de datos y las clases que accedían a ella, sin embargo luego se volvieron a crear. Se implementó el mecanismo Android para guardar preferencias.

C4) Vista de módulos

C4.1) Presentación primaria

La presentación primaria se muestra en dos imágenes separadas: en una el framework Protocolo y en otra el framework Sensores.

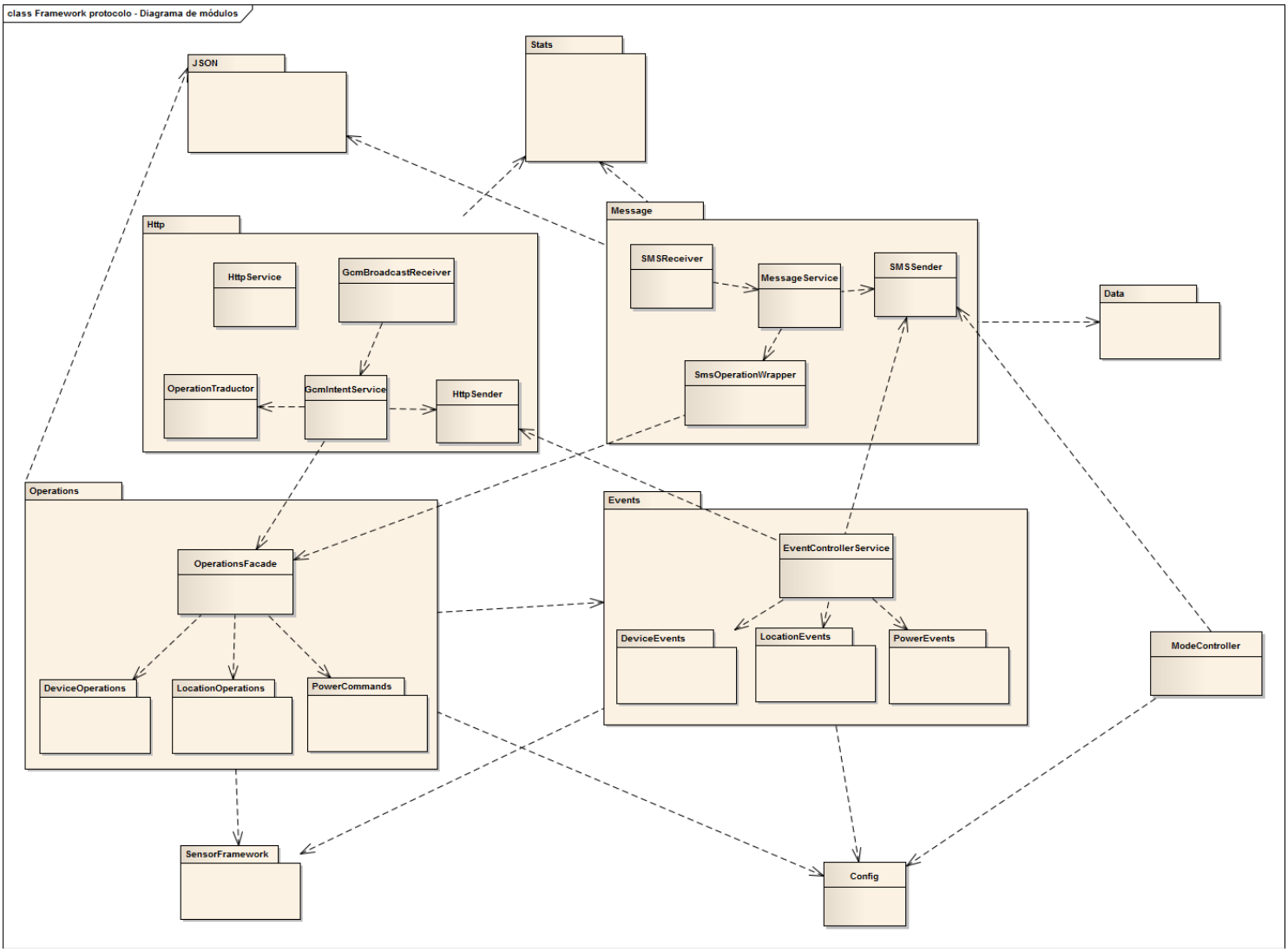


Figura 16. *Presentación primaria de la vista de módulos. Framework Protocolo*

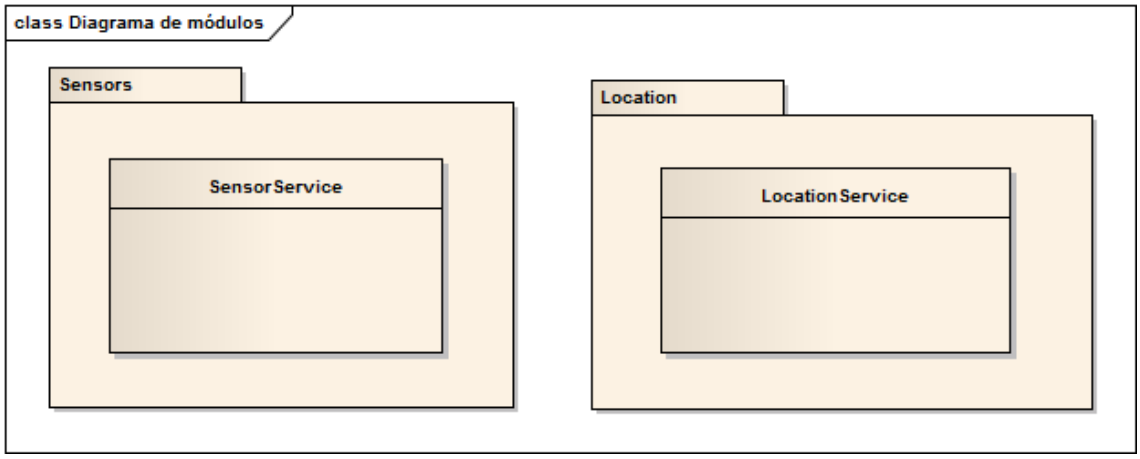


Figura 17. *Presentación primaria de la vista de módulos. Framework sensores*

C4.2) Catálogo de la vista

Elementos y sus propiedades

Framework protocolo

- **Message:** es el módulo encargado de recibir, enviar y leer mensajes. Una vez que le ha llegado un mensaje debe analizarlo para saber qué operación del módulo "Operations" se debe ejecutar. Tiene 4 clases:
  - **SMSReceiver:** extiende la clase BroadcastReceiver, para recibir los SMS. Una vez recibido el SMS, se lo pasa a la clase MessageService, mediante un Intent de Android. Es decir, esta clase es un filtro, consume un evento, lo procesa y genera otro evento.
  - **SMSSender:** es la clase encargada de enviar mensajes SMS.
  - **MessageService:** Obtiene los mensajes recibidos de la clase SMSReceiver e invoca a la clase SmsOperationWrapper, pasándole como argumento el SMS recibido. La clase SmsOperationWrapper le devolverá el resultado de la ejecución de la operación correspondiente al SMS, entonces, MessageService, enviará el resultado al servidor a través de la clase SMSSender.
  - **SmsOperationWrapper:** esta clase, a través del SMS recibido que le pasan como argumento en su constructor, es capaz de invocar a la operación correspondiente del módulo "Operations" y devolver el resultado de dicha operación a la clase MessageService.
- **JSON:** este módulo ofrece operaciones para transformar objetos/cadenas a objetos JSON y viceversa.
- **Stats:** es un módulo encargado de recoger las estadísticas de uso de la aplicación.
- **Context:** módulo utilizado para almacenar los parámetros de configuración del framework. Estos parámetros son utilizados en el protocolo.
- **Data:** este módulo es el encargado de ofrecer operaciones de acceso a la base de datos.

- **Operations:** este módulo implementa las operaciones propias de un nodo, tal como especifica el protocolo. Tiene una clase que actúa como fachada e invoca a las operaciones. Cada operación es una clase, cada clase tiene un método *execute* que realiza su operación y devuelve un String codificado en JSON con el resultado.
- **Events:** este es el módulo que implementa y controla los eventos de un nodo especificados en el protocolo. Tiene una clase EventController que es un servicio Android que ofrece operaciones para invocar a los eventos, y que controla ejecución periódica de eventos. Cada evento está representado por una clase separada.
- **ModeController:** este módulo es el encargado conectar y desconectar la red móvil periódicamente dependiendo de la configuración de energía almacenada en la clase SharedPreferences, obtenida en el módulo Context.

Framework sensores

- **SensorFramework:** este módulo ofrece operaciones de lectura de sensores y localización. Está compuesto por dos módulos:
    - **Sensors**: es el módulo encargado de leer los sensores del dispositivo.
    - **Location**: es el módulo encargado de obtener la localización del dispositivo.

Relaciones y sus propiedades

- Los módulos Message y Operations utilizan el módulo JSON para convertir cadenas a objetos JSON y viceversa.
- Los módulos Message y HTTP utiliza los módulos Operations para invocar a la operación evento correspondiente.
- Operations utiliza Events para lanzar un evento de error cuando alguna operación ha fallado.
- Events utiliza el módulo Message y HTTP para enviar los eventos al servidor.
- Message y HTTP usan Stats para almacenar estadísticas.
- Message utiliza Data almacenar mensajes que no se han podido enviar en la base de datos.
- Operations y Events utilizan la clase Context para obtener las preferencias de la aplicación, ya que estas pueden afectar al modo o tiempo de ejecución de operaciones y eventos. También utilizan el Framework Protocolo para operaciones y eventos que necesitan la localización o el valor de sensores del dispositivo.

Interfaces de los elementos

Corresponden a las interfaces de la vista componente y conector.

C4.3) Diagrama de contexto

En la siguiente imagen se puede ver la arquitectura de Android a gran nivel de abstracción:

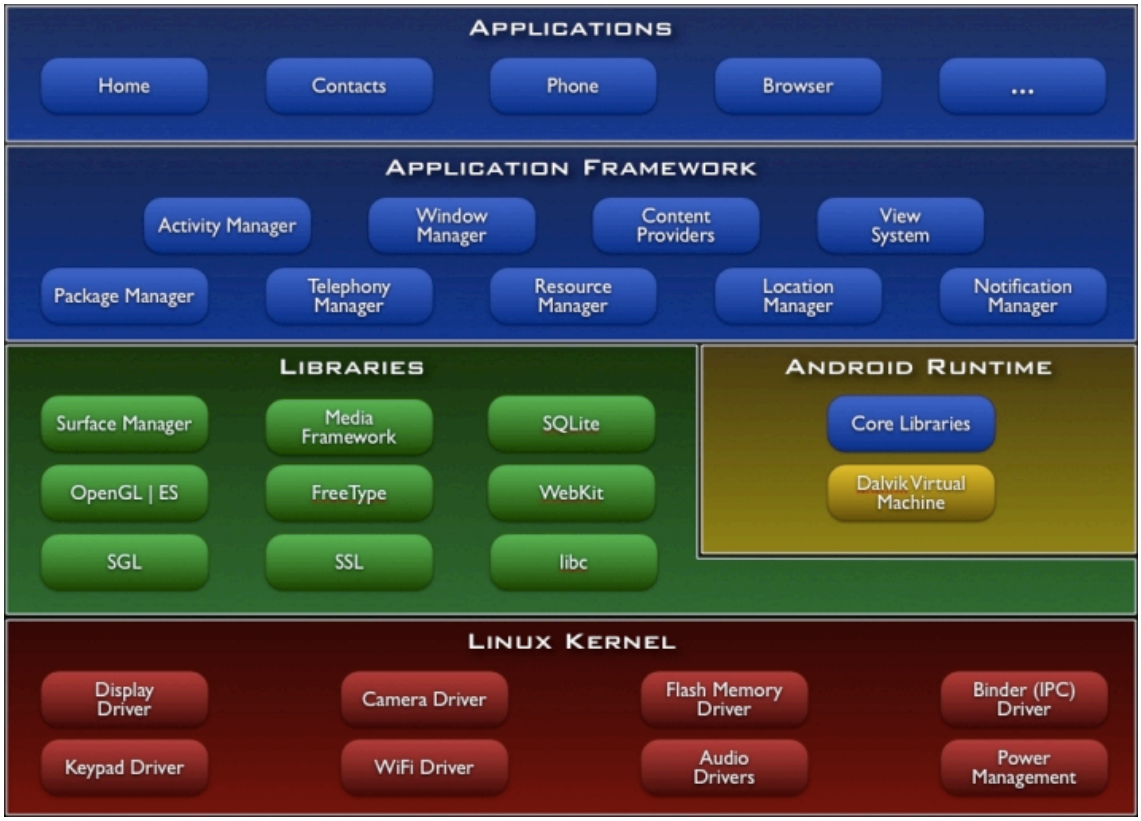


Figura 18. *Arquitectura Android - https://code.google.com/p/androidteam/wiki/AndroidSystemArch*

Los frameworks desarrollados en el TFG estarían en la capa Applitacions. Utilizaría los módulos de la capa Application Framwork, que a su vez utilizan los módulos de las capas inferiores.

C4.4) Guía de variabilidad

Se podría añadir más funcionalidad a las estadísticas. Que el módulo de operations estuviera relacionado con el módulo stats, y se almacenara que operaciones se invocan, en qué momento y cuántas veces, lo mismo con el módulo events.

C4.5) Exposición de razones

En este apartado se presentan las decisiones de diseño tomadas y los porqués de dichas decisiones:

1. **Decisión clase SMSSender separada**
    a. **Asunto**: Se separa una clase SMSSender en el módulo Message, cuya responsabilidad es el envío de mensajes SMS.
    b. **Decisión y estado**: Aceptado
    c. **Etiquetas**: SMS
    d. **Alternativas**: En primer lugar, la responsabilidad de enviar SMS recaía en la clase MessageService.
    e. **Argumento**: Se toma la decisión de separar esta responsabilidad en otra clase porque MessageService es un servicio de Android, y sólo se puede invocar a los métodos de un servicio si se hace un "bind" con la clase invocadora. El módulo Events necesita enviar mensajes y parecía más correcto separar esta responsabilidad en otra clase y así nos ahorramos tener que hacer "bind" del servicio MessageService con las clases de Events.
    f. **Implicaciones**: Se modificó el diagrama de módulos. Y se creó una clase SMSSender en el módulo Message.

2. **Decisión EventControllerService como servicio Android**
    a. **Asunto:** Se decide que EventControllerService sea un servicio Android en vez de una clase normal.
    b. **Decisión y estado**: Aceptado
    c. **Etiquetas**: Eventos
    d. **Alternativas:** Utilizar una clase normal de Java, que era instanciada desde el módulo Message.
    e. **Argumento:** Se ha decidido que sea un servicio Android porque siempre debe persistir durante el uso de la aplicación (ya que lanza periódicamente eventos), y no era correcto instanciarla desde el módulo Message, sino que es independiente a esto. De esta manera se desacoplan los dos módulos.
    f. **Implicaciones:** Se modificó el diagrama de módulos. Se modificó la clase EventControllerService para implementarla como servicio Android.

3. **Decisión pasar argumento Context a las operaciones**
    a. **Asunto:** Se decide que los módulos de Operations y Events reciban como argumento un objeto Context, que representa el contexto de la aplicación Android.
    b. **Decisión y estado**: Aceptado
    c. **Etiquetas**: Operaciones
    d. **Alternativas:** Que las clases que representan las operaciones, en vez de clases normales Java fueses Actividades o Servicios Android.
    e. **Argumento:** Se ha decidido pasar el objeto Context, porque es necesario para leer las preferencias de la aplicación (muchas operaciones y eventos utilizan

estas preferencias). Esta solución era mejor que todas las clases fuesen actividades o servicios, debido al aumento de complejidad del sistema que esto conlleva.

f. **Implicaciones:** Se modificó el diagrama de módulos. Y las interfaces de las operaciones.

4. Decisión patrón Singleton en una clase del módulo Location

a. **Asunto:** Se decide crear una clase LastLocation que implementa el patrón singleton, para acceder a la última localización obtenida del dispositivo.
b. **Decisión y estado**: Aceptado
c. **Etiquetas**: Localización
d. **Alternativas:** Guardar esta última localización en una base de datos.
e. **Argumento:** Se ha decidido utilizar el singleton, porque es una manera válida y más eficiente que guardar en base de datos, para compartir la última localización obtenida. Debido a que tanto como las operaciones y los eventos quieren acceder a esta información, se necesita que sea compartida, en este caso, a través del uso del patrón singleton.
f. **Implicaciones:** El módulo de Location, cuando recibe una nueva localización, queda encargado de actualizar el valor en el objeto singleton.

C5) Vista de distribución: estilo de despliegue

C5.1) Presentación primaria

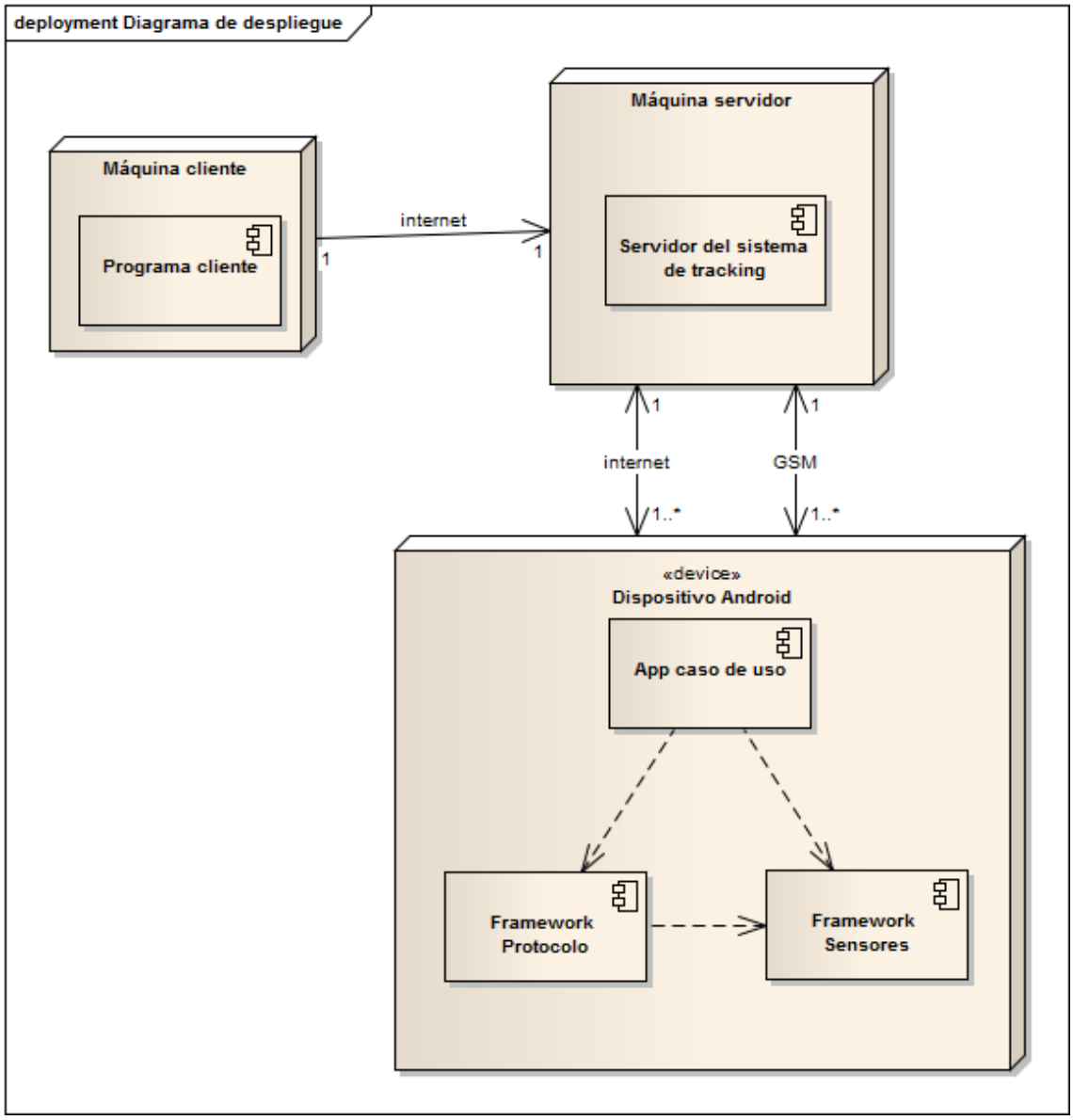


Figura 19. *Presentación primaria del estilo de despliegue de la vista de distribución*

C5.2) Catálogo de la vista

Elementos y sus propiedades

- El dispositivo Android, es un teléfono con sistema operativo Android, en el cual se ejecuta una aplicación caso de uso que utiliza los frameworks de protocolo y sensores. Dentro tiene los componentes:
    - La aplicación caso de uso, es una aplicación que utiliza los frameworks y añade nueva funcionalidad.
    - El framework protocolo, es el componente que implementa las operaciones y comunicación especificados en el protocolo.

    - El framework sensores se encarga de la lectura de sensores y localización del dispositivo.
- La máquina servidor, es una máquina donde se ejecuta el servidor del sistema de tracking. Tiene conexión a internet y conexión GSM (identificada por un número de teléfono). Dentro de ella está el componente Servidor que es la implementación del servidor, cumpliendo las especificaciones del protocolo.
- La máquina cliente es cualquier computador con acceso a internet. Dentro de ella se ejecuta un programa cliente que se comunica con el servidor y puede invocar sus operaciones.

Relaciones y sus propiedades

La máquina cliente se conecta a la máquina servidor mediante internet.

La máquina servidor se conecta con los dispositivos Android mediante internet (entornos online) y mediante GSM (entornos offline). Y los dispositivos Android a la máquina servidor de la misma manera.

C6) Mapeo entre vistas

Aquí se muestra una tabla con un mapeo entre la vista de componente y conector y la vista de módulos:

Elementos de la vista CyC	**Elementos de la vista de módulos**
Comunicaciones Online y Offline	Message Http
Conversor JSON	JSON
Estadísticas de uso	Stats
Operaciones y eventos	Operations Events
Acceso a base de datos	Data
Configuración framework	Config
Localización GPS	Location
Sensores del dispositivo	Sensors
Controlador estado del dispositivo	ModeController

D) Modelo dinámico

En este apartado se muestran unos diagramas de secuencia y colaboración para mostrar cómo interactúan los objetos del sistema.

