Implementación de un sistema de gestión “In Cloud” de redes WSN

Francisco Jesús Villaseñor García
Master Universitario Ingeniería de telecomunicación (plan antiguo)
Telemática

José López Vicario
Xavier Vilajosana Guillen

12/06/2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	Implementación de un sistema de gestión "In Cloud" de redes WSN
Nombre del autor:	Francisco Jesús Villaseñor García
Nombre del consultor:	José López Vicario
Nombre del PRA:	Xavier Vilajosana Guillen
Fecha de entrega (mm/aaaa):	06/2016
Titulación:	*Master Universitario Ingeniería de telecomunicación (plan antiguo)*
Área del Trabajo Final:	Telemática
Idioma del trabajo:	*Castellano*
Palabras clave	*Interfaz web, OpenWSN, OpenMote-CC2538*

Resumen del Trabajo (máximo 250 palabras):

Las redes inalámbricas de sensores WSN son uno de los elementos fundamentales de la Internet de las cosas IoT. Estas redes van desde el control de temperatura y humedad en interiores, a aplicaciones en ciudades inteligentes, seguridad en infraestructuras, etc. Algunos de los retos a los que se enfrentan las WSN son la minimización del consumo (aumento de la autonomía) y la optimización de los recursos de los sensores, que suelen ser muy limitados en términos de memoria y capacidad de proceso.

Un sistema de control remoto en estas redes que facilite su gestión y operación supondría un aumento de la flexibilidad y el ahorro de costes económicos y de recursos humanos. La gestión remota incluye el conocimiento del estado de la red (nodos activos, estado de sus baterías, etc.), la recuperación de los valores de los sensores de la misma, la modificación de parámetros de configuración (tiempo entre muestras, etc.) y, como objetivo final, la actualización remota de software (flash over the air, Over-the-air programming OTA) que facilita la distribución centralizada de actualizaciones.

Dentro de este proyecto se incluye el dotar a la red de sensores de una interfaz web que facilite la gestión de la misma y que permita el acceso remoto utilizando las redes y navegadores estándar. El desarrollo se propone sobre Raspberry PI como ejemplo de entorno de bajo coste y recursos limitados que podría incluirse como componente adicional a una red WSN. Raspberry PI promueve el uso de Python como lenguaje de programación, de modo que se utilizará este lenguaje y el soporte de alguno de los frameworks existentes para facilitar la implementación de sistemas web.

Es parte del proyecto el desarrollo de la aplicación que se integrará en

las motas de la red WSN para responder a las peticiones y ejecutar los comandos que se invoquen desde la interfaz web. Para el desarrollo se utilizará un entorno OpenMote-CC2538 como modelo de hardware de redes de sensores comercial disponible para el uso.

Abstract (in English, 250 words or less):

Wireless sensor networks WSN are one of the fundamental elements of the Internet of Things IoT. These networks go from temperature and humidity control systems inside buildings to applications in smart cities, security in infrastructures, etc. Some of the challenges for WSN are minimizing the consumption (increasing autonomy) and the optimization of the resources of the sensors, which are usually limited in terms of memory and processor.

A remote control system for these network would make their management and operation easier and would increase the flexibility and the economic and human resources savings. The remote management of the network include know the network status (active nodes, battery charge, etc.), get the sensor values, change the configuration parameters (time between samples, etc.) and as a final goal, the remote software update (flash over the air, over-the-air programming OTA) that allows centralized updates.

This project includes creating a web interface for the sensor network that would make the management easier and would allow remote access using standard browsers and networks. Raspberry PI is the development environment proposed as an example of low cost component with limited resources that could be included as an additional element in a WSN. Raspberry PI uses Python as the fundamental programming language, so this language will be used, with the support of some of the existing frameworks created to make the implementation of web systems fast and easy.

Part of the project is the development of the application which be integrated into the WSN network motes to attend requests and execute commands invoked from the web interface. The development environment includes for this purpose the OpenMote-CC2538 as a commercial hardware for network sensors available for use.

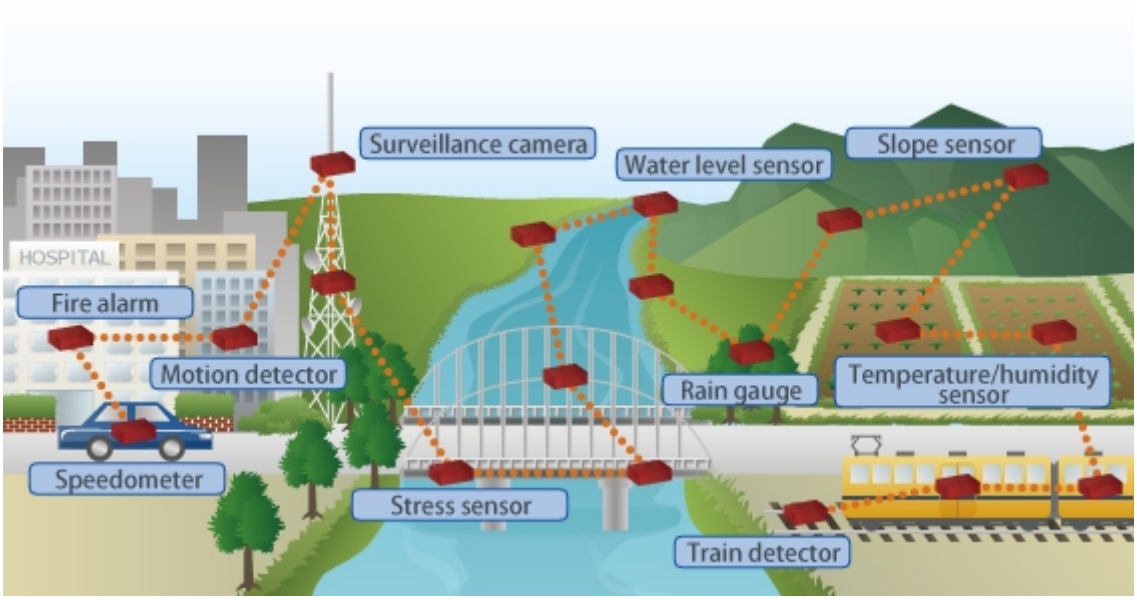
ÍNDICE

LISTA DE TABLAS Y FIGURAS

1. Introducción

1.1 Contexto y justificación del Trabajo

Las redes de sensores son cada vez más habituales y su uso se extiende cada vez a más ámbitos de aplicación: ciudades inteligentes, interior de edificios, entornos industriales, medio ambiente.



1.1.1 Usos de las redes de sensores
Fuente: http://www.cs.rochester.edu/

Los retos fundamentales a los que se enfrenta la evolución de este tipo de redes son minimizar el consumo con el fin de aumentar la autonomía y aumentar las prestaciones de las motas, con el fin de aumentar los servicios que estas redes pueden ofrecer.

Estas redes suponen un conjunto de motas que se comunican de manera inalámbrica WSN y que, por los entornos en los que operan, pueden estar geográficamente muy distribuidas e incluso resultar de difícil acceso.

La posibilidad de gestionar estas redes de manera remota, cambiando los parámetros de operación, planificando mantenimientos (gracias a conocer el estado de las motas, detectar motas inactivas, etc.) y, en última instancia, modificando el software de operación supone una mejora considerable en la gestión de estas redes.

Dentro de los mecanismos de acceso remoto ha cobrado una gran importancia el uso de elementos estándar con el fin de reutilizar tanto hardware como software dentro de las organizaciones. Por eso el empleo de navegadores web como base para las aplicaciones, el empleo de HTML (y más recientemente JavaScript y CSS) y el acceso a las aplicaciones con http/https como protocolo de comunicación es habitual en cualquier organización.

En los últimos tiempos, para aumentar tanto la mejora en el uso de recursos como la flexibilidad de acceso remoto, se ha incorporado al desarrollo

de proyectos el concepto de la “nube”. Las aplicaciones se despliegan en entornos centralizados que comparten recursos hardware e incluso software (reduciendo costes), que escalan en función de la demanda (no es ya necesario planificar recursos pensando en el peor caso) y el acceso mediante aplicaciones remotas, típicamente navegadores web, es parte inherente a este tipo de arquitecturas.

El objetivo del proyecto es proponer una solución de este tipo, basada en web y con posibilidad de despliegue en la nube, para una red de sensores.

1.2 Objetivos del Trabajo

Los principales objetivos de este Trabajo Fin de Máster son:

- Adquirir conocimiento sobre sistemas de redes de sensores: arquitectura hardware, sistemas operativos, estándares de aplicación y procedimientos de desarrollo y pruebas.
- Adquirir conocimiento de la plataforma hardware reducida de bajo coste Raspberry PI.
- Adquirir conocimiento de frameworks Python para el desarrollo de aplicaciones web.
- Implementar una aplicación web en entorno Python como interfaz de una red de sensores.
- Implementar un cliente web sencillo que acceda a esta interfaz.
- Implementar una aplicación para aplicación para una red de sensores que reciba comandos desde la interfaz web, ejecutando las acciones seleccionadas y devolviendo los datos requeridos.
- Desplegar y probar conjuntamente estas aplicaciones en un entorno Raspberry PI + OpenMote-CC2538.

1.3 Enfoque y método seguido

Se opta por el desarrollo del sistema desde el inicio, apoyándose en productos y herramientas de uso gratuito y preferiblemente OpenSource. Esto permite reducir costes y colaborar activamente con el desarrollo de la comunidad de software libre.

El desarrollo sigue una metodología en cascada del proyecto. Esto supone que los requisitos del desarrollo están bien establecidos (son básicamente inamovibles) al principio del proyecto y que se puede realizar una planificación detalla del mismo al comienzo. Estos aspectos se cumplen en el ámbito de este Trabajo Fin de Máster.

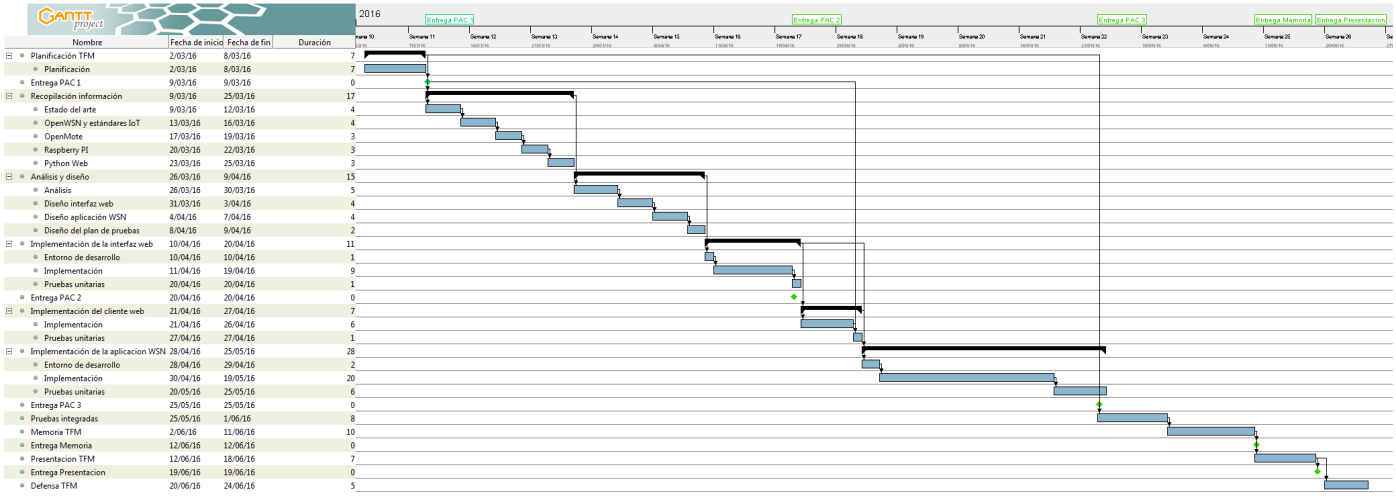
Se ha añadido al proyecto una fase inicial de adquisición de conocimientos de los estándares y productos relacionados con el proyecto con el fin de adquirir y/o afianzar el conocimiento teórico necesario para el mismo.

1.4 Planificación del Trabajo

La siguiente tabla resume los hitos incluidos en el desarrollo del proyecto.

Hito	Fecha	Contenido cubierto
PAC1	09/03/2016	Planificación del proyecto
PAC2	20/04/2016	Recopilación de información Análisis y Diseño Implementación de la interfaz Web
PAC3	25/05/2016	Implementación del Cliente Web Implementación aplicación WSN
Entrega de Memoria	12/06/2016	Redacción de la memoria
Entrega de Presentación	19/06/2016	Preparación de la presentación
Defensa TFM	20-24/06/2016	Defensa del proyecto

1.4.1 Tabla de hitos del proyecto



Nombre	Fecha de inicio	Fecha de fin	Duración
Planificación TFM	2/03/16	8/03/16	7
Planificación	2/03/16	8/03/16	7
Entrega PAC 1	9/03/16	9/03/16	0
Recopilación información	9/03/16	25/03/16	17
Estado del arte	9/03/16	12/03/16	4
OpenWSN y estándares IoT	13/03/16	16/03/16	4
OpenMote	17/03/16	19/03/16	3
Raspberry PI	20/03/16	22/03/16	3
Python Web	23/03/16	25/03/16	3
Análisis y diseño	26/03/16	9/04/16	15
Análisis	26/03/16	30/03/16	5
Diseño interfaz web	31/03/16	3/04/16	4
Diseño aplicación WSN	4/04/16	7/04/16	4
Diseño del plan de pruebas	8/04/16	9/04/16	2
Implementación de la interfaz web	10/04/16	20/04/16	11
Entorno de desarrollo	10/04/16	10/04/16	1
Implementación	11/04/16	19/04/16	9
Pruebas unitarias	20/04/16	20/04/16	1
Entrega PAC 2	20/04/16	20/04/16	0
Implementación del cliente web	21/04/16	27/04/16	7
Implementación	21/04/16	26/04/16	6
Pruebas unitarias	27/04/16	27/04/16	1
Implementación de la aplicacion WSN	28/04/16	25/05/16	28
Entorno de desarrollo	28/04/16	29/04/16	2
Implementación	30/04/16	19/05/16	20
Pruebas unitarias	20/05/16	25/05/16	6
Entrega PAC 3	25/05/16	25/05/16	0
Pruebas integradas	25/05/16	1/06/16	8
Memoria TFM	2/06/16	11/06/16	10
Entrega Memoria	12/06/16	12/06/16	0
Presentacion TFM	12/06/16	18/06/16	7
Entrega Presentacion	19/06/16	19/06/16	0
Defensa TFM	20/06/16	24/06/16	5

1.4.2 Diagrama de GANTT del proyecto

1.5 Breve sumario de productos obtenidos

El proyecto consistirá en los siguientes productos:

- Documentación:
  - Memoria del proyecto, que incluirá entre otros:
    - Resumen de tecnologías y productos involucrados
    - Arquitectura y análisis de la aplicación
    - Diseño técnico
    - Especificación de la pruebas
  - Presentación del proyecto
- Software
  - Cliente web, codificado en lenguajes de programación web (HTML5, CSS, JavaScript)
  - Servidor web, codificado en Python
  - Aplicación para motas de la red de sensores, codificada en C

1.6 Breve descripción de los otros capítulos de la memoria

Capítulo 2: Otros trabajos realizados. Recopilación de la información sobre estado del arte en cuanto a interfaces web de redes WSN, incluyendo productos comerciales, productos ya desarrollados y proyectos similares.

Capítulo 3: Estándares. Recopilación de información de los estándares que afectan a la IoT.

Capítulo 4: Tecnologías y productos. Recopilación de información de las tecnologías implicadas así como de los productos hardware y software que se van a utilizar en el desarrollo del proyecto.

Capítulo 5: Análisis. Propuesta de solución, análisis y arquitectura de la misma.

Capítulo 6: Diseño. Diseño detallado de la solución propuesta y de las pruebas a realizar.

Capítulo 7: Conclusiones. Conclusiones sobre el trabajo realizado y posibles líneas de avance futuro.

Capítulo 8: Glosario. Términos y acrónimos relevantes que aparecen en la memoria.

Capítulo 9: Bibliografía. Bibliografía consultada.

Capítulo 10: Anexos. Otros elementos de interés en el desarrollo del proyecto.

2. Otros trabajos realizados

Existen algunas soluciones tanto de investigación como comerciales que abordan el problema de dotar a una red WSN de una interfaz remota de algún tipo. En este capítulo se pretende hacer un breve recorrido por algunas de ellas para analizar la solución que se va a desarrollar en este Trabajo Fin de Máster de manera comparada frente a estas.

OpenRMS y jWebDust: en [1] se propone la integración de estos dos sistemas ya existentes para dotar a una red WSN de una gestión remota.

OpenRMS es un gestor de administración remoto de código abierto, que se engloba dentro de los conocidos como EMS (Enterprise Managment System). Orientado a la facilidad de uso y despliegue, define agentes abstractos y multiplataforma que permiten la gestión de las estaciones remotas procesando los comandos enviados por los usuarios.

jWebDust es un entorno de red de sensores que divide la red en dos partes: las motas, en las que se ejecuta TinyOS, y el resto de los elementos de red donde se ejecutan aplicaciones Java. Las motas forman parte de la capa de sensores (Sensor Tier) mientras que el resto de elementos se organizan en capas según su funcionalidad (datos, lógica, presentación). Permite manejar motas de diferentes tipos agrupadas en redes virtuales independientes, cada una con su propio centro de control.

Se propone la integración de estos dos elementos ya existentes exponiendo el API de jWebDust de modo que sea compatible con los agentes OpenRMS. De este modo OpenRMS proporciona la interfaz y la definición de tareas (entre ellas la actualización de software en las motas), que son ejecutadas en jWebDust.

TinyDB [2], desarrollado por la Universidad de Berkeley, es una interfaz de acceso a una red de sensores de tipo SQL. Representa la red de sensores como una base de datos relacional y permite el acceso a la misma mediante comandos tipo SQL (select, etc.). En las motas de la red se ejecuta TinyOS y el software de las motas lo proporciona TinyDB sin necesidad de programación adicional.

Los metadatos corresponden a los atributos (sensores y parámetros hardware/software internos) y los comandos que se ejecutan en TinyOS.

TinyDB se encarga de la gestión de la red manteniendo tablas internas con los nodos de la red y el enrutamiento.

La responsabilidad de quien despliega la red es cargar TinyOS y el software de TinyDB en las motas y escribir la aplicación que desee usando el API Java que proporciona TinyDB. También se proporciona Tiny Application Sensor Kit (TASK), un conjunto de aplicaciones Java sobre TinyDB para facilitar el despliegue de la red WSN.

MoteWorks [3] es una un conjunto de aplicaciones desarrollado por Crossbow para facilitar el despliegue de redes de sensores.

En las motas se ejecuta TinyOS y Xmesh, que es el software propietario encargado de la gestión de la comunicación entre las motas y el servidor. Los usuarios pueden incluir aplicaciones propias ya que se incluye un IDE para nesC (una versión del lenguaje de programación C pensado para tener en cuenta las limitaciones de memoria de las redes de sensores, orientado a componentes y especialmente diseñado para trabajar sobre TinyOS).

Un servidor se encarga de actuar de puente entre los clientes y actuar como caché de los datos de los sensores.

También se proporciona una interfaz cliente encargada de la gestión de la red denominada MoteView.

MoteWorks soporta diferentes tipos de motas y una serie de comandos para facilitar el despliegue, entre ellos el comando *flash* para grabar una imagen en la mota.

Cougar [4] es una aproximación similar a TinyDB desarrollado por la Universidad de Cornell, en el sentido que ve la red de sensores como una base de datos. No obstante, Cougar pretende que el sistema resida directamente en la motas, creando la imagen de un nodo único virtual sin que exista un servidor central real. Esta capa de gestión de datos distribuidos permitiría que el sistema creciese al crecer tanto el número de motas como la potencia de cálculo de las mismas.

Además Cougar propone optimizar los algoritmos de enrutamiento teniendo en cuenta los patrones propios de las redes de sensores en lugar de realizar optimizaciones asociadas a comunicaciones genéricas.

Sensor Web Enablement (SWE) [6] es un conjunto de estándares y buenas prácticas definidos por el Open Geospatial Consortium (OGC) con la intención de facilitar la conexión de diferentes tecnologías relacionadas con las redes de sensores.

Dentro de la especificación SWE se incluyen los siguientes estándares:

- O&M: codificaciones XML para las observaciones y mediciones.
- PUCK: protocolo para recuperar una descripción del sensor, el driver y otra información desde el propio dispositivo,
- SensorML: esquema XML para la descripción de los procesos del sensor y las observaciones.
- SOS: interfaz de servicio web para obtener observaciones y descripciones de los sensores.

- SPS: interfaz de servicio web mediante el cual un cliente puede determinar la viabilidad de la recogida de datos de una serie de sensores
- Modelo de datos SWE: los modelos de datos de bajo nivel para el intercambio de datos relacionados con los sensores.
- Modelo de servicio SWE: tipos de datos para uso común a través de los servicios SWE.

En [5] se propone una adaptación del estándar SWE mediante la semántica REST y el uso de JSON para simplificar el acceso y reducir el volumen de datos. Para ello se propone recubrir casa uno de los servicios web con una interfaz REST que traslade los datos de JSON al XML correspondiente. Además de utiliza una semántica JSON reducida para reducir aún más la sobrecarga del empleo de XML.

TWIST [7] es un entorno de test creado por la Universidad Técnica de Berlín que facilita las pruebas de la arquitectura y las aplicaciones sobre las motas en un entorno de gestión centralizado. No obstante la arquitectura se basa en USB y no en conexiones inalámbricas para facilitar el despliegue de software en las motas, por lo que no se puede tomas como ejemplo para aplicaciones reales. Con el fin de facilitar el aumento tanto del número de motas como del alcance de la red basada en HUBs USB, se utilizan nodos intermedios (o supernodos) como una capa intermedia entre el servidor y las motas. La arquitectura incluye un servidor que provee de una interfaz web que se alimenta de una base de datos relacional en la que se almacenan datos de configuración y de las motas registradas.

En la siguiente tabla se recogen las ventajas e inconvenientes de cada solución.

Solución	**Inconvenientes**	**Ventajas**
OpenRMS jWebDust	S.O. TinyOS Trata de manera diferente motas diferentes	Basada en aplicaciones ya existentes. Incluye interfaz para el usuario
TinyDB	S.O. TinyOS Requiere desarrollar la interfaz	Estándar SQL para la realización de la interfaz Aplicación de las motas incluida
MoteWorks	S.O. TinyOS Comercial	Solución completa y extensible. Incluye interfaz para el usuario
Cougar	S.O. TinyOS Requiere desarrollar la interfaz	Sin necesidad de nodos adicionales Aumento de prestaciones al mejorar el HW de las motas Optimizaciones de enrutamiento
SWE	A desarrollar, sólo se	Estándar OGC

	presenta el estándar. Basado en servicios web SOAP	
SWE/REST	A desarrollar.	Simplificación de SWE
TWIST	Entorno de pruebas Basado en USB	Gestión centralizada

2.1.1 Tabla resumen de las soluciones de interfaz remota WSN analizadas

Como puede comprobarse en la tabla anterior, la mayor parte de las soluciones adoptan TinyOS como sistema operativo (puede verse una breve descripción de TinyOS al final del punto 4.1) para la aplicación de las motas y no se basan en estándares (salvo SWE que es en sí mismo un estándar) sino en software propio.

