UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja
Máster universitario en Dirección e Ingeniería de Sitios Web
Rama Investigación

Sistema colaborativo de notificaciones integrado en ordenador de a bordo

Trabajo Fin de Máster

presentado por: García Fernández, Iván

Director/a: Adolfo Fuente, Aquilino Juan

Ciudad: Madrid
Fecha: 5/1/2014

Resumen

En la actualidad el mercado automovilístico mundial evoluciona a un ritmo muy alto, todos los años hay varios salones del automóvil; el salón Detroit, el salón de Ginebra, el salón de Tokio, el salón de Madrid, el salón de Frankfurt, el salón de París y el salón de Barcelona. En estos salones los fabricantes presentan públicamente nuevos modelos, nuevos prototipos, nuevas tecnologías, nuevas versiones, restylings, ediciones limitadas de vehículos cuyo objetivo es tentar al consumidor y generar nuevas necesidades para tratar de convencer a los clientes de que su vehículo esta obsoleto y que por consiguiente necesita uno nuevo con las ultimas novedades del mercado.

Durante el año 2014, presenté en la *Escuela de Ingeniería Informática de la Universidad de Oviedo* como trabajo final de carrera para la obtención de la titulación: Ingeniería Técnica en informática de gestión, un proyecto final de carrera titulado "*Implementación de un sistema de medición basado en microprocesador para vehículo antiguo*" o lo que es lo mismo, un ordenador de a bordo para un Citroën 2cv basado en las plataformas *Raspberry pi* y *Arduino*, con el objetivo de adaptar las nuevas tecnologías a un vehículo que podría considerarse completamente obsoleto por su longevidad.

La plataforma *Raspberry pi* y *Arduino* proporcionan una sólida base sobre la que desarrollar ideas relacionadas con el Internet de las cosas, por tanto, tomando como base el proyecto presentado en la universidad, se decidió evolucionarlo para aprovechar la conectividad a Internet e investigar una funcionalidad enmarcada en el Internet de las cosas. Por tanto, en este proyecto, la idea que se pretende investigar es una nueva funcionalidad enmarcada en el internet de las cosas e implementada sobre una arquitectura orientada a servicios, y que el ordenador de a bordo del automóvil pueda integrar esta nueva funcionalidad. El concepto que se ha pensado para este proyecto, es el de permitir al usuario de notificar y ser notificado de situaciones excepcionales en las vías de circulación. Además, los usuarios podrán visualizar los problemas existentes a través de una interfaz de web con mapas, y utilizar dicha interfaz para notificar incidencias. Este prototipo de sistema estará formado por los siguientes componentes:

- *Cliente integrado en el ordenador de a bordo*. Su objetivo es realizar peticiones a un servidor mediante servicios Web para informar de situaciones que el usuario

considere interesantes de notificar y notificar de las incidencias más cercanas al conductor.

- *Cliente Web*. Su objetivo visualizar los problemas que estén surgiendo en las vías de circulación en tiempo real. A través del uso de la *API* de mapas de *Google Maps*. Además puede comunicarse con el Servidor mediante servicios Web para informar de situaciones que el usuario considere interesantes de notificar.
- *Servicios Web*. Sus objetivos son consultar y dar de alta situaciones anómalas.

Palabras Clave

Raspberry Pi, Arduino, Python, Automóvil, ordenador de a bordo, Internet de las cosas, Servicios Web

Abstract

Currently the global automotive market evolves a very high evolution, every year there are several auto shows; Detroit, Geneva, Tokyo, Madrid, Frankfurt, Paris and Barcelona Motor shows are one example. In these Motor shows, manufacturers publicly new models, new prototypes, new technologies, new versions, restylings, vehicle's limited editions to try convince the customers that your vehicle is obsolete and that they need to buy a new vehicle to get the updates.

During 2014, I presented in the Computer Science University of Oviedo as final thesis for obtaining the degree in Computer Engineering management, a final year project entitled "Implementation of a measurement system based on microprocessor for old vehicle "or what is the same, a trip computer for a Citroën 2CV based in Raspberry Pi and Arduino platforms, in order to adapt new technologies in a vehicle that could be considered completely obsolete by its longevity.

The Raspberry Pi and Arduino platform provide a solid base on which develop Internet of things (IoT), therefore, based on the project presented at the university, it was decided to use the connectivity to Internet and develop a functionality base on the Internet of things . This field is new and growing very fast and it as many possibilities. Companies like Google decided that the operating System Android should be provided and added into vehicles to provide more value into the products. Both companies, Google and Automotive Sector know that the success of their systems depends on the work of the large community of software developers, software engineers and users.

In this case the idea is to develop a prototype for a car, carrying the integrated ability to report exceptional situations on the road, and also if the vehicle is equipped with GPS and maps, can be accessed at the problem records on the roads, if not available users can also display the problems through a Web interface maps via mobile connection.

Therefore, the goal to be achieved is to develop a system consisting of the following components:

- Customer integrated into the on-boarding computer of the vehicle. The objective is to communicate to a server using Web Services to report situations where the driver considers interesting to notify and notify any driver incidents
- Web Client. The objective is to visualize the problems that are emerging on the road in real time. Through the use of Google Maps API. Driver can also communicate to a server through services Web to report situations that the user considers interesting to notify.
- Web Services. The objective are to consult and high anomalous situations.

Keywords

Raspberry Pi, Arduino, Python, automobile, onboard computer, Internet of things, Web Services

Índice de contenidos

Índice de Figuras

Capítulo 1. Introducción

1.1 Resumen de la motivación y alcance del proyecto

En este apartado se tratarán varios aspectos, por una parte se comentará la motivación para la realización de este proyecto de investigación y por otra parte se acotará el alcance de la idea que se tiene intención de investigar y desarrollar.

En primer lugar esta idea tiene su origen en las lineas de investigación y desarrollo surgidas durante la realización del trabajo final de carrera que se toma como referencia para la evolución del ordenador de a bordo que se plantea en este trabajo de investigación, entre estas lineas de investigación, destacan las relacionadas con el desarrollo de nuevas funcionalidades basadas en la conectividad a internet, esta conexión abre un abanico de posibilidades enorme y la posibilidad de aplicación del paradigma del internet de las cosas en los automóviles. En este proyecto, se explorará una posible aplicación novedosa de este paradigma bajo una arquitectura orientada a servicios.

En segundo lugar, mis planes de futuro, mi idea original al cursar este máster de investigación es la de poder dedicarme paralelamente a mi vida profesional a la vía investigadora y poder realizar en un futuro un doctorado en la universidad de Oviedo, por lo que decidí que el área de la informática que más me llamaba la atención es precisamente la del internet de las cosas y concretando aún más, su aplicación en el ámbito de la automoción.

La idea principal que se pretende desarrollar, es un paradigma de comunicación entre vehículos, que debe permitir a los usuarios registrar y visualizar incidencias en las vías de circulación, para ello se desarrollan tres componentes software claramente diferenciados:

1. ***Servicios Web.*** Esta pieza es fundamental, debe permitir a los diferentes clientes que se desarrollen, consultar las incidencias y dar de alta incidencias en las vías de circulación de entre los tipos de incidencia que se ha decidido incluir en este proyecto (mantenimientos, accidente, retención y otros).
2. ***Ampliación del ordenador de a bordo***. Este componente consiste en ampliar el software del proyecto de ordenador de a bordo desarrollado como proyecto final de carrera. Esta ampliación informará al usuario de las incidencias cercanas y además permitirá a los usuarios notificar con las incidencias que detecten a los demás usuarios, mediante la utilización servicios web.

3- Cliente Web. Este componente software consiste en un cliente web, que permita a los usuarios visualizar en un mapa las distintas incidencias e incluso dar de alta nuevas incidencias.

1.2 Organización del documento

La organización de este trabajo se ha realizado en los once capítulos siguientes:

1- Introducción. En este capítulo se habla de la motivación para la realización del proyecto, se expone el alcance de la idea y se explica la organización de la memoria del proyecto.

2- Objetivos del proyecto*.* En este capítulo se definen los objetivos planteados para este trabajo final del máster en dirección e ingeniería de sitios web.

3- Estado actual de los contenidos científicos-técnicos*.* En este capítulo se hace un resumen del estado del arte y de las principales tendencias del mercado de automoción.

4- Planificación. En este capítulo se realiza la planificación temporal para el proyecto y una estimación en horas de la realización de este trabajo.

5- Análisis. En este capítulo se detalla el análisis funcional del sistema que se desea implementar.

6- Diseño del Sistema. En este capítulo y tomando como base el análisis funcional realizado en el capítulo anterior “Análisis”, se desarrollara el diseño técnico del sistema que deseamos implementar.

7- Implementación del Sistema*.* En este capítulo se documenta la fase de codificación, implementación y desarrollo del diseño técnico planteado en el capítulo “Diseño del sistema”.

8- Evaluación*.* En este capítulo se desarrollan las pruebas realizadas.

9- Conclusiones. En este capítulo se reflexiona acerca del proyecto y las posibles ampliaciones que podrían implementarse en futuras versiones.

10-Referencias Bibliográficas.

11-Apéndices.

Capítulo 2. Objetivos del Proyecto

2.1 Objetivos del Proyecto

La aplicación del paradigma del internet de las cosas en el campo de la automoción abre un gran abanico de posibilidades a los investigadores, en todo trabajo de investigación es muy importante y necesario acortar el alcance a una serie de objetivos, estos objetivos servirán para dar respuesta a la investigación. En este capítulo se expondrán los principales objetivos identificados:

- Definición de un paradigma de comunicación vehículo a vehículo utilizando arquitectura orientada a servicios y tecnologías web, y como ejemplo de aplicación de este paradigma, definición de un prototipo de sistema colaborativo que permita a los usuarios informar y ser informados de las diferentes incidencias que puedan surgir en las vías de circulación.
- Desarrollar un conjunto de *servicios Web* como parte del prototipo, que permitan desarrollar clientes y aprovechar los datos, tanto para nuestros prototipos como para las diferentes plataformas que se han presentado como el futuro inmediato de los vehículos conectados: *Apple Carplay*, *Android Auto* y *Windows in the car*, con independencia del cliente que los utilice.
- Desarrollar un prototipo de ordenador de a bordo, como ejemplo de integración de nuestro paradigma en un vehículo, tomando como base el ordenador de a bordo para *Citroën 2cv* diseñado como proyecto final de carrera titulado: “Implementación de un sistema de medición basado en microprocesador para vehículo antiguo”.
- Desarrollar un *cliente Web* con estándares Web, que permita la utilización de los servicios Web y por tanto forme parte del sistema colaborativo.
- Demostrar que es posible desarrollar funcionalidades nuevas enmarcadas en el *Internet de las cosas* para vehículos que podríamos calificar de obsoletos con la tecnología actual, y que por tanto se podrían diseñar productos que se adapten a este tipo de vehículos.

Capítulo 3. Estado actual de los conocimientos científico-técnicos

3.1 Aspectos Teóricos

3.1.1 Internet de las cosas

El concepto Internet de las cosas, fue propuesto por *Kevin Ashton* en el *MIT* (*Massachusetts Institute of Tecnology*) 1999[1], concretamente en el *Auto-ID Center*, en donde se realizaba investigaciones relacionadas con la identificación por radiofrecuencia en red y tecnologías de sensores. Este concepto de Internet de las cosas hace referencia a la interconexión digital de objetos cotidianos con internet (*Gaglio, Salvatore, Lo Re, Giuseppe (Eds.) , 2013*), también se utiliza con la denotación de conexión de dispositivos avanzada (*Nel Gershenfeld, Raffi Krikorian and Danny Cohene, 2004*), que supera el concepto tradicional maquina a maquina, y que cubre una gran variedad de dominios y protocolos. Las estimaciones de diferentes consultoras (Gartner[2] y Abi Research[3]) es que el número de dispositivos que estarán conectados a Internet el año 2020 serán aproximadamente 30000 millones, lo cual será posible gracias al nuevo protocolo *IPV6*.

Uno de los ámbitos de aplicación del Internet de las cosas, es el mundo de la automoción, el concepto de *vehículo conectado* cada vez toma más relevancia. Un vehículo conectado, es un vehículo que está equipado con acceso a internet (*Telefónica, 2014*), esto permite que se incorporen tecnologías que aprovechen la conexión a Internet ofreciendo funcionalidades nuevas al conductor o incluso que se desarrollen nuevas plataformas para aprovechar sistemas existentes como es el caso de la plataforma Vitruvius (Guillermo Cueva-Fernandez, Jordán Pascual Espada, Vicente García-Díaz, Cristian González García, Nestor Garcia-Fernandez., 2014). Algunos ejemplos de estas nuevas funcionalidades que se ofrecen a los usuarios son: avisos por exceso de velocidad, radios online, música en streaming, mapas online, prevención de accidentes, localización de vehículos, sistemas de asistencia al conductor, predictividad, ecall, recepción de email, búsquedas, localización de restaurantes, gasolineras, puntos de interés…

[1] Fuente http://www.rfidjournal.com/articles/view?4986
[2] Fuente: http://www.gartner.com/newsroom/id/2636073
[3] https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne

Cuando hablamos de aplicaciones en los vehículos conectados, frecuentemente se dividen entre dos grandes grupos, *in Car App*, que son aquellas integradas en el propio ordenador de a bordo del vehículo, y s*martphone App*, que son aquellas que se ejecutan en el smartphone.

Figura 3.1 Aplicación para smartphone Nissan Leaf

Hoy en día la mayoría de fabricantes ya ofrecen alguna solución para tener coches conectados, el incremento de los smartphone con conexión a Internet en el mercado permite en algunos casos, que la conexión a Internet sea utilizada a través del smartphone mientras que en otros casos es el propio vehículo el que incorpora el acceso a Internet. Cuando la conexión la incorpora el vehículo, en algunos casos, una aplicación para smartphone permite a los usuarios conectarse remotamente al vehículo y realizar algunas operaciones, como por ejemplo desbloquear el vehículo, encontrar la ubicación del vehículo, activar remotamente la climatización... Pese a todas las ventajas que pueden ofrecer los vehículos conectados, son muchos los retos que los fabricantes deben superar *(Forbes, 2014).*

3.1.2 Servicios Web REST

REST (Representational state transfer) es una técnica de arquitectura software, que describe una interfaz Web simple que utiliza *XML* y *http*, sin las abstracciones adicionales, de los protocolos basados en el intercambio de mensajes como SOAP. A los servicios Web REST a veces se les denomina *RESTful*., algunas características de *REST* son:

- Protocolo cliente servidor sin estado.

- Conjunto de operaciones bien definidas
- sintaxis universal para identificar recursos, cada recurso es direccionable únicamente a través de su *URI*
- Utilización de *hipermedios*, La representación del estado en un sistema *REST* es *HTML* o *XML* (habitualmente), por tanto es posible navegar de un recurso *REST* a otros siguiendo los enlaces, sin ningún tipo de infraestructura más.

Algunas de las ventajas de los servicios web *REST*: bajo consumo de recursos, instancias de proceso creadas explícitamente, el cliente no necesita información de enrutamiento a partir de la *URI* inicial, los clientes pueden tener una interfaz *listener*, genérica para las notificaciones, generalmente fácil de construir y acoplar.

Algunos de los inconvenientes que presentan los Servicios web *REST* son: descripción sintáctica/semántica muy informal, gran numero de objetos, ligado a un protocolo de transporte, los aspectos relativos a la seguridad, puede resultar una tarea complicada de implementar.

3.1.3 Ordenador de a bordo propuesto

3.1.3.1 Ordenador de a bordo para Citroën 2cv

Como base del ordenador de a bordo, tal y como hemos comentado, se ha tomado como base el proyecto final de carrera, titulado "*Implementación de un sistema de medición basado en microprocesador para vehículo antiguo*", presentado en Junio de 2014 en la *Universidad de Oviedo* para la obtención del titulo *ingeniería técnica informática de gestión*. Este proyecto consistió en un prototipo de ordenador de a bordo, diseñado para funcionar en un vehículo antiguo, concretamente en un *Citroën 2cv.*

El sistema diseñado consiste en una placa *Raspberry pi*, una placa *Arduino*, una pantalla *TFT* (*Thin-Film Transistor*), una pantalla *LCD* (*Lyquid Cristal Display*) con cinco botones y una serie de sensores: *GPS* (*Global Position System*), Giroscopio/Acelerómetro, *RTC* (*Real Time Clock*), sensores de temperatura y potenciómetro.

La *Raspberry pi*, es el corazón del sistema, dispone de un sistema operativo *Linux* basado en *Debian (Raspberry Pi, 2014)*, es la encargada de realizar toda la lógica, gestionar la entrada/salida de las pantallas *TFT* y *LCD (Fried, L., 2014)*, recibir y gestionar los datos del *GPS (Gómez, M.-b, 2013)* y comunicarse con la placa *Arduino* para solicitar información de los sensores mediante una conexión USB *(Universal Serial bus).* Esta parte del software

está escrito en lenguaje *Python* y tiene una arquitectura multihilo y multipantalla, mediante los botones de la pantalla *LCD* el usuario podrá ejecutar las distintas ordenes sobre el ordenador de abordo y además podrá alternar entre las distintas pantallas que se muestran en la pantalla *TFT*.

El *Arduino (Arduino-a, Arduino-b, Arduino-c, 2014)* se encarga de la lectura de sensores y transmitir los datos mediante *USB* y bajo demanda a la *Raspberry pi*, concretamente, lee los sensores de temperatura, el giroscopio/acelerómetro (*Gómez, M.-a, 2013*), el *RTC* y el potenciómetro.

El sistema consta seis pantallas, con sus correspondientes opciones menú y submenú, en todas las pantallas se muestra cierta información común: velocímetro, cuentakilómetros, posición (latitud y longitud), altitud, numero de satélites con los que está sincronizado el sistema, rumbo actual y dependiendo de la pantalla la siguiente información especifica:

- Pantalla principal: fecha y hora.
- Pantalla Inclinómetro: inclinación longitudinal y transversal
- Pantalla Temperatura: Temperatura interior y exterior.
- Pantalla Viaje: odómetro viaje A, odómetro viaje B y tiempo de viaje
- Pantalla Mantenimientos: En el caso de que este próximo un mantenimiento, mostrará la información relativa a dicho mantenimiento, mientras que si no hay un mantenimiento próximo, mostrará la información relativa al ultimo mantenimiento.
- Pantalla Alertas/avisos: Esta pantalla mostrará diversos avisos/alertas como pueden ser, mantenimiento necesario, riesgo de helada, nivel de combustible bajo e instalación de calandra necesaria.

Para conectarlo a Internet, utilizaremos un receptor *WIFI*, y aprovecharemos la conexión a Internet a través de compartir la conexión a internet mediante *WIFI* que incorporan la mayoría de los *smartphone*.

3.1.3.2 Citroën 2CV

El popularmente conocido como "*2 caballos*" (2cv) es un automóvil de bajo coste, producido por la marca francesa Citroën desde 1948 a 1990, durante este periodo se fabricaron un total de 3.872.583 unidades de 2cv.

Figura 3.2 Logotipo antiguo de Citroën

Este vehículo, tiene la carrocería de un sedán (4 puertas) pesa 600kg y respecto a sus dimensiones mide 3560mm de largo, 1480mm de alto, 1600 de ancho y una batalla de 2270mm. La estructura del 2cv esta formada por un chasis rígido de dos vigas a las que van sujetos todos los elementos mecánicos incluyendo la carrocería, lo cual le permitió a Citroën crear varios vehículos utilizando la misma base, como por ejemplo el *Dyane, Merari, Ami* y Ak400 (*Autoscout24, 2014*).

Figura 3.3 Citroën 2cv restaurado por el preparador Hermes[4]

[4] http://motorpasion.com

A lo largo de los 42 años que estuvo en producción, la carrocería, casi no sufrió ninguna modificación, tan solo la aparición de una tercera luna trasera, y modernización del capo, calandra pilotos, faros, salpicadero…

El sistema de propulsión del 2cv consiste en un motor ubicado en la parte delantera del vehículo, de dos cilindros *bóxer* (los cilindros están en posición horizontal y opuestos), refrigerado por aire y con carburador, originariamente tenia una cilindrada de 375 c.c. y 9 cv de potencia, y fue evolucionando con el paso de los años, aumentando la cilindrada, la relación de compresión, el diámetro de carburador, carburador de doble cuerpo, colectores de admisión-escape mas gruesos… hasta alcanzar los 602 c.c. de 35cv en los últimos años de su vida comercial (motor m28) .

El motor propulsa las ruedas delanteras mediante transmisiones de cardan simple en sus primeras versiones que posteriormente, evolucionaron en helicoidales.

La suspensión esta compuesta por brazos longitudinales sujetos en uno de sus extremos al chasis, y tensados por unas varillas fijadas a unos resortes helicoidales dispuestos de manera longitudinal a cada lado del chasis y un amortiguador por brazo. Este diseño único, permite alcanzar ángulos de escora bastante importantes en las curvas y se aprecia el característico andar cabeceante cuando se ve en circulación.

3.2 Estado del Arte

3.2.1 El mercado Actual de vehículos conectados

En este apartado se describirán las diferentes soluciones que se ofrecen en el mercado actual en relación a los vehículos conectados, cada fabricante tiene sus propio sistema y por tanto hablaremos de las principales marcas del mercado español (*Juan Antonio Corrales, Vicente Díaz y Hugo Valverde, 2014)*.

En primer lugar es importante señalar las técnicas que utilizan los fabricantes para que los vehículos se conecten a internet, existen tres maneras ampliamente difundidas:

- ***Dongle 3G***, la conexión a internet se realiza a través de un dispositivo *USB* que esta equipado con un módem inalámbrico para conectarse mediante conectividad *3G*.

- ***Tethering***, la conexión a internet se establece utilizando el smartphone u otros dispositivos, existen diferentes técnicas para realizar esta técnica, por cable, por bluetooth o por conexión *WIFI*.
- ***Embedded SIM***, la conexión a internet se realiza mediante una *SIM* integrada, y en ocasiones no accesible para el usuario, la idea de este sistema es que pueda utilizarse en comunicaciones *M2M* ("*Machine to Machine*" o también llamadas "maquina a maquina").

En segundo lugar, hay que diferenciar entre varios tipos de aplicaciones que son ofrecidas por los fabricantes a sus clientes, la tipología de aplicaciones es la siguiente:

- ***In car apps***, este tipo de aplicaciones son las que se ejecutan en el propio ordenador de a bordo del vehículo, sin necesidad de recurrir al smartphone del usuario.
- ***Smartphone apps***, este tipo de aplicaciones se ejecuta en el smartphone del usuario y dicha ejecución se ve reflejada en el sistema de *infotainment* del vehículo.
- ***App contenedoras***, este tipo de aplicaciones consiste en que el fabricante ofrece este tipo de aplicaciones que permiten a su vez acceder a otras aplicaciones, que pueden incluso ser un desarrollo de terceros.

Ahora que ya se ha detallado los tipos de aplicaciones, y las técnicas de conexión de los sistemas a internet, analizaremos las diferencias tendencias, en cuanto a ordenadores de a bordo (o sistemas de *infotainment*) para tener un vehículo conectado. Fundamentalmente podemos distinguir las siguientes tendencias:

- Vehículo conectado basado en un sistema independiente
- Vehículo conectado basado en aplicaciones para smartphone
- Vehículo conectado basado en ambos sistemas

3.2.1.1 Vehículo conectado basado en un sistema independiente

En esta tendencia, el sistema conectado se ejecuta íntegramente en el propio vehículo, mediante aplicaciones integradas en el propio sistema del vehículo. Esta es la solución adoptada por las siguientes marcas: *Toyota*, *Lexus*, *Citroën*, *Peugeot* y *Renault*.

Toyota Touch 2 & go[5]

Figura 3.4 Logotipo Toyota

El fabricante japonés *Toyota* ofrece en aquellos vehículos que incorporen el pack *touch 2 & go,* el sistema basado en *in car App*, con una tienda de aplicaciones propia. Para poder disfrutar de los servicios es necesario que el usuario se registre (y registre su vehículo) en el portal web para el cliente de *Toyota*. La conexión a internet en este sistema se realiza por *tethering* vía WIFI o bluetooth.

La tienda de aplicaciones contiene una serie de aplicaciones que proporcionan la siguiente funcionalidad: *búsqueda online Google*, *Google Street View*, información de aparcamiento, información de trafico, *TomTom places*, correo electrónico, redes sociales, precios de combustible, Panoramio...

Lexus Connected Services[6]

Figura 3.5 Logotipo Lexus

El fabricante japonés *Lexus* ofrece en aquellos vehículos que incorporen el *navegador Lexus Premium*, un sistema muy parecido al ofertado por *Toyota* (Ambas marcas pertenecen al grupo *Toyota*), lo que comercialmente se denomina *Lexus Connected Services.* Para poder disfrutar de los servicios es necesario que el usuario se registre (y registre su vehículo) en el portal web *MyLexus*. En este sistema la conexión a internet se realiza por *Tethering* vía *WIFI* o *Bluetooth*.

Este sistema consta de una tienda de aplicaciones propia que contiene una serie de aplicaciones que proporcionan la siguiente funcionalidad: *búsqueda online Google*, *Google*

[5] http://www.toyota.es/world-of-toyota/articles-news-events/2014/toyota-touch-2.json
[6] https://customerportal.lexus-europe.com/login/index.html

Street View, información de aparcamiento, información de trafico y *Panoramio*. Este sistema además es compatible con la tecnología *MirrorLink*.

Citroën Multicity Connect[7]

Figura 3.6 Logotipo Citroën

El fabricante francés *Citroën* ofrece un sistema basado únicamente en *in car Apps* con su tienda de aplicaciones propia, este sistema se denomina comercialmente *Citroën Multicity Connec*t, aunque en la actualidad dicho sistema solo esta disponible para algunos modelos de la gama C4. Para poder disfrutar de los servicios es necesario previamente que el usuario se registre en la página web *MyCitroen* para obtener el identificador de *Citroën* e introducirlo en el propio vehículo. En este sistema la conexión a internet se realiza a través de un *Dongle 3G* que suministra *Citroën*.

Algunos ejemplos de las funcionalidades que pueden aportar estas aplicaciones: *Guía Michelín*, *Michelín viajes*, *Wikipedia*, calculadora, conversor, juegos, climatología, *Michelín Trafico*, precios de carburante, Párking, *Facebook*, *MyCitroen*, lector de email...

Peugeot Connect App[8]

Figura 3.7 Logotipo Peugeot

El fabricante francés *Peugeot* ofrece un sistema basado únicamente en in car Apps, con su tienda de aplicaciones propia, este sistema es muy similar al sistema de Citroën (ambas marcas pertenecen al grupo *PSA*) y se denomina comercialmente *Peugeot Connect App*, aunque en la actualidad solo esta disponible en algunos modelos que incorporan Navegación, segunda toma *USB* y *CD/MP3*. Para poder disfrutar de los servicios es necesario previamente acceder a la página web *MyPeugeot* para obtener el identificador de

[7] http://www.citroen.es/tecnologia/citroen-multicity-connect.html
[8] http://servicios.peugeot.es/servicios-conectados-peugeot-connect-apps/

Peugeot e introducirlo en el propio vehículo. En este sistema la conexión a internet se realiza a través de un *Dongle 3G* que suministra *Peugeot*.

Algunos ejemplos de las funcionalidades que pueden aportar estas aplicaciones: *Guía Michelín*, *Michelín viajes*, *Wikipedia*, calculadora, conversor, juegos, climatología, *Michelín Trafico*, precios de carburante, Parking, *Facebook*, *MyPeugeot*, lector de email...

Renault R-Link[9]

Figura 3.8 Logotipo Renault

El fabricante francés Renault, ofrece un sistema basado en *in car Apps* con su tienda de aplicaciones propia, comercialmente se le denomina *R-Link*, y para disfrutarlo es necesario equipar el vehículo con el sistema multimedia *R-Link* y el usuario debe de registrarse (y registrar el vehículo) en la pagina web *myRenault*. En este sistema la conexión a internet se realiza mediante una tarjeta *SIM* embebida.

Algunos ejemplos de las funcionalidades que pueden aportar estas aplicaciones: *Rlink tweet*, lector de email, calendario, *R-Sound Effect*, gestión de tiempo de descanso, llamada de emergencia, eguide captur, euronews español, *TomTom places*, *búsqueda online Google*, *Coyote series, Rlink eguide*...

3.2.1.2 Vehículo conectado basado en aplicaciones para smarphones

En esta tendencia, el fabricante ofrece aplicaciones para *smartphone*, las cuales tal como comentamos anteriormente se ejecutan en el dispositivo móvil pero generalmente reflejan determinadas funcionalidades en el propio sistema de *infotainment* del vehículo y se suele poder gestionar estas aplicaciones con los controles del vehículo. Esta solución es adoptada por las siguientes marcas: *Opel* (y otras marcas del grupo *General Motors*, como *Chevrolet* y *Vauxhall*), *Ford* y *Mini*.

[9] http://www.renault.es/descubre-renault/innovacion-tecnologia/renault-r-link/

Opel Intellilink[10]

Figura 3.9 Logotipo Opel

El Fabricante alemán Opel, basa su idea de coche conectado en aplicaciones para smartphone, gestionados a través de la interfaz de voz del vehículo, en la actualidad no se ofrece aún en todos los modelos, ni todos los modelos disponen de las mismas aplicaciones.

El catalogo de servicios que están disponibles en estos momentos son: *Bringgo* (Aplicación de navegación que incluye mapas en 3D trafico en tiempo real y guiado por voz), *Glympse* (compartir ubicación en redes sociales como *Facebook*, *Twitter*, *SMS* o correo electrónico), *Stitcher* y *TuneIn Radio* (Ambos servicios de radio online).

Ford Sync Applink[11]

Figura 3.10 Logotipo Ford

El fabricante americano *Ford*, basa su idea de coche conectado en aplicaciones para smartphone, gestionados a través de la interfaz de voz del vehículo. A diferencia de otros sistemas, *Ford* ofrece este sistema en prácticamente todos sus modelos e incluso en las versiones de acceso.

El catalogo de servicios que están disponible en estos momentos son: *Glympse* (compartir ubicación en redes sociales como *Facebook*, *Twitter*, *SMS* o correo electrónico), *Spotify*, *Aupeo* y *Audioteka*.

[10] http://www.opel.es/microapps/adam/adam-intellilink.html
[11] http://es.ford.com/sdsupport/sync-technology/applink-overview-sync

Mini Connect[12]

Figura 3.11 Logotipo Mini

El fabricante alemán *Mini* opta por una aplicación para smartphone que se gestiona desde la interfaz del coche, esta aplicación contenedora se llama *Mini Connect*, y para disfrutar de esta funcionalidad es necesario equipar al vehículo con la *Radio Mini Visual Boost*, descargar la aplicación y realizar el registro a través de ella.

Esta aplicación dispone de varias funciones: *Drive Excitement* (un juego en el que el conductor gana puntos e insignias dependiendo de su estilo de conducción deportiva, evaluando aceleración, frenada, cambios de marcha y manejo del volante), *Minimalism Analyser* (similar al Drive Excitement pero con el objetivo de conducción eficiente), *Dynamic Music* (esta función a través de ciertas canciones que se descargan desde la aplicación, acelera o ralentiza el ritmo de la música en función del estilo de conducción), *Mission control* (aplicación que informa de temperatura exterior y otros parámetros) y aplicaciones de terceros *Audible*, *Snippy*, *amazon Music*, *aupeo personal radio*, *Deezer*, *Napster*, *TuneIn Radio*, *Glympse* y *Stitcher*. Respecto a las aplicaciones de terceros, es necesario descargarlas de la tienda de aplicaciones correspondiente y en algunos casos requieren de registro, algunas de ellas no están disponibles para Android.

Próximamente se lanzará en el mercado español *Mini Connected XL Journey Mate*, que incluye mas funcionalidades relacionadas con los viajes, información de trafico, información de aparcamiento, gasolineras, previsión del tiempo, *Car Finder* y *Last Mile* (estas dos ultimas son heredadas del sistema del *BMW*, recordemos que *Mini* pertenece al mismo grupo de *BMW* y se comparten sinergias entre las marcas).

3.2.1.3 Vehículo conectado basado en ambos sistemas

En ese caso los fabricantes optan por soluciones mixtas basadas en las dos tendencias anteriores. En algunos casos, el propio terminal móvil tiene acceso a los datos del vehículo (*FISITA. (2012)*). Ejemplos de fabricantes que han decidido seguir este camino son: *Infinity*, *Nissan*, *Volvo*, *Mercedes-Benz, Mazda*, *BMW* y *Audi*.

[12] http://www.mini.es/connected/

Infinity inTouch

Figura 3.12 Logotipo Infinity

El fabricante japonés *Infinity* incorpora en muchos de sus modelos de serie su sistema *inTouch*, para disfrutar de este sistema, el usuario debe de registrarse y dar de alta el vehículo en el portal *inTouch* de *infinity*.

El sistema *inTouch* por una parte incorpora de serie de *in Car Apps*, cuenta además con su tienda de aplicaciones, llamado *Infinity Garage*, en la actualidad dispone de aplicaciones que le permiten acceder a la siguiente funcionalidad: guía rápida, reloj, brújula, información sobre la conducción, calendario, email y notas sobre el mantenimiento.

Por otra parte la aplicación para smartphone llamada Infinity InTouch, permite gestionar las búsquedas de Google y la utilización de la red social Facebook.

Nissan Connect

Figura 3.13 Logotipo Nissan

La estrategia de este fabricante japonés es bastante similar a la utilizada por *Infinity* (recordemos que son marcas del mismo grupo y comparten sinergias). Este sistema viene de serie en algunos acabados de algunos modelos. La conexión a internet se realiza por *Tethering* vía *Bluetooth* o por cable *USB*. Para poder disfrutar de los servicios que ofrece este sistema es necesario registrarse en el portal web de *Nissan*.

