UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster en Ingeniería de Computadores y Redes

Trabajo Fin de Máster

Sistema Embebido para Vigilancia Remota de Vehículos

Autor: *Lucas Nunes da Silva Neto*

Director: *Carlos Tavares Calafate*

Septiembre de 2016

1 ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

De acuerdo con la visión del Internet de las Cosas (IoT), aparatos domésticos, actuadores y sistemas embebidos de todos los tipos estarán conectados unos a otros mediante Internet, formando una gran red y abriendo puertas a nuevos servicios basados en este paradigma. La creciente cantidad de aparatos que están adquiriendo capacidad de conectarse a Internet e intercambiar datos entre ellos mismos, gracias a la evolución de la IoT, aumenta nuestro campo de posibilidades de servicios que podrían ser suministrados a partir del uso de esta tecnología.

Para que las personas adopten cada vez más dispositivos que integren este entorno de IoT, contribuyendo así a su crecimiento, es necesario que los servicios ofrecidos sean precisos, fiables, y que estén involucrados con las actividades del día a día de las personas comunes, haciendo más cómodas sus vidas y proporcionando mayor seguridad.

La capacidad de conectarse a Internet es una característica cada vez más frecuente en los dispositivos actuales, los cuales tienen cada vez mayor capacidad de cómputo y funcionalidades. Tener estos dispositivos siempre a nuestro alcance puede hacer nuestras tareas diarias más sencillas y seguras.

1.1 MOTIVACIÓN

El Internet de las Cosas es capaz de ofrecer diversos tipos de servicios dependiendo de la rama en que se aplican, ya sea en la salud, escenarios militares o domésticos. Cada vez más, diferentes tipos de objetos, que van desde los más simples, como ropas e accesorios, hasta los más complejos como electrónicos y medios de transporte, disponen de conexión hacia Internet, y concretamente la IoT tiene como objetivo hacer con que estos objetos intercambien información, permitiendo analizar las opciones y elecciones habituales en nuestras vidas cotidianas de cara a tornarlas mejores y más eficientes.

En diversos países del mundo los índices de criminalidad han crecido en los últimos años, y prácticas como el robo de coches son cada vez más comunes en estos sitios. Actividades simples como conducir un coche por la ciudad pueden hacerse más seguras, desde el punto de vista de protección de tu bien material, si conocemos la ubicación del coche en todo momento, algo que puede ser posible usando los servicios que IoT nos ofrece.

Añadir capacidad de cómputo y de conexión a Internet en un coche, aún que por medio de un dispositivo externo, puede ser una solución para ahuyentar o intimidar a personas mal intencionadas, además de servir como ayuda en el momento de abordaje por parte de la policía a coches sospechosos basados en datos que provienen de un sistema embebido.

1.2 OBJETIVOS

El objetivo de este Trabajo Fin de Máster consiste en mostrar que es posible integrar objetos de nuestro día a día en un entorno IoT mediante dispositivos que ya hacen parte de este dominio, ofreciendo servicios y haciendo la vida más fácil no solamente para el propietario de dicho objeto, sino que también para personas en su entorno social.

Utilizar Internet y los dispositivos inteligentes como una herramienta adicional de seguridad, tanto para los propietarios de coches como para la policía, puede ser realmente útil ya que, por medio de un sistema embebido, podrán identificar coches sospechosos y actuar con más rapidez y precisión. De esta manera la policía se beneficiaria de los servicios ofrecidos por un sistema embebido, que será detallado en las próximas secciones, teniendo así la tecnología a su favor como una herramienta más para el combate a la criminalidad, lo que viene a ser el segundo objetivo de este Trabajo Fin de Máster.

Además, se analizarán posibilidades de mejora y adaptaciones del sistema en relación a su escalabilidad, estudiando quién o qué cosas más podrían acceder a estos datos provenientes de los coches, para que el sistema pueda ser aprovechado en diferentes entornos.

1.3 TRABAJOS RELACIONADOS

A continuación se presentan ejemplos de sistemas para llevar a cabo el desarrollo de aplicaciones automotrices que ofrecen servicios para este entorno y contribuyen para la conectividad de los coches.

1.3.1 eCall

La llamada de emergencia automática realiza una marcación al número 112 sin la intervención del conductor, cuando detecta que se ha producido un accidente grave (por ejemplo, si dispara un airbag). Este Inteligente sistema también envía las coordenadas GPS del coche, fecha, hora y sentido de la circulación.

Este sistema utiliza la tecnología GSM para la comunicación con los servicios de emergencia y por eso necesita una tarjeta SIM conectada a través de un módulo directamente al coche o bien, mediante un teléfono móvil del conductor conectado por bluetooth al coche [1].

1.3.2 BMW Connected Drive

Con esta aplicación los propietarios de coches BMW pueden tener acceso a sus coches a través del centro de servicios remotos BMW y disfrutar de un conjunto de funcionalidades ofrecidas por este sistema. Una de las funcionalidades es encontrar el coche por medio del posicionamiento global que es enviado al móvil del conductor y puede ser visto en un mapa en tiempo real. A parte de eso hay otras funcionalidades como ayuda a encontrar el coche en un aparcamiento lleno, donde por medio del móvil el conductor puede parpadear las luces de su coche para facilitar la ubicación [2].

1.3.3 CarCentinel

Sistema antirrobo fabricado en España y directamente conectado al móvil. Unos sensores de movimiento instalados en el coche permiten a través de esta APP, acceder a la posición del coche y incluso bloquearlo desde el móvil en caso de que te lo hayan robado. Sin embargo, otras informaciones más como el tiempo que lleva el coche arrancado, cuantos kilómetros ha recorrido y otras más son indicadas [3].

1.4 ESTRUCTURA DEL DOCUMENTO

Este Trabajo Fin de Máster se encuentra dividido en siete capítulos que van desde la introducción hasta las conclusiones y trabajo futuro, y que tratan de todos los temas que han surgido desde el inicio al final de esta investigación.

En el presente capítulo se ofrece una visión general del trabajo que se ha realizado. En el segundo capítulo hablaremos de la base del proyecto, que es la Internet de las Cosas, pasando por una visión general, su crecimiento, y mirando algunos ejemplos de aplicaciones existentes en este medio. El capítulo 3 describe las características del hardware utilizado para la realización del trabajo, detallando las características físicas más importantes de estos dispositivos. En el capítulo 4 se caracteriza el entorno de desarrollo del proyecto, detallando las instalaciones necesarias para la siguiente etapa. El quinto capítulo describe las herramientas de desarrollo utilizadas y su implementación en el ámbito del proyecto, empezando por una breve explicación de su propósito y uso dentro del sistema que se ha desarrollado. En el capítulo 6 se presentan los resultados obtenidos después de finalizar la etapa de desarrollo; en esta sección se analizarán las prestaciones y funcionalidades ofrecidas por el sistema embebido. Para finalizar, en el capítulo 7, se presentan las conclusiones del trabajo, así como los posibles trabajos futuros que podrían dar continuidad al presente proyecto.

2 INTERNET DE LAS COSAS

2.1 VISIÓN GENERAL

De acuerdo muchos, el avance del Internet tiene grandes impactos, como por ejemplo lo que afirma este autor: “Internet of Things (IoT), también llamado Internet of Objects, lo cambiará todo, incluidos a nosotros mismos”(Evans, 2011). Tal afirmación puede ser comprobada si considerarnos el impacto que Internet ha tenido sobre áreas como la educación, la comunicación, las empresas y la ciencia, por ejemplo. Claramente, el Internet es una de las creaciones más importantes y poderosas de toda la historia de la humanidad.

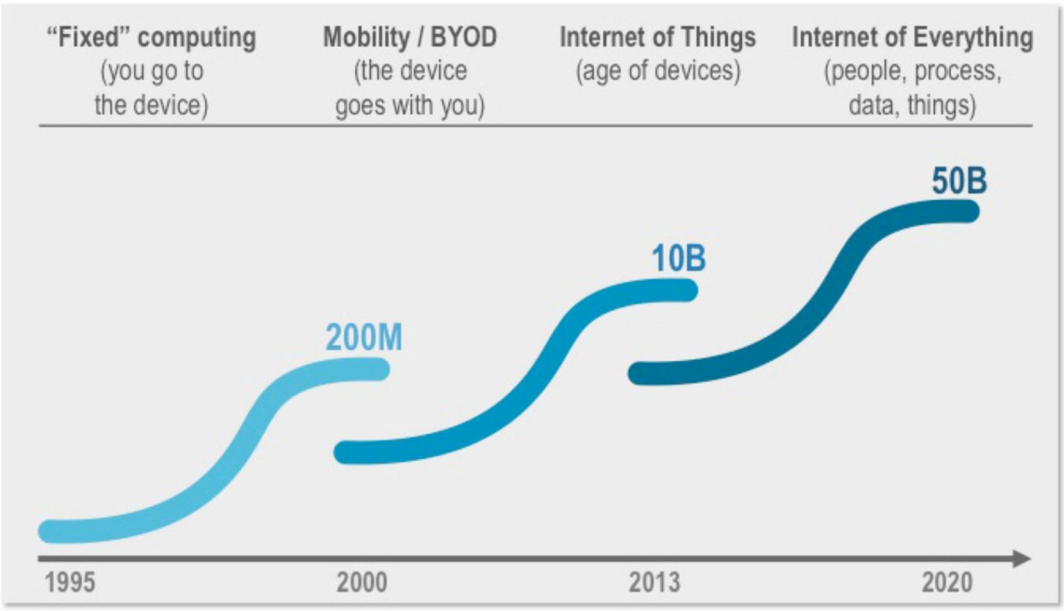
2.1.1 ¿Qué es el Internet de las Cosas?

La Internet de las Cosas (IdC), o Internet of Things (IoT), es un paradigma de comunicación que cada vez más está presente en nuestro entorno, que hace que los objetos (electrónicos, domésticos, digitales, analógicos y otros) de nuestro día a día estén equipados con micro controladores, transcodificadores y cadenas de protocolos que se adecuan para permitir la comunicación entre estos dispositivos y los usuarios. La idea básica de este paradigma es la omnipresencia de objetos o “cosas” cerca de nosotros que, a través de esquemas únicos de direccionamiento, son capaces de interactuar los unos con los otros para realizar tareas comunes.

2.2 ESTADO DEL ARTE TECNOLÓGICO

El Internet de las Cosas es sin duda uno de los desarrollos tecnológicos más importantes de la última década. Esta tecnología, por ser altamente heterogénea y disruptiva, ha experimentado un importante crecimiento, como se puede observar en la figura adjunta, llegando a números impresionantes como los 50 billones de dispositivos conectados a Internet en 2020, pasando entonces a ser una evolución del IoT conocida como Internet de Todo (Internet of Everything - IoE) [5].

Figura 1. Crecimiento del número de cosas conectadas al internet



Fuente: White paper, Cisco [5] p. 2.

Es importante señalar que las estimaciones presentadas por las fuentes anteriormente citadas no tienen en cuenta los rápidos avances producidos por la tecnología de Internet o de los dispositivos, lo que significa que estas cifras se basan en los conocimientos que se tiene hoy en día en tema de tecnología.

El éxito del internet de las Cosas depende, en gran medida, de una arquitectura bien definida que proporcionará una base segura, dinámica y escalable para su despliegue. Tal arquitectura tendrá que ser estándar, bien definida, segura y compatible con tecnologías anteriores.

2.2.1 Arquitecturas

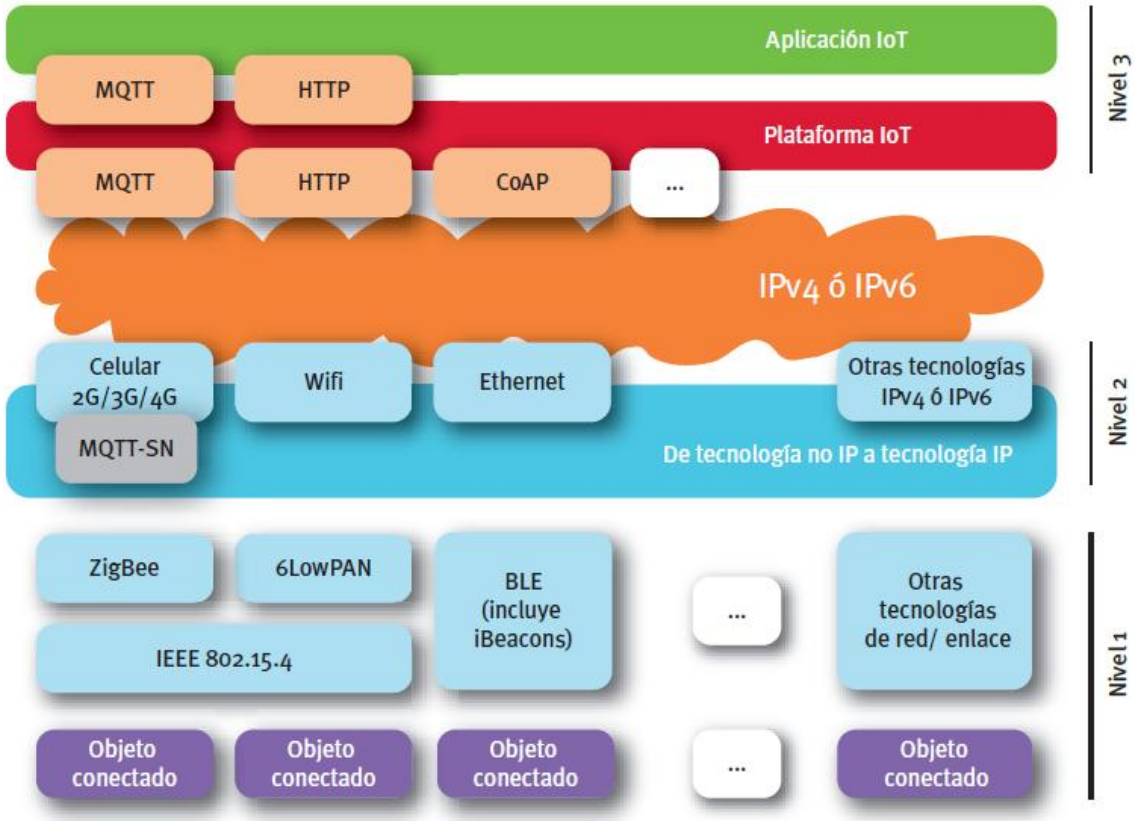
Actualmente es posible notar los distintos tipos de arquitecturas [6] implementadas dentro del IoT, mirando al nivel de objetos conectados e integración con la nube o plataforma IoT. En las siguientes secciones se dará una breve explicación acerca de cada una de ellas.

2.2.1.1 Arquitectura de tres niveles con objetos conectados sin protocolo IP

Esta primera arquitectura ha sido bastante utilizada durante años en entornos en los que la grande cantidad de dispositivos de baja capacidad y coste era la principal prioridad.

La figura 2 muestra el esquema en cuestión, y en el cual la arquitectura ha sido pensada para ahorrar tiempo de cómputo y energía de los nodos, que es lo que pasaría por el uso de tecnología con pila de protocolo IP.

Figura 1 - Arquitectura de tres niveles con objetos conectados sin protocolo IP



Fuente: Libro digital, EOI [6] p. 133.

En el primero nivel de esta arquitectura suele tener una conectividad punto a punto por medio de Bluetooth Low Energy (BLE) o bien sobre conexiones sobre ZigBee, Z-Wave, Wireless-HART ™, 6LowPAN [7], que es un caso especial de IP, o similares que se conectarán hacia una pasarela IP, y que utilizaría tecnologías comunes como WIFI (IEEE 802.11), Ethernet (IEEE 802.3) o Celular (GPRS, EDGE, UTMS…); podrían igualmente ser utilizadas otras tecnologías que son menos populares en estos tipos de soluciones, como es el caso de WiMax (IEEE 802.16).

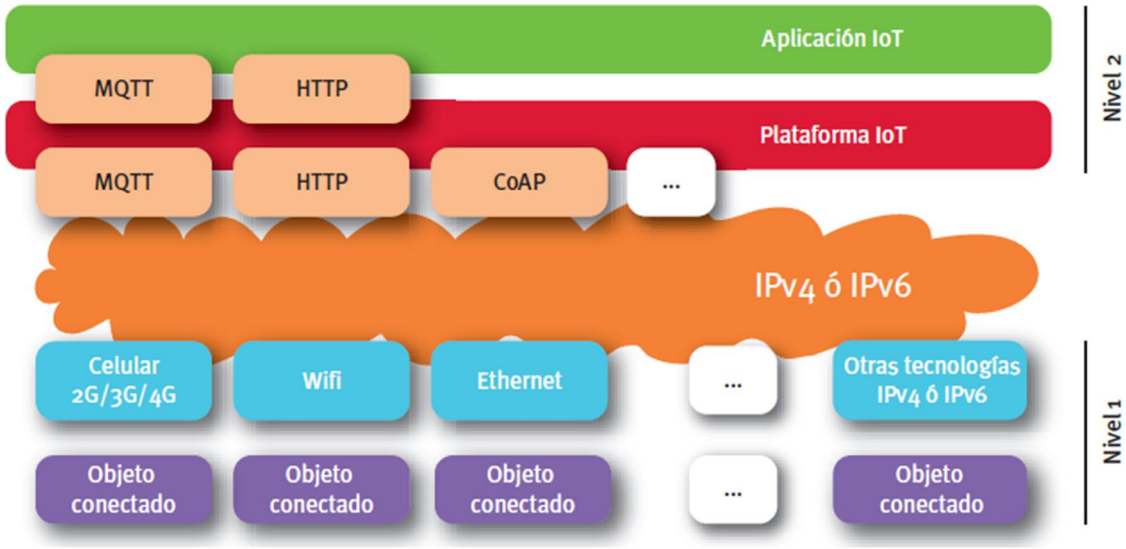
En la capa que sigue, una serie de servicios se ofrecerán para el desarrollo de aplicaciones que permitan la presentación de datos que serán

colectados en este nivel que para eso se interconectará con el cloud o plataforma IoT.