A continuación, en la Figura 20 se ve el diagrama de secuencia que representa el envío de un evento PowerLowEvent. El sistema operativo Android lanza un intent o evento global avisando de batería baja. La clase EventControllerService captura el intent y se invoca su método onReceive. Dicho método invoca a al método execute de la clase PowerLow (la clase que representa el evento PowerLowEvent. Cada operación y evento tiene una clase y un método execute). Dicho método execute devuelve un string en formato JSON que contiene el evento en formato listo para ser enviado al servidor. EventControllerService invoca al método sendMEssage de laa clase SMSSender, y ésta le envía el SMS al servidor.

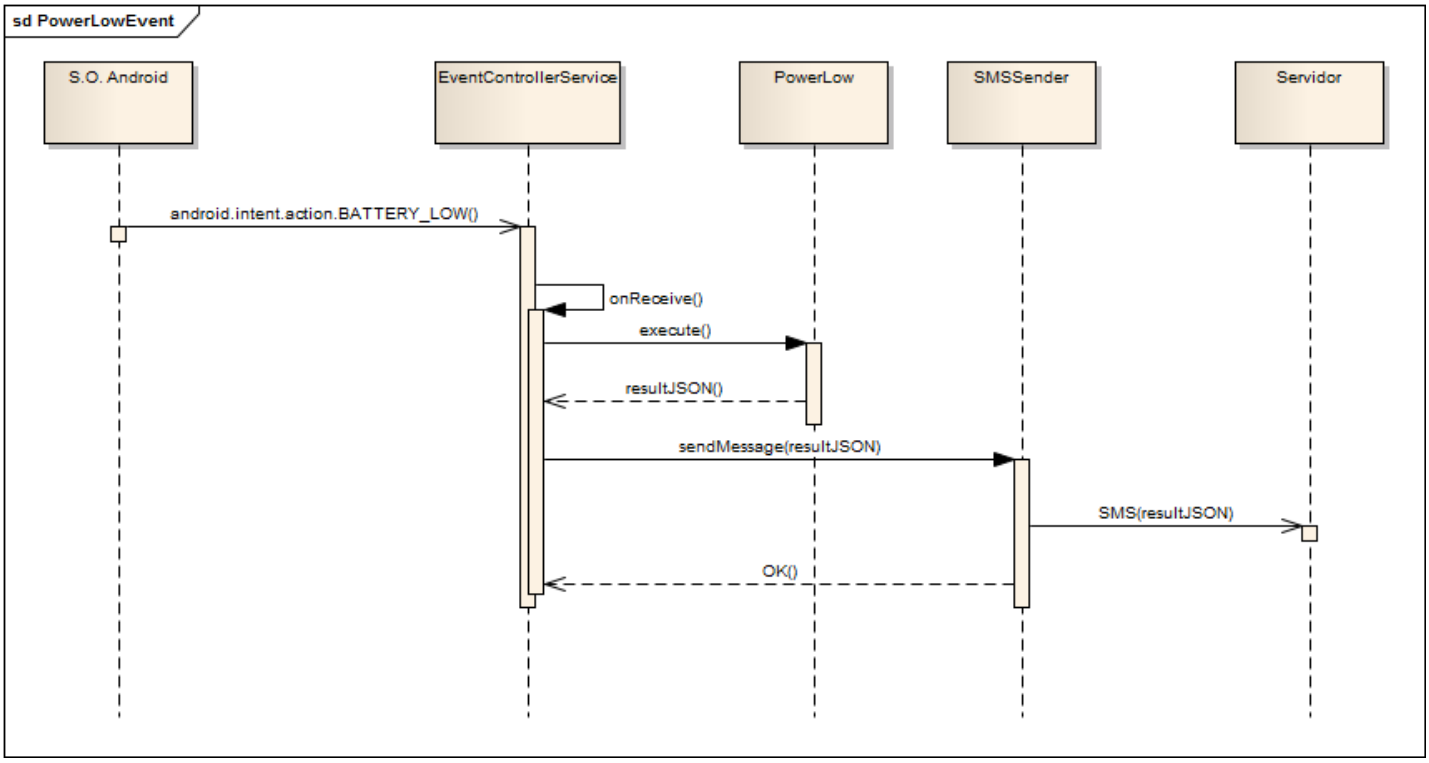


Figura 20. *Diagrama de secuencia evento PowerLow*

Después, en la Figura 21 se puede ver un diagrama de secuencia, en el cual el servidor del sistema, se comunica con el nodo Android a través de un mensaje SMS. Dentro de dicho SMS está codificada en JSON la operación PowerGetLevel (devuelve el nivel de batería del dispositivo). En el diagrama los mensajes que pone SMS(message) representan un mensaje SMS, siendo message la petición de la invocación PowerGetLevel. El resultado es otra cadena JSON que contiene el nivel de batería del nodo, y se le es devuelta al servidor mediante SMS.

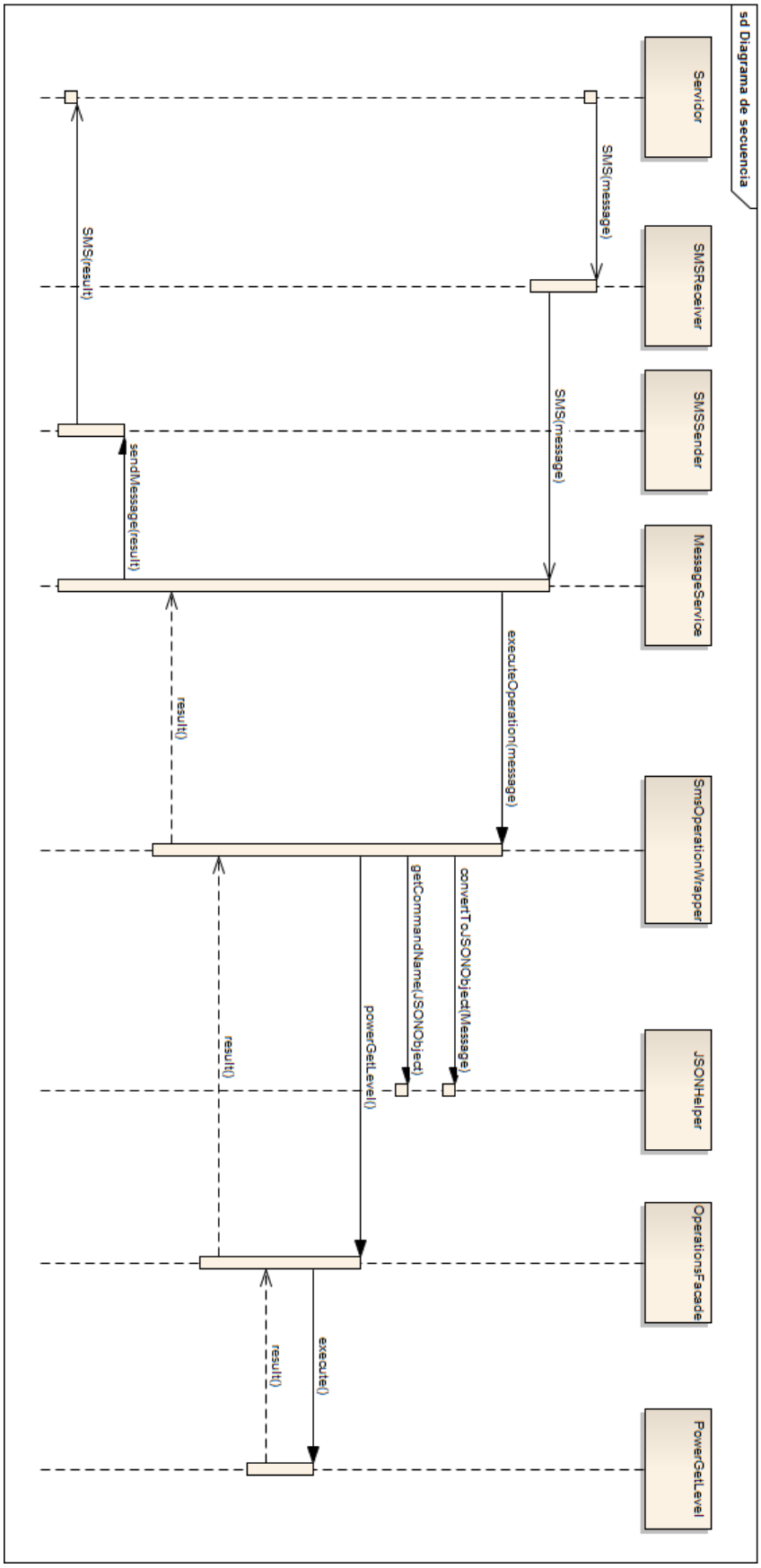


Figura 21. *Diagrama de secuencia, invocación PowerGetLevel*

En la Figura 21 se ha visto la invocación completa de una operación por parte del servidor, a través de SMS. En la Figura 22 se va a mostrar cómo se invocaría a la operación LocationGet. Por claridad, se va a obviar la invocación del servidor, y sólo se va a mostrar el diagrama desde OperationsFacade, sin embargo, la invocación sería igual que en el diagrama de la Figura 21.

Cuando se llama al método execute de LocationGet, éste obtiene una instancia del objeto LastLocation y a partir de ella le pide la localización. Después compone el mensaje JSON y se lo devuelve a OperationsFacade que le había invocado.

LastLocation es un objeto que sigue el patrón singleton, y sirve para almacenar la última localización obtenida. Dependiendo de la configuración del framework (se puede cambiar por el usuario o por operaciones del servidor) la localización se obtiene en un cierto intervalo, y cada vez que se obtiene se actualiza el valor en el objeto LastLocation, de esta manera LastLocation siempre almacena la localización más actual.

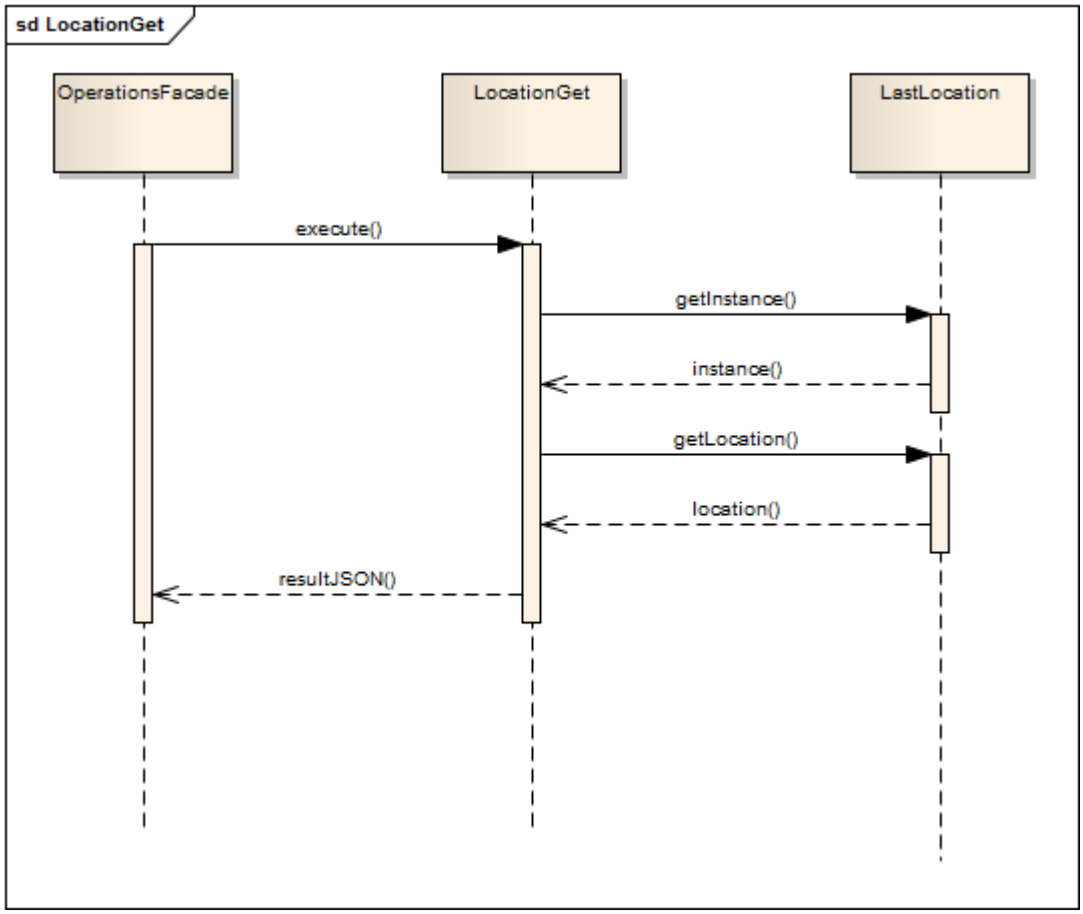


Figura 22. *Diagrama de secuencia LocationGet*

El siguiente diagrama de secuencia se centra en la invocación de una operación a través de HTTP, con notificaciones PUSH que ofrece el servicio Google Cloud Messaging. El servidor del sistema de tracking le envía el mensaje al GCM, y éste se lo envía al dispositivo Android mediante una notificación PUSH. Dicha notificación la recibe la clase GcmBroadcastReceiver, que le pasa el mensaje al servicio GcmIntentService. Éste, procesa el mensaje y llama a OperationFacade para que ejecute la operación adecuada. El resultado de la operación se le es devuelto en formato JSON (porque todas las operaciones devuelven el resultado en JSON). Pero, el protocolo especifica otro formato de datos diferente a JSON cuando se devuelven respuestas al servidor. Por tanto, el resultado se traduce invocando al objeto OperationTraductor. Una vez traducido se llama a HTTPSender, que envía el resultado al servidor. Todo lo anterior se puede ver en la Figura 23.

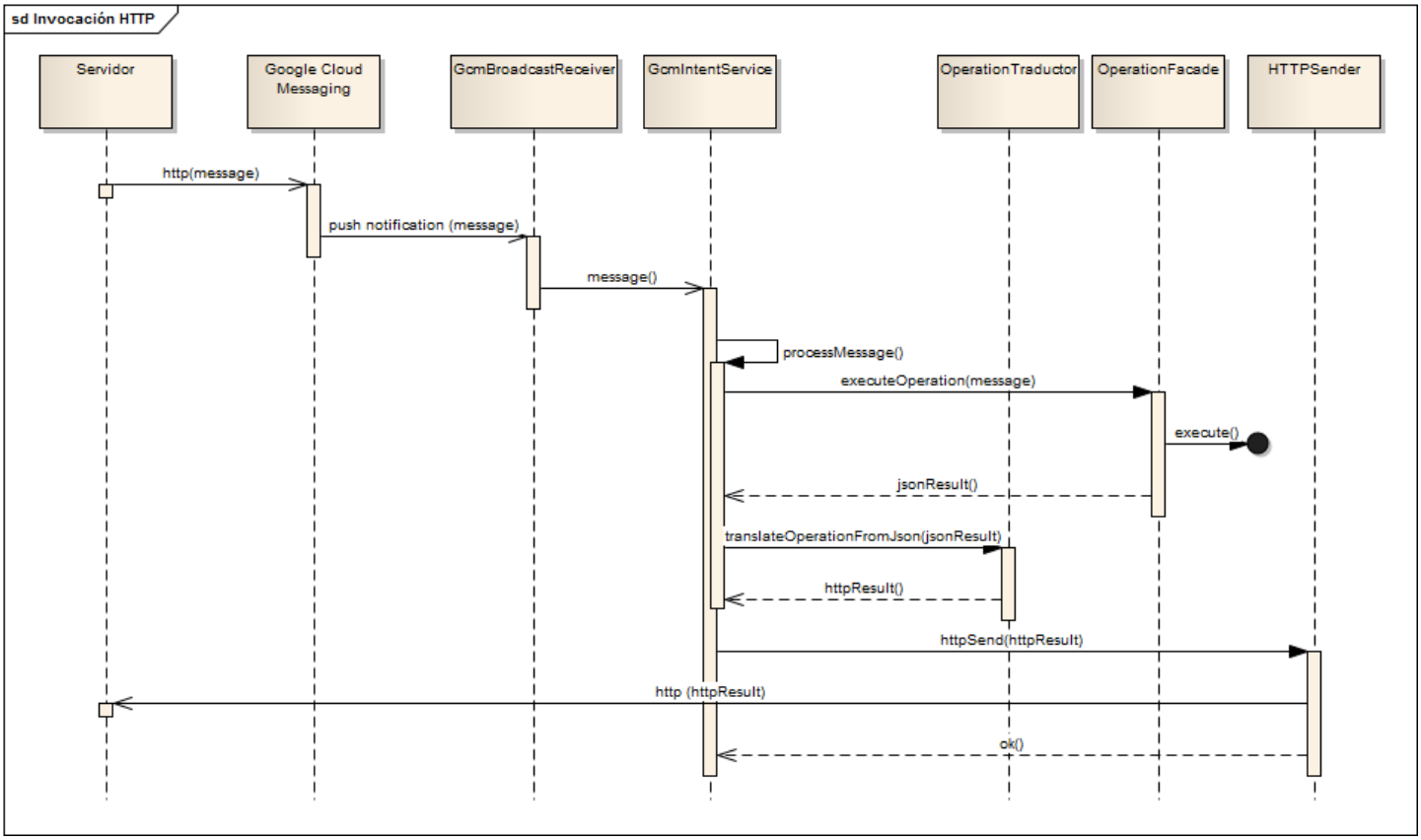


Figura 23. ***Diagrama de secuencia de la invocación de una operación por HTTP***

E) Adaptación y extensión del protocolo

En este anexo se describe, primero, los cambios realizados para adaptar el protocolo a dispositivos Android, y segundo, un análisis de los sensores Android para ver cómo se pueden incluir operaciones de lectura de dichos sensores en el protocolo.

E1) Adaptación del protocolo

El protocolo está pensado para ser capaz de trabajar en entornos offline, donde no se dispone de cobertura 3G. Sin embargo, también se soporta el trabajo en entornos online, siempre que el 3G esté disponible.

En un principio el protocolo sólo soportaba como nodos del sistema de tracking a dispositivos específicos para ello. Cada dispositivo tenía una dirección IP única y estática, por lo que la comunicación con el servidor vía HTTP es sencilla, ya que el servidor conoce las direcciones IP de todos los nodos, y los nodos conocen la dirección IP del servidor. Es por ello, que si el servidor central se quiere comunicar con un nodo, no tiene que hacer nada más que abrir una conexión HTTP con dicho nodo, puesto que conoce su IP fija y estática.

Al intentar integrar este modo de comunicación con un teléfono Android surge el problema de que éstos últimos no disponen de direcciones IP fijas, por lo tanto el servidor no se puede comunicar con los teléfonos Android que actúen como nodo. La comunicación de un teléfono Android al servidor sigue valiendo (el teléfono conoce la IP del servidor y puede abrir conexiones HTTP).

Para solucionar la comunicación HTTP del servidor a un nodo concreto se propone que tanto el servidor y los nodos Android implementen el mecanismo de notificaciones PUSH. Para ello se utilizará el servicio gratuito que ofrece Google para los desarrolladores de Android, el Google Cloud Messaging.

E1.1) Google Cloud Messaging

Es un servicio gratuito de Google, que ayuda a los desarrolladores a enviar datos desde los servidores a las aplicaciones Android (y viceversa). Estos datos pueden ser un mensaje que contenga hasta 4kb de información útil.

El servicio GCM se encarga de todos los aspectos de mantener la cola de mensajes y enviarlos a la aplicación Android que se ejecuta en el dispositivo. Una visión abstracta de la arquitectura del GCM se puede ver en la Figura 4.

Las principales características de este servicio se pueden resumir en:

- Permite que servidores externos (del programador) puedan enviar mensajes a sus aplicaciones Android.
- Permite que los dispositivos Android envíen mensajes al servidor.

- La aplicación que está en el dispositivo Android no necesita estar ejecutándose para recibir los mensajes. El sistema la despertará con un intent broadcast cuando el mensaje llegue.
- Le da el control a la aplicación de qué hacer con el mensaje recibido.
- Es necesaria una versión de Android superior a la 2.2 y tener la aplicación Google Play Store instalada.
- Utiliza una conexión de baja latencia y bajo ancho de banda para los servicios Google. Por ello es necesario tener una cuenta Google en el dispositivo (para dispositivos inferiores a Android 4.0.4).

E1.2) Nuevo funcionamiento para los nodos Android

Vista la solución anterior, la forma de que el servidor se comunique con los nodos será a través de notificaciones PUSH, utilizando el Google Cloud Messaging. Dentro de la notificación PUSH irá el mensaje en formato JSON, que contendrá la operación a ejecutar. Los mensajes JSON serán los mismos que ya se mandan a través de SMS en entornos offline.

Sin embargo, para realizar la comunicación en el otro sentido, es decir, para enviar respuestas desde los nodos al servidor, no es necesario utilizar GCM. Se puede utilizar el mecanismo HTTP que ya especificaba el protocolo, puesto que el servidor si tiene una IP fija a la que mandar los mensajes.

E2) Extensión del protocolo

Al haber introducido los dispositivos Android como posibles nodos en el protocolo, se puede aprovechar el hecho de que estos dispositivos incorporan sensores que pueden dar información útil del entorno en el que se encuentran.

Por ello, se ha decidido extender el protocolo para incorporar nuevas operaciones de lectura de dichos sensores.

Seguidamente, se van a presentar unas tablas explicando las características de los sensores en dispositivos Android.

E2.1) Listado de sensores soportados en la plataforma Android

En la Tabla 1 se pueden ver los sensores soportados por la plataforma Android.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with thegetRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATUREsensor in API Level 14	Monitoring temperatures.

Tabla 1. Tipos de sensores soportados por la plataforma Android

E2.2) Compatibilidad de los sensores según versión de Android

De los 13 sensores mostrados en la tabla anterior, no son todos compatibles con todas las versiones de Android, En la Tabla 2 se muestran las compatibilidades entre sensores y versiones de Android.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_AMBIENT_TEMPERATURE	**Yes**	n/a	n/a	n/a
TYPE_GRAVITY	**Yes**	**Yes**	n/a	n/a
TYPE_GYROSCOPE	**Yes**	**Yes**	n/a	n/a
TYPE_LIGHT	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_LINEAR_ACCELERATION	**Yes**	**Yes**	n/a	n/a
TYPE_MAGNETIC_FIELD	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_ORIENTATION	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_PRESSURE	**Yes**	**Yes**	n/a	n/a
TYPE_PROXIMITY	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_RELATIVE_HUMIDITY	**Yes**	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	**Yes**	**Yes**	n/a	n/a
TYPE_TEMPERATURE	**Yes**	**Yes**	**Yes**	**Yes**

Tabla 2. *Disponibilidad de sensores según plataforma*

E2.3) Sensores en el dispositivo Samsung GT-I9300 con Android 4.3

El teléfono utilizado para el desarrollo del trabajo, utilizado para ejecutar los framewroks y ejecutar los tests, ha sido un Samsung GT-I9300. Mediante el método getSensorList() de la clase SensorManager, se han sacado los sensores disponibles en dicho teléfono. Se puede ver en la Tabla 3.