El sistema *Connect* por una parte incorpora de serie de *in Car Apps*, que cuenta con su tienda de aplicaciones propia, en la actualidad dispone de aplicaciones que le permiten acceder a la siguiente funcionalidad: *Google send to car*, *POI*, precios de carburante, información de vuelos, y meteorología. Por otra parte la aplicación para smartphone llamada *Nissan Connect* permite gestionar las búsquedas de *Google* y la utilización de la red social *Facebook*.

Volvo Sensus Connected Touch

Figura 3.14 Logotipo Volvo

El fabricante sueco *Volvo* ofrece una solución que ha sido desarrollada por un tercero (*Parrot*, al igual que el sistema ofrecido por la marca *McLaren*). Este sistema se basa en *in Car Apps* y cuenta con su propia tienda de aplicaciones, llamada *Asteroid Market*. Este sistema esta disponible como extra en algunos modelos de la gama Volvo y comercialmente se denomina *Volvo Sensus Connected Touch*. Es interesante comentar que esta solución se basa en una versión modificada del sistema operativo Android. La conexión a internet se puede realizar mediante *Tethering* vía *WIFI* o cable *USB*, o mediante un *Dongle 3G*.

Las *in car Apps* que existen en estos momentos ofrecen los siguiente servicios: correo electrónico, *spotify*, *TuneIn* Radio, *Deezer*, *Live Radio*, *Road Trip*, Coyote *series*, calculadora, calendario, navegador y servicios *Volvo*.

Para completar la oferta de servicios, *Volvo* dispone de una aplicación para smartphone llamada *Volvo on call*, que permite gestionar determinadas funciones del vehículo remotamente y además permite obtener cierta información.

Mercedes-Benz Command OnLine APS/Drive Style App

Figura 3.15 Logotipo Mercedes-Benz

El fabricante alemán *Mercedes-Benz* ofrece como extras en algunos de sus modelos el sistema *Command OnLine APS*, para poder disfrutar de estos servicios es necesario registrar el vehículo en el portal web. En este sistema la conexión a internet se realiza por *Tethering* vía *Bluetooth* o a través de una tarjeta *SIM* integrada. Este sistema cuenta con su propia tienda de aplicaciones, entre las aplicaciones que incluye destacan: noticias, *Facebook*, normas de trafico de cada país, tiempo, radio por internet...

Se complementa la oferta de servicios con una aplicación contenedora para smartphone con sistema operativo *iOS* llamada *Drive Style*, la cual ofrece la siguiente funcionalidad: gestión de redes sociales (*Facebook*, *Twitter*, *Glympse*), búsqueda online *Google*, lugares de *Garmin*, radio por internet... Como nota anecdótica, hay que decir que han desarrollado una aplicación para smartwatch.

Mazda MZD Connect

Figura 3.16 Logotipo Mazda

El fabricante japonés *Mazda* denomina comercialmente a su solución *MZD Connect*, y para poder disfrutarlo es necesario que el vehículo disponga del navegador con pantalla táctil de 7 pulgadas. En este sistema la conexión a internet se realiza por *Tethering* vía *WIFI*. Es necesario que el usuario se registre en el portal web *mazda connect*, a través de la misma es posible descargarse los mapas y los datos relativos a servicios integrados, los cuales proporcionan la siguiente información: trafico, precios de combustible, búsqueda local, ahorro de combustible, mantenimiento...

Además para completar su oferta de servicios, este fabricante ha decidido utilizar como aplicaciones contenedoras para smartphone, dos aplicaciones desarrolladas por terceros: *Sticher* (esta aplicación permite escuchar radios online y podcast) y *Aha* (esta aplicación permite el manejo de redes sociales, información sobre hoteles, restaurantes el tiempo...).

BMW ConnectedDrive

Figura 3.17 Logotipo BMW

El fabricante alemán *BMW* dispone de el sistema denominan *ConnectedDrive*, que es un conjunto de tecnologías que conectan al conductor y al vehículo con su entorno. Este sistema esta disponible en prácticamente toda la gama de vehículos. Para disponer del

mismo, es necesaria la equipación *navegación profesional* y además es necesario registrar el vehículo en el portal Mi BMW *ConnectedDrive*. En este sistema la conexión a internet se realiza mediante una *SIM* integrada en el vehículo.

E n *BMW* se dispone de una aplicación contenedora para smartphone llamada *BMW Connected* y que da acceso a una serie de servicios propios como *Mlap Timer*, control remoto de determinados aspectos del vehículo mediante *MyRemote*, *Eco Pro Analyser*, Radios por internet, música en streaming, control de redes sociales, calendario, *RSS*...

El propio sistema integrado incluye además una serie de servicios como son: *BMW online Mail*, mis noticias, bolsa, cajeros automáticos hoteles, información de parking, tiempo, búsqueda, navegación, restaurantes, imágenes del destino...

Audi connect

Figura 3.18 Logotipo Audi

El fabricante alemán *Audi*, dispone de la posibilidad de convertir el vehículo en conectado (la denominación comercial de su sistema es *Audi Connect*) en prácticamente toda su gama de vehículos, para ello existe la equipación *MMI Naviation Plus* y *MMI Touch*. En este sistema la conexión a internet se realiza por *Tethering* vía *Bluetooth* o a través de una tarjeta *SIM* integrada. Para poder disfrutar de todos estos servicios es necesario registrar el vehículo en el portal Web *MyAudi*, en esta web, se podrán configurar diversos parámetros de los servicios.

Por una parte, *Audi* dispone de una aplicación contenedora para smartphone llamada *Audi MMI Connect*, aunque en realidad, tiene relativamente poco peso en comparación con el sistema integrado, pero permite diversa funcionalidad como por ejemplo datos de trafico, destinos con imágenes, sistema de audio en streaming, *car finder*...

Por otra parte y para completar este sistema, el propio vehículo dispone de una serie de funciones proporcionadas por *in car Apps*, dichas funciones son: acceso a redes sociales, *RSS*, información sobre viajes, tráfico, distintos medios de transporte, precios de combustible, eventos..

3.2.2 Los próximos lanzamientos

La principal característica que podemos deducir del mercado actual, en el que vimos las diferentes soluciones propuestas por los distintos fabricantes, es que existe una gran fragmentación, cada fabricante propone su sistema, aunque tenga muchos puntos en común con soluciones de otros fabricantes, algunas de las principales empresas tecnológicas del mundo han visto este problema y por tanto una oportunidad, y con el fin de crear un estándar que llegue a solucionar este problema de la fragmentación, se han desarrollado diferentes propuestas: *Google Android Auto*[13], *Apple CarPlay*[14], *MirrorLink*[15] y *Windows in the car*[16]. Todas estas propuestas se basan en aprovechar la integración del Smartphone Apps.

3.2.2.1 Google Android Auto

Figura 3.19 Logotipo Android Auto

Android Auto es la solución estándar desarrollada por *Google*, anunciada en la conferencia *Google I/O* 2014, para permitir a los dispositivos móviles *Android* interactuar con el ordenador de a bordo del vehículo. *Android Auto* promete a los usuario el control del sistema de navegación, reproducción de música, *SMS*, telefonía, navegación web, soporte a pantallas táctiles, pantallas con botones y control por voz, una de las fundamentales ideas sobre las que se ha enfatizado en la presentación de este sistema, es asegurar la conducción segura.

[13] Web oficial de Andoid Auto: https://developer.android.com/auto/index.html
[14] Web oficial de Apple Carplay: https://www.apple.com/es/ios/carplay/
[15] Web oficial de MirrorLink: http://www.mirrorlink.com/
[16] Presentado en la conferencia Build 2014: http://www.buildwindows.com/

Figura 3.20 Ordenador de a bordo con Android Auto

La idea principal es la de proyectar alguna de las funciones del smartphone en el sistema multimedia del coche, y a la vez, que las aplicaciones tengan acceso a diversos sensores y datos del automóvil. Dispone de un diseño simplificado, con esto se consigue disminuir en la medida de lo posible las posibles distracciones durante la conducción y facilitar la usabilidad del sistema.

Android Auto forma parte de la *Open automotive Alliance*[17], y durante principios del 2015, llegará al mercado el primer vehículo que soporte Android Auto, con el que serán compatibles, todos aquellos *Android* cuya versión sea igual o superior a *Android 5.0* "*Lollypop*".

3.2.2.2 Apple CarPlay

Figura 3.21 Logotipo Apple CarPlay

CarPlay es el estándar presentado por *Apple*, consiste en que sus dispositivos con iOS sean capaces de trabajar con sistemas integrados de los vehículos. Se dio a conocer durante el discurso de apertura de la conferencia mundial de desarrolladores *Apple* (*WWDM*) y en el salón del automóvil de Ginebra se presentó en un vehículo que ya incorporaba esta funcionalidad, siempre y cuando que el dispositivo *iOS* estuviese actualizado a 7.1.

[17] Web oficial OAA: http://www.openautoalliance.net/

Figura 3.22 Ordenador de a bordo con Apple CarPlay

La idea de este sistema es que desde los controles de los vehículos, se tenga acceso a diversas funcionalidades del dispositivo *iOS*, como por ejemplo el sistema *SIRI* en modo *eyes free*, para poder utilizar el manos libres, la navegación por satélite, el control del teléfono, el control de la música mediante *iTunes* y control del sistema *iMessage*.

CarPlay ya ha llegado recientemente al mercado de la mano de vehículos *Mercedes-Benz*, *Volvo*, *Ferrari*, *Honda* y *Hyundai*. Además existe un gran numero de marcas que están interesadas en este sistema: *BMW*, *Ford*, *General Motors*, *Jaguar*, Grupo *Daimler* AG, *Mitsubishi*, Grupo *Nissan*, Grupo *PSA*, *Subaru, Suzuki*, *Toyota*....

3.2.2.3 MirrorLink

Figura 3.23 Logotipo MirrorLink

MirrorLink es una solución estándar desarrollada en sus origines por Nokia, fué presentado en el 2013 por el *Connected Car Consortium*. Esta tecnología se encarga de duplicar la pantalla del *smartphone* en la pantalla Ivl (*In-Vehicle Infotainment)* del vehículo. En este estándar se hacen uso de tecnologías (en su mayoría no propietarias) como son el protocolo *TCP/IP*, conectividad *USB*, *Bluetooth*, protocolo *RTP* o el protocolo *VNC.*

Figura 3.24 Ordenador de a bordo con MirrorLink

El principal problema de esta tecnología, es que la compatibilidad esta vinculada a los fabricantes de smartphone con los que se ha llegado a algún acuerdo, por lo que si el conductor no tiene un smartphone que el fabricante decidiese incorporar a la plataforma. Por poner ejemplos de terminales compatibles que ya sean compatibles con esta tecnología, están los últimos modelos de *Nokia* con *Symbian*, algunos modelos de *Samsung*, *HTC*, *Panasonic*, *Sony*…

En la actualidad, ya hay algún modelo que incorpora como opción esta solución: *Volkswagen Polo*, *Lexus CT200H*, *Citroën C1*….Además, empresas como *JVC*, *Alpine* y *Sony*, disponen de dispositivos para automóviles que soportan esta tecnología. Existe una certificación *MirrorLink* para que solo las aplicaciones que tengan esta certificación puedan ser mostradas mientras se conduce, como medida de seguridad en la conducción.

3.2.2.4 Windows in the car

Figura 3.25 Logotipo Windows

La respuesta de *Microsoft* a *Android Auto* y *Apple CarPlay*, se ha propuesto en la conferencia *Build 2014* y se denomina *Windows in the car*, esta solución a diferencia de las dos anteriores, no es una tecnología completamente nueva, sino que se basa en el estándar *MirrorLink*, a la que se pretende que disponga de una interfaz de usuario similar a la que tiene el sistema operativo *Windows 8*.

Figura 3.26 Ordenador de a bordo con windows in the car

De este sistema aun hay pocos datos y fue el ultimo en ser presentado, aun así es importante señalar que *Microsoft* ya tiene experiencias previas en este negocio con el producto *Windows Embedded Automotive*, el cual cuenta con varias aplicaciones en el mercado como son: *Kia UVO*, *Ford Sync* y *Fiat Blue&me*.

Capítulo 4. Planificación

4.1 Ciclo de vida

En este apartado se explica el ciclo de vida que se ha decidido aplicar para este sistema Software y Hardware.

- **Idea o Concepto**: En esta fase se define lo que se quiere desarrollar y se presenta una propuesta.
- **Análisis**: En esta etapa se define funcionalmente el sistema y de define la especificación técnica.
- **Producción**: En esta etapa clave se realiza la implementación del software aplicación y se desarrolla la implementación Hardware sobre la que trabajará nuestro software.
- **Alfa**: El sistema esta operativo pero quedan muchos detalles por pulir y defectos que solucionar.
- **Beta**: En esta fase solo queda depurar errores para poder liberar el sistema cuando esté estable.
- **Congelación del código**: En esta fase el código se congela para solucionar los principales errores detectados en la fase Beta y se encuentra pendiente de recibir la aprobación para su liberación.
- **Liberación de la versión final**: El sistema se implanta en el vehículo.
- **Parches y Actualizaciones**: Representan soluciones a problemas detectados y también ampliación de funcionalidades.

De cara a los hitos que se han planificado en este proyecto, son destacables el prototipo (consecuencia del Análisis), la versión Alfa (consecuencia de la fase de producción), la versión Beta (Consecuencia de pulir la versión Alfa) y por ultimo la versión final (correspondiente a la liberación del código final y la presentación de este proyecto).

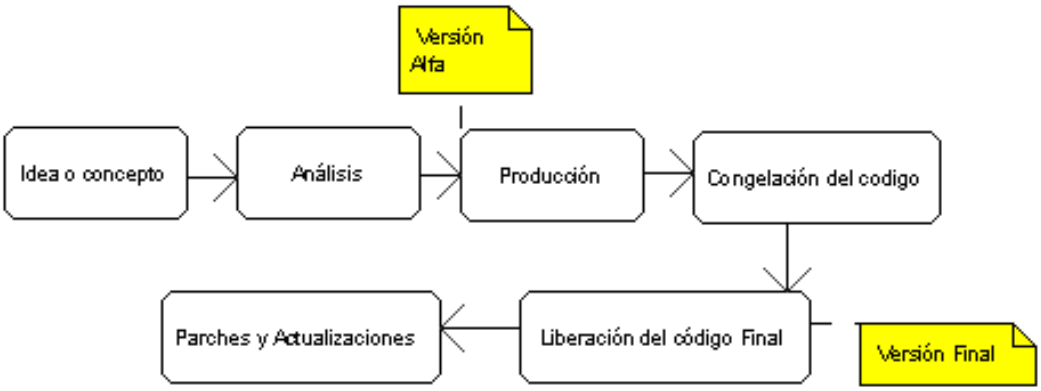

Figura 4.1.Ciclo de vida

4.2 Planificación Temporal

El modelo de ciclo de vida que se ha elegido para esta aplicación es el de un modelo evolutivo y además se aplica un prototipado, dado que en la primera iteración se desarrolla el prototipo. La elección de este modelo se da por la necesidad de flexibilidad a la hora de incluir nuevos requisitos o mejorar la funcionalidad, aspecto muy importante cuando hay que enfrentarse a un negocio en el que no hay experiencia previa, como es en este caso el de la automoción.

Cada iteración se basa en el ciclo de "requisitos – desarrollo - evaluación" de tal manera que una vez que ya se ha finalizado la fase de desarrollo, se evalúa el resultado y así se pueden añadir requisitos o revisar los requisitos actuales en caso de que la evaluación no resulte positiva. De esta manera se obtienen versiones más refinadas del producto final y éste a su vez va mejorando continuamente entrega tras entrega.

Figura 4.2. Modelo evolutivo

Para la realización de la planificación se han marcado tres hitos, que corresponden con las versiones del software, que son: prototipo Interfaz, versión Alfa y versión Final. Cada una de estas versiones es el resultado de una iteración y por tanto se han planificado cada una de las tres tareas que forman cada iteración, que son Requisitos, Desarrollo y Evaluación. En base a todo esto, se realizo un diagrama de *Gantt* para que quede reflejada la planificación temporal de este proyecto.

El resultado de la planificación es de 74 jornadas laborales de 3 horas, lo que hace un total de 222 horas, dichas jornadas quedan repartidas de la siguiente manera:

- Análisis del problema y estudio del estado del Arte 5 Jornadas (15 horas)
- Requisitos 20 Jornadas (60 horas)
- Desarrollo 37 Jornadas (111 horas)
- Evaluación 12 Jornadas (36 horas)

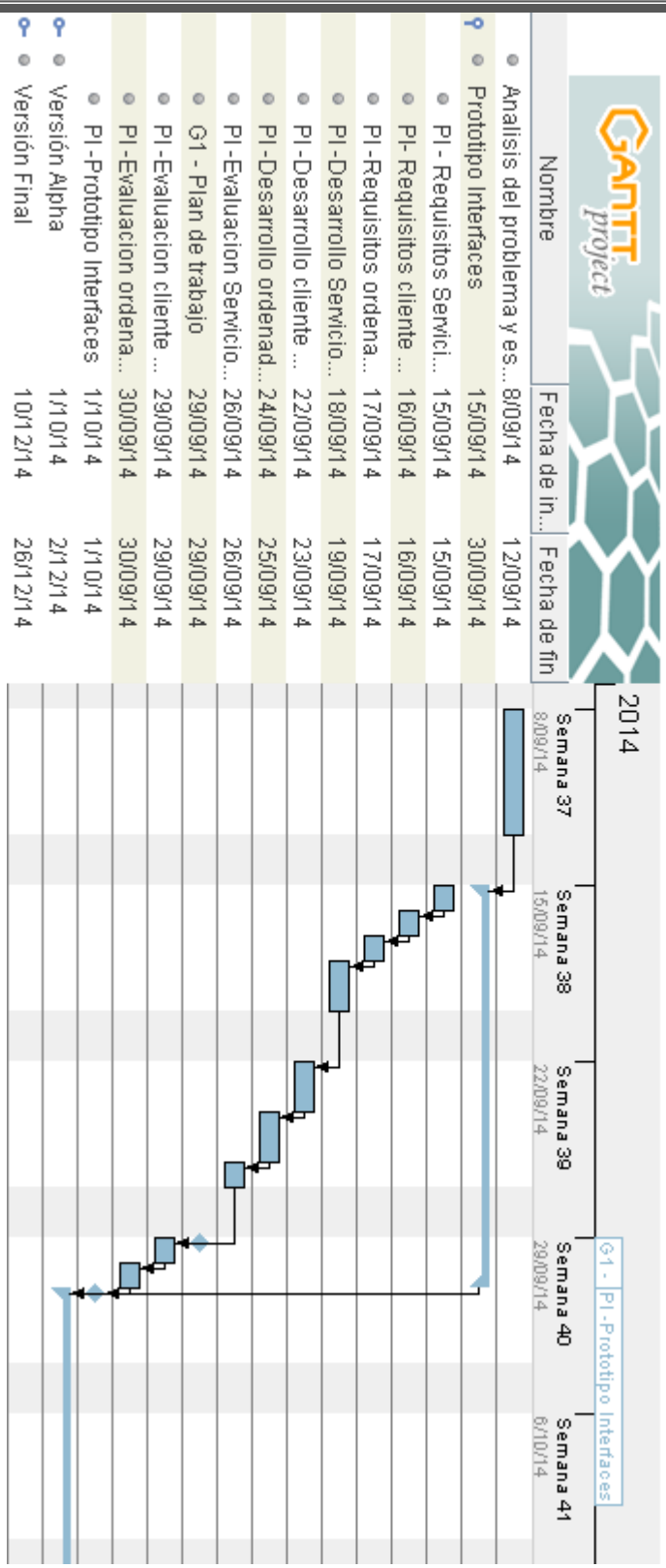

Figura 4.3 Diagrama de Gantt: Prototipo Interfaces

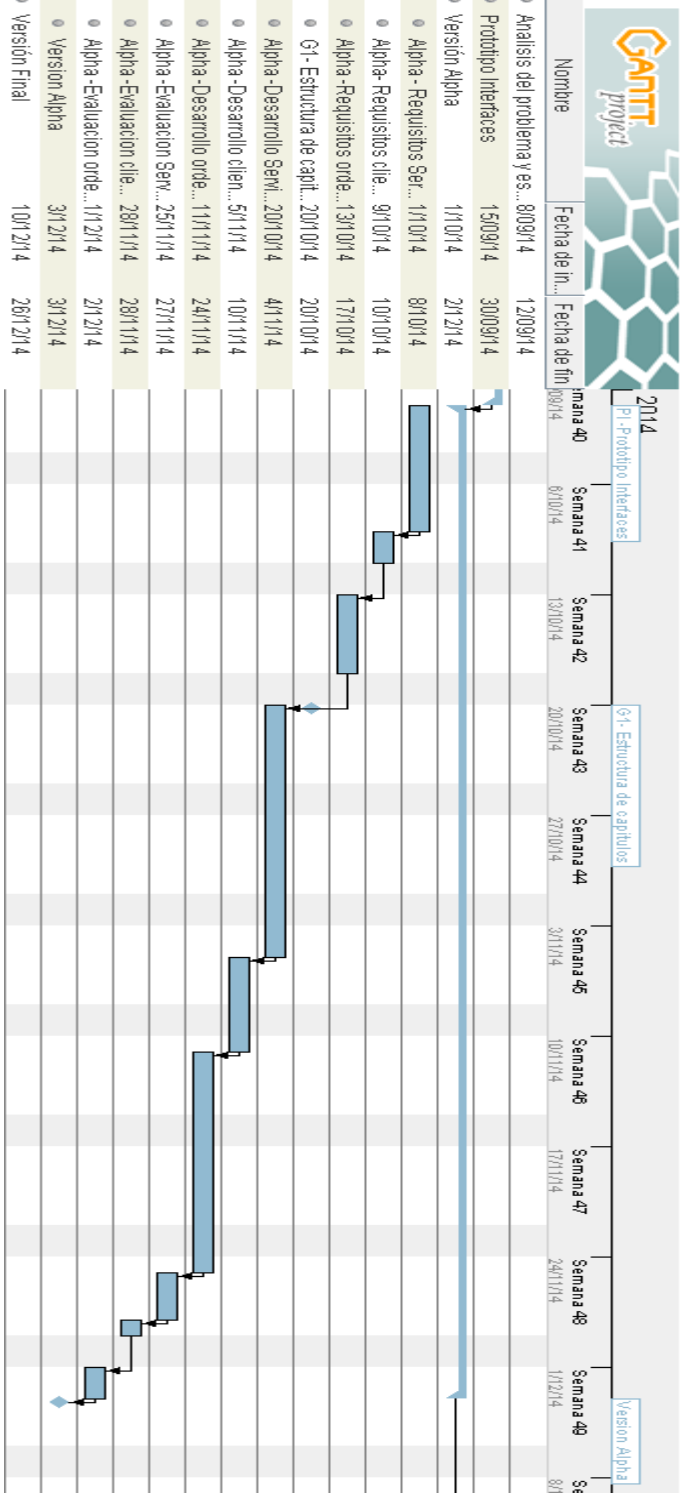

Figura 4.4 Diagrama de Gantt: Versión Alpha

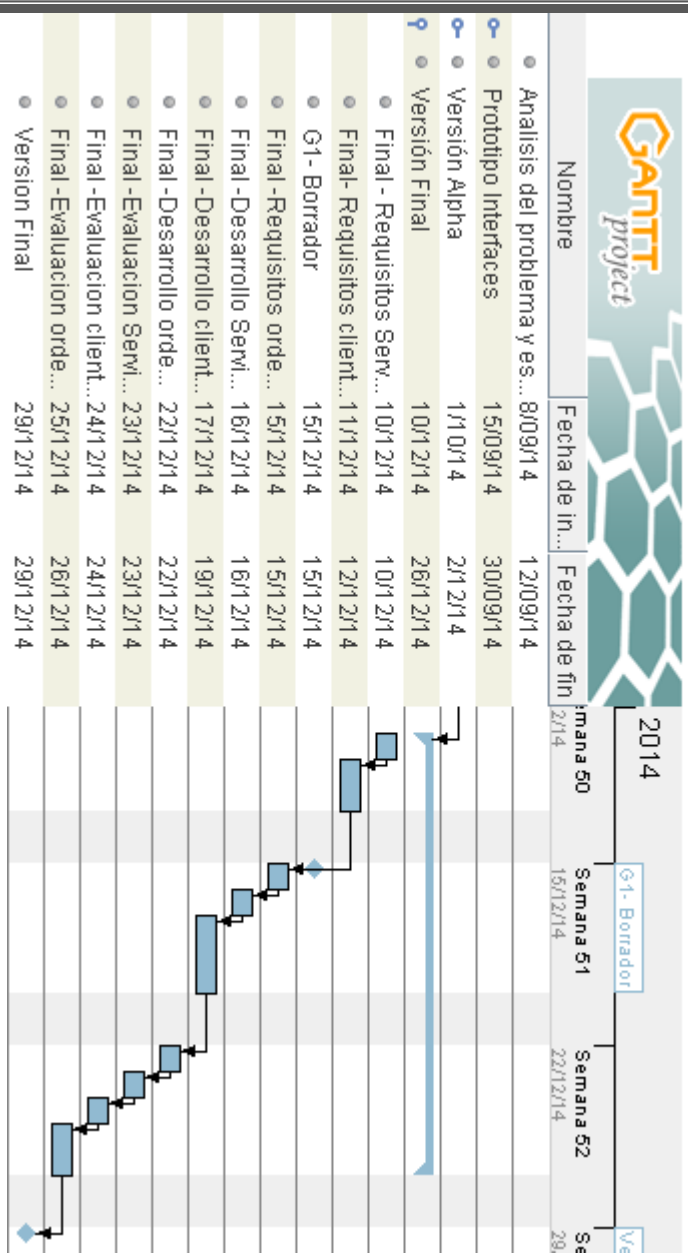

Figura 4.5 Diagrama de Gantt: Versión Final

Capítulo 5. Análisis

En este capitulo se detallará el proceso de análisis de este sistema, por tanto contendrá la especificación de requisitos y toda la documentación de análisis de la aplicación, posteriormente y tomando como base este análisis, se propondrá un diseño técnico, sobre el que se implementará la solución.

5.1 Definición del Sistema

5.1.1 Determinación del Alcance del Sistema

Para la realización de este prototipo es muy importante detallar correctamente el alcance del sistema, ya que son muchas las posibilidades que podrían tener sentido en este sistema, y si dejamos volar la imaginación, son muchas las ideas que podríamos implementar, por lo que es necesario acortar la solución propuesta. En esta sección definiremos el alcance de este sistema.

La idea consiste en desarrollar unos servicios web, que permitan a los usuarios geoposicionar determinados eventos que pueden suceder en las vías de circulación y consultar dichos eventos. Los eventos disponibles van a ser los siguientes:

- Accidente
- Atasco/Retención
- Mantenimiento
- Otros

Los servicios web desarrollados utilizarán el protocolo *REST*, y serán desarrollados con lenguaje *java*, con ayuda de diferentes módulos del framework *Spring*, con una base de datos *MySQL*, para desplegarlos en un servidor de aplicaciones *Tomcat*.

Para el aprovechamiento de estos servicios web, se desarrollará un módulo que complementará al ordenador de a bordo presentado como proyecto final de carrera en la Universidad de Oviedo, el cual ya hemos mencionado en capítulos anteriores. Este módulo permitirá a los usuarios mediante la botonera situada en la pantalla *LCD*, dar de alta incidencias y además, visualizar en la pantalla *TFT* mediante texto las incidencias más

cercanas a su posición (la incidencia de cada tipo más cercana) indicando el tipo de incidencia, junto con la distancia y la dirección en la que se encuentra. Este nuevo módulo se desarrollará en *Python* al igual que el resto del sistema que tomamos como base y llevará asociado una nueva opción de menú con sus correspondientes opciones de submenú y una vista para la pantalla *TFT* con la información que ya hemos comentado.

De manera complementaria, se desarrollará un cliente web, que aproveche los servicios web desarrollados y que permita la visualización en un mapa, de las incidencias activas dadas de alta en el sistema. Para ello se utilizará la *API* de *Google Maps* para *JavaScript* en su versión 3. La funcionalidad de este cliente web, se complementará con una vista adicional, en la que los usuarios, si así lo desean, podrán dar de alta incidencias desde el propio cliente Web. Este cliente web, se desarrollará con las tecnologías *HTML5*, *CSS3* y *JavaScript*.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

5.2.1.1 Requisitos funcionales

En la siguiente tabla se describen los diferentes requisitos funcionales que se han identificado para este sistema.

Código	Nombre Requisito	Descripción del Requisito
RF1	Ordenador de a bordo: alta accidente.	Desde el ordenador de debe permitir dar de alta incidencias de tipo accidente.
RF2	Ordenador de a bordo: alta mantenimiento.	Desde el ordenador de debe permitir dar de alta incidencias de tipo mantenimiento.
RF3	Ordenador de a bordo: alta retención.	Desde el ordenador de debe permitir dar de alta incidencias de tipo retención/atasco.
RF4	Ordenador de a bordo: alta otros eventos.	Desde el ordenador de debe permitir dar de alta incidencias de tipo otros.
RF5	Ordenador de a bordo: Indicación incidencias cercanas.	Desde el ordenador debe indicar al usuario la incidencia de cada tipo más cercana al vehículo.
RF6	Cliente Web: alta accidente.	Desde el cliente Web se debe permitir dar de alta incidencias de tipo accidente.
RF7	Cliente Web: alta	Desde el cliente Web se debe permitir dar de

	mantenimientos.	alta incidencias de tipo mantenimientos.
RF8	Cliente Web: alta retención.	Desde el cliente Web se debe permitir dar de alta incidencias de tipo retención.
RF9	Cliente Web: alta otros eventos.	Desde el cliente Web se debe permitir dar de alta incidencias de tipo otros.
RF10	Cliente Web: visualización de todas las incidencias.	Desde el cliente Web se debe permitir mediante un mapa la visualización todas las incidencias registradas.

5.2.1.2 Requisitos no funcionales

En la siguiente tabla se describen los diferentes requisitos funcionales que se han identificado para este sistema.

Código	Nombre Requisito	Descripción del Requisito
RNF1	Alimentación por batería de 12V de un automóvil.	El ordenador de a bordo debe poder instalarse en un vehículo alimentado eléctricamente por una batería de 12V
RNF2	Internet	El ordenador de a bordo debe disponer de conectividad a Internet
RNF3	Gestión de errores	En caso de no disponer de Internet, el sistema debe de ser coherente.
RNF4	Coherencia estética	El nuevo modulo debe tener el mismo *look and feel* que el sistema previamente desarrollado

5.2.2 Especificación de Casos de Uso

Se han dividido los casos de uso en dos diagramas, el primer diagrama, aquellos casos de uso que ejecute el ordenador de a bordo y el segundo diagrama aquellos casos de uso relativos al cliente Web.

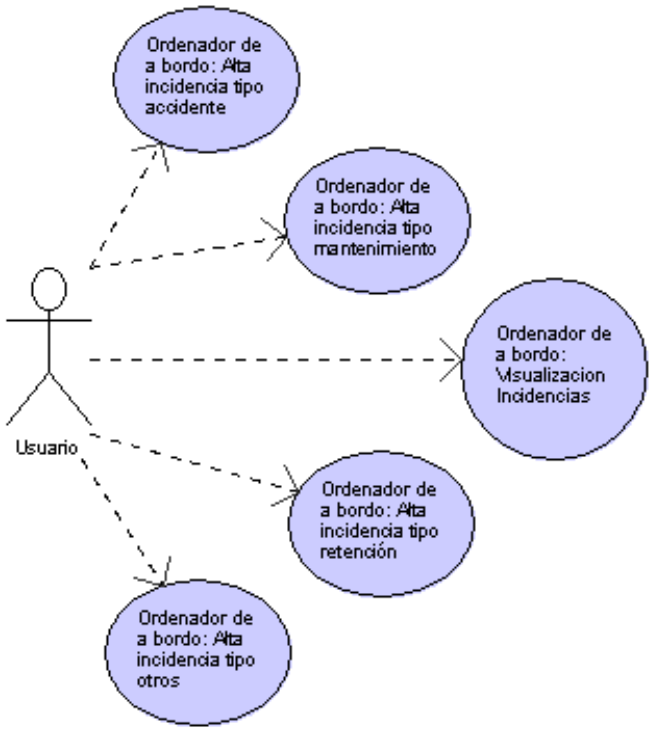

Figura 5.1. Casos de uso: Ordenador de a bordo

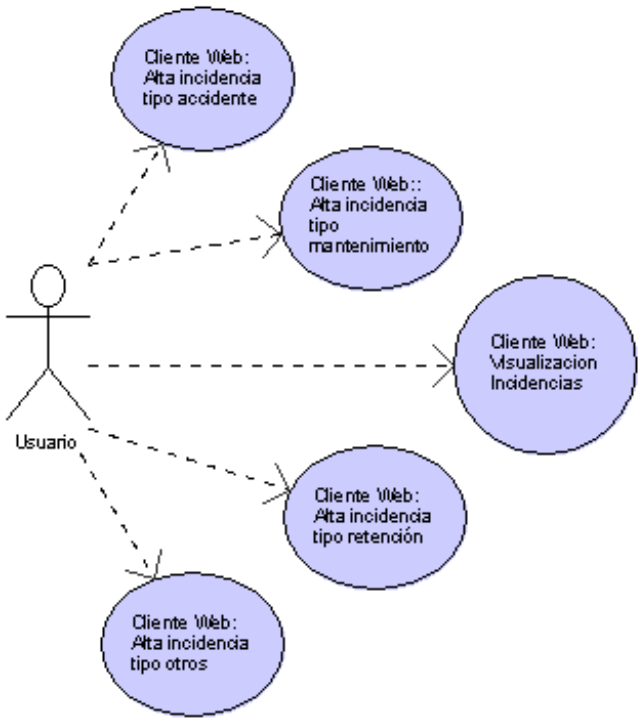

Figura 5.2. Casos de uso: Cliente web

A continuación se describirá brevemente el objetivo de cada uno de los casos de uso, para ello emplearemos una pequeña tabla con el nombre del caso de uso y la descripción.

Nombre del Caso de Uso
Ordenador de a bordo: alta incidencia accidente.
Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo accidente mediante el ordenador de a bordo.