2.2.1.2 Arquitectura de dos niveles con objetos conectados con protocolo IP

Esta arquitectura incorpora una tecnología con conectividad IP directamente, estilo WiFi o modem celular, donde los objetos equipados con conectividad IP son capaces de comunicarse directamente con el siguiente nivel, el cloud o plataforma IoT. Actualmente el mercado y organismos estandarizadores trabajan con soluciones celulares de bajo consumo y coste como el LTE-M, EC-GSM, Clean Slate e etc.

Figura 2 - Arquitectura de dos niveles con objetos conectados con protocolo IP



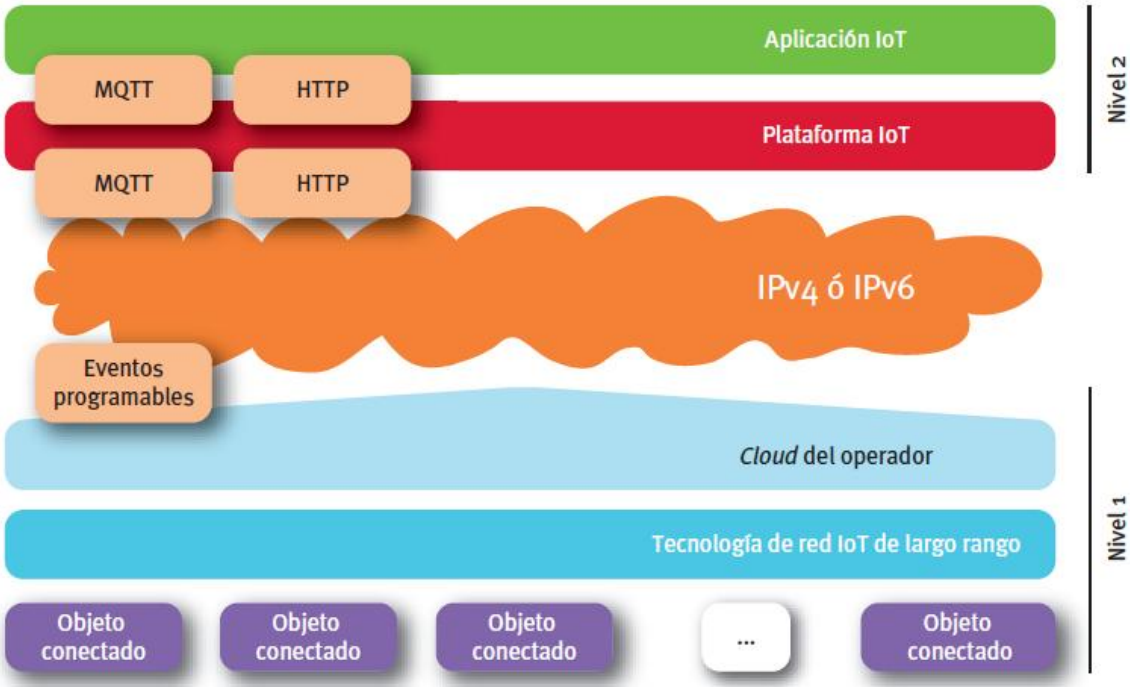
Fuente: Libro digital, EOI [6] p. 135.

Para este caso el hardware requerido para los objetos será más potente una vez que los protocolos utilizados por estos, como MQTT, CoAP o HTTP con servicios RESTful, serán más complejos que en el caso anterior, y exigirán más capacidad de proceso y memoria integrada.

2.2.1.3 Arquitectura de dos niveles con objetos conectados sin protocolo IP

En esta arquitectura se utilizan nuevos protocolos de red específicos para IoT con su propia red no IP, los cuales han surgido con la evolución de los protocolos existentes proporcionando nuevas soluciones técnicas.

Figura 3 - Arquitectura de dos niveles con objetos conectados sin protocolo IP



Fuente: Libro digital, EOI [6] p. 136.

Las tecnologías adoptadas en este caso ofrecen una interacción directa o casi directa entre el cloud y los objetos conectados, pudiendo así ser considerada una arquitectura de dos niveles.

Aproximaciones a este esquema de arquitectura han surgido como la de Sigfox, Weighless y LoRa, por ejemplo, ofreciendo comunicación de manera directa al segundo nivel.

2.2.2 Tecnologías IoT

Una vez analizados los principales grupos de arquitecturas implementadas dentro de IoT, se procederá a describir las tecnologías más relevantes dentro de este paradigma, clasificándolas en grupos en función de sus propios elementos y de los interfaces entre ellos. Para este estudio la clasificación dada se describirá en las próximas secciones.

2.2.2.1 Tecnologías de hardware

Los dispositivos hardware necesitan una capacidad de proceso y una programación en los mismos. Sin embargo, hay varios fabricantes que están ofreciendo soluciones para el estándar IEEE 802.15.14 como Z1 de Zolertia, por ejemplo, que es una plataforma de desarrollo para redes de sensores inalámbricos con fines de investigación que también es utilizada en

entornos de producción. También hay la plataforma de código abierto TelosB, que es una plataforma diseñada por la UC Berkeley para ofrecer capacidades de experimentación de última generación para desarrollo de investigaciones Low Power o experimentaciones didácticas. Otro ejemplo de solución de hardware para el desarrollo de IoT sería el OpenMote-CC2538, desarrollada por una empresa española, pero no se puede olvidar de la importante contribución que han tenido a la accesibilidad de desarrollo en el IoT las plataformas de hardware abierto, representadas por sus máximos exponentes: Arduino y Raspberry Pi.

El primero es una plataforma de hardware libre de sencilla programación y poca capacidad, pero muy accesible, y que fue desarrollada en Italia con fines educacionales. Los tres pilares del Arduino son: Hardware, Software y Comunidad activa. El segundo es un microordenador creado por la Raspberry Pi Foundation y por la Universidad de Crambridge, con el propósito de promover el estudio de la Ciencia de la Computación en las escuelas de ensino básico y secundario. Este dispositivo es capaz de ejecutar versiones básicas de sistemas operativos. Ambas han servido como plataformas a muchos desarrolladores, y como punto de partida para creación de nuevas ideas y conceptos.

2.2.2.2 Tecnologías de software

El IoT hereda diversas arquitecturas de microcontroladores, desde las más comunes en el mercado, basando los productos en núcleos ARM, ARV, PIC y 8o8x, hasta las más recientes como ARM-Cortex M, entre otros. Distintos sistemas operativos intentaron tomar posición sobre estos núcleos, como es el caso del VxWorks o eCos. Pero nuevas propuestas enfocadas en este sector siguen mejorando prestaciones, como es el caso del Contiki, TinyOS, mbed de ARM o el RIOT OS. Este último es un sistema operativo diseñado específicamente para IoT, y se basa en una arquitectura con un micro-kernel similar a Linux, pero totalmente adaptado a las características IoT como bajo consumo y capacidad de memoria y procesamiento limitados.

2.2.2.3 Tecnologías de comunicaciones en objetos conectados

Existen múltiples estándares de comunicación que se adaptan de acuerdo a diferentes necesidades.

En el primero caso podemos citar las tecnologías tradicionales de conectividad inalámbrica, o sea, el WiFi y la conectividad celular (2G, 3G, 4G), que tienen gran cobertura, son altamente soportadas, pero tienen la desventaja de consumir demasiada energía. Dentro de este primero grupo se pueden citar las tecnologías de comunicación nativas del IoT tales como Sigfox, Lora y Weighless, que son tecnologías de diseño barato, y que presentan bajo consumo logrando una buena cobertura pero con una tasa de datos reducida.

El el segundo caso estarían las tecnologías responsables por el despliegue inicial del IoT, como: ZigBee, Z-Wave, TinyMesh, IEEE 802.15.4 y 6LowPan, por ejemplo. Caracterizadas por bajo precio, bajo consumo energético y flexibilidad, estas tecnologías han sido claves para el éxito inicial del IoT, pero en la gran parte de los casos, donde tenemos despliegues amplios, se consideran poco apropiadas ya que trasladan al cliente final la operación de red.

Como gran tendencia, hay que considerar igualmente una rama de la tecnología de conectividad entre objetos y dispositivos móviles, en este caso, el Bluethooth, especificamente su última versión de bajo consumo, el BLE (Bluetooth Low Energy), y el NFC (Near Field Communication), que es una tecnología derivada del RFID. Estos dos últimos están mostrando ser elementos clave en esta interacción denominada área personal.

2.2.3 Aplicaciones

La potencialidad ofrecida por el Internet de las Cosas hace posible el desarrollo de un gran número aplicaciones, de las cuales solo una pequeña parte se encuentra ya disponible para nuestra sociedad. Muchos son los dominios o entornos donde aplicaciones de IoT pueden mejorar la calidad de nuestras vidas, como por ejemplo: aplicaciones domésticas, para el transporte, para la salud y los deportes, entre otras. En una gran cantidad de casos estos entornos están ahora equipados solamente con inteligencia primitiva, muchas de las veces sin ninguna capacidad de comunicación. Dar a estos dispositivos la oportunidad de comunicarse unos con los otros y manejar la información que proviene del alrededor implica tener distintos entornos con una gran variedad de aplicaciones a ser desarrolladas.

Dentro de las posibles aplicaciones se puede distinguir entre aquellas que pueden ser directamente aplicables por ser cercanas a nuestros hábitos, y aquellas futuristas, acerca de las cuales podemos solamente imaginar una vez que nuestras tecnologías o sociedad todavía no están listas para sus desarrollos. Una breve descripción acerca de estos dominios [7] anteriormente citados será dada en las próximas secciones.

2.2.3.1 Aplicaciones de dominio Logístico y de transporte

Los coches recién lanzados en el mercado, así como trenes, buses y hasta bicis, están empezando a llevar cada vez más sensores, actuadores y disponen de un mayor poder de cómputo. En algunos sitios las propias carreteras y los bienes transportados ya llevan chips y sensores que envían información importante para control de tráfico, ayudar en el control de inventario, y monitorizar el estado del bien transportado.

Es dentro de este grupo que se encuentran las aplicaciones direccionadas a redes vehiculares, cuyo el objetivo es mejorar la eficiencia, seguridad y comodidad en sistemas de transporte a través del uso de nuevas tecnologías de información y comunicación, principalmente aquellas basadas en redes de sensores. Coches, trenes y buses por las carreteras equipados con sensores, actuadores y poder de cómputo pueden proporcionar información importantes al conductor y/o al pasajero a fin de permitir una navegación mejor y más segura, incluyendo información como el estado de la carretera o atascos en las vías, siendo las técnicas para evitar de colisiones un ejemplo típico en este entorno. La comunicación puede ser de vehículo a vehículo o de infraestructura a vehículo, constituyendo de esta manera uno de los más importantes componentes de un ITS (Sistema de Transporte Inteligente) conocido como VANET (Vehicular Ad hoc Network).

Las redes vehiculares tienen un papel fundamental en la evolución del Internet de las Cosas, estableciendo una asociación más ubicua entre el mundo digital y el mundo físico en nuestra sociedad, y donde la movilidad asume un papel más destacado.

Mapas turísticos equipados con etiquetas que permiten que los móviles equipados con lector NFC busquen y llamen automáticamente en base a información proveniente de servicios web relativa a hoteles, restaurantes,

puntos turísticos o cualquier área del interés del turista, son características de aplicaciones que también pertenecen a este grupo y son conocidas como Mapas Aumentados.

2.2.3.2 Aplicaciones de cuidados de la salud

En el dominio de la salud muchas son las aplicaciones que pueden ser creadas con el Internet de las Cosas, permitiendo sacar muchos beneficios de este paradigma para esta área.

Rastreo de objetos y personas, identificación y autenticación de personas, colecta de datos automática y detección son los grupos de aplicaciones destacadas en este entorno.

El rastreo es una función centrada en la identificación de personas u objetos en movimiento que incluye rastreo en tiempo real, en los casos de monitorización del flujo de pacientes para mejorar a carga de trabajo en los hospitales, y rastreo de movimientos en puntos cuellos de botella para mejorar el acceso en determinadas áreas. Para el caso de bienes la técnica de rastreo es utilizada más comúnmente para mantenimiento, disponibilidad de estos bienes en el inventario cuando necesarios, y monitorización de su uso.

Identificación y autenticación es una técnica aplicada para reducir incidentes prejudiciales a los pacientes como: medicamentos, dopajes, tiempos o procedimientos equivocados. Con relación al personal, esta técnica es utilizada con fines de controlar el acceso y respaldar la moral de los empleados cuando abordan el tema de la seguridad de los pacientes. En relación a bienes materiales, esta técnica es utilizada para hacer cumplirse los requerimientos de procedimientos de seguridad, evitando robos o pérdidas de importantes instrumentos y productos.

La recolección y transferencia automática de datos se dirige sobre todo a reducir el tiempo de procesamiento de formularios, la automatización de procesos, cuidados automatizados, procedimiento de auditoria y gestión del inventario médico.

La posibilidad de recolectar información relacionada con la salud a través de dispositivos sensores conectados al paciente en casa ya es una realidad, ya no hace falta por ejemplo ir a farmacia “medir la presión”, ya que las

informaciones son recolectadas y compartidas mediante Internet, permitiendo un almacenamiento y análisis posterior. Los dominios de aplicaciones incluyen diferentes soluciones de telemedicina, vigilar el cumplimiento del paciente con sus recetas para cada medicación, y generación de alertas para el bienestar del paciente.

El principal objetivo de la aplicación del Internet de las Cosas en este entorno es simplificar la manera mediante la cual la información es disponible, y aumentar la velocidad con la cual ella puede ser utilizada en pro de la salud del paciente.

2.2.3.3 Aplicaciones para entornos inteligentes

Ambientes inteligentes son aquellos que dan suporte a las personas que ahí habitan por medio de dispositivos que hacen parte de una rama de la computación llamada computación ubicua o pervasiva. Tales entornos son discretos, interconectados, adaptables, dinámicos, integrados e inteligentes. Aplicaciones para esto tipo de entorno son fácilmente encontradas en las "casas inteligentes", por ejemplo donde sensores y actuadores son distribuidos por la casa a fin de hacer la vida del habitante más confortable en muchos aspectos, como por ejemplo el calentamiento de las habitaciones puede ajustarse según las preferencias de los residentes o de acuerdo con el clima; la iluminación puede cambiarse automáticamente de acuerdo con el hora del día; se puede ahorrar energía al apagar las luces sí no es detectada una persona en la habitación. Esta rama de aplicaciones para casas hace parte de un dominio específico para esto tipo de entorno conocido como domótica.

El uso de tecnologías asociadas al Internet de las Cosas en la fabricación y sus procesos, conocida como IIoT (Industrial Internet of Things), demanda la integración de grande parte de los conceptos y tecnologías inherentes al Internet de las Cosas, siendo el sector industrial considerado como el grande beneficiado por ésta. Entre las actividades demandadas en este entorno están la detección y actuación, el cómputo en nube, y el Big Data con análisis de datos automática y sistemática. Sin embargo, nuevos mecanismos de seguridad, privacidad e interfaz hombre-máquina son necesarios. Como ejemplo de aplicaciones para este entorno están el de suporte a las instrucciones de manufactura, en que diversos dispositivos

son equipados con sensores de radiofrecuencia y actuadores para que, a la medida que las actividades son realizadas, el estado de la actividad como un todo es modificada, haciendo con que los demás sensores y actuadores indiquen las próximas etapas. Otro típico ejemplo son aplicaciones enfocadas al rastreo de componentes y productos dentro de las fábricas, viabilizando un mapeo completo de qué componente estará en la casa del cliente en el momento en que el adquiere un producto, como por ejemplo una SmartTV. Tal inventario dinámico permitirá una identificación en tiempo real de qué componentes generan más problemas dentro de cada dispositivo, lo que puede influenciar la cadena de proveedores y la logística de distribución de las industrias.

El uso de aplicaciones para entornos inteligentes también puede estar presente en escenarios como gimnasios y museos, por ejemplo, o sea, en cualquier entorno donde la computación es utilizada para mejorar las actividades comunes.

2.2.3.4 Aplicaciones personales y sociales

En este grupo se encuentran las aplicaciones que nos permiten las interacciones con otras personas para mantener y construir relaciones sociales. Aquí podremos encontrar aplicaciones enfocadas en actualizar automáticamente las informaciones sobre las actividades sociales de los usuarios de portales web tales como Facebook o Twitter, por ejemplo. En este caso deberemos imaginar RFIDs que generan eventos acerca de personas y sitios, que después de reunidos y subidos, fornecen a los usuarios actualizaciones en tiempo real en sitios web de redes sociales.

Otro ejemplo de aplicación muy útil para este entorno son las herramientas anti-pérdidas, o sea, herramientas que ayudan los usuarios a encontrar objetos que ellos no recuerdan donde dejaron. Esta es una simples aplicación RFID para web que consiste en un mecanismo de búsqueda para objetos previamente etiquetados que permite al usuario ver las ultimas ubicaciones guardadas para cada objeto etiquetado.

Por último, pero no menos importante, los tipos de aplicaciones similares a las anteriores, que son las aplicaciones centradas en hurtos o robos; en este tipo se encuadra la aplicación desarrollada para esta tesina. Para este entorno algunas maneras distintas de implementación pueden ser

adoptadas para alcanzar el propósito final, que es permitir al usuario que sepas si algún objeto suyo está siendo movido de un área restricta, lo que indica que el objeto estará siendo hurtado. En este caso el evento habrá que ser notificado prontamente al propietario o a algún tipo de equipo de seguridad, por medio de SMS por ejemplo. Otra manera de alcanzar tal objetivo es en caso de robo en que el propietario toma conocimiento de la pérdida del objeto, pero necesita hacer con que las autoridades de seguridad sepan del ocurrido. Para este segundo caso el usuario podría acceder por Internet a su dispositivo embebido en este objeto y cambiar el estado de seguridad del mismo. A partir de ahí los actuadores harían con que algún tipo de mensaje fuera emitido a las autoridades de competentes.