TinyDB se presenta como la solución más madura dentro del ámbito de las posibilidades OpenSource. Esto, junto al hecho de que el desarrollo sobre TinyDB se basa en un estándar como SQL, hace que a se presente como la mejor solución a aplicar en la mayor parte de los casos.

Por otro lado la solución MoteWorks como solución madura y extensible (que permite al usuario adaptarla a su propio uso) es una propuesta adecuada si se opta por soluciones comerciales.

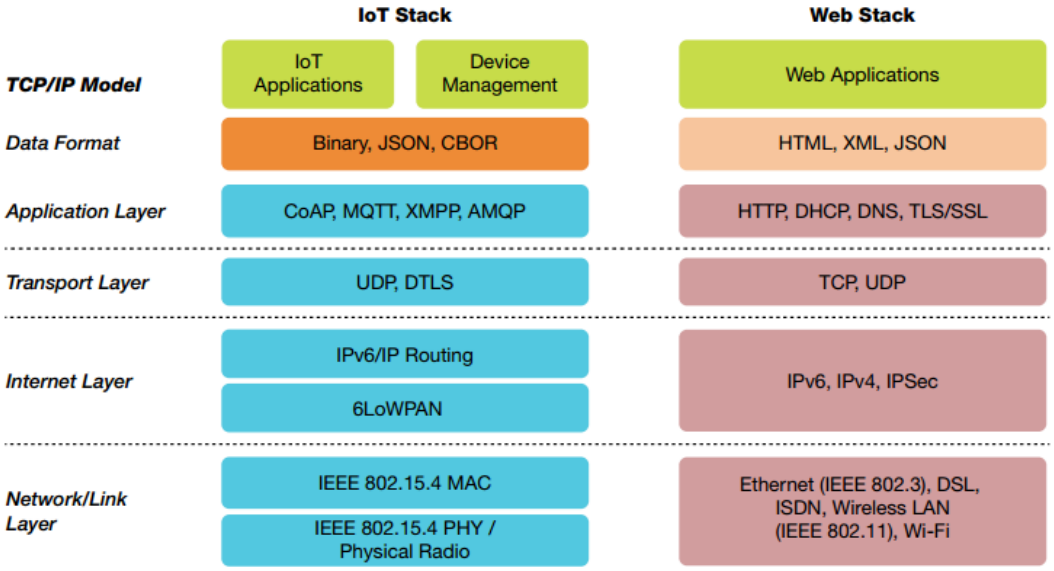
El desarrollo de este Trabajo Fin de Máster pretende adoptar una solución más general, basada en estándares, sistemas OpenSource no comerciales y tecnologías actuales, que haga que cualquier mota sobre cualquier sistema operativo que implemente estos estándares pueda ser gestionada a través de la interfaz de manera uniforme.

Este objetivo tiene una limitación inevitable, y es que parte de la aplicación se desarrolla sobre la mota, por lo que la adaptación de dicha aplicación al S.O. escogido o a las capacidades del HW es necesaria. Puesto que la solución que se presenta se basa en OpenWSN, en el mejor de los casos esta adaptación supondrá simplemente recompilar la aplicación para la plataforma HW seleccionada teniendo en cuenta las limitaciones de la misma en cuanto a sensores, información de estado, etc. Si se opta por un S.O. distinto, la adaptación de la aplicación sería necesariamente más costosa.

Sin embargo la parte de la interfaz con el usuario no necesita ser adaptada y sólo será necesario cargar los elementos de configuración, como los sensores no contemplados hasta ese momento, para proceder a la gestión de las nuevas motas.

3. Estándares de la IoT

En este capítulo se pretende realizar un resumen de los estándares definidos para las aplicaciones sobre redes WSN y que por tanto estarán de manera explícita o implícita involucrados en este Trabajo Fin de Máster.



3.1.1 Pila protocolos tradicional y de IoT
Fuente : https://www.linkedin.com/pulse/emerging-open-standard-protocol-stack-iot-aniruddha-chakrabarti

3.1 IEEE 802.15.4

Este estándar, publicado en 2003, define las capas físicas (PHY) y de acceso al medio (MAC) para redes de bajo consumo y en las que los datos generados por unidad de tiempo son relativamente bajos (por ejemplo WSN, transmisiones poco frecuentes y con pocos datos en cada una de ellas) frente a las redes tradicionales inalámbricas como WiFi. Otros objetivos de la norma, además de garantizar bajos consumos, son conseguir dispositivos tecnológicamente sencillos y con precios de fabricación y operación muy bajos (del orden de euros).

La norma define dos tipos de nodos posibles, pero es frecuente encontrar que no existe esta diferencia entre los dispositivos reales, aunque se diseñe la red de modo que cada elemento tenga un rol asignado.

- Dispositivos de función reducida o Reduced Function Device (RFD), con menos recursos a nivel hardware y software y que actúan como dispositivos final (los sensores de la red WSN).
- Dispositivos de función completa o Full Function Device (FFD), que pueden desempeñar cualquier rol dentro de la red: dispositivo final, coordinador de nodos o coordinador de la red.

La capa física busca un buen compromiso entre velocidad de transmisión, alcance y consumo de energía, suponiendo redes que se despliegan en el interior de un edificio. Aunque se definen diferentes frecuencias de operación, la más habitual es la banda ISM a 2.4GHz. Para lograr los objetivos propuestos utiliza modulación O-QPSK (Offset-Quadrature Phase-Shift Keying), con 2Mbps de velocidad física y 250Kbps de velocidad real, ya que cada bit de datos se codifica mediante 8 chips (bits físicos). Para la transmisión se utiliza DSSS (Direct Sequence Spread Spectrum) que divide el ancho de banda en una serie de canales (16) separados de tal forma que no interfieran entre sí (canales de 2MHz separados 5MHz). El uso de esta técnica logra entre otras cosas mayor inmunidad a desvanecimientos en frecuencia, mejor resistencia a interferencias externas e interferencias inter-símbolo y mejor aprovechamiento del ancho de banda. Las funciones de la capa física son entre otras:

- Activación y desactivación del transceptor radio.
- Detección de energía en el canal actual.
- Evaluación de canal libre mediante acceso múltiple por detección de portadora con prevención de colisiones (Carrier Sense Multiple Access with Collision Avoidance, CSMA-CA).
- Selección del canal en frecuencia.
- Transmisión y recepción de datos.

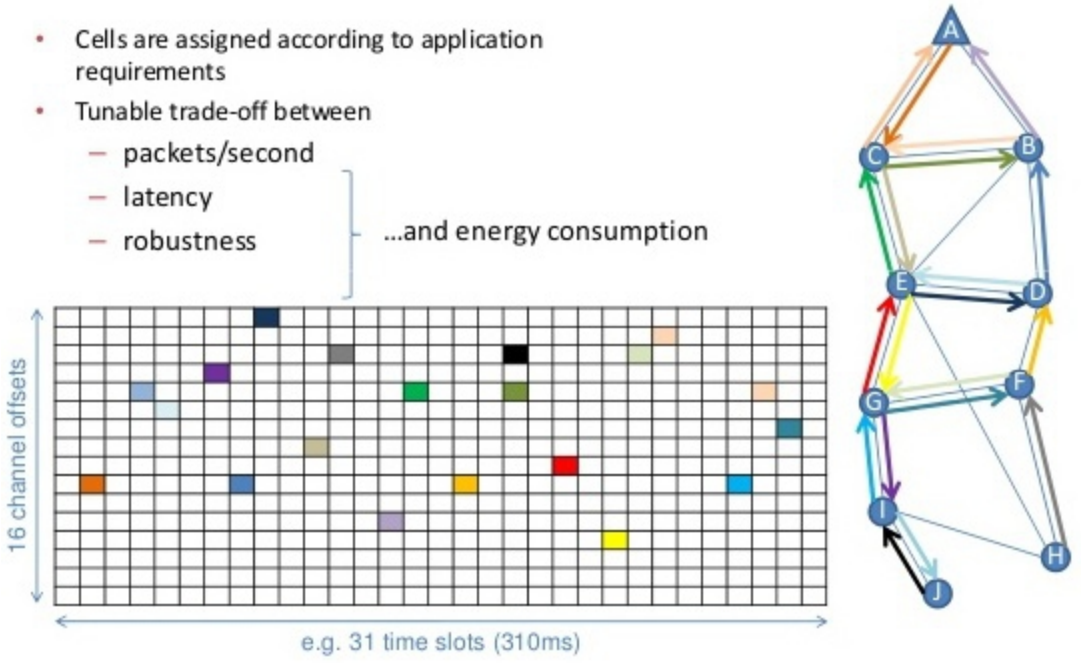
La capa de acceso al medio MAC es la responsable de la gestión de los canales de radio, proporcionando el enlace entre las capas superiores y la capa física. Las funciones de la capa de acceso al medio son entre otras:

- Generación y gestión de tramas beacon (baliza).
- Soporte de mecanismos de seguridad a nivel de dispositivo.
- Implementación de CSMA-CA como mecanismo de acceso al canal.
- Gestión y mantenimiento del mecanismo GTS (Guaranteed Time Slot).
- Provisión de enlace fiable entre dos nodos.

Desde el punto de vista de la transmisión, el receptor que escucha el canal esta percibiendo ruido cuando el transmisor emite un preámbulo de 128μs para que el receptor sepa que se va a producir la transmisión. Después emite el delimitador de inicio de trama (Start of Frame Delimiter, SFD) para indicar el comienzo de los datos. El primer byte indica la longitud por lo que las tramas son como máximo de 127 bytes útiles.

La enmienda **IEEE 802.15.4e** de 2012 tiene como objetivos dar un mejor soporte a los mercados industriales y permitir la compatibilidad con las modificaciones propuestas por el estándar WPAN Chino. Los modos de operación más importantes que incluye la enmienda son:

- **Time Slotted Channel Hopping (TSCH)**, que está diseñado para entornos industriales con necesidades de consumo de energía reducido y robustez frente a interferencias alta. La estrategia TSCH pasa por un lado por añadir diversidad de frecuencia a otros métodos de diversidad (codificación, modulación, etc.) y por otro lado permite que cada enlace nodo a nodo pueda usar un conjunto de frecuencias propio. Para ello utiliza tramas de balizamiento mejoradas que contienen información temporal para sincronización de la red, información sobre el salto de canal e información de la franja temporal en la que serán transmitidas las tramas y se esperan las confirmaciones. Gracias a que todos los dispositivos comparten la información del tiempo y canales es posible mitigar los efectos negativos de las interferencias y perdidas multi-camino.

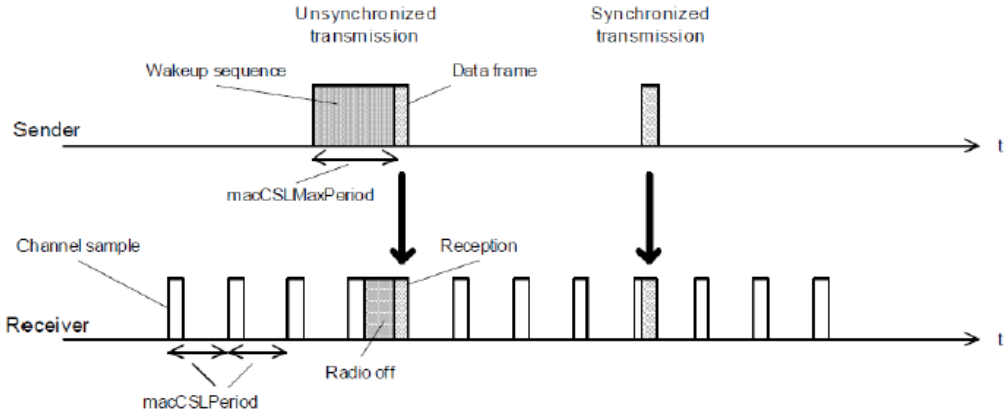


3.1.2 Estructura de canales de TSCH
Fuente: IETF 6TiSCH, a New Standardization Effort to Combine IPv6 Connectivity, Xavier Vilajosana http://es.slideshare.net/IPv6slides/18-xavier

- **Deterministic & Synchronous Multi-channel Extension (DSME)** para aplicaciones que requieran alta disponibilidad, eficiencia, escalabilidad y robustez.
- **Low Latency Deterministic Network (LLDN)** para aplicaciones que requieran muy poca latencia tales como robots, grúas, etc.
- **Radio Frequency Identification Blink (RFID Blink)** para la comunicación con dispositivos a partir de un identificador que cada uno posee como identificación, localización o seguimiento de objetos y personas.
- **Asynchronous Multi-Channel Adaption (AMCA)** para redes de gran tamaño y dispersión geográfica.

Además dentro de la enmienda se define el protocolo Low Energy (LE) que permite a los dispositivos operar bajo un ciclo de trabajo de la fracción del 1%. Los modos de operación relacionados son:

- **Coordinated Sampled Listening (CSL)** que incluye un ciclo de espera controlado por el atributo *macCSLPeriod*, que define el tiempo entre muestreos del canal. Pasado este tiempo se activa el nodo y si no se detecta actividad en el canal, se vuelve a repetir el ciclo de espera. Si se detecta una trama de wake-up y el nodo no debe procesarla, se pasa a estado inactivo el tiempo Rendezvous Time (RZ Time, el tiempo que falta hasta la transmisión de datos, incluido en la trama wake-up) y el tiempo equivalente a un ciclo de transmisión de datos y tras esto vuelve a comenzar el ciclo de espera. Si la trama wake-up debe ser procesada por el nodo, este pasa a estado inactivo el RZ Time y se activa tras este para recibir la trama de datos.



3.1.3 Esquema de transmisiones CSL
Fuente: [9] (presentación)

- **Receiver Initiated Transmission (RIT)** se basa en que es la capa de aplicación del receptor la que periódicamente pregunta al servidor si hay datos pendientes de transmitir. Con el fin de ahorrar peticiones se puede informar al servidor de repeticiones de las escuchas de modo que con una sola pregunta al servidor se cubre un mayor periodo de tiempo. Es muy eficaz a la hora de ahorrar energía pero añade más latencia que la estrategia CSL y además no soporta comunicación multicast.

La enmienda IEEE 802.15.4e también define los Information Element (IE) como un mecanismo extensible para incluir información de sincronización, estado de la red, etc. e incluir nuevas informaciones en versiones futuras. Son campos simples que incluyen únicamente un identificador, la longitud y el contenido. Pueden incluirse en la trama MAC a nivel de cabecera (si han de ser procesados por la propia capa MAC) o en los datos (si están destinados a capas superiores).

3.2 IETF 6LoWPAN

6LowPAN es el acrónimo de *IPv6 over Low power Wireless Personal Area Network* y es la capa de adaptación en entre la capa MAC de IEEE 802.15.4e y la capa superior IPv6.

En general toda red requiere una capa de adaptación a la capa IP que es la que permite que los paquetes de información circulen entre redes heterogéneas; además las redes IoT tienen una serie de características propias que las diferencias del resto: bajo consumo, alimentación por baterías, tiempos de espera sin transmisiones, baja tasa de datos, dispositivos de bajo coste, topologías mesh, etc. Uno de los problemas inmediatos que resuelve la especificación es que la mínima MTU (Maximun Transmision Unit) de IPv6 típica es de 1480 bytes mientras que en redes IEEE 802.15.4e de 128 bytes; sólo la cabecera del paquete IPv6 ocupa 40 bytes.

Las principales funciones de la capa 6LoWPAN son:

- Apilamiento de cabeceras: los datagramas LoWPAN contienen como prefijo un apilamiento de cabeceras encapsuladas.
- Compresión de cabeceras: se establecen métodos de compresión que permiten comprimir las cabeceras IPv6 y además añadir información de otras capas encima de la información IPv6 en una sola trama. Define HC1 como técnica optimizada en comprimir paquetes IPv6 en la cabecera (suprimiendo campos comunes que se repiten) y HC2 como método de compresión para la capa de transporte.
- Fragmentación: fragmentación y ensamblaje de tramas para todas aquellas tramas IPv6 que no puedan ser contenidas completamente en una sola trama IEEE 802.15.4.en fragmentos que incluirán su propia cabecera de fragmentación.
- Soporte de enrutado para redes mesh: define una cabecera mesh que contiene las direcciones del inicio y fin (inmutable) y otra cabecera con la información del siguiente salto para llegar al destino (se va modificando a medida que se va localizando el siguiente salto), creando de esta manera dos cabeceras.

6TiSCH es el grupo encargado de establecer las normas para adaptar TSCH a 6LoWPAN, reutilizando en la medida posible los protocolos existentes para minimizar el periodo de adaptación y alcanzando los niveles que se pueden requerir en un entorno industrial en lo que refiere a latencia, jitter, escalabilidad y fiabilidad.

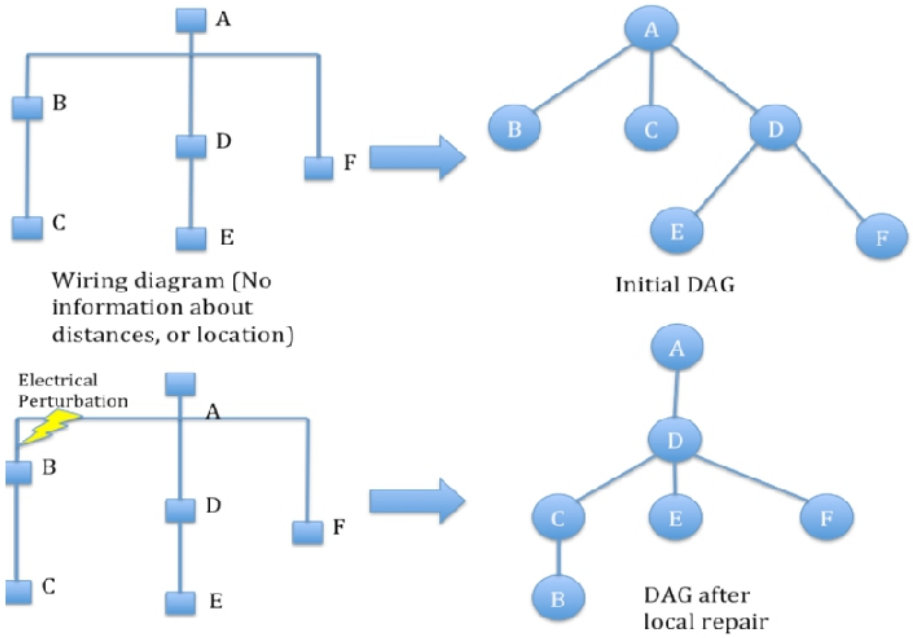
Aunque el grupo sigue trabajando ya se pueden ver algunos aspectos de la arquitectura, que incluye una nueva capa 6top entre 6LoWPAN y TSCH. 6top se basa en 6P (6top Protocol) que es el mecanismo que permite a nodos vecinos añadir/eliminar celdas en la planificación TSCH, y SF (6top Scheduling Function) que es la función que dentro de 6top implementa la decisión de que celdas deben eliminarse/añadirse.

3.3 IETF RPL

RPL (Low-power and lossy networks Routing Protocol) es el protocolo de enrutamiento de la pila de protocolos IoT, del tipo vector distancia (como RIP Routing Information Protocol en las redes TCP/IP tradicionales). RPL esta optimizado para redes en las que un los nodos realizan transmisiones pocas veces a un numero limitado de nodos.

RPL construye un gradiente en la red en la que a cada nodo se le asigna un rango. Este gradiente forma una árbol dirigido acíclico (DODAG, Destination Oriented Directed Acyclic Graph) en la que los nodos de menor rango se denominan LBRs (Low power and lossy network Border Router). Los árboles se forman mediante mensajes DIO (DAG Information Object), y si un nodo decide unirse al árbol reenvía este mismo mensaje a sus nodos hijos, aumentando así el rango.

Los árboles requieren un mantenimiento ya que la red cambia debido a la movilidad, calidad de los enlaces o desaparición de nodos en la red. Por ello el protocolo RPL ajusta ocasionalmente los árboles mediante un contador, evitando de este modo muchas actualizaciones redundantes en redes densamente pobladas. Si se detectan problemas graves de enrutado el contador se resetea para regenerar el árbol.



3.3.1 Ejemplo de reconstrucción del grafo RPL
Fuente: RPL: The IP routing protocol designed for low power and lossy networks, IPSO Alliance http://www.ipso-alliance.org/wp-content/media/rpl.pdf

Las comunicaciones habituales son comunicaciones:

- Multipunto-Punto (MP2P) o comunicaciones hacia arriba (upwards) en las que los nodos envían comunicaciones al LBR (por ejemplo sensores de una red WSN enviado información a un servidor). Suponen recorrer el

árbol siguiendo nodos vecinos de menor rango y, ocasionalmente, vecinos del mismo rango (si no existe un nodo vecino de menor rango).

- Punto-Multipunto (P2MP) o comunicaciones hacia abajo (downwards) del LBR a los nodos. Se basan en rutas pre-construidas y para ello los nodos envían mensajes DAO (Destination Advertisement Object) a los LBR, informado de sus nodos padres e hijos. Tiene un modo de trabajo denominado Storing Mode en el que determinados nodos de la red almacenan tablas de rutas y la ruta final se construye en base a las rutas parciales proporcionadas en cada uno de estos nodos.

3.4 IETF CoAP

Constrained Application Protocol (CoAP) es un protocolo de aplicación diseñado para redes M2M diseñado para entornos de baja energía y que, como 6LoWPAN, soportan paquetes de pequeño tamaño, con el fin de reducir el exceso de datos de HTTP; no obstante CoAP no es sólo un HTTP comprimido sino que añade características propias de las redes M2M. Está estandarizado a través de la RFC 7252 de Junio de 2014.

Las características básicas de CoAP son:

- Protocolo Web teniendo en cuenta las limitaciones de los entornos M2M.
- Usa UDP pero añade mecanismos para entrega fiable unicast y multicast.
- Mensajes asíncronos.
- Cabecera reducida y baja complejidad de parseo.
- Usa URI y tipos de contenido.
- Mapeo simple HTTP, permitiendo construir proxy que permitan el acceso a recursos CoAP a través de HTTP o que interfaces simples HTTP puedan canalizarse sobre COAP.
- Seguridad basada en Datagram Transport Layer Security (DTLS).

La primera diferencia con HTTP es que al utilizar UDP los mensajes son asíncronos por lo que los roles de cliente y servidor se difuminan y los dispositivos normalmente presentan ambos roles.

Aunque es un protocolo único, se divide en 2 capas:

- Capa de mensajes: se encarga de definir el formato del mensaje
- Capa de petición/respuesta: se encarga de definir el intercambio de mensajes

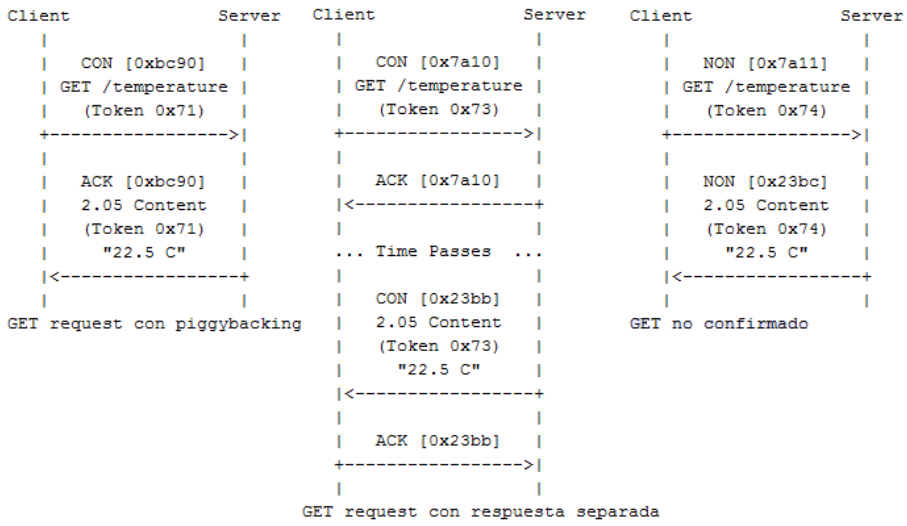
CoAP define 4 tipos de mensajes:

- CON: mensaje confirmado, lleva informado en la cabecera un ID de mensaje de 2 bytes que se utiliza para capturar la confirmación y para evitar duplicados en caso de retransmisiones.