Nombre del Caso de Uso
Ordenador de a bordo: alta incidencia mantenimiento.
Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo mantenimiento mediante el ordenador de a bordo.

Nombre del Caso de Uso
Ordenador de a bordo: alta incidencia retención.
Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo retención mediante el ordenador de a bordo.

Nombre del Caso de Uso
Ordenador de a bordo: alta incidencia otro.
Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo otros mediante el ordenador de a bordo.

Nombre del Caso de Uso
Ordenador de a bordo: visualización incidencias.
Descripción
El ordenador de a bordo indicará al usuario la incidencia de cada tipo más cercana.

Nombre del Caso de Uso
Cliente Web: alta incidencia accidente.

Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo accidente mediante el cliente Web.

Nombre del Caso de Uso
Cliente Web: alta incidencia mantenimiento.

Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo mantenimiento mediante el cliente Web.

Nombre del Caso de Uso
Cliente Web: alta incidencia retención.

Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo retención mediante el cliente Web.

Nombre del Caso de Uso
Cliente Web: alta incidencia otro.

Descripción
El usuario debe de tener la opción de dar de alta una incidencia de tipo otro accidente mediante el cliente Web.

Nombre del Caso de Uso
Cliente Web: visualización incidencias.

Descripción
El ordenador de a bordo indicará al usuario mediante un mapa las incidencias registradas en el sistema.

5.3 Análisis de Casos de Uso y Escenarios

A continuación, analizaremos los casos de uso y los escenarios, que deben de ser analizados en mayor detalle, para una correcta comprensión del caso de uso, para ello, en los siguientes subapartados, se detallaran más en profundidad, mediante tablas.

- Ordenador de a bordo: alta incidencia accidente.
- Ordenador de a bordo: alta incidencia mantenimiento.
- Ordenador de a bordo: alta incidencia retención.
- Ordenador de a bordo: alta incidencia otro.
- Ordenador de a bordo: visualización incidencias.
- Cliente Web: alta incidencia accidente.
- Cliente Web: alta incidencia mantenimiento.
- Cliente Web: alta incidencia retención.
- Cliente Web: alta incidencia otro.
- Cliente Web: visualización incidencias.

5.3.1 Ordenador de a bordo: alta incidencia accidente

Ordenador de a bordo: alta incidencia accidente	
Precondiciones	Ordenador de a bordo conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia accidente en el ordenador de a bordo 2. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 3. Se le indica al usuario que el alta se ha realizado correctamente 4. El sistema se conecta mediante Internet a un servicio Web que obtiene las incidencias registradas, con el objetivo de actualizar y recalcular las incidencias más cercanas
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el

	alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.2 Ordenador de a bordo: alta incidencia mantenimiento

Ordenador de a bordo: alta incidencia mantenimiento	
Precondiciones	Ordenador de a bordo conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia accidente en el ordenador de a bordo 2. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 3. Se le indica al usuario que el alta se ha realizado correctamente 4. El sistema se conecta mediante Internet a un servicio Web que obtiene las incidencias registradas, con el objetivo de actualizar y recalcular las incidencias más cercanas
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.3 Ordenador de a bordo: alta incidencia retención

Ordenador de a bordo: alta incidencia retención	
Precondiciones	Ordenador de a bordo conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia accidente en el ordenador de a bordo

	2. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 3. Se le indica al usuario que el alta se ha realizado correctamente 4. El sistema se conecta mediante Internet a un servicio Web que obtiene las incidencias registradas, con el objetivo de actualizar y recalcular las incidencias más cercanas
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.4 Ordenador de a bordo: alta incidencia otros

Ordenador de a bordo: alta incidencia otros	
Precondiciones	Ordenador de a bordo conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia accidente en el ordenador de a bordo 2. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 3. Se le indica al usuario que el alta se ha realizado correctamente 4. El sistema se conecta mediante Internet a un servicio Web que obtiene las incidencias registradas, con el objetivo de actualizar y recalcular las incidencias más cercanas
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.5 Ordenador de a bordo: visualización incidencias

Ordenador de a bordo: Visualización incidencias	
Precondiciones	Cliente Web conectado a Internet
Poscondiciones	Visualización de las incidencias.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario inicia el Sistema 2. El sistema comprobará esporádicamente las incidencias registradas a través del servicio Web 3. El usuario se posiciona en la opción de menú del nuevo modulo de Incidencias 4. El sistema filtra la incidencia más cercana por cada tipo 5. El sistema muestra por pantalla al usuario una incidencia de cada tipo, indicando la distancia y la dirección en la que se encuentra.
Variaciones	-
Excepciones	1. Fallo en el servicio Web: Se informa al usuario de que no se ha realizado correctamente la consulta al servicio Web. 2. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.6 Cliente Web: alta incidencia accidente

Cliente Web: alta incidencia accidente	
Precondiciones	Cliente Web conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia en el cliente Web 2. Selecciona el tipo de incidencia a persistir 3. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 4. Se le indica al usuario que el alta se ha realizado correctamente
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al

	usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.7 Cliente Web: alta incidencia mantenimiento

Cliente Web: alta incidencia mantenimiento	
Precondiciones	Cliente Web conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia en el cliente Web 2. Selecciona el tipo de incidencia a persistir 3. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 4. Se le indica al usuario que el alta se ha realizado correctamente
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.8 Cliente Web: alta incidencia retención

Cliente Web: alta incidencia retención	
Precondiciones	Cliente Web conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia en el cliente Web 2. Selecciona el tipo de incidencia a persistir 3. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 4. Se le indica al usuario que el alta se ha realizado correctamente
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el

	alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.9 Cliente Web: alta incidencia otros

Cliente Web: alta incidencia otros	
Precondiciones	Cliente Web conectado a Internet.
Poscondiciones	Incidencia registrada.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario pulsa la opción alta incidencia en el cliente Web 2. Selecciona el tipo de incidencia a persistir 3. El sistema se conecta mediante Internet a un servicio Web que persiste la incidencia indicada 4. Se le indica al usuario que el alta se ha realizado correctamente
Variaciones	-
Excepciones	1. Fallo en la base de datos: Se informa al usuario de que el alta no se ha llevado a cabo por un fallo en el sistema. 2. Fallo en el servicio Web: Se informa al usuario de que el alta no se ha realizado por un fallo en el sistema. 3. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.3.10 Cliente Web visualizar incidencias

Cliente Web visualizar incidencias	
Precondiciones	Cliente Web conectado a Internet
Poscondiciones	Visualización de las incidencias.
Actores	Iniciado y terminado por el usuario
Descripción	1. El usuario accede al cliente Web 2. El cliente Web consulta las incidencias registradas a través del servicio Web pertinente. 3. El sistema visualiza las incidencias en el cliente Web.
Variaciones	-

Excepciones	1. Fallo en el servicio Web: Se informa al usuario de que no se ha realizado correctamente la consulta al servicio Web. 2. Fallo en la conectividad a Internet: Se debe informar al usuario de que hay un problema con la conectividad a Internet.
Notas	-

5.4 Análisis de Interfaces de Usuario

5.4.1 Descripción de la interfaz.

El componente ordenador de a bordo mostrará al usuario dos pantallas, una pantalla *TFT* y una *LCD*. Además para actuar sobre el sistema, el usuario tendrá a su disposición una botonera física, que se encuentra justo debajo de la pantalla *LCD* y que contiene los siguientes botones:

- Botón OK: Su función es la de seleccionar opciones del submenú.
- Botón Arriba: Su función es la de acceder a la opción previa del submenú.
- Botón Abajo: Su función es la de acceder a la opción siguiente del submenú.
- Botón Izquierda: Su función es la de acceder a la opción previa del menú.
- Botón Derecha: Su función es la de acceder a la opción siguiente del menú.

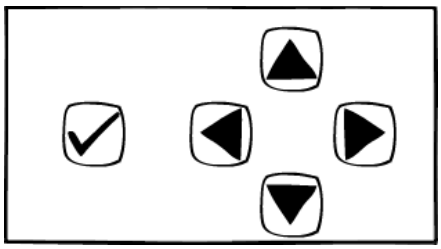

Figura 5.3 Prototipo de la botonera física

Todas las pantallas muestran cierta información común, que es la siguiente:

- Velocímetro digital con la velocidad instantánea en km/h.
- Odómetro con los kilómetros totales.

- Logotipo de la marca del vehículo.
- Dirección en la que circulamos, indicada por N, S, NW, NE, E, W, SW o SE.
- Información del *GPS*: numero de satélites con los que esta el sistema sincronizado, altitud (en metros sobre el nivel del mar), latitud, longitud.

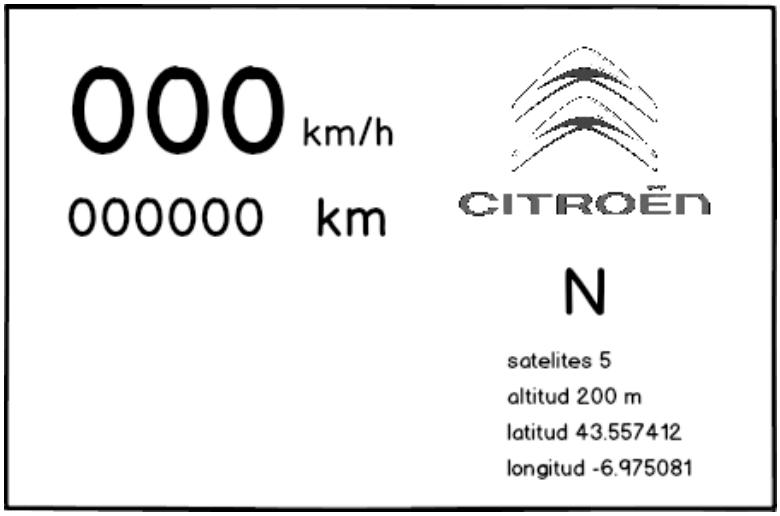

Figura 5.4 Boceto de Pantalla con la información común

La pantalla *TFT* incidencias, además de la información común muestra la incidencia más cercana por cada tipo. Esta pantalla es la que podemos ver a continuación.

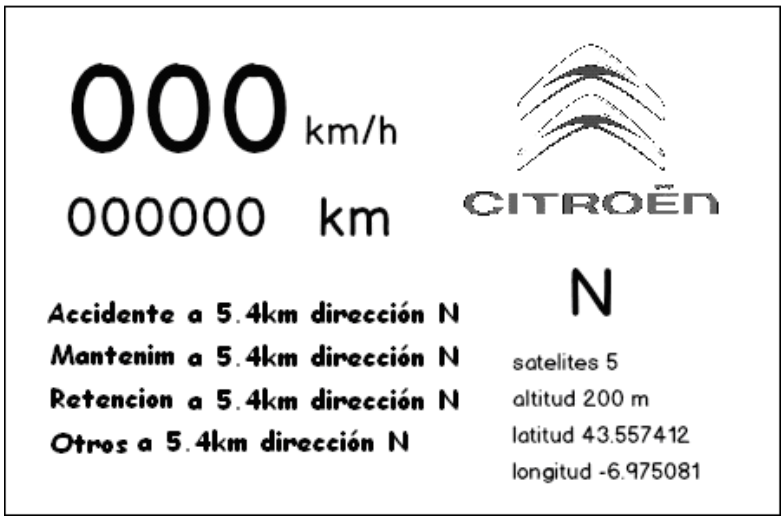

Figura 5.5 Boceto de Pantalla TFT Incidencias

Esta *pantalla TFT Incidencias*, se visualizará cuando el usuario se coloque en la *LCD* sobre la pantalla *LCD* de menú Incidencias que se muestra a continuación, a la que accede gracias al *botón izquierda* y al *botón derecha.*

Figura 5.6 Boceto Pantalla LCD menú incidencias

Esta opción de Menú tiene asociadas varios submenús, uno por cada tipo de incidente que se puede registrar:

- Alta Accidente: pulsando el botón OK se registrará una incidencia de tipo accidente y se mostrará la pantalla LCD de error o de alta realizada.
- Alta Mantenimiento: pulsando el botón OK se registrará una incidencia de tipo mantenimiento accidente y se mostrará la pantalla LCD de error o de alta realizada.
- Alta Atasco: pulsando el botón OK se registrará una incidencia de tipo atasco y se mostrará la pantalla LCD de error o de alta realizada.
- Alta Otros: pulsando el botón OK se registrará una incidencia de tipo otros y se mostrará la pantalla LCD de error o de alta realizada.

Figura 5.7 Boceto Pantalla LCD submenú alta accidente

Figura 5.8 Boceto Pantalla LCD submenú alta mantenimiento

Figura 5.9 Boceto Pantalla LCD submenú alta atasco

Figura 5.10 Boceto Pantalla LCD submenú alta Otros

Las últimas pantallas que forman este menú son las que corresponden al error de conexión y al alta realizada.

Figura 5.11 Boceto Pantalla LCD error de conexión

Figura 5.12 Boceto Pantalla LCD alta correcta

Finalmente, **el componente cliente web** consta de dos pantallas. La primera pantalla esta formada por un mapa y se muestran las diferentes incidencias con marcadores de colores, y un botón “Alta Incidencias” que nos permite acceder a la pantalla de las incidencias. Los diferentes colores representados son: Azul (Ubicación actual), Rojo (Accidente), Naranja (Atasco), Amarillo (Mantenimiento) y Morado (Otros).

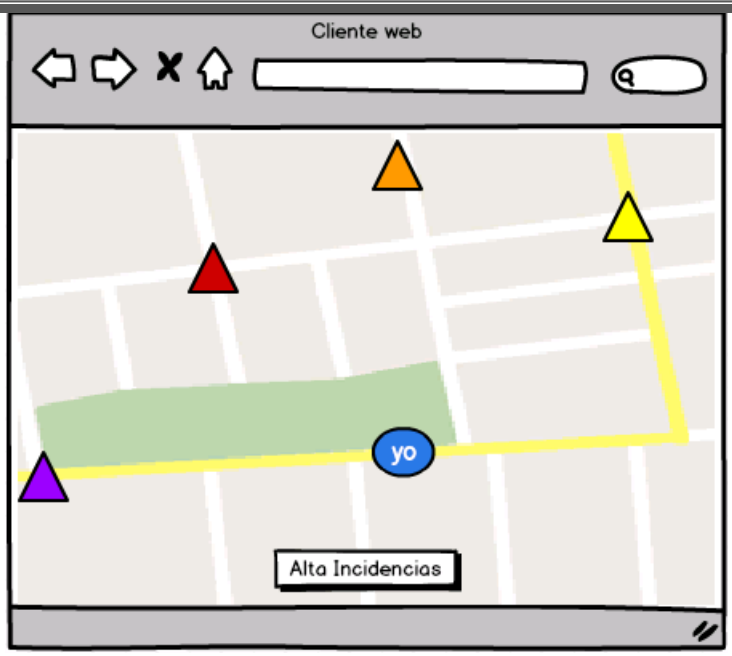

Figura 5.13 Prototipo cliente Web: mapa incidencias

La segunda pantalla esta formada por un fragmento de mapa en el que se muestra la posición que va a quedar registrada y cinco botones, cuatro de ellos indican el tipo de incidencia a registrar (botón Atasco, botón Mantenimiento, botón Accidente y botón Otros) y un quinto botón “Volver” para retornar a la pantalla anterior.

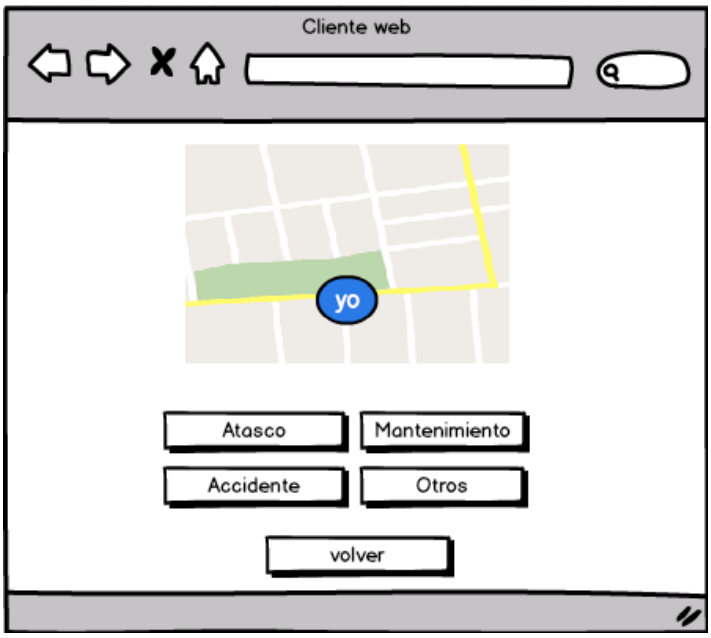

Figura 5.14 Prototipo cliente Web: Alta incidencias

5.4.2 Diagrama de Navegabilidad

5.4.2.1 Diagrama de navegabilidad: Ordenador de a bordo

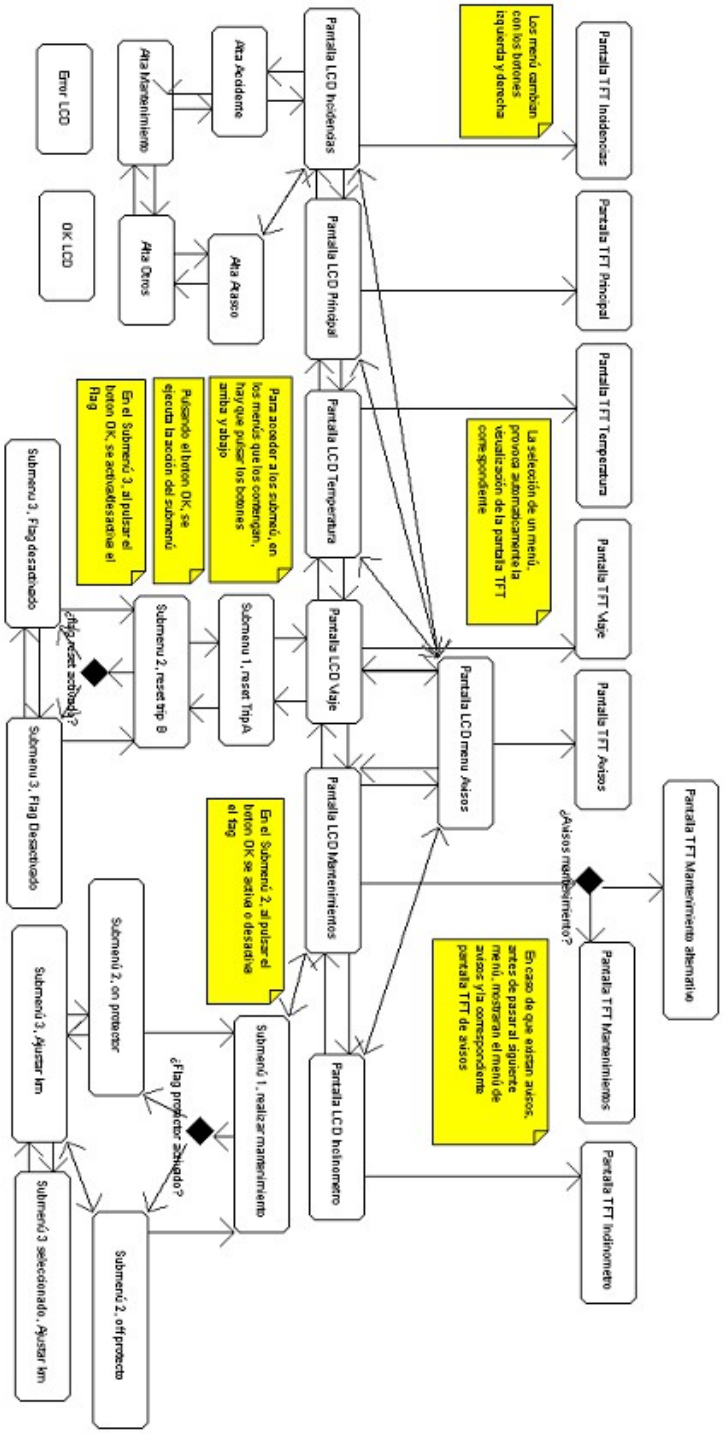

Figura 5.15 Diagrama de navegabilidad ordenador de a bordo

5.4.2.2 Diagrama de navegabilidad: Cliente Web

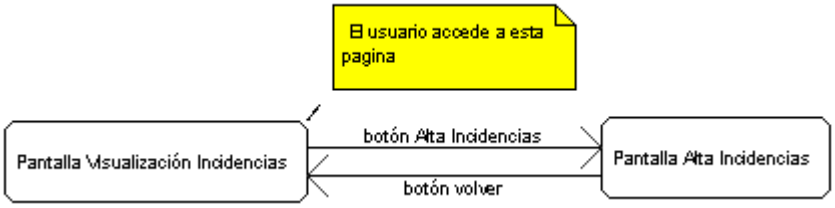

Figura 5.16 Diagrama navegabilidad cliente web

Capítulo 6. Diseño del Sistema

6.1 Arquitectura del Sistema

El paradigma que se pretende implementar en este trabajo final de máster como aplicación del internet de las cosas, consiste en una arquitectura distribuida y orientada a servicios (SOA) en la que varios clientes acceden a los servicios expuestos por operaciones de un servicio web. Por tanto, esta arquitectura estará compuesta de los siguientes componentes: operaciones expuestas como servicios web REST, una base de datos MySQL, un cliente web y por último una expansión del ordenador de a bordo que tomamos como base para este desarrollo y que dispondrá de la capacidad para realizar peticiones a servicios web REST. desplegarse en maquinas independientes.

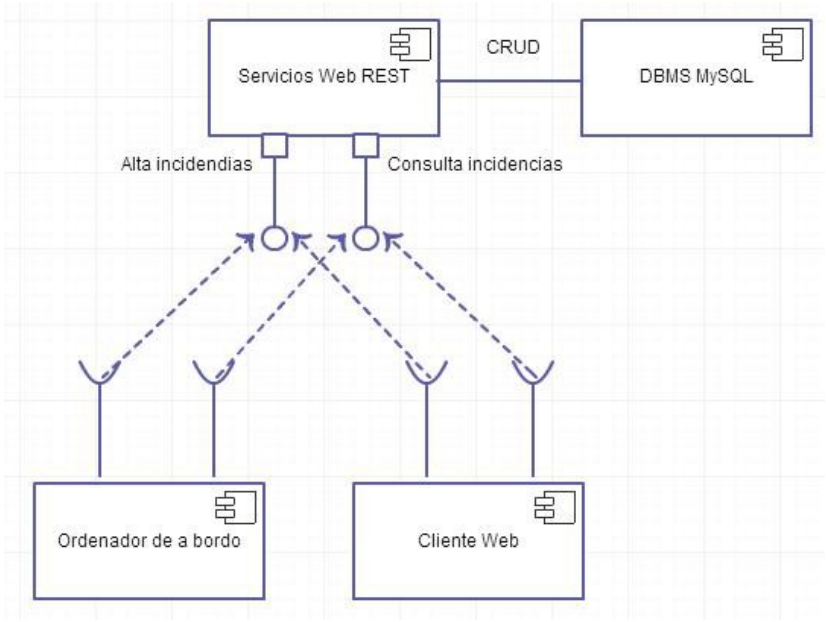

Figura 6.1 Diagrama de componentes

Por tanto en los siguientes apartados, se definirá la arquitectura del sistema, para ello, hablaremos de los diferentes componentes que componen el sistema de manera aislada.
Como podemos ver en el siguiente diagrama, tendremos un servidor *Apache* 2 para el cliente web, un servidor *Tomcat* para los servicios web y un servidor para la base de datos *MySQL y* finalmente el ordenador de a bordo consta de dos componentes software: el *firmware* del *Arduino* y el sistema de la *Raspberry pi*.

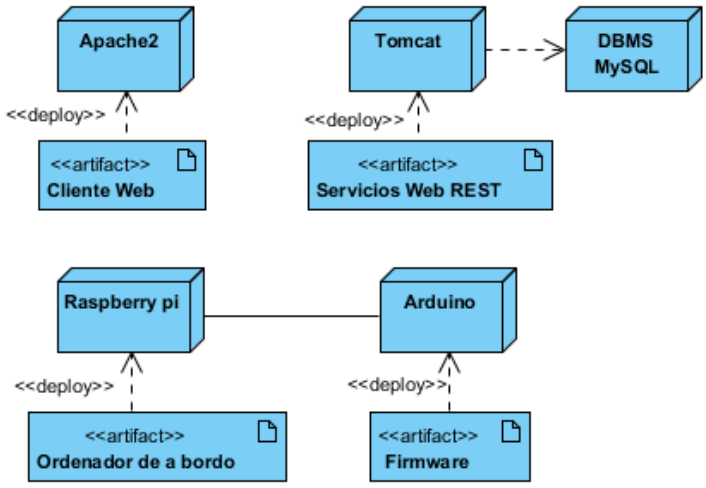

Figura 6.2 Diagrama de despliegue

6.1.1 Arquitectura de los Servicios Web

Los servicios Web diseñados, han sido diseñados para utilizar ser Servicios web RESTful, por las ventajas que nos aportan y por el buen resultado que han dado en la resolución de otros problemas *(Pautasso, Cesare, Wilde, Erik, Alarcon, Rosa (Eds.), 2013)*, y la superioridad con respecto a otras alternativas en determinados escenarios (Cesare Pautasso, O. Zimmermmann, F. Leymann, 2008). Estos servicios web han sido desarrollados en java, los servicios Web funcionaran bajo el protocolo *HTTP*. La arquitectura Software que se ha planteado para el desarrollo de estos servicios comienza con la división de los elementos que componen la aplicación por capas (patrón de diseño *Layer*), de tal manera que cada capa expone a la capa superior una interfaz (patrón de diseño *Facade*) y las diferentes dependencias entre capas, se inyectan con la ayuda del Framework *Spring-ioc* (patrón de diseño *Inversión of control*) Las capas están representadas por paquetes y los definidos para esta aplicación son las siguientes:

- Paquete Fachada: En este paquete se encuentra el Controlador de la aplicación, en este controlador se exponen los métodos que componen el catalogo de servicios *Web REST*. Por tanto en esta capa se validarán los datos de entrada y se delegarán aquellas funciones que no sean de su competencia a la capa inferior, la capa negocio. Los servicios que hemos definido son:

- /*getAllPointsOfInterest*. Esta operación devuelve una cadena JSON con todos los puntos de interés, dichos puntos pueden ser filtrados si se le pasa una coordenada con la ubicación del solicitante. Los parámetros que tiene esta operación son los siguientes:
  - Latitude, parámetro opcional, indica la latitud de la coordenada sobre la que se quiere realizar el filtrado.
  - Longitude, parámetro opcional, indica la longitud de la coordenada sobre la que se quiere realizar el filtrado.
  - Metros, parámetro opcional, indica la distancia máxima con respecto a la latitud y longitud indicada de los puntos que se desea filtrar.
- /*addPointOfInterest*. Esta operación inserta una nueva entidad en la aplicación, para ello se le indican los siguientes datos:
  - Latitude, parámetro obligatorio, indica la latitud de la incidencia que se desea insertar.
  - Longitude, parámetro obligatorio, indica la longitud de la incidencia que se desea insertar.
  - codeOrigin, parámetro obligatorio, código que indica el origen de la incidencia.
  - codeAlert, parámetro obligatorio, código que indica el tipo de incidencia que se desea dar de alta .

- Paquete de Lógica de Negocio. En esta capa se realizarán todas las transformaciones y filtrados necesarios sobre lo datos necesarios y la delegación de los accesos a los datos a la capa inferior, la capa de acceso a datos.
- Paquete de acceso a datos. En esta capa se realizan todas las comunicaciones con la base de datos, es decir, inserciones, borrados, actualizaciones y búsquedas sobre las entidades de la aplicación

Tendremos un paquete adicional *Util*, en el que tendremos una serie de clases auxiliares transversales a la aplicación, como son las clases *Util*, *Constantes*, y *FactoryPoint* (patrón de diseño *Factory*).

La base de datos que se va a utilizar estos servicios será una base de datos *MySQL*, y para facilitar la persistencia de datos se utilizará *Java Persistence API (JPA),* y los Framework *Spring-data* e *Hibernate.*

Finalmente se generará con *maven*, un *Java Archive(JAR)*, que gracias al Framework *spring boot*, se encarga de incrustar en el *JAR* un servidor *Tomcat 7*, cuando se ejecute el *JAR* generado, automáticamente se iniciará el servidor y estarán disponibles los servicios web *REST*, sin necesidad de ningún tipo de configuración adicional.

6.1.2 Arquitectura del ordenador de a bordo

La ampliación del ordenador de a bordo, desarrollada en *Python*, conserva la misma arquitectura hardware original con el añadido de conectividad a Internet por medio de un adaptador *WIFI* por *USB*, el motivo de la elección de este tipo de conexión, es que de momento esta funcionalidad es tan solo un prototipo académico, en un modelo de producción, se incorporaría al *Arduino* o a la *Raspberry pi* un modulo con conectividad *3G.*

Con respecto a la arquitectura software se conserva la idea original de una arquitectura multihilo con dos vistas: pantalla *LCD* y pantalla *TFT*. Para incluir las nuevas funcionalidades, se desarrollarán los siguientes puntos:

- Nuevo Hilo de ejecución paralela, cada cierto intervalo de tiempo realiza una llamada al servicio web *REST* para consultar la lista de puntos de interés, con el fin de poder indicar si hay incidencias cercanas.
- Nueva opción de menú en la pantalla *LCD* llamada “Incidencias”, con cuatro submenú: “Accidente”, ”Retención”, “Mantenimiento” y “Otros”. Cada una de estas opciones, dará de alta una nueva incidencia a través del servicio web, del tipo especificado.
- Nueva vista en la pantalla *TFT*, asociada a la nueva opción de menú, en ella se visualizarán la distancia con respecto a incidencias cercanas.

Por tanto para integrar estas nuevas funcionalidades, será necesario modificar el modelo de datos con la que trabaja en estos momentos la aplicación, el controlador *LCD* que gestiona las pulsaciones del usuario, y el servicio que actualiza la pantalla *TFT*, para visualizar la nueva informa.

6.1.3 Arquitectura del Cliente Web

El cliente Web está desarrollado en *HMTL5, CSS3* y *JavaScript*, nos ayudaremos del Framework *Bootstrap* y de la *API JavaScript* de *Google Maps* V3.

La idea que sobre la que órbita este cliente Web, es desarrollar dos páginas, una en la que se visualicen todas las incidencias y otra para dar de alta las incidencias nuevas de los cuatro tipos (Atasco, Accidente, Mantenimiento y Otros), por ello, el software estará organizado de la siguiente manera:

- Archivo *index.html*, este archivo representa a la pantalla de inicio del cliente Web, en ella se puede ver un mapa con todas las incidencias y un botón para acceder a la página de *alta.html.*
- Archivo *alta.html*, este archivo representa a la pantalla en la que se dan de alta las incidencias, contiene un mapa con la posición actual, un botón por cada tipo de incidencia (cuatro tipos de incidencias) y un botón para volver a la página *index.html*.
- Carpeta scripts, esta carpeta contiene todos los ficheros Javascript utilizados por el cliente Web.
    - Archivo index.js, este fichero Javascript, contiene el código que se ejecuta en la página index.html.
    - Archivo alta.js, este fichero Javascript, contiene el código que se ejecuta en la página alta.html.
- Carpeta styles, esta carpeta contiene todos los ficheros *css* utilizados por el cliente Web.
    - Archivo index.css, este fichero *css* se utiliza para dar estilo a la página *index.html.*
    - Archivo alta.css, este fichero *css* se utiliza para dar estilo a la página *alta.html.*

Finalmente este cliente se desplegará sobre un servidor *Apache* 2.

6.2 Diseño de Clases

6.2.1 Servicios Web

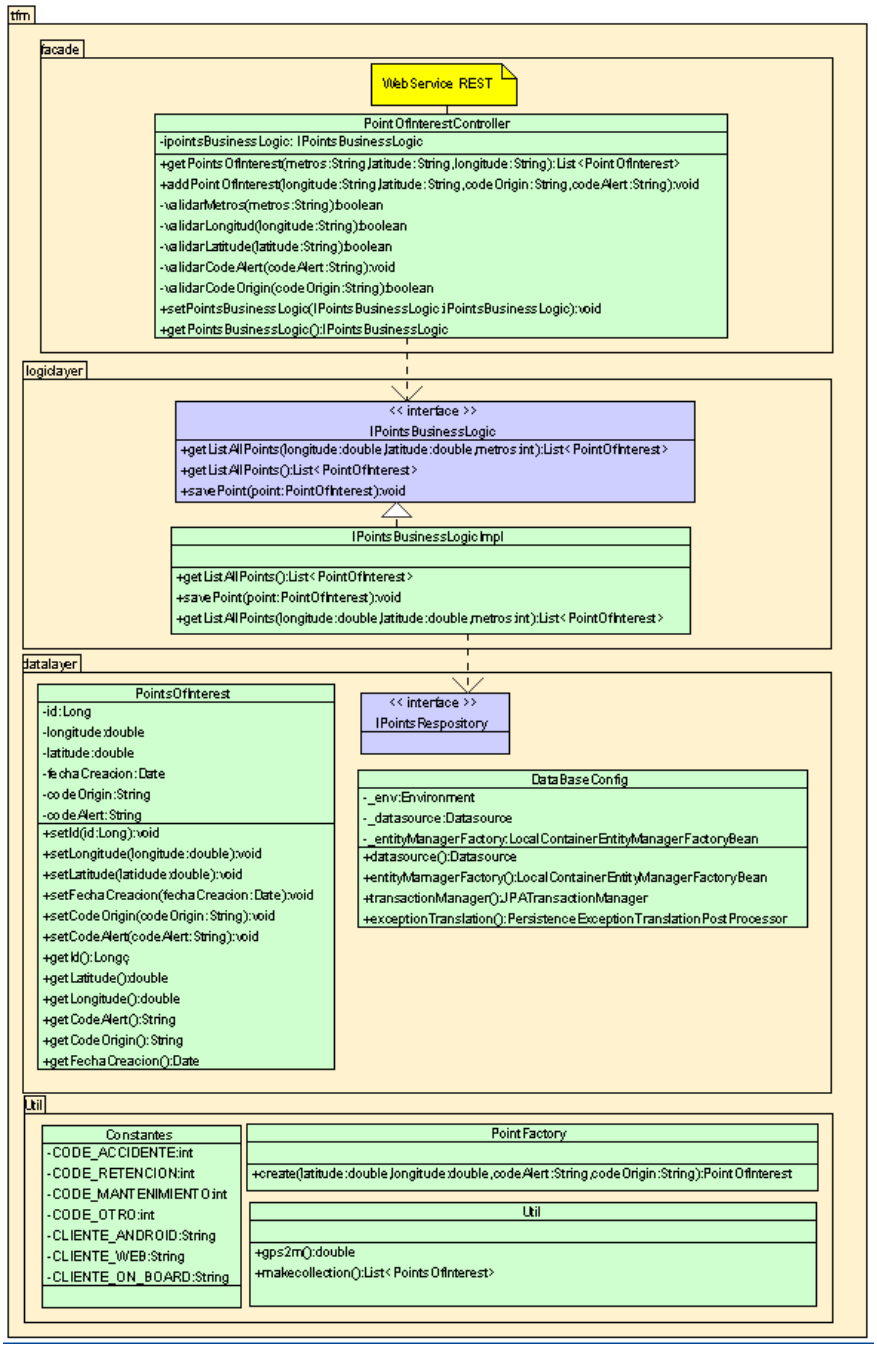

Figura 6.3 Diagrama de Clases: Servicios Web

6.2.2 Ordenador de a bordo.

En el siguiente diagrama de clases se muestra la estructura de la nueva funcionalidad implementada y las clases existentes con las que se relacionan las nuevas.