Este grupo de aplicaciones puede ser muy útil en sitios donde la seguridad es un problema, principalmente en sitios donde prácticas de hurto y robos ocupan grande parte de las estadísticas de crímenes.

2.2.3.5 Aplicaciones futurísticas

Aquí se encuentran todas las aplicaciones que permitirán el crecimiento del Internet de las cosas, aplicaciones en que nuestras tecnologías disponibles son incapaces de cumplir los requisitos de implementación de este tipo de proyectos.

Estos tipos de aplicaciones, al necesitar de tecnología que aún está siendo estudiada o que aún está por venir, se hace más interesante en términos de investigación por su potencial impacto. Un análisis interesante de estos tipos de aplicaciones se hace en el proyecto SENSEI FP7 [8] donde hemos sacados dos ejemplos más atractivos.

El primero ejemplo son los taxis robots, pues se cree que, en las ciudades del futuro, los taxis robots se moverán por toda la ciudad como enjambres, proveyendo el servicio donde solicitado de manera eficiente y oportuna. Los taxis robots responderían en tiempo real a los movimientos de tráfico de la ciudad y serian calibrados para disminuir congestión y cuellos de botella por la ciudad, pudiendo moverse por las carreteras con o sin la presencia de un conductor humano a velocidades optimizadas y evitando accidentes con ayuda de sensores de proximidad que les repelen magnéticamente de otros objetos en la carretera. La ubicación del usuario

es rastreada automáticamente por GPS, y les es permitido solicitar un taxi en una determinada ubicación en determinado horario simplemente seleccionando un punto específico del mapa.

El segundo ejemplo para los amantes de videojuegos es una sala de juegos mejorada en la cual, al igual que los jugadores, está equipada con una variedad de dispositivos para captar localización, movimiento, humedad, aceleración, temperatura, ruido, voz, informaciones visuales, frecuencia cardiaca y presión sanguínea. La sala utiliza estos datos para medir los niveles de energía y excitación del jugador, y para controlar las actividades del juego a la medida en que estos datos cambian. Varios objetos son entonces distribuidos por la sala y el objetivo del juego es arrastrarse y saltar de un punto a otro sin tocar el piso hasta llegar a un punto posicionado en una pantalla en la pared, donde quien llegar primero vence. Su controlador reconoce etiquetas RFID en objetos de la habitación, y para puntuar el jugador debe tocar los objetos con esta etiqueta. El juego adapta los niveles de dificultad de acuerdo con las conquistas de los jugadores, manteniendo así altos niveles de excitación detectados por los sensores.

Vemos también que el concepto de computación usable se expandió y se incorporó al Internet de las Cosas, principalmente por la ubicuidad de la conexión con el Internet. Dentro de este contexto podemos poner como ejemplo las lentes de contacto inteligentes, que permitirán medir en tiempo real la glucosa y la presión intraocular. Hay muchos desafíos por delante, pero al que las investigaciones indican que, en el futuro, tendremos una integración muy íntima entre los wearables e nuestros cuerpos.

Estas aplicaciones sirven de ejemplo a lo que ha sido destacado por algunos autores: “el IoT debe ser considerado como parte del Internet global del futuro en que probablemente sea radicalmente diferente del Internet que utilizamos hoy en día”(Atzori, Iera, Morabito, 2010).

3 DETALLES DE DESARROLLO

En esta sección se describirá todo el entorno en el que se desarrolló la aplicación presentada en esta tesis. La primera parte describirá el hardware utilizado y sus detalles relevantes en la construcción de la solución propuesta.

La segunda parte de este capítulo describirán las herramientas de desarrollo relativas al software del sistema tales como librerías, lenguaje de programación adoptada, y frameworks, entre otros.

3.1 HARDWARE UTILIZADO

Se hace necesaria la descripción del hardware utilizado en este proyecto para una total comprensión de su construcción y funcionamiento. El equipo utilizado en este proyecto seguramente no es el más adecuado para estos fines, ya que uno de los objetivos de esta tesis es hacer que el sistema logre cumplir con lo propuesto, utilizando recursos de hardware didácticos, y por lo tanto genéricos, que podrían hacer el papel de un producto manufacturado creado estrictamente para estos fines.

3.1.1 Raspberry Pi

Raspberry Pi es un ordenador de bajo coste, del tamaño de una tarjeta de crédito, que se conecta en un monitor de computador o de TV, utiliza un teclado standard y un ratón. Es capaz de realizar todo lo que se espera que un ordenador de mesa haga, desde navegar en el Internet y reproducir video de alta definición, hasta crear hojas de cálculo, procesamiento de textos y jugar videojuegos [9].

Figura 4 - Raspberry Pi 2

Fuente: Site Raspberry Pi Foundation[1].

El dispositivo exhibido en la figura 4 es una Raspberry Pi 2 modelo B, idéntica a la que se utiliza en este Trabajo de Fin de Máster, y sus principales detalles técnicos se describen a seguir:

- Procesador ARM Cortex-A7 quad-core 900MHz
- 1GB LPDDR2 SDRAM (2x memorias)
- 4 puertos USB 2.0 y 1 puerto micro USB para alimentación
- 1 puerto Ethernet (IEEE 802.3)
- 1 lector de tarjetas MicroSD
- Salida de Vídeo y Áudio – HDMI y AV 3.5mm Jack
- Conector para GPIOs con 2x20 pines
- Conector para Cámara
- Conector para Pantalla

La Raspberry Pi 2 tiene un peso aproximado de 45g (sin carcasa) y dimensiones de 85,60x56,5 mm, siendo capaz de ejecutar versiones adaptadas de sistemas operativos de escritorio como por ejemplo Raspbian y Windows 10 IoT Core, entre otros. Estas características, sumadas a las anteriores ya descritas, hacen de este dispositivo una excelente plataforma para desarrollo de IoT.

[1]Disponible en: https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/ Acceso en Julio de 2016.

3.1.2 GPS/GLONASS U-blox7

Este es un módulo receptor de señales procedentes del GNSS (Sistema Global Navegación por Satélite), que engloba los señales emitidos a través de GPS (Global Positioning System), GLONASS (la versión rusa del sistema de navegación global por satélite), SBASS (Satellite Based Augmentation System) y QZSS (Quasi-Zenith Satellite System).

Figura 5 - Receptor GPS/Glonass USB

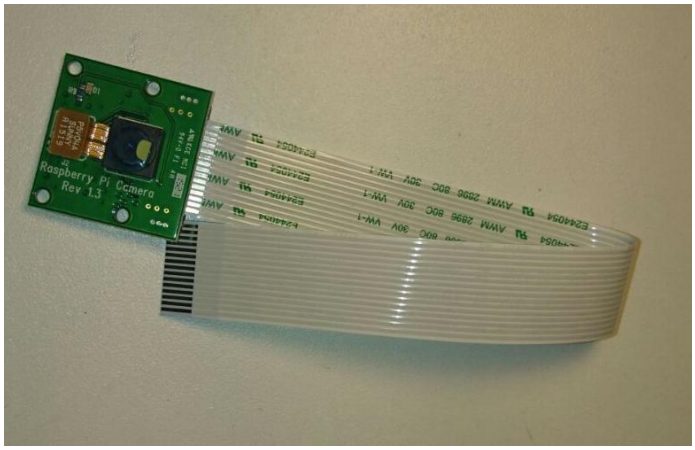
Fuente: Imagen obtenida por el autor.

Para los casos en que los principales requisitos incluyan bajo coste y bajo consumo, u-blox7 es la respuesta perfecta. Este dispositivo es optimizado para bajo consumo y para aplicaciones de baja gama donde una única recepción GNSS es suficiente. Por ser uno de los receptores GNSS múltiple de más bajo consumo en el mercado, esta fue la mejor elección para crear el prototipo del proyecto.

3.1.3 Módulo de cámara para Raspberry Pi

Una vez que se ha propuesto en este Trabajo de Fin de Máster la captura de imágenes desde el interior del coche, se ha elegido, de entre las diversas maneras de hacerlo, la manera quizás más directa y sencilla, que es conectando a través del conector CSI un módulo de cámara proprio para Raspberry Pi.

Figura 6 - Módulo de cámara para Raspberry Pi



Fuente: Imagen obtenida por el autor.

Originalmente diseñada y producida por la Raspberry Pi Foundation en Reino Unido, este módulo cuenta con una cámara de 5MP de resolución de imagen, o grabación de vídeo en alta definición con resolución de 1080p a 30fps, y la tarjeta donde se encuentra la cámara tiene por dimensiones 25mm x 20mm x 9mm, pesando poco más de 3 gramos.

3.2 SOFTWARE ADOPTADO

Así como en la parte de hardware, los softwares elegidos para realización de este proyecto son aquellos utilizados en clase por diferentes asignaturas durante el Máster en Ingeniería de Computadores y Redes de la Universidad Politécnica de Valencia (UPV). Las próximas secciones describirán básicamente el software utilizado y su papel dentro de este proyecto.

3.2.1 JAVA

El lenguaje de programación Java fue desarrollado por James Gosling, juntamente con otros colaboradores, al inicio de la década de 1990, en la empresa Sun Microsystems. Este lenguaje orientado a objetos fue proyectado para enfrentar los retos de desarrollo de aplicaciones en el contexto de los entornos heterogéneos distribuidos por toda la red [10].

Actualmente este lenguaje de programación al que pertenece a Oracle es el más utilizado del mundo, y está presente en la gran mayoría de dispositivos móviles como smartphones, tablets y otros dispositivos inteligentes. Sin embargo, también es muy utilizada para desarrollo web. Así que, sin duda, su gran popularidad fue punto muy importante en la elección de este lenguaje en este proyecto ya que, por su difusión en el mundo, una gran cantidad de librerías y frameworks para Java están disponibles, lo que hace más rápido y seguro el desarrollo del proyecto.

Otra característica muy importante de Java, y que sin duda también fue un punto importante para su elección en este Trabajo de Fin de Máster, es su alta compatibilidad o mejor, su independencia de sistema operativo, ya que como Java utiliza el concepto de máquina virtual, que funciona como una capa extra entre el sistema operativo y la aplicación, siendo responsable por la traducción de las tareas o llamadas de sistema operativo donde está ejecutándose, no hace falta preocuparse con el sistema operativo específico en que la aplicación se ejecuta.

3.2.2 JavaScript

JavaScript es un lenguaje de programación basado en scripts y estandarizada por el ECMA International (asociación especializada en la estandarización de sistemas de información) y hace parte del conjunto de normas que definen la tecnología del contenido World Wide Web [11]. Se trata de un lenguaje de scripts de páginas Web que se ejecuta del lado del cliente de la aplicación, y es dinámica, orientada a objetos, y creada con una sintaxis parecida a del lenguaje C.

3.2.3 JavaServer Faces

Uno de los frameworks más utilizados hoy en día es el JavaServer Faces (JSF). Su grande atractivo es ser el framework oficial de Java EE para Web, mientras que los demás pertenecen a terceros; la decisión de utilizarlo fue debido nuestra experiencia con esta tecnología en anteriores proyectos.

JSF es un modelo basado en componentes para desarrollo Web que ayuda en la creación de aplicaciones del mundo real, y también a crear su propia interface de usuario en el lado del servidor. La principal idea de un framework de componentes es abstraer muchos de los conceptos de la Web y del protocolo HTTP, proporcionando un entorno de programación similar a la programación para Desktop. JSF es una tecnología designada para simplificar la creación de aplicaciones Java para Web haciéndolos funcionar de manera similar a las típicas aplicaciones de interface gráfica de usuario basadas en eventos [12].

3.2.4 Primefaces

Primefaces es un popular framework de interface de usuario para JSF que puede ser utilizado para agilizar el desarrollo de aplicaciones sofisticadas para empresas o para sitios web estándares.

Las librerías de componentes para JSF, en este caso Primefaces, posibilitan que las aplicaciones web tengan una apariencia moderna y también una mejor usabilidad, permitiendo así la misma experiencia y practicidad de aplicaciones de escritorio (desktop). Sin embargo, muchas librerías para JSF tales como Primefaces utilizan inúmeras características y estándares HTML5 en sus diversos componentes, lo que es muy importante porque deja la implementación con JSF en concordancia con los más recientes estándares de desarrollo.

Entre las ventajas consideradas para la elección del Primefaces para este Trabajo de Fin de Máster están la gran cantidad de componentes (hablamos de más de una centena), sencilla instalación y uso, componentes adicionales para desarrollo móvil, integración con JQuery y HTML5. Sin embargo, tiene una documentación muy clara y con muchos ejemplos de fácil uso.

3.2.5 Google Maps JavaScript API

Como la ubicación del dispositivo es un requisito funcional en esta aplicación, se hizo necesario el uso de una API de localización geográfica para interpretar los dados obtenidos a través del módulo GPS.

La API Google Maps JavaScript ha sido proyectada para cargar rápidamente y funcionar bien en dispositivos móviles, su código es directamente implementado en la capa de visión de la aplicación, ya que se trata de JavaScript, haciendo así el desarrollo más sencillo y ágil.

Esta API es fornecida con un conjunto de controles incorporados para uso en mapas que pueden ser modificados por medio de código en un campo específico de opciones del mapa en JavaScript, tales controles son:

- El control Zoom – muestra los botones “+” y “-” para cambiar el nivel de zoom del mapa.
- El control Map Type – permite al usuario elegir entre los diferentes tipos de mapas.
- El control Street View – Tiene un icono que puede ser arrastrado para activar el modo Street View
- El control Rotate – Ofrece una combinación de rotación y inclinación para mapas con imágenes oblicuas.
- Control Scale – Un elemento de escala del mapa.

Actualmente la gran mayoría de navegadores web, tanto en móviles como desktop, dan suporte a geolocalización.

3.2.6 Java Marine API

Como nuestra aplicación recibe datos que son obtenidos por medio de un módulo receptor de señales GPS y que posteriormente llegan a uno de los puertos USB donde estará conectado, sea en el ordenador (en la etapa de implementación) o en la Raspberry Pi (en la fase de pruebas), ellos llegan en formato texto (para dispositivos en el estándar NMEA), y son tratados por medio de una API para hacer el trabajo más sencillo y practico.

Java Marine API es una librería analizadora del estándar NMEA 0183 para JAVA cuyo objetivo es facilitar el acceso a los datos fornecidos por dispositivos GPS, por ejemplo. Los datos ASCll son convertidos al modelo de

eventos con interfaces y implementaciones para los mensajes seleccionados. Esta librería ofrece también suporte a la salida de datos permitiendo modificación del contenido de frase con formatación y validación de datos. Entre los mensajes, uno que nos será muy útil será el GLL (Global Latitude y Longitude), que nos retornará la latitud, longitud y hora acuerdo con la UTC.

3.2.7 RabbitMQ

Este será nuestro gestor de cola de mensajes (message broker). En otras palabras, será el programa que hará de intermediario de la comunicación entre las aplicaciones. Un gestor de cola de mensajes puede ser visto como un conjunto de colas donde los mensajes son almacenados en estas colas y son enviados de acuerdo con la orden de llegada (FIFO).

El RabbitMQ es uno de los muchos tipos de message broker existentes, y es una implementación de código abierto de servicios de comunicación basada en el protocolo AMQP (Advanced Message Queueing Protocol), que define un conjunto de normas y especificaciones para interoperabilidad de servicios de mensajería. Esto, así como muchos de los softwares descritos anteriormente, ha sido utilizado en actividades de clase en una asignatura del Máster Universitario en Ingeniería de Computadores y Redes de la UPV, siendo este el motivo de su elección para este proyecto.

3.2.8 Hibernate

Hibernate es una de las soluciones de mapeo objeto-relacional más populares en el mundo de persistencia en Java. Él proporciona una solución para mapeo de tablas de base de datos para clases Java. Hibernate copia los datos de la base para una clase, sin embargo, permite que las clases sean guardadas en la base de datos, y en este proceso los objetos son transformados en una o más tablas.

Este framework ofrece muchas funcionalidades que justifican su empleo en cualquier proyecto escalable, tales como control transaccional, identificador de objeto, mapeo objeto-relacional, cache, consultas por demanda, proxy y otras más.

Una de las grandes ventajas de utilizar Hibernate es que el desarrollo de la aplicación se hace más rápido, por no tener que escribir códigos SQL por las clases Java, aún que este framework también permita, pues la idea de Hibernate es encapsular todo el código SQL JDBC, quedando transparente al desarrollador, y esta es la razón por la cual decidimos utilizarlo en este proyecto.

3.2.9 MySQL

El MySQL es el sistema gestor de base de datos que utilizaremos en nuestro sistema de vigilancia embebido. Decidimos optar por una base de datos para mantener nuestros datos ordenados y siempre listos para consulta. MySQL una de las bases de datos relacionales más importantes, siendo además gratuita y tiene una instalación sencilla para todos los sistemas operativos.

3.2.10 Apache Tomcat

El Apache Tomcat es la referencia oficial en la implementación de las especificaciones para servlets Java y JavaServer Page (JSP). Sin embargo, es el contenedor de aplicaciones Java para Web más utilizado del mundo [13].

Tomcat es adecuado para las aplicaciones Java de la vida real, que son típicamente más complejas, orientadas a objetos y hacen un uso extensivo de las librerías Java, como es el caso de las aplicaciones científicas, además de tener altas tasas de lectura y escritura, haciendo un bueno uso de grandes cantidades de hilos.

3.2.11 WampServer

WampServer es una aplicación que instala un entorno de desarrollo Web en Windows, y que nos permite crear aplicaciones Web con Apache2, PHP y base de datos MySQL. Sin embargo, es posible gestionar fácilmente las bases de datos a través de la herramienta PhpMyAdmin que se incluye en su paquete de instalación.