- NON: mensaje no confirmado, no requiere respuesta.
- ACK: confirmación a un mensaje CON. Puede incluir los datos de la respuesta en el mensaje (piggybacking).
- RST: mensaje de reseteo, por ejemplo cuando el receptor no es capaz de procesar el mensaje recibido.

Una transacción CoAP esta soportada a través de un Token incluido en la cabecera, de hasta 8 bytes. Los modelos de comunicación CoAP son:

- Confirmado con piggybacking: en el ACK del servidor va el id del mensaje CON original, el token original y los datos de la respuesta.
- Confirmado sin piggybacking: en el ACK del servidor va el id del mensaje CON original, sin token ni datos; posteriormente el servidor responde con otro mensaje CON con otro id, el token original y los datos de respuesta y el ACK del cliente incluye el id de este segundo mensaje.
- No confirmado: el cliente envía un mensaje NON con un token y el servidor responde con otro mensaje NON con el mismo token.



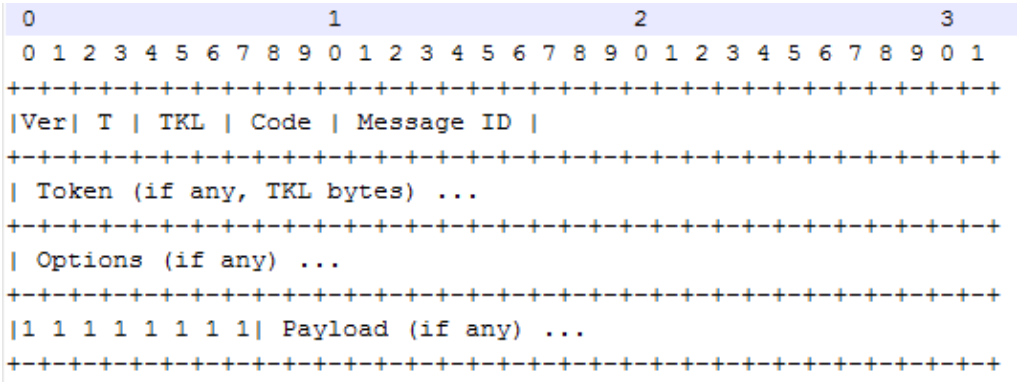
3.4.1 Comunicaciones CoAP

Las retransmisiones para garantizar fiabilidad se gestionan mediante un timeout inicial aleatorio y un contador que al inicio es 0; si pasa el timeout sin recibir confirmación ACK o resteo RST se retransmite el mensaje y se duplica el timeout y se incrementa el contador, hasta que este llega al máximo de retransmisiones. Si se alcanza el máximo o se ha recibido un mensaje RST, la transmisión se considera fallida. Este mecanismo de crecimiento exponencial del timeout para retransmisiones se utiliza como mecanismo de control de flujo para prevenir la congestión de la red.

Los campos básicos de la cabecera del mensaje CoAP son:

- Ver es la versión de CoAP, actualmente 1.
- T es el tipo de mensaje: CON (0), NON (1), ACK (2), RST (3).

- TKL es la longitud del Token en bytes 0-15, pero 9-15 están reservadas así que la longitud máxima es 8 bytes.
- Code es el código de la respuesta tipo c.dd donde c es la clase y dd el detalle.
- Los campos de opciones se forman concatenando las opciones, cada una formada por un código, la longitud y el valor. Deben ir ordenadas por el código y no se incluye el código como tal sino un Delta que permite calcularlo en base al código anterior.



3.4.2 Cabecera CoAP

La especificación CoAP solo define un tamaño máximo de mensaje, aunque lo ideal sería que un mensaje CoAP se incluyese en un único datagrama UDP.

CoAP define un subconjunto de las operaciones HTTP, en una arquitectura REST, de semántica similar a sus equivalentes HTTP:

- GET (consulta), seguro (no ejecuta acción de modificación en el servidor) e idempotente (ejecutarlo repetidas veces no supone cambios en el estado del servidor).
- PUT (modificación/creación) y DELETE (borrado), no seguros e idempotentes.
- POST (creación/modificación/borrado) no seguro y no idempotente.

Los códigos de respuesta son igualmente un subconjunto de los códigos HTTP, con tres clases 2 (correcto), 4 (error cliente) y 5 (error servidor). En la respuesta 2.XX es donde se refleja mayor diferencia frente a HTTP: mientras que en HTTP una respuesta correcta es normalmente 2.00 "OK", en CoAP se definen respuesta específicas para GET (2.05 "content" o 2.03 "valid"), para la creación con PUT o POST (2.01 "created"), para la modificación con PUT o POST (2.04 "changed") y para el borrado con POST o DELETE (2.02 "deleted").

CoAP define el conjunto de opciones posibles. Algunas de ellas son:

- Content-Format: CoAP limita el número de formatos (Media Types)) posibles a text/plain;charset=utf-8 (valor por defecto), application/link-

format, application/xml, application/octet-stream, application/exi y application/json.

- Accept: establece la preferencia de formato del cliente
- Max-Age: tiempo máximo que se puede cachear una respuesta hasta que deje de ser válida.
- ETag: identificador local de un recurso cuya representación puede cambiar.
- Condicionales IF-Match y If-None-Match: indican al servidor condiciones a comprobar para ejecutar la operación pedida. If-None-Match es útil para evitar crear múltiples veces el mismo recurso en caso de clientes actuando en paralelo.

Las URIs CoAP son similares a las HTTP del tipo coap[s]://host [:puerto] / path [? query].

Al realizar el descubrimiento automático de servidores, el puerto por defecto es 5683, 5684 para coaps, aunque en redes 6LoWPAN es más eficiente usar un puerto en el rango 61616-61631. En caso de no conocer la dirección puede enviar un mensaje multicast *"All CoAP Nodes"* a la dirección IPv6 FF0X::FD.

En el descubrimiento automático de recursos, el servidor debe soportar la petición *GET /.well-known/core* que devuelve la lista de todos los recursos (por ejemplo sensores) disponibles.

La seguridad se basa en el empleo de DTLS (Datagram Transport Layer Security) similar al TLS de HTTP.

CoAP es en parte un subconjunto de HTTP por lo que un proxy CoAP-HTTP es directo. Un proxy HTTP-CoAP debe tener en cuenta que hay métodos HTTP no soportados (TRACE, OPTIONS, CONNECT deben generar un error y HEAD debe tratarse como GET), la diferencia en los códigos de respuesta y que existen cabeceras HTTP no soportadas como opciones en CoAP.

CoAP logra un significativo ahorro de consumo frente al empleo de HTTP. En experimentos realizados se observa que una transacción en un servidor CoAP sobre Contiki supone un consumo de 0.774mW frente a un consumo de 1.333mW en un servidor HTTP en el mismo entorno, y que la transacción CoAP implica154 bytes por los 1451 de la petición HTTP equivalente **[14]**.

3.4 OASIS MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo de colas (orientado a mensajes) tipo publicador/subscriptor ligero, para comunicaciones entre máquinas M2M, diseñado inicialmente por IBM y que actualmente es un

estándar de OASIS. MQTT tiene una topología en estrella donde los publicadores y consumidores se suscriben a brokers que son los encargados de redirigir los mensajes de unos a otros.

El funcionamiento se basa en tópicos (topics); cada mensaje se envía a uno de estos tópicos y los clientes pueden estar suscritos a varios simultáneamente. Los tópicos están organizados jerárquicamente (por ejemplo /edificio1/planta1/sensor1/temperatura) y la suscripción admite comodines de modo que se puede suscribir a un nivel (/edificio1/planta1/sensor1/+) o a todos los niveles (/edificio1/planta1/#).

MQTT tiene tres modos de servicio:

- Sin garantía de entrega
- Entregar al menos una vez, se cumple si un cliente recibe el mensaje, aunque pueden recibirlo varios.
- Entregar solo una vez

Además se puede solicitar al broker que mantenga los mensajes, de modo que si un cliente se suscribe al tópico después de que el mensaje se enviara, aún podrá recibir el mensaje.

Inicialmente diseñado para TCP, se añadió un mapeo UDP para dispositivos de bajo consumo y un servidor de indexado de tópicos de soporte.

3.5 Impacto de los estándares IoT en el Trabajo Fin de Máster

Puesto que este trabajo se centra en la interfaz web para la red WSN, CoAP será el que más incidencia tenga en el desarrollo y por tanto ha sido recogido en mayor detalle.

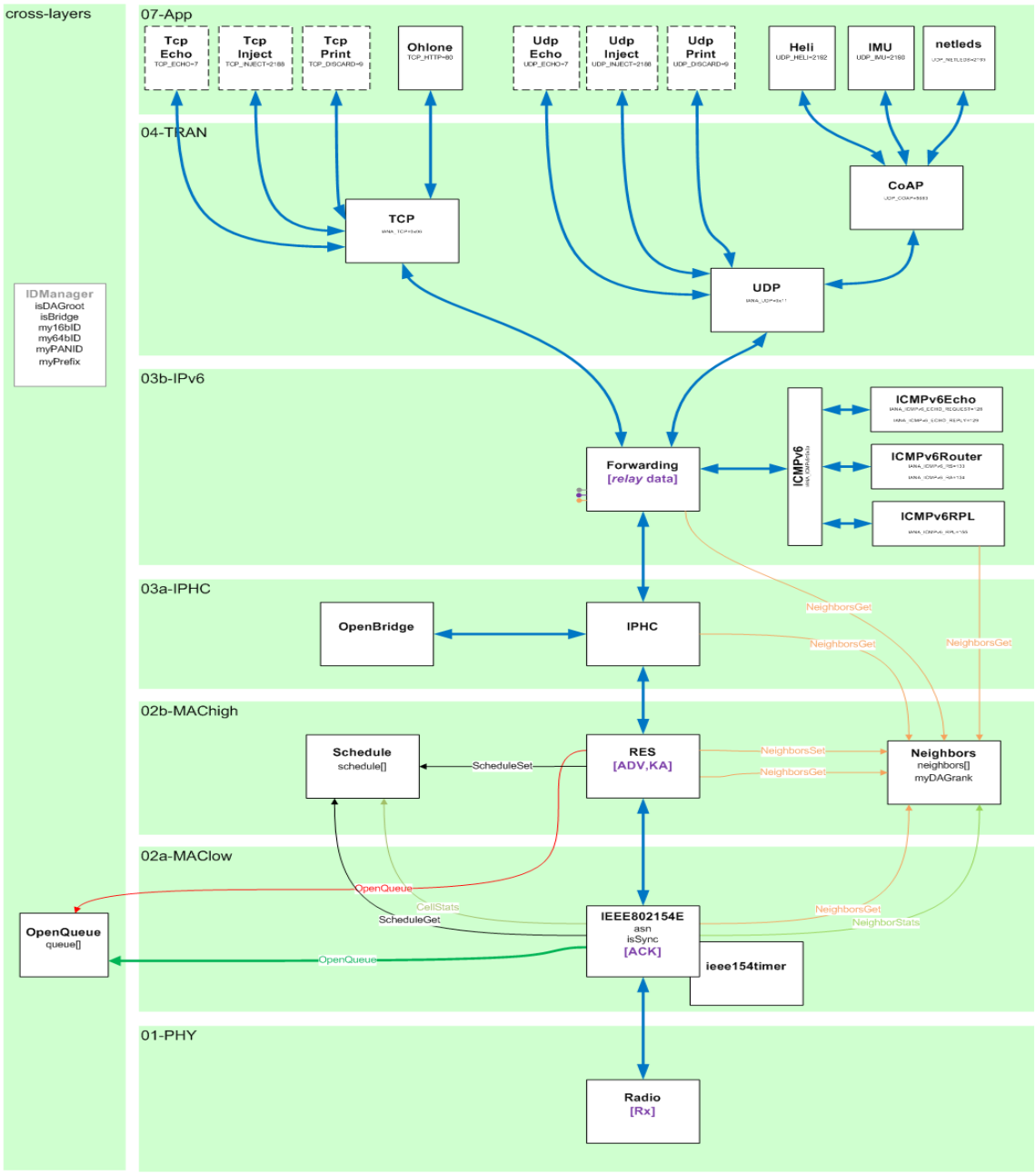
Se han analizado los tipos de mensajes y las comunicaciones a las que dan lugar, diferenciando claramente entre el ID del mensaje (identificación de la confirmación a un mensaje) y el Token (identificación de la transacción petición / respuesta). También se ha analizado la semántica REST, los códigos de respuesta, el formato de las URLs y los formatos de contenido aceptados en CoAP; todos estos aspectos serán fundamentales a la hora de diseñar las interacciones CoAP y los mensajes que se utilizaran para las comunicaciones.

No obstante se ha pretendido realizar un recorrido por los estándares más importantes en cumplimiento del objetivo planteado para este Trabajo Fin de Máster de adquisición de conocimiento sobre redes WSN.

4. Productos y tecnologías

En este capítulo se pretende realizar un resumen de los diferentes productos y tecnologías que se van a utilizar para el desarrollo del Trabajo Fin de Máster. No es el objetivo profundizar es gran medida en ninguna de ellas, pero si recoger sus aspectos fundamentales y el motivo por el que se han escogido.

4.1 OpenWSN



4.1.1 Pila de protocolos OpenWSN
Fuente [15]

OpenWSN es un proyecto de Software libre, bajo licencia BSD, iniciado por Thomas Watteyne de la Universidad de Berkeley como una implementación de la pila de protocolos IoT, con soporte para múltiples plataformas hardware (soportan tanto arquitecturas de 16 bits cómo de 32 bits, incluidas las de tipo ARM Cortex-M).

La característica que lo diferencia de otros como Contiki es que implementa TSHC en la capa de acceso al medio; es por tanto la primera implementación OpenSource de la arquitectura completa IEEE 802.15.4e.

El núcleo de OpenWSN inicialmente era el sistema OpenOS, que es el planificador principal de las interrupciones hardware y de los temporizadores. Determina las tareas a ejecutar en base a una prioridad y, a medida que las tareas van siendo ejecutadas a través de una función callback asociada a la tarea, esta se va eliminando de la lista de tareas pendientes. Cuando la lista queda vacía el microcontrolador pasa un estado latente a la espera de nuevas interacciones que conlleven nuevas tareas.

Desde 2015 hay una adaptación de OpenWSN para qué pueda utilizar FreeRTOS como núcleo de la implementación del protocolo de la pila de protocolos. FreeRTOS es un sistema operativo de tiempo real para dispositivos embebidos, bajo licencia GPL, diseñado para ser simple y sencillo, fácil de portar, con soporte para para ejecuciones multi-hilo (y los mecanismos necesarios de sincronización como semáforos, etc.). Incluye también un modo de operación para aplicaciones de bajo consumo. La diferencia fundamental respecto al uso de OpenOS es que se permite al usuario definir tareas de FreeRTOS qué pueden ser parte de la aplicación sin interferir en la actividad de OpenWSN.

El diseño del entorno OpenWSN es un conjunto de capas que se añaden unas encima de otras para proporcionar el protocolo completo. Además proporciona algunas funciones adicionales como el asignado automático de direcciones de 64 bits que requiere IEEE 802.15. 4 lo que permite utilizar radios Atmel que no disponen de este tipo de identificador, y también la capacidad de modificar fácilmente la potencia de salida.

OpenWSN también proporciona un conjunto de herramientas alrededor del propio proyecto.

- OpenVisualizer es una aplicación desarrollada en Python que se ejecuta en un equipo y que permitirá interactuar con las motas WSN conectadas. La comunicación con cada nota se llevaba a cabo a través el puerto serie y muestra la información relevante del estado interno de cada mota y su conectividad. Además permite la interacción con las aplicaciones que se ejecutan dentro de la mota.
- OpenSIM es un software especial encargado de simular múltiples motas sobre entornos Windows o Linux.
- CoAP, se proporciona un módulo CoAP desarrollado en Python y que se puede usar externamente como cliente y servidor, además de incluir soporte CoAP en el software que se ejecuta en las propias motas.

Otras sistemas operativos que se pueden utilizar en redes WSN son:

- **Contiki**, que es uno de los S.O. de software libre con mayor penetración en dispositivos embebidos. Está diseñado para permitir que dispositivos sencillos (de bajo coste y bajo consumo) ejecuten aplicaciones con interacción completa con redes IP (incluyendo IPv4 y los protocolos IoT como CoAP). Incluye un cargador dinámico de módulos, lo que permite modificar el comportamiento de los sistemas incluso después de su implantación, y un simulador de red para facilitar la tarea desarrollo y depuración de errores previa al despliegue final en el dispositivo.
- **RIOT**, qué se basa en una arquitectura micro-kernel similar a Linux, pero estaba adaptado a dispositivos embebidos, con bajo consumo y limitada capacidad de memoria y procesamiento. Incluye soporte para programación multi-hilo, con menos de 25 bytes por hilo, capacidad de funcionamiento en tiempo real y permite la gestión de memoria dinámica facilitando la programación, lo que le hace muy atractivo para desarrolladores.
- **TinyOS**, es sistema operativo para dispositivos inalámbricos de baja potencia, lo que lo hace muy adecuado para WSN. Está escrito en nesC como un conjunto de tareas y procesos que cooperan. Está liberado con licencia BSD y es el más maduro ya se inicia en el año 2000. Los programas se construyen como componentes (algunos son abstracciones del HW) y se conectan entre sí usando las interfaces. TinyOS proporciona interfaces propios para la comunicación, enrutamiento y almacenamiento entre otros. Se basa en peticiones asíncronas FIFO (first in first out) de corta duración para mantener alta concurrencia con una sola pila, lo que supone desarrollar uniendo muchos controladores de eventos. Para aplicaciones más complejas se ha diseñado una librería multi-hilo.

4.2 OpenMote-CC2538

OpenMote-CC2538 es una plataforma hardware en torno al procesador CC2538 de Texas Instruments que es un procesador con arquitectura ARM Cortex-M3 de 32 bits, lo que normalmente se conoce como un SoC (System on Chip).

El microprocesador incluye 32KB de RAM, 512 KB de flash y corre a 32MHz. La radio opera a 2.4GHz dentro de la banda ISM y se incluyen además otros periféricos comunes tales como GPIOs (pines de entrada/salida de propósito general), ADC (conversores analógico/digital) y temporizadores.

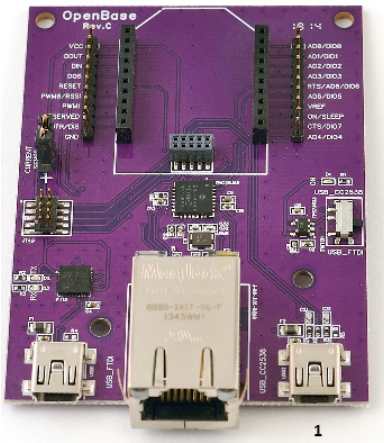
La arquitectura de OpenMote es modular y el procesador CC2538 es el núcleo de esta arquitectura. Además del microprocesador se incluye en la placa un convertidor-reductor de corriente continua, un cristal a 32MHz que utiliza como reloj del procesador, otro a 32.7KHz que es el reloj de tiempo real, además de los botones, el conector de antena, etc. OpenMote es completamente compatible con el factor de forma Xbee lo que significa que

puede ser integrado desde un ordenador personal utilizando XBee Explorer Dongle de Sparkfun.

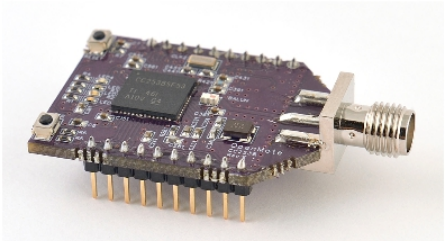
Los otros elementos de la arquitectura modular de OpenMote son:

- OpenBase, que se conecta al núcleo a través de una cabecera XBee y que tiene como propósitos fundamentales:
    - permitir la programación y depuración de código utilizando puntos de ruptura a través de una sonda con conector estándar ARM JTAG de 10 pines
    - permitir la comunicación con el ordenador personal a través del puerto serie o USB, habilitando la depuración utilizando la sentencia printf de C
    - permitir la comunicación de la red de sensores con Internet a través de una interfaz Ethernet de 10/100MB por segundo sin la necesidad de un servidor adicional.
- OpenBattery es la placa interfaz con el núcleo, también conectada a través de una cabecera Xbee, que incluye el soporte para baterías externas (lo que permite operar autónoma) y tres tipos de sensores: un sensor de temperatura y humedad (SHT21), un sensor de aceleración (ADXL346) y un sensor de luz (MAX44099), que se conectan todos ellos al núcleo CC2538 través de un bus I2C

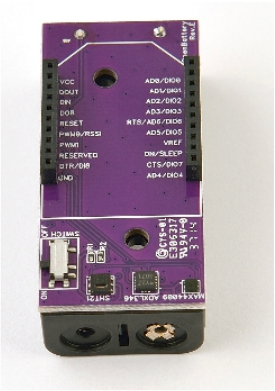
OpenMote tiene licencia de Hardware libre tipo CERN Open Hardware License v1.2.



1



2



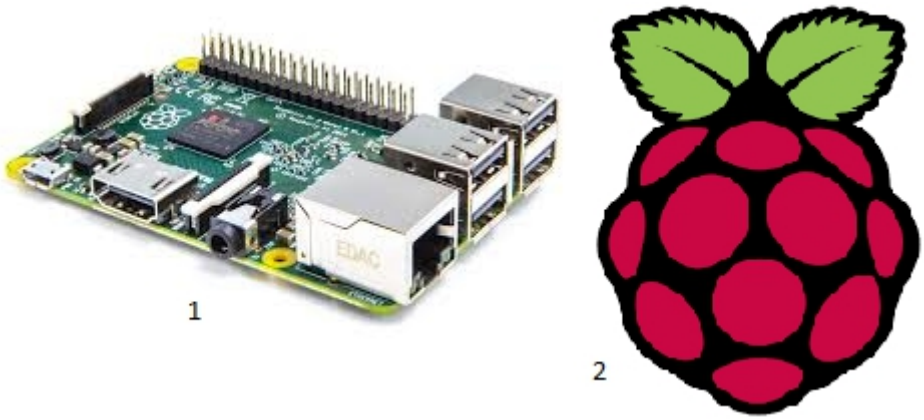
3

4.2.1 Hardware OpenMote-CC2538
1 OpenBase 2 OpenMote 3 OpenBattery Fuente [8]

4.3 Raspberry PI

Raspberry PI es lo que se denomina un Single Board Computer (un ordenador que tiene todo incluido en una única placa) del tamaño aproximado de una tarjeta de crédito, desarrollado por la Fundación Raspberry PI en un intento de promover la enseñanza de la tecnologías de las comunicaciones en las escuelas y en los países en vías de desarrollo.