Nombre del sensor	Tipo de sensor
LSM330DLC 3-axis Accelerometer	1
AK8975C 3-axis Magnetic field sensor	2
Orientation Sensor	3
iNemoEngine Orientation sensor	3
CM36651 Light sensor	5
CM36651 Proximity sensor	8
Corrected Gyroscope Sensor	4
LSM330DLC Gyroscope sensor	4
iNemoEngine Gravity sensor	9
Gravity Sensor	9
LPS331AP Pressure Sensor	6
iNemoEngine Rotation_Vector sensor	11
Rotation Vector Sensor	11
iNemoEngine Linear Acceleration sensor	10
Linear Acceleration Sensor	10

Tabla 3. *Sensores disponibles en Samsung GT-I9300*

Los sensores que tienen el mismo tipo miden y devuelven lo mismo. (10 tipos de sensores tiene el Samsung GT-I9300, y 15 sensores).

E2.4) Unidades y tipos devuelto de cada sensor

Según el tipo de sensor, se devuelve una medida diferente con diferentes unidades. Sin embargo, al leerlos, todos devuelven el mismo tipo de datos Java: un array de floats, de entre 1 a 3 componentes. En la Tabla 4 se detallan las medidas devueltas y en qué unidades.

Tipo de sensor	Tipo devuelto	Unidades
TYPE_ACCELEROMETER	**Float[] values :** values[0]: Acceleration minus Gx on the x-axis values[1]: Acceleration minus Gy on the y-axis values[2]: Acceleration minus Gz on the z-axis	**Unidades reales:** m/s^2
TYPE_AMBIENT_TEMPERATURE	**Float[] values :** values[0]: ambient (room) temperature	**Unidades reales:** degree Celsius.
TYPE_GRAVITY	**Float[] values :** values[0]: Gravity on the x-axis	**Unidades reales:** m/s^2

	values[1]: Gravity on the y-axis values[2]: Gravity on the z-axis	
TYPE_GYROSCOPE	**Float[] values :** values[0]: Angular speed around the x-axis values[1]: Angular speed around the y-axis values[2]: Angular speed around the z-axis	**Unidades reales:** radians/second
TYPE_LIGHT	**Float[] values :** values[0]: Ambient light level	**Unidades reales:** SI lux
TYPE_LINEAR_ACCELERATION	**Float[] values :** values[0]: Linear acceleration on the x-axis values[1]: Linear acceleration on the y-axis values[2]: Linear acceleration on the z-axis	**Unidades reales:** m/s^2
TYPE_MAGNETIC_FIELD	**Float[] values :** values[0]: magnetic field on the x-axis values[1]: magnetic field on the y-axis values[2]: magnetic field on the z-axis	**Unidades reales:** uT
TYPE_ORIENTATION	**Float[] values :** values[0]: Azimuth, angle between the magnetic north direction and the y-axis, around the z-axis (0 to 359). 0=North, 90=East, 180=South, 270=West values[1]: Pitch, rotation around x-axis (-180 to 180), with positive values when the z-axis moves toward the y-axis. values[2]: Roll, rotation around the x-axis (-90 to 90) increasing as the device moves clockwise.	**Unidades reales:** angles in degrees
TYPE_PRESSURE	**Float[] values :** values[0]: Atmospheric pressure	**Unidades reales:** hPa (millibar)
TYPE_PROXIMITY	**Float[] values :** values[0]: Proximity sensor distance	**Unidades reales:** cm

TYPE_RELATIVE_HUMIDITY	**Float[] values :** values[0]: Relative ambient air humidity in percent	**Unidades reales:** %
TYPE_ROTATION_VECTOR The rotation vector represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ around an axis <x, y, z>.	**Float[] values :** **3 componentes del vector value** The three elements of the rotation vector are <x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)>	**Unidades reales:** unitless
TYPE_TEMPERATURE	**Float[] values :** values[0]: Acceleration minus Gx on the x-axis values[1]: Acceleration minus Gy on the y-axis values[2]: Acceleration minus Gz on the z-axis	**Unidades reales:** m/s^2

Tabla 4. Tipos y unidades devueltos por cada sensor

F) Diseño y documentación del módulo de comunicaciones

Como ya se ha dicho en anteriores ocasiones, el módulo de comunicaciones del framework protocolo debe soportar, tanto comunicaciones en entornos online (HTTP), como en entornos offline (SMS). Es por ello, que en el módulo de comunicaciones se pueden diferencias dos módulos más pequeños, uno para cada tipo de comunicación. En la Figura 24 se puede ver los dos módulos con las clases que los forman:

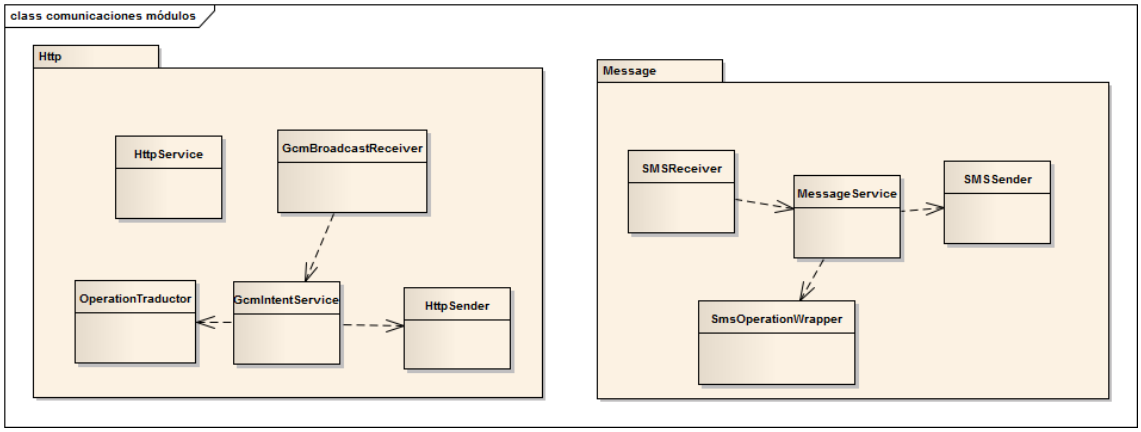


Figura 24. *Clases del módulo de comunicaciones*

El módulo Http es el responsable de comunicarse en entornos online mediante HTTP y notificaciones PUSH, y el módulo Message es el encargado de la comunicación en entornos offline mediante SMS.

F1) Módulo Message

Este módulo está formado por cuatro clases. Cada clase tiene una responsabilidad bien diferenciada. Se muestra en la Figura 25, el diagrama de clases del módulo, incluyendo los métodos y atributos de las mismas.

La clase SMSReceiver, es la encargada de recibir todos los SMS que son enviados al dispositivo. Extiende la clase BroadcastReceiver, para poder subscribirse a los eventos de tipo SMS que entran en el dispositivo. Se registra con prioridad alta, para que sea la primera que recibe los SMS (si no, otras aplicaciones del dispositivo podrían recibir los mensajes y ninguna más los recibiría).

Tiene un método onReceive, que es llamado cuando el sistema operativo le envía un SMS. Este SMS le llega codificado en formato PDU. El método lo decodifica y guarda en una variable el número de teléfono del destinatario, y en otra guarda el cuerpo del mensaje. Por último, lanza un evento, que incluye las dos variables. Este evento lo recibirá la clase MessageService.

La clase MessageService, es un servicio de Android. El cual tiene una clase interna que extiende un BroadcastReceiver suscrito a los eventos que envía la clase SMSReceiver. Cada vez que recibe un evento, obtiene el destinatario y el cuerpo del SMS. Invoca a la clase SMSOperationWrapper pasándole el cuerpo del mensaje. Esta clase se encargará de ejecutar

la operación adecuada según el mensaje, y devolverá el resultado de la operación. MessageService enviará ese resultado al servidor, a través de la clase SMSSender.

SMSOperationWrapper es la clase que conoce todas las operaciones posibles en el protocolo. Su función es invocar a la operación correcta, según el mensaje que ha enviado el servidor, y devolver el resultado a la clase MessageService.

Por último, la clase SMSSender es la encargada de envíar los SMS de vuelta al servidor. Tiene 4 métodos principales:

- **sendMessage**: si la red está conectada, manda el mensaje directamente invocando al método sendMessageNow. En caso contrario, almacena en mensaje en base de datos con el método storeMessage (Se almacenan con un flag de si han sido enviados o no).
- **sendMessageNow**: envía el mensaje al servidor
- **storeMessage**: almacena el mensaje en base de datos
- **sendStoredMessages**: intenta mandar todos los mensajes de la base de datos que tengan el flag de no enviado. Los mensajes que consiga enviar al servidor, les actualiza el flag en base de datos a enviado.

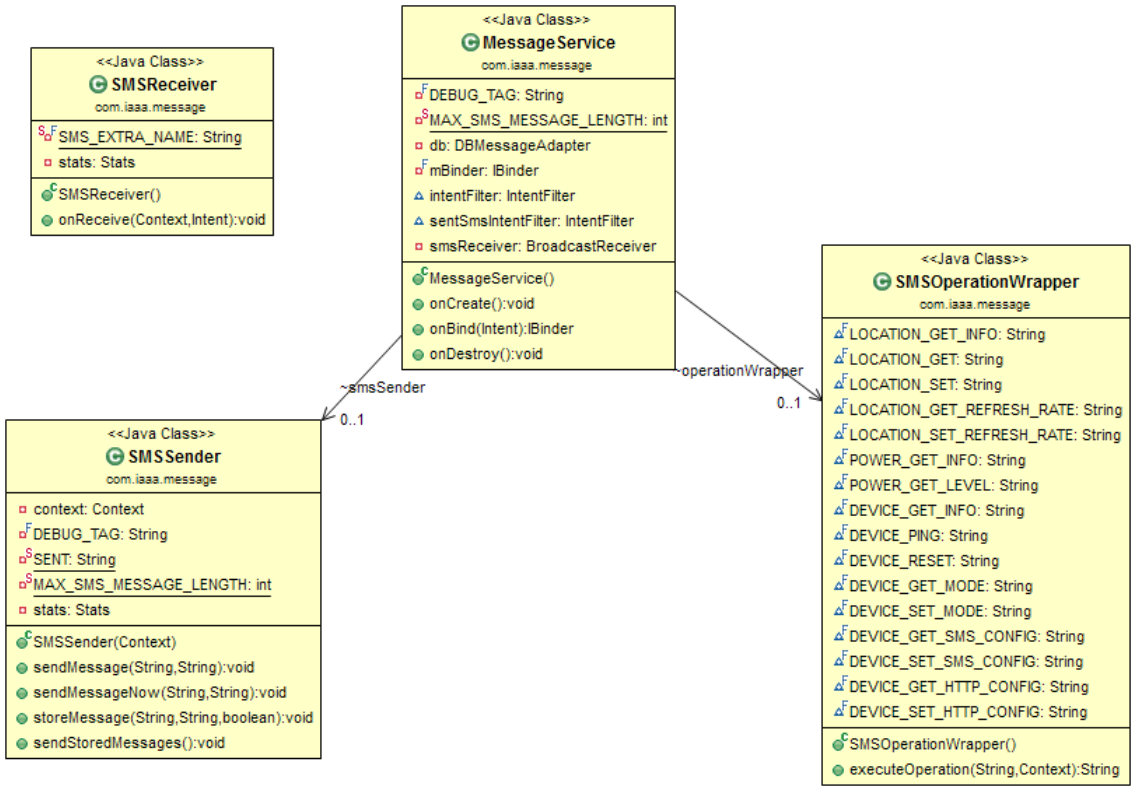


Figura 25. *Diagrama de clases, módulo Message*

F2) Módulo HTTP

Este es el módulo encargado de las comunicaciones con el servidor vía HTTP, en entornos con cobertura 3G. Consta de dos servicios Android, un BroadcastReceiver y tres clases.

En la Figura 26. *Diagrama de clases del módulo HTTP*, se puede ver el diagrama de clases del módulo.

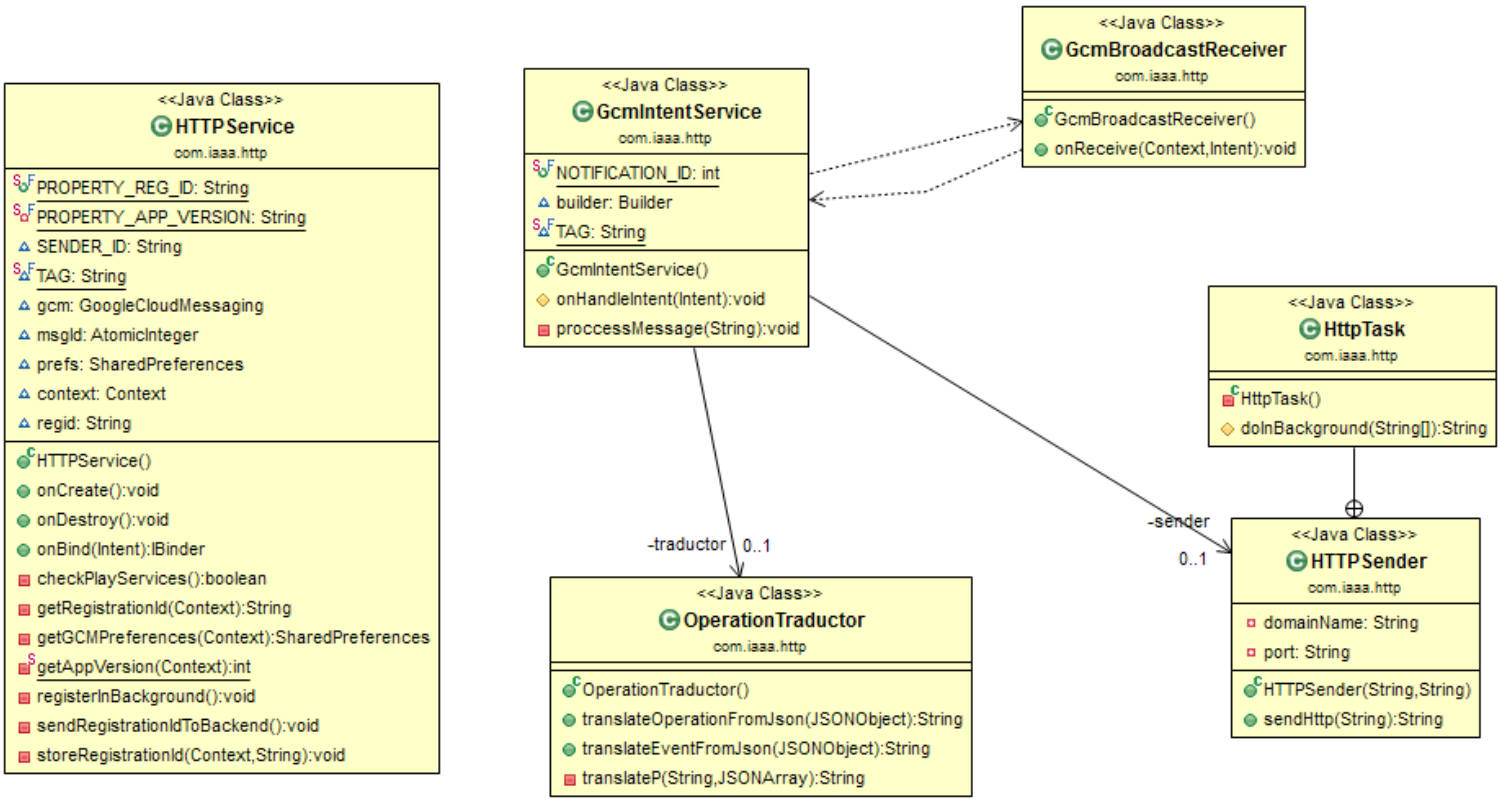


Figura 26. *Diagrama de clases del módulo HTTP*

El servicio HTTPService comprueba si está registrado el servidor para recibir notificaciones PUSH, en caso contrario lo registra y le manda su identificación de Google Cloud Messaging al servidor, para que éste le pueda enviar notificaciones PUSH.

El GcmBroadcastReceiver recibe las notificaciones PUSH del servidor. Mientras que el servicio GcmIntentService es el encargado de procesar dichas notifiaciones. Cuando le llega una notifiación invoca a la operación necesaria. Más tarde, almacena el resultado de la operación y lo envía al servidor mediante la clase HTTP Sender, traducida al lenguaje del protocolo HTTP mediante la clase OperationTraductor.

La clase HTTP sender envía el resultado de las operaciones mediante HTTP sin utilizar las notificaciones PUSH, puesto que el servidor del sistema tiene una IP fija conocida por los dispositivos.

G) Diseño módulo operaciones

En este anexo se explica cómo se ha diseñado e implementado el módulo de operaciones, que es el encargado de ejecutar las operaciones especificadas en el protocolo y devolver el resultado en formato JSON, listo para enviar al servidor.

El módulo se ha construido siguiendo el patrón de diseño de comportamiento Command. Que en programación orientada a objetos se utiliza para implementar un modelo petición-respuesta con bajo acoplamiento.

En el patrón command, la petición es enviada al invocador y él la pasa al objeto command encapsulado. Éste ejecuta la operación adecuada dependiendo de la petición que ha recibido. En este caso no se han utilizado clases receiver como en el patrón command habitual.

De esta manera hay un bajo acoplamiento entre las operaciones y su invocador, ya que sólo es necesario invocar al método execute() de cada clase (operación) para ejecutarla, y el resultado le es devuelto al invocador sin que él tenga saber cómo ha sido implementada la operación.

En la Figura 27 se puede ver el diagrama de clases del módulo operaciones (no muestra todas las clases por legibilidad del diagrama, sólo muestra algunas operaciones).

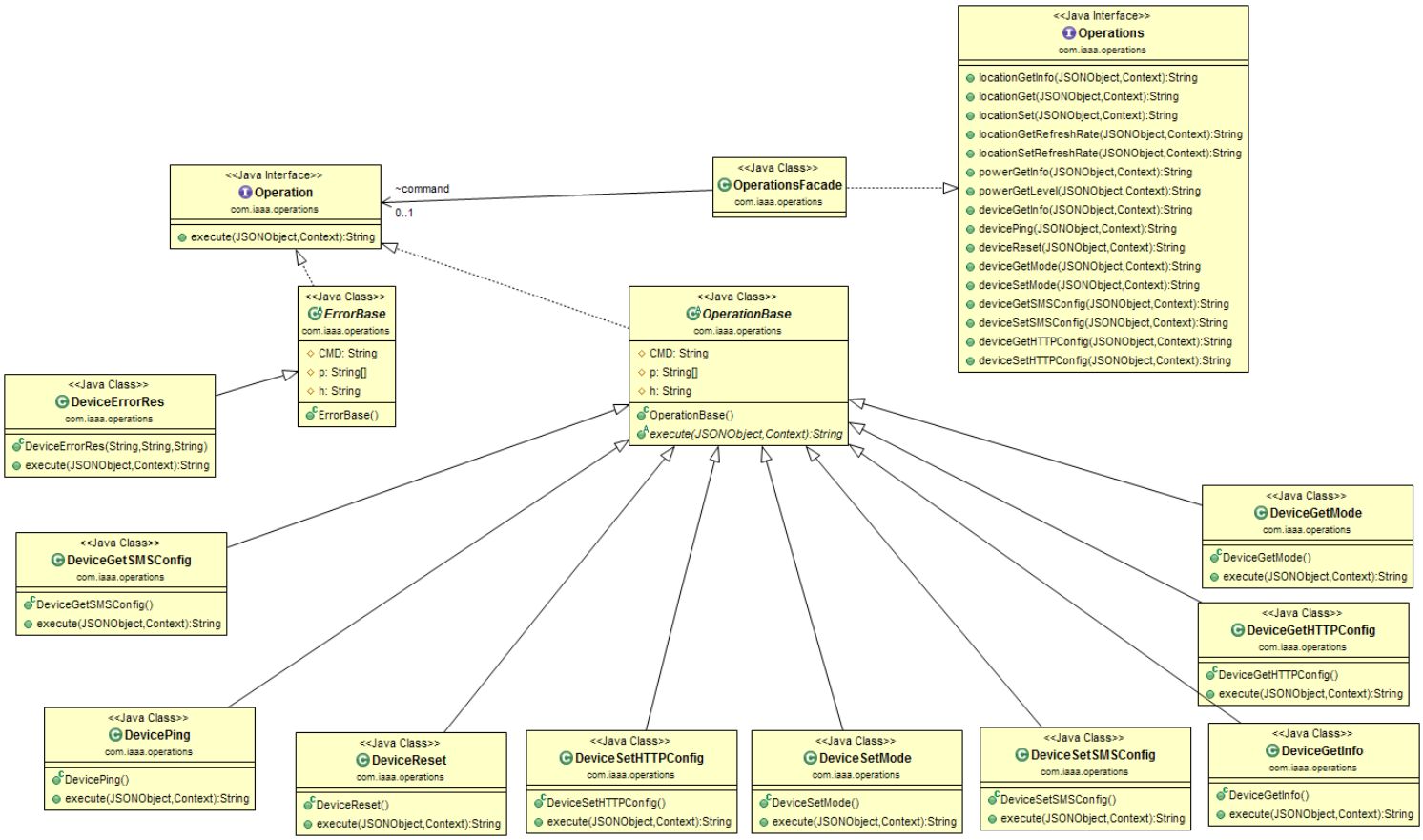


Figura 27. *Diagrama de clases (acortado) del módulo Operaciones*

Se puede ver en el diagrama de la Figura 27, que la estructura del módulo sigue un patrón command. Operation es la interfaz command, de la cual heredan las clases abstractas OperationBase y ErrorBase, que son clases de las cuales heredan las operaciones y errores respectivamente. Todas estas clases de operaciones y errores tienen el método execute que ejecuta la operación o error correspondiente y devuelve la respuesta en formato JSON en el formato necesario para enviársela al servidor. OperationsFacade es la clase invocadora el patrón, que implementa la interfaz Operations. Es la interfaz con la cual se comunicarán los otros módulos que llaman a las operaciones (Message y Http).

G1) Estructura de las clases de operaciones

Cómo se ha diseñado el módulo siguiendo el patrón command, cada Operación y Error es una clase que tiene un método execute(), que ejecuta dicha operación y devuelve el resultado en JSON. Además estas clases heredan de la clase OperationBase o ErrorBase, que además del método execute(), tienen unos atributos 'CMD', 'p', y 'h' que heredan todas sus subclases. Esto tiene una explicación, y es que para devolver la respuesta en formato JSON que entienda el servidor, se utiliza la biblioteca de Google Gson. Esta librería es capaz de serializar objetos, en cadenas JSON fijándose en sus atributos.