Este programa nos instalará todo que necesitamos para empezar nuestro desarrollo Web de manera automática, y su uso es muy intuitivo.

4 IMPLEMENTACIÓN

En este capítulo se describirán las etapas de desarrollo del sistema embebido para vigilancia de vehículos, su entorno de pruebas y las tecnologías aplicadas.

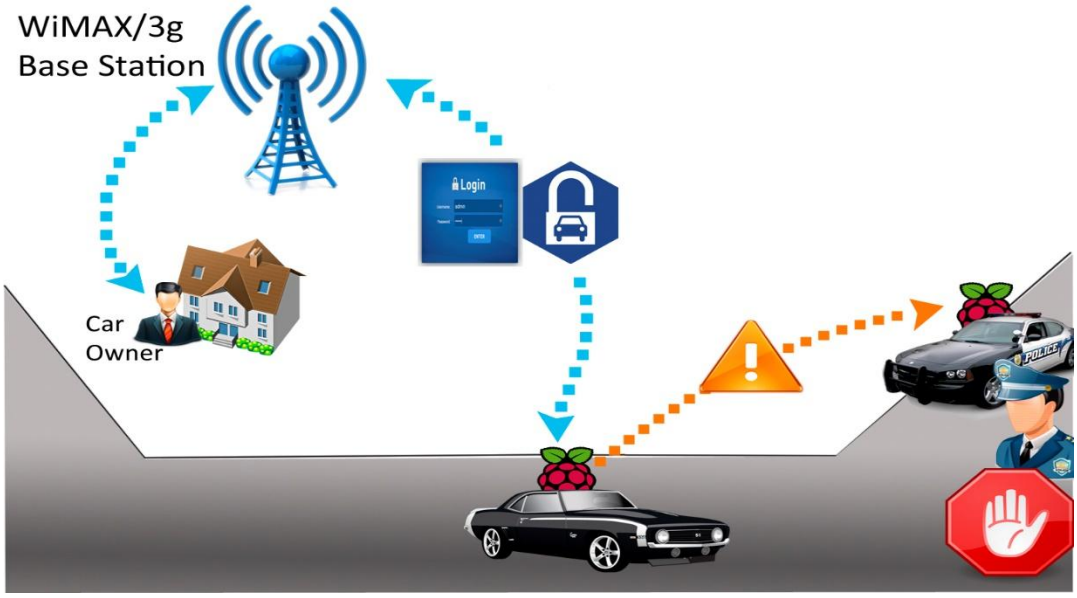
Una vez descritas las tecnologías hardware y software utilizadas, se introducirán las descripciones del proceso de implementación del proyecto, aunque antes de eso es necesario comprender como está diseñado este proyecto, en otras palabras, lo que hemos planeado desarrollar.

4.1 DISEÑO DEL PROYECTO

El presente proyecto trata de un sistema embebido para vigilancia de vehículos que retornará datos de un determinado vehículo, en este caso el que lleva el dispositivo (Raspberry Pi) puesto. Sin embargo, ofrecerá funcionalidades simples para el usuario como el rastreo del coche, sacar foto del interior del coche y cambiar su estado de seguridad, funcionalidades que serán explicadas con más detalle posteriormente. La figura 7 ilustra la idea general de este sistema.

La imagen nos muestra cómo debe funcionar nuestro sistema en un caso de robo de vehículo. Primeramente, el propietario del coche, tras la pérdida de su vehículo, debe conectarse a Internet, por medio de su móvil, tableta u ordenador, y acceder a la aplicación web que está ejecutándose en la Raspberry Pi instalada en el coche. Una vez conectado, la aplicación solicita la autenticación del usuario mediante un login, el cual puede basarse en su cuenta Google, para entonces exhibir la pantalla principal del sistema, que permitirá al usuario acceder los datos del vehículo, previamente introducidos en la base de datos, seguir la ubicación del vehículo y cambiar el status de seguridad del mismo, lo que accionará otra funcionalidad de este sistema, que es el envío de un mensaje que contiene datos del coche y un alerta de peligro a todos los dispositivos que están suscritos al canal de publicación.

Figura 7 - Esquema del sistema de vigilancia



Fuente: Imagen creada por el autor.

Este sistema tornará más eficaz el trabajo de la policía en la detección de coches sospechosos permitiendo, de esta manera, que los propietarios que han tenido sus coches robados puedan recuperarlos.

Una vez descrita la misión de este sistema, explicaremos como logramos alcanzar cada etapa de su desarrollo en las próximas secciones.

4.2 PREPARANDO EL ENTORNO

Antes de seguir con la implementación propiamente dicha optamos por preparar el entorno de ejecución de la aplicación, en este caso estamos la plataforma Raspberry Pi, donde os principales componentes del sistema serán instalados, así como el ordenador donde el código fuente fue desarrollado y los demás componentes fueran probados.

Con la excepción del sistema operativo, todos los componentes de software instalados en la Raspberry Pi fueran anteriormente instalados en un ordenador portátil en su respectiva versión para cada plataforma. Eso significa que, en la etapa inicial se simuló en el ordenador tanto la parte cliente como la del servidor de este sistema de vigilancia. El ordenador en

este caso tiene instalado el sistema operativo Windows 10, procesador Intel Core i5 2.7GHz, y cuenta con 4GB de memoria RAM; más detalles acerca de este dispositivo son irrelevantes para el proyecto teniendo en cuenta que estos detalles no cambiarían las prestaciones o funcionalidades del sistema, ya que se trata de un componente opcional, que fue utilizado para agilizar el desarrollo, pues todo el proceso podría haberse realizado directamente en la Raspberry Pi.

4.2.1 Instalación del sistema operativo

Esta etapa ocurrió únicamente en la Raspberry Pi, pues el ordenador ya se encontraba con su sistema operativo debidamente instalado y configurado.

Para la realización de este proyecto elegimos por popularidad y por ser un sistema operativo oficial y basado en Debian, el sistema operativo Raspbian Jessie. Este sistema es optimizado para el hardware de la Raspberry Pi y cuenta con cerca de 35.000 paquetes de sistema optimizados para mejores prestaciones para esta plataforma.

La instalación de este sistema operativo en la Raspberry Pi se hizo a través de una tarjeta de memoria de 32GB previamente formateada para certificarse de que no había ningún contenido que pudiese causar errores durante la instalación. Una vez descargado el sistema operativo en el formato .iso directamente de su sitio web oficial[2] desde nuestro ordenador, seguimos con los pasos de instalación descritos en una página[3] disponible en el mismo sitio web donde hemos descargado el Raspbian Jessie.

Finalizada la instalación, insertamos la tarjeta en la Raspberry Pi y la encendemos para comprobar el correcto funcionamiento del sistema operativo.

[2] Disponible en: https://www.raspberrypi.org/downloads/raspbian/

[3] Dispobible en: https://www.raspberrypi.org/documentation/installation/installing-images/windows.md

4.2.2 Instalación y configuración de la base de datos

Tal y como se describió en el diseño del proyecto, nuestro sistema necesita hacer consultas a una base de datos para recoger informaciones del coche, propietario y etc.

Para el almacén de estos datos decidimos utilizar un sistema gestor de base de datos, para ofrecer algo de robustez y escalabilidad a nuestra aplicación si lo comparamos con otro medio para almacenamiento de datos, como podría ser un ficheros de texto directamente.

El MySql fue instalado tanto en el entorno de simulación (ordenador portátil) como en la plataforma final del proyecto, en este caso la Raspberry Pi. En el ordenador el MySql se ha instalado a través del WampServer para proporcionar un entorno de desarrollo web y facilitar la gestión de la base de datos.

En la Raspberry Pi el Mysql fue instalado directamente por por terminal del sistema operativo a través del comando:

```
sudo apt-get install mysql-server --fix-missing
```

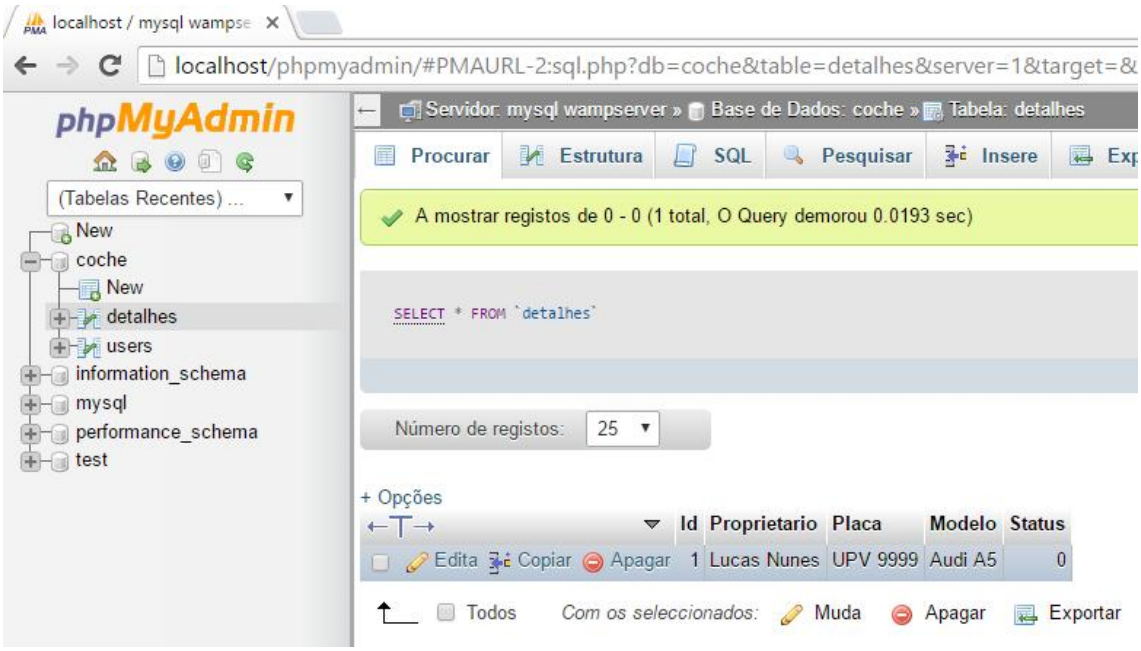
para la parte del servidor, donde durante la instalación hemos configurado una contraseña de administrador y el comando:

```
sudo apt-get install mysql-client
```

que nos va permitir instalar el cliente mysql para conectarnos a nuestra base de datos local.

Con el Mysql instalado en ambos dispositivos, hemos creado una base de datos en el ordenador, y poblado con datos ficticios que serían referentes al coche y su propietario, como nos muestra la figura 8.

Figura 8 - Base de datos poblada



Fuente: Datos creados por el autor.

La tabla detalles tiene cinco columnas, que en un primer momento serían suficientes para un entendimiento completo del funcionamiento de este sistema. Tales columnas son:

- Id – Identificador único de línea de registro (clave externa).
- Proprietario – El supuesto nombre del propietario del vehículo.
- Placa – La matrícula del coche
- Modelo – El modelo del coche
- Status – El estado de seguridad del vehículo: 0 para seguro; 1 para peligro.

Los datos necesarios en la base de datos para las primeras pruebas en nuestra aplicación ya están listos en esta etapa, y posteriormente lo que hicimos fue importar el esquema propuesto, incluyendo los datos, a servidor Mysql instalado dentro de la Raspberry Pi, copiando y pegando el fichero de base de datos, coche.db en nuestro caso, en el destino, y ejecutando el siguiente comando en la consola Mysql:

```
source /home/coche.db
```

4.2.3 Instalación y configuración del Apache Tomcat

La instalación de nuestro contenedor web fue lo más sencillo posible, simplemente extrayendo el software en la carpeta destino donde se desea ejecutar el servicio, ya que el Tomcat también puede ser instalado a través de un ejecutable, instalando sus componentes automáticamente tras la entrada de un comando por consola. Una vez que ya teníamos un Tomcat debidamente configurado y funcionando en el ordenador, todo lo que hicimos fue copiar el directorio desde el ordenador hacia un directorio que hemos elegido dentro de la Raspberry Pi.

Los ficheros de configuración de este contenedor son distintos dependiendo del sistema operativo, siendo los de extensión .bat para el sistema operativo Windows y los ficheros del tipo .sh para distribuciones basadas en el kernel Linux, que es el caso del Raspbian. Hay ficheros de configuraciones globales, independientes del sistema operativo, que en este caso son algunos ficheros xml donde definimos el puerto de conexión, usuario, contraseña y permisos de acceso; tales configuraciones pueden ser observadas en las figuras 9 y 10.

Figura 9 - Fichero tomcat-users.xml

```
<tomcat-users>
    <!--
  NOTE:  By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary.
-->
    <!--
  NOTE:  The sample user and role entries below are wrapped in a comment
  and thus are ignored when reading this file. Do not forget to remove
  <!.. ..> that surrounds them.
-->
    <!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
    <user password="12345" roles="manager-script,admin" username="lucas"/>
</tomcat-users>
```

Fuente: Print Screen del fichero de configuración del tomcat.

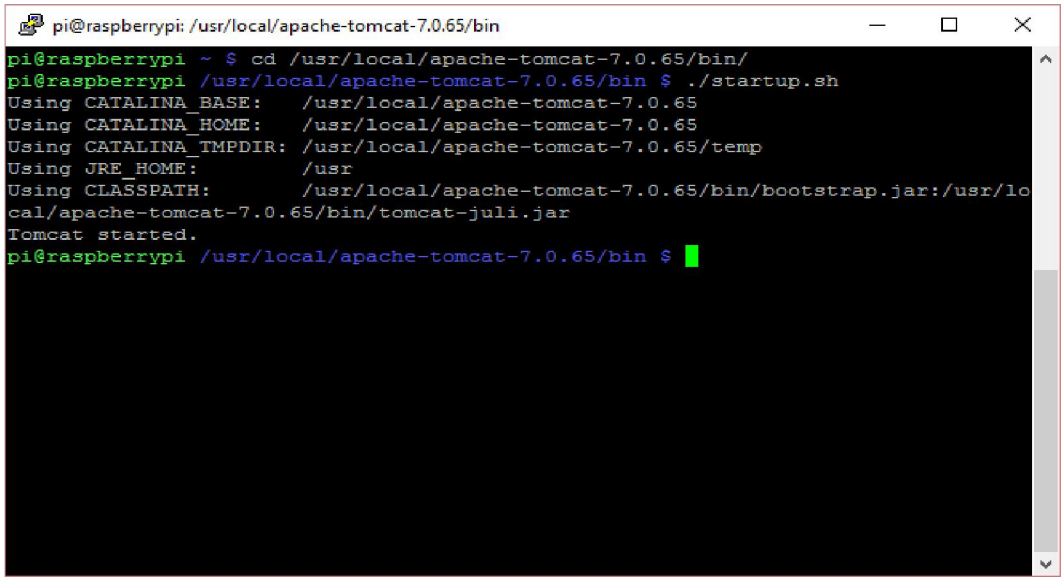
Figura 10 - Fichero server.xml

```
    Documentation at /docs/config/service.html
  -->
  <Service name="Catalina">

    <!--The connectors can use a shared executor, you can define one or more n
    <!--
    <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="150" minSpareThreads="4"/>
    -->


    <!-- A "Connector" represents an endpoint by which requests are received
         and responses are returned. Documentation at :
         Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
         Java AJP  Connector: /docs/config/ajp.html
         APR (HTTP/AJP) Connector: /docs/apr.html
         Define a non-SSL HTTP/1.1 Connector on port 8080
    -->
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <!-- A "Connector" using the shared thread pool-->
    <!--
```

Fuente: Print Screen del fichero de configuración del tomcat.

Estos ficheros se encuentran dentro de la carpeta ‘conf’ del Apache Tomcat 7.0.65, y nuestro contenedor se ha ubicado en la Raspberry Pi dentro del directorio /usr/local/apache-tomcat-7.0.65. Para probar si el Tomcat está funcionando ejecutamos el fichero startup.sh dentro de la carpeta bin, como muestra la figura 11. Si todo ocurrió bien, se mostrará un mensaje de que el Tomcat se está ejecutando.

Figura 11 - Arranque del Tomcat



Fuente: Creado por el autor.

Hasta este punto tenemos la Raspberry Pi con el sistema operativo instalado, el sistema gestor de base de datos Mysql igualmente instalado, y el contenedor de aplicaciones web funcionando. De esta manera nuestra plataforma está lista para recibir la aplicación que fue desarrollada, la cual será descrita con más detalle en las próximas secciones.

4.2.4 Creación de la aplicación Web

Toda la aplicación web fue desarrollada en el ordenador portátil para tornar más sencillo y ágil esta etapa, para posteriormente ser solamente migrada hacia nuestra plataforma IoT.

La aplicación concentra gran parte de las tecnologías descritas en el apartado tres, y su descripción enfocará los aspectos más relevantes de esta etapa, explicándose en el transcurrir de las secciones el método empleado para cada tecnología.

4.2.4.1 Creación del proyecto del tipo Aplicación Web

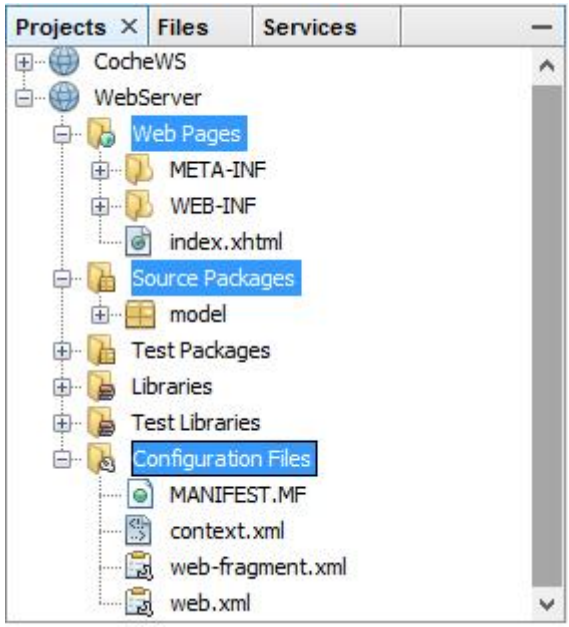
Para la creación de nuestro sistema hemos optado por hacer uso de una IDE (Integrated Development Environment) para hacer nuestro trabajo más ágil y práctico. En este contexto hemos elegido el NetBeans IDE por ser cómodo y tener una interface limpia, facilitando la creación de aplicaciones Web basadas en Java EE con JSF.