En la primera generación de Raspberry PI del 2012 incluía un modelo básico A y un modelo avanzado B, que evolucionaron a los modelos A+ y B+. La serie Raspberry PI 2 es de 2015 e incluye únicamente el modelo B, aunque posteriormente se ha desarrollado un modelo denominado Raspberry PI Zero de prestaciones y costes limitados. La generación Raspberry PI 3 es de febrero de 2016.



4.3.1 Raspberry PI
1 Hardware Raspberry PI2 2 Logo Fuente: https://www.raspberrypi.org/

Los precios están entre los 20€ y los 35€, salvo el modelo Raspberry PI Zero que está en torno a los 5€.

El diseño general incluye un procesador ARM de 32 bits, un procesador gráfico incluido también entre la placa, además de memoria RAM compartida con el procesador gráfico. Para el almacenamiento del sistema operativo se utiliza una tarjeta SD o MicroSD externa y se incluyen puertos USB (el número depende del modelo), un puerto HDMI (para vídeo) y un conjunto de pines de entrada salida de propósito general GPIO (cuyo número varía también con el modelo). No cuenta con carcasa ni fuente de alimentación. El procesador es un Broadcom BCM2835 a 700MHz para Raspberry PI, un procesador BCM2836 a 900MHz (arquitectura ARM Cortex-A7) para Raspberry PI 2 y un procesador BCM2837 a 1.2GHz (arquitectura ARM Cortex-A53) para Raspberry PI 3. La memoria RAM de Raspberry PI 1 era de 256MB (ampliadas posteriormente a 512MB) mientras que Raspberry PI 2 y 3 cuentan con 1GB, salvo el modelo Raspberry PI Zero que tiene 512MB y corre a 1GHz. No cuenta con un reloj de tiempo real, aunque sé que se le puede añadir externamente mediante un conector I2C; además incluye otros dispositivos como cámara de vídeo o cámara infrarroja. Inspirado en el modelo de Arduino Shields (que añade periféricos que se conectan a la capa principal apilándose unos encima de otros) se ha incluido el accesorio HAT (hardware attached on top) que permite añadir otros periféricos.

La Fundación Raspberry PI promueve el uso de Python como lenguaje programación fundamental pero se pueden utilizar otros lenguajes cómo C++, Java, etc.

En cuanto a sistemas operativos, el recomendado es Raspbian que se mantiene independientemente por la Fundación Raspberry PI y está basado en Linux Debian, pero se pueden encontrar otros sistemas operativos como versiones de Ubuntu y OpenLec, incluso Windows 10. Raspbian incluye pre-instalado otro tipo de software como Minecraft, Mathematica Wolfram Language, etc.

En cuanto al tipo de licencia, no es hardware libre pero si es de uso libre tanto a nivel educativo como particular; el software en cambio sí es OpenSource dado que el sistema operativo estás basado en Linux.

4.4 Python web frameworks

Python es un lenguaje de programación, creado por Guido van Rossum en 1991 y mantenido por la Python Software Foundation, que debe su nombre a los humoristas británicos Monty Python.

Es un lenguaje de programación interpretado cuyo principal objetivo es el código legible y formó parte de la iniciativa a iniciativa "Computer Programming for Everybody" (CP4E) que pretendía hacer la programación más accesible a más gente y lograr un conocimiento básico de programación.

Es un lenguaje de programación multiparadigma, ya que incluye orientación a objetos, programación imperativa y programación funcional, usa tipos dinámicos (no es necesario declarar los tipos de las variables) y es multiplataforma.

Aunque no es un lenguaje tan usado como Java o PHP en entornos web, tiene múltiples frameworks que facilitan el desarrollo de aplicaciones web; en https://wiki.python.org/moin/WebFrameworks puede verse una lista bastante completa de los mismos.

Los más populares son probablemente Django, Flask y Pyramid.

- Flask, que se define como un "microframework", nace a mediados de 2010 (con la ventaja de aprender de frameworks previos) y está orientado a proyectos web sencillos. No incluye herramientas de bootstraping (creación de estructura inicial de proyecto) y usa blueprints (versiones reducidas de la aplicación inicial Flask) para dividir una aplicación en módulos.
- Django es un framework muy completo, nacido en 2006 y orientado a facilitar la creación de aplicaciones web complejas conociendo solo Python. Su filosofía es incluir todo dentro del framework (templates para páginas web, patrones de rutas, ORM para acceso a base de datos, incluso un módulo de administración de base de datos).

- Pyramid es un framework intermedio entre los anteriores, nacido en 2006 y que, aunque como Django está orientado a aplicaciones complejas, deja en manos del desarrollador las decisiones en cuanto a base de datos, rutas, etc. que Django ya incluye.

Los tres frameworks incluyen templates para generar páginas html con información propia; Django incluye su propio sistema de templates, Pyramid de acuerdo a su filosofía lo deja al desarrollador y Flask incluye un sistema propio basado en el de Django.

4.5 SQLite

SQLite es una biblioteca que implementa una base de datos relacional transaccional, con soporte para SQL sin necesidad de un proceso en un servidor. El código de SQLite es de dominio público, sin necesidad de licencia.

SQLite lee y escribe directamente en un archivo en disco la base de datos completa, es multiplataforma (tanto la base de datos como el fichero, que se puede copiar a otra plataforma), compacta (menos de 500KB) y con poco consumo de memoria, lo que la hace una opción habitual en dispositivos de prestaciones limitadas (como teléfonos móviles o dispositivos embebidos), configuración de aplicaciones, etc.

Frente a la mayoría de las bases de datos relacionales que usan tipos estáticos en las columnas, SQLite permite almacenar cualquier tipo de dato en (casi) cualquier columna, lo que la hace muy adecuada para trabajar con lenguajes de programación dinámicos (sin tipos) como Python. Igualmente los registros tienen longitud variable, frente a otras bases de datos donde se declara la longitud de las columnas.

SQLite soporta transacciones ACID aunque sea interrumpido por fallos del sistema.

Está escrita en ANSI-C; además entornos como Python proveen de manera nativa implementaciones y APIs de acceso a SQLite.

A modo de curiosidad, los ficheros de SQLite incluyen, en lugar de licencia al ser de dominio público, una bendición:

May you do good and not evil
May you find forgiveness for yourself and forgive others
May you share freely, never taking more than you give

4.6 Tecnologías WEB

HTML (HyperText Markup Language) es el lenguaje de marcado para la creación de páginas web. Es un estándar a cargo del W3C (World Wide Web Consortium).

Desarrollado por Tim Berners-Leeen el CERN en 1990 como una ampliación de SGML, su última versión es HTML5 de 2014.

El objetivo del lenguaje es crear páginas con contenidos de imágenes, videos, etc. sin incrustar estos contenidos, sino haciendo referencia al recurso que representa dicho contenido. Además incluye enlaces o links a otras páginas para permitir una navegación dinámica.

El programa encargado de visualizar la página se denomina navegador web o browser y es el encargado de mostrar la página adecuadamente uniendo los contenidos.

JavaScript es un lenguaje interpretado, estandarizado como ECMAScript. La última versión ECMAScript6 es de 2015 y aún no está soportada íntegramente por los navegadores web.

JavaScript tiene una notación similar a Java y C (aunque no es tipado), orientado a objetos y basado en prototipos.

Está pensado para ejecutarse en el entorno del navegador web, aunque existen versiones que se ejecutan en el entorno del servidor web y, más recientemente, como lenguaje de un entorno de servidor (node.js).

El uso principal dentro del navegador web es interactuar con el DOM (Document Object Model) de la página web, permitiendo dinámicamente cambiar la apariencia e información de la misma.

AJAX (Asynchronous JavaScript And XML) es un procedimiento que permite invocar recursos del servidor en segundo plano, mientras el navegador web mantiene el primer plano para el usuario. Cobre gran importancia a raíz de la creación de aplicaciones RIA (Rich Internet Applications).

Por cuestiones de seguridad los navegadores suelen implementar la política "Same Origin Policy" que no permite acceder a servidores distintos del que se descargó la página. Para evitar este problema los servidores pueden incluir en las respuestas la cabecera "Access-Control-Allow-Origin" indicando que orígenes pueden acceder a la información o * si es accesible desde cualquier origen.

JSON (JavaScript Object Notation) es un formato de intercambio basado en la notación de objetos JavaScript, pero que actualmente es un estándar que se puede utilizar en cualquier entorno.

Se suele comparar con XML, frente al que más compacto, sencillo y flexible, lo que lo hace más atractivo para flujos de información mientras que XML es más adecuado para documentos, con datos más jerarquizados y anidados.

Los elementos básicos de JSON son los objetos, listas y datos primitivos (cadenas, números, booleanos), con los que se construye la información.

BSON (Binary JSON) es una especificación para la representación de objetos JSON en formato binario para ser más eficiente y compacto; además extiende los tipos soportados por JSON.

CSS (Cascading Style Sheets) es un lenguaje usado para definir la presentación de una página web. Se considera una buena política separar presentación (CSS) de contenido (HTML) para facilitar cambios en la misma.

Es un estándar a cargo del W3C y la última versión es CSS3, que es modular y estos módulos se han ido liberando en diferentes fechas.

Permite definir la presentación de manera corporativa (todas las páginas tienen la misma presentación porque usan las mismas CSS).

4.7 Librerías WEB

AngularJS es un framework JavaScript de código abierto, desarrollado por Google, y que ha cobrado gran popularidad en los últimos años en el desarrollo de aplicaciones SPA (en una sola página).

El objetivo de AngularJS es proporcionar el patrón Modelo Vista Controlador (MVC) en aplicaciones del navegador web.

AngularJS proporciona un sistema de data-binding que sincroniza modelo y vista (cuando una variable presentada en la vista se modifica en el modelo, por ejemplo por la invocación de un recurso del servidor o por ser un valor modificado por el usuario, el cambio se refleja automáticamente en la vista). Esto elimina la manipulación directa de DOM desde JavaScript.

Además proporciona:

- un sistema de comunicación con el servidor para invocaciones AJAX.
- un mecanismo (directivas) para crear componentes reutilizables
- internacionalización de las aplicaciones.
- validación automática de formularios.

Todos estos mecanismos permiten simplificar el código y facilitar el test automático de aplicaciones, para lo que AngularJS incluye algunas facilidades adicionales como Mocks de servicios de servidor.

Bootstrap (o Twitter Bootstrap) es un framework web de código abierto para desarrollo de aplicaciones web, centrado en la parte de presentación, desarrollado inicialmente por Twitter.

Consiste en una serie de hojas de estilos CSS para facilitar la creación de botones, formularios, alertas, ventanas emergentes, etc. También incluye estilos para facilitar el "responsive design" o presentación adaptable a diferentes tipos de dispositivos (PCs, tabletas, teléfonos móviles).

También incluye una serie de componentes adicionales tales como pestañas, carruseles de imágenes, etc. que requieren también JavaScript.

Jasmine es un framework de pruebas para JavaScript del tipo "behaviour-driven" (guiado por comportamiento) orientado a facilitar el desarrollo de pruebas unitarias, que no requiere de DOM para su ejecución. Es independiente de cualquier otro framework y posee una sintaxis muy clara.

Las funciones básicas son las *suites* que es un conjunto de pruebas unitarias y los *specs* que describen mediante una función la prueba unitaria y donde se comprueba que los resultados del código son los esperados por el desarrollador (expectativas en Jasmine, para que un spec sea correcto todas las expectativas incluidas deben ser cumplidas). Además facilita la inicialización de los datos de prueba (funciones beforeEach y afterEach) antes de cada spec.

Permite anexar suites (describe dentro de describe) para facilitar la ordenación de las pruebas. También permite crear funciones para complejas para validar resultados a interés del desarrollador.

Aunque se distribuye de manera separada, existen muchos entornos como node.js que incluyen herramientas para la automatización de las pruebas usando Jasmine.

Plot.ly es una librería JavaScript OpenSource (inicialmente código cerrado pero abierto en 2015) que, sobre la base de D3, permite la realización sencilla de gráficos en el navegador, utilizando SVG (Sacalar Vector Graphics) para gráficos 2D y WebGL para gráficos 3D. Soporta los gráficos más habituales (tarta, barras, etc.) y otros avanzados como mapas.

Está desarrollada en entorno node.js y dispone de opciones de configuración modificando el código fuente y regenerando la librería.

D3 (Data-Driven Documents) es una librería JavaScript orientada a la manipulación de documentos que facilita la manipulación del DOM y la generación de HTML, CSS y SVG dinámicamente.

OpenLayers es una librería rápida y fácil de usar para inclusión de mapas en páginas web, similar a Google Maps API, OpenSource bajo licencia BSD y actualmente dentro de la Open Source Geospatial Foundation.

La versión actual es la 3. Incluye soporte para fuentes de mapas (Tiled) como OpenStreetMaps, BingMaps, etc. (no GoogleMaps desde la versión 3 por las restricciones propias de Google) y fuentes vectoriales como GeoJSON, KLM y otros. También permite utilizar imágenes estáticas y otras fuentes XY.

Incluye soporte para animaciones, dibujos y marcados sobre el mapa, lo que la hace uno de los frameworks de mapas JavaScript más populares.

4.8 Empleo de estos productos / tecnologías

Se ha optado por el desarrollo de la parte de la aplicación correspondiente a las motas sobre OpenWSN por ser la primera

implementación completa de la pila de protocolos IEEE 802.15.4e, además de tener desarrollado el proyecto para el HW OpenMote-CC2538 sobre el que se pretende desplegar la aplicación final.

En lo que refiere al servidor, se opta por establecer como objetivo del Trabajo Fin de Máster el despliegue sobre Raspberry PI por ser un elemento de bajo coste que fácilmente podría utilizarse como elemento adicional de una red de sensores con este propósito.

Se ha decidido la utilización de Python como lenguaje del lado del servidor puesto que es el recomendado por Raspberry PI; además se mantiene la coherencia con OpenWSN que desarrolla sus los elementos adicionales (OpenVisualicer, implementación de CoAP, ejemplos, tests, etc.) con Python.

Dentro de los frameworks web analizado se opta por Flask al tratarse de una aplicación web poco compleja básicamente REST, que es el tipo de proyecto para el que está pensado Flask.

Como motor para la persistencia de la información del proyecto se opta por el empleo de SQLite como BBDD por su simplicidad, por estar basada en un fichero (sin necesidad de instalación de programas adicionales) y por tener Python incluida una implementación de SQLite de manera nativa.

Para el desarrollo de la interfaz de usuario se ha decidido el empleo de una aplicación JavaScript/REST/SPA por ser la tendencia actual en el desarrollo de este tipo de aplicaciones, delegando parte de la lógica al lado del cliente y descargando al servidor del renderizado de la parte visual, centrándose en la lógica de negocio.

Para este desarrollo de opta por AngularJS por ser el framework más popular, que ha experimentado más crecimiento en los últimos años y por ser un punto intermedio en cuanto a sencillez de uso, facilidad de aprendizaje y capacidades frente a otros frameworks JavaScript como Ember o Backbone.js.

Para la presentación dentro de las páginas se ha optado por el empleo de un framework CSS porque simplifica el desarrollo de los aspectos de presentación y permite conseguir aplicaciones con estética y comportamientos complejos sin un gran conocimiento de CSS; dentro de estos se ha escogido Bootstrap por ser el framework CSS más popular y además por existir un componente para la integración con AngularJS.

Para las pruebas unitarias se ha optado por Jasmine por ser el soporte de pruebas unitarias JavaScript más utilizado y existir abundante documentación y ejemplos, además de ser la opción de AngularJS para sus pruebas unitarias.

5. Análisis

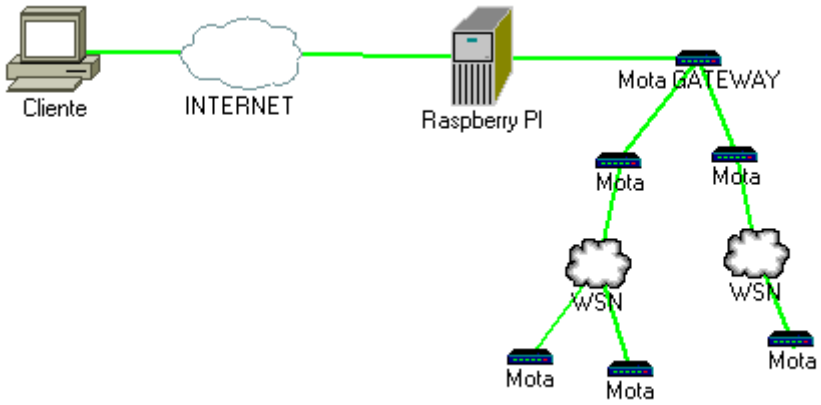
En este capítulo se pretende realizar el análisis completo de la aplicación a desarrollar.

El objetivo de esta aplicación es facilitar la gestión remota de una red de sensores WSN utilizando una interfaz web accesible mediante navegadores estándar y que se pueda desplegar en la nube.

La gestión remota debe permitir el conocimiento del estado de la red, la recuperación de los valores de los sensores, y la modificación de parámetros de configuración que permitan modificar el comportamiento de la misma.

5.1 Arquitectura

En este apartado se define la arquitectura del proyecto.



5.1.1 Diagrama de red del sistema

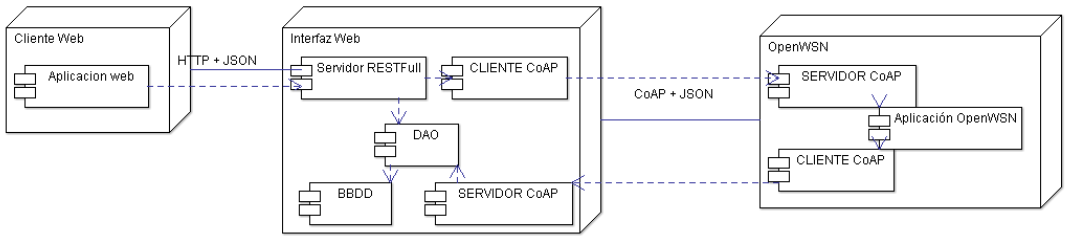
Los elementos básicos son:

- Cliente desde el que se pretende acceder a los datos y/o gestionar la red WSN.
- Servidor (Raspberry PI) encargado de actuar como pasarela a la red WSN
- Mota GATEWAY, mota de acceso a la red WSN
- Motas con sensores. Las motas forman una topología mallada (o topología mesh) inalámbrica en las que para que un nuevo elemento (mota) se una no le hace falta estar dentro de alcance de la mota gateway sino que basta con estar dentro del alcance de otra mota que a su vez estará conectada (directamente o no) a la mota gateway.

Se opta por una arquitectura que incluye un servidor adicional entre la red WSN y los usuarios, lo que dota de mayor flexibilidad a la aplicación y

permite descargar las motas de tareas, facilitando así la consecución de los objetivos de bajo consumo que acompañan a este tipo de redes.

La comunicación entre las motas y el servidor se va a realizar mediante el empleo de protocolos estándar IoT, en concreto CoAP, lo que permite que cualquier mota con cualquier sistema operativo que implemente dichos estándares puede ser incluida para su gestión, con la adaptación de la aplicación en la mota correspondiente. Este elemento es una de las principales ventajas de la solución adoptada frente a otras soluciones OpenSource o comerciales ya existentes.



5.1.2 Diagrama de arquitectura del sistema

En el diagrama de arquitectura se pueden ver en detalle los componentes del sistema:

- Cliente web: equipo con un navegador desde el que el usuario accede al sistema.
    - Aplicación web: la aplicación web son un conjunto de recursos estáticos (HTML, CSS, JavaScript) que se ejecutan en el equipo cliente, aunque su ubicación para la descarga desde el cliente puede ser otra, incluso el propio servidor de la interfaz Web. Se trata de una aplicación SPA con soporte del framework Angular-JS que accede a los datos del servidor vía peticiones AJAX y renderiza estos datos en la vista HTML.
- Interfaz Web: es el equipo que actúa como servidor, en este caso una Raspberry PI con sistema operativo Raspbian. La comunicación entre el cliente Web y la Interfaz Web se realiza mediante protocolos estándar http/https y el intercambio de datos se realiza en formato JSON.
    - Servidor RESTFull: pasarela entre el cliente web y la WSN. Atiende las peticiones de la aplicación web y devuelve los datos obtenidos de la base de datos. Puesto que el lenguaje de programación recomendado en Raspbian es Python, se utiliza uno de los frameworks en este lenguaje para el desarrollo de aplicaciones web, en este caso Flask. La versión de Python que se utilizará es la 2.7.
    - DAO: interfaz entre los otros componentes y la base de datos. Realiza las operaciones de consulta y actualización de la BBDD. También es el encargado de inicializar la base de datos en caso de ser necesario.

    - BBDD: base de datos relacional que almacena en un conjunto de tablas los datos de la aplicación. Por simplicidad se utiliza SQLite, ya que el entorno de ejecución de Python cuenta con una implementación por defecto de esta base de datos relacional.
    - CLIENTE CoAP: cliente encargado de enviar mensajes a las motas de la red WSN.
    - SERVIDOR CoAP: servidor encargado de atender las peticiones de las motas de la red WSN cuando estas envían información de manera autónoma.
- Mota OpenWSN: mota de la red WSN dotada de sensores. La comunicación entre la interfaz web y la mota se realiza mediante CoAP.
    - CLIENTE CoAP: cliente encargado de enviar mensajes al servidor cuando lo requiera la aplicación.
    - SERVIDOR CoAP: servidor encargado de recibir las peticiones del servidor de interfaz y trasladarlas a la aplicación para su ejecución.
    - Aplicación OpenWSN: aplicación que lee los sensores de la mota para obtener los valores, consulta el estado de la mota y gestiona la configuración de la aplicación.

5.2 Parámetros de configuración y de estado

Parámetros de configuración: son aquellos parámetros que pueden alterar el funcionamiento de la aplicación instalada en la mota o el comportamiento de la aplicación web.

Los que afectan a la aplicación instalada en la mota son:

- Tiempo entre muestras: es el tiempo que la tarea que se ejecuta en la mota se activa para recuperar información de sensores y de los parámetros de estatus y la envía al servidor.
- Servidores: servidores a los que se desea cargar la información. Al instalar la aplicación en la mota se cargan los servidores por defecto en ese momento. Este parámetro sirve para actualizar esta información dinámicamente en las motas ya instaladas (y permitir por ejemplo un cambio de servidor) sin que sea necesario reinstalar la aplicación en todas las motas.
- Lista de sensores activos: lista de sensores que se van a leer en la mota y de los que se va a informar a la aplicación web. Desde la interfaz se puede enviar T para indicar todos activos o N para indicar ninguno activo

Además cuando se modifiquen los parámetros, se enviará a la mota una fecha de sincronización (parámetro transparente al usuario) que permitirá conocer si la mota está sincronizada, comparando el valor que devuelve la

mota para este parámetro con el valor almacenado en la aplicación web en la última modificación de los mismos.

Los parámetros utilizados en la aplicación web para mejorar la experiencia del usuario en el uso de la aplicación son:

- Información libre: es un campo de información libre para el administrador (anotaciones, etc.).
- Descripción de la mota: sirve para facilitar la localización de la misma en la interfaz web.
- X,Y: parámetros de localización de la mota en el mapa.