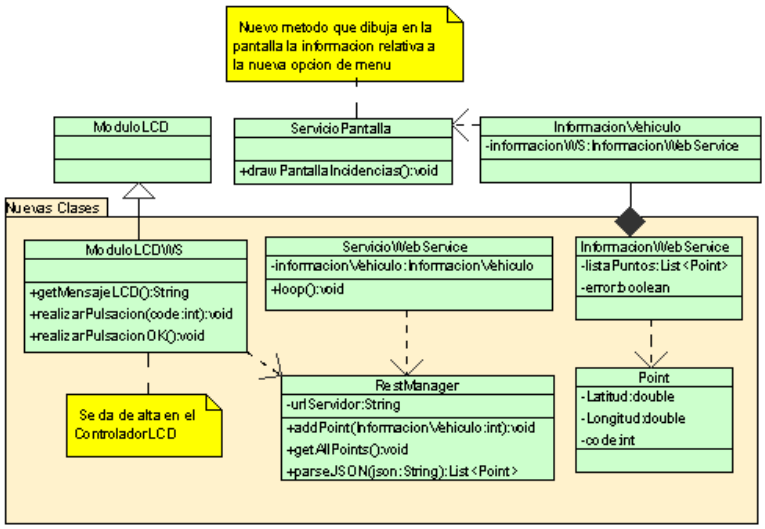

Figura 6.4 Diagrama de Clases: Hilo principal

6.2.3 Cliente Web

En este caso en lugar de un diagrama de clases, tiene más sentido un diagrama en el cual se relacionen los componentes software que componen el cliente Web.

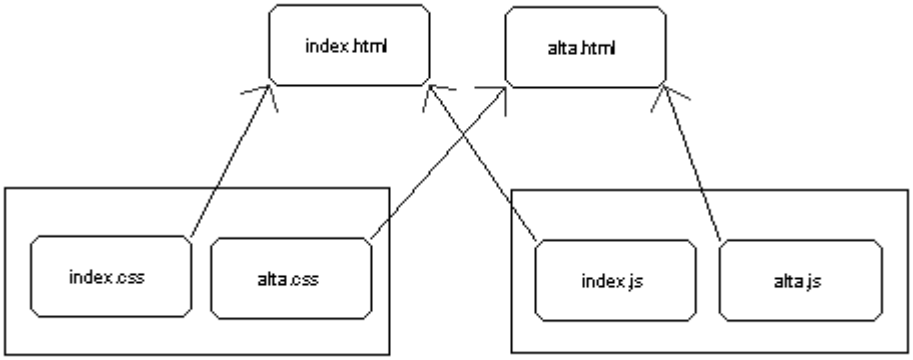

Figura 6.5 Diagrama de Páginas: Cliente Web

6.3 Diagramas de Interacción y Estados

6.3.1 Servicios Web

6.3.1.1 Operación addPointOfInterest

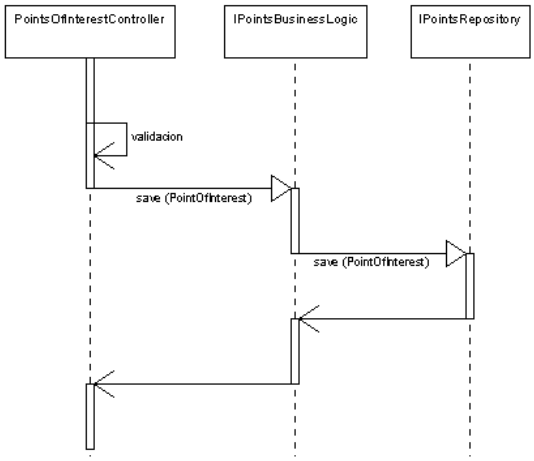

Figura 6.6 Diagrama de secuencia: Operación addPointOfInterest

6.3.1.2 Operación getPointsOfInterest

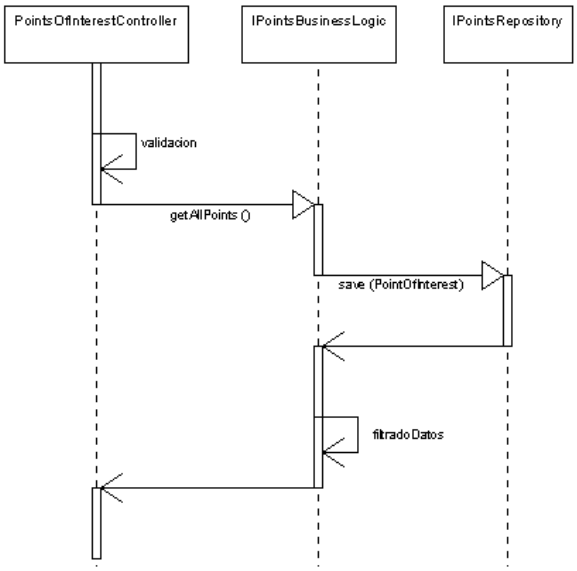

Figura 6.7 Diagrama de secuencia: Operación getPointsOfInterest

6.3.2 Ordenador de a bordo

6.3.2.1 Alta incidencias ordenador de a bordo

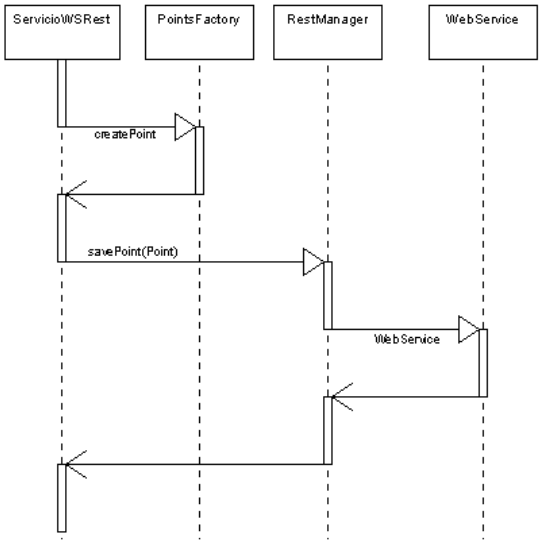

Figura 6.8 Diagrama de secuencia: Alta incidencias

6.3.2.2 Obtención de incidencias ordenador de a bordo

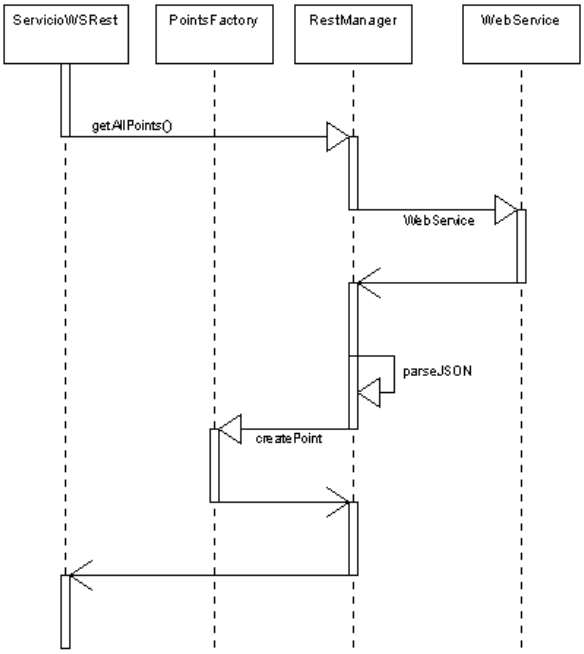

Figura 6.9 Diagrama de secuencia: Obtención de incidencias Ordenador de a Bordo

6.4 Diagramas de Actividades

Además de todos los diagramas de secuencia se han representado varios diagramas de actividades para representar varias funciones con el fin de que su interpretación quede más clara.

6.4.1 Ordenador de a bordo

6.4.1.1 Hilo principal

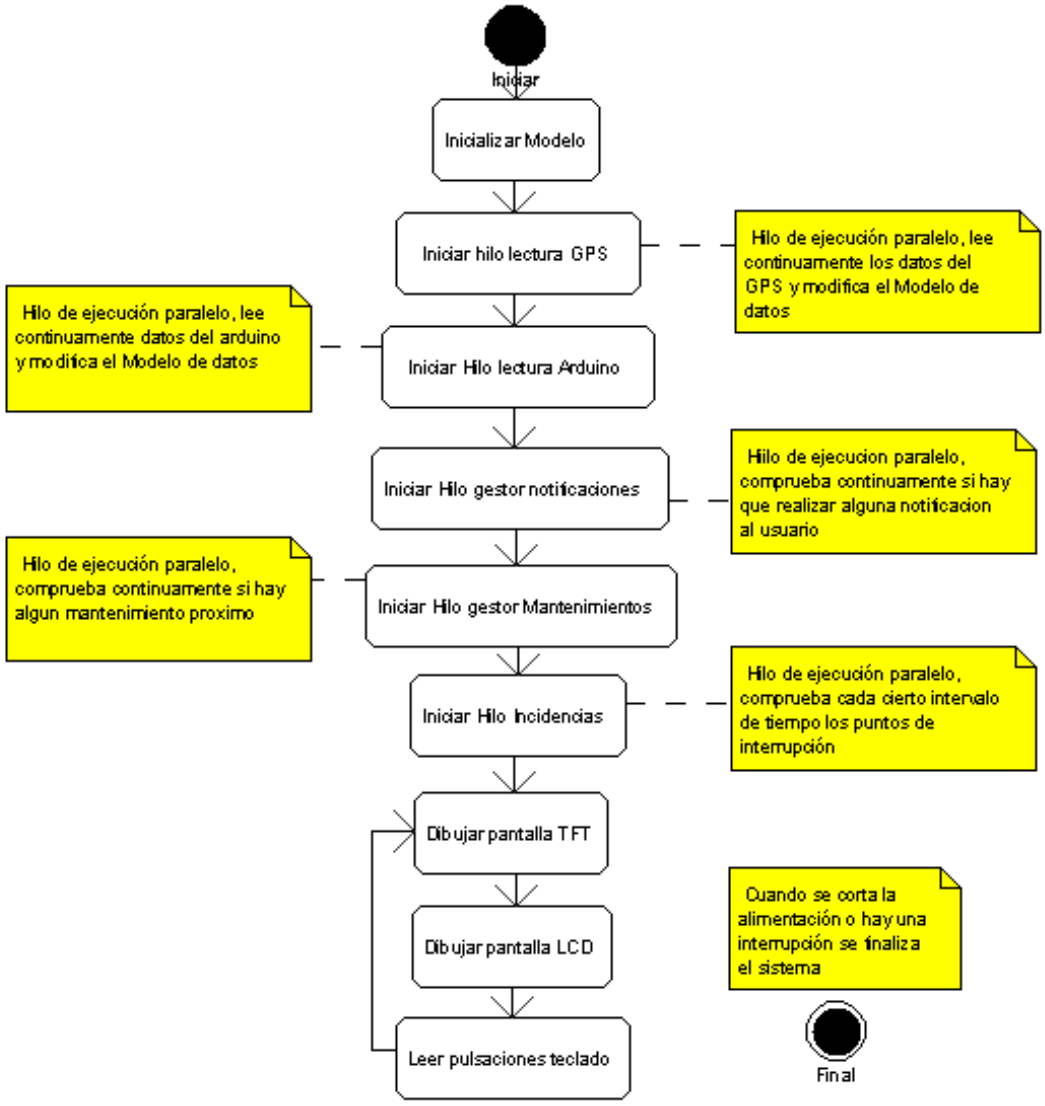

Figura 6.10 Diagrama de actividad: Hilo principal

6.4.1.2 Hilo Incidencias

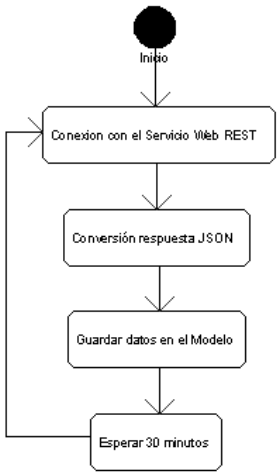

Figura 6.11 Diagrama de actividad: Hilo Incidencias

6.5 Diseño de la interfaz

6.5.1 Ordenador de a bordo

En esta sección se presentaran la nueva pantalla del ordenador de a bordo y los elementos que la componen, tanto la pantalla *LCD* como la pantalla *TFT*.

Figura 6.12 Logotipo de la marca del vehículo

Para conservar la coherencia con el resto del ordenador de a bordo la pantalla TFT se compone de una parte fija, que no variará entre pantallas, que contiene la siguiente información:

- Odómetro con los kilómetros totales.
- Logotipo de la marca del vehículo.

- Velocidad instantánea.
- Indicador de dirección.
- Información sobre el combustible.
- Información sobre el GPS: altitud, latitud, longitud, satélites con los que esta sincronizado el sistema.

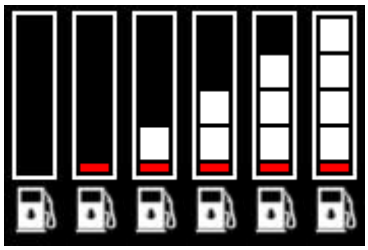

Figura 6.13 Indicador de nivel de combustible

La pantalla incidencias, muestra además de la información común a todas las pantallas, las incidencias más cercanas de cada tipo si las hubiere, indicando en cada caso, el tipo, la distancia y la dirección en la que se encuentra la incidencia.

Figura 6.14 Pantalla TFT Incidencias

La opción de menú incidencias, al seleccionarla se visualiza la pantalla incidencias en la pantalla TFT. Tiene asociadas varias opciones de submenú, una por cada posible incidencia, es decir: mantenimiento, retención, accidente y otros.

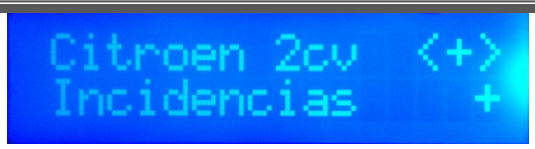

Figura 6.15 Pantalla LCD Incidencias

Figura 6.16 Pantalla LCD submenú Mantenimiento

Figura 6.17 Pantalla LCD submenú retención

Figura 6.16 Pantalla LCD submenú accidente

Figura 6.17 Pantalla LCD submenú Otro

Por ultimo en caso de que la comunicación con el servidor falle, se mostrará la siguiente pantalla *LCD*.

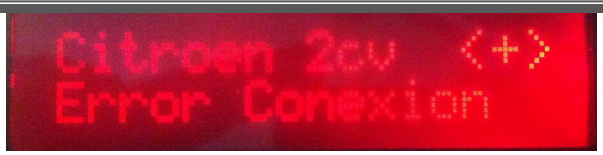

Figura 6.18 Pantalla LCD Error de conexión

6.5.2 Cliente web

En esta sección se presentaran las diversas pantallas que componen el cliente web, tal y como se ha comentado en secciones anteriores, se compone de dos pantallas.
La primera de las pantallas, se corresponde con la pantalla de visualización de las incidencias y se compone de un mapa, marcadores y un botón para acceder a la pantalla de alta de incidencias. Los marcadores se corresponden con los siguientes colores: Azul (Ubicación actual), Rojo (Accidente), Naranja (Atasco), Amarillo (Mantenimiento), Morado (Otros).

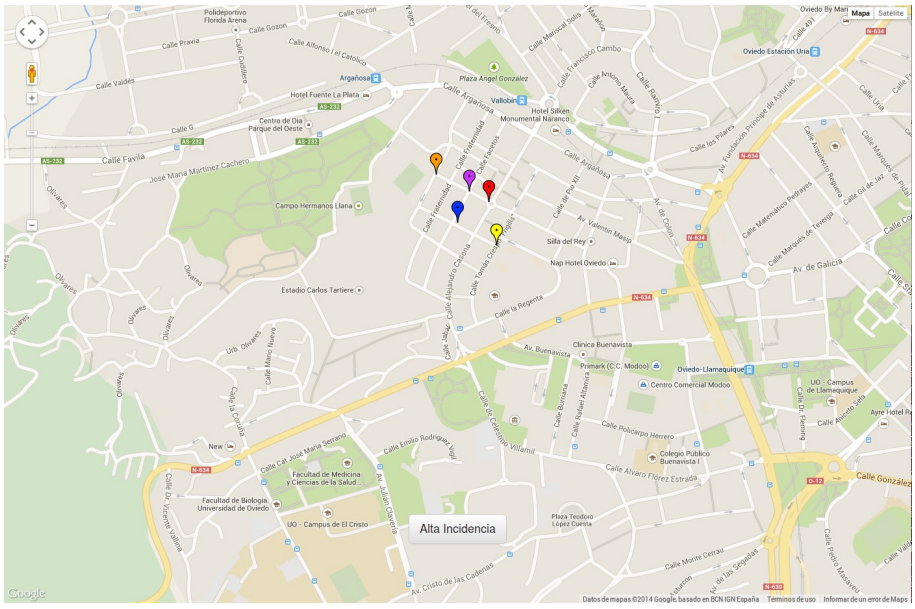

Figura 6.19 Pantalla Pringipal

La segunda pantalla incluye un mapa, sobre el que se puede pulsar para corregir la posición sobre la que se desea realizar la incidencia y un botón por cada tipo de incidencia.

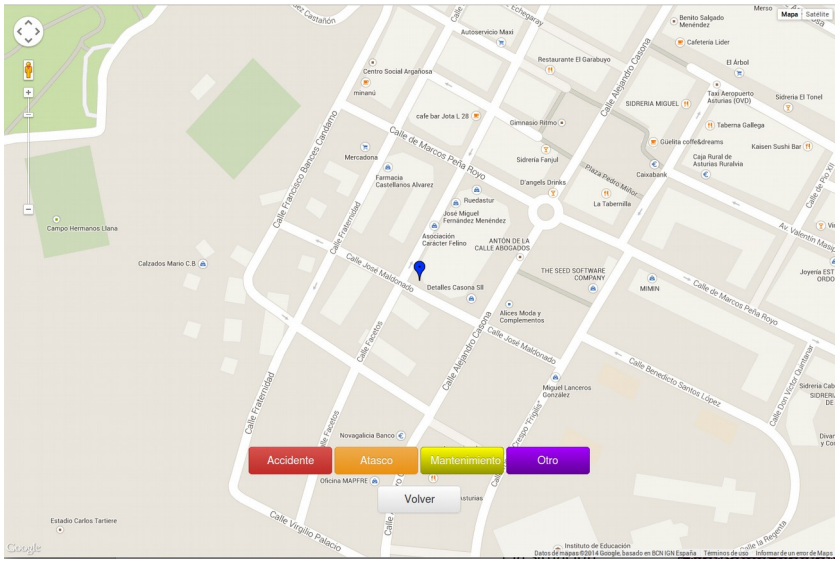

Figura 6.20 Pantalla Alta

6.5.3 Servicios Web

Los servicios Web, pese a no tener una interfaz visual, si tienen una interfaz con operaciones que pueden ser invocadas por distintos clientes, por tanto en esta sección se describe la interfaz que se ha desarrollado y que como podemos ver, no tiene prácticamente diferencias con respecto al prototipo original:

- **/getPointsOfInterest**. Esta operación devuelve una cadena JSON con todos los puntos de interés, dichos puntos pueden ser filtrados si se le pasa una coordenada con la ubicación del solicitante. Los parámetros que tiene esta operación son los siguientes:
  - Latitude, parámetro opcional, indica la latitud de la coordenada sobre la que se quiere realizar el filtrado.
  - Longitude, parámetro opcional, indica la longitud de la coordenada sobre la que se quiere realizar el filtrado.
  - Metros, parámetro opcional, indica la distancia máxima con respecto a la latitud y longitud indicada de los puntos que se desea filtrar.
- **/addPointOfInterest**. Esta operación inserta una nueva entidad en la aplicación, para ello se le indican los siguientes datos:
  - Latitude, parámetro obligatorio, indica la latitud de la incidencia que se desea insertar

- Longitude**,** parámetro obligatorio, indica la longitud de la incidencia que se desea insertar.
- codeOrigin**,** parámetro obligatorio, código que indica el origen de la incidencia
- codeAlert**,** parámetro obligatorio, código que indica el tipo de incidencia que se desea dar de alta

Capítulo 7. Implementación

7.1 Lenguajes de Programación y herramientas

En esta sección se indican los lenguajes de programación y las herramientas utilizadas durante la implementación de cada uno de los elementos software que componen nuestro prototipo.

En primer lugar, para el desarrollo de los servicios web, se ha utilizado el lenguaje de programación *Java* y el *framework Spring*. Respecto a las herramientas, se ha utilizado el entorno de desarrollo *eclipse*, la herramienta de control de paquetes *maven* y un repositorio *GIT* hospedado en *Bitbucket*.

En segundo lugar, en el desarrollo del cliente web, se han utilizado los lenguajes web *HTML5*, *CSS3*, *JavaScript y además el framework Bootstrap* y la *API* de *Google Maps V3*. Respecto a las herramientas, se han utilizado varios navegadores web, un editor de textos y un repositorio *GIT* hospedado en *Bitbucket*.

Finalmente en el desarrollo de la ampliación del ordenador de a bordo se ha utilizado el lenguaje *Python*. Las herramientas utilizadas para esta parte del desarrollo son, el entorno webIde y un repositorio *GIT* hospedado en *Bitbucket.*

7.2 Desarrollo del Sistema

7.2.1 Descripción de las Clases: servicios Web

7.2.1.1 Application.java

Para comenzar este apartado de descripción detallada de las clases relativas a los servicios Web, empezaremos definiendo la clase Application, esta clase se encuentra en el paquete *tfm*, y representa a la clase que se ejecuta cuando se inician los Servicios Web *REST*.

La aplicación se empaqueta en un archivo *Java Archive* (*JAR*) generado con *maven*, y tiene la particularidad de que *spring-boot*, inyecta un *Tomcat* 7, de tal manera que cuando se ejecuta el *JAR,* se inician el *tomcat* y quedan los servicios web disponibles por el puerto 8080.

Nombre	Tipo	Descripción	Hereda de…
Application	-	Clase Principal	-
Responsabilidades			
Número	**Descripción**		
1	Inicializa la aplicación		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público static	Void	main	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
En el ***pom.xml***, en la propiedad ***start-class***, se indica el valor ***tfm.Application***, para que se considere la clase principal.			

7.2.1.2 PointOfInterestController.java

La clase *PointOfInterestController* es el único controlador de la aplicación, se encuentra en el paquete *tfm.facade* y en esta clase se definen las operaciones que implementan los servicios Web *REST* expuestos, la configuración de los propios servicios Web, se realiza mediante las anotaciones a las que da soporte el framework *Spring-mvc*.

Esta clase se encarga de validar los datos de entrada y delegar la ejecución de la lógica de negocio a la capa de negocio mediante el atributo de IPointsBusinessLogic al que se le aplica el patrón inversión de control y se inyecta la implementación mediante el Framework *Spring-ioc*. Las operaciones que se definen en este servicio Web son las siguientes:

- *Método void addPointOfInterest*. Esta operación es la encargada de añadir nuevos *PointOfInterest* al sistema, y recibe los parámetros: latitud, longitud, código Origen y código de Alerta.
- *Método List<PointOfInterest> getPointsOfInterest*. Esta operación es la encargada de devolver el listado con todos los puntos del sistema, opcionalmente se puede pasar como parámetro el radio de distancia en metros y una ubicación especifica (latitud y longitud) para realizar un filtrado y que la lista solo muestre los que se encuentran a cierta distancia. Devuelve una Lista de objetos *PointsOfInterest* en formato *JSON*

Nombre	Tipo	Descripción
PointOfInterestController	-	Controlador

Responsabilidades	
Número	**Descripción**
1	Controlador de los Servicios Web.
2	Delega la lógica de negocio a la capa de negocio.
3	Validación de los datos de entrada: metros.
4	Validación de los datos de entrada: latitud.
5	Validación de los datos de entrada: longitud.
6	Validación de los datos de entrada: codeAlert.
7	Validación de los datos de entrada: codeOrigin.

Métodos		
Acceso | Modo	**Tipo de Retorno**	**Nombre**
Publico	List<PointOfInterest>	getPointsOfInterest
Público	Void	addPointOfInterest
Público	void	setPoinsBusinessLogic
Privado	Boolean	validarMetros
Privado	Boolean	validarLatitud
Privado	Boolean	validarLongitud
Privado	Boolean	validarCodeAlert
Privado	Boolean	validarCodeOrigin

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	IPointsBusinessLogic	iPointsBusinessLogic

Observaciones
Configuración de los servicios Web REST con anotaciones (spring-mvc) Inversión de control en IPointsBusinessLogic (spring-ioc).

7.2.1.3 IPointsBusinessLogic.java

La interfaz *IPointsBusinessLogic* se encuentra en el paquete *tfm.LogicLayer*, y representa a la interfaz que define las operaciones que se exponen al controlador *PointOfInterestController*, las cuales son las siguientes:

- Método *void savePoint*(). Este método lo invoca el controlador y es el encargado de delegar a la capa de acceso a datos la inclusión en la base de datos de un *PointOfInterest* registrado por un usuario a través del servicio Web *REST*.
- Método *List<PointOfInterest> getListAllPoints()*. Este método lo invoca el controlador y es el encargado delegar la obtención del listado de PointOfInterest a la capa de acceso a datos y devolverla
- Método *List<PointOfInterest> getListAllPoints* (*int metros, double latitude, double longitude*). A este método lo invoca el controlador y es el encargado delegar la

obtención del listado de *PointOfInterest* a la capa de acceso a datos, filtrarla en función de los parámetros dados y devolver la lista filtrada.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
IPointsBusinessLogic	Interfaz	Interfaz de negocio	-
Responsabilidades			
Número	**Descripción**		
1	Define la operación de negocio: savePoint		
2	Define la operación de negocio: getListAllPoints (sin parámetros)		
3	Define la operación de negocio: getListAllPoints (con parámetros)		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	Void	savePoint	-
Público	Void	getListAllPoints	-
Público	Void	getListAllPoints	Metros(int) Latitud (double) Longitud (double)
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
-			

7.2.1.4 IPointsBusinessLogicImpl.java

La clase *IPointsBusinessLogicImpl* se encuentra en el paquete *tfm.logicLayer* y representa a la implementación de la interfaz *IPointsBusinessLogic*. Esta clase se encarga de realizar la lógica de negocio, que en este caso consiste en filtrar los datos de salida y delegar la ejecución de las acciones que requieran de consultar o insertar *PointsOfInterest* en la base de datos, a la capa de acceso a datos mediante el atributo de *IPointsRepository* al que se le aplica el patrón inversión de control y se inyecta la implementación mediante el Framework *Spring-ioc*.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
IPointsBusinessLogicImpl	-	Implementación interfaz IPointsBusinessLogic	-
Responsabilidades			
Número	**Descripción**		
1	Implementa la operación de negocio savePoint		
2	Implementa la operación de negocio getListAllPoints (sin parámetros)		
3	Implementa la operación de negocio getListAllPoints (con parámetros)		

4	Delega el acceso a datos a la capa de acceso a datos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Publico	IPointsRepository	getRepositorio	-
Público	Void	setRepositorio	Repositorio (IPointsRepository)
Público	Void	savePoint	Punto (PointOfInterest)
Público	List<PointOfInterest>	getListAllPoints	
Público	List<PointOfInterest>	getListAllPoints	Metros (int) Latitud (double) Longitud (souble)
Privado	List<PointOfInterest>	filtrarLista	Lista (List<PointOfInterest>)
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	IPointsRepository	repositorio
Observaciones			
Inversión de control en IPointsRepository (spring-ioc).			

7.2.1.5 IPointsRepository.java

La interfaz *IPointsRepository* se encuentra en el paquete *tfm.dataLayer*, y representa a la interfaz que define las operaciones que se exponen a la capa de negocio, esta interfaz hereda de la clase *CrudRepository* que proporciona el Framework *Spring-data*, que gestiona por medio de *JPA* las entidades utilizadas en esta aplicación. *Spring-data* se encarga de implementar toda la clase, y para ello tan solo hay que parametrizar la entidad utilizada, que en este caso es *PointOfInterest* y el tipo de la clave primaria utilizada, que en esta aplicación es *long*.

Por tanto, aunque no declaremos ningún método en la interfaz, la clase que implementa a esta interfaz y que nos genera automáticamente el framework *Spring data*, contiene los siguientes métodos:

- Método *long count*(). Devuelve el número de *PointsOfInterest* que hay insertados en la base de datos
- Método *void delete*(*long id*). Borra de la base de datos el *PointOfInterest* que tenga asociado el identificador (id*)* que se ha indicado en el parámetro.
- Método *void delete* (*Iterable<PointOfInterest> lista*). Borra de la base de datos una colección de *PointOfInterest* que se le ha indicado por parámetro.

- Método *void delete* (*PointOfInterest* punto). Borra de la base de datos un *PointOfInterest* que se ha indicado como parámetro.
- Método *void deleteAll*(). Borra todos los *PointOfInterest* que hay registrados en la base de datos.
- Método *boolean exists* (*long id*). Busca un *PointOfInterest* en base a su identificador (*id)* y si existe en la base de datos, devuelve verdadero, en caso contrario devuelve falso.
- Método *Iterable<PointOfInterest> findAll()*. Busca en la base de datos y devuelve todos los *PointOfInterest* que hay registrados en la base de datos.
- Método *Iterable<PointOfInterest> findAll* (*Iterable<PointOfInterest> iterables*). Busca en la base de datos y devuelve una colección de *PointOfInterest* en la base de datos.
- Método *PointOfInterest findOne* (*long id*). Busca en la base de datos un *PointOfInterest* en base al identificador (id) que se le pasa como parámetro.
- Método *Iterable <PointOfInterest> save* (*Iterable <PointOfInterest> iterables*). Almacena en la base de datos un conjunto de *PointOfInterest*, y devuelve las entidades guardadas
- Método *PointOfInterest save (PointOfInterest punto).* Almacena en la base de datos un *PointOfInterest* y devuelve el *PointOfInterest* almacenado.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
IPointsRepository	-	Interfaz acceso a datos	CrudRepository
Responsabilidades			
Número	**Descripción**		
1	Define la interfaz de acceso a datos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
-			

7.2.1.6 DatabaseConfig.java

La clase *DatabaseConfig* se encuentra en el paquete *tfm.datalayer* y es la clase encargada de configurar el acceso a la base de datos. A los atributos *Environment*, *Datasource* y *LocalContainerEntityManagerFactoryBean* se les aplica el patrón inversión de control y se inyecta la implementación mediante el Framework *Spring-ioc*.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
DatabaseConfig	-	Configuración BBDD	-
Responsabilidades			
Número	**Descripción**		
1	Configurar acceso a base de datos		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	LCEMFB *	entityManagerFactoryBean	-
Público	Datasource	datasource	-
Público	JPATM **	transactionManager	-
Público	PETPP ***	exceptionTranslation	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	LCEMFB *	_entityManagerFactory
Privado	-	Environment	_environment
Privado	-	Datasource	_datasource
Observaciones			
La configuración de la base de datos lleva anotaciones que proporciona Spring-data Inversión de control en Datasource (spring-ioc). Inversión de control en Environment (spring-ioc). Inversión de control en LocalContainerEntityManagerFactoryBean (spring-ioc). * -LocalContainerEntityManagerFactoryBean ** - JPATransaction;amager *** - PersistenceExceptionTranslationPostProcessor			

7.2.1.7 PointsOfInterest.java

La clase *PointOfInterest* se encuentra en el paquete *tfm.dataLayer* y representa el modelo de datos que utilizamos en la aplicación.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
PointOfInterest	-	Clase Principal	-
Responsabilidades			
Número	**Descripción**		
1	Modelo de la aplicación.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**

Público	Date	getFechaAlta	-
Público	Void	setFechaAlta	Fecha (Date)
Público	Double	getLongitude	-
Público	Void	setLongitude	Longitude (double)
Público	Double	getLatitude	-
Público	Void	setLatitude	Latidude (double)
Público	String	getCodeOrigin	-
Público	Void	setCodeOrigin	codeOrigin (String)
Público	String	setCodeAlert	-
Público	Void	getCodeAlert	codeAlert (String)
Público	Long	gemid	
Público	Void	setId	Id (long)
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Privado	-	Date	fechaAlta
Privado	-	double	longitude
Privado	-	double	latitude
Privado	-	String	codeOrigin
Privado	-	String	codeAlert
Privado	-	Long	id
Observaciones			
Lleva las anotaciones pertinentes que requiere el Framework Hibernate para trabajar con JPA. El id es autogenerado.			