Por ejemplo, la operación DeviceGetInfo, que devuelve información del dispositivo, tiene como resultado una cadena JSON como la siguiente:

{"CMD":"DeviceGetInfoRes","p":["<deviceID>","<deviceVersion>"],"h":"XX"}

Se puede ver que tiene 3 claves: 'CMD', 'p' y 'h', que coinciden con los atributos que tienen todas las clases de operaciones.

Las respuestas en JSON de todas las operaciones siguen el mismo esquema. Una clave 'CMD', con el nombre de la respuesta, una clave 'p' con un array de los parámetros de respuesta de la operación, y una clave 'h' con el hash de seguridad del mensaje.

La operación DeviceGetInfo está implementada como se ve en la Figura 28. La estructura es la misma en todas las operaciones. Se le asigna al atributo 'CMD' el valor según la operación. Al atributo 'h' se le asigna su valor, y por último, se ejecuta las acciones necesarias de la operación y sus resultados se almacenan en el array 'p'.

De esta manera utilizando la biblioteca Gson, la generación de la respuesta en JSON es inmediata, utilizándose el método operationToJSONString() de la clase JSONHelper (que utiliza Gson), y nada más.

Todas las operaciones siguen el mismo patrón, y los errores también (tienen otros atributos definidos en la clase ErrorBase). Con este diseño se ha ahorrado tiempo y código de traducir las operaciones a JSON. Además es sencillo añadir nuevas operaciones, gracias al patrón command. Tan solo hay que añadir una nueva clase que herede de OperationBase e implementar adecuadamente el método execute(). La integración al sistema y la traducción a JSON de la nueva operación se hacen de forma inmediata, mejorando su escalabilidad y mantenimiento.

```java
package com.iaaa.operations;

import org.json.JSONObject;

import android.content.Context;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;

import com.iaaa.json.JSONHelper;


/**
 * Clase que representa la operación de DeviceGetMode
 * @author Ricardo Pallas
 */
public class DeviceGetMode extends OperationBase {

    /**
     * Constructor del objeto operación DeviceGetMode
     */
    public DeviceGetMode(){
        super();
        super.CMD = "DeviceModeRes";

    }

    /**
     * Ejecuta la operación que devuelve el modo de operación del nodo.
     */
    @Override
    public String execute(JSONObject jsonObject, Context context) {

        //Obtener preferencias porque especifican el powermode
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(context);
        String mode = sharedPref.getString("pref_powerModeType", "");

        //Generar respuesta
        super.h = "XY";
        super.p = new String[1];
        super.p[0] = mode;


        //Convertir a JSON String
        JSONHelper jsonHelper = new JSONHelper();
        String result = jsonHelper.operationToJSONString(this);

        //Devolver respuesta
        return result;
    }


}
```

Figura 28. *Código de la operación DeviceGetMode*

H) Diseño del módulo ModeController

La responsabilidad de este módulo es controlar el estado de la red (apagar las conexiones 3G y GSM) dependiendo del modo de energía en el que se está ejecutando el framework.

Este módulo no es de gran tamaño, sólo está compuesto por dos clases, pero su implementación trajo varias complicaciones. En la Figura 29 se muestra su diagrama de clases.

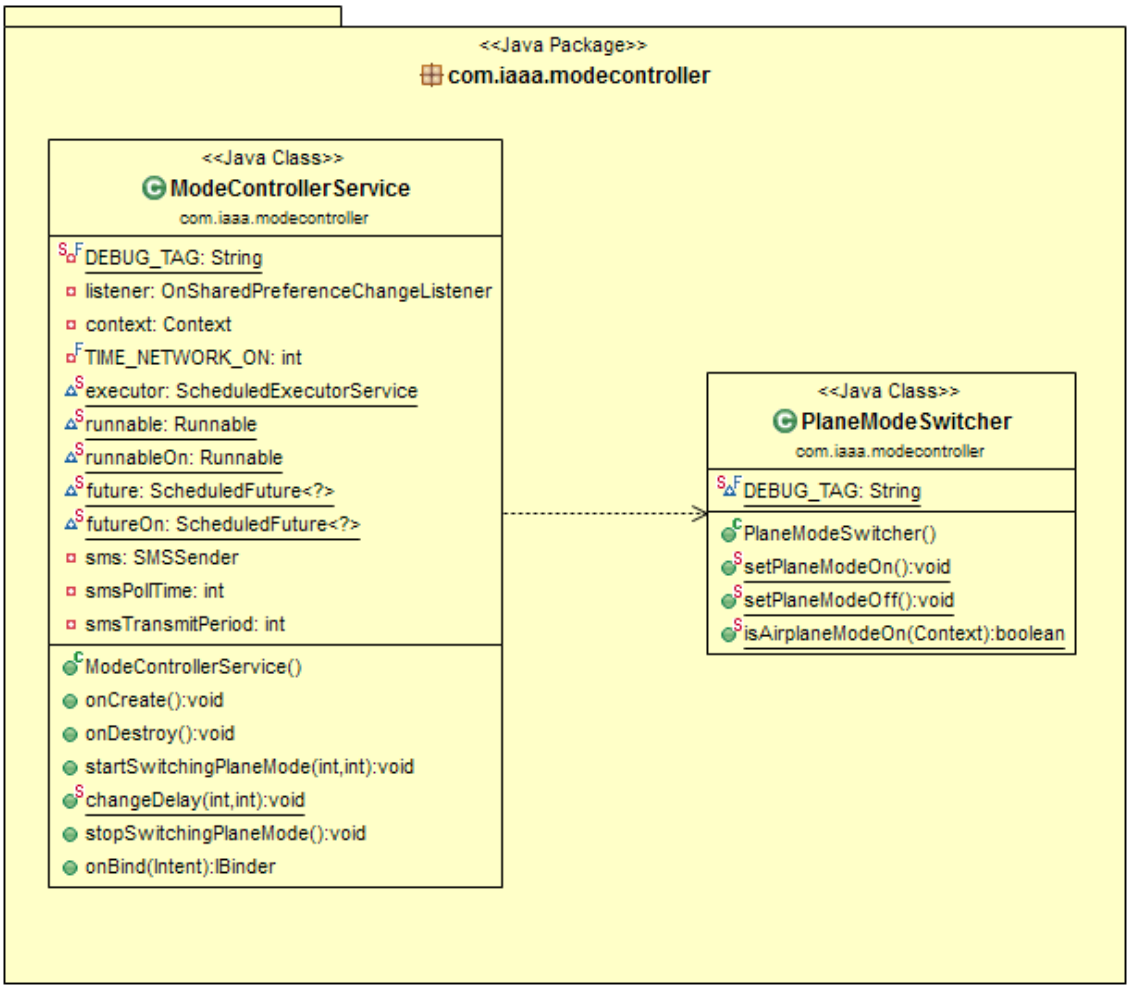


Figura 29. *Diagrama de clases módulo ModeController*

La clase PlaneModeSwitcher es una clase estática que ofrece tres métodos:

- **setPlaneModeOn:** activa el modo avión del teléfono.
- **setPlaneModeOff:** desactiva el modo avión del teléfono
- **isAirPlaneModeOn:** devuelve verdad si el modo avión está activado, y falso en caso contrario.

El SDK de Android no ofrece estas funcionalidades de tan bajo nivel, por lo que para implementar los métodos se ha tenido que utilizar la Shell de Android. Se obtiene una Shell

desde Java y se ejecutan comandos para activar y desactivar el modo avión. Además, para ejecutar estos comandos se necesitan permisos de root.

La clase ModeControllerService es un servicio de Android que se ejecuta en segundo plano. Su principal responsabilidad controlar el apagado y encendido de la red (a través del modo avión) de forma periódica dependiendo de la configuración actual de la aplicación

H1) Problemática en el diseño e implementación del módulo

H1.1) Problema apagar las conexiones

El protocolo especifica unos modos de energía específicos en los cuales debe operar un nodo dentro del sistema de tracking. Dichos modos se pueden ver en la Tabla 5.

Mode	Description
0	WAIT: The device is waiting to be configured. GSM: ON GPS: OFF
1	RUN 1: Extreme Low Power mode smsPollTime = 86400000ms (once at day) smsTransmitPeriod = 86400000ms (once at day) locationRefreshRate = 86400000ms (once at day)
2	RUN 2: Very Low Power mode smsPollTime = 86400000ms (once at day) smsTransmitPeriod = 86400000ms (once at day) locationRefreshRate = 900.000 (15 min.)
3	RUN 3: Low Power mode smsPollTime = 900.000ms (15 min.) smsTransmitPeriod = 900.000ms (15 min.) locationRefreshRate = 900.000ms (15 min.)
4	RUN 4: Always Locate mode GSM = ON GPS = Always Locate Mode smsTransmitPeriod = 60.000ms (1 min.) locationRefreshRate = 60.000ms (1 min.)

5	RUN 5: Custom smsPollTime = Adjusted by command smsTransmitPeriod = Adjusted by command locationRefreshRate = Adjusted by command

Tabla 5. ***Modos de energía que debe soportar un nodo***

En ellos se puede ver que se ajusta el tiempo de envío y recibo de SMS y el tiempo de refresco de la localización. Éste último no tiene inconveniente, ya que el framework de sensores ha sido diseñado de manera que se pueda fijar y cambiar un intervalo de refresco. Sin embargo, en cuanto a los mensajes, la única opción es apagar la red para no enviar y recibir SMS.

Lo primero que se hizo, fue estudiar las maneras de abordar este problema en Android. Primero se encontró que la clase ConnectivtyManager controlaba el estado de las redes del teléfono, pero ningún método apaga la red GSM. Analizando su código fuente se vio que tenía un método oculto, llamado setMobileDataEnabled(). Se probó a utilizarlo, accediendo a él a través de reflexión. El método funcionaba bien pero sólo apagaba la red 3G, por lo que los SMS seguían funcionado.

Después, otra opción es sobrescribir las configuraciones globales de Android poniendo la red como desconectada o conectada a través de código. Pero la opción se descartó, ya que para ello son necesarios los permisos WRITE_SECURE_SETTINGS, los cuales solo los pueden tener aplicaciones del sistema.

Por último se llegó a la solución de ese problema, utilizando otra solución. Y es mediante la activación/desactivación del modo avión del dispositivo, ya que este apaga o enciende todas las conexiones. Para encenderlo y apagarlo se hace a través de comandos de la Shell de Android y además se necesitan permisos de superusuario (Root). Los comandos utilizados se encapsularon en los métodos setPlaneModeOn() y setPlandeModeOff() de la clase PlaneModeSwitcher, su código se muestra en la Figura 30. Los comandos utilizados son:

- Para activar el modo avión:
  - “su”
  - “settings put global airplane_mode_on 1”
  - “am broadcast -a android.intent.action.AIRPLANE_MODE --ez state true”
  - “exit”
- Para desactivar el modo avión:
  - “su”
  - “settings put global airplane_mode_on 0”
  - “am broadcast -a android.intent.action.AIRPLANE_MODE --ez state false”
  - “exit”

H1.2) Problema de control de intervalos

Una vez resuelto el problema anterior, y teniendo encapsulada en una clase estática la funcionalidad de encender y apagar el modo avión (además de un método que indica si este está activo o no) se tenía que implementar el servicio Android ModeControllerService, de manera que controlase los intervalos de tiempo en los cuales está activado y desactivado el modo avión, según el modo de energía actual.

Por tanto las funciones que debía soportar ModeControllerService se pueden resumir en:

- Mantener el modo avión apagado durante el tiempo especificado
- Mantener el modo avión encendido durante el tiempo especificado
- Alternar entre modo avión apagado y modo avión encendido.
- Si un usuario a través de la interfaz, o el servidor del sistema a través de una operación, cambian el modo de energía, ModeControllerService debe cambiar en caliente los intervalos en los cuales el modo avión está apagado y el modo avión está encendido.

```
	/**
	 * Activa el modo avión en el teléfono
	 */
	public static void setPlaneModeOn(){
		try{
			//Se obtiene una shell con permisos root
			Process su = Runtime.getRuntime().exec("su");
			DataOutputStream outputStream = new
DataOutputStream(su.getOutputStream());
			//se sobrescriben las configuraciones globales del
dispositivo
			outputStream.writeBytes("settings put global
airplane_mode_on 1\n");
			outputStream.flush();
			//Se manda un broadcast en el dispositivo para que se
enteren de las nuevas configuraciones
			outputStream.writeBytes("am broadcast -a
android.intent.action.AIRPLANE_MODE --ez state true\n");
			outputStream.flush();
			//Se sale
			outputStream.writeBytes("exit\n");
			outputStream.flush();
			su.waitFor();
		}catch(IOException e){
			e.printStackTrace();
		}catch(InterruptedException e){
			e.printStackTrace();
		}
		//Log.d(DEBUG_TAG, "Plane mode is now on");
	}
	/**
	 * Desactiva el modo avión en el teléfono
	 */
	public static void setPlaneModeOff(){
		try{
			//Se obtiene una shell con permisos root
			Process su = Runtime.getRuntime().exec("su");
			DataOutputStream outputStream = new
DataOutputStream(su.getOutputStream());
			//se sobrescriben las configuraciones globales del
dispositivo
			outputStream.writeBytes("settings put global
airplane_mode_on 0\n");
			outputStream.flush();
			//Se manda un broadcast en el dispositivo para que se
enteren de las nuevas configuraciones
			outputStream.writeBytes("am broadcast -a
android.intent.action.AIRPLANE_MODE --ez state false\n");
			outputStream.flush();
			//Se sale
			outputStream.writeBytes("exit\n");
			outputStream.flush();
			su.waitFor();
		}catch(IOException e){
			e.printStackTrace();
		}catch(InterruptedException e){
			e.printStackTrace();
```

Figura 30. Código de las operaciones que controlan el modo avión

Escenario de ejemplo:

El framerwork está corriendo en un dispositivo Android y está configurado con el modo de energía 3: Low Power mode. Por lo tanto, la red debe de estar apagada 15 minutos (encender modo avión), y encendida 1 minuto para que dé tiempo a recibir y enviar los SMS.

El intervalo sería el siguiente:

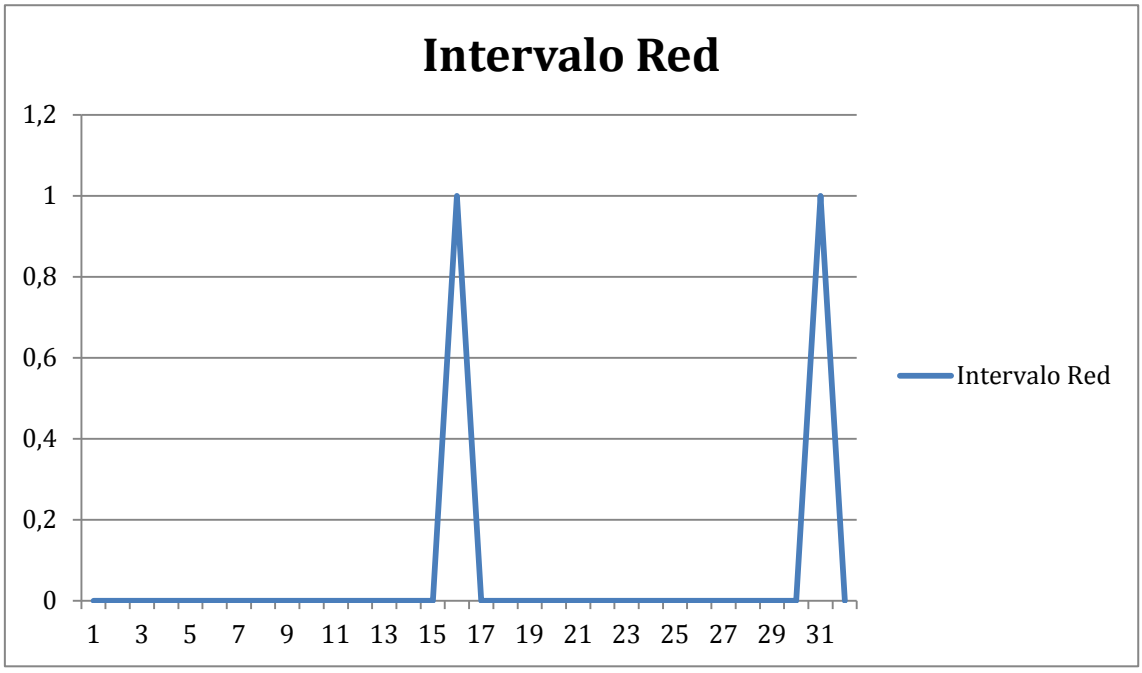


Figura 31. *Gráfico intervalo de conexión/desconexión de red en modo de energía 3*

La Figura 31, muestra el intervalo de conexión/desconexión de la red en el modo de energía 3. En el eje Y se muestra un valor numérico en el cual el 0 representa que las redes están apagadas (modo avión encendido), y en 1 representa que las redes están encendidas (modo avión apagado).

El eje X muestra el tiempo en minutos. Se puede ver que 15 minutos las redes están apagadas, para mantenerse encendidas 1 minuto, después se vuelve a apagar 15 minutos para volverse a encender durante otro minuto y así sucesivamente.

Dependiendo del modo de energía estos intervalos cambian, por lo que se debe de implementar un método genérico que esté parametrizado con los tiempos de red encendida y apagada.

Implementar este funcionamiento en Android fue problemático. Para solucionarlo se estudiaron varias posibles soluciones:

- **AlarmManager:** clase de Android para programar alarmas que ejecutan tareas
- **ScheduledExecutorService**: clase Java/Android para lanzar tareas repetitivas.
- **Hanlder**: clase Android para ejecutar objetos Runnable en un tiempo fijo.
- **Quartz Scheduler**: biblioteca Java muy completa para programar tareas.
- **BuzzBoz SDK:** biblioteca Android para programar tareas.

El problema de **AlarmManager**, es que, en intervalos de tiempo pequeños funcionaba bien para solucionar el problema, sin embargo en intervalos grandes (1 día en el modo 1 o modo 2) la ejecución de conexión/desconexión de red no es inmediata al principio del intervalo, sino que se ejecuta en un punto aleatorio en el intervalo, para ahorrar energía. Es decir, si la aplicación está en el modo de energía 3 (se conecta la red cada 15 minutos) y, justamente cuando la red está encendida llega una operación que cambia el modo de energía al 1 (se conecta la red una vez al día), se debería apagar la red de inmediato durante un día. Pero lo que pasaría con AlarmManager es que no se apagaría inmediatamente, si no que se apagaría en un punto aleatorio del intervalo y la red podría estar encendida 20 minutos, por ejemplo. No es el resultado deseado.

El problema de **Handler** es que está desaconsejado para intervalos grandes de tiempo, y sólo permite un thread. No se podrían tener dos threads, uno que controle el encendido de la red y otro que controle el apagado de la red.

El problema de **BuzzBox SDK**, es que solo permite tener un intervalo fijo y no dos. En este problema se necesitan dos intervalos diferentes de tiempo, uno de apagado y otro de encendido.

Quartz Scheduler, es la más potente y completa de todas las opciones barajadas, sin embargo no es posible ejecutarla en Android. En la figura se puede ver el error producido al intentar ejecutarlo.

```
04-02 08:25:04.290: E/AndroidRuntime(15952): FATAL EXCEPTION: main
04-02 08:25:04.290: E/AndroidRuntime(15952): java.lang.NoClassDefFoundError: java.beans.Introspector
04-02 08:25:04.290: E/AndroidRuntime(15952): at
org.quartz.impl.StdSchedulerFactory.setBeanProps(StdSchedulerFactory.java:1393)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
org.quartz.impl.StdSchedulerFactory.instantiate(StdSchedulerFactory.java:849)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
org.quartz.impl.StdSchedulerFactory.getScheduler(StdSchedulerFactory.java:1519)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
com.iaaa.modecontroller.ModeControllerService.startSwitchingPlaneMode(ModeControllerService.java:166)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
com.iaaa.modecontroller.ModeControllerService.onCreate(ModeControllerService.java:149)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
android.app.ActivityThread.handleCreateService(ActivityThread.java:2705)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
android.app.ActivityThread.access$1600(ActivityThread.java:153)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
android.app.ActivityThread$H.handleMessage(ActivityThread.java:1351)
04-02 08:25:04.290: E/AndroidRuntime(15952): at android.os.Handler.dispatchMessage(Handler.java:99)
04-02 08:25:04.290: E/AndroidRuntime(15952): at android.os.Looper.loop(Looper.java:137)
04-02 08:25:04.290: E/AndroidRuntime(15952): at android.app.ActivityThread.main(ActivityThread.java:5289)
04-02 08:25:04.290: E/AndroidRuntime(15952): at java.lang.reflect.Method.invokeNative(Native Method)
04-02 08:25:04.290: E/AndroidRuntime(15952): at java.lang.reflect.Method.invoke(Method.java:525)
04-02 08:25:04.290: E/AndroidRuntime(15952): at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:739)
04-02 08:25:04.290: E/AndroidRuntime(15952): at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:555)
04-02 08:25:04.290: E/AndroidRuntime(15952): at dalvik.system.NativeStart.main(Native Method)
```

Figura 32. ***Error producido al tratar ejecutar la biblioteca Quartz en Android***

La solución final fue lanzar un objeto de la clase **ScheduledExecutorService,** con un pool de 2 threads, uno que controla el apagado y otro el encendido de la red. Al principio esta solución no funcionaba bien porque se desfasaba unos segundos la ejecución periódica, por lo que hubo que utilizar su método scheduleAtFixedRate() que es exacto.

Se ha programado de la siguiente manera:

- el método startSwitchingPlaneMode inicia los intervalos. Se le pasa como parámetro dos enteros, timeNetworkOn y timeNetworkOff, que son los tiempos en los cuales la red debe estar encendida y apagada, en segundos.
- El thread que enciende la red, se cada (timeNetworkOn + timeNetworkOff) segundos.
- El thread que apaga la red se lanza cada (timeNetworkOn + timeNetworkOff) segundos, pero con un retraso inicial de timeNetworkOn segundos.
- De esta manera se consigue solucinar el problema

Ejemplo de funcionamiento:

Para conseguir los intervalos del modo de energía 3 (ver Figura 31) se llamaría al método startSwitchingPlaneMode con los parámetros timeNetworkOn=60 segundos y timeNetworkOff=900 segundos.

El thread que enciende la red se lanzaría cada 960 (900 + 60) segundos desde el momento que se ejecuta, y el que apaga la red se lanza cada 960 (900 + 60) segundos, con un retraso inicial de 60 segundos.