PARÁMETRO	MOTA	WEB	POR DEFECTO
Tiempo entre muestras	Entero en milisegundos	Entero en segundos	300 segundos
Servidores	Array de estructuras IPv6 + puerto	Lista de IPv6: puerto	[BBBB::1]:5386 (openVisualizer)
Lista de sensores activos	Indicador de activo en cada sensor	Lista de ids de los sensores activos	Todos activos
Información libre	NO APLICA	Texto libre	-
Descripción	NO APLICA	Texto libre	ip:puerto de la mota
X,Y	NO APLICA	Posición X e Y en pixel	0, 0

5.2.1 Resumen de los parámetros de configuración

Parámetros de estado: parámetros que informa la mota acerca de su estado; pueden entenderse como sensores que se leen pero que no proporcionan información del ambiente sino de la propia mota. Son:

- Estado batería: carga de la batería de la mota en ese momento.
- Temperatura CPU: temperatura del microcontrolador.

Los parámetros de configuración que afectan a la aplicación en la mota pueden ser comunes a toda la red WSN, por lo que habrá una opción para sincronizar estos parámetros simultáneamente en toda la red.

Los parámetros de configuración se guardan fuera de la mota, ya que se dispone de un almacenamiento persistente. De este modo ante un reinicio de la mota o la sustitución de la misma (siempre que conserve la misma IP) puede restaurarse la configuración de la misma.

5.3 Descubrimiento

Descubrimiento es el proceso por el que la Interfaz Web conoce las motas que forman parte del sistema.

El proceso de descubrimiento va a ser responsabilidad de la mota; cuando una mota se incluye en la red WSN, la aplicación que se ejecuta en la mota tiene definidos por defecto un valor para los parámetros [Tiempo entre muestras] y [Servidores]. Esto implica que va a enviar información a los servidores establecidos de manera automática cada cierto tiempo. Esta información va a servir a la interfaz web para saber que motas están activas en cada momento.

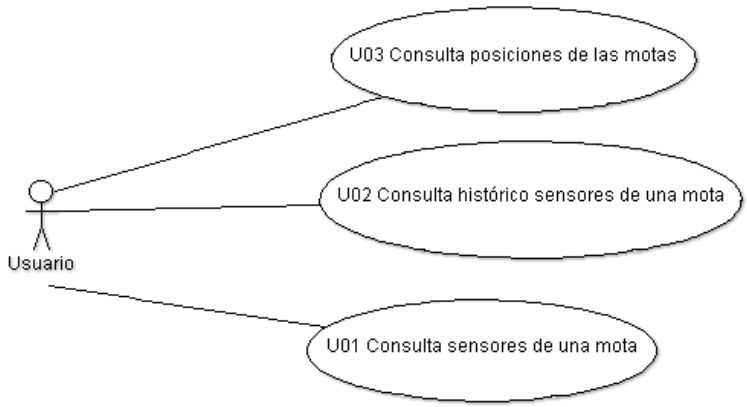
5.4 Casos de uso

En este apartado se pretende describir el funcionamiento de la aplicación a través de los casos de uso de la misma.

Para facilitar el seguimiento se han dividido los casos de uso por funcionalidad. Se incluye un diagrama de casos de uso de la funcionalidad y por cada caso de uso el detalle del mismo cuyo aspecto fundamental es el detalle de la interacción usuario-aplicación (flujo normal).

5.4.1 Casos de uso de consulta

Son los procesos que puede ejecutar un usuario sin permisos adicionales y que permiten conocer los valores de los sensores de las motas, tanto actuales como históricos (con una representación gráfica de los mismos).



5.4.1 Diagrama de casos de uso de consulta

CÓDIGO	U01	**NOMBRE**	Consulta sensores de una mota
DESCRIPCIÓN		El usuario consulta los valores de los sensores de una mota	

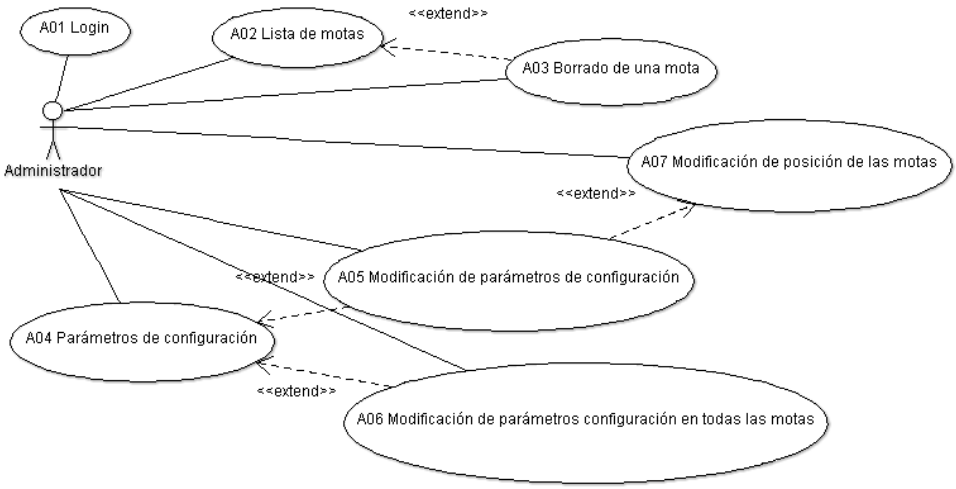
PRECONDICIONES	
FLUJO NORMAL	1/ El usuario accede a la opción de consulta 2/ El sistema muestra la lista de motas disponibles 3/ El usuario selecciona una mota 4/ El sistema muestra la última información de valores de sensores recibida de la mota y la fecha de la misma
ALTERNATIVO	

CÓDIGO	U02	**NOMBRE**	Consulta histórico sensores de una mota
DESCRIPCIÓN		El usuario consulta los valores a lo largo del tiempo de un sensor de una mota	
PRECONDICIONES			
FLUJO NORMAL		1/ El usuario accede a la opción de histórico 2/ El sistema muestra la lista de motas disponibles 3/ El usuario selecciona una mota 4/ El sistema muestra los sensores de la mota 5/ El usuario selecciona un sensor 6/ El sistema muestra la información recuperada y la fecha de generación de la misma. - También muestra la información en formato gráfico	
ALTERNATIVO			

CÓDIGO	U03	**NOMBRE**	Consulta posiciones de las motas
DESCRIPCIÓN		El usuario consulta la posición de la mota sobre un mapa, pudiendo acceder a los valores de los sensores de la mota pulsando sobre el mapa.	
PRECONDICIONES			
FLUJO NORMAL		1/ El usuario accede a la opción de localización 2/ El sistema muestra un mapa con las posiciones de las motas 3/ El usuario selecciona una mota 4/ El sistema muestra la última información de valores de sensores recibida de la mota y la fecha de la misma	
ALTERNATIVO			

5.4.2 Casos de uso de la administración de la red WSN

Mediante la ejecución de estos procesos el usuario con permisos adicionales puede conocer el estado de la red WSN y de sus motas y modificar el comportamiento de la red.



5.4.2 Diagrama de casos de uso de administración de la red

CÓDIGO	A01	**NOMBRE**	Login
DESCRIPCIÓN		El usuario administrador accede al sistema	
PRECONDICIONES			
FLUJO NORMAL		1/ El usuario accede a la opción de login 2/ El sistema muestra el formulario de login con la opción de usuario/contraseña 3/ El usuario introduce la información 4/ El sistema considera al usuario logado	
ALTERNATIVO		4.1/ El sistema muestra un mensaje indicando que el usuario/contraseña no es válido	

CÓDIGO	A02	**NOMBRE**	Lista de motas
DESCRIPCIÓN		El administrador consulta la lista de motas en el sistema	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción de consulta de la administración 2/ El sistema muestra la lista de motas disponibles, junto con la fecha de alta y la fecha de última actualización de información - El sistema muestra al lado de cada mota un mensaje de información si la fecha actual excede en un tiempo de 1 hora la fecha de última actualización - El sistema muestra una alerta si algún parámetro de estado ha alcanzado su valor crítico. - El sistema muestra un aviso si la mota es nueva o está desincronizada respecto a la última actualización de la configuración de la que se tiene constancia. - El sistema muestra al lado de cada mota un botón de borrar.	

	3/ El usuario selecciona una mota 4/ El sistema muestra el valor de los parámetros de estado de la mota
ALTERNATIVO	4.1/ El sistema muestra un mensaje indicando el error producido

CÓDIGO	A03	**NOMBRE**	Borrado de una mota
DESCRIPCIÓN		El administrador elimina una mota del sistema	
PRECONDICIONES		El administrador ha accedido a la lista de motas (A02)	
FLUJO NORMAL		1/ El usuario selecciona la mota y pulsa eliminar 2/ El sistema elimina la información de la mota - El sistema elimina también la información de sus valores	
ALTERNATIVO		2.1/ El sistema muestra un mensaje indicando el error producido	

CÓDIGO	A04	**NOMBRE**	Parámetros de configuración
DESCRIPCIÓN		El administrador consulta los parámetros de configuración de una mota	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción de configuración de la administración 2/ El sistema muestra la lista de motas disponibles 3/ El usuario selecciona una mota 4/ El sistema muestra un formulario con los valores de los parámetros de configuración de la mota	
ALTERNATIVO			

CÓDIGO	A05	**NOMBRE**	Modificación de parámetros de configuración
DESCRIPCIÓN		El administrador modifica los parámetros de configuración en una mota	
PRECONDICIONES		El administrador ha accedido a la lista de parámetros de configuración y ha seleccionado una mota (A04)	
FLUJO NORMAL		1/ El usuario modifica los valores de los parámetros y pulsa guardar 2/ El sistema solicita confirmación 3/ El usuario confirma la solicitud 4/ El sistema modifica el valor de la configuración de la mota	
ALTERNATIVO		4.1/ El sistema muestra un mensaje indicando el error producido	

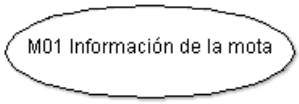
CÓDIGO	A06	**NOMBRE**	Modificación de parámetros configuración en

			todas las motas
DESCRIPCIÓN		El administrador modifica los parámetros de configuración en todas las motas	
PRECONDICIONES		El administrador ha accedido a la lista de parámetros de configuración y ha seleccionado una mota (A04)	
FLUJO NORMAL		1/ El usuario modifica los valores de los parámetros y pulsa modificar en todas las motas 2/ El sistema solicita confirmación 3/ El usuario confirma la solicitud 4/ El sistema modifica la configuración en todas las motas	
ALTERNATIVO		4.1/ El sistema muestra un mensaje indicando el error producido	

CÓDIGO	A07	**NOMBRE**	Modificación de posición de las motas
DESCRIPCIÓN		El administrador modifica los la posición de las motas en el mapa	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción localización en la administración 2/ El sistema muestra el mapa de localización de las motas 3/ El usuario modifica la posición de alguna/algunas motas desplazándolas en el mapa y pulsa guardar 4/ El sistema modifica la posición de las motas en el mapa	
ALTERNATIVO		4.1/ El sistema muestra un mensaje indicando el error producido	

5.4.3 Procesos en la mota

Descripción de los procesos que se ejecutan en la mota para la recolección de información.



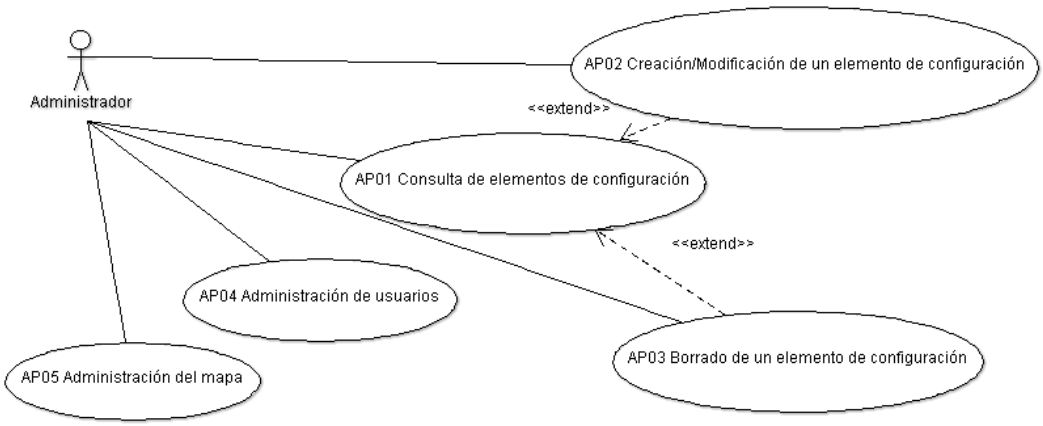
5.4.3 Diagrama de casos de uso procesos mota

CÓDIGO	M01	**NOMBRE**	Información de la mota
DESCRIPCIÓN		La mota envía automáticamente información de los valores de sus sensores y de su estatus	
PRECONDICIONES			

FLUJO NORMAL	1/ La mota cada periodo marcado por [Tiempo entre muestras] envía a la lista de servidores marcada por [Servidores] un mensaje con la información de sus sensores y de los parámetros de estatus 2/ La interfaz web almacena la información recibida en la base de datos
ALTERNATIVO	

5.4.4 Casos de uso de administración de la aplicación

Mediante la ejecución de estos procesos un usuario con permisos adicionales puede administrar la información propia de la aplicación (dar de alta nuevos tipos de sensores, modificar los mensajes que se van a mostrar en caso de alerta, etc.).



5.4.4 Diagrama de casos de uso administración de la aplicación

CÓDIGO	AP01	**NOMBRE**	Consulta de elementos de configuración
DESCRIPCIÓN		El administrador consulta los datos de configuración de una de las tablas maestras (sensores, tipos de parámetros, parámetros)	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción de menú parámetros en la administración de la aplicación 2/ El sistema muestra la lista de las tablas maestras administrables (sensores, tipos de parámetros, parámetros, parámetros de estatus)	
		3/ El usuario selecciona una tabla maestra	
		4/ El sistema muestra los valores actuales de esa tabla	
ALTERNATIVO			

CÓDIGO	AP02	**NOMBRE**	Creación/Modificación de un elemento de configuración

DESCRIPCIÓN	El administrador crea/modifica un elemento de configuración de una tabla maestra
PRECONDICIONES	El administrador ha accedido a los datos de la tabla maestra (AP01)
FLUJO NORMAL	1/ El usuario selecciona un elemento de configuración 2/ El sistema carga los valores del elemento en el formulario 3/ El usuario completa los valores y pulsa aceptar 4/ El sistema modifica el valor del elemento de configuración en la tabla maestra
ALTERNATIVO	1.1/ El usuario introduce directamente los valores del formulario de un nuevo elemento de configuración 4.1/ El sistema crea un nuevo elemento de configuración en la tabla maestra

CÓDIGO	AP03	**NOMBRE**	Borrado de un elemento de configuración
DESCRIPCIÓN		El administrador elimina un elemento de configuración de una tabla maestra	
PRECONDICIONES		El administrador ha accedido a los datos de la tabla maestra (AP01)	
FLUJO NORMAL		1/ El usuario selecciona un elemento de configuración y pulsa eliminar. 2/ El sistema elimina el valor del elemento de configuración en la tabla maestra	
ALTERNATIVO			

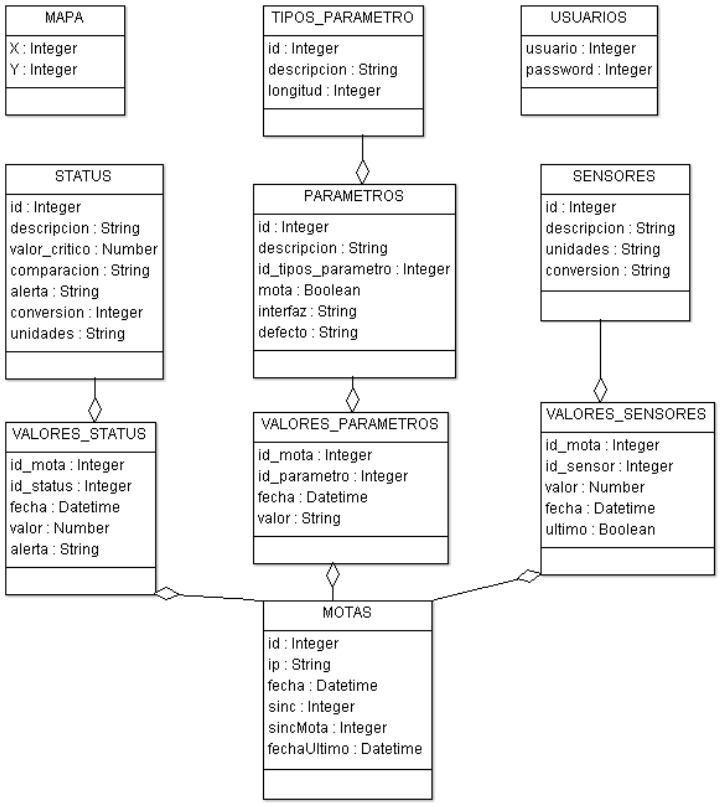
CÓDIGO	AP04	**NOMBRE**	Administración de usuarios
DESCRIPCIÓN		El administrador da de alta, elimina o modifica la contraseña de un usuario administrador	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción de menú usuarios de la administración de la aplicación 2/ El sistema muestra los valores de los administradores dados de alta 3/ El usuario selecciona un administrador 4/ El sistema carga el nombre del administrador en el formulario 5/ El usuario modifica la contraseña y pulsa Aceptar 6/ El sistema modifica la contraseña del administrador almacenándola en forma HASH en el sistema 7/ El usuario selecciona un administrador y pulsa eliminar 8/ El sistema elimina el administrador del sistema	
ALTERNATIVO		3.1/ El usuario introduce directamente los valores del formulario del nuevo usuario	

	6.1/ El sistema crea un nuevo usuario en el sistema

CÓDIGO	AP05	**NOMBRE**	Administración del mapa
DESCRIPCIÓN		El administrador modifica el mapa de localización de las motas	
PRECONDICIONES		El usuario está logado (A01)	
FLUJO NORMAL		1/ El usuario accede a la opción de menú mapa de la administración de la aplicación 2/ El sistema muestra un formulario para subir una imagen del mapa 3/ El usuario completa los datos 4/ El sistema modifica el mapa	
ALTERNATIVO			

5.5 Modelo de datos

En este apartado se va a recoger el modelo de tablas que va a permitir el almacenamiento persistente de la información de la aplicación. Se incluye el diagrama del modelo y por cada tabla el detalle de la misma y de sus atributos y relaciones.



5.5.1 Diagrama de modelo de datos

5.5.1 Tablas maestras

Tablas que almacenan información propia de la aplicación (usuarios, tipos de sensores, etc.).

NOMBRE	MAPA	
DESCRIPCIÓN	Tabla que información sobre el mapa	
ATRIBUTOS	X	Ancho del mapa
	Y	Alto del mapa
OBS.	El mapa se almacena como fichero externo para facilitar su acceso.	

NOMBRE	USUARIOS	
DESCRIPCIÓN	Tabla que los usuarios administradores	
ATRIBUTOS	usuario	identificador del usuario
	password	Contraseña del usuario en SHA512 hexadecimal
CLAVE	usuario	

NOMBRE	SENSORES	
DESCRIPCIÓN	Tabla que almacena los tipos de sensores. El id debe coincidir con el id que envía al aplicación instalada en la motas.	
ATRIBUTOS	id	identificador del sensor en la mota
	descripcion	descripción del sensor
	unidades	unidades en que se mide el valor devuelto por el sensor
	conversion	función de conversión del valor devuelto por el sensor a las unidades de trabajo
CLAVE	id	

NOMBRE	TIPOS_PARAMETRO	
DESCRIPCIÓN	Tabla que almacena los tipos de parámetros que se pueden administrar (descripción y validaciones).	
ATRIBUTOS	id	identificador del tipo de parámetro
	descripcion	descripción
	longitud	longitud máxima del parámetro
CLAVE	id	

NOMBRE	PARAMETROS

DESCRIPCIÓN	Tabla que almacena los que se pueden administrar. El id debe coincidir con el id que espera/envía al aplicación instalada en la motas.	
ATRIBUTOS	id	identificador del sensor en la mota
	descripcion	descripción
	id_tipos_parametro	identificador del tipo de parámetro
	mota	parámetro que aplica a las motas (1) o no (0)
	interfaz	nombre con el que se publica este campo en la interfaz web en la consulta de motas
	defecto	valor que se inserta en la tabla de VALORES_PARAMETROS cuando llega una nueva mota al sistema
CLAVE	id	
RELACIONES	id_tipo_parametro → TIPOS_PARAMETRO(id)	

NOMBRE	STATUS	
DESCRIPCIÓN	Tabla que almacena los parámetros de estado de la mota que se pueden conocer. El id debe coincidir con el id que envía al aplicación instalada en la motas.	
ATRIBUTOS	id	identificador del parámetro de estado
	descripcion	descripción
	valor_critico	valor crítico que genera una alerta
	comparacion	comparación que genera la alerta
	alerta	mensaje de alerta si se produce
	conversion	función de conversión del valor devuelto por el sensor de estado a las unidades de trabajo
	unidades	unidades de medida del valor del sensor
CLAVE	id	

5.5.2 Tablas de valores

Tablas que almacenan información propia de la red WSN. La tabla central será la que recoja las motas de la red; a partir de esta tendremos las tablas que recojan los valores de los sensores, los valores de configuración y los valores de los parámetros de estado de las motas.

NOMBRE	MOTAS
DESCRIPCIÓN	Tabla con las motas que existen en el sistema.

ATRIBUTOS	id	identificador interno de la mota; se asigna automáticamente
	ip	IPv6 de la mota, identificador a nivel de la red WSN
	fecha	fecha de alta de la mota
	fechaUltimo	fecha de la última recepción de información de esta mota
	sinc	sincronización de los parámetros de configuración
	sincMota	sincronización de los parámetros de configuración informada por la mota
CLAVE	id	

NOMBRE	VALORES_SENSORES	
DESCRIPCIÓN	Tabla con los valores de los sensores recibidos de las motas. Puede haber múltiples valores de un sensor por cada mota, los nuevos mensajes recibidos se insertan.	
ATRIBUTOS	id_mota	identificador de la mota
	id_sensor	identificador del sensor
	fecha	fecha del valor recibido, formato
	valor	valor recibido
	ultimo	indicador de último valor recibido para la mota y el sensor
CLAVE	id_mota, id_sensor, fecha	
RELACIONES	id_mota → MOTAS(id) id_sensor → SENSORES(id)	

NOMBRE	VALORES_STATUS	
DESCRIPCIÓN	Tabla con los valores de los parámetros de estado recibidos de las motas. Sólo puede haber un valor de un parámetro de estado por cada mota, los nuevos mensajes recibidos actualizan.	
ATRIBUTOS	id_mota	identificador de la mota
	id_status	identificador del parámetro de estado
	fecha	fecha del valor recibido
	valor	valor recibido
	alerta	mensaje de alerta por alcanzar el valor crítico de un parámetro de estado
CLAVE	id_mota, id_status	
RELACIONES	id_mota → MOTAS(id)	

	id_status → STATUS(id)

NOMBRE	VALORES_PARAMETROS	
DESCRIPCIÓN	Tabla con los valores de los parámetros de configuración de las motas. Sólo puede haber un valor de un parámetro de configuración por cada mota.	
ATRIBUTOS	id_mota	identificador de la mota
	id_parametro	identificador del parámetro de configuración
	fecha	fecha del valor recibido, formato yyyy-MM-dd hh:mm:ss
	valor	valor recibido
CLAVE	id_mota, id_parametro	
RELACIONES	id_mota → MOTAS(id) id_parametro → PARAMETROS(id)	

5.6 Mensajes REST Cliente ↔ Interfaz Web

En este apartado se recogen los mensajes que se van a intercambiar entre el servidor y la interfaz de usuario para la consulta y la administración de la red WSN y de la propia aplicación. Se sigue la semántica REST en la que GET se utiliza para consultas, DELETE para borrado y PUT/POST para la creación o modificación de información.