7.2.1.8 Constantes.java

La clase Constantes se encuentra en el paquete tfm.util y contiene todas las constantes utilizadas a lo largo de la aplicación.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Constantes	-	Contenedor Constantes	-
Responsabilidades			
Número	**Descripción**		
1	Contenedor de las constantes		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público	static	Int	CODE_ACCIDENTE
Público	Static	Int	CODE_RETENCION
Público	Static	Int	CODE_MANTENIMIENTO
Público	Static	Int	CODE_OTRO
Público	Static	String	CLIENTE_ANDROID
Público	Static	String	CLIENTE_WEB
Público	Static	String	CLIENTE_ON_BOARD

Observaciones
-

7.2.1.9 FactoryPoints.java

La clase FactoryPoints se encuentra en el paquete tfm.util y contiene las funciones necesarias para construir nuevos objetos de tipo PointOfInterest.

Nombre	Tipo	Descripción	Hereda de…
FactoryPoints	-	Clase Factory PointOfInterest	-
Responsabilidades			
Número	**Descripción**		
1	Creación de PointOfInterest		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público static	PointOfInterest	CreatePoint	Latitude (double) Longitude (double) codeOrigin (String) codeAlert (String)
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
-			

7.2.1.10 Util.java

La clase *Util* se encuentra en el paquete *tfm.util* y contiene las funciones auxiliares comunes a la aplicación, que en este caso son dos:

- Método *double gps2m (float lat_a, float lon_a, float lat_b, float lon_b).* Este método se encarga de calcular la distancia existente entre dos coordenadas definidas por su latitud y longitud, dicha distancia la devuelve en metros.
- Método *List<PointOfInterest> makeCollection (Iterable <PointOfInterest> iterable)*. Este método se encarga de convertir un Iterable de *PointOfInterest* en una *Lista* de *PointOfInterest*.

Nombre	Tipo	Descripción	Hereda de…
Util	-	Clase Utilidades	-
Responsabilidades			
Número	**Descripción**		
1	Implementar métodos de utilidades para la aplicación		
Métodos			

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
Público static	double	Gps2m	Lat_a (float) Lat_b (float) Lon_a (float) Lon_b (float)
Público static	List <PointOfInterest>	makeCollection	Iterable <PointOfInterest> inter
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
-			

7.2.2 Descripción de las Clases: Ordenador de a bordo

7.2.2.1 ServicioWebService.py

Para comenzar este apartado de descripción detallada de las clases relativas a la ampliación del ordenador de a bordo, empezaremos definiendo el nuevo hilo de ejecución paralela que realiza la llamada a la operación de consulta de los servicios Web a delegando la invocación a la clase *RestManager* cada cierto intervalo de tiempo, esta clase se denomina *ServicioWebService* y se debe de lanzar en el fichero *main.py*, cuando se inicializan y lanzan el resto de hilos de ejecución paralela. Para la inicialización de esta clase se le debe de pasar como parámetro una referencia a la clase *InformacionVehiculo*, que previamente ha de estar inicializada.

Nombre	Tipo	Descripción	Hereda de…
ServicioWebService	-	Clase Principal	-
Responsabilidades			
Número	**Descripción**		
1	Invocar cada al servicio Web de consulta de incidencias cada intervalo de media hora.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Publico	Void	loop	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
privado	-	InformacionVehiculo	informacionVehiculo
Observaciones			
El método *loop* invocará al servicio Web cada 30 minutos.			

7.2.2.2 ModuloLCDWS.py

La clase *ModuloLCDWS*, hereda de la clase *ModuloLCD* y representa a una nueva opción de menú, con sus correspondientes submenú, que se visualiza en la pantalla *LCD*. Para darlo de alta y que el sistema lo visualice es necesario actualizar la clase *ControladorLCD*, que es la clase que gestiona un listado con objetos de la clase *ModuloLCD*. Esta clase se encarga de indicarle al controlador que mensajes mostrar, que color mostrar y recibe las pulsaciones del usuario para tratarlas y realizar las acciones que el usuario desea.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
ModuloLCDWS	-	Modulo LCD dedicada a los Web Services	ModuloLCD
Responsabilidades			
Número	**Descripción**		
1	Indicar al controlador el mensaje a mostrar en la primera línea de la pantalla LCD		
2	Indicar al controlador el mensaje a mostrar en la segunda línea de la pantalla LCD		
3	Indicar al controlador el color de la pantalla a mostrar		
4	Resolver las pulsaciones del usuario		
5	Invocar al servicio Web si el usuario desea dar de alta una nueva incidencia		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Público	String	getMensajeLCD	-
Público	void	realizarPulsacion	Code:int
Público	Void	realizarPulsacionOK	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
-Dispone de las opciones de submenú: “Accidente”, “Mantenimiento”, “Retencion” y “Otros”, dependiendo de la elección, delegará en la clase RestManager para que realice la llamada al servicio Web adecuado para dar de alta la incidencia.			

7.2.2.3 InformacionWebService.py

La clase *InformacionWebService* representa al modelo de la nueva vista, esta clase hay que declararla e inicializarla como un nuevo atributo del modelo de nuestro ordenador de a bordo, la clase *InformacionVehiculo*.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
InformacionWebService	-	Modelo de la nueva vista “Incidencias”	-

Responsabilidades			
Número	**Descripción**		
1	Modelo de la vista “Incidencias”		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público	-	Lista<Points>	listaPoints
Público	-	String	error
Observaciones			
-			

7.2.2.4 Point.py

La clase *Point.py* representa una incidencia en el modelo de la aplicación, en el ordenador de a bordo, una lista de este tipo de objetos es almacenada en *InformacionWebService*, dicha lista se actualiza periódicamente gracias a la clase *ServicioWebService*, que se encarga de que el sistema invoque al servicio Web cada treinta minutos.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Point	-	Representación de una incidencia	-
Responsabilidades			
Número	**Descripción**		
1	Modelo que representa a una incidencia		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Público	-	Double	longitud
Público	-	Double	latitud
Público	-	String	codeAlerta
Observaciones			
-			

7.2.2.5 RestManager.py

La clase *RestManager.py*, es la encargada de realizar todas las llamadas a los servicios Web *REST* que requiera el ordenador de a bordo, concretamente los que hemos diseñado para este trabajo final de máster y además se encarga de transformar las cadenas *JSON* que devuelve el Servicio Web en una lista de *Point* para guardarlas en el modelo.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
RestManage	-	Invocador WS	-

r		REST	
Responsabilidades			
Número	**Descripción**		
1	Invocación a WS obtención lista de incidencias		
2	Invocación a WS añadir nueva incidencia		
3	Transformación de JSON a List<Point>		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
Publico estático	List<Point>	main	informacionVehiculo:informacionVehiculo
Publico estático	Void	addPoint	informacionVehiculo:InformacionVehiculo
Publico estático	List<Point>	parseJSON	Json:String
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
privado	-	String	__urlServer
Observaciones			

7.2.3 Descripción de las Páginas: cliente Web

En este caso dado que no tenemos clases, describiremos las páginas *HTML* y los componentes que las forman.

7.2.3.1 Index.html

El documento *index.html*, se corresponde con la pantalla del cliente que visualiza las incidencias, para ello se utiliza la API JavaScript de *GoogleMaps V3* y se accede a la operación correspondiente de los servicios Web. Por otra parte también incorpora un botón para acceder a la pagina de alta (documento *alta.html*).

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Index.html	HTML	Página Principal	-
Responsabilidades			
Número	**Descripción**		
1	Contiene un mapa de Google Maps		
2	Visualiza las incidencias registradas sobre el mapa		
3	Contiene un botón para acceder a la pantalla de altas		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

-	-	-	-
Observaciones			
- Los estilos de esta página están en el archivo styles/index.css - Los Javascript de esta pagina están en el archivo scripts/index.js			

7.2.3.2 Alta.html

El documento *alta.html*, se corresponde con la pantalla en la que el cliente da de alta incidencias, en ella el cliente visualiza la ubicación actual en un mapa, para ello se utiliza la *API JavaScript* de *GoogleMaps V3*.

Por otra parte también incorpora un botón para acceder a la pagina de inicio (documento *index.html*) y cuatro botones (uno por cada tipo de incidencia) para poder dar de alta una incidencia, teniendo en cuenta la posición actual.

Nombre	**Tipo**	**Descripción**	**Hereda de…**
Alta.html	HTML	Página Alta	-
Responsabilidades			
Número	**Descripción**		
1	Contiene un mapa de Google Maps		
2	Contiene un botón para el alta de una incidencia del tipo Accidente		
3	Contiene un botón para el alta de una incidencia del tipo Atasco		
4	Contiene un botón para el alta de una incidencia del tipo Mantenimiento		
5	Contiene un botón para el alta de una incidencia del tipo Otros		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
-	-	-	-
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
-	-	-	-
Observaciones			
- Los estilos de esta página están en el archivo styles/alta.css - Los JavaScript de esta pagina están en el archivo scripts/alta.js			

Capítulo 8. Evaluación

8.1 Pruebas Unitarias

En este primer apartado se detallarán las pruebas unitarias realizadas sobre los módulos software desarrollados, es decir: servicios web, ordenador de a bordo y cliente web. Estas pruebas se realizarán siguiendo la metodología caja negra.

8.1.1 Servicios Web

Pruebas unitarias: Servicios web	
Prueba: PU001	**Resultado Esperado**
Consulta de incidencias	El servicio web debe devolver todas las incidencias activas registradas en el sistema.
	Resultado
	Prueba correcta.
Prueba: PU002	**Resultado Esperado**
Alta de incidencia de tipo Accidente.	El servicio web debe de registrar una nueva incidencia de tipo Accidente, esta incidencia debe quedar registrado correctamente en el servicio web.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU003	**Resultado Esperado**
Alta de incidencia de tipo Atasco.	El servicio web debe de registrar una nueva incidencia de tipo Atasco, esta incidencia debe quedar registrado correctamente en el servicio web.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU004	**Resultado Esperado**
Alta de incidencia de tipo Mantenimiento.	El servicio web debe de registrar una nueva incidencia de tipo Mantenimiento, esta incidencia debe quedar registrado correctamente en el servicio web.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU005	**Resultado Esperado**
Alta de incidencia de tipo Otros.	El servicio web debe de registrar una nueva incidencia de tipo Otros, esta incidencia debe quedar registrado correctamente en el servicio web.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU006	**Resultado Esperado**

Alta de incidencia sin dato coordenada longitud.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU007	**Resultado Esperado**
Alta de incidencia sin dato coordenada latitud.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU008	**Resultado Esperado**
Alta de incidencia sin dato código de alerta.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU009	**Resultado Esperado**
Alta de incidencia sin dato código de origen.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU010	**Resultado Esperado**
Alta de incidencia con dato coordenada longitud en formato erróneo.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU011	**Resultado Esperado**
Alta de incidencia con dato coordenada latitud en formato erróneo.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU012	**Resultado Esperado**
Alta de incidencia con dato código alerta en formato erróneo.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU013	**Resultado Esperado**
Alta de incidencia con dato código origen en formato erróneo.	El servicio web debe de devolver un mensaje de error indicando un problema en la validación de los datos de entrada.
	Resultado Obtenido

	Prueba correcta.

8.1.2 Ordenador de a bordo

Pruebas unitarias: Ordenador de a bordo	
Prueba: PU014	**Resultado Esperado**
Navegación por los distintos menús del sistema, comprobando que se visualiza la nueva opción de menú incidencias.	Mediante los botones de izquierda y derecha se navega a través de las distintas opciones de menú y se comprueba que se visualiza la nueva opción de menú.
	Resultado
	Prueba correcta.
Prueba: PU015	**Resultado Esperado**
Navegación por medio del nuevo submenú de incidencias del sistema.	Mediante los botones de arriba y abajo se navega a través de las distintas opciones de menú y se comprueba que se visualizan todas las opciones de submenú.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU016	**Resultado Esperado**
Intento de alta sin satélites sincronizados.	El sistema debe de responder con un mensaje indicando que hay un problema al intentar realizar el Alta.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU017	**Resultado Esperado**
Intento de alta sin conexión a internet.	El sistema debe de responder con un mensaje indicando que hay un problema con la conexión a internet.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU018	**Resultado Esperado**
Alta incidencia tipo Accidente.	El sistema debe de registrar una incidencia de tipo Accidente e informar al usuario de que se ha realizado el alta correctamente.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU019	**Resultado Esperado**
Alta incidencia tipo Atasco.	El sistema debe de registrar una incidencia de tipo Atasco e informar al usuario de que se ha realizado el alta correctamente.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU020	**Resultado Esperado**
Alta incidencia tipo Mantenimiento.	El sistema debe de registrar una incidencia de tipo Mantenimiento e informar al usuario de que se ha realizado el alta correctamente.
	Resultado Obtenido
	Prueba correcta.

Prueba: PU021	**Resultado Esperado**
Alta incidencia tipo Otros.	El sistema debe de registrar una incidencia de tipo Otros e informar al usuario de que se ha realizado el alta correctamente.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU022	**Resultado Esperado**
Visualización de la incidencia de cada tipo más cercana.	El sistema debe de mostrar, cuando se ha seleccionado la opción de menú incidencias la incidencia de cada tipo mas cercana, indicando la dirección y la distancia, a la que se encuentra la incidencia de la posición actual.
	Resultado Obtenido
	Prueba correcta.

8.1.3 Cliente web

Pruebas unitarias: Cliente web	
Prueba: PU023	**Resultado Esperado**
Navegación desde la pantalla principal hasta la pantalla de alta.	Al pulsar el botón alta, se debe acceder desde la pantalla principal a la pantalla Alta.
	Resultado
	Prueba correcta.
Prueba: PU024	**Resultado Esperado**
Navegación desde la pantalla de alta hasta la principal.	Al pulsar el botón volver, se debe acceder desde la pantalla de alta a la pantalla principal.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU025	**Resultado Esperado**
Visualización de todos los tipos de incidencias en el mapa.	En la pantalla principal, se debe visualizar la posición actual y todas las incidencias activas registradas en el sistema.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU026	**Resultado Esperado**
Alta incidencia tipo Accidente.	Se registra a través de la página de alta una incidencia de tipo Accidente pulsando el botón correspondiente y el sistema, registra la incidencia y devuelve al usuario a la pantalla principal.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU027	**Resultado Esperado**
Alta incidencia tipo Atasco.	Se registra a través de la página de alta una incidencia de tipo Atasco pulsando el botón correspondiente y el sistema, registra la incidencia y devuelve al usuario a la pantalla principal.
	Resultado Obtenido
	Prueba correcta.

Prueba: PU028	**Resultado Esperado**
Alta incidencia tipo Mantenimiento.	Se registra a través de la página de alta una incidencia de tipo Mantenimiento pulsando el botón correspondiente y el sistema, registra la incidencia y devuelve al usuario a la pantalla principal.
	Resultado Obtenido
	Prueba correcta.
Prueba: PU029	**Resultado Esperado**
Alta incidencia tipo Otro.	Se registra a través de la página de alta una incidencia de tipo Otro pulsando el botón correspondiente y el sistema, registra la incidencia y devuelve al usuario a la pantalla principal.
	Resultado Obtenido
	Prueba correcta.

8.2 Pruebas de integración

Estas pruebas comprueba el funcionamiento global del sistema bajo distintas circunstancias y observamos la respuesta del sistema en su conjunto, trabajando lo mas similar posible al contexto de entorno final.

Pruebas integración	
Prueba: PI001	**Resultado Esperado**
Alta de incidencia desde el ordenador de a bordo.	Se debe de realizar una incidencia desde el ordenador de a bordo y se debe comprobar que en la base de datos, se ha registrado correctamente esta nueva incidencia.
	Resultado
	Prueba correcta.
Prueba: PI002	**Resultado Esperado**
Alta incidencia desde el cliente web.	Se debe de realizar una incidencia desde el ordenador de a bordo y se debe comprobar que en la base de datos, se ha registrado correctamente esta nueva incidencia.
	Resultado Obtenido
	Prueba correcta.
Prueba: PI003	**Resultado Esperado**
Visualización de incidencia en el cliente web, registrada desde el ordenador de a bordo.	Se registra una incidencia desde el ordenador de a bordo y se comprueba que se visualiza correctamente desde el cliente web.
	Resultado Obtenido
	Prueba correcta.
Prueba: PI004	**Resultado Esperado**
Visualización de incidencia en el ordenador de a bordo, registrada desde el cliente web.	Se registra una incidencia desde el cliente web y se comprueba que se visualiza correctamente desde el ordenador de a bordo.
	Resultado Obtenido
	Prueba correcta.

Capítulo 9. Conclusiones y trabajos futuros.

9.1 Conclusiones

Para este proyecto creo que se han cumplido todos los objetivos planteados al inicio del mismo y además se han aplicado los conocimientos y destrezas adquiridos en numerosas asignaturas de este máster de dirección e ingeniería de sitios web.

La implementación de una aplicación que utilice paradigma propuesto de comunicación vehículo a vehículos utilizando tecnologías web y arquitectura orientadas a servicios para notificar incidencias en las vías de circulación de manera colaborativa, es perfectamente viable con las herramientas y el hardware actual, prueba de ello es, que se ha conseguido el objetivo de desarrollar un prototipo funcional de este sistema como ejemplo de aplicación de nuestro paradigma.

Además hay que destacar lo extensible que es la utilización de este paradigma, ya que no solo es aplicable en la integración con un ordenador de a bordo a nivel académico, sino que además es posible desarrollar diferentes clientes que aprovechen la plataforma desarrollada con facilidad. Cualquier dispositivo con conexión a internet y con capacidad para invocar a servicios web REST, podría aprovechar nuestra plataforma.

Este prototipo garantiza el cumplimiento de otro de los objetivos propuestos, al estar basado en tecnologías estándar es posible disponer de un conjunto heterogéneo de clientes, en este trabajo se han visto varios: un cliente integrado en el ordenador de a bordo, un cliente web y en los anexos se propone un prototipo de aplicación Android. Por tanto será posible desarrollar en el futuro clientes basados en sistemas que aún no se han lanzado al mercado como Apple Carplay, Android Auto y Windows in the car, siempre y cuando mantengan retrocompatibilidad con las tecnologías web existentes en la actualidad.

Como hemos visto, la plataforma *Raspberry pi* y la plataforma *Arduino* ofrecen una gran versatilidad, la posibilidad de aplicar soluciones relacionadas con el internet de las cosas en los vehículos se multiplican gracias a estas nuevas plataformas, esto hace que las investigaciones estén limitadas únicamente por la imaginación del investigador, ya que a prácticamente a todos los objetos se les pueden plantear soluciones que aporten valor añadido al producto relacionado con el internet de las cosas.

Para concluir simplemente comentar que estoy muy satisfecho con el trabajo realizado y mi intención, por una parte es la de continuar trabajando en mi prototipo, aumentando su

funcionalidad, su estabilidad, su fiabilidad, mejorando aquellos aspectos que puedan ser mejorados. Por otra parte, continuaré investigando sobre distintas aplicaciones del internet de las cosas focalizándome en el campo de la automoción, y que realmente aporten valor añadido a los conductores y tengan futuro gracias a la utilización de paradigmas que utilicen arquitecturas orientadas a servicios como la expuesta en este trabajo.

9.2 Trabajo Futuro

Este proyecto y el ordenador de a bordo desarrollado tienen la virtud de que son muy extensibles, y son muchas las posibilidades que se pueden desarrollar e investigar, además de mis propias ideas, los usuarios que probaron la interfaz del prototipo proporcionaron ideas bastante interesantes. Además hablaremos de ideas futuras sobre los ordenadores de a bordo y algunas funcionalidades que se podrían llegar a implementar.

Esta sección la dividiremos en cuatro secciones; la primera sección dedicada a la nueva funcionalidad del ordenador de a bordo, la segunda sección dedicada a algunas ideas relacionadas con el internet de las cosas, la tercera sección dedicada al ordenador de a bordo y la cuarta y ultima sección dedicada al futuro de los vehículos conectados.

9.2.1 Mejoras en la nueva funcionalidad

Más tipos de incidencias. Los usuarios comentaron que sería interesante más tipos de eventos, quizás por situaciones meteorológicas adversas, carreteras cortadas e incluso se podrían incluir eventos personalizados. También puede resultado interesante aumentar el nivel de detalle sobre las incidencias

Inclusión de fotos. Esta característica consiste en incluir una cámara delantera al vehículo con el fin de en los casos que resulte interesante, enviar una imagen al sistema, para su posible visualización sobre los clientes que incorporen mapas.

Análisis de datos. Esta idea consiste en analizar los datos registrados por los usuarios, con el fin de encontrar patrones cuyo análisis pueda resultar interesante, por ejemplo, tramos con concentración de accidentes, retenciones o mantenimientos.

Control por voz. Los usuarios comentaban que resultaría muy útil para no perder la concentración en la carretera, que el sistema tuviese un control por voz, para gestionar las

altas y que mediante voz se notificase las incidencias en las vías de circulación cuando fuesen inminentes.

Filtrado de tipos de incidencia. Esta ampliación consiste en filtrar en el cliente web, los distintos tipos de incidencia con la intención de mostrar solo los tipos de incidencia seleccionados.

9.2.2 Mejoras sobre el internet de las cosas en la automoción

El internet de las cosas brinda un amplio abanico de posibilidades, algunas ideas que se me ocurren para investigar en el campo de la automoción:

Compartir datos de consumo. Que los vehículos compartiesen a intervalos de por ejemplo 100 kilometro información a cerca del consumo, características del motor, ubicación, con el objetivo de poder sacar estadísticas y determinar, por ejemplo zonas en las que lo vehículos consumen más combustible, consumos medios y en general, tener datos reales que puedan ser de utilidad general.

Calculo de rutas alternativas. Basándose en estimaciones de duración de incidentes, y de registros en la vía de circulación, se podrían incluir estos datos para calcular rutas alternativas si fuese necesario, en función de los tiempos estimados en los que se acabará la incidencia.

9.2.3 Mejoras en el ordenador de a bordo

Algunas lineas de trabajo relacionadas con mejoras para el ordenador de a bordo pueden ser las siguientes:

Pantalla Táctil. Para esta mejora es necesario remodelar la vista de las pantallas para adaptarlas a la nueva resolución y añadir los controles táctiles que faciliten la visualización de todas las pantallas.

Reproductor de audio en streaming. Esta mejora consiste en aprovechar las capacidades de la *Raspberry Pi* para dotar a nuestro sistema de la funcionalidad de reproductor de archivos de audio y la posibilidad de conectarse a servicios de musica en streaming.

Receptor radio. Esta mejora consiste en incorporar al sistema, un circuito receptor de señales de radio y utilizarlo para sintonizar diversos canales de audio, almacenar en memoria diversos canales de audio e incluso añadir la capacidad de escuchar radios online en *streaming*.

Mejorar el sistema de medición de las temperaturas interiores y exteriores. Esta idea consiste en mejorar el actual sistema de medición de temperaturas, en estos momentos tenemos un sensor de temperatura interior y otro exterior, para mejorar este sistema, lo propio sería añadir más sensores para obtener la temperatura media, y obtener un dato más preciso.

Navegador GPS. Durante las pruebas, al comentar que mi sistema incluía *GPS*, los *Betatester* preguntaban que como se ponían los mapas, lo cierto es que si bien inicialmente no me lo plantee, una posible ampliación podría ser incluir la visualización de mapas y la ubicación en el mismo, así como trazar rutas y mostrar información en el propio mapa.

Avisos por velocidad. Esta mejora consiste en incluir avisadores por superar distintas velocidad, la idea de esta ampliación es la de poder activar/desactivar avisos, que salten a ciertas velocidades, para prevenir al conductor en determinadas circunstancias.

Velocidades medias. Esta ampliación consiste en obtener las velocidades medias para las distancias recorridas en los odómetros A y B

Indicador de revoluciones. Uno de los elementos que no están presentes en este sistema, es el cuenta revoluciones, que tampoco está presente en el cuadro original del *Citroën 2cv*, pero existen dispositivos mecánicos que sirven para medir las revoluciones a las que gira el motor por unidad de tiempo, por tanto la idea sería utilizar uno de estos dispositivos, digitalizarlo, obtener el dato de las revoluciones por minuto y finalmente visualizar dicha información en las pantallas pertinentes.

Consumos. Esta ampliación consiste en sacar datos acerca de los consumos del vehículo, y mostrar el consumo medio realizado en la distancia marcada por los odómetros A y B y por supuesto el consumo medio de combustible para el odómetro total, para ello podríamos tener en cuenta las distintas distancias recorridas y calcular los litros de combustible que se han gastado con la ayuda del aforador de combustible.

Indicador de cambio de marcha eficiente. Esta ampliación no es posible desarrollarla hasta que se implementen otras ampliaciones (indicador de revoluciones y consumos), se trata de instalar varios sensores que le permitan al sistema saber que velocidad hay engranada en cada momento y mediante formulas matemáticas que relacionen varios parámetros, entre los que destacan, velocidad instantánea, potencia, revoluciones, par motor, marcha en la que se circula y consumo de combustible, sea capaz de indicarle al usuario si debe, subir o bajar de marcha para conseguir una circulación eficiente.

Indicador de cinturones de seguridad. Esta idea consiste en añadir testigos de presión a los asientos y un sensor que indique que están puestos los cinturones de seguridad, de tal

manera, que el testigo de presión detecte si el asiento esta ocupado o no, y en caso necesario informar al conductor de que no están puestos los cinturones de seguridad en los respectivos asientos.

Estado de la batería. Esta ampliación consiste en incluir un indicador con la carga y estado de la batería del vehículo

Integrar los indicadores de encendido de intermitentes y luces. En la actualidad, todos los vehículos disponen en su cuadro de mandos indicadores que indican si están encendidas las luces o los intermitentes, se podría implementar algo similar para nuestro sistema.

Iconos en pantalla de Avisos. Los usuarios que probaron el sistema transmitieron que sería interesante, tener un icono permanente visible en caso de que existan avisos, y que los mensajes de la pantalla avisos, podrían sustituirse por iconos descriptivos en lugar de los mensajes de texto que tenemos en estos momentos en la aplicación.

Integración del sistema de calefacción y la pantalla de temperaturas. El sistema de calefacción del 2cv, es relativamente sencillo y totalmente mecánico, pero ya se han desarrollado sistemas, para sustituir este sistema mecánico por motores eléctricos activados con un interruptor, por tanto la idea sería avanzar un paso más en este desarrollo y que sea la *Raspberry Pi*, en función de las temperaturas (interiores y exteriores), y de lo que indique el usuario el que active o apague este sistema de calefacción.

Interfaz *ODB2.* Siguiendo la trayectoria de los fabricantes de automóviles, esta ampliación consistiría en preparar una interfaz *ODB2*, que muestre la información a través de dicha interfaz.

Desarrollar placas *PCB (Printed Circuit Board).* Cuando el prototipo tenga las suficientes ampliaciones desarrolladas, la idea es desarrollar dos placas *PCB*, que se instalen, como un *shield*, una para la *Raspberry Pi* y otra para el *Arduino*, para evitar problemas de conexión de la protoboard.

9.2.4 El futuro de los ordenadores de a bordo

En este apartado abordaremos algunas ideas sobre el futuro de los ordenadores de a bordo, como ya hemos visto, han sido presentados en los diferentes salones del automóvil sistemas que conectan el smartphone con el ordenador de a bordo (carPlay, android Auto, Windows in the car), con este paradigma, se consiguen dos beneficios muy importantes, por una parte, los desarrolladores no necesitan hacer demasiado esfuerzo para adaptar sus

aplicaciones a los vehículos y por la otra, se consigue compartir la conexión a Internet. El problema que tiene este sistema es que hace a los usuarios dependientes de tener un terminal móvil, por tanto, aunque es una solución a corto plazo, en realidad se convertirá en una solución complementaria a la propia evolución de los ordenadores de a bordo y pienso que el futuro, pasa por sistemas integrados independientes, y que aunque se conecten con el smatphone para ciertas funciones, o incluso que también incorporen estas soluciones, sean completamente independientes y compatibles entre si, este aspecto es clave de estos sistemas integrados, el desarrollo de estos sistemas debe ser algo completamente estándar, para evitar el problema de la fragmentación y esfuerzos innecesarios a los desarrolladores y fabricantes.

Esta nueva generación de ordenadores de a bordo, no solo se aprovecharía de los servicios que proporcionan los fabricantes, sino que podría utilizar servicios de muchos grupos de interes (*Hsu, Robert C.-H., Shangguang, Wang (Eds.), 2014*), como por ejemplo del gobierno (concretamente Dirección General de trafico), servicios de terceros (aseguradoras, estaciones de servicio, talleres, sistemas colaborativos como el desarrollado en este trabajo...), información de otros vehículos (tecnología v2v *(Car2Car, 2014)*) o incluso mediante tecnologías de radio frecuencia comunicaciones con señales o incluso agentes de tráfico. Es importante conocer quien es la entidad adecuada para ofrecer la información que requiera cada servicio). Algunas ideas avanzadas que podrían incorporar estos sistemas son las siguientes.

Conducción automática y toma de decisiones sobre la ruta actuales. Esta idea consiste en la automatización de la conducción, ya se están investigando sistemas autónomos de conducción, pero además el propio sistema alterará la ruta en función del estado de las vías de circulación.

Gestión automática de las citas previas para mantenimientos. La idea de esta funcionalidad, es que cuando el vehículo necesite pasar por el taller, el sistema se encargue de solicitar automáticamente una cita previa, consultando con el usuario las preferencias en cuanto a fecha, hora y taller.

Gestión automática de perfiles. La idea de este funcionalidad es que cuando un conductor se suba a un vehículo cualquiera, el propio vehículo identifique al conductor, compruebe que es un conductor autorizado y se adapte a las preferencias del conductor, que podrían ser: climatización, reglaje de la suspensión, sintonización de emisoras, reglaje de espejos, reglaje del asiento, avisadores de velocidad.

Sistemas de prevención de accidentes. Esta idea consiste en que el propio sistema tome el control del vehículo en situaciones de peligro inminente para prevenir accidentes.

Sistemas de prevención de infracciones. Esta funcionalidad consiste en que el propio sistema impida al usuario cometer infracciones, como por ejemplo por exceso de velocidad limitando la velocidad máxima.

Sistemas de monitorización de infracciones. La idea de esta funcionalidad es que el sistema registrará cada infracción cometida y se lo notificará al usuario, y en caso de las infracciones muy graves, directamente a la Dirección General de Tráfico.

Sistema de monitorización para conductores. La idea de este sistema es la de monitorizar el comportamiento de los conductores, su estilo de conducción, de esta manera los buenos conductores, podrían tener recompensas en forma de descuentos en seguros, premios en forma de puntos en el carnet, descuentos en impuestos, de la misma forma, que los malos conductores, tendrían una serie de penalizaciones.

Monitorización de las condiciones del conductor. Mediante sensores biométricos, el sistema impediría al conductor arrancar el vehículo (por ejemplo: el conductor pretende arrancar su vehículo y supera la tasa de alcoholemia permitida), le obligaría a detenerse cuando fuese un riesgo para la seguridad vial (por ejemplo: el conductor muestra signos de cansancio extremo) , e incluso tomaría el control del vehículo para detenerse en zonas seguras bajo ciertas condiciones de peligro inminente y notificaría a las autoridades sanitarias pertinentes (por ejemplo: desvanecimiento del conductor).

Información complementaria en zonas de velocidad controlada. Los dispositivos utilizados por la DGT, en muchos casos (no todos, ya que existen los radares de tramo) En aquellas zonas de velocidad controlada por radar, el sistema enviará al radar los datos de la velocidad durante cierta distancia, esto sería más justo dado que no tendrían en cuenta solo la velocidad instantánea en un determinado momento, sino que se tendría en cuenta la velocidad media durante el tramo.