De esta forma: el primer thread enciende la red, se mantiene 60 segundos hasta que se lanza el segundo thread que la apaga. Se vuelve a encender 900 segundos después (se había apagado a los 60 segundos de encender, y el que enciende se ejecuta cada 960) se mantiene encendida 60 segundos hasta que se vuelve a ejecutar el thread que la apaga (va con 60 segundos de retraso con el que la enciende) y así sucesivamente.

H1.3) Cambios en módulo Message y Base de datos

Al haber implementado los modos de energía, es posible que, dependiendo del modo, la red no esté disponible más que una vez al día. Pero, durante ese tiempo que la red está apagada, es posible que el nodo quiera mandar eventos por SMS de su localización, batería u otra información, y no será posible al estar el modo avión conectado.

Con el objetivo de no perder esos eventos que se deben enviar al servidor, se tuvo que crear una base de datos y modificar la lógica del módulo de mensajes. La solución al problema es intentar enviar el evento por SMS; si se tiene red se envía directamente al servidor, y si no se tiene, se almacena en base de datos para enviarlo en otro momento. Además, cada vez que se conecta la red, se intentan mandar todos los mensajes de la base de datos que no habían sido enviados. Con esta solución, al servidor le llegarán todos los eventos.

El módulo de acceso a base de datos y el modelo de datos utilizado se detalle en el anexo I)

Los cambios que se hicieron en el módulo Message, fue el añadir métodos de almacenar mensaje y enviar todos los mensajes de base de datos, explicados en el anexo F1) .

I) Diseño módulo Data y modelo base de datos

En un principio se desestimó el uso de base de datos, ya que se decidió almacenar las configuraciones del protocolo con el mecanismo que ofrece Android de almacenamiento de Preferencias en ficheros XML. Sin embargo, debida a la implementación de modos de energía, explicada en el anexo H) , surgió la necesidad de almacenar los mensajes no enviados en la base de datos, para poder enviarlos cuando sea posible. La base de datos es SQLite, la cual es ligera y fácil de operar con Android.

I1) Modelo de datos para almacenar mensajes

Es un modelo de datos sencillos, ya que la base de datos sólo se utiliza para almacenar mensajes. La información que se desea guardar de un mensaje es:

- Número de teléfono del remitente
- Cuerpo del mensaje
- Fecha en la cual se almacena el mensaje
- Un flag para saber si el mensaje ha sido enviado ya o no
- Una id para diferenciar mensajes

En la Figura 33 se puede ver la tabla de mensajes.

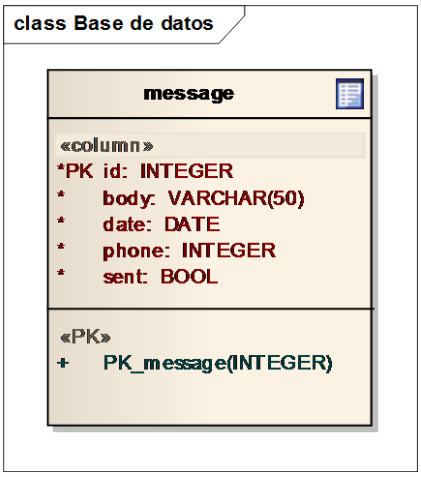


Figura 33. Modelo de datos, para almacenar SMS

I2) Diseño módulo Data

El módulo Data consta de una clase DBMessageAdapter que ofrece operaciones para insertar/borrar/recuperar mensajes de la base de datos. Además tiene una clase interna DatabaseHelper que extiende la clase SQLiteOpenHelper para la creación de la base de datos.

En la figura se puede ver el diagrama de clases del módulo, en el cual se muestran los métodos de la clase DBMessageAdapter.

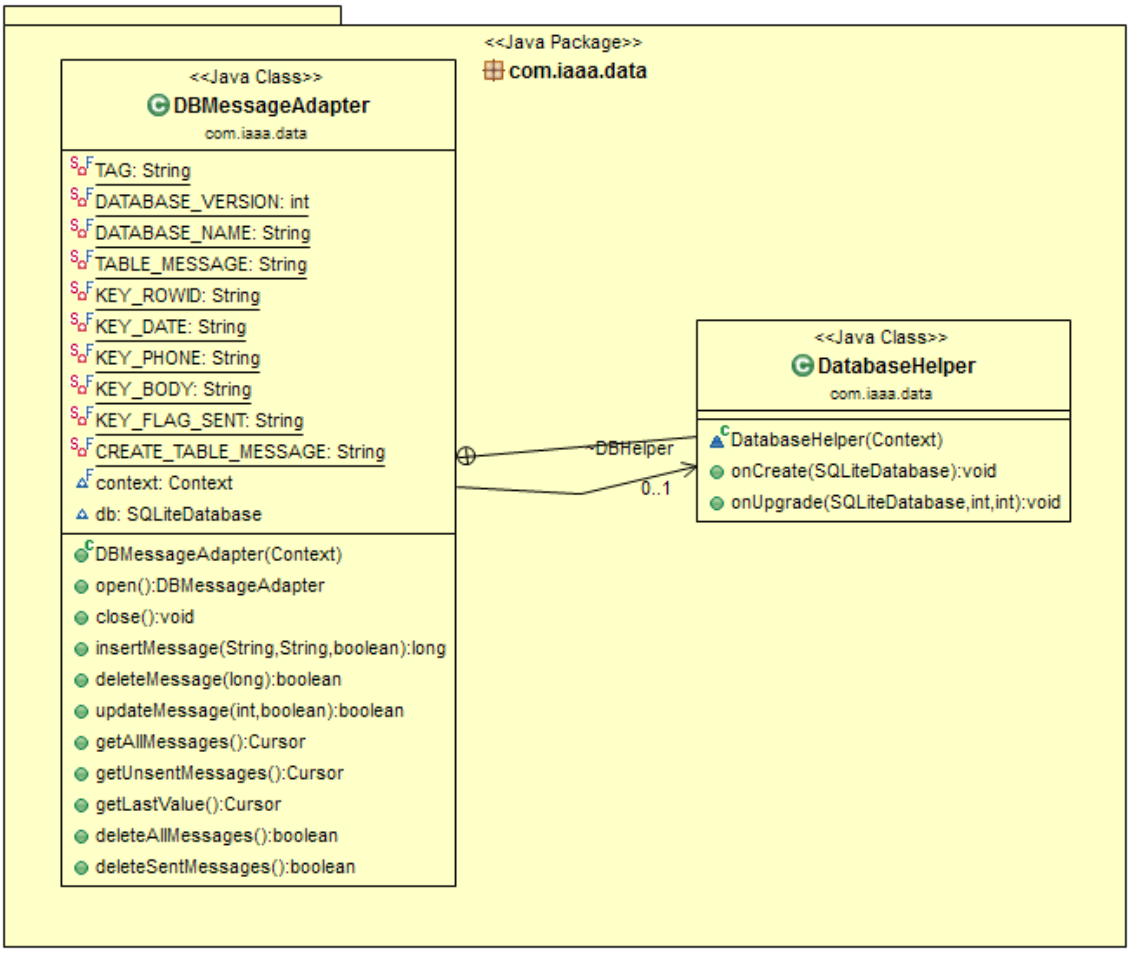


Figura 34. *Diagrama de clases del módulo Data*

En la tabla Tabla 6 se pueden ver los métodos de la clase DBMessageAdapter, con sus tipos devueltos y una explicación de su función.

Tipo devuelto	Método y descripción
void	**close**() Cierra la base de datos
boolean	**deleteAllMessages**() Borra todos los mensajes en la base de datos
boolean	**deleteMessage**(long rowId) Borra el mensaje con Id == rowId de la base de datos
boolean	**deleteSentMessages**() Borra todos los mensajes en la base de datos que tengan el flag isSent a true, es decir los mensajes que ya han sido enviados.
android.database.Cursor	**getAllMessages**() Recupera todos los mensajes almacenados en la base de datos
android.database.Cursor	**getLastValue**() Devuelve el último mensaje almacenado en la base de datos
android.database.Cursor	**getUnsentMessages**() Devuelve todos los mensajes almacenados que no han sido enviados, es decir que su flag sent es false
long	**insertMessage**(java.lang.String phoneNumber, java.lang.String body, boolean sent) Almacena un mensaje en la base de datos
DBMessageAdapter	**open**() Abre la base de datos
boolean	**updateMessage**(int rowId, boolean isSent) Actualiza el valor del flag que indica si el mensaje ha sido enviado del mensaje con id == rowId

Tabla 6. *Métodos de la clase DBMessageAdapter*

J) Diseño framework sensores

El framework de sensores es utilizado por el framework protocolo, pero podría ser utilizado por cualquier aplicación que necesitase leer los sensores del dispositivo y obtener su localización de forma periódica. Es por ello que se ha implementado como una biblioteca Android.

Se divide en dos partes: sensores y localización. Esta división está hecha porque, aunque el GPS se podría considerar como un sensor más, su lectura es diferente a la de los demás sensores.

J1) Módulo localización

En la Figura 35 se puede ver el diagrama de clases de la parte de localización del framework.

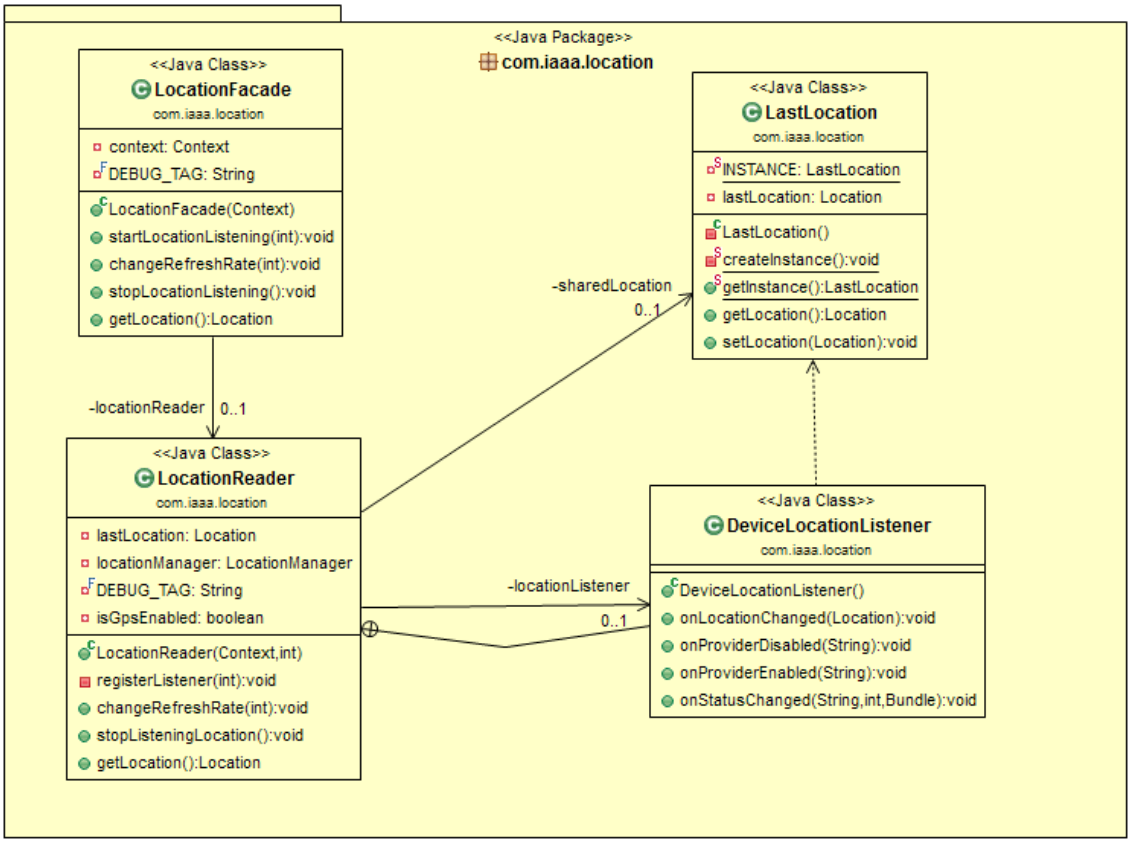


Figura 35. *Diagrama de clases del módulo de localización*

EL módulo de localización tiene cuatro clases principales LocationFacade, LocationReader, DeviceLocationListener y LastLocation. La clase DeviceLocationListener es una clase internet de LocationReader.

- **LocationFacade**, es la clase que sirve de interfaz, utilizando el patrón fachada, al exterior. Ofrece una serie de métodos para que el framework protocolo pueda invocarlos y obtener la localización del dispositivo, éstos son:

- startLocationListening(int refreshRate): construye el objeto LocationReader que empieza a escuchar la localización a través del GPS en intervalos especificados por refreshRate (tasa de refresco de la localización).
- changeRefreshRate(int refreshRate): cambia la tasa de refresco, una vez iniciada la escucha, a la especificada por el parámetro refreshRate.
- stopLocationListening(): detiene la escucha del GPS y deja de obtener la localización.
- getLocation(): devuelve la última localización obtenida.

- **LocationReader:** Es la clase encargada de empezar a obtener la localización del dispositivo, detener las escuchas y devolver la última localización obtenida. Cuando se construye el objeto, se le pasa a su constructor la tasa de refresco de obtención de localización. Ofrece métodos para cambiar esta tasa, y para parar las escuchas. Lo que en realidad hace es registrar un LocationListener que escucha los cambios de localización. Ese LocationListener es su clase interna DeviceLocationListener.

- **DeviceLocationListener:** es la clase interna encargada de suscribirse a los eventos de cambio de localización. Cuando la localización cambia se llama a su método onLocationChanged(Location location) (la frecuencia de llamada a este método depende del parámetro refreshRate del constructor de la clase LocationReader). En dicho método, la última localización se almacena en la clase singleton LastLocation.

- **LastLocation:** Esta clase sigue un patrón Singleton. Su función es almacenar la última localización obtenida. Al seguir el patrón Singleton, sólo existe una instancia y se puede compartir. Esto es así porque el framework protocolo accede a dicha localización y la clase DeviceLocationListener también. Cuando DeviceLocationListener obtiene un nueva localización, actualiza el valor de LastLocation. Y cuando la operación LocationGet, del framework protocolo, es invocada, lee el valor de LastLocation para obtener la última localización y enviársela al servidor.

Se podrían registrar las localizaciones en base de datos o en un fichero, pero se pensó que utilizar un patrón Singleton era adecuado en este caso, porque, no importaba el historial de localizaciones (se pueden borrar las antiguas) y es de acceso más rápido que base de datos (si el la tasa de refresco es pequeña habría demasiadas transacciones).

J2) Módulo sensores

Este es el módulo encargado de leer los valores de los sensores del dispositivo. Su estructura de clases interna es parecida a la de Localización, teniendo una clase fachada que ofrece una interfaz al exterior, y una clase encargada de la lectura de los sensores. Además se ha definido una clase SensorValue que almacena la lectura de cualquier sensor Android (véase la sección E2.1)). En la Figura 36 se puede ver el diagrama de clases del módulo. Se han incluido métodos para leer el sensor de luz y de luminosidad (para no hacer más grande el diagrama), sin embargo todos los sensores se pueden leer de la misma manera, ya que los valores obtenidos se almacenan como un objeto SensorValue.

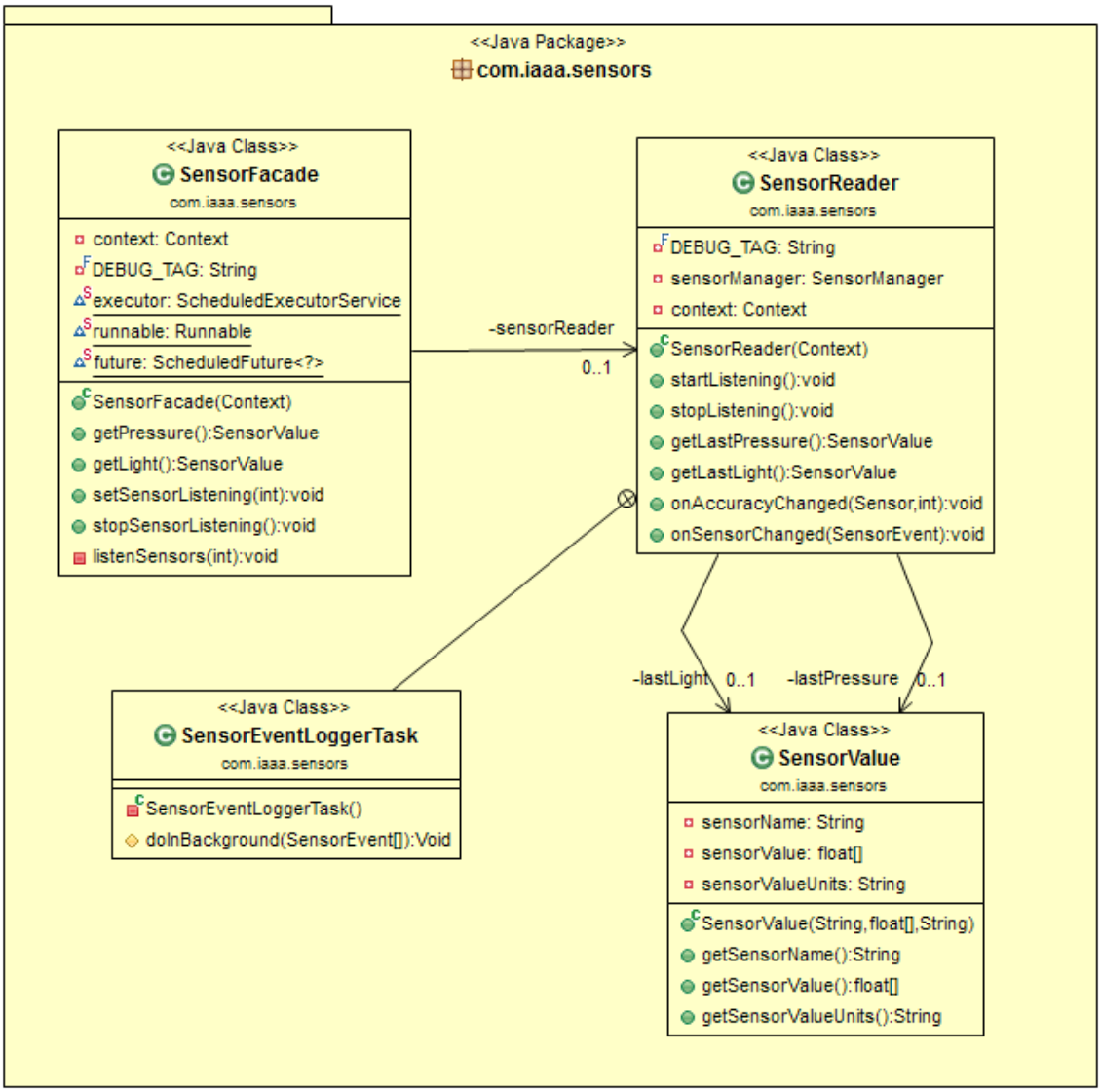


Figura 36. *Diagrama de clases del módulo de sensores*

A continuación se detalla el funcionamiento de cada clase:

- **SensorFacade:** como ya se ha dicho es la fachada que ofrece las operaciones al exterior. Sus principales métodos son:
  - setSensorListening (int interval): empieza a leer los valores de los sensores (todos), actualizando dicha lectura en intervalos, especificados por el parámetro interval.
  - stopSensorListeing(): para de leer los sensores
  - getPressure(): obtiene el objeto SensorValue con el último valor leído por el sensor de presión.
  - getLight():obtiene el objeto SensorValue con el último valor leído por el sensor de luminosidad.

Si se añaden la lectura de más sensores se añade un método get para obtener su valor.

- **SensorReader:** es la clase encargada de leer los valores de los sensores. Implementa la intefaz SensorEventListener. Ofrece métodos para hacer una escucha de valores del sensor y otra para parar la escucha. Cuando un valor de un sensor cambia y

SensorReader escucha el cambio, se lanza en otro thread la clase SensorEventLoggerTask que almacena los valores. Se lanza en otro thread porque podrían cambiar muchos sensores de golpes y ralentizar la aplicación, que, al usuario le parecería que se atasca. Una vez ha leído los cambios de los sensores, se desregistra de dichos cambios y no los vuelve a leer. Es la clase SensorFacade la que vuelve a llamar a SensorReader en un intervalo de tiempo para que vuelva a leer los sensores.

- **SensorValue:** Esta clase sirve para almacenar los valores de cualquier sensor Android. Tiene un atributo para guardar el nombre del sensor leído, otro para guardar las unidades en las cuales se expresa dicho valor, otro atributo de tipo Date para almacenar la fecha de la lectura, y por último, tiene un atributo que es un array de floats para almacenar el valor o valores leídos (dependiendo del sensor puede variar entre 1 a 3 valores).

Por lo tanto, este módulo es capaz de leer los sensores del dispositivo en intervalos de tiempo configurables, para ahorrar energía. En este TFG se han utilizado, como ejemplo, los sensores de luz presión y campo magnético, que se muestran en las gráficas de la aplicación de caso de uso. Sin embargo, el diseño del módulo facilita la incorporación de otros sensores del dispositivo, siendo casi inmediata.

K) Pruebas

Las pruebas realizadas a la aplicación se pueden dividir en pruebas unitarias, pruebas de aceptación y pruebas de integración.

K1) Pruebas unitarias

Las pruebas unitarias de los frameworks se han hecho utilizando el framework de testing JUnit. Se han escrito tests automáticos y se ejecutaban en JUnit para probar los frameworks. Esto se ha podido hacer ya que Android un framework de testing basado en JUnit. Este framework provee de una API para testing, que está basada en la API JUnit y extendida con un framework de instrumentación y clases de test específicas para Android (ver Figura 9).

K1.1) Android Testing API

Como se ha dicho, la API de testing de Android, basada en la API JUnit, además de los ofrecidos por JUnit, provee de instrumentación y clases de test específicas:

JUnit

Es posible utiliza la clase de JUnit TestCase, para probar clases que no utilizan APIs Android. Sin embargo en el TFG no se ha utilizado (excepto en las pruebas del conversor JSON), ya que casi todas las clases utilizan alguna API de Android.

Para ello se ha utilizado la clase AndroidTestCase, que extiende la clase TestCase, y con ella es posible probar objetos con dependencias Android. Además de proveer el framework JUnit, la clase AndroidTestCase, ofrece métodos específicos de setup, teardown y helpers. Esta es la clase con la cual se han probado casi todas las clases del framework, excepto una con instrumentación.