Mensajes relacionados con el login a la aplicación:

POST /login: Login en la aplicación
- JSON entrada: objeto con
  - usuario: usuario de acceso a la aplicación
  - password: contraseña de acceso a la aplicación

GET /login: Consulta el usuario logado en la aplicación

DELETE /login: Logout de la aplicación

Mensajes de consulta:

GET /sensores: Listado de los sensores dados de alta

GET /parametros: Listado de los parámetros dados de alta

GET /motas: Listado de las motas con la información básica

GET /motas/:id/valores: Listado de los últimos valores de sensores de una mota

GET /motas/:id/valores/:idSensor: Listado de los últimos valores de un sensor en una mota

Mensajes de administración de la red WSN:
GET /adm/motas:Listado de motas para el administrador

DELETE /adm/motas/:id: Borra una mota del sistema

GET /adm/motas/:id/configuracion: Listado de los valores de los parámetros de configuración de una mota para el administrador

GET /adm/motas/:id/status: Listado de los valores de los parámetros de estado de una mota para el administrador

PUT /adm/mota/:id/configuracion: Modifica el valor de parámetros de configuración en una mota
- JSON entrada: array de
    - id: identificador único del parámetro de configuración
    - valor: valor del parámetro de configuración

PUT /adm/motas/configuracion: Modifica el valor de parámetros de configuración en todas las motas
- JSON entrada: array de
    - id: identificador único del parámetro de configuración
    - valor: valor del parámetro de configuración

Mensajes relacionados con la administración de la aplicación:

GET /administracion/:tabla: Consulta los valores de la tabla maestra

POST /administracion/:tabla: Modifica/crea los valores de la tabla maestra
- JSON entrada: objeto con
    - campoX: representa el campo X de la tabla
    - valorX: representa el valor del campo X de la tabla

DELETE /administracion/:tabla/:id: Elimina los valores de la tabla maestra

Mensajes relacionados con el mapa:

GET /mapa: retorna la imagen del mapa

GET /mapa/info: retorna las dimensiones del mapa

POST /mapa: Modifica el mapa
- JSON entrada: objeto con
    - X: ancho de la imagen del mapa

- Y: alto de la imagen del mapa
- mapa: imagen del mapa en Base64

5.7 Mensajes CoAP Mota ↔ Interfaz Web

En este apartado se recogen los mensajes que se intercambia la mota con la aplicación en el servidor, así como la justificación del tipo de mensaje CoAP seleccionado.

El formato de los mensajes será siempre JSON (application/json) y son siempre mensajes POST ya que suponen actualización de información.

Mensaje Mota → Interfaz Web:

POST /uocapp: Información de la mota a la interfaz web

- JSON ENTRADA: *{"s":{"1":…., "2":....}, "e":{"1":….}, "f":'...'}*
  - s: estructura con la información de sensores
    - "1", "2", …: identificadores de los sensores
    - cada id tiene su valor asociado
  - e: estructura con la información de estado
    - "1", "2", …: identificadores de los parámetros de estado
    - cada id tiene su valor asociado
  - f: sincronización de los parámetros

Este mensaje será de tipo NO CONFIRMADO (NON) ya que se pretende reducir la sobrecarga de la red WSN y la información es periódica por lo que la pérdida de un mensaje no supone un inconveniente grave.

Mensaje Interfaz Web → Mota:

POST /uocapp: Modificación de los parámetros de configuración de la mota

- JSON ENTRADA: *{"0":'...', "1":…., "2":....}*
  - "1", "2", …: identificadores de los parámetros de configuración. "0" corresponde al parámetro de sincronización.
  - cada id tiene su valor asociado

Este mensaje será de tipo CONFIRMADO (CON) ya que se trata de información crítica que se envía a la mota para modificar su comportamiento (la perdida del mensaje supone que la mota queda desactualizada a la espera de la actuación del administrador) y es esporádica (no se va a sobrecargar la red WSN).

IMPORTANTE: limitaciones de OpenWSN

Durante el desarrollo se ha comprobado que OpenWSN soporta únicamente paquetes de 127 bytes y que no tiene implementada la fragmentación (ver https://openwsn.atlassian.net/wiki/questions/57475074/how-to-change-the-packet-size-in-openwsn-). Esto obliga a hacer una división lógica de los mensajes anteriores para cumplir con estos límites.

En el caso de los mensajes de la mota se envía cada estructura de datos ("e" de los valores de estado, "s" de los valores de sensores, "f" de sincronización) en un mensaje separado. En el caso de "e" y "s" no más de 3 valores por mensaje.

En el caso de los mensajes hacia la mota, el parámetro de configuración de la lista de servidores se separa enviando un mensaje por cada servidor de la lista en formato "XFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEEEE" donde X es el índice de la lista, F es la Ipv6 extendida sin separaciones (colapsando '0's con el carácter ':') y E es el puerto. Además se envía un mensaje "X-" indicando el último índice informado (por ejemplo si la lista tiene 2 servidores se enviará un mensaje "2-" tras enviar los dos servidores de la lista). El resto de los parámetros se pueden enviar en un único mensaje.

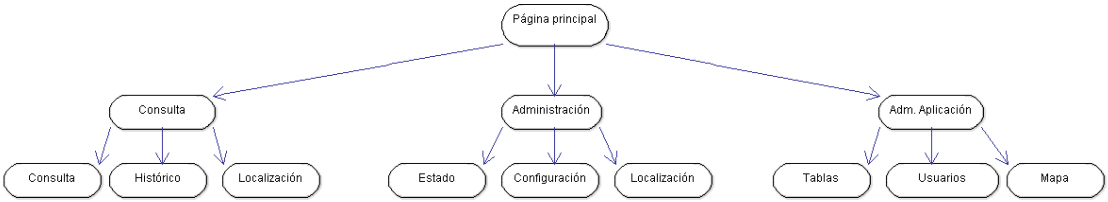
6. Diseño

En este apartado se pretende describir el diseño detallado de los diferentes componentes de la aplicación, identificando

6.1 Diseño del cliente WEB

Para el diseño del cliente WEB se opta por Angular-JS como framework base para el desarrollo de la aplicación SPA.

Además se utiliza el soporte de Bootstrap para los aspectos de presentación (menús, etc.), de D3 y PLOT.LY para la representación gráfica y de OPENLAYERS para la presentación de mapas.



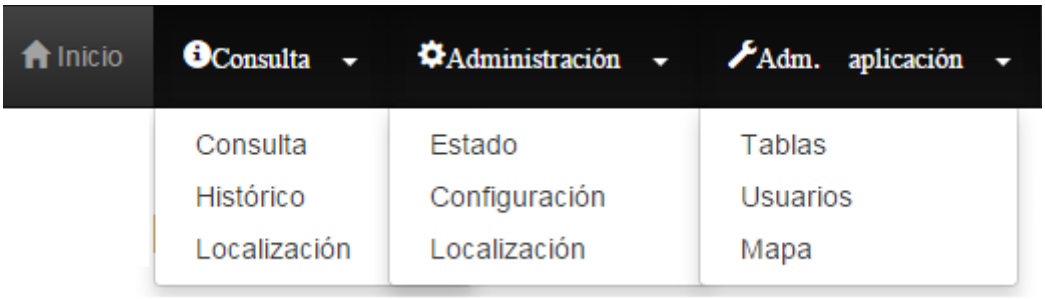
6.1.1 Mapa web

El menú que va a contemplar el cliente Web son:

- Página inicial: descripción del proyecto
- Consulta: consulta de los datos por un usuario sin autenticación previa
  - Consulta: se muestra una lista de las motas. El usuario selecciona una mota y el sistema muestra los valores de los sensores de la mota.
  - Histórico: se muestra un desplegable con las motas y con los sensores disponibles. El usuario selecciona una mota y un sensor y pulsa consultar y el sistema muestra los últimos valores del sensor de la mota, incluida una presentación gráfica.
  - Localización: se muestra un mapa con la localización de las motas. Las motas sin posición se muestran en el lateral del mapa. El usuario selecciona una mota y el sistema muestra los valores de los sensores de la mota.
- Administración: consulta/modificación de datos por un usuario logado
  - Estado: se muestra una lista de las motas. El usuario selecciona una mota y pulsa Eliminar para borrar la mota. El usuario selecciona una mota y el sistema muestra los valores que reflejan el estado de la mota.
  - Configuración: se muestra un desplegable con las motas. El usuario selecciona una mota y el sistema muestra los valores de los parámetros de configuración. El usuario puede modificar estos valores y pulsar modificar para que queden reflejados los cambios.

- Localización: se muestra un mapa con la localización de las motas. Las motas sin posición se muestran en el lateral del mapa. El usuario puede desplazar las motas por el mapa y pulsar guardar para modificar su posición.

- Adm. Aplicación: consulta/modificación de tablas maestras por un usuario logado
  - Tablas: se muestra un desplegable con las tablas administrables. El usuario selecciona una tabla y se muestran los valores de la tabla, con un icono para eliminar el valor. Debajo se muestra el formulario, si se selecciona un valor se carga en el formulario. Si se pulsa aceptar se salvan los valores.
  - Usuarios: se muestran los usuarios, con un icono para eliminar el valor que sólo aparece si hay más de un usuario en la lista. Debajo se muestra el formulario, si se selecciona un usuario se carga en el formulario. Si se pulsa aceptar se crea el usuario o se modifica la contraseña.
  - Mapa: se muestra un formulario con los campos de ancho y alto y un botón para seleccionar una imagen. Si se pulsa aceptar se actualiza el mapa.



6.1.2 Menú web

6.2 Diseño de la interfaz WEB

Decisiones de diseño:

Se opta por utilizar el módulo CoAP de Python que viene con OpenWSN, mortificándolo para que al recurso que atiende la petición cuando actúa como servidor (coapResource) le llegue la información del emisor (sender).

Modulos Python

UOCAPP (uocapp.py): Módulo principal: inicia la aplicación(cliente CoAP, el servidor CoAP y la aplicación Flask). También se encarga de cerrar las conexiones al finalizar la aplicación.

INIT (uocapp_init.py): Inicia la BBDD (crea la estructura e inicia los datos básicos). Debe ejecutarse antes de iniciar la aplicación la primera vez.

FLASK (uocapp_flask.py): Aplicación web con Flask

- *app*: instancia de Flask que representa la aplicación web y mantiene los datos de configuración
- *sensores*: GET que devuelve la lista de sensores
- *parametros*: GET que devuelve la lista de parámetros
- *motas*: GET que devuelve la lista de motas
- *ultimos_valores_motas*: GET que devuelve para una mota los últimos valores de sus sensores
- *valores_mota_sensor*: GET que devuelve para una mota y un sensor el histórico de sus valores
- *adm_motas*: GET que devuelve la lista de motas (información extendida para el administrador). Requiere login
- *adm_borra_mota*: DELETE que elimina una mota. Requiere login
- *adm_mota_configuracion*: GET que dada una mota recupera los valores de sus parámetros de configuración. Requiere login
- *adm_mota_status*: GET que dada una mota recupera los valores de sus parámetros de estado. Requiere login
- *adm_mota_configuracion_modificacion*: POST que modifica los parámetros de configuración de la mota tanto en la BBDD como en la mota utilizando el cliente CoAP. Requiere login
- *adm_motas_configuracion_modificacion*: POST que modifica los parámetros de configuración de todas las motas recuperadas de la BBDD, tanto en la BBDD como en la mota utilizando el cliente CoAP (sólo los parámetros que afectan a la mota, el resto son ignorados). Requiere login
- *login*: POST que comprueba si el usuario/password es correcta y si es así guarda el usuario en la sesión web
- *logout*: DELETE que elimina el usuario en la sesión web. Requiere login
- *get_login*: petición DELETE que recupera el usuario en la sesión web
- *get_administracion*: GET que devuelve los datos de una tabla maestra. Requiere login
- *update_administracion*: POST que modifica los datos de una tabla maestra. Requiere login
- *delete_administracion*: DELETE que elimina un registro de una tabla maestra. Requiere login
- *get_mapa*: GET que devuelve la imagen del mapa
- *get_mapa_info*: GET que devuelve las dimensiones del mapa

- *update_mapa*: POST que actualiza el mapa y las dimensiones. Requiere login

DAO (uocapp_dao.py): acceso a datos. Los métodos que presentan una lógica más compleja son:

- *motasExtendido*: recupera los datos de la tabla MOTAS. Obtiene de VALORES_PARAMETROS aquellos valores que tienen indicado en el campo interfaz que deben devolverse a la parte web
- *borra_mota*: elimina los datos de la mota. Para ello:
  1. elimina los datos de las tablas de valores por id_mota
  2. elimina los datos de la tabla MOTAS por id
- *inserta_datos_sensor*: inserta los datos de sensores recibidos de la mota. Para ello
  1. Comprueba si la mota existe por la ip de la misma. Si no existe la inserta en la tabla MOTAS y de inserta en la tabla VALORES_PARAMETROS los valores por defecto de cada uno de ellos.
  2. Si la mota ya existía, actualiza la tabla VALORES_SENSORES para eliminar la marca de último para los valores de la mota que la tuviesen
  3. Inserta en la tabla VALORES_SENSORES con la marca de último los nuevos valores
  4. Inserta/actualiza en la tabla VALORES_STATUS los nuevos valores
  5. Se actualiza la tabla MOTAS con el indicador de sincronización recibido de la mota sincMota y la fecha actual como fecha de última recepción de la información
- *inserta_valores_parametros*: inserta los valores de los parámetros de la mota. Para ello
  1. Inserta en la tabla VALORES_PARAMETROS por el campo id_mota
  2. Actualiza la tabla MOTAS el campo campo de sincronización con la mota fechaSinc

COAP_CLIENTE (uocapp_coap_cli.py): Cliente CoAP para acceso a las motas

COAP_SERVIDOR (uocapp_coap_srv.py): Servidor CoAP para atender la recepción de información desde las motas

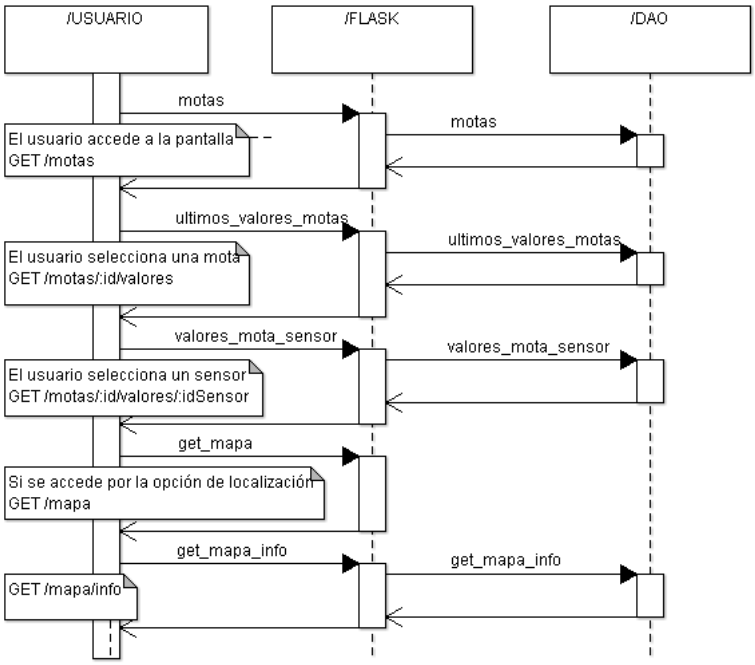
- POST: recibe una petición POST.
  1. Recupera los valores de estado para conocer la conversión a aplicar al valor recibido, el valor crítico de cada elemento de estado y su condición de comparación; aplica la

conversión al valor de estado recibido de la mota y comprueba comparando el valor generado con el crítico utilizando la condición de comparación si ese parámetro genera una alerta.

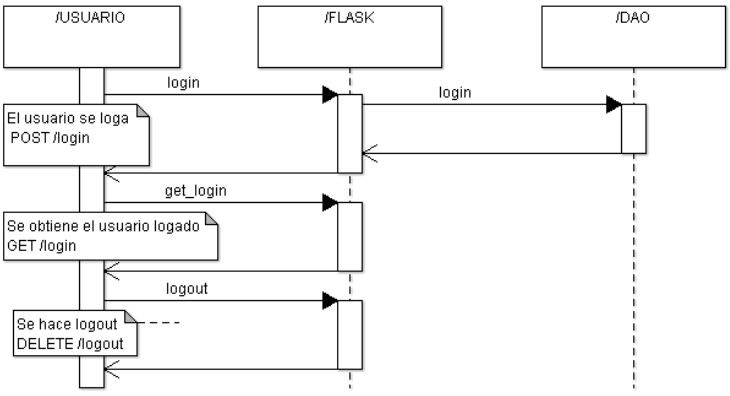
2. Recupera los valores de los sensores para conocer la conversión a aplicar al valor recibido; aplica la conversión al valor del sensor recibido de la mota.
3. Los datos recibidos (con las conversiones aplicadas) más la alerta calculada se insertan en BBDD.

LOG (uocapp_logging.py): configuración del log

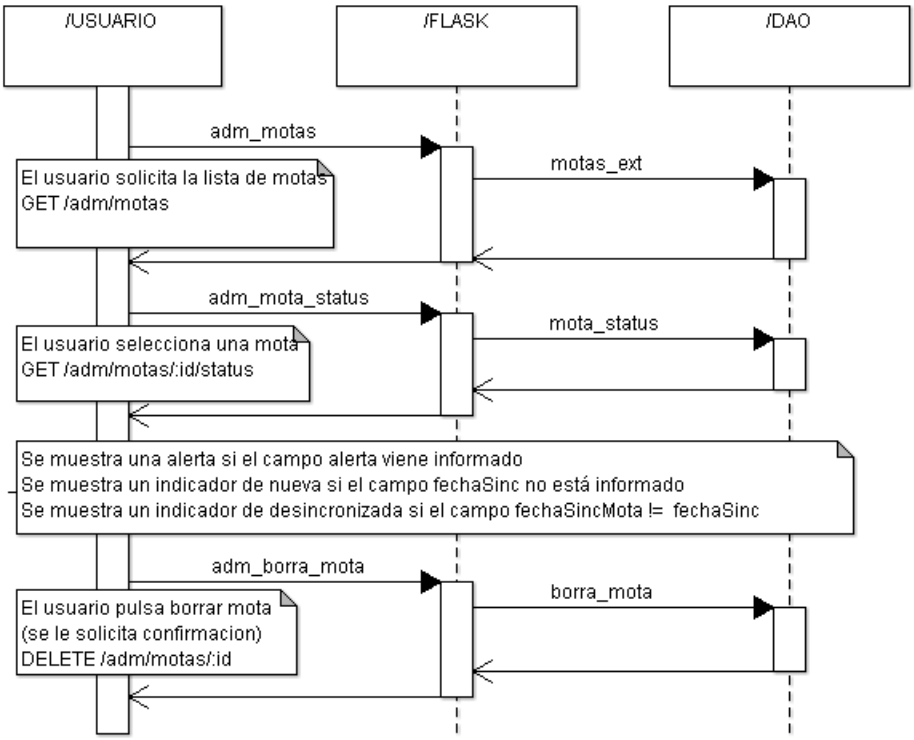
Diseño de los casos de uso de consulta:



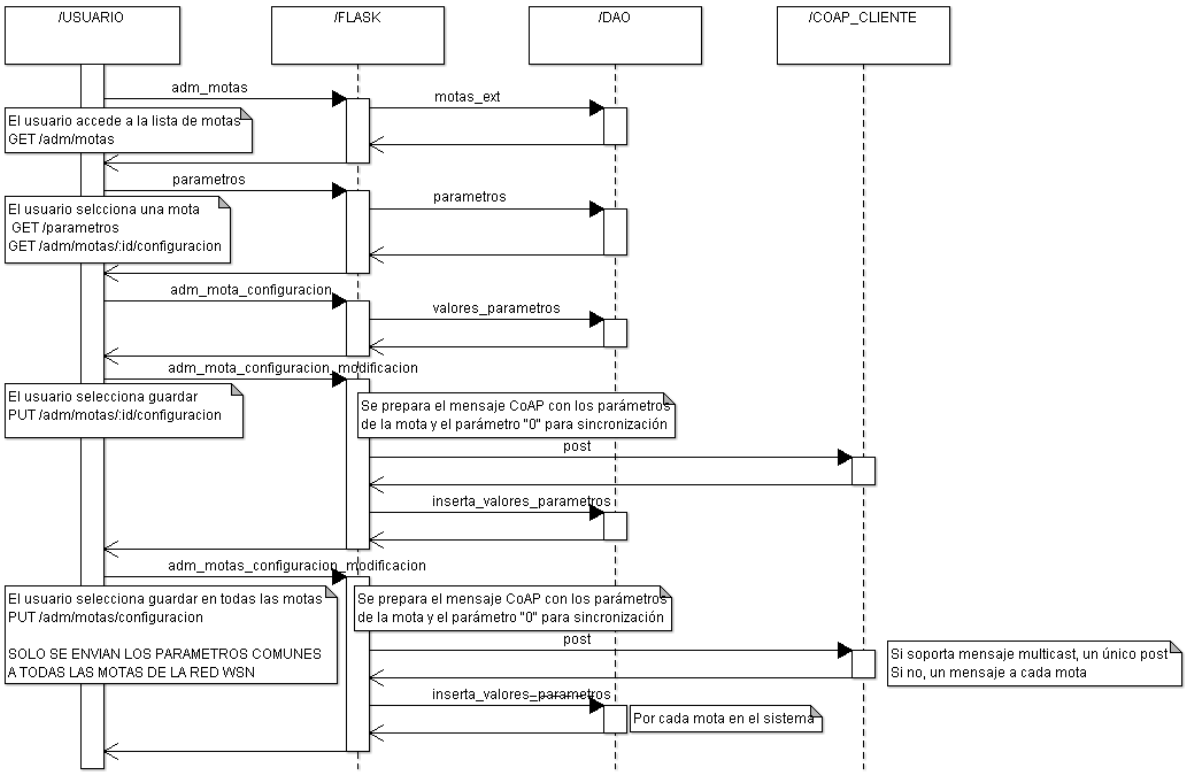
6.2.1 Diagrama de secuencia casos de uso U01, U02, U03



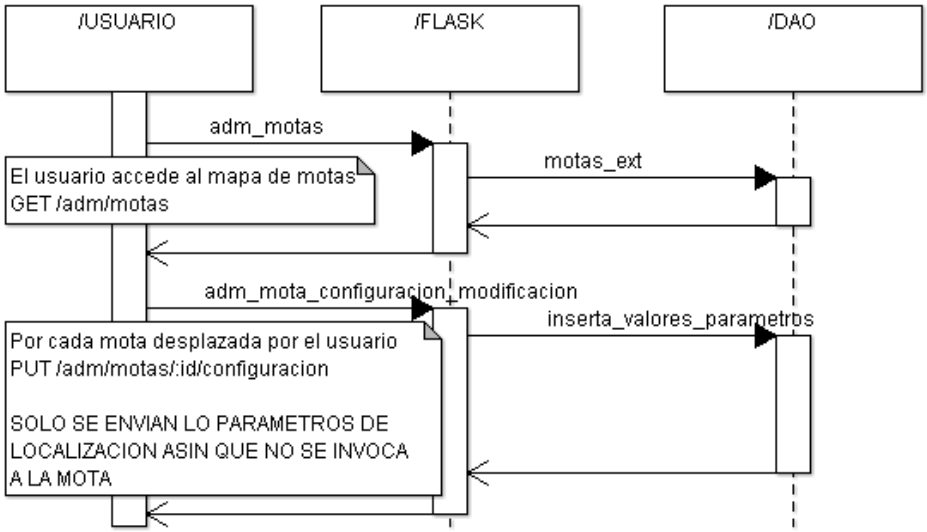
6.2.2 Diagrama de secuencia caso de uso A01