En la pantalla de creación del nuevo proyecto serán presentadas, además del nombre del proyecto y ubicación, las opciones del servidor de aplicaciones Web que deseamos utilizar, que en nuestro caso es el Apache Tomcat 7.0.65, Java en su versión EE, y una sección con opciones relativas a los frameworks que deseamos utilizar. En esta última parte un listado con los Frameworks disponibles será exhibido en un cuadro para que posamos marcar aquellos que deseamos utilizar, y en otro cuadro las opciones relativas a cada framework seleccionado y sus respectivos componentes.

Para nuestra aplicación hemos seleccionado como frameworks el Hibernate y el JSF 2.2 disponibles en las librerías nativas de la IDE; para este último framework hemos elegido como componente el PrimeFaces, cuyas librerías también ya vienen junto con el NetBeans.

Después de creado, nuestro proyecto quedará con una estructura tal y como se muestra en la figura 12, donde hemos destacado las principales capas que hacen parte de su estructura.

Figura 12 - Estructura de la aplicación web



Fuente: Print Screen de la IDE Netbeans.

El JSF incorpora características de un framework MVC (Model View Controller) para la Web, y de un modelo de interfaces gráficas basadas en eventos. La Idea del MVC es dividir la aplicación en tres capas: modelo, vista y controlador. En el JSF, el controlador es compuesto por un servlet denominado FacesServlet, ficheros de configuración, y un conjunto de manipuladores de acciones y observadores de eventos. Sin embargo, también es responsable por recibir peticiones Web, redirigirlas para el controlador, y entonces enviar una respuesta.

Los ficheros de configuración son responsables por realizar asociaciones, mapeos de acciones, y definiciones de reglas de navegación. Los manipuladores de eventos son responsables por recibir datos procedentes de la capa de vista, acceder al modelo, y entonces devolverlos al FacesServlet.

Una vez creado el proyecto del tipo aplicación web en el Netbeans, se ha empezado a desarrollar el código Java para construir nuestro sistema de vigilancia.

4.2.4.2 La clase Detalhes

No hay un estándar para decir necesariamente por donde debemos empezar el desarrollo de aplicaciones web. Lo que sí hay son buenas practicas que siempre hay que tener en cuenta. Partiendo de este concepto hemos empezado por la clase *Detalhes*, la cual servirá de modelo a nuestro objeto principal, que contiene los siguientes atributos:

```
@Entity
@Table(name ="detalhes", schema = "coche")
public class Detalhes implements Serializable{

public Detalhes(){

}

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "Id")
private int id;

@Column(name = "Proprietario")
private String proprietario;

@Column(name = "Placa")
private String placa;

@Column(name = "Modelo")
private String modelo;

@Column(name = "Status")
private int status;
```

Tal y como podemos comprobar, la clase *Detalhes* tiene los mismos campos y atributos de la tabla en la base de datos que hemos presentado en la sección 4.2.2, y eso no por casualidad. Hay que tener en cuenta que esta clase está mapeada. Las líneas que empiezan con '@' son anotaciones, un recurso de Java que nos permite insertar metadatos relativos a nuestra clase, atributos y métodos. Estas anotaciones podrán ser leídas por

frameworks y librerías (*Hibernate* en nuestro caso) para que tomen decisiones basadas en estas configuraciones.

La anotación @Entity indica que los objetos de esta clase se almacenen en la base de datos. @Table nos permite definir el nombre de la tabla en la que deseamos mapear nuestra clase y el esquema. @Id indica que el atributo es clave primaria y @GeneratedValue hace con que la clave sea generada por la base de datos; en nuestro caso va a ser auto-incremental. La anotación @Column sirve para relacionar el atributo a una columna existente en la tabla.

4.2.4.3 La clase DaoDetalhes

Una vez definido el modelo, nuestro objeto está casi listo para ser manipulado en la base de datos. Esto será posible al final de la configuración de la clase *DaoDetalhes* y sus instancias, necesarias para manipulación de datos.

La clase *DaoDetalhes* representa las operaciones de manipulación de datos que deseamos hacer en nuestro sistema (grabar, leer, editar, remover), que pueden ser añadidas o insertadas según necesidad. Una de las ventajas que el *Hibernate* trajo a nuestra aplicación fue hacer los códigos más limpios y aparentemente más sencillos para algunas operaciones, como en el caso de la siguiente operación de editar registros en la base:

```
public Detalhes alterarDetalhe(Detalhes detalhe){
     try {
            detalhe = (Detalhes)this.dao.salva(detalhe);
            return detalhe;
          } catch (Exception ex) {
            Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Erro na Classe: 'srm.DaoDetalhes.alterarArea' !");
            return null;
             }
    }
```

En solamente una línea cambiamos los datos contenidos en este objeto, y en la línea siguiente el método lo devuelve. La clase *DaoDetalhes* crea una instancia de la clase *Dao*, que es la "raíz" de las operaciones de manipulación de datos, y corresponde al nivel más baio para el programador que está utilizando el framework *Hibernate*, ya que las operaciones ahí contenidas están encapsuladas por este framework, lo que hace con que nuestro trabajo sea más ágil. El código responsable por el cambio de los datos en este nivel es el siguiente:

```
public Object salva (Object dto) throws Exception {
                  try{
                           this.beginTransaction();
                           dto = this.session.merge(dto);
                           this.commit();
                           this.session.evict(dto);
                        this.session.flush();
                        this.session.clear();
                  } catch (Exception e) {
                           this.rollback();
                           throw new Exception(e.toString());
                  }
                  return dto;
         }
```

Todo lo que se encuentra por detrás de los comandos ahí presentados está encapsulado por la librería *HibernateUtil*. Todo el código de las clases anteriormente presentadas se encuentra en los anexos.

4.2.4.4 La clase DetalhesController

Con nuestra clase modelo ya definida, podemos avanzar ahora para la clase controladora donde creamos los métodos que representan las funcionalidades de nuestro sistema, que no son necesariamente relativas a persistencia.

Esta clase sería el corazón de nuestro sistema pues es aquí donde se determina su funcionamiento. Hemos visto como mapeamos la clase para que pueda ser grabada en la base de datos, como cambiamos un registro en la base de datos por el *Hibernate*, y ahora mostraremos la parte del

código de nuestro controlador que realiza la llamada al método que permite cambiar los datos y, aparte de esto, envía un mensaje vía AMQP utilizando la librería del RabbitMQ, como podemos observar en el siguiente código:

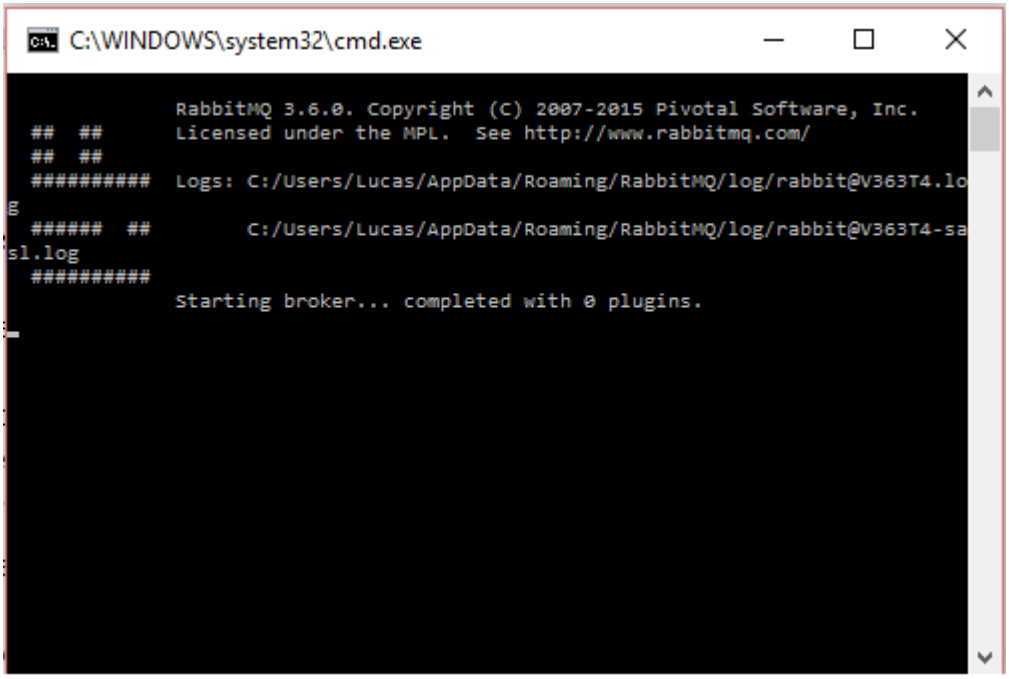
```
public void alterandoDetalhe() throws IOException{


      daoDetalhe = new DaoDetalhes();
      daoDetalhe.alterarDetalhe(this.detalhe);
      String status = "";
      if(this.detalhe.getStatus()==0){
          status = "SEGURO";
      }else{
          status = "PERIGO !!!";
      }
    ConnectionFactory factory = new ConnectionFactory();

    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE NAME, false, false, false, null);
    String message = "\n"+this.detalhe.getModelo()+" leva o status: "+status;
    channel.basicPublish("", QUEUE NAME, null, message.getBytes());
    System.out.println(" [x] Sent '" + message + "'");

    channel.close();
    connection.close();

    }
```

Tal y como podemos observar, nuestro método alterandoDetalhe() instancia nuestra clase *DaoDetalhe*, lo que nos permite utilizar los métodos contenidos en ella sin más problemas. En la segunda línea de nuestro código dentro del método invocamos al método alterarDetalhe(Detalhe) pasando como parámetro nuestro objeto instanciado en nuestro controlador, que estará sujeto a modificaciones de acuerdo a invocaciones de métodos modificadores de sus atributos, para ser actualizado en la base. Creamos en seguida un String que simboliza el status del vehículo, y que cambiará de acuerdo al valor de la variable status asignada en nuestro objeto *detalhe*.

A partir de este momento la parte de envío de mensaje entra en acción, donde empezamos por crear una nueva factoría de conexión, le pasamos el nombre del host que fue configurado en la parte que estará escuchando (que será posteriormente explicada) y toda la parte que es requerida por esta librería a la hora de mantener un sistema de mensajería en un determinado canal. La String message está compuesta por el atributo modelo del coche y su status, que fue indicado por el usuario. La publicación de este mensaje serializado será hecha en el canal recién establecido, y posteriormente será cerrado antes de la conexión.

Para que la parte de envío de mensaje en nuestro sistema funcione, antes hay que instalar el RabbitMQ y arrancar el gestor de cola de mensajes, como podemos ver en la figura 13:

Figura 13 - Arranque del gestor de cola de mensajes



Fuente: Print Screen del arranque de servicio en RabbitMQ.

Es importante tener en cuenta que la parte que recibirá los mensajes tendrá que tener instalada, y el servicio se tendrá que ejecutar, en nuestro caso la parte responsable por recibir los mensajes se trata del ordenador portátil, que recibirá los mensajes enviados por la Raspberry Pi.

Luego en seguida estará la parte del código de la clase Java Recv, la cual es responsable por la recepción de los mensajes ubicada en el dispositivo receptor:

```
private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");


    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

    QueueingConsumer consumer = new QueueingConsumer(channel);
    channel.basicConsume(QUEUE_NAME, true, consumer);

    while (true) {
      QueueingConsumer.Delivery delivery = consumer.nextDelivery();
      String message = new String(delivery.getBody());
      System.out.println(" [x] Received '" + message + "'");
    }
  }
```

En el emisor la configuración es muy similar: abrimos una conexión y un canal, y declaramos la cola a partir de la cual consumiremos los mensajes. Tenga en cuenta que aquí tendremos que declarar nuestra cola porque tendremos que iniciar el receptor antes del emisor.

Con estos pasos la parte de envío y recepción de mensajes ya está lista y, teniendo esto en cuenta, volveremos a la parte que encapsulará y mostrará estas funcionalidades de manera amigable al usuario.

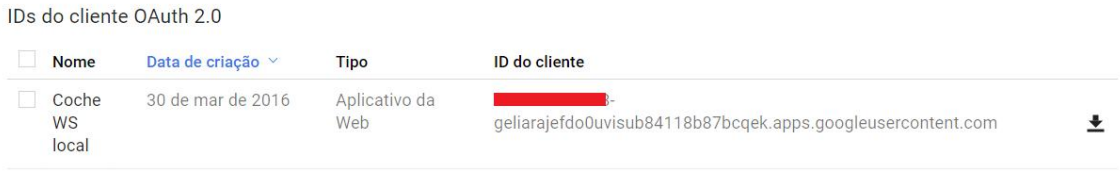
4.2.4.5 La capa de vista

En esta capa se encontrarán las principales funcionalidades de Primefaces empleadas en nuestro sistema. Sin embargo, otra funcionalidad

es desarrollada en esta etapa, la de login usando la cuenta del Google, como veremos a posteriormente.

Antes de avanzarnos con el desarrollo del código de la funcionalidad de login con cuenta Google, primero hay que crear un proyecto y un ID de cliente en la consola de desarrollo Google, tal y como hemos hecho en la figura 14:

Figura 14 - ID de cliente de API Google creada



Fuente: Print Screen de la consola de desarrollo web de Google.

Una vez creada la identificación de nuestra aplicación web, podemos seguir con el desarrollo del código que nos permitirá hacer el login mediante la plataforma Google:

```
<script src="https://apis.google.com/js/platform.js?onload=renderButton"
async="true" defer="true"></script>
    <ui:decorate template="./padrao.xhtml">
        <ui:define name="centro">


            <h:head>
                <meta name="google-signin-client_id" content="XXXXXXXX-
geliarajefdo0uvisub84118b87bcqek.apps.googleusercontent.com”/>
                <title>Smart Surveillance</title>
            </h:head>
```

Este es el fichero index.html, el primero destino del usuario cuando accede nuestro sistema. En esta primera parte de su código, que se encuentra dentro de la etiqueta ‘<script>’, estamos cargando la librería de la plataforma Google. En las dos siguientes etiquetas estamos apenas cargando los modelos que hemos creado para nuestra página.

En la etiqueta ‘<meta>’ es donde especificamos nuestra identificación de aplicación web que se muestra en la figura 14.

Siguiendo con la implementación de la autenticación utilizando la API del Google tendremos el siguiente código:

```
<div id="my-signin2"/>
<script type="text/javascript">
function renderButton() {
         gapi.signin2.render('my-signin2', {
         'scope': 'profile email',
         'width': 280,
         'height': 50,
         'longtitle': true,
         'theme': 'dark',
         'onsuccess': onSuccess,
         'onfailure': onFailure }); }
</script>
```

En esta parte del código insertamos el botón por medio de esta *div*, que está siendo insertada en la página, y a continuación cargamos un script que gestionará las configuraciones de estilo de este botón. En seguida veremos los scripts que se ejecutan basados en eventos que pueden pasar en el momento del login:

```
<script type="text/javascript">
function onSignIn(googleUser) {
 var profile = googleUser.getBasicProfile();
}
function onSuccess(googleUser) {
         var para, hiddenInput;
         para = document.getElementById('hidden1');
         hiddenInput = document.createElement('input');
         hiddenInput.type = 'hidden';
         hiddenInput.name = 'hidden';
hiddenInput.value = 'Usuario: ' + googleUser.getBasicProfile().getName();
                        para.appendChild(hiddenInput);
  window.location.href="http://localhost:8080/CocheWS/Administracion/admin2.upv"
seta();
                    }
                    function onFailure(error) {
```

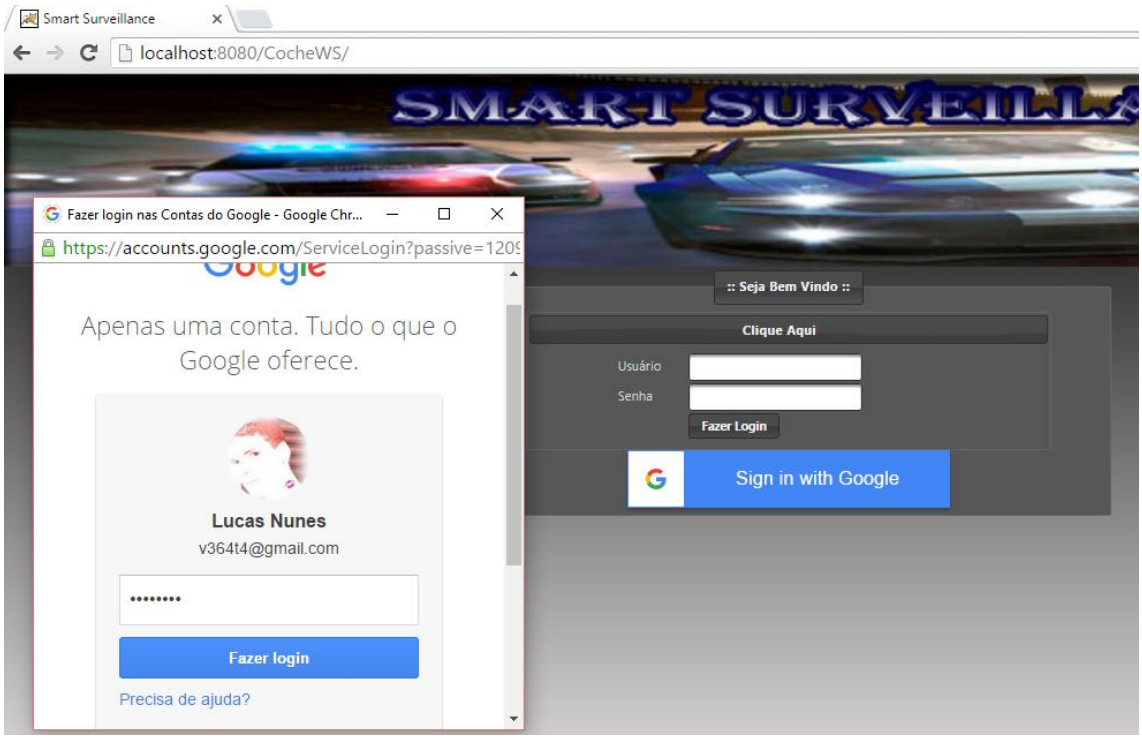
```
                    console.log(error); }
```

En esta parte del código el usuario se almacena en la variable ‘profile’ como aquel que acaba de entrar con su cuenta. Sin embargo, si todo ha tenido éxito, el método ‘onSuccess’, que recibe un usuario Google como parámetro, será ejecutado, y es ahí donde inserimos como valor de un elemento html el nombre del usuario autenticado para ser exhibido en la pantalla, y lo redireccionamos a la página de administración. Este fichero también incluye el código de los elementos gráficos abajo presentados:

```
<p:fieldset legend=":: Seja Bem Vindo ::">
         <p:panel style="width: 450px" header="login" >

                  <f:facet name="header">
                           <h:outputText value="Clique Aqui" />
                  </f:facet>
         <h:panelGrid columns="2" width="300" >
         <h:outputText value="Usuário" />
         <p:inputText id="login"  required = "fasle" />

         <h:outputText value="Senha" />
         <p:password id="senha"  required="false" /><br/>
                 <p:commandButton value="Fazer Login" ajax="false"
                   action="#{loginController.doGet(req, resp)}"/>

</h:panelGrid>

</p:panel>
```

Este código se encarga del modelado gráfico de nuestro sistema, haciendo con que se parezca a un sistema de escritorio, con una interfaz de usuario más amigable. El resultado de los códigos presentados hasta el momento es lo siguiente:

Figura 15 - Vista inicial del sistema.