Instrumentation

La instrumentación de Android es un conjunto de métodos de control o “ganchos” del sistema Android. Es decir, a través de ella, se puede controlar el ciclo de vida de los componentes Android y cómo se cargan las aplicaciones. Es posible llamar a los métodos onCreate, onPause, onResume, etc., cuando el programador lo especifique.

Test case classes

Android ofrece varias clases de caso de prueba que extienden TestCase y Assert. Estas clases tienen los métodos específicos de Android de setup, teardown y otros.

Para probar un proyecto, se debe de crear un nuevo proyecto de Test Android, especificando el proyecto que se desea probar. Por ejemplo, en el TFG se ha creado el proyecto de Test que se puede ver en la Figura 37.

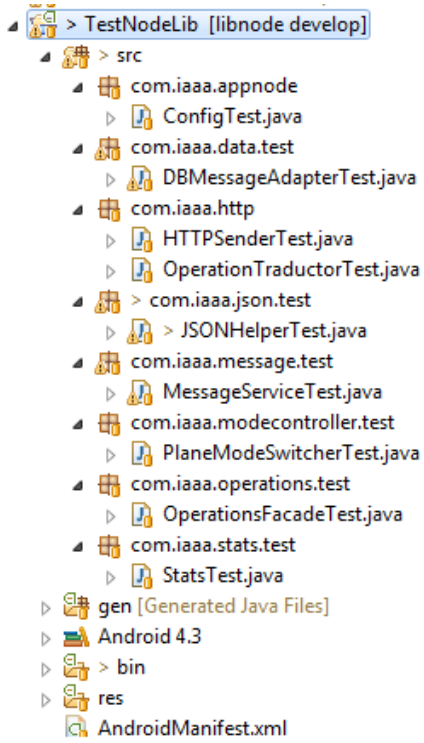


Figura 37. *Proyecto para probar el framework protocolo con JUnit*

Este proyecto de test tiene la misma estructura de paquetes del proyecto que se prueba. Cada clase prueba a una clase del proyecto original, y tienen el mismo nombre añadiendo Test al final. Además, cada método de las clases a probar, es probado en un método en las clases de Test. Tienen el mismo nombre añadiendo Test delante del método.

En la Figura 38 se puede ver, a modo de ejemplo, el código de la clase de Test para la clase JSONHelper (conversor JSON). La clase de prueba extiende TestCase, porque JSONHelper no utiliza las API de Android. Otro ejemplo se muestra en la Figura 39, donde se muestra parte del código de la clase de que prueba a DBMessageAdapter (clase del módulo Data cuya función es ofrecer métodos para almacenar/recuperar SMS en la base de datos). Esta clase de prueba si que extiende AndroidTestCase, porque para hacer las pruebas se utiliza la clase Context de Android, y AndroitTestCase da un objeto Context para realizar las pruebas.

```
public class JSONHelperTest extends TestCase {

	JSONHelper jsonHelper;
	String command = "{\"CMD\":\"LocationGetInfoReq\",\"h\":\"XX\"}";

	protected void setUp() throws Exception {
		super.setUp();
		jsonHelper = new JSONHelper();
	}
	protected void tearDown() throws Exception {
		super.tearDown();
	}

	public void testConvertToJSONObject() {
		JSONObject expected = new JSONObject();
		try {
			expected.put("CMD","LocationGetInfoReq");
			expected.put("h","XX");
		} catch (JSONException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		JSONObject jsonObject = jsonHelper.convertToJSONObject(command);
		assertEquals(expected.toString(), jsonObject.toString());
	}

	public void testOperationToJSONString() {
		String expected =
"{\"CMD\":\"LocationRes\",\"p\":[\"latitude\",\"nsIndicator\",\"longitude\",\"ewIndicator
\",\"utcTime\"],\"h\":\"XX\"}";
		JSONObject jsonExpected = jsonHelper.convertToJSONObject(expected);
		ClasePrueba clasePrueba = new ClasePrueba();
		String jsonString = jsonHelper.operationToJSONString(clasePrueba);
		JSONObject jsonObject = jsonHelper.convertToJSONObject(jsonString);
		assertEquals(jsonObject.toString(), jsonExpected.toString());

	}

	private class ClasePrueba {
		 private String CMD = "LocationRes";
		 private String[] p = {"latitude","nsIndicator","longitude",
"ewIndicator","utcTime"};
		 private String h = "XX";
		 private String opcional = null;
		 ClasePrueba() {
		   // no-args constructor
		 }
	}

	public void testHttpRequestToJSON(){
		String httpRequest = ("CMD=DeviceGetInfoReq&h=XX");
		String jsonRequest = "{\"CMD\":\"DeviceGetInfoReq\",\"h\":\"XX\"}";
		JSONObject expectedJSONObject =
jsonHelper.convertToJSONObject(jsonRequest);
		JSONObject actualJSONObject = jsonHelper.httpRequestToJSON(httpRequest);

		assertEquals(expectedJSONObject.toString(), actualJSONObject.toString());
	}

}
```

Figura 38. *Ejemplo pruebas unitarias con JUnit, módulo conversor JSON*

```
public class DBMessageAdapterTest extends AndroidTestCase {

	private DBMessageAdapter db;

	protected void setUp() throws Exception {
		super.setUp();
		db = new DBMessageAdapter(getContext());
		db.open();
		db.deleteAllMessages(); //borrar si ya había alguno
		boolean sent = false;
		for (int i=0; i<50; i++){ //Insertar valores de prueba
			if(i%2==0){
				sent = true;
			}
			else{
				sent = false;
			}
			db.insertMessage("6284816"+i, "Operacion número: "+ i, sent);
		}
		db.close();
	}

	protected void tearDown() throws Exception {
		super.tearDown();
		db.open();
		db.deleteAllMessages();
		db.close();
	}

	public void testGetAll(){
		db.open();
		Cursor messages = db.getAllMessages();
		int numMensajes = 0;
		if(messages!=null){
			while (messages.moveToNext()) {
				numMensajes++;
			}
			db.close();
		}
		assertEquals(50, numMensajes);
	}

	public void testGetUnsentMessages(){
		db.open();
		Cursor messages = db.getUnsentMessages();
		int numMensajes = 0;
		if(messages!=null){
			while (messages.moveToNext()) {
				numMensajes++;
				if(messages.getInt(4)==1){
					fail("Un mensaje recuperado si que había sido
enviado");
				}
			}
			db.close();
		}
		assertEquals(25, numMensajes);
	}


	[…]
```

Figura 39. *Extracto del código de la clase de prueba del módulo Data (que accede a la BD)*

K1.2) Otras pruebas unitarias

Sin embargo, no todas las pruebas se pueden automatizar con JUnit en Android, puesto que no ofrece instrumentalización para probar clases que extienden BroadcastReceivers, por ejemplo.

Para este tipo de pruebas no automatizables, se ha utilizado una plantilla para hacer pruebas unitarias. Dicha plantilla se utilizó en la asignatura de Verificación y Validación. Consta de una columna con pruebas y otra con resultados. Cada fila representa una prueba, que tiene un título, unas precondiciones, unas acciones para ejecutarla y unos resultados esperados. En la columna de resultados se apunta si se ha pasado cada prueba, y se puede añadir un comentario opcional.

Un ejemplo se puede ver en la Figura 40, que muestra las pruebas unitarias del servicio que controla el apagado y encendido de la red del dispositivo, y otro ejemplo se muestra en la Figura 41, que reflejan las pruebas hechas el módulo de Localización.

Framework protocolo Test unitario
Mode controller

FECHA	19/03/2014
ENCARGADO	Ricardo Pallás

PRUEBAS						RESULTADOS	
Id	Descripción/propósito	Precondiciones/acciones previas	Acciones de ejecución	Resultados esperados		Android 4.3	Comentarios
1. Creación de grupo: casos de prueba para clases de equivalencia válidas							
1.1	Al iniciar por primera vez el programa quita el modo avión si estaba puesto	Se acaba de encender el programa. Se tienen permisos root	Se ejecuta el programa	El modo avión ha sido desactivado si estaba activado previamente		OK	
1.2	Al iniciar por primera vez el programa, toma los valores por defecto de la configuración	Se ejecuta por primera vez el programa	Se ejecuta el programa	Se ve un log indicando los parámetros de configuración		OK	
1.3	Al cambiar el modo de ejecución , se actualizan automáticamente los parámetros de intervalos de tiempo		Se cambia el modo de ejecución manualmente, o los cambia el servidor	El log indica que ha habido un cambio en los intervalos de tiempo		OK	
1.4	Se conecta y desconecta la red de manera periódica, según los intervalos introducidos			Se conecta el modo avión durante un tiempo y se desconecta durante otro tiempo. Esos dos tiempos están fijados por variables.		OK	
1.5	Se cambian los intervalos de conexión y desconexión de red, mientras el programa se está ejecutando		Se cambia el modo de ejecución manualmente, o los cambia el servidor, lo cual invoca al método changeDelay automáticamente	Se conecta el modo avión durante un tiempo y se desconecta durante otro tiempo. Esos dos intervalos han cambiado a los nuevos fijados.		OK	
1.6	Al cerrar el programa se quita el modo avión (si estaba conectado)	Se deben tener permisos root	Se cierra el programa	El modo avión se desconecta.		OK	
1.7	Cuando se desconecta el modo avión, se mandan los mensajes almacenados en la base de datos que no han sido enviados		El modo avión es desconectado por el programa dentro de los intervalos fijados	Se mandan los mensajes que etaban almacenados en la base de datos que no habian sido enviados.		OK	

Figura 40. *Ejemplo plantilla pruebas unitarias: controlador de estado de red*

Framework sensores: Test unitario
Obtener localización

FECHA	18/03/2014
ENCARGADO	Ricardo Pallás

PRUEBAS						RESULTADOS	
Id	Descripción/propósito	Precondiciones/acciones previas	Acciones de ejecución	Resultados esperados		Android 4.3	Comentarios
1. Creación de grupo: casos de prueba para clases de equivalencia válidas							
1.1	Se obtiene la localización del dispositivo en un intervalo dado	Se tiene el GPS encendido y accesible (entorno abierto)	Se llama al método startLocationListening de la clase LocationFacade pasandole un intervalo de refresco en segundos	Se obtiene una nueva localización siguiendo el intervalo de tiempo,		OK	Se pueden ver en el LogCat las actualizaciones de localización y el tiempo transcurrido entre ellas.
1,2	Se obtiene la última localización obtenida del dispositivo	Se tiene el GPS encendido y accesible (entorno abierto)	Se llama al método getLocation de la clase LastLocation	El método devuelve un objeto Location con la última localización obtenida		OK	
2. Creación de grupo: casos de prueba para clases de equivalencia no válidas							
2.1	Obtener la localización con el GPS apagado	El GPS del dispositivo se encuentra apagado.	Se llama al método startLocationListening de la clase LocationFacade pasandole un intervalo de refresco en segundos	No se obtienen nuevas localizaciones		OK	
2.2	Se obtiene la última localización del dispositivo pero nunca se ha obtenido ninguna	Se encuentra el GPS apagado o en entornos cerrados sin acceso al GPS.	Se llama al método getLocation de la clase LastLocation	El método devuelve null		OK	

Figura 41. *Ejemplo plantilla pruebas unitarias: obtener localización*

K2) Pruebas de aceptación

Además de las pruebas unitarias, al final de caca ciclo de desarrollo se han realizado pruebas de aceptación de los requisitos implementados en ese ciclo.

Se han realizado 3 pruebas de aceptación, porque el trabajo se ha dividido en 3 iteraciones. En las pruebas de aceptación se comprobaba la definición de hecho y los criterios de aceptación para cada requisito.

Definición de hecho

Un requisito se considera realizado cuando cumpla las siguientes condiciones:

- Se ha implementado el código con comentarios
- Se han pasado pruebas unitarias con JUnit o con plantilla de pruebas
- Se han pasado las pruebas satisfactoriamente
- Se ha generado documentación Javadoc
- Se ha subido el código al repositorio

Criterios de aceptación

Se cumple la funcionalidad esperada del requisito con un uso de recursos (memoria, tiempo y energía) razonable.

Y cada prueba de aceptación se ha reflejado en la carpeta del proyecto TFG\5_pruebas\Pruebas aceptación, cada una en un fichero Excel separado, cuyo contenido se puede ver en las siguientes tablas (ver Tabla 7, Tabla 8 y Tabla 9).

Requisito	Definición de hecho	Criterios de aceptación
Enviar SMS	OK	OK
Recibir SMS	OK	OK
Almacenar preferencias y configuración	OK	OK
Implementación de todas las operaciones (excepto de localización y sensores)	OK	OK

Tabla 7. *Pruebas de aceptación primera iteración*

Requisito	Definición de hecho	Criterios de aceptación
Obtener localización (por intervalos)	OK	OK
Leer sensores (por intervalos)	OK	OK
Envío de todos los eventos	OK	OK
Operaciones de localización	OK	OK
Estadísticas	OK	OK

Tabla 8. *Pruebas de aceptación segunda iteración*

Requisito	Definición de hecho	Criterios de aceptación
Control de modos de energía	OK	OK
Mejora comunicaciones SMS para no perder SMS cuando no hay red	OK	OK
Base de datos de SMS	OK	OK
Comunicaciones HTTP y notificaciones PUSH	OK	OK
Aplicación caso de uso	OK	OK

Tabla 9. *Pruebas de aceptación tercera iteración*

L) Manual utilización frameworks

En este anexo se explica los parámetros que hay que configurar y las configuraciones necesarias para poder utilizar los frameworks desarrollados en una aplicación Android.

Cuando se ha creado una aplicación Android, lo primero que hay que hacer, es modificar su fichero AndroidManifest.xml, para añadir los permisos necesarios que utiliza el framework, además de declarar sus servicios, actividades y registrar los BroadcastReceivers.

Permisos que hay que añadir:

```
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

<permission android:name="com.iaaa.appnode.gcm.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
<uses-permission android:name="com.iaaa.appnode.gcm.permission.C2D_MESSAGE"
/>
```

Estos permisos son para recibir/enviar SMS, acceder a internet, acceder a la localización del teléfono, leer el estado del teléfono y de la red. Además las notificaciones PUSH requieren los permisos de obtener cuentas, impedir que el teléfono entre en modo inactivo y recibir mensajes PUSH. Por último, se necesitaría el permiso de acceder al teléfono en modo superusuario, pero no es necesario especificarlo para que funcione correctamene.

Después, dentro de las etiquetas <application> del AndroidManifest, se declaran los servicios del framework:

```
<service android:name="com.iaaa.message.MessageService" />
<service android:name="com.iaaa.events.EventControllerService" />
<service android:name="com.iaaa.modecontroller.ModeControllerService" />
<service android:name="com.iaaa.http.HTTPService" />
<service android:name="com.iaaa.http.GcmIntentService" />
```

Más adelante, se declara la actividad de configuraciones(es la pantalla que permite ajustar las configuraciones por parte del usuario), y la de estadísticas, que muestra estadísticas de uso:

```
<activity android:name="com.iaaa.appnode.SettingsActivity" > </activity>
<activity android:name="com.iaaa.stats.StatsActivity" > </activity>
```

Por último, se registran los BroadcastReceivers necesarios para recibir eventos, de tipo SMS y notificaciones PUSH:

```
<receiver
	android:name="com.iaaa.http.GcmBroadcastReceiver"
	android:permission="com.google.android.c2dm.permission.SEND" >
	<intent-filter>
		<action android:name="com.google.android.c2dm.intent.RECEIVE" />
		<category android:name="com.example.gcm" />
	</intent-filter>
</receiver>

<receiver android:name="com.iaaa.message.SMSReceiver" >
	<intent-filter android:priority="1500" >
		<action android:name="android.provider.Telephony.SMS_RECEIVED"
	/>
	</intent-filter>
</receiver>
```

Una vez configurado el fichero AndroidManifest.xml, ya se puede añadir el código en la aplicación que para lanzar los servicios del framework. En este ejemplo, el código se va a colocar en el método onCreate() de la Activity principal de la aplicación Android en la cual se quieren usar los frameworks. Se ha decidido así, porque onCreate() es el método que se llama al crear la Activity, y como es la Activity principal, se llamará al ejecutar la aplicación.

```
private static final int RESULT_SETTINGS = 1; //id preferencias
private SMSSender smsSender; //Para enviar SMS

@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);


	// Check device for Play Services APK.
	if (checkPlayServices()) {
		// If this check succeeds, proceed with normal processing.
		Log.i("APP_TEST", "Google play services: OK");
	}
	else {
		Toast.makeText(this, "Google play services no instalados,
		dispositivo no compatible",Toast.LENGTH_SHORT).show();
	}
	startService(new
	Intent(MainActivity.this,com.iaaa.message.MessageService.class));
	startService(new
	Intent(MainActivity.this,com.iaaa.events.EventControllerService.class)
	);
	startService(new
	Intent(MainActivity.this,com.iaaa.modecontroller.ModeControllerService
	.class));

	smsSender = new SMSSender(this);
}
```

En este código se han lanzado los 3 servicios necesarios y se han comprado, con el método checkPlayServices() si se tenían instalados en el dispositivo los servicios de Google. A continuación se muestra el código de dicho método:

```
private boolean checkPlayServices() {
	int resultCode =
	GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);
	if (resultCode != ConnectionResult.SUCCESS) {
		if (GooglePlayServicesUtil.isUserRecoverableError(resultCode)) {
			GooglePlayServicesUtil.getErrorDialog(resultCode, this,
			PLAY_SERVICES_RESOLUTION_REQUEST).show();
		} else {
			Log.i("APP_TEST", "This device is not supported.");
			finish();
		}
		false;
	}
	return true;
}
```

Esta comprobación es necesaria, porque, tener instalados los servicios de Google es obligatorio para recibir notificaciones PUSH. Google también recomienda hacer la comprobación en el método onResume() de la Activity:

```
@Override
protected void onResume() {
	super.onResume();
	//Se comprueban los servicios de Google
	checkPlayServices();
}
```

Por último, al cerrar la aplicación, Android llama al método onDestory(). Es un buen lugar para parar los servicios del framework:

```
@Override
protected void onDestroy() {
	super.onDestroy();
	//Para los servicios
	stopService(new
	Intent(MainActivity.this,com.iaaa.message.MessageService.class));
	stopService(new
	Intent(MainActivity.this,com.iaaa.events.EventControllerService.class)
	);
	stopService(new
	Intent(MainActivity.this,com.iaaa.modecontroller.ModeControllerService
	.class));
}
```

Con dicho código, la aplicación ya tendría los frameworks funcionado y ya podría ser parte del sistema de tracking. Aparte, el framework protocolo, tiene 2 pantallas o Activitys, que muestran una interfaz para que el usuario pueda configurar los parámetros de configuración, y otra que muestra las estadísticas. Se va a incluir código de ejemplo para mostrar las dos pantallas.

Para mostrar la pantalla de configuración y poder cambiar los parámetros manualmente, se puede llamar a esta función, al pulsar un botón por ejemplo. (*RESULT_SETTINGS* es una constante cuyo valor es 1).

```
public void settingsClick(View v) {
	Intent i = new Intent(this, com.iaaa.appnode.SettingsActivity.class);
	startActivityForResult(i, RESULT_SETTINGS);
}
```

Y una vez cambiado un parámetro, cuando se cierra la ventana de configuraciones, se puede capturar ese cambio en la actividad principal mediante la siguiente función:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
super.onActivityResult(requestCode, resultCode, data);

	switch (requestCode) {
	case RESULT_SETTINGS:
		break;

}
```

El código para mostrar la pantalla de estadísticas es muy similar:

```
public void showStats(View v) {

	Intent intent = new Intent(this, com.iaaa.stats.StatsActivity.class);
	startActivity(intent);