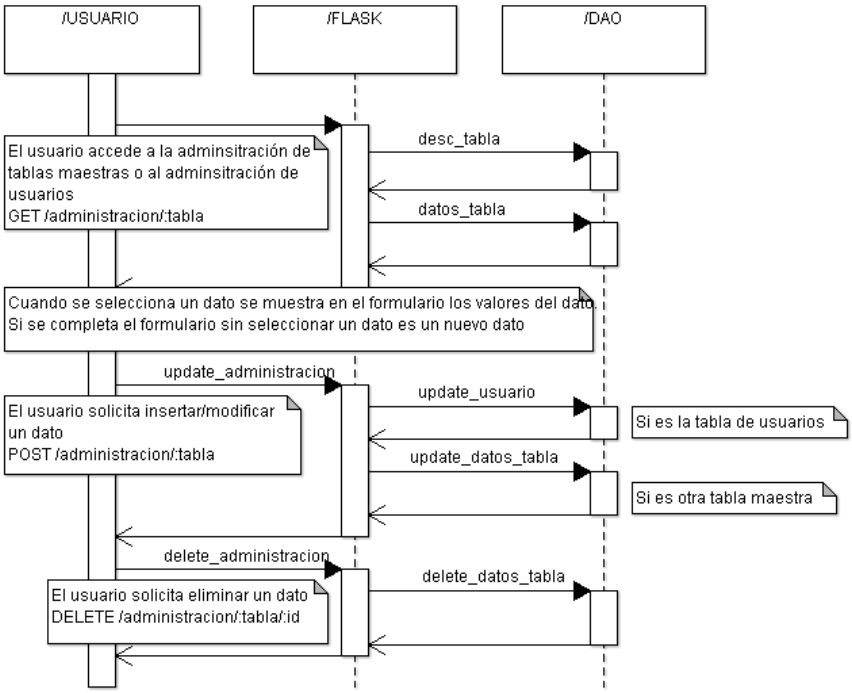
6.2.3 Diagrama de secuencia casos de uso A02, A03



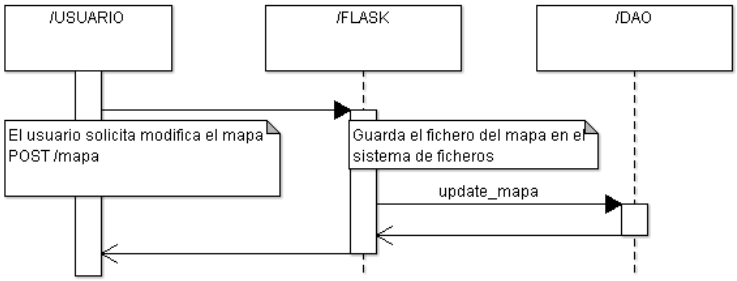
6.2.4 Diagrama de secuencia de los casos de uso A04, A05, A06



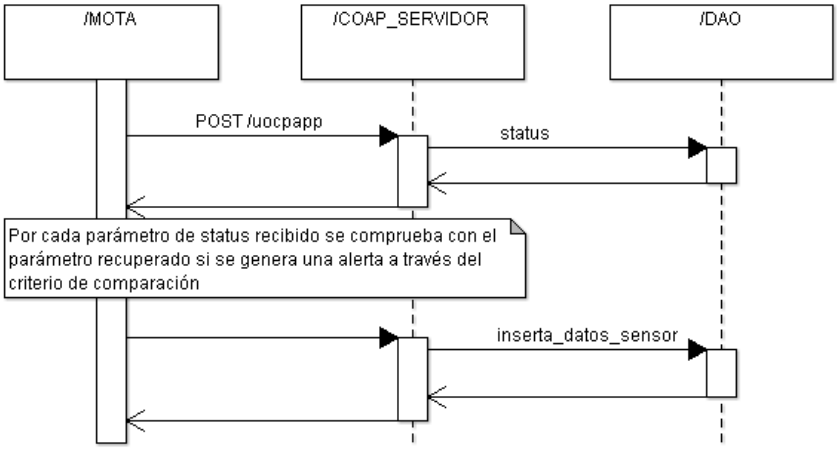
6.2.5 Diagrama de secuencia caso de uso A07



6.2.6 Diagrama de secuencia casos de uso AP01, AP02, AP03, AP04



6.2.7 Diagrama de secuencia caso de uso AP05



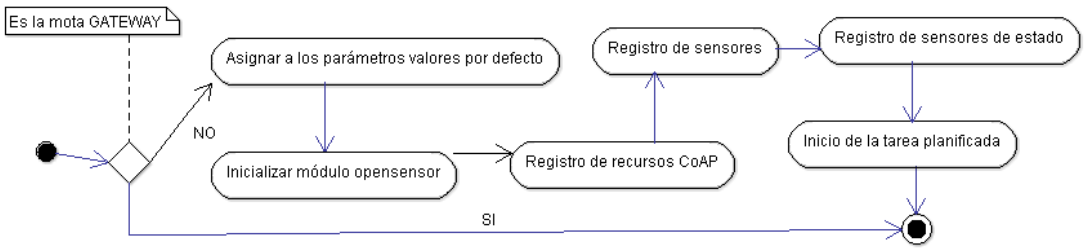
6.2.8 Diagrama de secuencia caso de uso M01

6.3 Diseño de la aplicación WSN

En este apartado se describe el diseño de la aplicación que se ejecuta en las motas.

uocapp_init

En este método se inicializa la aplicación, registrando los recursos necesarios. Si no es la mota que actúa como Gateway, debe registrar el recurso CoAP “uocapp” para recibir los mensajes CoAP POST, registrar los sensores que se van a leer, tanto los de valores como los de estado, e iniciar el temporizador que va a establecer cuando la mota envía información al servidor.



6.3.1 Inicialización de la aplicación

uocapp_tarea_planificada

Es la función que se invoca desde el temporizador. Esta función recorre la lista de sensores (sólo los que están marcados como activos) para obtener su valor actual invocando a su función callbackRead, recorre la lista de sensores de estado para obtener su valor actual invocando a su función callbackRead. Con estos valores construye el mensaje JSON y se invoca a la función *uocapp_coap_send* que envía el mensaje como POST NON a todos los

servidores que encuentra en el parámetro de configuración correspondiente marcando el contenido como application/json.

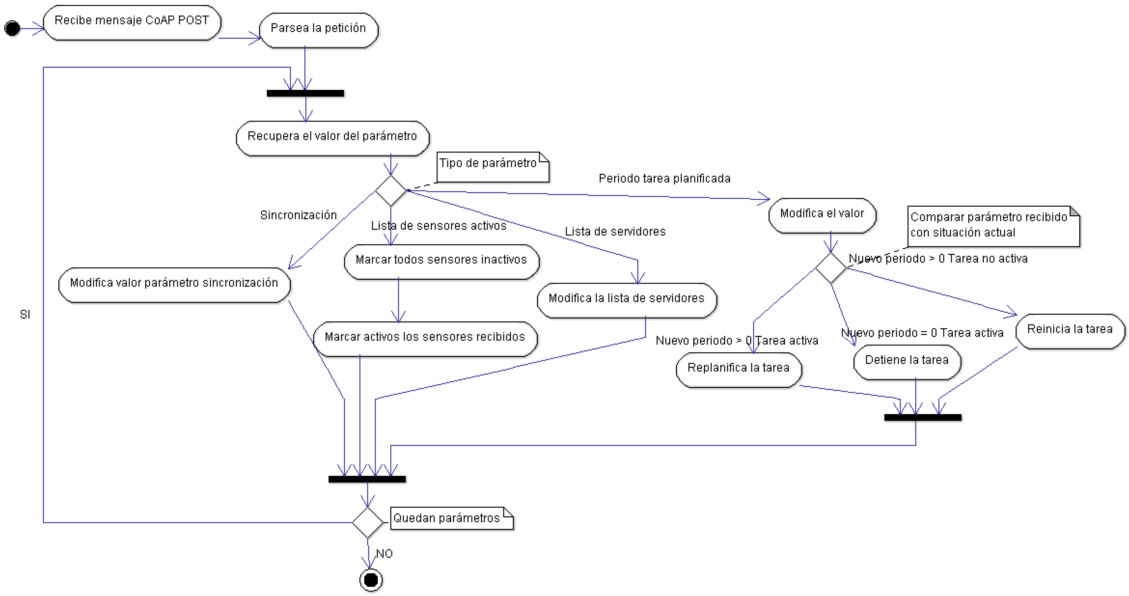
La lectura de los sensores se realiza mediante el método *callbackRead* que devuelve el valor entero obtenido directamente del sensor. Existe la opción de invocar a *callbackConvert* que devuelve el valor de la medida convertido a sus unidades habituales (ºC, etc., decimal en lugar de entero). La dificultad de tratar con decimales para generar el mensaje JSON y el hecho de que no todos los sensores disponen de esta función hace recomendable tratar la conversión a nivel del servidor.

La respuesta no se envía directamente sino que se asigna al planificador para que se remita de acuerdo a la prioridad asignada (prioridad de tareas CoAP).

6.3.2 Tarea planificada

uocapp_coap_receive

Es la función que se invoca cuando se recibe una petición CoAP. En este caso recibe el mensaje POST e invoca a *uocapp_modificar_parametros* para modificar los parámetros recibidos. La respuesta no incluye payload y devuelve el código de respuesta 2.04 Changed para indicar que el proceso ha sido correcto.



6.3.3 Modificación parámetros mota

Para modificar los parámetros se parsea la petición y cada parámetro es tratado de manera diferente:

- Si el parámetro es el de sincronización, simplemente se cambia el valor

- Si es la lista de sensores activos si el parámetro recibido es T se marcan todos como activos, si es N se marcan todos como inactivos y si es una lista se marcan todos los sensores como inactivos y posteriormente se activan los recibidos. Tendrá efecto cuando se vuelva a ejecutar la tarea planificada
- Si es la lista de servidores se modifica esta información, que tendrá efecto cuando se vuelva a ejecutar la tarea planificada.
- Si es el periodo de la tarea planificada se cambia el valor y se modifica la planificación. Si la tarea esta activa y el nuevo valor del periodo es distinto de 0, se replanifica; si la tarea esta activa y el nuevo parámetro es 0 se detiene; y si la tarea está inactiva y el nuevo valor de la planificación es distinto de 0 se reinicia la tarea con el nuevo periodo recibido.

Notas de implementación:

- Se utiliza como referencia la aplicación *csensors* incluida como código de ejemplo en OpenWSN; en esta aplicación se realizan algunas de las funciones que debe hacer la nueva aplicación.
- El mecanismo que proporciona OpenWSN para saber si la mota es la mota Gateway es i*dmanager_getIsDAGroot*; debe comprobarse antes de registrar recursos CoAP, tareas planificadas, etc.
- Se crea una nueva aplicación *uocapp* dentro del directorio /openapps
- Es necesario invocar el método *uocapp_init* dentro de *openapps_init* en *openapps.c* y añadir un identificado único de la aplicación COMPONENT_UOCAPP en *opendefs.h*
- Es necesario modificar los ficheros de construcción del proyectos de scons *SConscript* para que incluyan la nueva aplicación en los directorios /openapps y /openstack
- El recurso CoAP "uocapp" registrado solo recibe mensajes POST. Se implementa el método GET para devolver en un mensaje el valor actual de los parámetros de configuración de cara a facilitar las pruebas unitarias.
- La compilación por defecto genera error de linkado al tratar de utilizar funciones de manejo de cadenas que trabajan con memoria dinámica (*sprintf*, etc.). Para evitarlas se crean funciones que las reemplacen para conversión a enteros y formar el mensaje JSON.

7 Pruebas

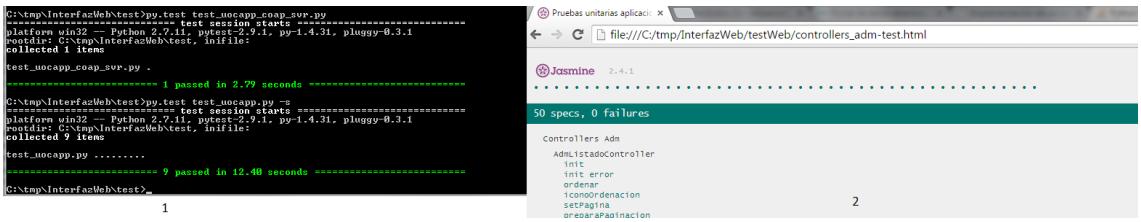
En este apartado se describen las pruebas que se van a realizar sobre los diferentes componentes de la aplicación y la estrategia para su ejecución.

7.1 Pruebas unitarias

Las pruebas unitarias tienen como objetivo probar las diferentes partes de la aplicación sin interactuar entre sí. Además su automatización facilita las pruebas de regresión (pruebas de que todo sigue funcionando ante cambios de la aplicación).

Para la parte cliente se dispone de un servidor JSON que permite simular la parte servidora. La automatización se realiza con Jasmine para probar el código desarrollado, sin realizar pruebas e2e (end to end) que prueban también la parte de la vista HTML; para ejecutarlas basta con visualizar las diferentes páginas HTML del directorio de test.

Para la parte servidora se prueba mediante mensajes que envía un cliente REST de los que existen disponibles en los diferentes navegadores; para simular la mota se crear un servidor y un cliente con el módulo CoAP de python. La automatización se realiza con py.test que facilita este proceso; para ejecutarlas se ejecuta el comando *py.test test_...py* en los componentes ubicados en el directorio de test.



7.1.1 Pruebas unitarias automatizadas
[1] InterfazWeb con py.test [2] Cliente web con Jasmine

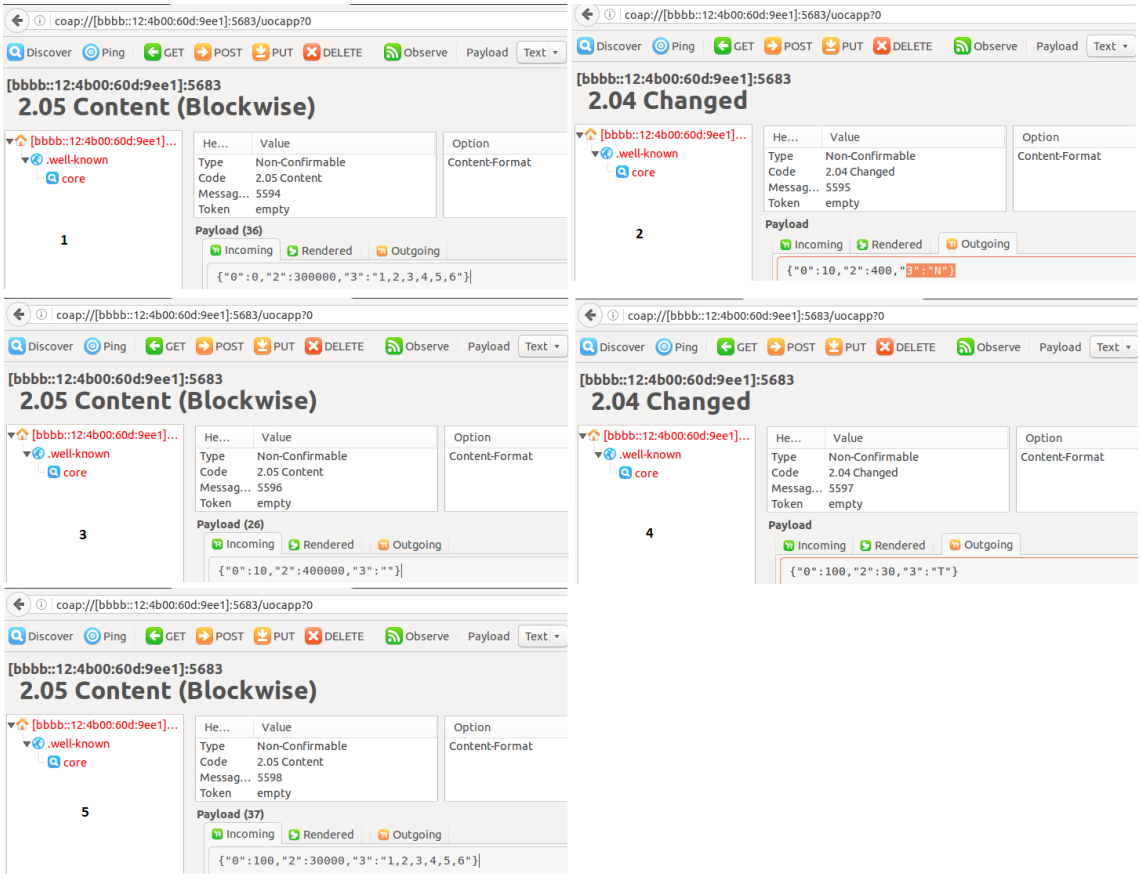
Para la aplicación instala en la mota no existe un mecanismo de automatización de las pruebas.

Se hace uso del plugin COPPER del navegador Mozilla-Firefox para enviar mensajes CoAP. Se puede enviar el mensaje POST /uocapp para modificar los parámetros y comprobar que se ha modificado la configuración enviando el mensaje GET /uocapp?x que devuelve los valores de los parámetros de configuración. Las opciones de x son:

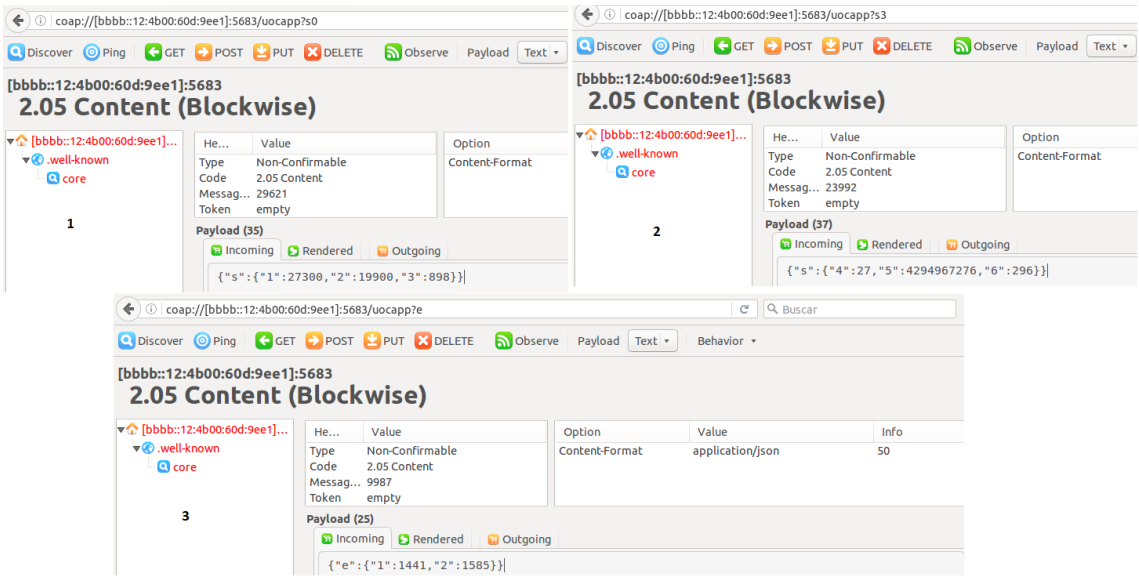
- 0: devuelve los parámetros tiempo entre muestras y sensores activos
- 1[y]: devuelve el servidor de índice y en la lista

- e: devuelve el valor de los sensores asociados a parámetros de estado (mensaje que la mota enviará a través de la tarea planificada).
- s[y]: devuelve los valores de los sensores (mensaje que la mota enviará a través de la tarea planificada) empezando por el índice y y no más de 3 valores.
- i: devuelve la lista de sensores en la mota (T temperatura, H humedad, X,Y,Z aceleración, L luz, t temperatura CPU, v batería).

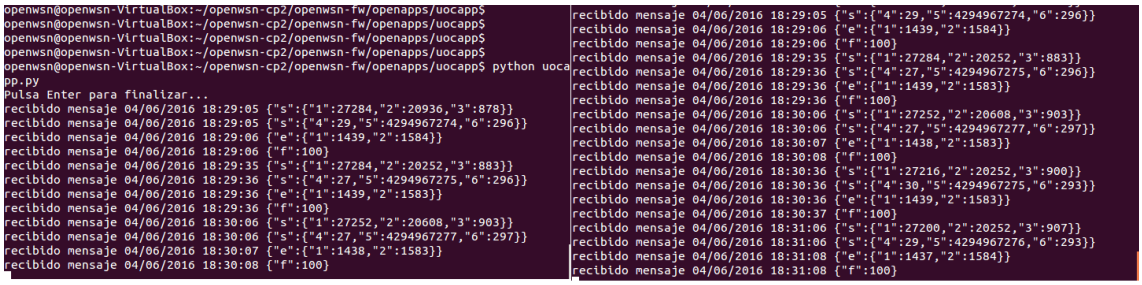
Para probar el funcionamiento de la tarea planificada se incluye la aplicación Python uocapp.py que actúa como servidor CoAP recibiendo los mensajes de la tarea planificada y mostrando por consola la información recibida.