Capítulo 10. Bibliografía

10.1 Bibliografía

Guillermo Cueva-Fernandez, Jordán Pascual Espada, Vicente García-Díaz, Cristian González García, Nestor Garcia-Fernandez. (2014). Vitruvius: An expert system for vehicle sensor tracking and managing application generation. 2014, de Elsevier
Sitio web: http://www.sciencedirect.com/science/article/pii/S1084804514000605

Juan Antonio Corrales, Vicente Díaz y Hugo Valverde. (2014). Estudio Anual de coches conectados 2014. 2014, de Interactive Advertisin Bureau
Sitio web: http://www.iabspain.net/wp-content/uploads/downloads/2014/07/Informe-coches-conectados-2014.pdf

Telefónica. (2014). Connected Car Industry Report 2014. 2014, de Telefónica
Sitio web: https://m2m.telefonica.com/multimedia-resources/connected-car-industry-report-2014-espanol

FISITA. (2012). Proceedings of the FISITA 2012 World Automotive Congress. 2012, de Springer
Sitio web: http://www.springer.com/computer/communication+networks/book/978-3-319-11166-7

Forbes. (2014). The connected Car. 2014, de Forbes Sitio web:
http://www.forbes.com/pictures/mkk45ihlk/10-obstacles-for-connected-cars/

Car2Car. (2014). Car2Car Documentation. 2014, de Car2Car Communication Consortium
Sitio web: https://www.car-2-car.org/index.php?id=8

Hsu, Robert C.-H., Shangguang, Wang (Eds.). (2014). Internet of Vehicles - Technologies and Services. september 2014, de editorial Springer
Sitio web: http://www.springer.com/computer/communication+networks/book/978-3-319-11166-7

Gaglio, Salvatore, Lo Re, Giuseppe (Eds.). (2013). Advances onto the Internet of Things. 2013, de Springer Sitio web:
http://www.springer.com/engineering/computational+intelligence+and+complexity/book/978-3-319-03991-6

Pautasso, Cesare, Wilde, Erik, Alarcon, Rosa (Eds.). (2013). Free Preview REST: Advanced Research Topics and Practical Applications. 2013, de Springer
Sitio web: http://www.springer.com/engineering/signals/book/978-1-4614-9298-6

Cesare Pautasso, O. Zimmermmann, F. Leymann. (2008). RESTful Web Services vs. Big Web Services: Making the right Architectural Decision. 2008, del World Wide Web Conference Sitio web: http://www.jopera.org/docs/publications/2008/restws

Nel Gershenfeld, Raffi Krikorian and Danny Cohene (2004). The Internet of Thing. 2004, de Scientific American Sitio web: http://cba.mit.edu/docs/papers/04.10.i0.pdf

Autoscout24. (2014). Citroën 2cv. 2014, de Autoscout24.com
Sitio web: http://www.autoscout24.es/modellos/citroen/citroen-2cv/

Python. (2014). Python documentation. 2014, de Python Software Foundation
Sitio web: https://docs.python.org/2/

Raspberry Pi. (2014). Raspberry Pi quicl start guide. 2014, de Raspberrypi.org
Sitio web: http://www.raspberrypi.org/help/quick-start-guide/

Atlassian. (2014). Bitbucket 101. 2014, de Atlassian.com
Sitio web: https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101

Fried, L. (2014). Adafruit RGB Negative 16x2+keypad kit for Raspberry. 2014, de Adafruit.com
Sitio web: https://learn.adafruit.com/adafruit-16x2-character-lcd-plus-keypad-for-raspberry-pi

Cooper, T. (2014). Adafruit webIDE. 2014, de Adafruit.com
Sitio web: https://learn.adafruit.com/webide/overview

Arduino-a (2014). Arduino IDE Tutorial. 2014, de Arduino.cc
Sitio web: http://arduino.cc/es/Tutorial/HomePage#.Uxzkzz95MqM

Arduino-b(2014). Arduino language Reference. 2014, de Arduino.cc
Sitio web: http://arduino.cc/en/Reference/HomePage#.Uxzf_D95MqM

Arduino-c (2014). Arduino Playground. 2014, de Arduino.cc
Sitio web: http://playground.arduino.cc/

Couto, J. (2012). Arduino y los sensores de temperatura Ds18b20. mayo 2012, de Taller Arduino Sitio web: http://tallerarduino.com/2012/05/04/arduino-y-sensores-de-temperatura-ds18b20/

Gómez, M.-a (2013). Inclinómetro digital con Arduino. marzo 2013, de Código, Tips y programas Varios Sitio web: http://fuenteabierta.teubi.co/2013/03/inclinometro-digital-con-arduino-uso-de.html

Gómez, M.-b (2013). Velocímetro digital con Raspberry y venus GPS. mayo 2013, de Código, Tips y programas Varios.
Sitio web: http://fuenteabierta.teubi.co/2013/05/velocimetro-digital-con-raspberry-pi-y.html

Capítulo 11. Anexos

11.1 Manual de Instalación

11.1.1 Instalación del ordenador de a bordo

Dado que no se realizaron cambios a nivel de Hardware en el ordenador a excepción del modulo *WIFI*, no es necesario hacer ningún paso adicional para tener funcionando el ordenador de a bordo, por lo que la guía que contiene el proyecto final de carrera que se ha tomado como base para este proyecto contiene los pasos necesarios para la correcta configuración del proyecto.
Para dotar al sistema de conexión a Internet, simplemente se utilizará un receptor *WIFI* por *USB*, posteriormente en el modelo de producción podría incluirse un modulo de comunicación *3G*.

11.1.2 Despliegue de los Servicios WEB

Para la instalación del servicio Web debemos seguir fundamentalmente dos pasos, el primero la preparación de la base de datos, y el segundo, desplegar los servicios Web (en este prototipo hemos instalado la base de datos en el mismo equipo que la aplicación web, pero en el diseño original se instalan en maquina separadas):

- **PASO 1: Instalación de base de datos MySQL**, durante el proceso de instalación se indicará la contraseña *root* y se creará un *schema* que se denominará *tfm*, al iniciarse la aplicación, *Hybernate* se encargará de generar las tablas necesarias en el *schema tfm*.
- **PASO 2: instalación de los servicios Web**, dado que utilizamos *spring-boot*, lo único que tenemos que hacer es ejecutar el *JAR* generado (comando *java -jar*). Con esto este *JAR* se iniciará un servidor *Tomcat 7* que contiene los servicios Web generados.

11.1.3 Despliegue del cliente Web

Para el despliegue del cliente web, instalamos un servidor web, como por ejemplo *Apache* 2 y pegamos el contenido del cliente web en la carpeta */var/www/html* (el sistema operativo elegido es un *Ubuntu Server*, en otros sistemas operativos la ruta especificada puede variar).

11.2 Manual de Usuario: Ordenador de Abordo

En esta sección se detallara la ampliación del manual de usuario del sistema, y se explicarán las nuevas funcionalidades del sistema. Para esta ampliación, el usuario sigue teniendo disponible una pantalla *TFT*, una pantalla *LCD* y un teclado de 5 botones. Los botones que tiene el usuario disponible para poder interactuar con el sistema son los siguientes: Botón OK (1), botón Izquierda (2), botón Derecha (5), botón Arriba (3), botón Abajo (4).

Figura 11.1 Botones disponibles en la pantalla LCD

Para cambiar de menú, el usuario debe de pulsar los botones izquierda y derecha (“<+>”) para ir alternando entre los distintos menús disponibles, que son todos, menos el menú Avisos y Alertas, que en el caso de que exista activado algún aviso o alguna alerta, se intercalará en el cambio de menú.

Algunos menús, tienen asociados uno o varios submenú, el usuario podrá identificarlos, porque tendrán al final de la segunda línea de la pantalla *LCD* el símbolo “+”, para cambiar de submenú el usuario debe pulsar los botones arriba y abajo.

11.2.1 Menú Incidencias

Cuando se accede al menú de incidencias, se visualiza en la pantalla *LCD* el texto *"Incidencias"* y se visualiza en la pantalla TFT la información relativa a las incidencias registradas en las vías de circulación, concretamente la incidencia más cercana de cada tipo de incidencia (Accidente, Atasco, mantenimiento y otros) indicando la distancia a la que se encuentra (en kilómetros) y la dirección en la que se encuentra (N, NE, E, SE, S, SW, W, NW).

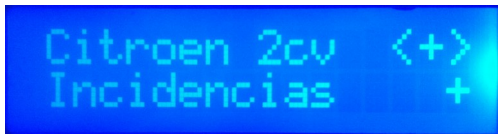

Figura 11.2 Pantalla LCD Incidencias

Figura 11.3 Pantalla TFT Incidencias

Esta opción de menú tiene asociadas varias opciones de submenú, cada una de ellas, da de alta una incidencia del tipo descrito por la opción de submenú, para realizar el alta, simplemente hay que pulsar el botón OK sobre la incidencia deseada.

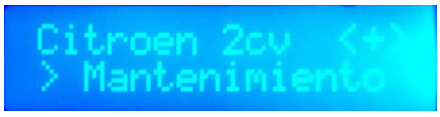

Figura 11.4. Pantalla LCD submenú 1

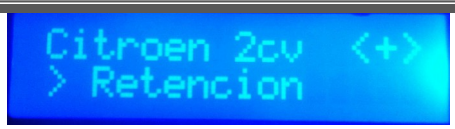

Figura 11.5 Pantalla LCD submenú 2

Figura 11.6 Pantalla LCD submenú 3

Figura 11.7 Pantalla LCD submenú 4

Una vez pulsada la opción seleccionada, el sistema mostrara en la pantalla un mensaje indicando que la acción se ha realizado correctamente. En caso contrario se mostrará un mensaje indicando que se ha producido un error.

Figura 11.8 Pantalla LCD Error

11.3 Manual de Usuario: Cliente Web

El cliente web desarrollado dispone de dos pantallas: la pantalla principal y la pantalla de altas, en esta sección explicaremos el funcionamiento de dichas pantallas, cada vez que se accede a una de las pantallas, es posible que se solicite al usuario compartir su situación con el navegador, para poder posicionarlo en el mapa.

Al acceder a la **pantalla principal** se visualizaran todas las incidencias registradas en el sistema y que aún estén activas con distintos colores, dichos colores representan a los distintos tipos de incidencia. Además dispone de un botón "*Alta Incidencia*" para acceder a la pantalla de alta.

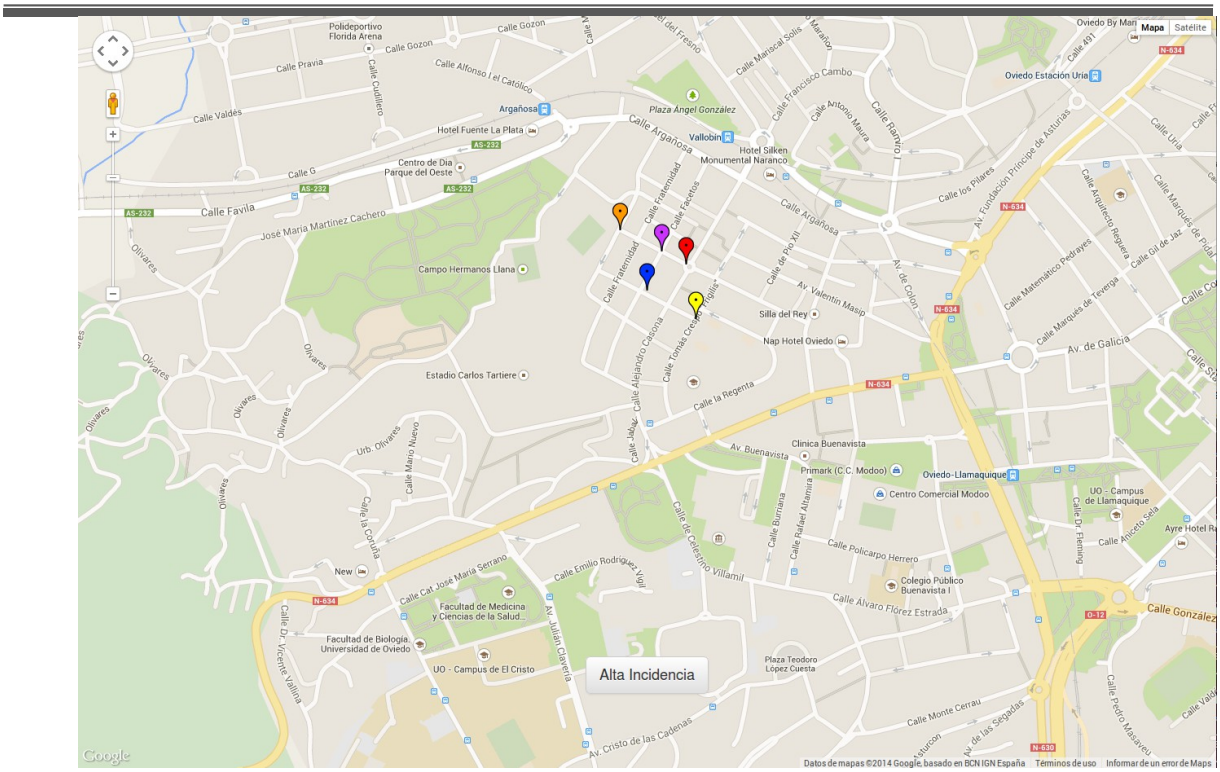

Figura 11.9 Pantalla principal

La pantalla de altas, está compuesta por cinco botones:

- Botón "*Volver*". Este botón sirve para devolver al usuario a la pantalla principal
- Botón "*Accidente*". Este botón sirve para dar de alta una incidencia de tipo "*Accidente*", una vez realizada con éxito se devuelve al usuario a la pantalla principal.
- Botón "*Atasco*". Este botón sirve para dar de alta una incidencia de tipo "*Atasco*", una vez realizada con éxito se devuelve al usuario a la pantalla principal.
- Botón "*Mantenimiento*". Este botón sirve para dar de alta una incidencia de tipo "*Mantenimiento*", una vez realizada con éxito se devuelve al usuario a la pantalla principal.
- Botón "*Otro*". Este botón sirve para dar de alta una incidencia de tipo "*Otro*", una vez realizada con éxito se devuelve al usuario a la pantalla principal.

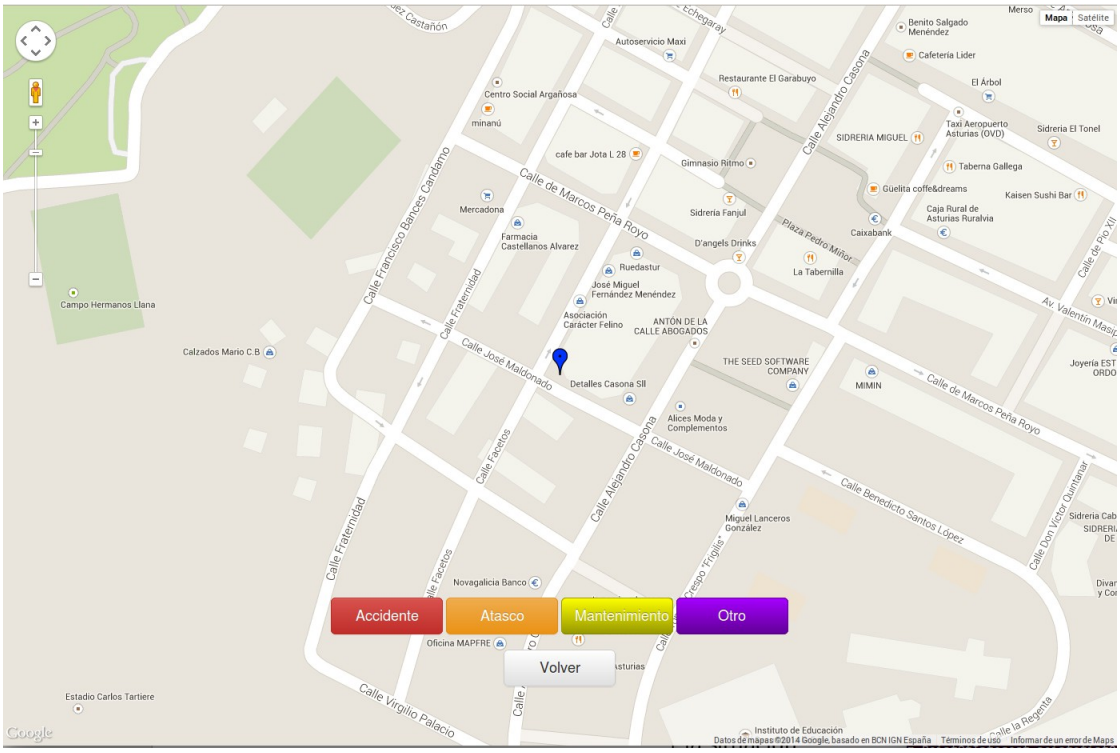

Figura 11.10 Pantalla alta

11.4 Aplicación Android

Este anexo presenta un prototipo de aplicación Android que se incluye como anexo, dado que no entraba en la planificación inicial. Esta aplicación podría implementarse para lograr una rápida adopción del sistema por parte de los usuarios, gracias al chip GPS y a la conectividad a Internet de los terminales Android, la aplicación se podría comunicar con los Servicios Web desarrollados en este proyecto y los usuarios podrían aprovechar las ventajas de este sistema de manera sencilla.

Además para complementar la funcionalidad, esta aplicación podría comunicarse con el ordenador de a bordo (Si se le implementase un modulo que retransmitiese datos por *bluetooth*), o con las centralitas de los vehículos modernos, mediante la interfaz ODB2. Por tanto el sistema consta de cuatro pantallas, dos relativas a las funciones clásicas del ordenador de a bordo y otras dos que representan a la nueva funcionalidad

La **primera pantalla** representa a la **visualización de incidencias** en el mapa, la aplicación muestra las incidencias, se realiza una llamada al servicio Web pertinente y se visualizan clasificadas por colores. Además la pantalla dispone de tres botones, uno para acceder a la pantalla que da de alta las incidencias y otros dos para cambiar de pantalla. Los colores de los marcadores se corresponden con: naranja (Mantenimiento), rojo **(**Accidente), violeta (Retención), Amarillo (Otro) y Azul (Posición Actual).

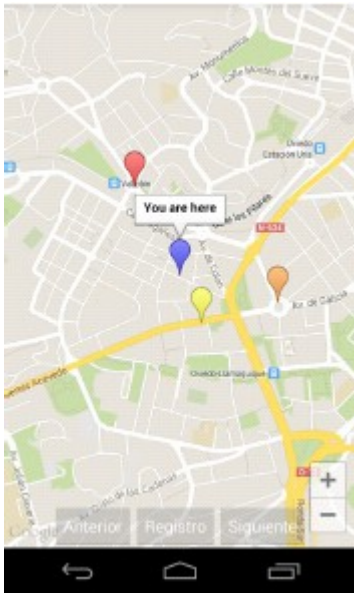

Figura 11.11 Pantalla TFT Principal

L a **segunda pantalla** corresponde al **alta de la incidencia**, en ella se puede ver la ubicación actual y cinco botones, uno de ellos, corresponde a la función de volver a la pantalla anterior (la pantalla de visualización de incidencias) y los otros cuatro, corresponden al tipo de incidencia que se quiera dar de alta: Accidente, Retención, Mantenimiento y Otros.

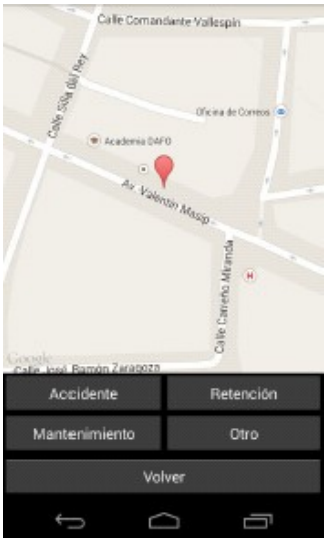

Figura 11.12 Pantalla Android 2

L a **tercera pantalla** corresponde con la **pantalla principal** del ordenador de a bordo y contiene información que se obtiene del ordenador de a bordo, dicha información se actualiza cada segundo. La pantalla es completada por dos botones (Anterior y Siguiente) que permiten la navegación entre las distintas pantallas de la aplicación. La información que podemos visualizar en esta pantalla es la siguiente:

- Velocímetro, en Km/h.
- Odómetro con la cantidad total recorrida por el vehículo.
- Hora.
- Temperatura exterior.

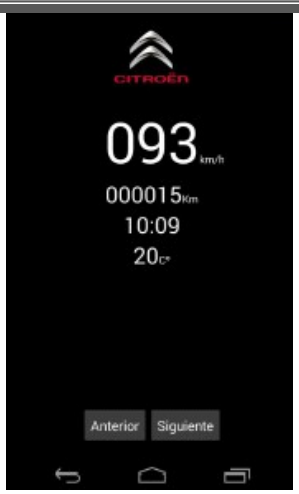

Figura 11.13 Pantalla Android 3

La **cuarta pantalla** se corresponde con la **información sobre el viaje** del ordenador de a bordo, y contiene información que se obtiene del sistema, dicha información se actualiza cada segundo, y concretamente podemos ver la siguiente información:

- Velocímetro, en Km/h
- Odómetro con la cantidad de kilómetros total recorrida por el vehículo.
- Odómetro con la cantidad de kilómetros recorrida durante el viaje A por el vehículo.
- Odómetro con la cantidad de kilómetros recorrida durante el viaje B por el vehículo.

Además dispone de dos botones (Reset), que permiten resetear los odómetros de viaje A y B. Al igual que las otras pantallas tiene de dos botones (Anterior y Siguiente) que permiten la navegación entre las pantallas que conforman la aplicación.

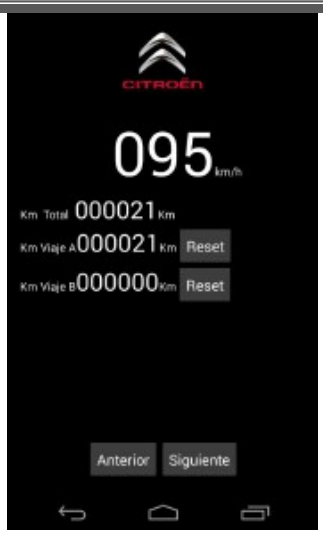

Figura 11.14 Pantalla Android 4

11.5 Descripción lenguajes de Programación

11.5.1 Python

Figura 11.15. Logotipo de Python

La versión de *Python* utilizada en el desarrollo de la ampliación del ordenador de a bordo es la versión 2.7, que viene por defecto instalada en la distribución *Raspbian* que hemos instalado en la *Raspberry Pi*. *Python* es un lenguaje de programación interpretado, cuya filosofía hace hincapié en que se disponga de una sintaxis muy limpia y que favorezca el código legible.

Se trata de un lenguaje de programación multiparadigma ya que soporta *orientación a objetos*, *programación imperativa* y en menor medida, *programación funcional*, en este caso, emplearemos el paradigma de la *orientación a objetos*. *Python* usa tipado dinámico y conteo de referencias para la administración de la memoria y, además es un lenguaje multiplataforma *(Python, 2014)*.

En la actualizad es administrado por la *Python Software Foundation* y posee una licencia de código abierto, denominada *Python Software Foundation License*, que es compatible desde la versión 2.1.1. con la licencia publica general de *GNU* e incompatible con ciertas versiones anteriores

Python fue desarrollado a finales de los ochenta por *Guido van Rossum* en el *centro para las matemáticas y la informática* (*CWI, Centrum Wiskunde & Informática*) en los Países ajos, como sucesor del lenguaje de programación ABC, capaz de interactuar con el sistema operativo *Amoeba*.

La última versión en pruebas es la *3.4.0 alpha4* (20 de octubre 2013) y las últimas versiones estables son:

- 3.3.2 (15 de mayo del 2013)
- 2.7.5 (15 de mayo del 2013)

Estas dos versiones se mantienen, dado que *Python2* y *Python3*, son versiones incompatibles pero la principal diferencia ente dichas versiones, lo explica muy bien una frase que se puede leer en la página Web de *Python*: "*Python 2.x es el legado, Python 3.x es el presente y futuro del lenguaje*."

Como curiosidad sobre este lenguaje, debe su nombre, a la afición de su creador por los humoristas británicos *Monty Python*.

11.5.2 Java

Figura 11.16 Logotipo de Java

Java, es el lenguaje utilizado para desarrollar la parte relativa a los servicios Web. *Java* es un lenguaje de programación de propósito general, orientado a objetos, basado en clases, concurrente, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Permite a los desarrolladores de aplicaciones escribir los programas una vez y ejecutarlos en cualquier dispositivo sin necesidad de ser recompilado (*write once, run everywere*). Java es uno de los lenguajes de programación más populares, especialmente, en aplicaciones Web (Cliente - Servidor).

En sus orígenes, java fue desarrollado por *James Gosling* de *Sun Microsystems* (*Sun* fue adquirida por *Oracle*) y publicado en 1995 como un componente fundamental de la plataforma java de *Sun* Microsystems. La sintaxis del lenguaje Java es en gran medida un derivado de C y C++, pero tiene menos utilidades de bajo nivel. Las aplicaciones java, son compiladas a *bytecode* y se ejecutan en cualquier Maquina Virtual Java, lo que hace que Java sea independiente de la arquitectura de la computadora sobre la que se ejecuta.

Sun Microsystem desarrolló la implementación de referencia original para los compiladores Java, maquinas virtuales y librerías de clases en 1991 y las público por la primera vez en 1995. A partir de mayo de 2007 Sun volvió a licenciar la mayoría de sus tecnologías java bajo licencia pública general de *GNU*. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, como por ejemplo el *compilador de Java de GNU* y el *GNU Classpath.*

11.5.3 Spring Framework

Figura 11.17 Logotipo de Spring Framework

Spring es un Framework para el desarrollo de aplicaciones y contenedor de inversión de control para la plataforma *java* y es una solución de código abierto. En este proyecto se ha utilizado para el desarrollo de los servicios Web. Su creador fue *Rod Johnson*, quien publicó el Framework en junio de 2003 bajo la licencia *Apache* 2.0. El primer lanzamiento importante fue la versión 1.0 en marzo del 2004, actualmente la versión actual es la 4.1.1.

Aunque las características fundamentales de *Spring* Framework pueden ser usadas en cualquier aplicación java, existen extensiones para la construcción de aplicaciones Web sobre la plataforma *JavaEE* que han llegado a ser consideradas por la comunidad como una alternativa, complemento y en algunos casos sustituto del modelo *EJB* (*Enterprise JavaBean*) . *Spring* Framework comprende diversos módulos que proveen un rango de servicios:

- Contenedor de Inversión de control, que permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo mediante la inyección de dependencias.
- Programación orientada a aspectos.
- Acceso de datos.
- Gestión de transacciones.
- Modelo vista controlador, un Framework basado en *http* y *servlets*, que provee herramientas para la extensión y personalización de aplicaciones Web y servicios *REST*.
- Framework de acceso remoto, permite la importación y exportación estilo *RPC*, de objetos Java a través de redes que soporten *RMI*, *CORBA* y protocolos basados en *HTTP* incluyendo servicios web *SOAP*.
- Convención sobre configuración.

- Procesamiento por lotes.
- Autenticación y autorización, procesos de seguridad configurables que soportan un rango de estándares, protocolos, herramientas y practicas a través del subproyecto *Spring Security*.
- Administración Remota. Configuración de visibilidad y gestión de objetos java para la configuración local o remota vía *JMX*.
- Mensajes. Registro configurable de objetos receptores para el consumo de mensajes.
- Testing. Soporte de clases para el desarrollo de unidades de prueba e integración.

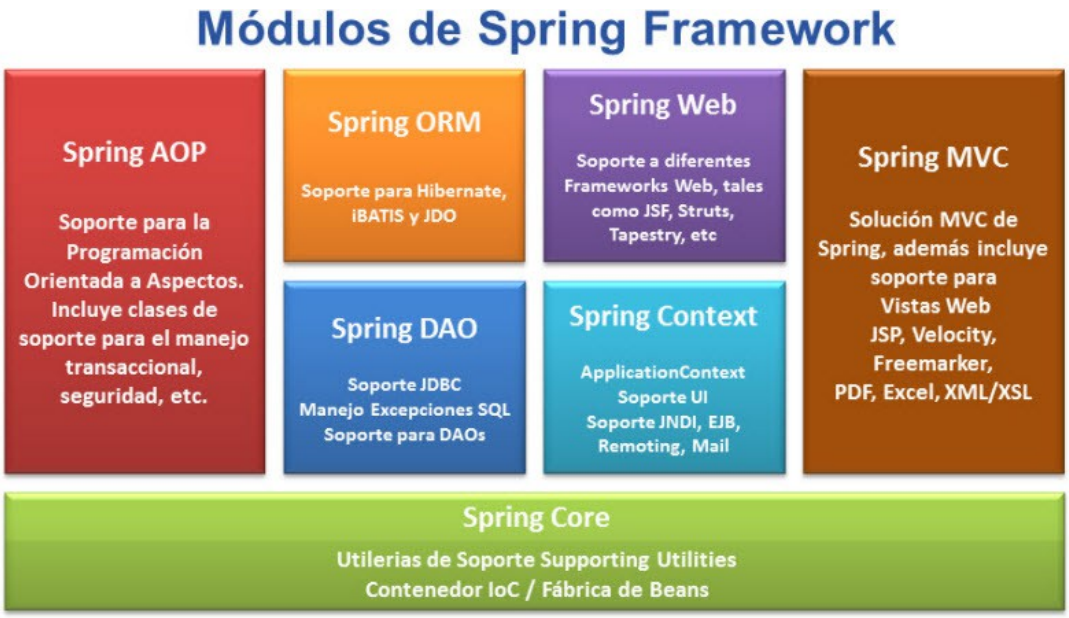

Figura 11.18 Módulos de Spring framework

11.5.4 MySQL

Figura 11.19 Logotipo de MySQL

MySQL es un sistema de gestión de bases de datos relacional fundado por *David Axmark*, *Allan Larsson* y *Michael Widenius*, que hemos utilizado para almacenar los datos que registra el Servicio Web. Fundamentalmente está desarrollado en *ANSI C*, entre las características mas destacadas de *MySQL* es que es multiusuario, multihilo y está ampliamente utilizada con aproximadamente seis millones de instalaciones.

Oracle es la empresa que desarrolla *MySQL* y dispone de dos tipos de licencias, por un lado se ofrece como *GNU GPL* y por otra, en caso de que la aplicación sea software privativo, se debe comprar una licencia específica para poder usar *MySQL* es usado o ha sido usado por grandes sitios Web como *Wikipedia*, *Google*, *Facebook*, *Twitter*, *Flickr* y *Youtube*.

11.5.5 HTML5

Figura 11.20 Logotipo de HTML5

HTML5 (Hipertexto Markup Language, versión 5) se ha utilizado para el desarrollo del cliente Web, es la quinta revisión del lenguaje HTML, el desarrollo de este lenguaje de marcado está regulado por el Consorcio *W3C*. *HTML5* especifica dos variaciones de la sintaxis para *HTML*, el *HTML* y el *XHTML5* que será servida como *XML*. La versión definitiva de esta revisión fue publicada en Octubre del 2014. Al ser un estándar *HTML* busca ser un lenguaje que permita que cualquier página Web escrita en una determinada versión pueda ser interpretada de la misma forma por cualquier navegador.

Dado que *HTML5* no es reconocido por los navegadores antiguos, se recomienda a los usuarios que sea actualicen a las nuevas versiones de los navegadores para poder disfrutar de todas las posibilidades que tiene *HTML5*.

El lenguaje *HTML* basa su filosofía de desarrollo en la referenciación. Para añadir un elemento externo a la página, este no se incrusta directamente en el código de la página. Sino que se hace una referencia al elemento mediante texto. De esta manera la página Web contiene solo texto mientras que la interpretación recae sobre el navegador Web, que se encarga de unir todos los elementos y visualizar la pagina final.

11.5.6 CSS3

Figura 11.21 Logotipo de CSS3

Las *CSS* (*Cascading Style Sheets*, o también llamadas hojas de estilo en cascada) se ha utilizado para el desarrollo del cliente Web, es un lenguaje usado para definir la presentación de un documento estructurado escrito en *HTML*, *XHTML* o en *XML*. El *W3C* (*World Wide Web Consortium*) es el encargado de definir la especificación de las hojas de estilo, que sirven de estándar para los navegadores (agentes de usuario), actualmente la última versión definida es *CSS3.*

Fueron pensadas para separar el las estructura y el contenido de la presentación. La información de los estilos puede incluirse en un fichero con extensión *CSS* externo al documento *HTML*, o en la propia cabecera del documento *HTML*, o en cada etiqueta mediante el atributo style.

11.5.7 JavaScript

Figura 11.22 Logotipo de Python

JavaScript es un lenguaje de programación interpretado, se ha utilizado para el desarrollo del cliente Web, es un dialecto del estándar del *W3C ECMAScript*, se define como orientado a objetos, imperativo, basado en prototipos, débilmente tipado y dinámico.

Se utiliza fundamentalmente en su forma del lado del cliente, implementado como parte de los navegadores Web, permitiendo mejoras en la interfaz de usuario y paginas Web dinámicas, aunque existe una forma de *JavaScript* del lado del servidor.

JavaScript se diseño con una sintaxis parecida a C, aunque adopta nombres y convenciones del lenguaje de programación *Java*, sin embargo *Java* y *JavaScript* no están relacionados, tiene semánticas y propósitos diferentes. Todos los navegadores modernos interpretan el código *JavaScript* integrado en las paginas web. Para interactuar con una pagina web se provee a *JavaScript* de una implementación del *Document Object Model* (DOM).

Figura 11.23 Logotipo de Google Maps

Uno de los aspectos importantes que utilizamos en el desarrollo del *Cliente Web*, es la *APi JavaScript V3* de *Google Maps*. *Google Maps* fué anunciado en *Google Blog* en febrero del 2005, en sus orígenes solo soportaba a usuarios de *Internet Explorer* y *Mozilla Firefox* pero el soporta para *Opera* y *Safari* se agregó a finales del mismo mes del lanzamiento.

En Junio del 2005, *Google Maps* lanzó su *API*, haciendo oficialmente modificable casi cualquier aspecto de la interfaz original, con la contraseña de desarrollador, la *API* es libre de uso para cualquier sitio Web.

11.5.8 Bootstrap

Figura 11.24 Logotipo de Bootstrap

Twitter Bootstrap es un Framework de software libre diseñado para el diseño de sitios y aplicaciones Web. Esta basado en *HTML* y *CSS*, así como en extensiones *JavaScript* (opcionales). *Bootstrap* contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos. En la actualidad *Bootstrap* es el proyecto más popular *GitHub* y es usado por muchas organizaciones, entre ellas la *NASA* (*National Aeronautics and Space Administration*) y la *MSNBC*

11.6 Herramientas y Programas Usados para el Desarrollo

11.6.1 Adafruit webIDE

Para desarrollar el código Python que se ejecuta en la *Raspberry Pi*, se ha utilizado el entorno Adafruit webIDE, que aunque aun esta en fase beta, es lo suficientemente estable como para desarrollar este proyecto.

Figura 11.25 Logotipo de Adafruit webIDE

El entorno Adafruit webIDE es la manera más fácil de ejecutar código en una *Raspberry Pi* (y también en la placa *BeagleBone*), simplemente hay que conectar la *Raspberry Pi* a la red local, y se accede al editor de código a través del navegador Web de cualquier ordenador de la red local, previa identificación en el sistema *(Cooper, T., 2014),* en este caso, los equipos utilizados para desarrollar software contaban con los navegadores *Firefox* y *Chrome*.