}
```

Este código de ejemplo ha mostrado como utilizar los frameworks en una aplicación Android cualquiera. Como se puede ver, no hace falta escribir ningún código para lanzar el framework sensores porque ya se encarga el framework protocolo, aunque si se ha visto que se han tenido que añadir permisos como el de acceder a la localización del dispositivo.

Otra posibilidad sería lanzar el framework al encenderse el dispositivo y pararlo al apagar el dispositivo. O encenderlo al pulsar un botón y apagarlo con otro botón. Las posibilidades son muchas y queda a la elección del programador cuando iniciarlo y pararlo.

M) Aplicación caso de uso

Como ya se ha adelantado en la sección 2.6, se ha creado una aplicación caso de uso que utiliza los frameworks desarrollados en el TFG. El objetivo principal de esta aplicación es hacer que el dispositivo que la ejecuta se integre dentro del sistema de tracking (pudiendo el usuario iniciar y parar la integración cuando desee, y modificar los parámetros de funcionamiento), y registrar los valores leídos de la localización y sensores del dispositivo, mostrándolos en gráficas y mapas.

La aplicación tiene dos pantallas principales, la de Inicio y la Simulador. Ambas se pueden ver en la **¡Error! No se encuentra el origen de la referencia.** y la Figura 43. Pantalla Simulador,

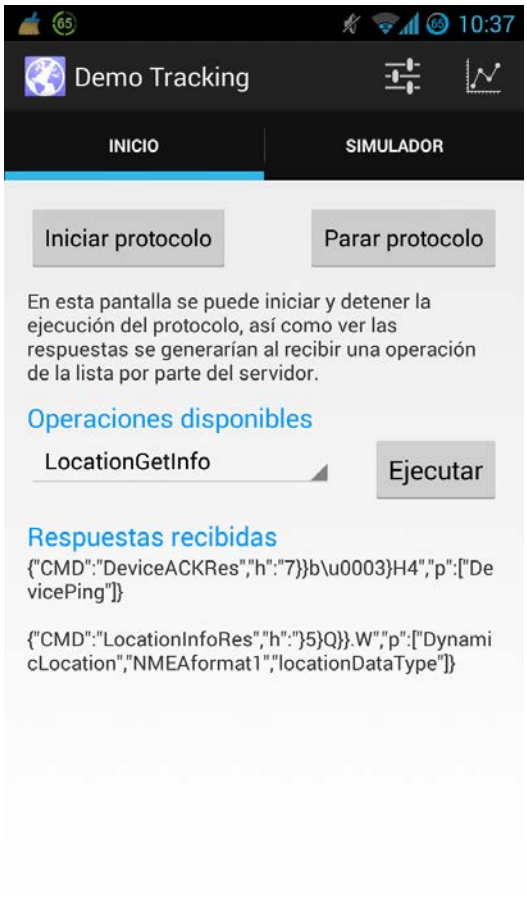


Figura 42. Pantalla Inicio

Figura 43. Pantalla Simulador

En la pantalla Inicio hay dos botones para iniciar y parar la ejecución del protocolo, es decir, que se lancen los servicios creados en los frameworks y el teléfono empiece a actuar como un

nodo dentro del sistema de tracking. También se pueden ver los mensajes que el nodo generaría como respuesta a una petición de operación por parte del servidor.

En la pantalla de Simulador se pueden ver los últimos valores leídos del GPS y de los sensores. Éstos se pueden actualizar pulsando al botón de refresco de la barra de acción. Debajo de los últimos valores hay dos botones “Ver mapa” y “Ver gráfico”. Éstos llevan a la pantalla de mapa y de gráficos, respectivamente.

En la pantalla de mapa se puede ver un mapa de IDEZar con controles de zoom. Dentro del mapa aparecen puntos que reflejan las localizaciones obtenidas del GPS del dispositivo. Estas localizaciones se han obtenido durante el funcionamiento del dispositivo como nodo dentro del sistema de tracking, y el intervalo de obtención depende de los parámetros del protocolo. Los puntos están unidos por una línea roja para mostrar la ruta del dispositivo. Los puntos y la ruta se pueden actualizar pulsando sobre el botón de refresco de la barra de acción de la pantalla de mapa. Esta pantalla se puede ver en Figura 44.

Figura 44. *Pantalla de Mapas, mostrando la ruta recorrida por el dispositivo.*

Para ver la pantalla de gráficos hay que seleccionar un tipo de gráfico y pulsar sobre el botón ver gráfico de la pantalla simulador. Hay un gráfico por cada sensor del dispositivo, y éste muestra la evolución de los valores del sensor en el tiempo, mediante un gráfico de barras interactivo, en el cual se puede hacer zoom y desplazar vertical y horizontalmente. En la Figura 45, Figura 46 y Figura 47 se pueden ver diferentes gráficas para cada tipo de sensor.

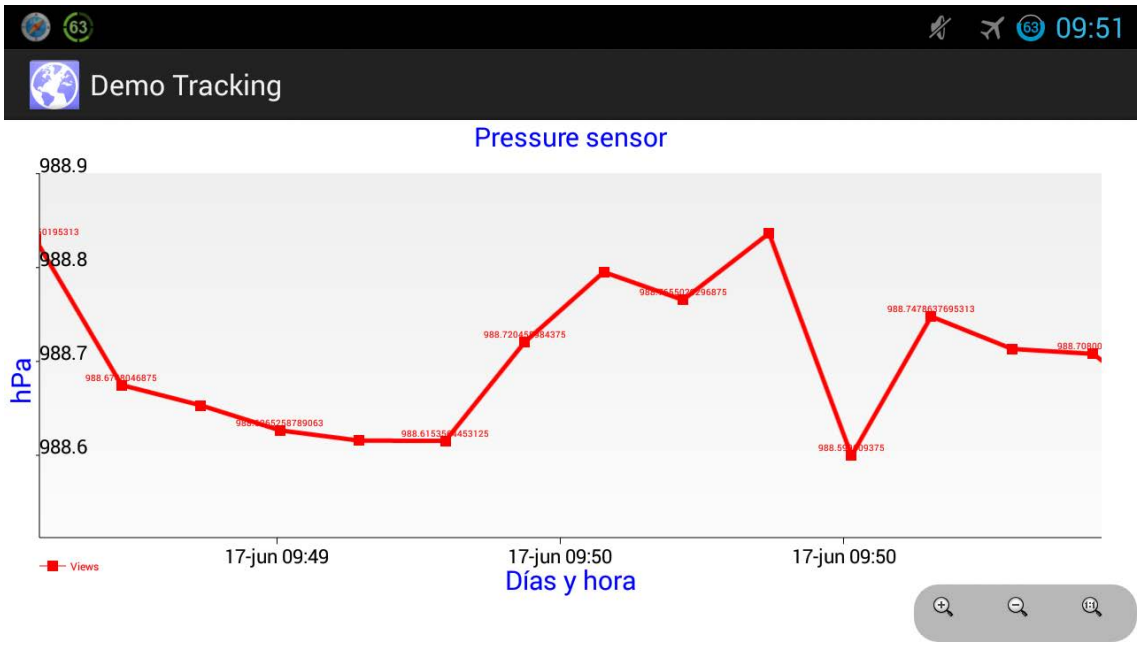


Figura 45. *Gráfica para el sensor de presión*

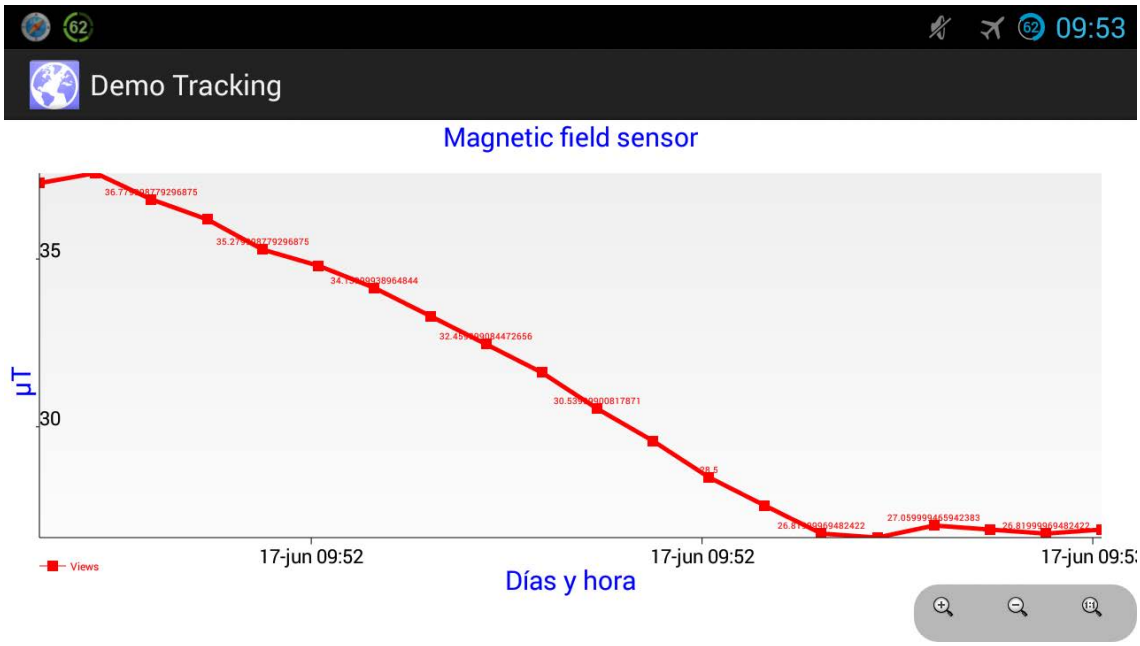


Figura 46. Gráfica para el sensor magnético

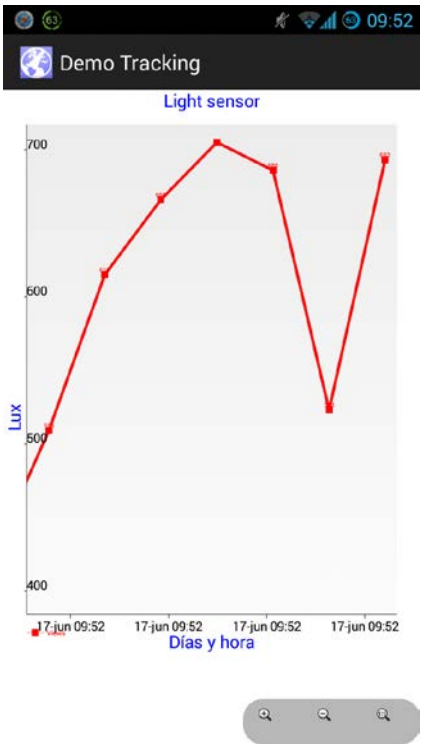


Figura 47. Gráfica para el sensor de luz

Por último, en la barra de acción de las dos pantallas principales de la aplicación se puede acceder a una pantalla de configuraciones y otra de estadísticas. En la pantalla de configuraciones (Figura 48. *Pantalla de configuraciones* y Figura 49) el usuario puede ver y cambiar todos los parámetros del nodo (por ejemplo, su modo de energía o la tasa de obtención de localizaciones).

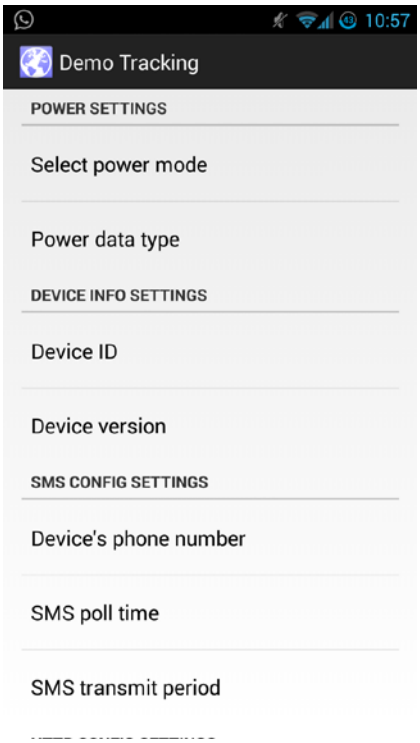


Figura 48. *Pantalla de configuraciones*

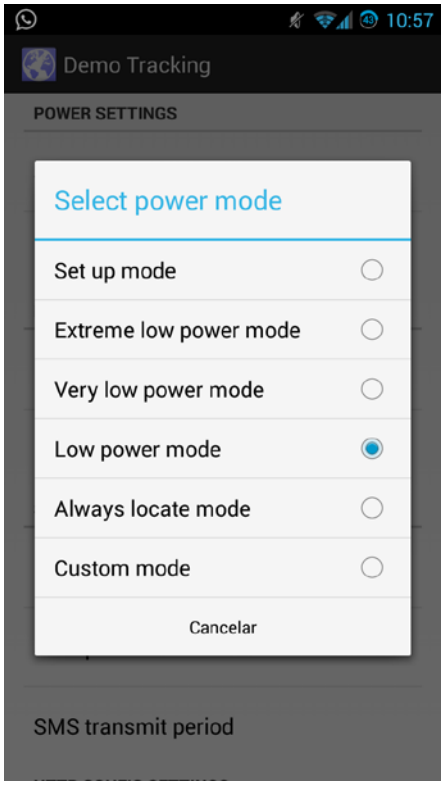


Figura 49. *Ejemplo selección modo de energía*

En la pantalla de estadísticas se muestran el número de mensajes recibidos y enviados, así como el número de operaciones enviadas y recibidas a través de HTTP (Figura 50).

Figura 50. *Pantalla de estadísticas*

N) Gestión del proyecto

En esta sección se detalla la gestión del trabajo de fin de grado realizada. Se va a detallar la planificación inicial del trabajo, la metodología llevada a cabo, la gestión de configuraciones y los esfuerzos reales invertidos.

N1) Fases y actividades del proyecto

En este apartado se enumeran los posibles riegos del proyecto, la planificación inicial y el reparto de tareas.

N1.1) Riesgos

Aquí se presentan los principales riesgos identificados con prioridad, descripción y estrategia de mitigación o contingencia.

1-Dependencias bloqueantes
Tipo: organización.
Prioridad: alta.
Descripción: Problemas de dependencia de otras partes del sistema de tracking que se desarrollan en paralelo. Por ejemplo, el servidor.
Estrategia: Buena comunicación diaria y bidireccional entre los desarrolladores de otras partes para solucionar las dependencias.

2-Cambios en el protocolo
Tipo: organización.
Prioridad: alta.
Descripción: Posibles cambios en el protocolo (formato de datos o interacción) por otras partes del sistema.
Estrategia: Justificación de los cambios, reuniones para informar de los mismos.

3-Problema de versiones
Tipo: configuración.
Prioridad: media-alta.
Descripción: Posibles problemas que puedan surgir de las versiones del software y otros elementos de configuración.
Estrategia: Documentar cada cambio en el repositorio de control de versiones y establecer políticas de nombrado para las copias de seguridad y/o los documentos modificados.

4- Perdida de datos
Tipo: configuración.
Prioridad: media.
Descripción: Problemas que puedan surgir que hagan que los datos se corrompan, borren o no puedan recuperarse.
Estrategia: Copias de seguridad de toda la información de forma semanal o cada vez que se realizan modificaciones importantes.

5- Fallo del dispositivo móvil de prueba
Tipo: configuración.
Prioridad: media.

Descripción: Fallo hardware o software del dispositivo móvil que impide testear y ejecutar el protocolo.
Estrategia: Utilizado emulador de Android en un PC. Simular SMS, coordenadas GPS y sensores mediante Android Debug Bridge (ADB).

N1.2) Estimaciones globales y esfuerzos iniciales

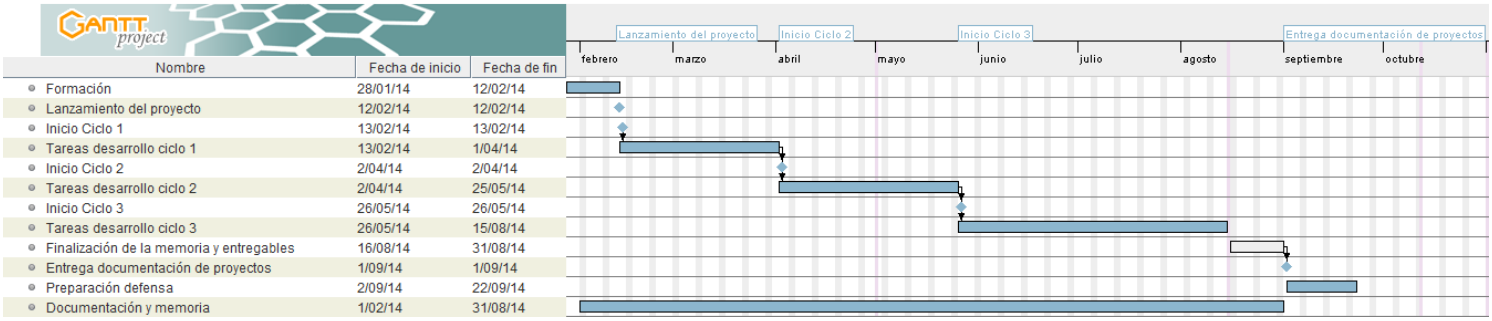
Este trabajo de fin de grado fue propuesto por el grupo IAAA. Se estimó que encajaría en el número de créditos que vale. Es decir, dado que son 12 créditos, el tiempo necesario, son 300 horas. La longitud del trabajo se ajustó a las 300 horas desde el momento que fue propuesto.

Aplicando los indicadores de coste según COCOMO sobre la estimación inicial de 300 horas:

- Experiencia en aplicaciones: 1.13
- Experiencia en el lenguaje de programación: 0.95

$$1.13 \times 0.95 \times 300 = 322.05 \text{ horas}$$

A continuación, en la Figura 51, se puede ver el cronograma y planificación iniciales. Primero, hay una etapa de formación en las tecnologías de 14 días. Después se muestran las fechas y duraciones de los 3 ciclos o sprints de desarrollo. Paralelamente desde el 1 de febrero hasta el 31 de agosto se va haciendo la documentación y memoria del trabajo. Por último, se indica un periodo de finalización de memoria y entregables, día de entrega del trabajo, y preparación de la defensa, que es desde el 2 de septiembre al 22.



Nombre	Fecha de inicio	Fecha de fin
Formación	28/01/14	12/02/14
Lanzamiento del proyecto	12/02/14	12/02/14
Inicio Ciclo 1	13/02/14	13/02/14
Tareas desarrollo ciclo 1	13/02/14	1/04/14
Inicio Ciclo 2	2/04/14	2/04/14
Tareas desarrollo ciclo 2	2/04/14	25/05/14
Inicio Ciclo 3	26/05/14	26/05/14
Tareas desarrollo ciclo 3	26/05/14	15/08/14
Finalización de la memoria y entregables	16/08/14	31/08/14
Entrega documentación de proyectos	1/09/14	1/09/14
Preparación defensa	2/09/14	22/09/14
Documentación y memoria	1/02/14	31/08/14

Figura 51. *Diagrama de Gantt de la planificación inicial del proyecto*

N1.3) División de tareas y estimaciones por ciclo

Se hizo una descomposición del trabajo en tareas más pequeñas de manera que su estimación fuese más sencilla, debido a que estimar tareas grandes a largo plazo es prácticamente imposible.

Siguientemente, se muestra las tareas por ciclo y su estimación inicial:

Ciclo 1 (84:30h)

- Formación: 30h
- Enviar mensajes (SMS)
  - Lógica 3:00h
  - Pruebas 1:00h
  - Documentación 1:00h
- Recibir mensajes (SMS)
  - Lógica 6:00h
  - Pruebas 1:00h
  - Documentación 1:00h
- Codificar/decodificar JSON
  - Lógica 4:00h
  - Pruebas 1:00h
  - Documentación 0:30h
- Device commands
  - Lógica 7:00h
  - Pruebas 1:30h
  - Documentación 1:00h
- Power commands
  - Lógica 2:00h
  - Pruebas 1:00h
  - Documentación 0:30h
- Encapsular framework
  - Lógica 8:00h
  - Pruebas 4:00h
  - Documentación 3:00h
- Guardar configuraciones
  - Lógica 5:00h
  - Pruebas 2:00h
  - Documentación 1:00h

Ciclo 2 (80-90h) (63,5h)

- Estadísticas de uso
  - Registro y almacenamiento 5:00h
  - Pruebas 1:00h
  - Creación pantalla estadísticas 3:00h
- Lectura de sensores
  - Lógica 10:00h
  - Pruebas 2:00h
  - Documentación 1:00h

- Obtener localización GPS
  - Lógica 14:00h
  - Pruebas 2:00h
  - Documentación 1:00h
- Device events
  - Lógica 3:00h
  - Pruebas 1:00h
  - Documentación 0:30h
- Location events
  - Lógica 6:00h
  - Pruebas 2:30h
  - Documentación 1:00h
- Power events
  - Lógica 2:00h
  - Pruebas 1:00h
  - Documentación 0:30h
- Location commands
  - Lógica 4:00h
  - Pruebas 2:00h
  - Documentación 1:00h

Ciclo 3 (85h)

- Base de datos
  - Modelo 0:30h
  - Lógica 3:00h
  - Pruebas y documentación 1:30h
- Configuración SMS según modo energía
  - Lógica 8:00h
  - Pruebas y documentación 2:00h
- Gestión de errores protocolo 2:00h
- Traductor JSON/ HTTP del protocolo
  - Lógica 2:00h
  - Pruebas y documentación 1:00h
- Almacenamiento de mensajes
  - Reintentar enviar mensajes no enviados 2:00
  - Lógica 2:00h