Fuente: Print Screen obtenido por el autor.

El método invocado pos login nos redirecciona a la página principal de nuestro sistema, que muestra datos del coche, ofrece la opción de cambiar su status de seguridad, y nos muestra la ubicación del dispositivo que lleva el módulo GPS. El fichero es el admin2.xhml, y su código más relevante será explicado a continuación:

```
<script async="true" defer="true"

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA5ftfhNYwGWDe5uFfJberGsrPtD0
XnWSU&amp;signed_in=true&amp;callback=initMap"></script>
```

Primero cargamos la librería de la plataforma Google que nos permitirá implementar el mapa Google en nuestra página con todas sus funcionalidades.

A continuación crearemos nuestro mapa juntamente con las funciones que deseamos añadirle:

```
<script>
 function success(position) {
latitude = parseFloat(document.getElementById('lat').value);
longitude = parseFloat(document.getElementById('lon').value);
googleLatLng = new google.maps.LatLng(latitude, longitude);

var mapOtn={ zoom:8, center:googleLatLng, mapTypeId:google.maps.MapTypeId.ROAD };
var Pmap=document.getElementById("map"); var map=new google.maps.Map(Pmap, mapOtn);
addMarker(map, googleLatLng, "Seu carro", "Esta aqui"); }
```

La posición inicial del mapa es modificada automáticamente de acuerdo a los datos recibidos por GPS, y los valores de latitud y longitud son enviados para dos elementos html básicos del tipo input creados en nuestra página. Sin embargo, se ha decidido hacer algo más para tornar más sencillo el rastreo, como podremos ver:

```
function addMarker(map, googleLatLng, title, content){


var markerOptn={
        position:googleLatLng,
        map:map,
        title:title,
        animation:google.maps.Animation.DROP
         };
        var pos = new google.maps.MVCObject();
         var marker=new google.maps.Marker(markerOptn);

         var infoWindow=new google.maps.InfoWindow({ content: content,
                                                    position: googleLatLng});
    google.maps.event.addListener(marker, "click", function(){
                infoWindow.open(map);
         });
        setInterval(function(){
                     latitude = parseFloat(document.getElementById('lat').value);
                     longitude = parseFloat(document.getElementById('lon').value);
                     googleLatLng = new google.maps.LatLng(latitude,
                                                    longitude);
                   pos.set('googleLatLng', new google.maps.LatLng(latitude,
                                                    longitude));
                   marker.bindTo('position', pos, 'googleLatLng');
                 }, 2000);
}
```

Adicionamos un marcador en el mapa que indicará la posición del coche. Sin embargo, este marcador actualiza su posición automáticamente cada dos segundos, por lo que el script vuelve a consultar pasado este tiempo los valores que están llegando a los elementos HTML definidos anteriormente.

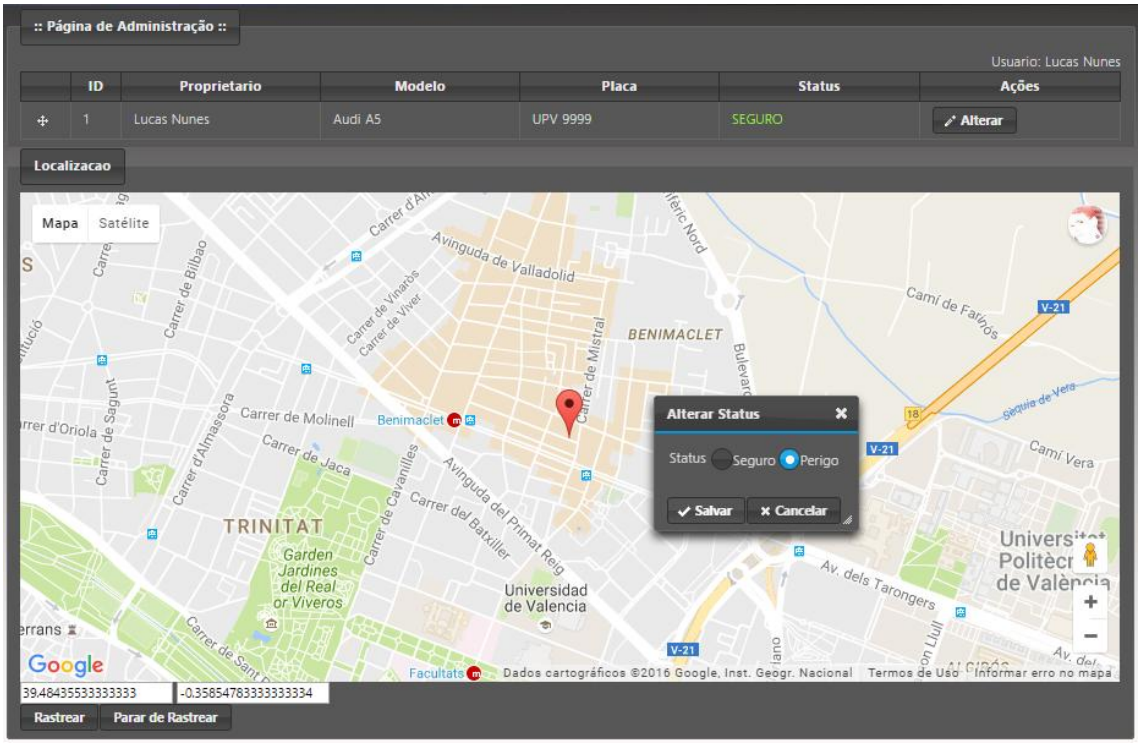
Aparte de eso, con el uso de Primefaces montamos una tabla con los atributos del coche y su opción de cambio de status de seguridad, que invocará el método explicado en la sección del controlador:

```
<p:dialog header="Alterar Status" widgetVar="AlterarDialogo">
             <p:outputPanel id="alterarDetalhe">
                 <h:panelGrid columns="2">
                     <h:outputLabel value="Status"/>
                     <p:selectOneRadio id="console"
value="#{detalhesController.detalhe.status}">
                        <f:selectItem itemLabel="Seguro" itemValue="0" />
                        <f:selectItem itemLabel="Perigo" itemValue="1" />
                     </p:selectOneRadio>
                 </h:panelGrid>
                 <br/>
                 <p:commandButton value="Salvar"
action="#{detalhesController.alterandoDetalhe()}" icon="ui-icon-check" ajax="false"
/>
                 <p:commandButton value="Cancelar"
onclick="PF('AlterarDialogo').close()" icon="ui-icon-close" />
```

El código descrito muestra la ventana que se abre cuando pinchamos en la opción de cambiar el status de seguridad del coche donde, dependiendo del valor que enviamos, su status cambiará. El botón ‘Salvar’ llamará el método no controlador que llamará el método *actualizar* en la clase ‘DaoDetalhe’, permitiendo así grabar el nuevo valor en la base de datos. El controlador ‘DetalhesController’ también enviará los mensajes utilizando el RabbitMQ.

El resultado del código presentados anteriormente es el siguiente:

Figura 16 - Página de administración



Fuente: Print Screen del sistema de vigilancia.

Tal y como se puede ver en la imagen, hay un marcador que nos indica la posición de nuestra Raspberry Pi. El mapa insertado en nuestra página cuenta con recursos típicos del Google Maps tales como vista de satélite, Street view, y controles de zoom. Se podrían añadir más funcionalidades al mapa, pero no serían esenciales en esta etapa de desarrollo.

4.2.4.6 Instalación y Implementación del PiCamara

En esta sección se ofrecerá una breve explicación de cómo se ha instalado el módulo de la cámara para Raspberry pi y de su implementación.

Primero se hace la instalación del componente hardware, en este caso el módulo de la cámara que será insertado en el conector situado entre el puerto HDMI y el conector AV, como muestra la figura 17:

Figura 17 - Camara instalada en la Raspberry Pi.

Fuente: Imagen obtenida por el autor.

Una vez instalado este componente, encendemos la Raspberry Pi para invocar el terminal y entrar con el siguiente comando:

```
Sudo raspi-config
```

Lo cual nos abrirá una ventana con opciones de ajustes de nuestra Raspberry Pi. Lo que tendremos que hacer aquí es navegar hasta la opción de la cámara y activar el suporte a este dispositivo.

Figura 18 - Activación del módulo de cámara

Fuente: Imagen obtenida por el autor.

Después, seleccionamos la opción de finalizar y reiniciamos la Raspberry Pi para que los nuevos ajustes sean aplicados.

La aplicación que hemos utilizado para sacar fotos desde la cámara instalada en la Raspberry es la *raspistill*, que ya viene instalada en el sistema operativo que estamos utilizando, en nuestro caso el Raspbian, y se ejecuta por medio de líneas de comando.

Todo que tuvimos que hacer en este caso fue invocar el terminal desde Java para introducir con los comandos y argumentos correspondientes para sacar fotos desde nuestra cámara. Esto se resume básicamente en el método siguiente:

```
public void TakePicture(){
         try{
              StringBuilder sb = new StringBuilder(_raspistillPath);
                           sb.append(" -n -bm");
                           sb.append(" -t " + _picTimeout);
                           sb.append(" -w " + _picWidth);
                           sb.append(" -h " + _picHeight);
                           sb.append(" -q " + _picQuality);
                           sb.append(" -e " + _picType);
                           sb.append(" -v -o " + _picName);
                    Runtime.getRuntime().exec(sb.toString());
                    System.out.println(sb.toString());
                    Thread.sleep(6000);
         }catch (Exception e){
             e.printStackTrace();
                     }
         }
```

A continuación encapsulamos este código en un método de nuestro controlador principal, y además insertamos un botón en la capa de vista para permitir al usuario sacar fotos cuando lo desee.

5 RESULTADOS EXPERIMENTALES

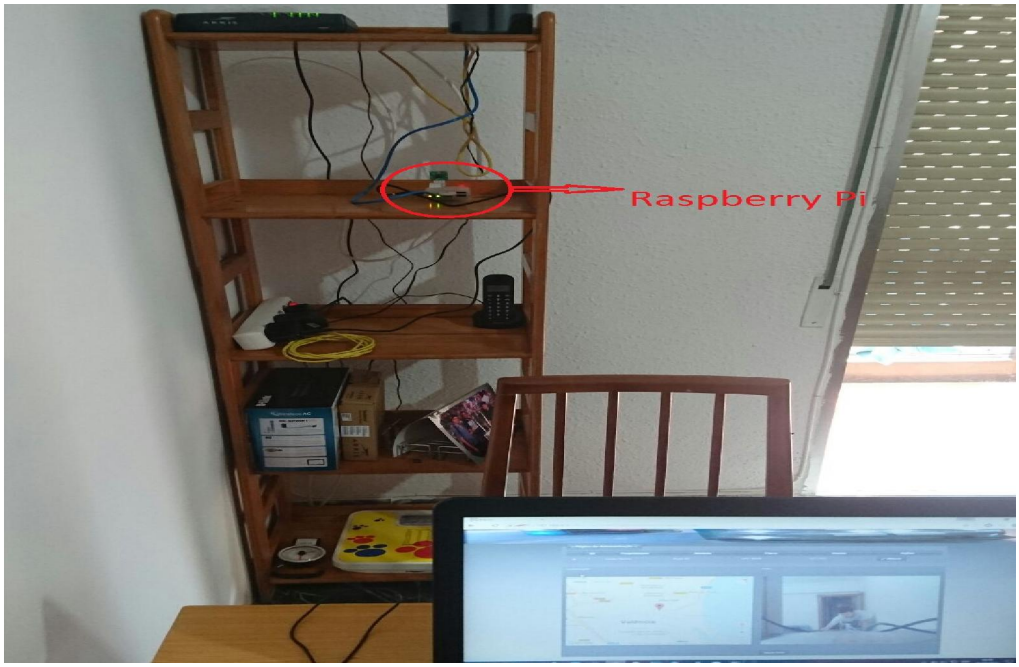
El entorno de pruebas de este sistema embebido para vigilancia, que ha permitido realizar pruebas para validar cada funcionalidad del sistema, incluyendo montaje de los componentes hardware y desarrollo software, tuvo lugar en una única ubicación, que ofrecía una red doméstica con acceso a internet, suficiente para hacer las pruebas necesarias.

La estrategia seguida para las pruebas fue la más sencilla posible, teniendo en cuenta que no haría falta simular entornos más complexos para comprobar el funcionamiento del sistema. Lo que logramos hacer fue conectar la Raspberry Pi con un router inalámbrico por su puerto Ethernet, para obtener una IP en nuestra red doméstica. Igualmente conectamos nuestro ordenador, donde hemos desarrollado todo el código que posteriormente fue adaptado para la Raspberry, a esta misma red mediante la conexión wifi.

Con los dos componentes (Raspberry Pi y ordenador) conectados en la misma red, y por lo tanto pertenecientes al mismo rango IP, podemos de una manera simples comprobar el estado de conexión entre ellos, o sea, si los dos componentes se ven y a través de nuestro router, y poder así gestionar las direcciones atribuidas a cada dispositivo cuando haga falta. Sin embargo, necesitamos saber las direcciones IP para configurar nuestro servicio de mensajería. En esta etapa arrancamos nuestro servicio gestor de cola de mensajes en nuestro ordenador.

Como podemos ver en la figura 19, nuestro esquema está montado e listo para que las pruebas sean ejecutadas:

Figura 19 - Entorno listo para pruebas



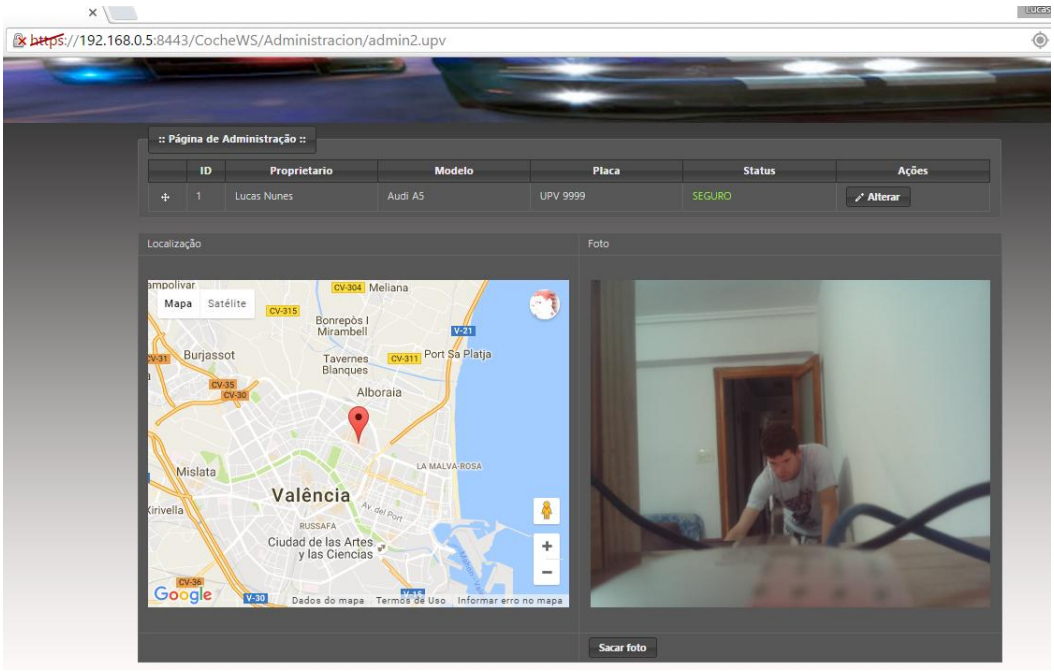
Fuente: Imagen obtenida por el autor.