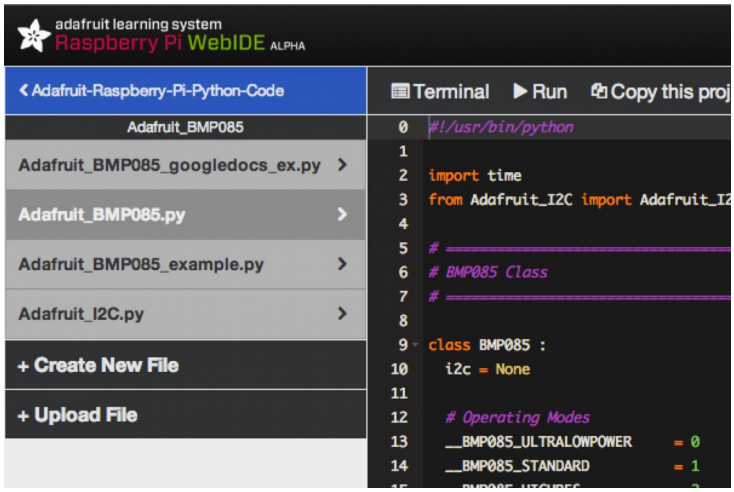

Figura 11.26 Captura de Adafruit webIDE

Además de poder ejecutar código *Python*, también admite otros lenguajes de programación como por ejemplo *JavaScript* o *Ruby*, pero en este proyecto solo se ha utilizado *Python* *Adafruit webIDE*, dispone de un terminal, con el que se pueden mandar fácilmente comandos a la *Raspberry pi* desde el navegador. Para terminar, una de las funciones más importantes, es que el código es versionado en un repositorio *GIT* local y se sincroniza con

un repositorio remoto, en la plataforma *Bitbucket*, a la que se puede acceder en cualquier momento y desde cualquier sitio.

Se ha elegido este software por ser la manera más sencilla de desarrollar software en una *Raspberry pi* y además de manera transparente gestiona el mismo el control de versiones.

11.6.2 Repositorio GIT

El Sistema de control de versiones utilizado para este proyecto es *GIT*, *GIT* fue diseñado por *Linus Torvalds*, fue diseñado pensando en la eficiencia y confiabilidad del mantenimiento de versiones, en aplicaciones que tienen un gran numero de ficheros. Es usado por ejemplo, por el grupo de programación del *kernel Linux*.

Figura 11.27 Logotipo de GIT

Inicialmente se pensó como un motor de bajo nivel sobre el cual, se pudiese escribir una interfaz de usuario o *frontend*, sin embargo, *GIT* se ha convertido en un sistema de control de versiones con funcionalidad plena.

El mantenimiento del software *GIT*, actualmente esta supervisado por *Junio Hamaro*, quien recibe contribuciones al código de cerca de 280 programadores.

El sistema de versiones *GIT*, cuenta con las siguientes características relevantes:

- Fuerte apoyo al desarrollo no lineal, y como consecuencia, rapidez en la gestión de ramas y mezclado de diferentes versiones.
- Gestión Distribuida.
- Los almacenes de información pueden publicarse por *HTTP, FTP, rsync* o mediante protocolo nativo, ya sea *TCP/IP* simple o a través del cifrado *SSH*.
- *GIT* puede emular servidores CVS.
- Los repositorios *Subversion y svk* pueden trabajar directamente con *git-svn*.
- Gestión eficiente de proyectos grandes.

- Todas las versiones previas a un cambio determinado, implican la notificaron de un cambio posterior en cualquiera de ellas a ese cambio (lo que se denomina autenticación criptográfica).
- Resulta algo más caro trabajar con ficheros concretos frente a proyectos.
- Los renombrados se trabajan basándose en similitudes entre ficheros.
- Realmacenamiento periódico de paquetes, lo que hace que sea relativamente eficiente en la escritura de cambios y relativamente ineficiente para la lectura de datos si no se realiza el reempaquetado cada cierto tiempo.

Se ha elegido este tipo de control de versiones, ya que por defecto es el que viene configurado en el *Adafruit webIDE* y por ser uno de los repositorios mas populares en la actualidad.

11.6.3 Bitbucket.org

Bitbucket es un servicio de alojamiento basado en Web, propiedad de *Atlassian Software*, escrito en *Python*, para proyectos que utilizan un sistema de control de versiones *GIT* o *Mercurial*, esta plataforma ofrece planes comerciales y gratuitos, pero para este caso concreto de este proyecto utilizaremos un plan gratuito (*Atlassian, 2014*).

Figura 11.28 Logotipo de Bitbucket

Los planes gratuitos permiten un número ilimitado de repositorios privados que no se muestran en las páginas de perfil y permite hasta cinco usuarios.

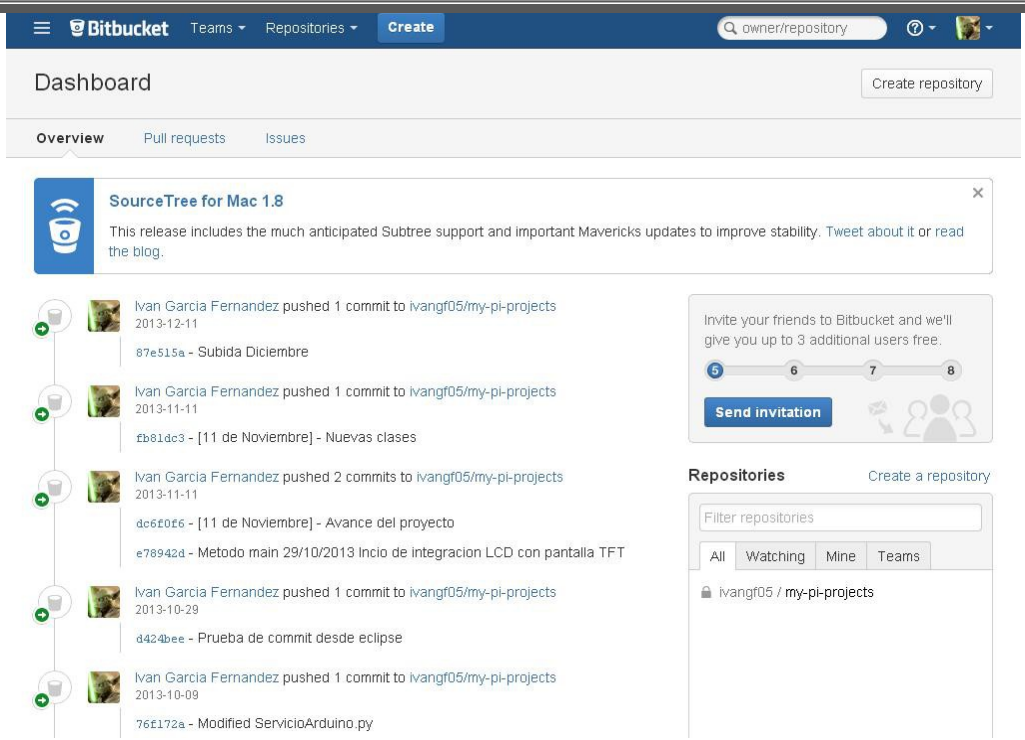

Figura 11.29 Captura Bitbucket.org

Además permite crear una Wiki para el proyecto (Para este proyecto no la he usado, pero resulta de mucha utilidad en proyectos colaborativo) y un Tracker para gestionar las incidencias y proposiciones de mejora, esta herramienta, resulta muy interesante para llevar un control de las gestiones de cambio y de los defectos software que se van detectando durante el desarrollo y la fase de *Testing*, por supuesto su utilidad aumenta cuando se trabaja en equipo. En la imagen que tenemos a continuación se puede ver, el uso de esta funcionalidad.

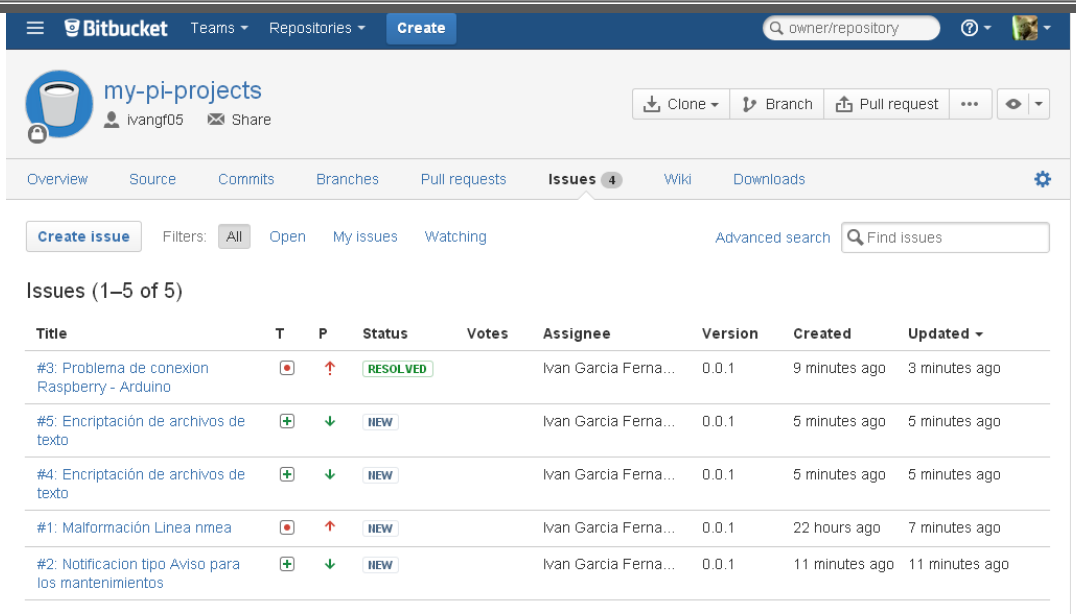

Figura 11.30 Detalle del tracker de Bitbucket

Por último, *Adafruit webIDE* facilita la integración con este sistema y por eso se ha elegido este servicio de alojamiento.

11.6.4 Eclipse IDE

Dado que se ha decidido que el lenguaje de programación utilizado para desarrollar este proyecto, será Java, se ha utilizado el entorno de desarrollo integrado Eclipse, en su versión 4.2.2 Classic Juno.

Figura 11.31 Logotipo de eclipse

Eclipse fue desarrollado originariamente por *IBM*, y en la actualidad es desarrollado por la *Fundación Eclipse*[18], una organización independiente, sin ánimo de lucro, que fomenta una comunidad de código abierto.

La definición que da el proyecto eclipse acerca de su software es "*una especie de herramienta universal, un IDE abierto y extensible para todo y nada en particular*".

[18] Fundación eclipse – http://www.eclipse.org/org/

Originariamente *Eclipse* fue liberado bajo la *Common Public License*, pero después fue licenciado bajo la *Eclipse Public License*, la organización *Free Software Foundation*[19], ha dicho que ambas son licencias de Software libre si bien es cierto que son incompatibles con la conocida Licencia publica general de *GNU* (*GNU GPL*).

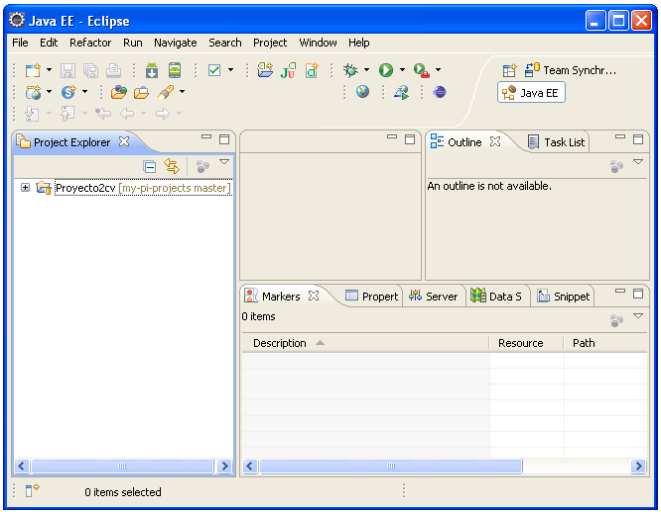

Figura 11.32 Eclipse utilizado en el proyecto

11.6.5 Maven

Maven es una herramienta de software para la gestión y construcción de proyectos en lenguaje Java creada por Jason van Zyl, de Sonatype, en 2002. Tiene un modelo de configuración de construcción simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la *Apache Software Foundation*.

Figura 11.33 Logotipo de maven

Maven utiliza un *Project Object Model* (*POM*) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

[19] Free Software foundation – http://www.fsf.org/es/

Una característica clave de *Maven* es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar *plugins* de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos *Open Source* en *Java*, de *Apache* y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio *Maven* 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en *Java*, pero su adopción ha sido muy lenta. *Maven* provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

11.7 Código Fuente Servicios Web

11.7.1 Paquete "*tfm*"

11.7.1.1 Fichero "*Application.java*"

```
package tfm;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 * Clase que arranca la aplicacion
 * @author igarciaf
 */

@Configuration
@ComponentScan
@EnableAutoConfiguration
public class Application {
    public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
    }
}
```

11.7.2 Paquete "*tfm.facade*"

11.7.2.1 Fichero "*PointOfInterestController.java*"

```
package tfm.facade;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import tfm.dataLayer.PointOfInterest;
import tfm.logicLayer.IPointsBusinessLogic;
import tfm.util.Constantes;
import tfm.util.FactoryPoints;

@RestController
@Configuration
public class PointOfInterestController {
  @Autowired
  private IPointsBusinessLogic iPointsBusinessLogic;

   //http://localhost:8080/getPointsOfInterest
  @RequestMapping("/getPointsOfInterest")
   public @ResponseBody List<PointOfInterest>
getPointsOfInterest(@RequestParam(value="metros",
          required=false) String metros, @RequestParam(value="latidude", required=false)
String latitude,
```

```
                @RequestParam(value="longitude", required=false) String longitude) {
                if(this.validateMetros(metros)==false || this.validateDouble(latitude) == false
||                                                      this.validateDouble(longitude) ==
false){
                        return iPointsBusinessLogic.getListAllPoints();
                }
                return

iPointsBusinessLogic.getListAllPoints(Double.parseDouble(latitude),Double.parseDouble(l
ongitude),      Integer.parseInt(metros));
   }

//http://localhost:8080/addPointOfInterest?
// latidude=latidude&codeOrigin=codeOrigin&codeAlert=codeAlert&longitude=longitude&
   @RequestMapping("/addPointOfInterest")
   public @ResponseBody void addPointOfInterest(
                @RequestParam(value="latidude", required=true) String latitude,
                @RequestParam(value="longitude", required=true) String longitude,
                @RequestParam(value="codeOrigin", required=true) String codeOrigin,
                @RequestParam(value="codeAlert", required=true) String codeAlert) {
        PointOfInterest point = FactoryPoints.createPoint (latitude, longitude, codeOrigin,
codeAlert);
        if(point != null){
                iPointsBusinessLogic.savePoint(point);
        }
   }

  private boolean validateMetros(String metros) {
        if(metros == null || metros.trim().compareToIgnoreCase("") == 0){
                return false;
        }else{
                int metrosInt = 0;
                try{
                        metrosInt = Integer.parseInt(metros);
```

```
                }catch(Exception e){
                        return false;
                }
                if(metrosInt < 0){
                        return false;
                }
        }
                return true;
        }

    private boolean validateDouble(String valor) {
        if(valor == null || valor.trim().compareToIgnoreCase("") == 0){
                return false;
        }else{
                double valueDouble = 0;
                try{
                        valueDouble = Double.parseDouble(valor);
                }catch(Exception e){
                        return false;
                }
                if(valueDouble < 0){
                        return false;
                }
        }
                return true;
        }

    public void setIPointsBusinessLogic(IPointsBusinessLogic iPointsBusinessLogic) {
                this.iPointsBusinessLogic = iPointsBusinessLogic;
        }
}
```

11.7.2.2 Fichero "*SimpleCORSFilter.java*"

```
package tfm.facade;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;

@Component
public class SimpleCORSFilter implements Filter {

	public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
			throws IOException, ServletException {
		HttpServletResponse response = (HttpServletResponse) res;
		response.setHeader("Access-Control-Allow-Origin", "*");
		response.setHeader("Access-Control-Allow-Methods", "POST, GET,
OPTIONS, DELETE");
		response.setHeader("Access-Control-Max-Age", "3600");
		response.setHeader("Access-Control-Allow-Headers", "x-requested-with");
		chain.doFilter(req, res);
	}
	public void init(FilterConfig filterConfig) {}
	public void destroy() {}
}
```

11.7.3 Paquete "*tfm.logiLayer*"

11.7.3.1 Fichero "*IPointsBusinessLogic.java*"

```
package tfm.logicLayer;

import java.util.List;
import tfm.dataLayer.PointOfInterest;

/**
 * Interfaz que expone las operaciones de la capa de negocio
 * @author igarcia
 */
public interface IPointsBusinessLogic {
	public  void savePoint(PointOfInterest point);
	public  List<PointOfInterest> getListAllPoints();
	public List<PointOfInterest> getListAllPoints(double lat, double lng, int metros);
}
```

11.7.3.2 Fichero "*IPointsBusinessLogicImpl.java*"

```
package tfm.logicLayer;

import tfm.dataLayer.IPointsRepository;
import tfm.dataLayer.PointOfInterest;
import tfm.util.Util;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
```

```
 * Clase que contiene la implementacion de la interfaz IPointsBusinessLogic
 * @author igarciaf
 */
@Component
public class IPointsBusinessLogicImpl implements IPointsBusinessLogic {

	@Autowired
	private IPointsRepository repositorio;

	@Override
	public void savePoint(PointOfInterest point) {
		repositorio.save(point);
	}

	@Override
	public List<PointOfInterest> getListAllPoints() {
		List<PointOfInterest> myList = (List<PointOfInterest>)
Util.makeCollection(repositorio.findAll());
		List<PointOfInterest> myFilteredList = new ArrayList<PointOfInterest>();
		for(PointOfInterest punto: myList){
		   Calendar calendar = Calendar.getInstance();
		   calendar.add(Calendar.DAY_OF_YEAR, -2);
		   Date fechaMenos2 = calendar.getTime();
			if( punto.getFecha().getTime() > fechaMenos2.getTime()){
				myFilteredList.add(punto);
			}
		}
		return myFilteredList ;
	}

	@Override
	public List<PointOfInterest> getListAllPoints(double lat, double lng, int metros) {
		List<PointOfInterest> myList = this.getListAllPoints();
		List<PointOfInterest> myFilteredList = new ArrayList<PointOfInterest>();
```

```
                for(PointOfInterest punto: myList){
                        int distancia = (int) Util.gps2m(punto.getLatitude(),
punto.getLongitude(), lat , lng);
                        if(distancia < metros){
                                myFilteredList.add(punto);
                        }
                }
                return myFilteredList ;
        }

        public IPointsRepository getRepositorio() {
                return repositorio;
        }

        public void setRepositorio(IPointsRepository repositorio) {
                this.repositorio = repositorio;
        }
}
```

11.7.4 Paquete "*tfm.dataLayer*"

11.7.4.1 Fichero "*DatabaseConfig.java*"

```
package tfm.dataLayer;

import java.util.Properties;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import
org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
```

```
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.annotation.EnableTransactionManagement;

/**
 * Clase que contiene toda la configuracion de la base de datos
 * @author igarciaf
 */
@Configuration
@EnableTransactionManagement
public class DatabaseConfig {

  @Autowired
  private Environment _env;

  @Autowired
  private DataSource _dataSource;

  @Autowired
  private LocalContainerEntityManagerFactoryBean _entityManagerFactory;

  /**
   * Method dataSource
   * Defines the database connection informations in a DataSource object.
   */
  @Bean
  public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(_env.getProperty("db.driver"));
    dataSource.setUrl(_env.getProperty("db.url"));
    dataSource.setUsername(_env.getProperty("db.username"));
    dataSource.setPassword(_env.getProperty("db.password"));
    return dataSource;
```

```
}

/**
 * Method entityManagerFactory
 * Declare the JPA entity manager factory.
 */
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
  LocalContainerEntityManagerFactoryBean entityManagerFactory =
      new LocalContainerEntityManagerFactoryBean();
      entityManagerFactory.setDataSource(_dataSource);
  entityManagerFactory.setPackagesToScan(
      _env.getProperty("entitymanager.packagesToScan"));
      // Vendor adapter
  HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
  entityManagerFactory.setJpaVendorAdapter(vendorAdapter);
      // Hibernate properties
  Properties additionalProperties = new Properties();
  additionalProperties.put(
      "hibernate.dialect",
      _env.getProperty("hibernate.dialect"));
  additionalProperties.put(
      "hibernate.show_sql",
      _env.getProperty("hibernate.show_sql"));
  additionalProperties.put(
      "hibernate.hbm2ddl.auto",
      _env.getProperty("hibernate.hbm2ddl.auto"));
  entityManagerFactory.setJpaProperties(additionalProperties);
      return entityManagerFactory;
}

/**
 * Method transactionManager
 * Declare the transaction manager.
```

```
   */
  @Bean
  public JpaTransactionManager transactionManager() {
    JpaTransactionManager transactionManager =
        new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(
        _entityManagerFactory.getObject());
    return transactionManager;
  }

  /**
   * Method persistenceExceptionTranslationPostProcessor
   * PersistenceExceptionTranslationPostProcessor is a bean post processor
   * which adds an advisor to any bean that's annotated with \@Repository so
   * that any platform-specific exceptions are caught and then rethrown as one
   * Spring's unchecked data access exceptions (i.e. a subclass of
   * DataAccessException).
   */
  @Bean
  public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
    return new PersistenceExceptionTranslationPostProcessor();
  }
}
```

11.7.4.2 Fichero "*IpointsRepository.java*"

```
package tfm.dataLayer;

import org.springframework.data.repository.CrudRepository;

/**
 * Interfaz que expone las operaciones que se realizan sobre la entidad PointOfInterest
 * @author igarciaf
 */
```

```
public interface IPointsRepository extends CrudRepository<PointOfInterest, Long>{
}
```

11.7.4.3 Fichero "*PointOfInterest.java*"

```
package tfm.dataLayer;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 * Entidad PointOfInterest
 * @author igarciaf
 */
@Entity
public class PointOfInterest {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id = 0L;
    private double latitude = 0.0d;
    private double longitude = 0.0d;
    private String codeOrigin = "0";
    private String codeAlert = "0";
    private Date fecha;

        public Long getId() {
                return id;
        }
        public void setId(Long id) {
```

```
		this.id = id;
	}
	public double getLatitude() {
		return latitude;
	}
	public void setLatitude(double latitude) {
		this.latitude = latitude;
	}
	public double getLongitude() {
		return longitude;
	}
	public void setLongitude(double longitude) {
		this.longitude = longitude;
	}
	public String getCodeOrigin() {
		return codeOrigin;
	}
	public void setCodeOrigin(String codeOrigin) {
		this.codeOrigin = codeOrigin;
	}
	public String getCodeAlert() {
		return codeAlert;
	}
	public void setCodeAlert(String codeAlert) {
		this.codeAlert = codeAlert;
	}
	public Date getFecha() {
		return fecha;
	}
	public void setFecha(Date fecha) {
		this.fecha = fecha;
	}
}
```

11.7.5 Paquete "*tfm.dataLayer*"

11.7.5.1 Fichero "Constantes.*java*"

```
package tfm.util;

/**
 * Clase que contiene todas las constantes de la aplicacion
 * @author igarciaf
 */
public class Constantes {
	public static int CODE_ACCIDENTE = 0;
	public static int CODE_RETENCION = 1;
	public static int CODE_MANTENIMIENTO = 2;
	public static int CODE_OTRO = 3;
	public static String CLIENTE_ANDROID = "Android";
	public static String CLIENTE_WEB = "Web";
	public static String CLIENTE_ON_BOARD = "OnBoard";
}
```

11.7.5.2 Fichero "*FactoryPoints.java*"

```
package tfm.util;

import java.util.Date;
import tfm.dataLayer.PointOfInterest;

public class FactoryPoints {

	public static PointOfInterest createPoint(String latitude,
			String longitude, String codeOrigin, String codeAlert) {
		PointOfInterest point = new PointOfInterest();
		try{
			point.setFecha(new Date());
```

```
                        point.setCodeAlert(codeAlert);
                        point.setCodeOrigin(codeOrigin);
                        point.setLatitude(Double.parseDouble(latitude));
                        point.setLongitude(Double.parseDouble(longitude));
                }catch(Exception e){
                        return null;
                }
                return point;
        }
}
```

11.7.5.3 Fichero "*Util.java*"

```
package tfm.util;

import java.util.ArrayList;
import java.util.List;

/**
 * Clase de Utilidades que contiene metodos que pueden ser de utilidad a lo largo de la aplicacion
 * @author ivan
 */
public class Util {

        public static <PointOfInterest> List<PointOfInterest> makeCollection(Iterable<PointOfInterest> iter) {
            List<PointOfInterest> list = new ArrayList<PointOfInterest>();
            for (PointOfInterest item : iter) {
                list.add(item);
            }
            return list;
        }
```

```
        public static double gps2m(double lat_a, double lng_a, double lat_b, double lng_b)
{
                double pk = (double) (180/3.14169);
            double a1 = lat_a / pk;
            double a2 = lng_a / pk;
            double b1 = lat_b / pk;
            double b2 = lng_b / pk;
            double t1 = Math.cos(a1)*Math.cos(a2)*Math.cos(b1)*Math.cos(b2);
            double t2 = Math.cos(a1)*Math.sin(a2)*Math.cos(b1)*Math.sin(b2);
            double t3 = Math.sin(a1)*Math.sin(b1);
            double tt = Math.acos(t1 + t2 + t3);
            return 6366000*tt;
        }
}
```

11.7.6 Fichero "*src/main/resources/application.properties*"

```
# Database
db.driver: com.mysql.jdbc.Driver
db.url: jdbc:mysql://localhost:3306/tfm
db.username: root
db.password: root

# Hibernate
hibernate.dialect: org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql: true
hibernate.hbm2ddl.auto: update
entitymanager.packagesToScan: tfm
```

11.7.7 Fichero "*application-context.xml*"

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">
<context:component-scan base-package="tfm"/>
</beans>
```

11.7.8 Fichero "*pom.xml*"

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework</groupId>
    <artifactId>2cv-rest-service</artifactId>
    <version>0.1.0</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.1.7.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
```

```
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.data</groupId>
			<artifactId>spring-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-orm</artifactId>
		</dependency>
		<dependency>
			<groupId>org.hibernate.javax.persistence</groupId>
			<artifactId>hibernate-jpa-2.0-api</artifactId>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
		</dependency>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
		</dependency>
		<dependency>
```

```
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-server</artifactId>
                <version>1.8</version>
            </dependency>
            <dependency>
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-json</artifactId>
                <version>1.8</version>
            </dependency>
        </dependencies>
        <properties>
            <start-class>tfm.Application</start-class>
        </properties>
        <build>
            <resources>
                <resource>
                    <directory>src/main/resources</directory>
                    <includes>
                        <include>application.properties</include>
                        <include>application-context.xml</include>
                    </includes>
                </resource>
            </resources>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
        <repositories>
            <repository>
                <id>spring-releases</id>
                <url>http://repo.spring.io/libs-release</url>
```

```
                </repository>
        </repositories>
        <pluginRepositories>
                <pluginRepository>
                        <id>spring-releases</id>
                        <url>http://repo.spring.io/libs-release</url>
                </pluginRepository>
        </pluginRepositories>
</project>
```

11.8 Código fuente cliente web

11.8.1 Fichero "*index.html*"

```
<!DOCTYPE html>
<html>
        <head>
                <title>TFM 2014 - Ivan Garcia Fernandez - Inicio</title>
                <meta charset="utf-8">
                < l i n k                                                r e l = " s t y l e s h e e t "
href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap.min.css">
                < l i n k                                                r e l = " s t y l e s h e e t "
href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap-theme.min.css">
                <link rel="stylesheet" type="text/css" href="styles/index.css">
                <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
                <script
src="https://cdn.jsdelivr.net/bootstrap/3.3.0/js/bootstrap.min.js"></script>
                <script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
                <script src="scripts/index.js"></script>
        </head>
        <body>
                <div id="map-canvas"></div>
                <div id="botonera" align="center" >
```

```
                    <a href="alta.html" class="btn btn-lg btn-default" >Alta
Incidencia</a>
                </div>
        </body>
</html>
```

11.8.2 Fichero "*styles/index.css*"

```
html, body, #map-canvas {
        height: 100%;
        margin: 0px;
        padding: 0px
}
#botonera{
        position:absolute;
        bottom:10%;
        margin-left: auto;
   margin-right: auto;
        width: 100%;
}
```

11.8.3 Fichero "*scripts/index.js*"

```
Sad
var POSICION = "0033FF";
var ACCIDENTE = "FF0000";
var ATASCO = "FF9900";
var OTRO = "CC33FF";
var MANTENIMIENTO ="FFFF00"
var map;

// Inicializacion del mapa
function initialize() {
        var mapOptions = {
        zoom: 16
```

```
        };
        map = new google.maps.Map(document.getElementById('map-canvas'),
mapOptions);
        //Capa de Trafico
        var trafficLayer = new google.maps.TrafficLayer();
        trafficLayer.setMap(map);
        // Try HTML5 geolocation
        if(navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(function(position) {
                        var pos = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);

                        $.ajax({
                                url: "http://localhost:8080/getPointsOfInterest"
                                //url:
"http://54.171.43.166:8080/getPointsOfInterestURL&jsoncallback=?";
                        }).then(function(data) {
                                for (i = 0; i < data.length; i++) {
                                        punto = data[i];
                                        p o s M                 =                 n e w
google.maps.LatLng(punto.latitude,punto.longitude);
                                        var pinColor = getColorMarcador(punto.codeAlert);
                                        var txtAlert = getTxtMarcador(punto.codeAlert);
                                        var pinImage = crearPinImage(pinColor);
                                        var marker = new google.maps.Marker({
                                                position: posM,
                                                map: map,
                                                icon: pinImage,
                                                title: txtAlert
                                        });
                                }
                        });
                        var pinColorPosicion = POSICION;
                        var pinImagePosicion = crearPinImage(pinColorPosicion);
```

```
                        marker = new google.maps.Marker({
                                position: pos,
                                map: map,
                                icon: pinImage,
                                title: 'Posicion Actual'
                        });
                        // Establecemos el centro del mapa
                        map.setCenter(pos);
                }, function() {
                        handleNoGeolocation(true);
                });
        } else {
                // Browser doesn't support Geolocation
                handleNoGeolocation(false);
        }
}
// Cargar el mapa
google.maps.event.addDomListener(window, 'load', initialize);

// Funcion que devuelve el color en función del codigo
function getColorMarcador(code){
        switch(code){
           case "0":
                        return ACCIDENTE;
                        break;
           case "1":
                        return ATASCO;
                        break;
                case "2":
                        return MANTENIMIENTO;
                        break;
           case "3":
                        return OTRO;
                        break;
```

```
                default:
                        return POSICION;
        }
}

// funcion que devuelve el texto en funcion del codigo
function getTxtMarcador(code){
        switch(code){
           case "0":
                        return "Accidente";
                        break;
           case "1":
                        return "Atasco";
                        break;
           case "2":
                        return "Mantenimiento";
                        break;
           case "3":
                        return "Otro";
                        break;
                default:
                        return "Posicion Actual";
        }
}

// Funcion que maneja el error si el navegador no es compatible
function handleNoGeolocation(errorFlag) {
        if (errorFlag) {
                var content = 'Error: The Geolocation service failed.';
        } else {
                var content = 'Error: Your browser doesn\'t support geolocation.';
        }
        var options = {
                map: map,
```

```
                position: new google.maps.LatLng(60, 105),
                content: content
        };
        var infowindow = new google.maps.InfoWindow(options);
        map.setCenter(options.position);
}
function crearPinImage (pinColor){
    return pinImage = new google.maps.MarkerImage("http://chart.apis.google.com/chart?
chst=d_map_pin_letter&chld=%E2%80%A2|" + pinColor,
      new google.maps.Size(21, 34),
      new google.maps.Point(0,0),
      new google.maps.Point(10, 34));
}
```

11.8.4 Fichero "*alta.html*"

```
<!DOCTYPE html>
<html>
        <head>
                <title>TFM 2014 - Ivan Garcia Fernandez - Alta</title>
                <meta charset="utf-8">
                < l i n k                                        r e l = " s t y l e s h e e t "
href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap.min.css">
                < l i n k                                        r e l = " s t y l e s h e e t "
href="https://cdn.jsdelivr.net/bootstrap/3.3.0/css/bootstrap-theme.min.css">
                <link rel="stylesheet" type="text/css" href="styles/alta.css">
                <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
                <script
src="https://cdn.jsdelivr.net/bootstrap/3.3.0/js/bootstrap.min.js"></script>
                <script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
                <script src="scripts/alta.js"></script>


        </head>
```

```
        <body>
                <div id="map-canvas"></div>
                <div id="botonera" align="center" >
<a id="btnAlta" href="index.html" onclick="onClickVolver();" class="btn btn-lg btn-default">Volver</a>
                </div>
        <div id="botonera2" align="center">
<a id="btnAccidente" href="index.html" onclick="onClickAccidente();" class="btn btn-lg btn-danger">Accidente</a>
<a id="btnAtasco" href="index.html" onclick="onClickAtasco();" class="btn btn-lg btn-warning">Atasco</a>
<a id="btnMantenimiento" href="index.html" onclick="onClickMantenimiento();" class="btn btn-lg btn-customAmarillo">Mantenimiento</a>
<a id="btnOtro" href="index.html" onclick="onClickOtro();" class="btn btn-lg btn-customMorado">Otro</a>
                </div>
        </body>
</html>
```