  - Pruebas y documentación 1:00h
- Gestión modos energía
  - Activar/desactivar modo avión 5:00h
  - Lógica control energía 15:00h
- Enviar recibir HTTP
  - Enviar 5:00h
  - Recibir PUSH 10:00h
- App caso de uso 25:00h

Todos los ciclos

- Memoria: 80h
- Planificación: 1:00h
- Reuniones 1:00h
- Gestión 15:00h

Además de las tareas de desarrollo, también hay que contar el esfuerzo de planificación y gestión.

N1.4) Esfuerzos reales de tareas por ciclo

Siguientemente, se muestra las tareas por ciclo y los esfuerzos reales de cada una. Las tareas de documentación de cada tarea no tenían sentido controlarlas por separado por la poca utilidad en que tendría en proyectos futuros. La suma de todas se ha incluido en la tarea de memoria.

Ciclo 1 (75h + tareas todos los ciclos)

- Enviar mensajes (SMS)
  - Lógica 1:30h
  - Pruebas 2:00h
- Recibir mensajes (SMS)
  - Lógica 2:00h
  - Pruebas 1:30h
- Codificar/decodificar JSON
  - Lógica 1:30h
  - Pruebas 1:00h
- Device commands
  - Lógica 5:30h
  - Pruebas 0:00h
- Power commands
  - Lógica 3:00h
  - Pruebas 0:30h
- Encapsular framework
  - Lógica 0:30h
  - Pruebas 0:30h
- Guardar configuraciones
  - Lógica 1:00h
  - Pruebas 1:30h
- Formación: 53:00h

Ciclo 2 (36h + tareas todos los ciclos)

- Estadísticas de uso
  - Registro y almacenamiento 2:00h
  - Pruebas 0:30h
  - Creación pantalla estadísticas 0:30h
- Lectura de sensores
  - Lógica 3:30h
  - Pruebas 0:30h
- Obtener localización GPS
  - Lógica 4:30h
  - Pruebas 0:30h
- Device events
  - Lógica 2:30h
  - Pruebas 0:00h
- Location events
  - Lógica 8:30h
  - Pruebas 0:30h
- Power events
  - Lógica 3:00h
  - Pruebas 0:30h
- Location commands
  - Lógica 8:30h
  - Pruebas 1:00h

Ciclo 3 (80h + tareas todos los ciclos)

- Base de datos
  - Modelo 2:00h
  - Lógica 1:00h
  - Pruebas 1:00h
- Configuración SMS según modo energía
  - Lógica 3:00h
  - Pruebas 1:00h
- Gestión de errores protocolo 3:00h
- Traductor JSON/ HTTP del protocolo
  - Lógica 3:30h
  - Pruebas 1:30h
- Almacenamiento de mensajes
  - Reintentar enviar mensajes no enviados 4:00h
  - Lógica 1:00h
  - Pruebas 1:00h
- Gestión modos energía
  - Activar/desactivar modo avión 5:00h
  - Lógica control energía 15:00h
- Enviar recibir HTTP
  - Enviar 1:30h
  - Recibir PUSH 6:30h

- App caso de uso 30:00h

Todos los ciclos (111:30h)

- Memoria: 107:00h
- Planificación: 1:00h
- Reuniones 0:30h
- Gestión 10:00h

El segundo ciclo tuvo menos carga de trabajo debido a los trabajos de otras asignaturas, cuyas entregas coincidían y no se tuvo tanto tiempo. Además de las 36 horas reales de desarrollo habría que incluir 40 horas de memoria que se hicieron durante dicho ciclo.

N1.5) Fichero de esfuerzos

Los esfuerzos se han ido recopilando en una hoza Excel, que sigue una plantilla estándar del IAAA. Cada 30 minutos de esfuerzo se refleja en la plantilla como una unidad.

A continuación, en la Figura 52, se muestra una gráfica con los esfuerzos reales de desarrollo y formación, recopilados en el fichero de esfuerzos.

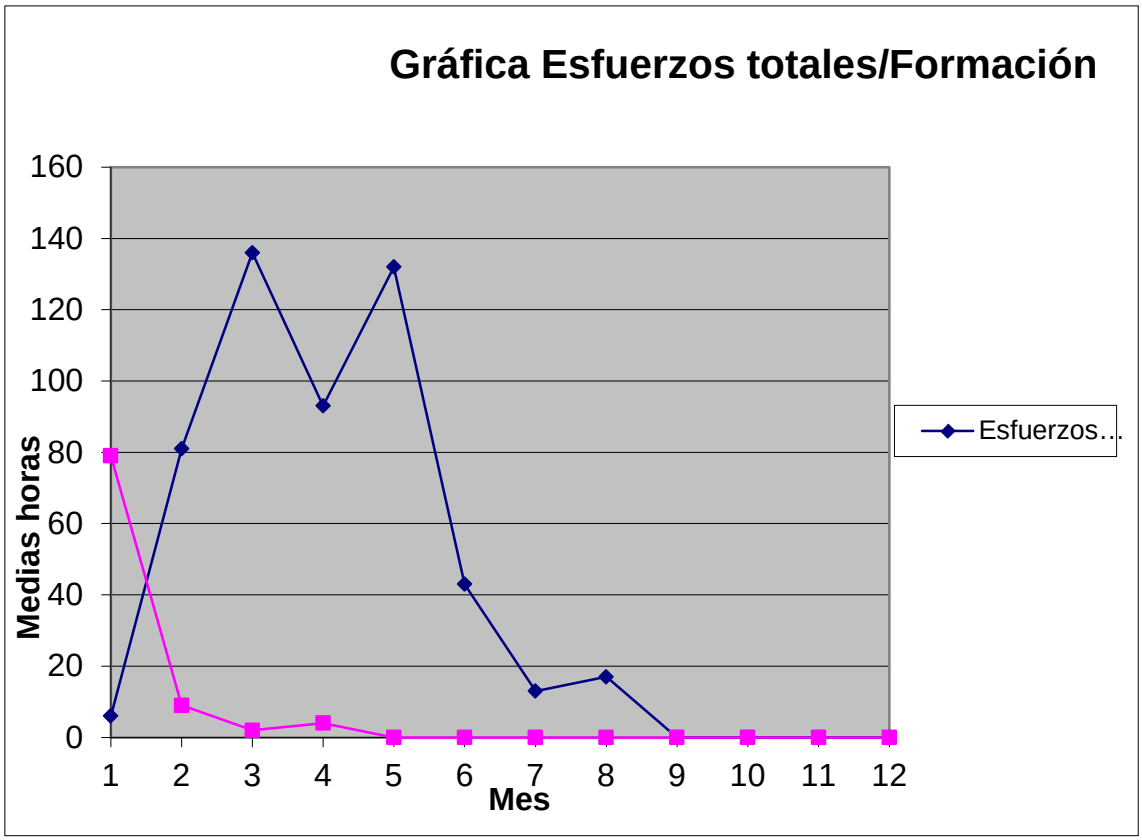


Figura 52. *Gráfica esfuerzos totales/formación*

Los esfuerzos totales de formación han sido de 47 horas, mientras que desarrollo, gestión y memoria han sido de 260,5 horas. Es decir, todo el esfuerzo ha sido de 317,5 horas (la estimación inicial con COCOMO fueron de 322,05).

N1.6) Procesos de seguimiento y control

En este apartado se muestran las gráficas de burndown de esfuerzos y tareas de cada ciclo. Cada punto de esfuerzo corresponde con 30 minutos de trabajo. En las gráficas de tareas, cada unidad es una tarea del ciclo.

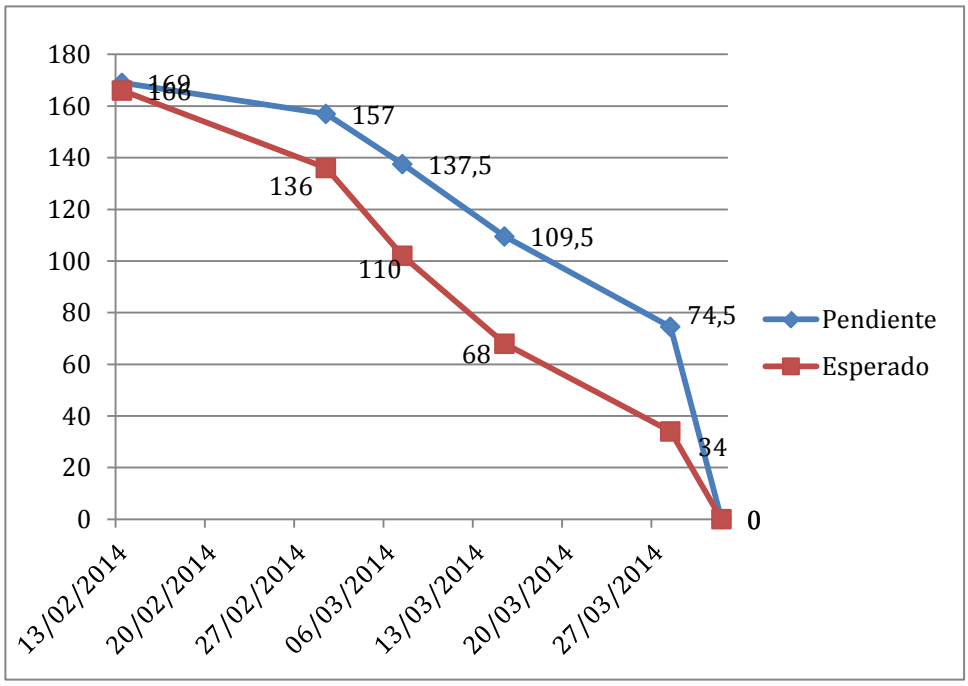


Figura 53. *Gráfica burndown esfuerzos ciclo 1*

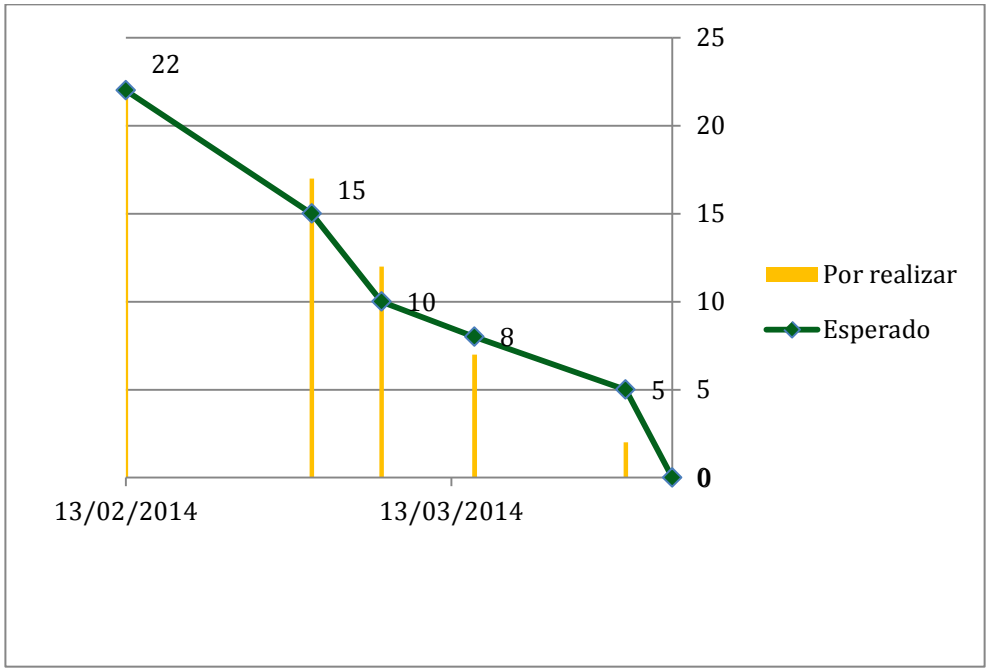


Figura 54. *Gráfica burndown tareas ciclo 1*

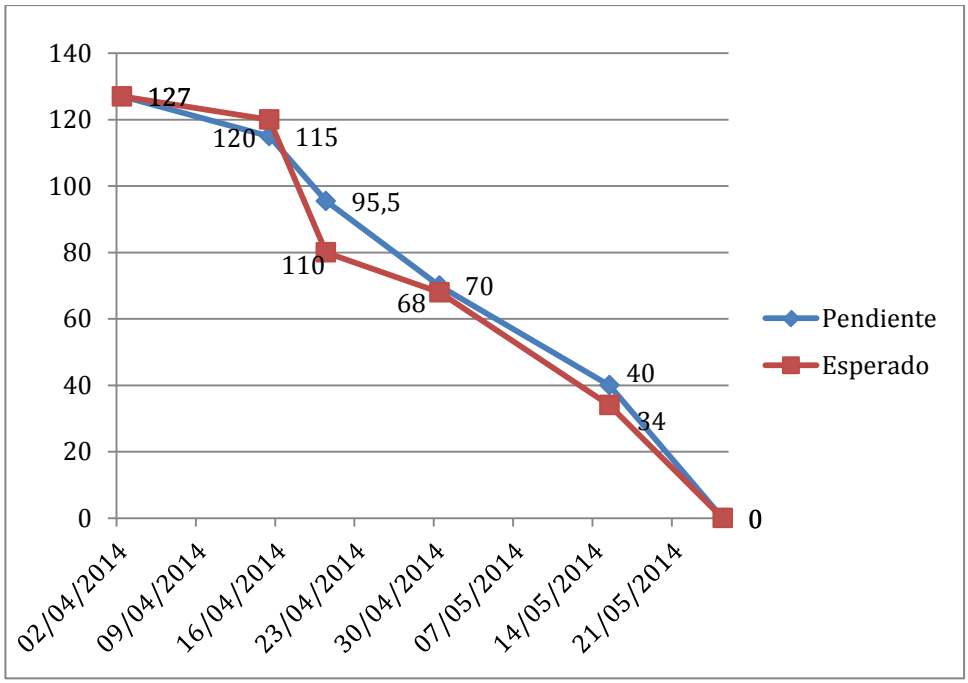


Figura 55. *Gráfica burndown esfuerzos ciclo2*

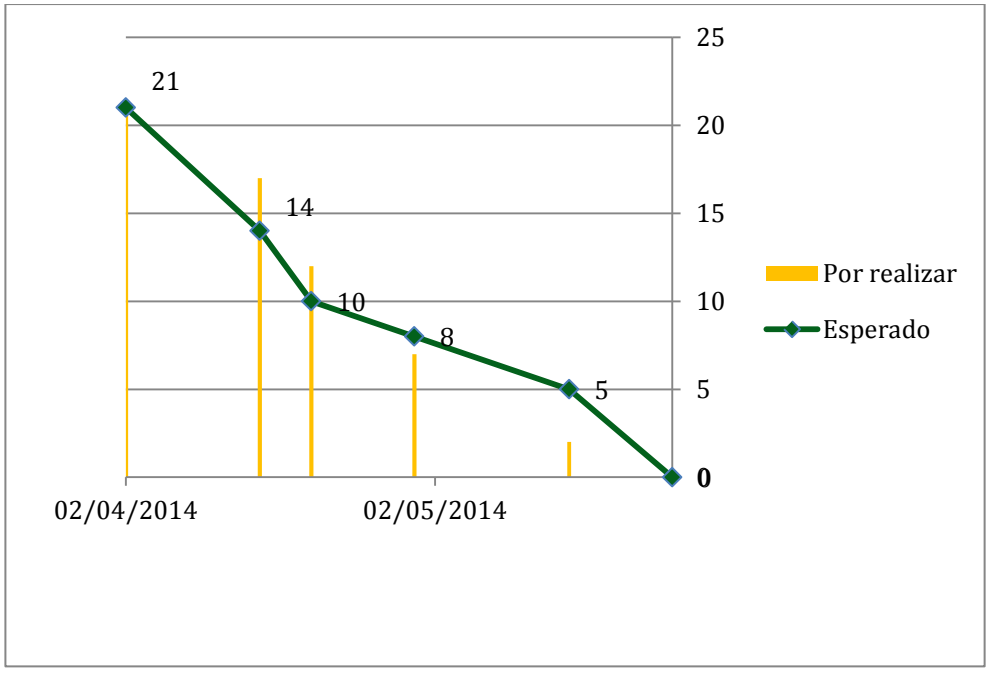


Figura 56. *Gráfica burndown tareas ciclo 2*

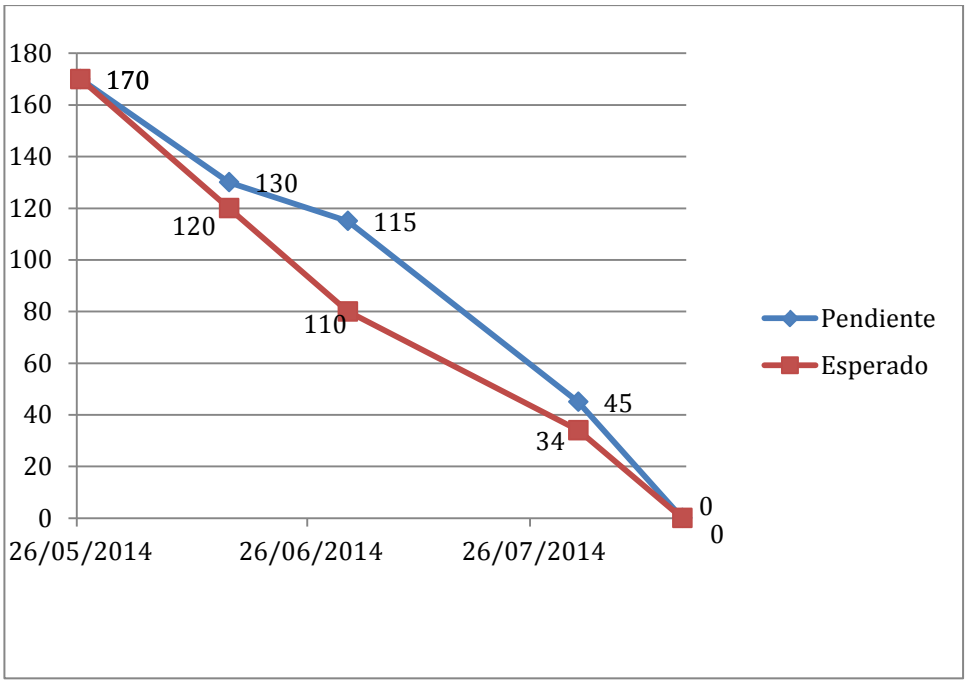


Figura 57. *Gráfica burndwon esfuerzos ciclo 3*

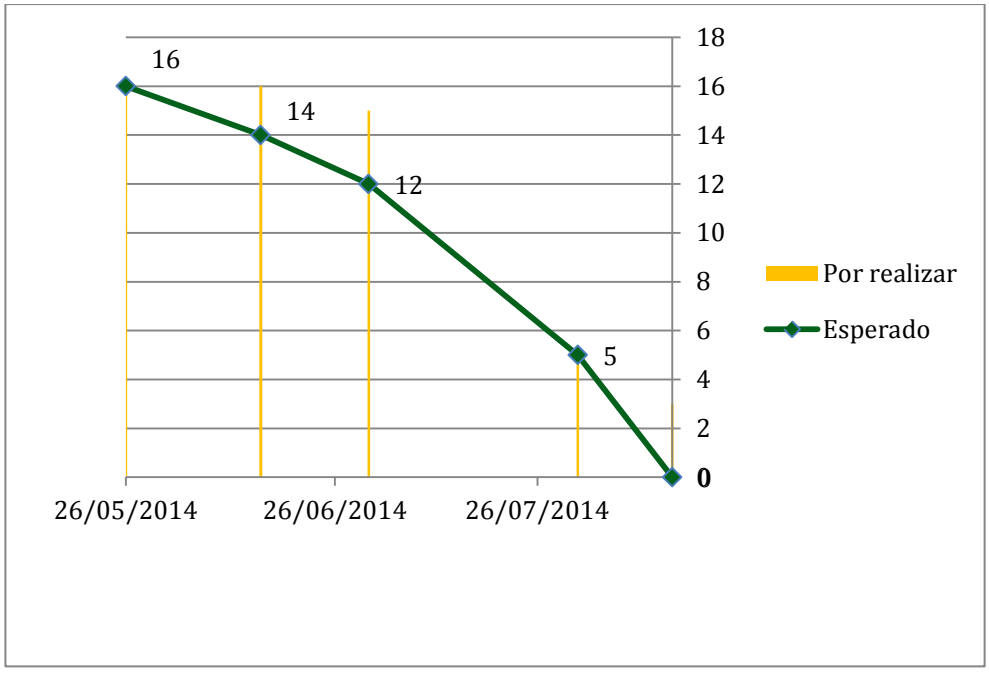


Figura 58. *Gráfica burndwon tareas ciclo 3*

N2) Gestión de configuraciones

Se ha llevado a cabo una gestión de configuraciones similar a las llevadas en asignaturas como Proyecto Software, Gestión de Proyecto Software o Laboratorio del Software, con las cuales se tiene experiencia y, además, dieron buen resultado.

N2.1) Infraestructura y puesta en marcha

Las herramientas y software de desarrollo que se han usado en el TFG son de software libre en la mayoría, excepto dos, que se han utilizado en el ordenador laboratorio con licencias compradas por el grupo IAAA, y Microsoft Office con una licencia de estudiante.

Estas herramientas son:

- **Eclipse:** con la versión 7 de Java utilizado como entorno de desarrollo.
- **Android SDK:** utilizado para el desarrollo con Android, e integrado con Eclipse.
- **Drivers Samsung:** para utilizar un dispositivo Samsung con Android, para ejecutar, depurar y testear el código desarrollado.
- **BitBucket:** repositorio privado GIT para el control de versiones.
- **Microsoft Visio:** utilizado en la elaboración de diagramas informales
- **Enterprise Architect:** utilizado en la elaboración de diagramas formales UML.
- **Micosoft Office 2010:** utilizado en la redacción y elaboración de la memoria y presentación.
- **Tortoise SVN:** utilizado para el control de versiones en el repositorio SVN del grupo IAAA.

N2.2) Formatos y estándares propios utilizados

Para una mayor compatibilidad posible, coherencia en el proyecto y para evitar problemas futuros, se definieron los siguientes estándares a utilizar durante todo el trabajo:

- Codificación de los ficheros en UTF-8
- Código fuente en inglés
- Código fuente formateado, con comentarios auto-explicativos y con formato JavaDoc.
- Código fuente siguiendo las pautas de estilo y de buenas prácticas de Java[3].

N2.3) Control de versiones

Para conseguir un buen nivel de control sobre las versiones de los elementos de configuración del proyecto se ha empleado un sistema GIT, en concreto, se ha utilizado un repositorio privado GIT. Además se ha utilizado también un sistema de control de versiones SVN del grupo IAAA.

[3] http://javapractices.com

N2.4) Entregables, elementos de configuración

En este proyecto, se ha decidido que los entregables y elementos de configuración son:

- El código fuente de la aplicación y frameworks.
- Un fichero APK con la aplicación caso de uso y frameworks empaquetados.
- Memoria

Por lo que se ha llevado un riguroso control de versiones sobre ellos.

N3) Aseguramiento de la calidad

Para mantener el orden, calidad y coherencia de todo el proyecto se ha decidido seguir una serie de estándares o buenas prácticas a lo largo de todo el trabajo realizado.

N3.1) Estándar de codificación de esfuerzos

Se han contabilizado los esfuerzos dedicados al TFG siguiendo la metodología del IAAA (utilizada previamente en asignaturas de Ingeniería del Software). Consiste en rellenar diariamente unas hojas de recopilación siguiendo el siguiente estándar, cada tarea en la cual se ha dividido el proyecto tiene una codificación:

4 códigos de 2 dígitos

85 - fijo para todas las tareas del proyecto de la asignatura
13 - número de equipo, fijo para todas las tareas del equipo
ff - primer nivel de tarea
nn - segundo nivel de tarea

Primer nivel de tarea (ff)

45 – Gestión
46 – Formación
47 – Desarrollo
50 – Integración
60 – Cliente
99 – Otros

Códigos fijos de tareas de gestión (45.nn)

10 – Tareas de gestión
12 – Dirección del proyecto
13 – Gestión de planificación
14 – Gestión de configuraciones
15 – Gestión de desarrollo
16 – Calidad

Códigos fijos de realización de tareas (47.nn)

Cada tarea un código 47.nn, siendo nn un número del 0 al 99.

3.3.2. Políticas de nombrado

El nombrado de los ficheros en general ha seguido el siguiente patrón:

- e13-AAAA
- AAAA es un nombre elegido descriptivo y fácil de entender
- AAAA debe comenzar por una letra mayúscula
- AAAA debe tener las palabras separadas por el carácter _

El nombrado de los ficheros de esfuerzos ha seguido el siguiente patrón:

- 282_Libro_recopilacion_esfuerzos_2014.xls

3.3.3. Estructura del directorio del proyecto

El directorio raíz del proyecto tiene la siguiente estructura:

- 1_gestion
- 2_A&D
- 3_documentacion
- 4_software
- 5_pruebas
- Z_cosas

Bibliografía

Documentación online
http://www.smartposition.nl/resources/sms_pdu.html
https://stackoverflow.com/
http://developer.android.com/reference/packages.html
http://developer.android.com/guide/index.html
http://developer.android.com/google/gcm/index.html
http://www.journaldev.com/1624/command-design-pattern-in-java-example-tutorial
http://docs.oracle.com/javase/7/docs/api/
http://www.javapractices.com

Libros
Beginning Android 4 Application Development (http://isbn.directory/book/978-1-1181-9954-1)
Android Application Testing Guide (http://isbn.directory/book/978-1849513500)
Applying UML and Patterns (http://isbn.directory/book/978-0137488803)
Documenting Software Architectures (http://isbn.directory/book/978-0321552686)

Acrónimos

HTTP: Hypertext transfer protocol.
SMS: Short Message Service.
PDU: Protocol Data Unit
GPS: Global Positioning System.
GCM: Google Cloud Messaging
JSON: JavaScript Object Notation
GSL: GeoSpatiumLab
GSM: Global System for Mobile communications
GPRS: General Packet Radio Service
3G: Tercera generación (de transmission de voz y datos a través de UTMS)
UTMS: Universal Mobile Telecommunications System
XML: eXtensible Markup Language
APP: Application
IAAA: Grupo de Sistemas de Información Avanzados
DB: Database
TFG: Trabajo Fin de Grado

También cabe destacar las siguientes definiciones:

Entorno online: Áreas con cobertura 3G.
Entorno offline: Áreas sin cobertura 3G.
Servidor: Una instancia de un servicio. Un servidor invoca operaciones de un nodo.
Servicio: Parte de la funcionalidad que es ofrecida por una entidad a través de interfaces.
Nodo: dispositivo que envía información (de sensores y geográfica) a un servidor.

Índice de figuras

Índice de tablas