Una vez que la Raspberry Pi estará conectada al router, tendremos que saber qué dirección IP le fue atribuida para ponerla en nuestra URL a la hora de acceder a la nuestra aplicación Web, podremos descubrirla accediendo la página de configuración del router y mirando los listados de direcciones IP ya asignados.

Con la dirección IP que se ha averiguado, desde nuestro ordenador intentaremos acceder a nuestra aplicación Web digitando la dirección atribuida y el puerto del servicio que hemos configurado en la etapa de configuración de nuestro servidor Web ejecutándose en la Raspberry.

Una vez logramos acceder a nuestra aplicación Web, tendremos que comprobar si todo estará bien con nuestra base de datos, y también dentro de nuestra plataforma IoT, haciendo el login; si todo está de bien con la base de datos y las informaciones insertadas en los campos de usuario y contraseña, seremos redireccionados a la página principal de nuestro sistema:

Figura 20 - Pantalla inicial del sistema

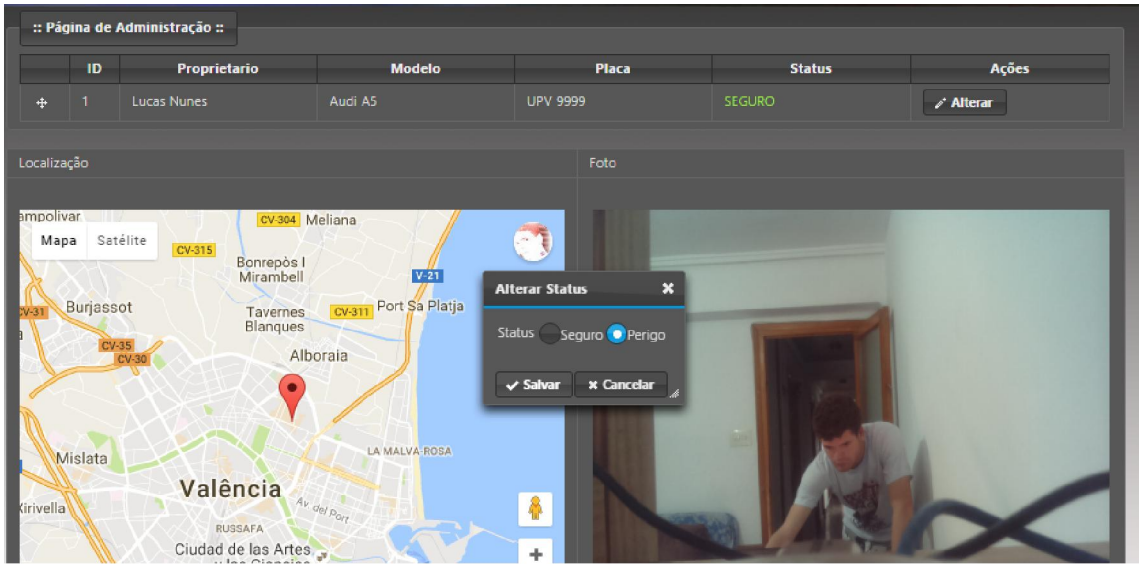


Fuente: Print screen de la pantalla principal del sistema.

En esta página podremos comprobar que los datos insertados previamente en la base de datos son correctos, así como ver la correcta ubicación de nuestro dispositivo que está señalada por un marcador en el mapa.

La siguiente etapa es comprobar si nuestro sistema de entrega de mensajes estará funcionando bien haciendo clic sobre la opción ‘alterar status’, donde una ventanilla surge y nos pregunta qué estado deseamos elegir; en nuestro caso elegiremos el estado ‘PERIGO’ (peligro).

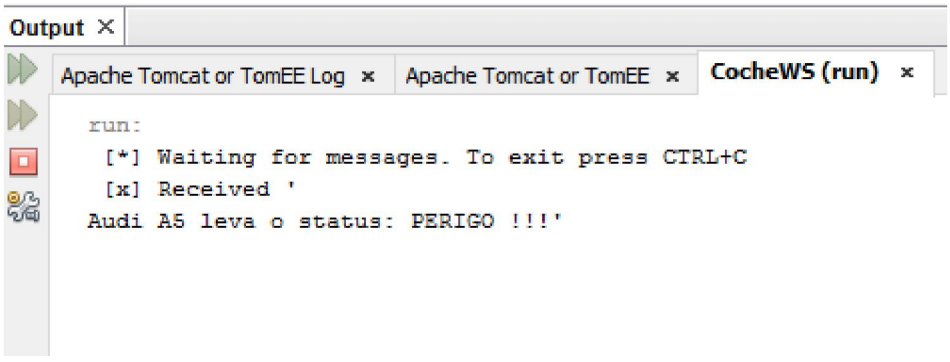
Figura 21 - Ventanilla de cambio de estado



Fuente: Print screen de la pantalla principal del sistema.

En este momento, y de acuerdo a lo que fue explicado en la sección de desarrollo, nuestro sistema tendría que grabar el nuevo estado en la base de datos y hacer el envío de un mensaje conteniendo datos del coche, bien como su estado, para los consumidores de mensaje, que en este caso será nuestro ordenador que estará con la clase Java de recepción de mensajes escuchando todo lo que sea enviado.

Figura 22 - Mensaje enviada desde la RPI al Ordenador.

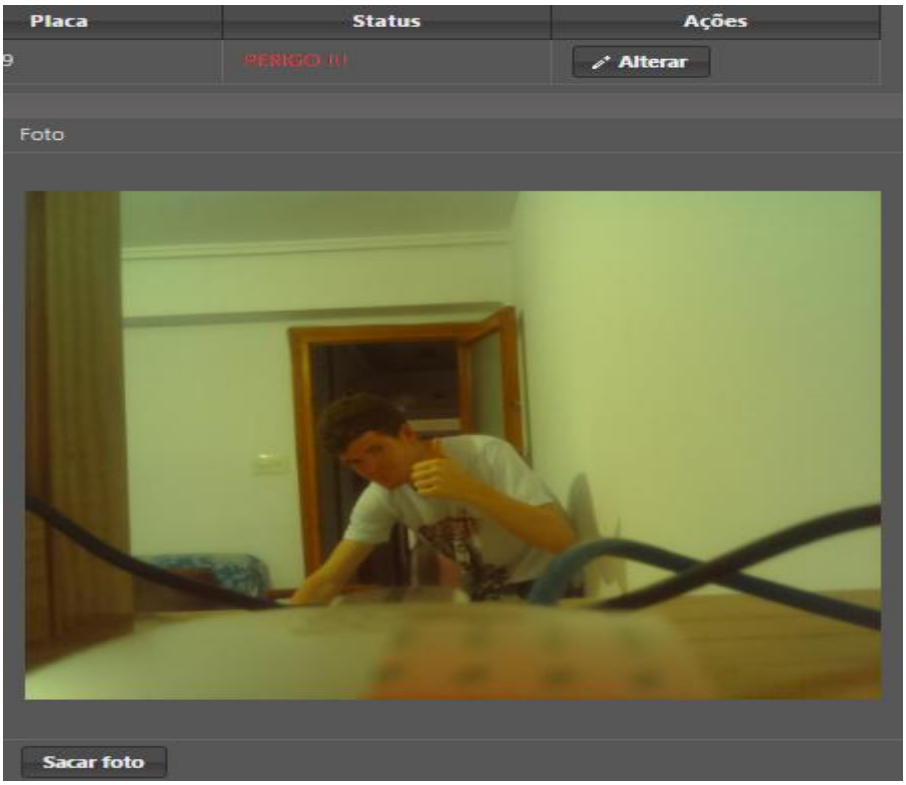


Fuente: Print de la consola de la clase receptora de mensajes.

Tal y como podemos ver en la imagen, nuestras principales funcionalidades han sido comprobadas con éxito. Por último, nos falta probar la funcionalidad de sacar foto desde nuestro ordenador utilizando la cámara que está instalada en nuestra Raspberry clicando en el botón

‘sacar foto’, momento en que el comando accionará la aplicación de la cámara dentro de la Raspberry Pi, y al refrescar de la página podremos ver la foto que ha sido sacada al lado del mapa.

Figura 23 - Foto obtenida por la funcionalidad implementada



Fuente: Print screen de la pantalla principal del sistema.

Hemos visto que todas las funcionalidades de nuestro sistema obtuvieran éxito en las probas, y que los resultados fueran satisfactorios dentro de las limitaciones del hardware utilizado. Por ejemplo, el módulo receptor de GPS lleva una antena interna que es de baja ganancia, y por eso los datos recibidos tardan en llegar y por veces se pierden con facilidad dependiendo del entorno donde estamos; en nuestro caso probamos con nuestros equipos dentro de una finca, lo que hace más difícil la recepción de las señales GPS debido a la gran cantidad de material entre la antena receptora y el satélite. Tal problema se soluciona de manera sencilla si probamos con nuestra Raspberry en una zona abierta.

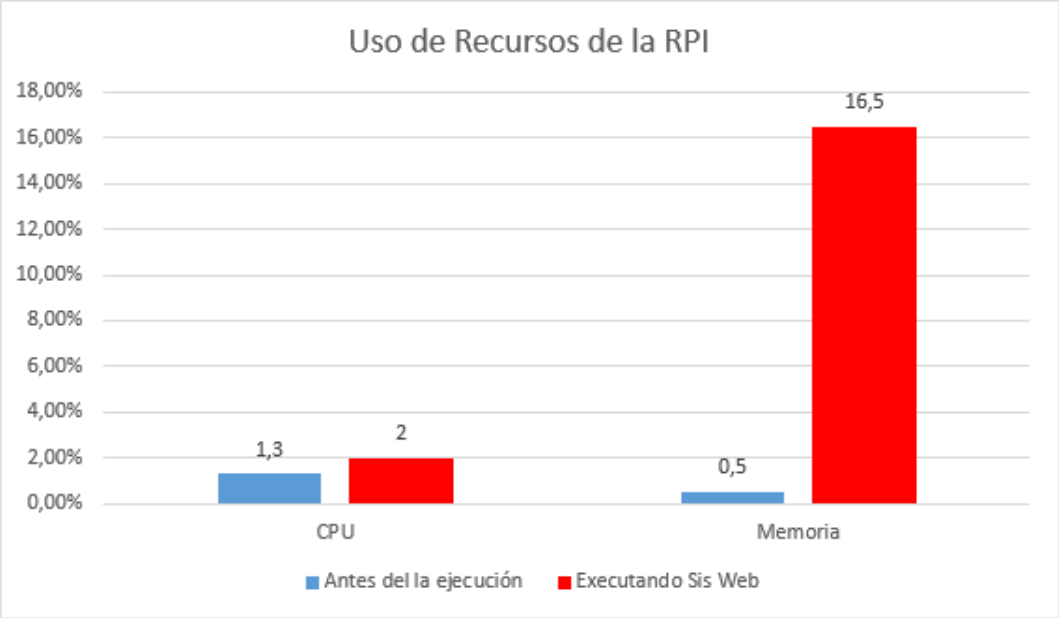
Es importante destacar que hemos realizados estas pruebas en un entorno fijo, lo cual es distinto del entorno para el cual se desea utilizar nuestro sistema, que sería en coches. Una solución para este problema

sería utilizar un cargador externo como fuente de alimentación de nuestra Raspberry Pi, garantizando así su movilidad.

Para el tema de las redes a utilizar en caso de movilidad total de nuestro sistema, podría acoplarse un modem 3G USB a nuestra Raspberry Pi, y aprovechar así la elevada cobertura de esta red. En este caso la comunicación entre los dispositivos seria por medio de un dominio o IP fija, que podría ser ofrecido de manera gratuita por servicios como el famoso NoIP, para poner un ejemplo.

Sin embargo, al final de las pruebas hemos evaluado el uso de los recursos en la Raspberry Pi antes y mientras se ejecuta nuestra aplicación, y hemos obtenido la siguiente gráfica:

Figura 24 - Uso de recursos de la Raspberry Pi



Fuente: Gráfica obtenida por el autor.

Tal y como podemos ver, los valores porcentuales relativos al uso de la CPU si mantienen próximos antes y durante la ejecución de nuestro sistema, mientras que el consumo de memoria utilizada en la RPi, que totaliza 883MB, se eleva significativamente mientras ejecutamos nuestra aplicación. Eso es debido a que el contenedor Web hace una reserva de memoria para la máquina virtual; la cantidad de memoria virtual utilizada por el Tomcat puede ser ajustada en el fichero *catalina.sh* .

6 CONCLUSIONES Y TRABAJOS FUTUROS

El propósito de este Trabajo Fin de Máster ha sido la implementación de un sistema embebido para vigilancia de vehículos capaz de obtener ubicación, datos y imágenes del vehículo que lleva nuestra plataforma IoT.

La labor realizada en este Trabajo Fin de Máster se puede dividir en dos partes: el desarrollo en un entorno de pruebas, que en este caso sería nuestro ordenador, y la implementación y adaptación del código para las pruebas en un entorno real basado en Raspberry Pi.

Tomando como ejemplo este sistema que hemos desarrollado para un tipo de entorno específico conseguimos de cierta manera integrar los vehículos en el universo IoT, lo cual es uno de los desafíos de aplicaciones para estos entornos: hacer con que los dispositivos que no están aún integrados en los mismos puedan interactuar con los que ya disponen de esta tecnología.

De manera general, el trabajo realizado en este proyecto se valora positivamente, aunque hay puntos que podrían ser mejorados, funcionalidades añadidas, y pequeños cambios para mejorar las prestaciones. A continuación, se expone una serie propuestas que podrían ser oportunas para la mejora del proyecto:

- **Mejora de la interfaz gráfica**. La interfaz de usuario de nuestro sistema cumple básicamente con los requisitos funcionales establecidos, pero sabemos que a nivel de lo que sería una aplicación comercial habría que mejorar un poco este aspecto. Destacar que hicimos uso del Primefaces para tener en cuenta dichos requisitos futuros, aunque dentro del abanico de opciones que nos ofrece esta herramienta hemos utilizado lo más básico. De todos modos, los objetivos del proyecto no se centran en mejoras del diseño gráfico.
- **Comandos enviados hacia el coche**. De cada a determinar cómo puede ser escalable nuestro sistema, se ha estudiado la conveniencia de incluir varias funcionalidades, y una de ellas ha sido la posibilidad de permitir al usuario enviar un comando por la aplicación Web en la Raspberry Pi, y que ésta pueda transmitir este comando directamente para el coche, ya sea apagarlo, disparar una señal de alerta o alarma de seguridad, etc. Esta idea se ha mantenido en

segundo plano, pudiendo ser incorporada o no dependiendo del tiempo que llevaríamos en desarrollar las principales funcionalidades que en este trabajo hemos implementado.

En líneas generales, el trabajo realizado en esta tesina me ha proporcionado una grande experiencia en una nueva plataforma de desarrollo, ya que ésta ha sido mi primera vez desarrollando un sistema completo para Raspberry Pi.

7 REFERENCIAS

[1] «Interactive Adversiting Bureau,» [En línea]. Available: http://www.iabspain.net/wp-content/uploads/downloads/2014/07/Informe-coches-conectados-2014.pdf. [Último acceso: 30 8 2016].

[2] «BMW Mobilie Applications,» [En línea]. Available: http://www.bmw.com/com/en/owners/bmw_apps_2013/apps/my_bmw_remote_app/. [Último acceso: 30 8 2016].

[3] «Protege tu vehiculo,» [En línea]. Available: https://www.protegetuvehiculo.es/. [Último acceso: 30 8 2016].

[4] D. Evans, «Internet de las cosas - Cómo la próxima evolución,» Abril 2011.

[5] Joseph Bradley, Joel Barbier, Doug Handler, «Embracing the Internet of Everything To Capture Your Share of $14.4 Trillion,» Cisco, 2013.

[6] Mario Cruz Vega, Pablo Oliete Vivas, Christian Morales Rios, Carlos González Luis, Bruno Cendón Martín, Alberto Hernández Seco, EOI Escuela De Organización Industrial, PWC, Las tecnologías IoT dentro de la industria conectada 4.0, FSC, 2015.

[7] Luigi Atzori, Antonio Iera, Giacomo Morabito, «The Internet of Things: A Survey,» Junio 2010.

[8] Igor Tomié, Srdan Krčo, Divna Vučković, Alex Gluhak, Pirabakaran Navaratnam, «SENSEI traffic modelling,» *17th Telecommunications forum TELFOR,* pp. 24-26, Noviembre 2009.

[9] «Raspberry Pi Foundation,» [En línea]. Available: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/. [Último acceso: 13 Julio 2016].

[10] H. M. James Gosling, «The Java Language Environment - A White Paper,» 1995.

[11] M. H. Sarath Gude, «JavaScript: The Used Parts,» *IEEE 38th Annual International Computers, Software and Applications Conference,* 2014.

[12] H. Z. Chávez, «JavaServer Faces: An Excellent Learning Tool,» *IEEE Distributed Systems Online, vol. 6, no. 2,* 2005.

[13] Y. L. C.-L. W. King Tin Lam, «A Performance Study of Clustering Web Application Servers with Distributed JVM,» *2008 14th IEEE International Conference on Parallel and Distributed Systems,* 2008.

8 ANEXOS

En este anexo se incluye el código completo de las principales clases que componen este proyecto, las cuales fueran referidas en capítulos anteriores, así como algunos ficheros de configuración.