11.8.5 Fichero "*styles/alta.css*"

```
.btn-customAmarillo{ background-color: hsl(60, 100%, 30%) !important; background-repeat: repeat-x; filter: progid:DXImageTransform.Microsoft.gradient(startColorstr="#feff00", endColorstr="#989900"); background-image: -khtml-gradient(linear, left top, left bottom, from(#feff00), to(#989900)); background-image: -moz-linear-gradient(top, #feff00, #989900); background-image: -ms-linear-gradient(top, #feff00, #989900); background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0%, #feff00), color-stop(100%, #989900)); background-image: -webkit-linear-gradient(top, #feff00, #989900); background-image: -o-linear-gradient(top, #feff00, #989900); background-image: linear-gradient(#feff00, #989900); border-color: #989900 #989900 hsl(60, 100%, 25%); color: #fff !important; text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.33); -webkit-font-smoothing: antialiased; }
.btn-customMorado { background-color: hsl(279, 100%, 30%) !important; background-repeat: repeat-x; filter:
```

```
progid:DXImageTransform.Microsoft.gradient(startColorstr="#a500ff",
endColorstr="#630099"); background-image: -khtml-gradient(linear, left top, left bottom, from(#a500ff), to(#630099)); background-image: -moz-linear-gradient(top, #a500ff, #630099); background-image: -ms-linear-gradient(top, #a500ff, #630099); background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0%, #a500ff), color-stop(100%, #630099)); background-image: -webkit-linear-gradient(top, #a500ff, #630099); background-image: -o-linear-gradient(top, #a500ff, #630099); background-image: linear-gradient(#a500ff, #630099); border-color: #630099 #630099 hsl(279, 100%, 25%); color: #fff !important; text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.33); -webkit-font-smoothing: antialiased; }
html, body, #map-canvas {
        height: 100%;
        margin: 0px;
        padding: 0px
}
#botonera{
        position:absolute;
        bottom:8%;
        margin-left: auto;
   margin-right: auto;
        width: 100%;
}
#botonera2{
        position:absolute;
        bottom:15%;
        margin-left: auto;
   margin-right: auto;
        width: 100%;
}
.btn{
    width:10%;
      height:1%;
}
```

11.8.6 Fichero “scripts/alta.js”

```
var COLOR_POSICION = "0033FF";
var servidor = "http://54.171.43.166:8080"
var map;
var pos;
var marker;

function initialize() {
	//Opciones de configuracion del Mapa
	var mapOptions = {
		zoom: 18
	};
	// Creamos el Mapa
	map = new google.maps.Map(document.getElementById('map-canvas'),
mapOptions);
	// Traffic Layer
	var trafficLayer = new google.maps.TrafficLayer();
	trafficLayer.setMap(map);
	// Try HTML5 geolocation
	if(navigator.geolocation) {
		navigator.geolocation.getCurrentPosition(function(position) {
			pos = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
			// Creamos el marcador de la posicion
			var pinColor = COLOR_POSICION;
			var pinImage = crearPinImage(pinColor);
			marker = new google.maps.Marker({
				position: pos,
				map: map,
				icon: pinImage,
				title: 'Posicion Actual'});
			// Establecemos el centro del mapa
			map.setCenter(pos);
```

```
                }, function() {
                handleNoGeolocation(true);
                });
        } else {
                // Browser doesn't support Geolocation
                handleNoGeolocation(false);
        }
        // Evento Onclick
        google.maps.event.addListener(map, 'click', function(evento) {
                // borramos el marcador
                marker.setMap(null);
                //Obtengo las coordenadas separadas
                pos = new google.maps.LatLng(evento.latLng.lat(), evento.latLng.lng());
                var pinColor = COLOR_POSICION;
                var pinImage = crearPinImage(pinColor);
                marker = new google.maps.Marker({
                        position: pos,
                        map: map,
                        icon: pinImage,
                        title: 'Posicion Actual'});
        });
}

google.maps.event.addDomListener(window, 'load', initialize);

// Funcion que gestiona el mapa de no existir la geolocalizacion
function handleNoGeolocation(errorFlag) {
        if (errorFlag) {
                var content = 'Error: The Geolocation service failed.';
        } else {
                var content = 'Error: Your browser doesn\'t support geolocation.';
        }
        var options = {
                map: map,
```

```
                position: new google.maps.LatLng(60, 105),
                content: content
        };
        var infowindow = new google.maps.InfoWindow(options);
        map.setCenter(options.position);
}

function onClickAccidente(){
        var latitude = pos.lat();
        var longitude = pos.lng();
        var codeOrigin = "Web";
        var codeAlert = "0";
        var urlTxt = servidor + "/addPointOfInterest?latidude=" + latitude+ "&codeOrigin="
+codeOrigin +"&codeAlert=" +codeAlert+ "&longitude=" +longitude;
        $.ajax({
                url: urlTxt
        }).then(function(data) {
                alert("Alerta Accidente");
        });
}

function onClickAtasco(){
        var latitude = pos.lat();
        var longitude = pos.lng();
        var codeOrigin = "Web";
        var codeAlert = "1";
        var urlTxt = servidor + "/addPointOfInterest?latidude=" + latitude+ "&codeOrigin="
+codeOrigin +"&codeAlert=" +codeAlert+ "&longitude=" +longitude;
        $.ajax({
                url: urlTxt
        }).then(function(data) {
                alert("Alerta Atasco");
        });
}
```

```
function onClickMantenimiento(){
	var latitude = pos.lat();
	var longitude = pos.lng();
	var codeOrigin = "Web";
	var codeAlert = "2";
	var urlTxt = servidor + "/addPointOfInterest?latidude=" + latitude+ "&codeOrigin="
+codeOrigin +"&codeAlert=" +codeAlert+ "&longitude=" +longitude;
	$.ajax({
		url: urlTxt
	}).then(function(data) {
		alert("Alerta Mantenimiento");
	});
}

function onClickOtro(){
	var latitude = pos.lat();
	var longitude = pos.lng();
	var codeOrigin = "Web";
	var codeAlert = "3";
	var urlTxt = servidor + "/addPointOfInterest?latidude=" + latitude+ "&codeOrigin="
+codeOrigin +"&codeAlert=" +codeAlert+ "&longitude=" +longitude;
	$.ajax({
		url: urlTxt
	}).then(function(data) {
		alert("Alerta Otro");
	});
}
function crearPinImage (pinColor){
    return pinImage = new google.maps.MarkerImage("http://chart.apis.google.com/chart?
chst=d_map_pin_letter&chld=%E2%80%A2|" + pinColor,
      new google.maps.Size(21, 34),
      new google.maps.Point(0,0),
      new google.maps.Point(10, 34));
```

```
}
```

11.9 Código fuente ampliación ordenador de a bordo

11.9.1 Fichero "*main.py*"

```
#!/usr/bin/python
# Version 0.0.1 - Mark 1 - ivangf05

from ServicioArduino import ServicioArduino
from ServicioLCD import ServicioLCD
from ServicioPantalla import ServicioPantalla
from ServicioGPS import ServicioGPS
from ServicioWebService import ServicioWebService
from ServicioNotificaciones import ServicioNotificaciones
from ServicioMantenimientos import ServicioMantenimientos
from InformacionVehiculo import InformacionVehiculo
from Util import *

import serial
import signal
import threading

#Control del bucle
continueReading = True
#Creamos el informacionVehiculo
informacionVehiculo = InformacionVehiculo()
#informacionVehiculo.satelites = 1
#creamos el ServicioGPS
gps = ServicioGPS(informacionVehiculo)
# Conexion en serie para el GPS (conexion con la raspberry)
ser = serial.Serial('/dev/ttyAMA0', baudrate = 9600 , interCharTimeout = None )
# Hilo de lectura GPS
tGPS = threading.Thread(target = gps.receiving, args = (ser, )).start()
```

```
# Creamos el Servicio Arduino
servicioArduino = ServicioArduino (informacionVehiculo)
# Hilo de lectura Arduino
tArduino = threading.Thread(target = servicioArduino.receiving, args = ( )).start()
# Hilo de web service
tWS = threading.Thread(target = servicioWS.receiving, args = ( )).start()
# Creamos el ServicioNotificaciones
servicioNotificaciones = ServicioNotificaciones (informacionVehiculo)
# Hilo Notificaciones
tNotificaciones = threading.Thread(target = servicioNotificaciones.comprobarNotificaciones, args = ()).start()
# Creamos el servicioMantenimientos
servicioMantenimientos = ServicioMantenimientos (informacionVehiculo)
# Hilo Mantenimientos
tMantenimientos = threading.Thread(target = servicioMantenimientos.comprobarMantenimientos, args = ()).start()
servicioPantalla = ServicioPantalla (informacionVehiculo)
servicioLCD = ServicioLCD(informacionVehiculo)
servicioLCD.draw()
servicioPantalla.draw()

def end_read(signal,frame):
  global continue_reading
  continue_reading = False
  tGPS.stop()
  sys.exit(0)

while(continueReading):
    servicioPantalla.draw()
    signal.signal(signal.SIGINT, end_read)
    servicioLCD.comprobarPulsacion()
    servicioLCD.draw()
```

11.9.2 Fichero "*informacionVehiculo.py*"

```
#!/usr/bin/python
# Version 0.0.2 - Mark 1 - ivangf05

from InformacionNotificaciones import InformacionNotificaciones
from InformacionViaje import InformacionViaje
from InformacionMantenimientos import InformacionMantenimientos
from InformacionTemperaturas import InformacionTemperaturas
from InformacionWebServices import InformacionWebServices

class InformacionVehiculo:

    # Constructor de InformacionVehiculo
    def __init__(self):
        print "inicializando InformacionVehiculo..."
        self.__ruta = "resources/datosInformacionVehiculo.txt"
        self.informacionMantenimientos = InformacionMantenimientos()
        self.informacionViaje = InformacionViaje()
        self.informacionNotificaciones = InformacionNotificaciones()
        self.informacionTemperaturas = InformacionTemperaturas()
        self.informacionWebServices = InformacionWebServices()
        self.hora = "00:00"
        #self.fecha = "00/00/0000"
        self.fecha = "01/05/2014"
        self.direccion = "-"
        self.velocidad = 0
        self.totalDistancia = 0 # km totales recorridos
        self.longitud = 0
        self.latitud = 0
        self.altitud = 0
        self.satelites = 1
        self.angleXZ = 0
        self.angleYZ = 0
```

```
        self.subPantalla = 0
        self.COD_PANTALLA_AVISOS = -1
        self.COD_PANTALLA_PRINCIPAL = 0
        self.COD_PANTALLA_INCLINOMETRO = 1
        self.COD_PANTALLA_TEMPERATURA = 2
        self.COD_PANTALLA_VIAJE = 3
        self.COD_PANTALLA_MANTENIMIENTOS = 4
        self.COD_PANTALLA_WEBSERVICES = 5
        self.__cargarDatosFichero()

    #Metodo que carga de un fichero los datos de la informacion del vehicul
    def __cargarDatosFichero (self):
        #TODO posible encriptacion
        # Formato 0#0#0#0#F
        try:
            f = open(  self.__ruta)
            contenidoArchivo = f.read()
            f.close()
            if(len(contenidoArchivo.split("#")) == 5):
                #[0] - distancia total
                distanciaTotal = float(contenidoArchivo.split("#")[0])
                #[1] - distancia trip A
                distanciaTrip1 = float(contenidoArchivo.split("#")[1])
                #[2] - distancia trip b
                distanciaTrip2 = float(contenidoArchivo.split("#")[2])
                #[3] - ultima medicion de deposito
                deposito = float(contenidoArchivo.split("#")[3])
                #[4] - flag de reset de combustible
                flagResetCombustible = contenidoArchivo.split("#")[4]
                booleanFlagCombustible = False
                if(flagResetCombustible == "T"):
                    booleanFlagCombustible = True
                    #print ("El Flag vale True")
                #print ("fichero de datos [4]: " + str(flagResetCombustible))
```

```
            #Guardamos los datos en el modelo
            self.totalDistancia = distanciaTotal
            self.informacionViaje.tripA = distanciaTrip1
            self.informacionViaje.tripB = distanciaTrip2
            self.informacionViaje.combustibleUltimo = deposito
            self.informacionViaje.flagResetRepostaje = booleanFlagCombustible
        except Exception as exception:
            print "Excepcion en InformacionVehiculo: "+ str(excepcion)

    # Metodo que guarda en un fichero los datos de la informacion del vehiculo
    def guardarDatosFichero(self):
        #posible encriptacion
        #Formato 0#0#0#0#F
        totalDistancia = self.totalDistancia
        tripA = self.informacionViaje.tripA
        tripB = self.informacionViaje.tripB
        deposito = self.informacionViaje.combustible
        booleanFlag =  self.informacionViaje.flagResetRepostaje
        flag = "F"
        if(booleanFlag == True):
            flag = "T"
            cadenaGuardar =  str(totalDistancia) + "#" + str(tripA) + "#" + str(tripB) + "#" +
str(deposito) + "#" + str(flag)
        f = open(self.__ruta, "w")
        try:
            f.write(cadenaGuardar)
            f.close()
        except Exception as exception:
            f.close()
            print "Excepcion en InformacionVehiculo: "+ str(excepcion)
```

11.9.3 Fichero "*ControladorLCD.py*"

```
#!/usr/bin/python
# Version 0.0.1 - Mark 1 - ivangf05

from ModuloLCD import ModuloLCD
from ModuloLCDMantenimientos import ModuloLCDMantenimientos
from ModuloLCDViaje import ModuloLCDViaje
from ModuloLCDWS import ModuloLCDWS

class ControladorLCD:
    # Constructor del ControladorLCD
    def __init__(self, informacionVehiculo):
        print "Inicializando ControladorLCD..."
        self.CODIGO_ALERTA = "1"
        self.CODIGO_AVISO = "0"
        self.BOTON_ARRIBA = 3
        self.BOTON_ABAJO = 2
        self.BOTON_DERECHA = 1
        self.BOTON_IZQUIERDA = 4
        self.BOTON_OK =  0
        self.COLOR_AZUL = 4
        self.COLOR_AMARILLO = 3
        self.COLOR_ROJO = 1
        self.informacionVehiculo = informacionVehiculo
        self.__posicionUltima = 0
        self.cambioPantalla = True
        self.__hayNuevaAlerta = False
        self.__hayNuevoAviso = False
        self.__visualizadaAlerta = False
        self.__visualizadoAviso = False
        self.modulos = []
        self.moduloAvisos = self.__cargarModuloAvisos()
        self.moduloAlertas = self.__cargarModuloAlertas()
```

```
                    self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_PRINCIPAL,
self.__cargarModuloPrincipal())
                self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_INCLINOMETRO,
self.__cargarModuloInclinometro())
                self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_TEMPERATURA,
self.__cargarModuloTemperatura())
                            self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_VIAJE,
self.__cargarModuloViaje())
            self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_MANTENIMIENTOS,
self.__cargarModuloMantenimientos())
                self.modulos.insert(self.informacionVehiculo.COD_PANTALLA_WEBSERVICES,
self.__cargarModuloWebServices())

    # Metodo que comprueba si hay avisos o Alertas, devuelve True o False
    def _comprobarAvisosAlertas(self):
        if((self.__comprobarAlertas() is True) or (self.__comprobarAvisos() is True)):
            return True
        else:
            return False

     # Metodo que comprueba si hay Alertas y activa el flag __hayNuevaAlerta si procede
hacerlo.
    def __comprobarAlertas(self):
        if(self.__hayNotificacionesTipo( self.CODIGO_ALERTA) is True):
            if((self.__hayNuevaAlerta is False) and (self.__visualizadaAlerta is False)):
                self.__hayNuevaAlerta = True
                return True
            return True
        else:
            self.__hayNuevaAlerta = False
            self.__visualizadaAlerta = False
            return False
```

```
    # Metodo que comprueba si hay Avisoy activa el flag __hayNuevaAviso si procede hacerlo.
    def __comprobarAvisos(self):
        if(self.__hayNotificacionesTipo(self.CODIGO_AVISO) is True):
            if((self.__hayNuevoAviso is False)and (self.__visualizadoAviso is False)):
                self.__hayNuevoAviso = True
                return True
            return True
        else:
            self.__hayNuevoAviso = False
            self.__visualizadoAviso = False
            return False


    # Metodo que comprueba si hay Notificaciones de un tipo con el flag de activado a True
    def __hayNotificacionesTipo(self, tipo):
                                                    listaNotificaciones = self.informacionVehiculo.informacionNotificaciones.tablaHash.values()
        for notificacion in listaNotificaciones:
            if(notificacion.tipo == tipo):
                if(notificacion.activado is True):
                    return True
        return False

    # Metodo que recibe un codigo de pulsacion, y lo trata en consecuencia
    def realizarPulsacion(self, code):
        if(self.informacionVehiculo.subPantalla != -1):
            if(self.modulos[self.informacionVehiculo.subPantalla].tieneFuncionalidad(code) is True ):
                self.modulos[self.informacionVehiculo.subPantalla].realizarPulsacion(code)
            else:
                self.pulsacionControlador(code)
        else:
            self.pulsacionControlador(code)
```

```
    # Metodo encargado de realizar la accion de la pulsacion por parte del propio ControladorLCD
  def pulsacionControlador(self, code):
    if(code == 1):#Right
      self.__posicionSiguiente()
    if(code == 4):#left
      self.__posicionAnterior()

  # Metodo que devuelve el mensaje LCD que debe visualizarse en la pantalla
  def getMensajeLCD(self):
    print "Llamada a obtener mensajeLCD"
    if(self.informacionVehiculo.subPantalla != -1):
      if((self.__hayNuevaAlerta is True) or (self.__hayNuevoAviso is True)):
        if((self.__hayNuevaAlerta is True) and (self.__visualizadaAlerta is False)):
          self.__posicionUltima =  self.informacionVehiculo.subPantalla
          self.informacionVehiculo.subPantalla = -1
          if(self.__hayNotificacionesTipo(self.CODIGO_ALERTA) == True):
            self.__visualizadaAlerta = True
            return self.moduloAlertas.getMensajeLCD()
        if((self.__hayNuevoAviso is True) and (self.__visualizadoAviso is False)):
          self.__posicionUltima =  self.informacionVehiculo.subPantalla
          self.informacionVehiculo.subPantalla = -1
          if(self.__hayNotificacionesTipo(self.CODIGO_AVISO) == True):
            self.__visualizadoAviso = True
            return self.moduloAvisos.getMensajeLCD()
        return self.modulos[self.informacionVehiculo.subPantalla].getMensajeLCD()
      else:
        return self.modulos[self.informacionVehiculo.subPantalla].getMensajeLCD()
    else:
      if(self.__hayNotificacionesTipo(self.CODIGO_ALERTA) == True):
        return self.moduloAlertas.getMensajeLCD()
      if(self.__hayNotificacionesTipo(self.CODIGO_AVISO) == True):
        return self.moduloAvisos.getMensajeLCD()
```

```
    #Metodo que develve el color que debe visualizarse en la pantalla LCD
    def getColorLCD(self):
        if(self.informacionVehiculo.subPantalla != -1 ):
            return self.modulos[self.informacionVehiculo.subPantalla].getColorLCD()
        else:
            if(self.__hayNotificacionesTipo(self.CODIGO_ALERTA)  == True):
                return self.moduloAlertas.getColorLCD()
            if(self.__hayNotificacionesTipo(self.CODIGO_AVISO) == True):
                return self.moduloAvisos.getColorLCD()


     # Metodo que intenta retroceder al anterior Menu y en caso necesario se mueve a
Alertas/Avisos
    def __posicionAnterior(self):
        #La posicion actual es distinta de la pantalla Avisos/Alertas
        if(self.informacionVehiculo.subPantalla != -1):
            #Hay que comprobar si hay Alertas/Avisos
            if((self.__hayNotificacionesTipo(self.CODIGO_ALERTA) == True ) or
(self.__hayNotificacionesTipo (self.CODIGO_AVISO) is True) ):
                #Hay Alertas/Avisos
                self.__posicionUltima =  self.informacionVehiculo.subPantalla
                self.informacionVehiculo.subPantalla = -1
                self.cambioPantalla = True
            else:
                #No hay Alertas/Avisos
                self.__retrocederPosicion()
        else:
            self.informacionVehiculo.subPantalla = self.__posicionUltima
            self.__retrocederPosicion()


     # Metodo que intenta avanzar al siguiente Menu y en caso necesario se mueve a
Alertas/Avisos
    def __posicionSiguiente(self):
        #La posicion actual es distinta de la pantalla Avisos/Alertas
```

```
        if(self.informacionVehiculo.subPantalla != -1):
            #Hay que comprobar si hay Alertas/Avisos
if((self.__hayNotificacionesTipo(self.CODIGO_ALERTA) == True ) or
(self.__hayNotificacionesTipo (self.CODIGO_AVISO) is True) ):
                #Hay Alertas/Avisos
                self.__posicionUltima =  self.informacionVehiculo.subPantalla
                self.informacionVehiculo.subPantalla = -1
                self.cambioPantalla = True
            else:
                #No hay Alertas/Avisos
                self.__avanzarPosicion()
        else:
            self.informacionVehiculo.subPantalla = self.__posicionUltima
            self.__avanzarPosicion()

        # Metodo que realiza el retroceso a la posicion anterior de menu, lo utiliza
__posicionAnterior()
    def __retrocederPosicion(self):
        posicion = self.informacionVehiculo.subPantalla - 1
        if posicion < 0:
            posicion = self.modulos.__len__() -1
        self.informacionVehiculo.subPantalla = posicion
        self.cambioPantalla = True
        if(posicion == self.informacionVehiculo.COD_PANTALLA_MANTENIMIENTOS):
            self.modulos[self.informacionVehiculo.subPantalla].inicializarFlagPantallaAjuste()
        self.modulos[self.informacionVehiculo.subPantalla].posicion = 0

        # Metodo que realiza el avance a la posicion anterior de menu, lo utiliza
__posicionSiguiente()
    def __avanzarPosicion(self):
        posicion = self.informacionVehiculo.subPantalla  + 1
        if posicion >= self.modulos.__len__():
            posicion = 0
        self.informacionVehiculo.subPantalla = posicion
```

```
        self.cambioPantalla = True
        if(posicion == self.informacionVehiculo.COD_PANTALLA_MANTENIMIENTOS):
            self.modulos[self.informacionVehiculo.subPantalla].inicializarFlagPantallaAjuste()
        self.modulos[self.informacionVehiculo.subPantalla].posicion = 0

    # Metodo que indica si hay o no cambios de pantalla
    def hayCambiosPantalla(self):
        self._comprobarAvisosAlertas()
        if(self.cambioPantalla is True):
            self.cambioPantalla = False
            return True
        else:
            if(self.informacionVehiculo.subPantalla != -1):
                return self.modulos[self.informacionVehiculo.subPantalla].hayCambiosPantalla()
            return False

    # Metodo que carga el Modulo LCD del menu Principal
    def __cargarModuloPrincipal(self):
        print "Carga del modulo LCD Principal..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Principal"]
        listaCodigos = []
         moduloInicio = ModuloLCD (self.informacionVehiculo, self.COLOR_AZUL, mensaje,
listaSubmenus, listaCodigos)
        return moduloInicio

    # Metodo que carga el Modulo LCD del menu Inclinometro
    def __cargarModuloInclinometro(self):
        print "Carga del modulo LCD Inclinometro..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Inclinometro"]
        listaCodigos = []
            moduloInclinometro = ModuloLCD (self.informacionVehiculo, self.COLOR_AZUL,
mensaje, listaSubmenus, listaCodigos)
```

```
        return moduloInclinometro

    # Metodo que carga el Modulo LCD del menu Temperatura
    def __cargarModuloTemperatura(self):
        print "Carga del modulo LCD Temperaturas..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Temperaturas"]
        listaCodigos = [self.BOTON_ABAJO,self.BOTON_ARRIBA]
            moduloTemperaturas = ModuloLCD (self.informacionVehiculo, self.COLOR_AZUL,
mensaje, listaSubmenus, listaCodigos)
        return moduloTemperaturas

    # Metodo que carga el Modulo LCD del menu Viaje
    def __cargarModuloViaje(self):
        print "Carga del modulo LCD Viaje..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Viaje", "Reset Trip A", "Reset Trip B", "Aut TripB"]
        listaCodigos = [self.BOTON_ABAJO,self.BOTON_ARRIBA, self.BOTON_OK]
            moduloViaje = ModuloLCDViaje (self.informacionVehiculo, self.COLOR_AZUL,
mensaje, listaSubmenus, listaCodigos)
        return moduloViaje

    # Metodo que carga el Modulo LCD del menu Mantenimientos
    def __cargarModuloMantenimientos(self):
        print "Carga del modulo LCD Mantenimientos..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Mantenimientos", "Realizar Mant", "Protector", "Ajustar KM"]
        listaCodigos = [self.BOTON_ABAJO,self.BOTON_ARRIBA, self.BOTON_OK]
            moduloMantenimientos = ModuloLCDMantenimientos (self.informacionVehiculo,
self.COLOR_AZUL, mensaje, listaSubmenus, listaCodigos)
        return moduloMantenimientos

    # Metodo que carga el Modulo LCD del menu Alertas
    def __cargarModuloAlertas(self):
```

```
        print "Carga del modulo LCD Alertas..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Alertas"]
        listaCodigos = []
                moduloAlertas = ModuloLCD (self.informacionVehiculo, self.COLOR_ROJO,
mensaje, listaSubmenus, listaCodigos)
        return moduloAlertas

    # Metodo que carga el Modulo LCD del menu Avisos
    def __cargarModuloAvisos(self):
        print "Carga del modulo LCD Avisos..."
        mensaje = "Citroen 2cv  <+>"
        listaSubmenus = ["Avisos"]
        listaCodigos = []
            moduloAvisos = ModuloLCD (self.informacionVehiculo, self.COLOR_AMARILLO,
mensaje, listaSubmenus, listaCodigos)
        return moduloAvisos

    # Metodo que carga el Modulo LCD del menu Mantenimientos
    def __cargarModuloWebServices(self):
        print "Carga del modulo LCD Mantenimientos..."
        mensaje = "Citroen 2cv"
        listaSubmenus = ["Incidencias", "Retencion", "Averia", "Mantenimiento", "Otro"]
        listaCodigos = [self.BOTON_ABAJO,self.BOTON_ARRIBA, self.BOTON_OK]
                moduloWS = ModuloLCDWS (self.informacionVehiculo, self.COLOR_AZUL,
mensaje, listaSubmenus, listaCodigos)
        return moduloWS
```

11.9.4 Fichero "*ModuloLCDWS.py*"

```
#!/usr/bin/python
# Version 0.0.2 - Mark 1 - ivangf05


import sys
```

```
from ModuloLCD import ModuloLCD
from RestManager import RestManager
class ModuloLCDWS(ModuloLCD):

   # Constructor de ModuloLCDWS
            def __init__(self, informacionVehiculo,color,      primeraLinea,  listaSubMenu,
listaCodFuncion):
       print "Inicializando ModuloLCDWS..."
          ModuloLCD.__init__(self, informacionVehiculo, color, primeraLinea, listaSubMenu,
listaCodFuncion)
       self.llamadaWS = False
       self.error = False

   # Metodo que retorna la cadena que debe mostrar el ModuloLCD
   def getMensajeLCD(self):
                        primeraLinea  =  self._addCadenaSignoFin(self.primeraLinea,  "<+>",
self.TAMANIO_LCD )
       if(self.llamadaWS == True):
          print ("llamadaWS es True")
          self.llamadaWS = False
          if(self.error == True):
             segundaLinea ="Error Conexion"
             self.color = 1
          if(self.error == False):
             segundaLinea ="Ok"
             self.color = 4
          return str(primeraLinea) +'\n' + str(segundaLinea)
       self.color = 4
       segundaLinea = self.listaSubMenu[self.posicion]
       if(self.posicion != 0):
          segundaLinea = self._addCadenaSignoInicio(segundaLinea, ">")
       if(self.posicion == 0):
          segundaLinea = self._addCadenaSignoFin(segundaLinea, "+", self.TAMANIO_LCD
)
```

```
        return str(primeraLinea) +'\n' + str(segundaLinea)

    # Metodo que realiza la pulsacion sobre el ModuloLCDWS
    def realizarPulsacion(self, code):
        self.cambioPantalla = True
        if(code == 0):#ok
            self.__realizarPulsacionOK()
        if(code == 3):#up
            self._posicionAnterior()
        if(code == 2):#down
            self._posicionSiguiente()

    # Metodo que realiza la accion del boton OK sobre el Modulo
    def __realizarPulsacionOK(self):
        try:
            self.error = False
            self.llamadaWS = True
            if(self.posicion == 1):
                RestManager.addPoint("0", self.informacionVehiculo)
            if(self.posicion ==2):
                RestManager.addPoint("1", self.informacionVehiculo)
            if(self.posicion == 3):
                RestManager.addPoint("2", self.informacionVehiculo)
            if(self.posicion == 4):
                RestManager.addPoint("3", self.informacionVehiculo)
            RestManager.getAllPoints(self.informacionVehiculo)
        except Exception:
            self.error = True
            print("Error inesperado:", sys.exc_info()[0])
```

11.9.5 Fichero "*InformacionWebServices.py*"

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
# Version 0.0.2 - Mark 1 - ivangf05

class InformacionWebServices:

    # Constructor Point
    def __init__(self):
        print "inicializando InformacionWebServices..."
        self.listaPoints = []
        self.error = False
```

11.9.6 Fichero "*Point.py*"

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Version 0.0.2 - Mark 1 - ivangf05

class Point:

    # Constructor Point
    def __init__(self, latitude, longitude, codeAlert):
        print "inicializando Point..."
        print ("Nuevo Punto lat " + str(latitude) + "lon " + str(longitude))
        self.latitude = latitude
        self.longitude = longitude
        self.codeAlert = codeAlert
        self.distance = -1
        self.dir = ""

    def printValue(self):
                        resultado = Point.getNombreAlerta(self.codeAlert) + " a " +
str("{0:.2f}".format(self.distance)) + " km direccion" + self.dir
        print(resultado)
        return resultado
```

```
    @staticmethod
    def getNombreAlerta(code):
        if(str(code) == "0"):
            return "Accidente"
        if(str(code) == "1"):
            return "Atasco"
        if(str(code) == "2"):
            return "Mantenimiento"
        if(str(code) == "3"):
            return "Otro"
```

11.9.7

11.9.8 Fichero "*RestManager.py*"

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Version 0.0.2 - Mark 1 - ivangf05

from InformacionViaje import InformacionViaje
from InformacionVehiculo import InformacionVehiculo
from Point import Point
import urllib
import urllib2

class RestManager:
    __ipServer = "http://192.168.0.195:8080"
    #__ipServer = "http://localhost:8080"

    # Constructor del ManagerRest
    #def __init__(self, informacionVehiculo):
    #    print "inicializando ManagerRest..."
    #    self.__informacionVehiculo = informacionVehiculo
```

```
    # Metodo addPoint
    @staticmethod
    def addPoint(tipoIncidencia, informacionVehiculo):
        url = RestManager.__ipServer + "/addPointOfInterest"
        headers = {"Content-Type": "application/json"}
                                                                                #?
latidude=latidude&codeOrigin=codeOrigin&codeAlert=codeAlert&longitude=longitude&
                    #data = {"latidude": str(self.__informacionVehiculo.latitud), "longitude":
str(self.__informacionVehiculo.longitud), "codeOrigin": "OnBoard", "codeAlert":
str(tipoIncidencia)}
        #try:
        url = url +"?latidude="+str(informacionVehiculo.latitud)
        url = url +"&longitude="+str(informacionVehiculo.longitud)
        url = url +"&codeOrigin=OnBoard"
        url = url +"&codeAlert="+str(tipoIncidencia)
        print (url)

        request = urllib2.Request(url)
        # post form data
        #request.add_data(urllib.urlencode(data))
        for key, value in headers.items():
            request.add_header(key, value)
        response = urllib2.urlopen(request)
        print response.info().headers
        print response.read()
        #except Exception:
            #Tratamiento de la excepcion
            #print "Oops!  That was no valid number.  Try again..."

    # Metodo allPoint
    @staticmethod
    def getAllPoints(informacionVehiculo):
        url = RestManager.__ipServer + "/getPointsOfInterest"
        headers = {"Content-Type": "application/json"}
```

```
    #  try:
      request = urllib2.Request(url)
      for key, value in headers.items():
          request.add_header(key, value)
      response = urllib2.urlopen(request)
      #opener = urllib2.build_opener()
      #response = opener.open(request)

      print response.info().headers
      strResponse = response.read().decode('utf-8')
      print strResponse
      listaPoints = RestManager.parseJSON(strResponse)
      informacionVehiculo.informacionWebServices.listaPoints = listaPoints
    #  json_data = json.loads(response.read().decode('utf-8'))
          #simpleparse(str(response.read()))
     # except Exception:
     #     print "Oops!  That was no valid number.  Try again..."

  @staticmethod
  def parseJSON(cadenaJSON):
      print ("cadena a convertir" + cadenaJSON)
      listaStringObjetos =  cadenaJSON.split("}")
      listaObjetos = []
      #print("objetos " + str(len(listaStringObjetos)))
      for cadenaObjeto in listaStringObjetos[:-1]:
          latitud =""
          longitud =""
          codigo =""
          print("objeto " + str(cadenaObjeto))
          listaAtributos = cadenaObjeto.split(",")
          for cadenaAtributo in listaAtributos[1:]:
              print("atributo " + str(cadenaAtributo))
              # Obtenemos el nombre del atributo
              #print("sizesplit: "+str(len(cadenaAtributo.split(""))))
```

```
            strNombre = cadenaAtributo.split("")[1]
            print("nombre Atributo:" + strNombre)
             print("----------------------------" + str(strNombre == "latitude") + " " + strNombre + " "
+ "latitude")
            if(strNombre == "latitude"):
               strLat = cadenaAtributo.split(':')[1]
               print ("EL Atributo es latitude " + strLat)
               latitud = strLat
            if(strNombre == "longitude"):
               strLon = cadenaAtributo.split(':')[1]
               #print ("EL Atributo es longitude " + strLon)
               longitud = strLon
            if(strNombre == "codeAlert"):
               strAlert = cadenaAtributo.split("")[3]
               #print ("EL Atributo es codeAlert " + strAlert)
               codigo = strAlert
         point = Point (latitud, longitud, codigo)
         listaObjetos.append(point)
      #print("size lista object result " +  str(len(listaObjetos)))
      return listaObjetos



# TEST CODE
#informacionVehiculo = InformacionVehiculo()
#restManager = RestManager(informacionVehiculo)
#RestManager.addPoint("1", informacionVehiculo)
#RestManager.getAllPoints( informacionVehiculo)
#gestorViaje = GestorViaje(informacionVehiculo)
#gestorViaje.informacionVehiculo.guardarDatosFichero()
```