8.1 DETALHES.JAVA

```
package srm.model;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
*
* @author Lucas
*/
@Entity
@Table(name ="detalhes", schema = "coche")
public class Detalhes implements Serializable{

public Detalhes(){

}

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "Id")
private int id;

@Column(name = "Proprietario")
private String proprietario;

@Column(name = "Placa")
private String placa;

@Column(name = "Modelo")
private String modelo;

@Column(name = "Status")
private int status;

public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getProprietario() {
return proprietario;
}
public void setProprietario(String proprietario) {
this.proprietario = proprietario;
}

public String getPlaca() {
return placa;
}
```

```
public void setPlaca(String placa) {
this.placa = placa;
}
public String getModelo() {
return modelo;
}
public void setModelo(String modelo) {
this.modelo = modelo;
}
public int getStatus() {
return status;
}
public void setStatus(int status) {
this.status = status;
}
}
```

8.2 DETALHESCONTROLLER.JAVA

```
package srm.controller;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import java.util.List;
import javafx.scene.control.TableColumn.CellEditEvent;
import javax.annotation.PostConstruct;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.context.FacesContext;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import srm.dao.DaoDetalhes;
import srm.model.Detalhes;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.ConnectionFactory;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
import javax.faces.bean.ViewScoped;
import net.sf.marineapi.nmea.event.SentenceEvent;
import net.sf.marineapi.nmea.event.SentenceListener;
import net.sf.marineapi.nmea.io.SentenceReader;
import net.sf.marineapi.nmea.sentence.GLLSentence;
import net.sf.marineapi.nmea.sentence.SentenceId;
import net.sf.marineapi.nmea.sentence.SentenceValidator;

/**
 *
 * @author Lucas
 */
@ManagedBean

public class DetalhesController implements SentenceListener{

    public DaoDetalhes daoDetalhe;
    private Detalhes detalhe;
    private final static String QUEUE_NAME = "hello";
    private List<Detalhes> listaDetalhes;
     Double lat;
     Double lon;
    private DecimalFormatSymbols symbols;
    private SentenceReader reader;
    boolean ler = true;
    public static double latt;
    public DetalhesController(){

        daoDetalhe = new DaoDetalhes();
        this.detalhe = new Detalhes();
    }

    @PostConstruct
    public void inicializar(){
        try{
          if
(FacesContext.getCurrentInstance().getExternalContext().getSessionMap().containsKey
("id")){
                //Recebe o ID informado
                int id = new
Integer(FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get(
"id").toString());

                //Busca o objeto e preenche os valores na pagina
                detalhe = daoDetalhe.retornaDetalhesPor(id);
                //Remove o atributo da sessão para utilizar novamente.
```

```
                //
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().remove("idTe
cnico");
            }
         }
        catch (Throwable ex){

        }

    }

    public Double getLat() {
        return lat;
    }

    public void setLat(double lat) {
        this.lat = lat;
    }

    public void setLon(double lon) {
        this.lon = lon;
    }

    public Double getLon() {
       return lon;
    }
    public void atualizaLatLon(){
        this.getLat();
        this.getLon();
       // init();
        System.out.println("\n\n*******\n Chamou o lat: "+this.getLat()+ "e o lon:
"+this.getLon()+"\n******");
    }
    public Detalhes getDetalhe() {
        return detalhe;
    }

    public void setDetalhe(Detalhes detalhe) {
        this.detalhe = detalhe;
    }

    public void alterandoDetalhe() throws IOException{

      daoDetalhe = new DaoDetalhes();
      daoDetalhe.alterarDetalhe(this.detalhe);
      String status = "";
      if(this.detalhe.getStatus()==0){
          status = "SEGURO";
```

```
      }else{
          status = "PERIGO !!!";
      }
    ConnectionFactory factory = new ConnectionFactory();

    factory.setHost("192.168.0.168");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    String message = "\n"+this.detalhe.getModelo()+" leva o status: "+status;
    channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
    System.out.println(" [x] Sent '" + message + "'");

    channel.close();
    connection.close();

    }
    public void jogarIdSessao(){


FacesContext.getCurrentInstance().getExternalContext().getSessionMap().put("id",
detalhe.getId());

    }

      public List<Detalhes> getListaDetalhes() {
        if (this.listaDetalhes == null) {
            this.listaDetalhes = daoDetalhe.retornarTodosArea();
        }
        return listaDetalhes;
    }
    public void setListaArea(List<Detalhes> listaArea) {
        this.listaDetalhes = listaArea;
    }


    public String direcionamentoPag(String dir, String pagina) {

        return "/" + dir + "/" + pagina + ".upv?faces-redirect=true";

    }
    public String paginaAdmin(){
     return  this.direcionamentoPag("Administracion","admin");
    }

    public void rastrear(){
       ler = true;
```

```
        System.out.println("\n rastrear invocado: "+ler);
        init();
    }


    @Override
    public void readingPaused() {
        System.out.println("-- Paused --");
    }

    @Override
    public void readingStarted() {
        System.out.println("-- Started --");
    }

    @Override
    public void readingStopped() {
        ler = false;
        try{
                    SerialPort teste = (SerialPort)
CommPortIdentifier.getPortIdentifier("/dev/ttyACM0").open("/dev/ttyACM0", 30);
                    teste.close();

        }catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("-- Stopped --");
    }


    @Override
    public void sentenceRead(SentenceEvent event) {
        if(event.getSentence().getSentenceId().equals("GLL")){
             GLLSentence gll = (GLLSentence) event.getSentence();
             if(getLat()==null && getLon()==null){
                 setLat(gll.getPosition().getLatitude());
                 setLon(gll.getPosition().getLongitude());
                 SerialPortExample spe = new SerialPortExample(getLat(), getLon());

             }


             System.out.println("\nDados chegando ao sentenceRead:\n
"+getLat()+"\n\n "+getLon());

        }else{
            System.out.println("nao e gll ? "+event.toString());
        }
```

```
        System.out.print("id do evento: "+event.getSentence().getSentenceId());
              // reader.stop();
    }
    private SerialPort getSerialPort() {
                try {

                                        SerialPort teste = (SerialPort)
CommPortIdentifier.getPortIdentifier("COM14").open("COM14", 30);//tinha 30 aqui
                                            teste.setSerialPortParams(9600,
SerialPort.DATABITS_8,

         SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);

                            InputStream is = teste.getInputStream();
                            InputStreamReader isr = new InputStreamReader(is);
                            BufferedReader buf = new BufferedReader(isr);


         System.out.println("Scanning port " + teste.getName());

// try each port few times before giving up
         for (int i = 0; i < 5; i++) {
                  try {
                            String data = buf.readLine();
                                     if (SentenceValidator.isValid(data)) {

         System.out.println("NMEA data found!");
                                            return teste;
                                                      }
                                            } catch (Exception ex) {
                                            ex.printStackTrace();
                                            }
                                    }
                                    is.close();
                                    isr.close();
                                    buf.close();


                            System.out.println("NMEA data was not found..");

                  } catch (Exception e) {
                            e.printStackTrace();
                  }

                  return null;
         }

    private void init() {
```

```
                try {
                        SerialPort sp = getSerialPort();

                        if (sp != null) {
                                InputStream is = sp.getInputStream();
                                SentenceReader sr = new SentenceReader(is);
                                sr.addSentenceListener(this);
                                if(ler!=true){
                                sr.stop();
                             } else {
                                sr.start();
                         }


                        }

                } catch (IOException e) {
                        e.printStackTrace();
                }
         }

    public void tirarFoto() throws IOException{
        RaspiStil raspi = new RaspiStil();
        System.out.println("\n\n chamou o metodo de tirar foto \n\n");
      try{
          raspi.TakePicture();
        System.out.println("\n\n foto obtida \n\n");
          }catch (Exception e){
          e.printStackTrace();
      }
    }

}
```

8.3 RECV.JAVA

```
         package srm.controller;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.QueueingConsumer;

public class Recv {
```

```
    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");


    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

    QueueingConsumer consumer = new QueueingConsumer(channel);
    channel.basicConsume(QUEUE_NAME, true, consumer);

    while (true) {
      QueueingConsumer.Delivery delivery = consumer.nextDelivery();
      String message = new String(delivery.getBody());
      System.out.println(" [x] Received '" + message + "'");
    }
  }
}
```

8.4 DAODETALHES.JAVA

```
package srm.dao;


import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.Transaction;
import srm.model.Detalhes;
import srm.util.HibernateUtil;

/**
 *
 * @author Lucas
```

```
 */
public class DaoDetalhes {

    private Dao dao;

    public DaoDetalhes(){

        this.dao = new Dao();

    }

     public Object inserirArea(Detalhes detalhe){

        try {
            detalhe = (Detalhes)this.dao.salva(detalhe);

            return detalhe;
        } catch (Exception ex) {
           Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);
           System.out.println("Erro na Classe: 'srm.DaoDetalhe.inserirDetalhe' !");
            return null;
        }


    }

    public Detalhes alterarDetalhe(Detalhes detalhe){
          try {
                   detalhe = (Detalhes)this.dao.salva(detalhe);
           //        this.log.criaLog("Usuário
:"+LoginController.usuarioLogado.getNome()+" - alterou area :
"+area.getDescricao(), new Throwable().getStackTrace()[0].getMethodName());
                  return detalhe;
               } catch (Exception ex) {
                   Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null,
ex);
                   System.out.println("Erro na Classe:
'srm.DaoDetalhes.alterarArea' !");
                   return null;
                }

    }
    public void update(Detalhes detalhe){
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction t = session.beginTransaction();
        session.update(detalhe);
        t.commit();
    }
```

```
    public List<Detalhes> retornarTodosArea(){

         try {
                  return this.dao.listaTudo(Detalhes.class);
               } catch (Exception ex) {
                   Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null,
ex);
                   System.out.println("Erro na Classe:
'srm.DaoDetalhes.retornarTodosDetalhes' !");
                   return null;
                }
    }

   public Detalhes retornaDetalhesPor(int id){
         try {
                   Detalhes detalhe;
                   detalhe = (Detalhes)this.dao.procura(id, Detalhes.class);
                  return detalhe;
               } catch (Exception ex) {
                   Logger.getLogger(Detalhes.class.getName()).log(Level.SEVERE,
null, ex);
                   return null;
               }
    }
}
```

8.5 HIBERNATE.CFG.XML

```
        <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost/coche</property>
```

```
        <property name="hibernate.hbm2ddl.auto">none</property>
        <property name="hibernate.show_sql">false</property>
        <property name="hibernate.format_sql">true</property>
        <property name="connection.autoReconnect"> true </property>
        <property name="connection.autoReconnectForPools"> true </property>

        <mapping class="srm.model.Detalhes"/>
        <mapping class="srm.model.Usuario"/>

    </session-factory>
</hibernate-configuration>
```

8.6 ADMIN2.XHTML

```
        <?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">
  <script async="true" defer="true"

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA5ftfhNYwGWDe5uFfJberGsrPtD0
XnWSU&amp;signed_in=true&amp;callback=initMap"></script>

    <h:head>
        <title>Admin</title>

    </h:head>
    <ui:decorate template="../padrao.xhtml">
        <ui:define name="centro2">

    <h:body>
     <h:form id="forma">

         <div align="left">
         <p:fieldset legend=":: Página de Administração ::">
             <div align="right">
                 <h:outputText value=" #{loginController.usuario}" />
             </div>
```

```
        <p:dataTable var="detalhe" value="#{detalhesController.listaDetalhes}"
widgetVar="detalheTable" id="pesquisarDetalhe"
                        style="alignment-adjust: auto; width: 900px" >


            <p:column style="width:20px">
                <h:outputText id="dragIcon"
                              styleClass="ui-icon ui-icon-arrow-4" />
                <p:draggable for="dragIcon" revert="true" />
            </p:column>

             <p:column headerText="ID" style="width:20px">

                    <h:outputText value="#{detalhe.id}" />

             </p:column>

             <p:column headerText="Proprietario">

                 <h:outputText value="#{detalhe.proprietario}" />

             </p:column>

             <p:column headerText="Modelo">
                 <h:outputText value="#{detalhe.modelo}" />
             </p:column>

             <p:column headerText="Placa">
                 <h:outputText value="#{detalhe.placa}" />
                 <input type="hidden" id="carro" value="#{detalhe.placa}"/>
             </p:column>
             <p:column headerText="Status">
                 <h:outputText style="color: greenyellow; alignment-baseline:
central" id="status1" rendered="#{detalhe.status==0}" value=" SEGURO " />
                 <h:outputText style="color: red" id="status2"
rendered="#{detalhe.status==1}" value=" PERIGO !!!" />
             </p:column>
             <p:column headerText="Ações">
                 <p:commandButton style="size: 12px" value="Alterar" icon="ui-icon-
pencil"  action="#{detalhesController.jogarIdSessao}"
update=":forma:alterarDetalhe"  oncomplete="PF('AlterarDialogo').show()"
title="Alterar Detalhe">

                    <f:setPropertyActionListener value="#{detalhe}"
target="#{detalhesController.detalhe}"/>

                 </p:commandButton>
             </p:column>
```

```
        </p:dataTable>

        </p:fieldset>
            </div>
        <p:dialog header="Alterar Status" widgetVar="AlterarDialogo">
            <p:outputPanel id="alterarDetalhe">
                <h:panelGrid columns="2">
                    <h:outputLabel value="Status"/>
                    <p:selectOneRadio id="console"
value="#{detalhesController.detalhe.status}">
                       <f:selectItem itemLabel="Seguro" itemValue="0" />
                       <f:selectItem itemLabel="Perigo" itemValue="1" />
                    </p:selectOneRadio>
                </h:panelGrid>
                <br/>
                <p:commandButton value="Salvar"
action="#{detalhesController.alterandoDetalhe()}" icon="ui-icon-check" ajax="false"
/>
                <p:commandButton value="Cancelar"
onclick="PF('AlterarDialogo').close()" icon="ui-icon-close" />

            </p:outputPanel>

        </p:dialog>

        <p:fieldset id="field" legend="Localizacao" >
            <h:form id="mapa">

                <div  id="map" style="width: 899px; height: 400px" align="right"/>
                <p:outputPanel id="latLon"  autoUpdate="true">
                    <input id="lat" value="#{serialPortExample.latitude}"/>
                    <input id="lon"  value="#{serialPortExample.longitude}"/>

               <!-- <p:remoteCommand name="atualiza" update="latLon"
actionListener="#{detalhesController.atualizaLatLon()}"/>-->
                <p:poll interval="2" update="latLon"
listener="#{detalhesController.atualizaLatLon()}" />
                    </p:outputPanel>
                </h:form>

            <p:commandButton value="Rastrear"
action="#{detalhesController.rastrear()}" />
            <p:commandButton value="Parar de Rastrear"
action="#{detalhesController.readingStopped()}" />
            <p:outputPanel id="panelFoto" autoUpdate="true">
            <p:fieldset legend=":: Foto ::">
                <p:graphicImage value="../Imagens/test.jpg" alt="Imagen tomada" />
                <br/>
```

```
                <p:commandButton process="@form" update="forma:panelFoto"
value="Sacar foto" actionListener="#{detalhesController.tirarFoto()}" />

            </p:fieldset>
            </p:outputPanel>
            </p:fieldset>


     </h:form>

           <script>
var watchID = null;

        var latitude = parseFloat(document.getElementById('lat').value);
        var longitude = parseFloat(document.getElementById('lon').value);

         if( navigator.geolocation ){
 navigator.geolocation.watchPosition(success, fail, optn);
     //setInterval(function(){

     //       updateMap(latitude, longitude);

     //   }, 3000);
    }else{
          $("p").html("HTML5 Not Supported");
     }
$("button").click(function(){

         if(watchID)
          navigator.geolocation.clearWatch(watchID);

         watchID = null;
         return false;
});

 var optn = {
                        enableHighAccuracy: true,
                           timeout: Infinity,
                           maximumAge: 0
                  };
//}

 function success(position)
{
         latitude = parseFloat(document.getElementById('lat').value);
         longitude = parseFloat(document.getElementById('lon').value);
          googleLatLng = new google.maps.LatLng(latitude,
                                                  longitude);
```

```
         var mapOtn={
zoom:8,
center:googleLatLng,
mapTypeId:google.maps.MapTypeId.ROAD
         };

         var Pmap=document.getElementById("map");

         var map=new google.maps.Map(Pmap, mapOtn);
         addMarker(map, googleLatLng, "Seu carro",
                  "SATISH B<br /><b>About Me:</b>http://technotip.com/about/");


}

 function addMarker(map, googleLatLng, title, content){


var markerOptn={
        position:googleLatLng,
        map:map,
        title:title,
        animation:google.maps.Animation.DROP
         };
        var pos = new google.maps.MVCObject();
         var marker=new google.maps.Marker(markerOptn);

         var infoWindow=new google.maps.InfoWindow({ content: content,
                                                     position: googleLatLng});
    google.maps.event.addListener(marker, "click", function(){
                infoWindow.open(map);
         });
        setInterval(function(){
                     latitude = parseFloat(document.getElementById('lat').value);
                     longitude = parseFloat(document.getElementById('lon').value);
                     googleLatLng = new google.maps.LatLng(latitude,
                                                           longitude);
                   pos.set('googleLatLng', new google.maps.LatLng(latitude,
                                                           longitude));
                   marker.bindTo('position', pos, 'googleLatLng');
                 }, 2000);
}

function fail(error)
{
         var errorType={
0:"Unknown Error",
1:"Permission denied by the user",
```

```
2:"Position of the user not available",
3:"Request timed out"
         };

         var errMsg = errorType[error.code];

         if(error.code == 0 || error.code == 2){
                  errMsg = errMsg+" - "+error.message;
         }

         $("p").html(errMsg);
}

function updateMap(lat, lon) {
        var latitude = parseFloat(document.getElementById('lat').value);
        var longitude = parseFloat(document.getElementById('lon').value);
         var googleLatLng = new google.maps.LatLng(latitude,
                                                   longitude);
        mapholder=document.getElementById('map');

    var myOptions={
      zoom:13,
center:googleLatLng,
mapTypeId:google.maps.MapTypeId.ROAD
  };

  var map=new google.maps.Map(document.getElementById("map"),myOptions);
  var marker=new google.maps.Marker({position:googleLatLng,map:map,title:"You are
here!"});
}
//geramapa();
    </script>

    </h:body>
        </ui:define>
    </ui:decorate>

</html>
```