7.1.2 Pruebas unitarias con COPPER: valores de los parámetros en la mota
[1] Consulta de los valores [2] Modificación con POST [3] Consulta tras modificación
[4] Nueva modificación con POST [5] Consulta tras segunda modificación

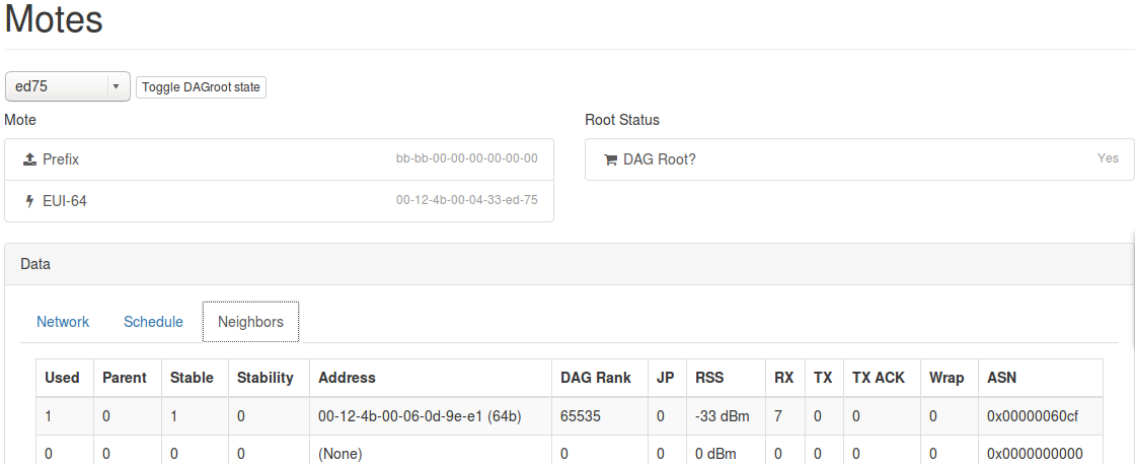


7.1.3 Pruebas unitarias con COPPER: valores de los sensores
[1] Sensores de temperatura, humedad, luz [2] Sensores de aceleración
[3] Sensores de estado



7.1.4 Pruebas unitarias tarea planificada
Recepción de mensajes en la aplicación Python creada para las pruebas unitarias

7.2 Pruebas de integración



7.2.1 Estado de la red WSN con OpenVisualizer

Las pruebas de integración tienen como objetivo probar los diferentes componentes de la aplicación conjuntamente. Se puede proceder a una integración escalonada, si se dispone de módulos que simulen las diferentes

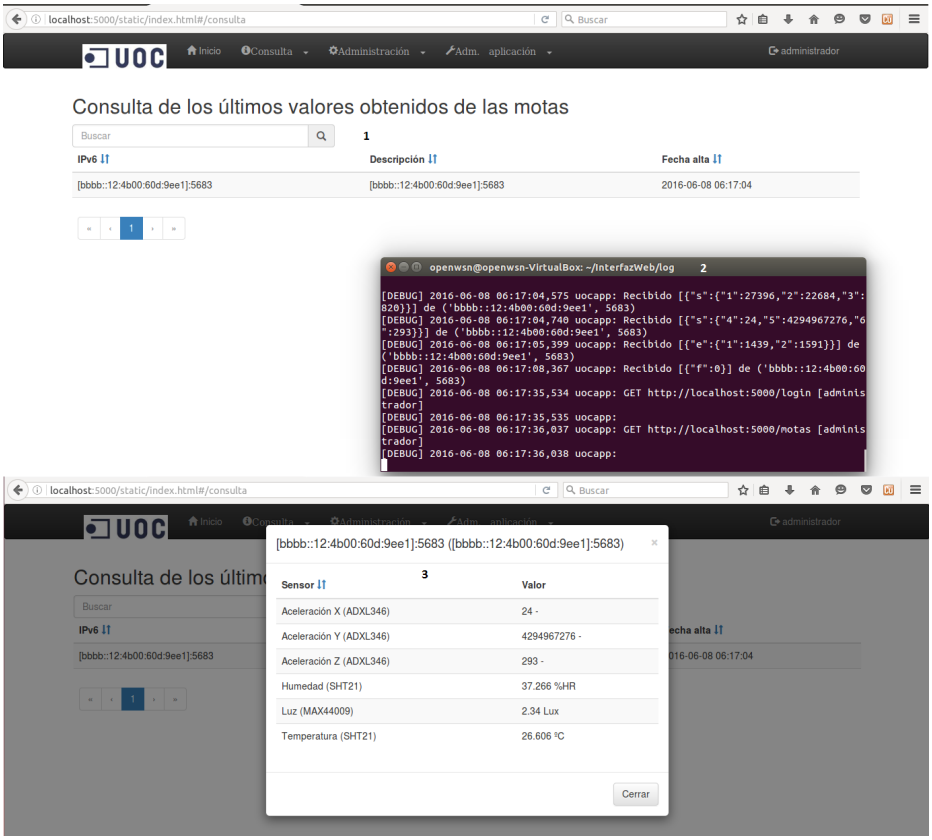
partes del sistema, o en un único paso. La primera facilita la detección de los errores y su localización, por lo que se va a seguir esta estrategia.

Las pruebas deben cubrir los diferentes casos de uso definidos.

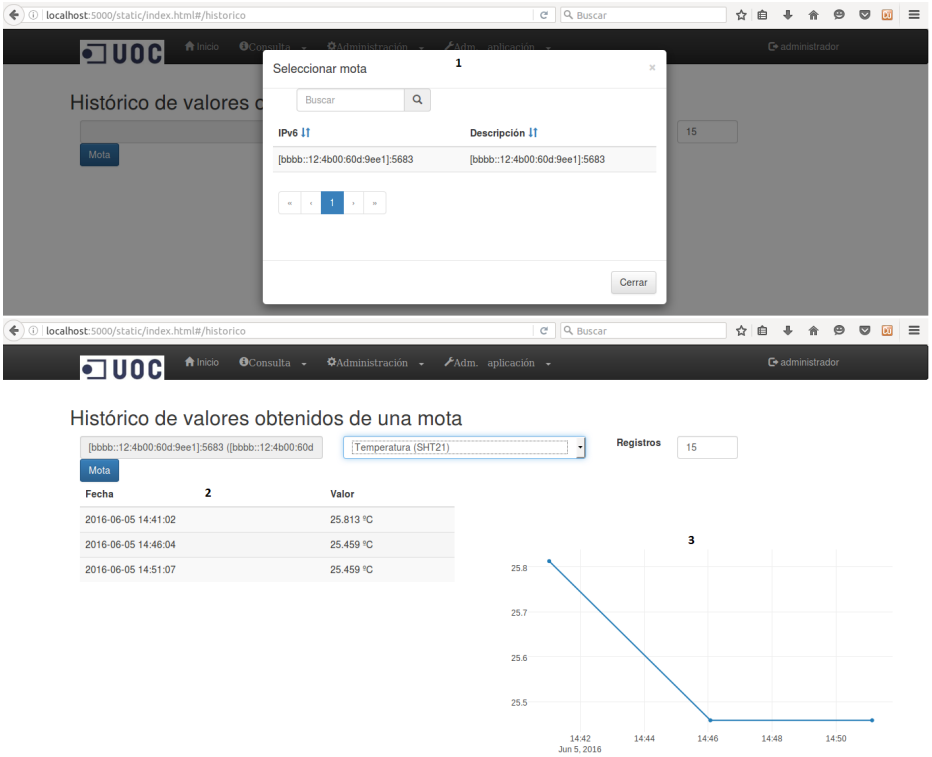
7.2.1 Consultas

Caso de Uso	Prueba	Descripción
U01	CP-U01	Acceder a la aplicación, visualizar la lista de motas
	CP-U02	Seleccionar una mota, visualizar los últimos valores de los sensores
U02	CP-U03	Acceder a la aplicación, visualizar la lista de motas
	CP-U04	Seleccionar una mota, cargar la lista de sensores. Seleccionar un sensor, ver los valores de ese sensor a lo largo del tiempo
U03	CP-U05	Acceder a la aplicación, visualizar el mapa con las posiciones de las motas.
	CP-U06	Seleccionar una mota, visualizar los últimos valores de los sensores

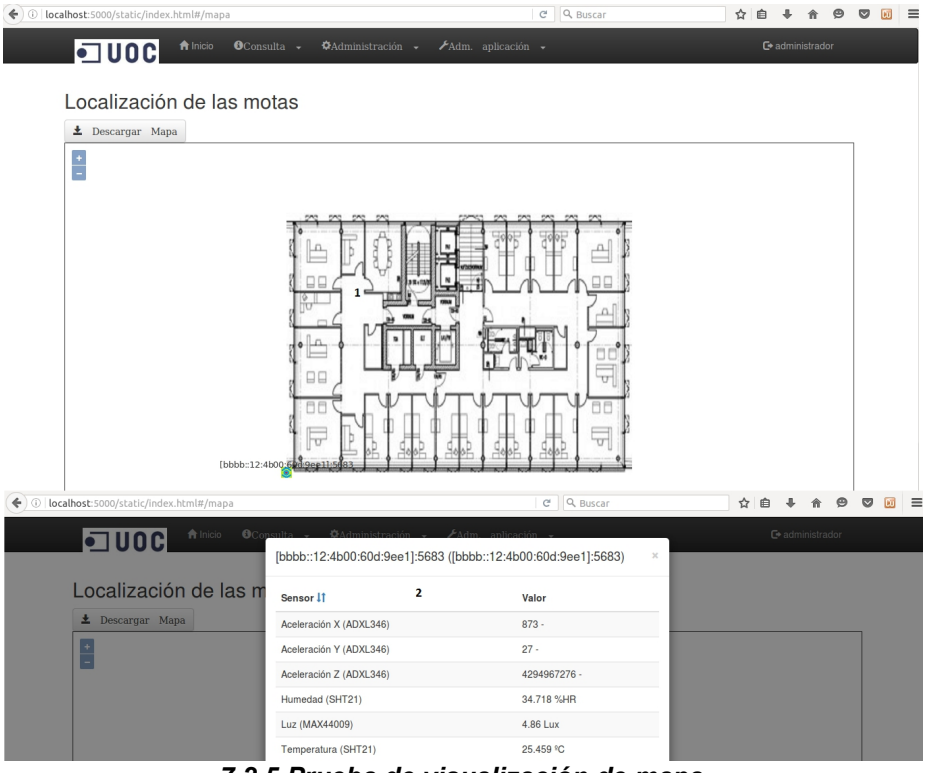
7.2.2 Tabla de casos de prueba consultas



7.2.3 Prueba consulta de los valores de los sensores
Casos de prueba CP-U01 CP-U02 CP-M01
[1] Listado de motas [2] Log de la aplicación con el mensaje recibido de la mota
[3] Detalle de los valores



7.2.4 Prueba de histórico de valores
Casos de prueba CP-U03 CP-U04
[1] Selección de la mota [2] Tabla de valores (cada 5 minutos, tiempo por defecto)
[3] Gráfico asociado

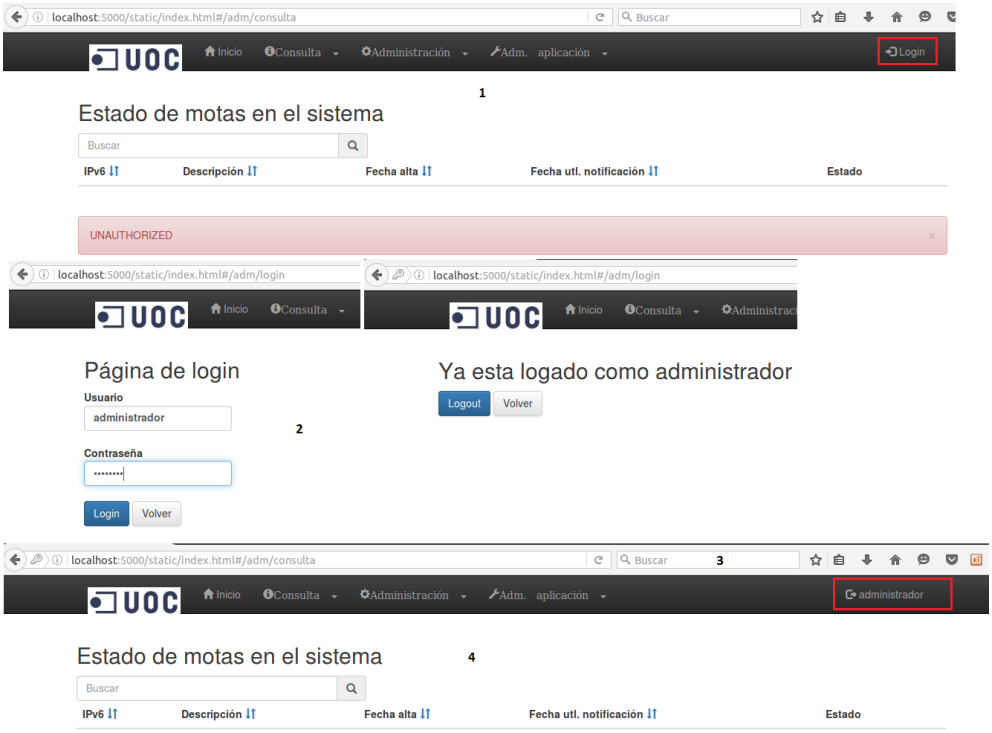


7.2.5 Prueba de visualización de mapa
Casos de prueba CP-U05 CP-U06
[1] Mapa con la motas [2] Pulsación sobre la mota del mapa

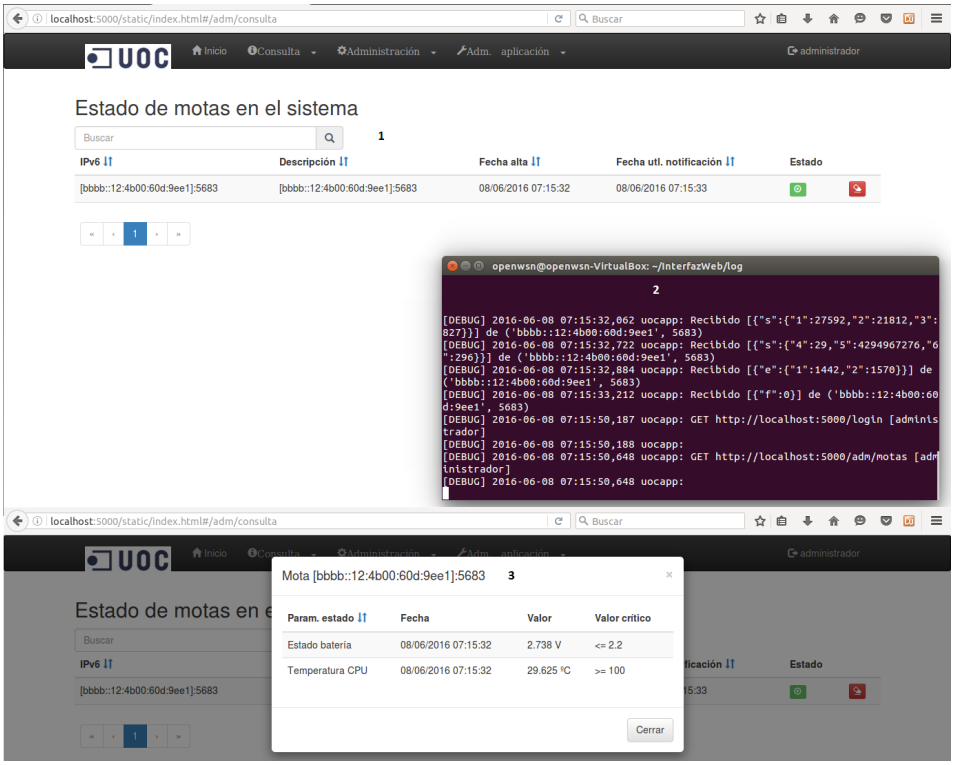
7.2.2 Administración de la red WSN

Caso de Uso	Prueba	Descripción
A01	CP-A01	Acceder a las funciones de administración sin logarse, se obtiene un error
	CP-A02	Logarse en la aplicación con un usuario/contraseña invalido, se obtiene un error
	CP-A03	Logarse en la aplicación, el usuario aparece logado y puede acceder a otras funciones de administración
	CP-A04	Deslogarse de la aplicación, el usuario ya no aparece logado y no se puede acceder a otras funciones de administración
A02 A03	CP-A05	Acceder a la administración, visualizar la lista de motas, se muestran las motas con alerta, nuevas y desincronizadas
	CP-A06	Seleccionar una mota, se muestran los valores de los parámetros de estatus
	CP-A07	Seleccionar una mota, eliminarla, desaparece de la lista
A04 A05 A06	CP-A08	Acceder a la administración de parámetros de configuración, visualizar la lista de motas
	CP-A09	Seleccionar una mota, se carga el formulario con sus datos de configuración
	CP-A10	Seleccionar una mota, modificar los parámetros, pulsar guardar. Tras la confirmación, los datos están modificados tanto en la aplicación como los que aplican (tiempo entre muestras y servidores) en la mota
	CP-A11	Seleccionar una mota, modificar los parámetros, pulsar guardar en todas las motas. Tras la confirmación, los datos tiempo entre muestras y servidores están modificados tanto en la aplicación como en todas las motas
A07	CP-A12	Acceder a la administración de la posición de las motas, se visualiza un plano con la posición de las mismas
	CP-A13	Desplazar algunas motas, pulsar guardar, las motas permanecen desplazadas al volver a acceder al mapa

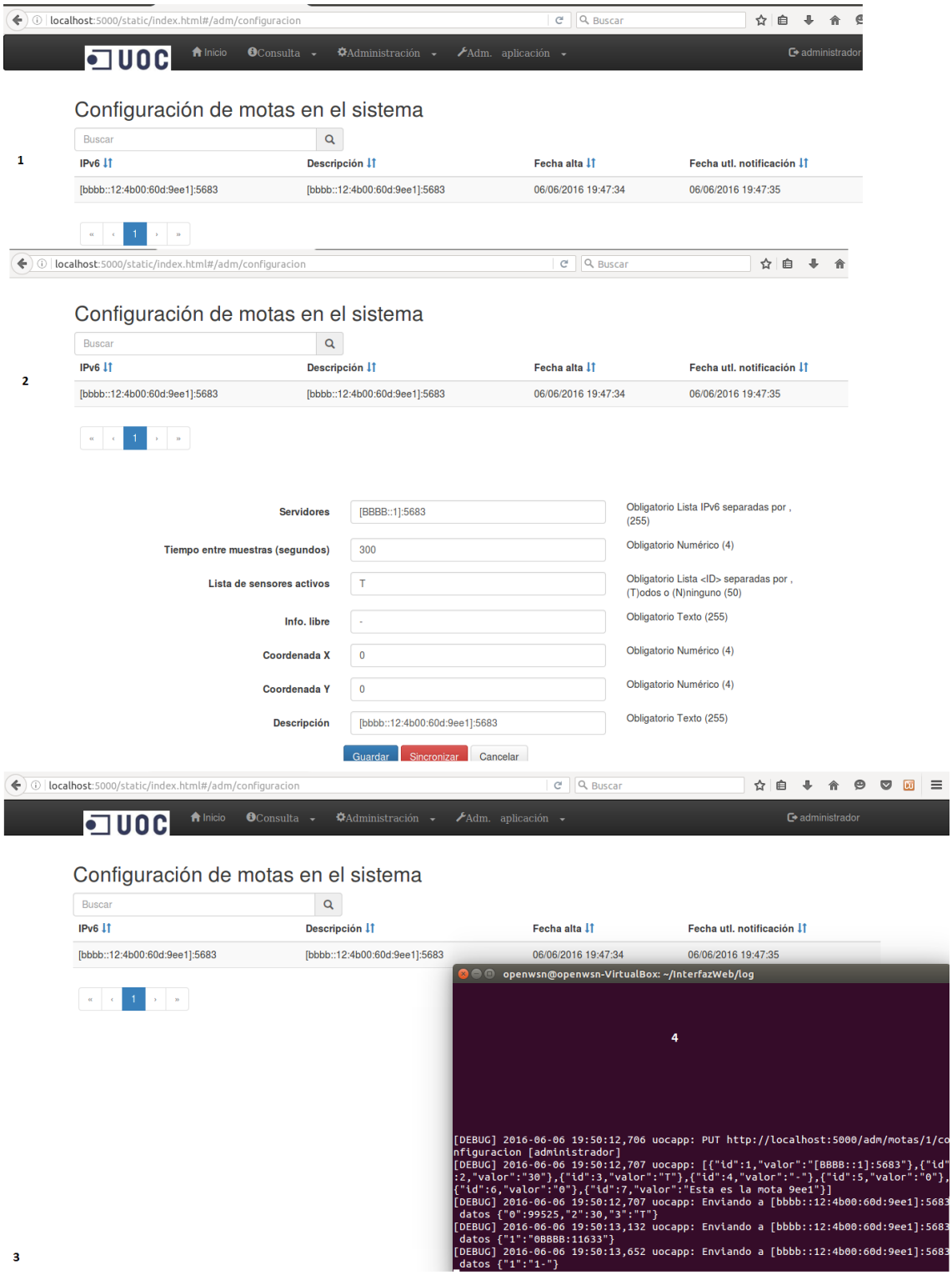
7.2.6 Tabla de casos de prueba administración de la red WSN



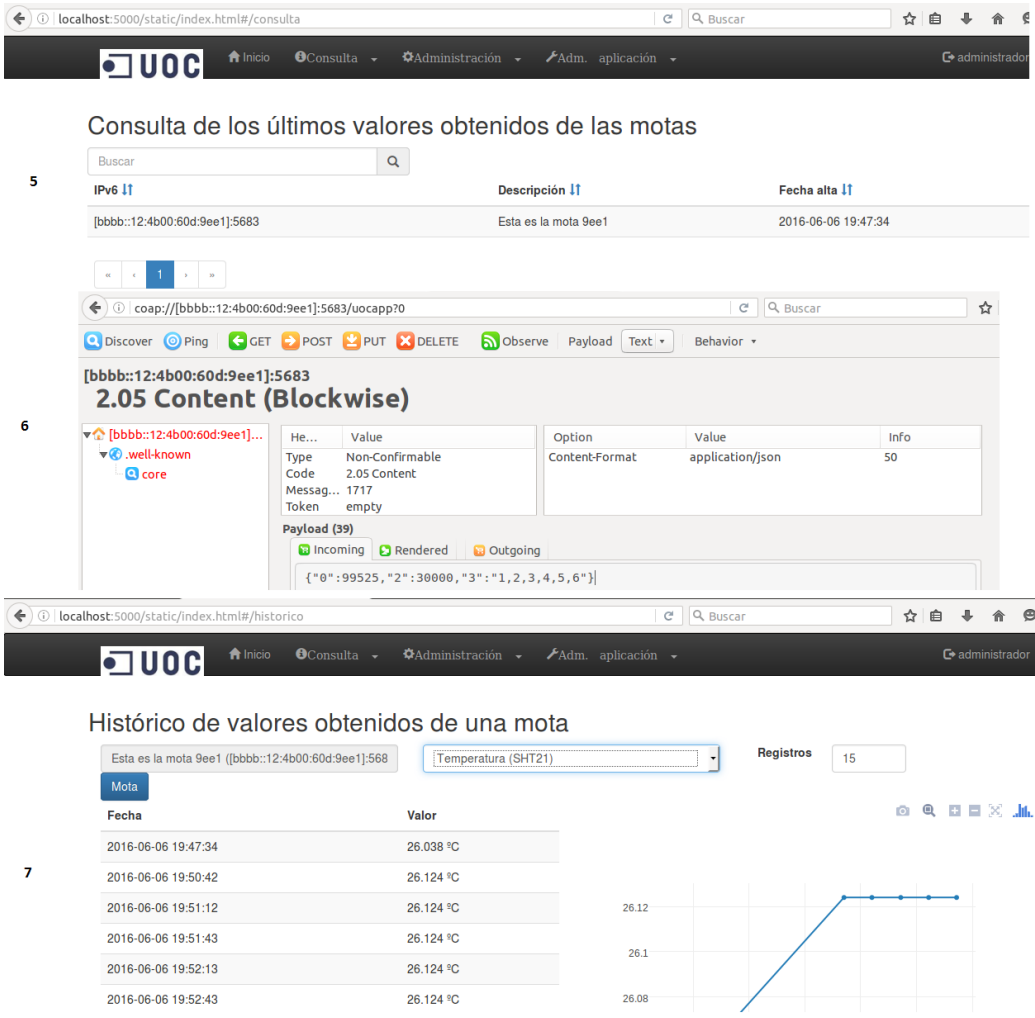
7.2.7 Prueba de login
Casos de prueba CP-A01 CP-A02 CP-A03
[1] Acceso sin logar (error) [2] Ventana de login [3] Mensaje de login correcto [4] Acceso logado



7.2.8 Prueba de consulta del estado de la mota
Casos de prueba CP-A05 CP-A06
[1] Listado de las motas con su estado [2] Log de la aplicación [3] Detalle los valores de estado



7.2.9 Prueba de administración de la configuración de la mota (I)
Casos de prueba CP-A08 CP-A09 CP-A10 CP-A11
[1] Listado de las motas [2] Modificar configuración de la mota (tiempo entre muestras a 30 y descripción) [3] Guardar [4] Log de la aplicación

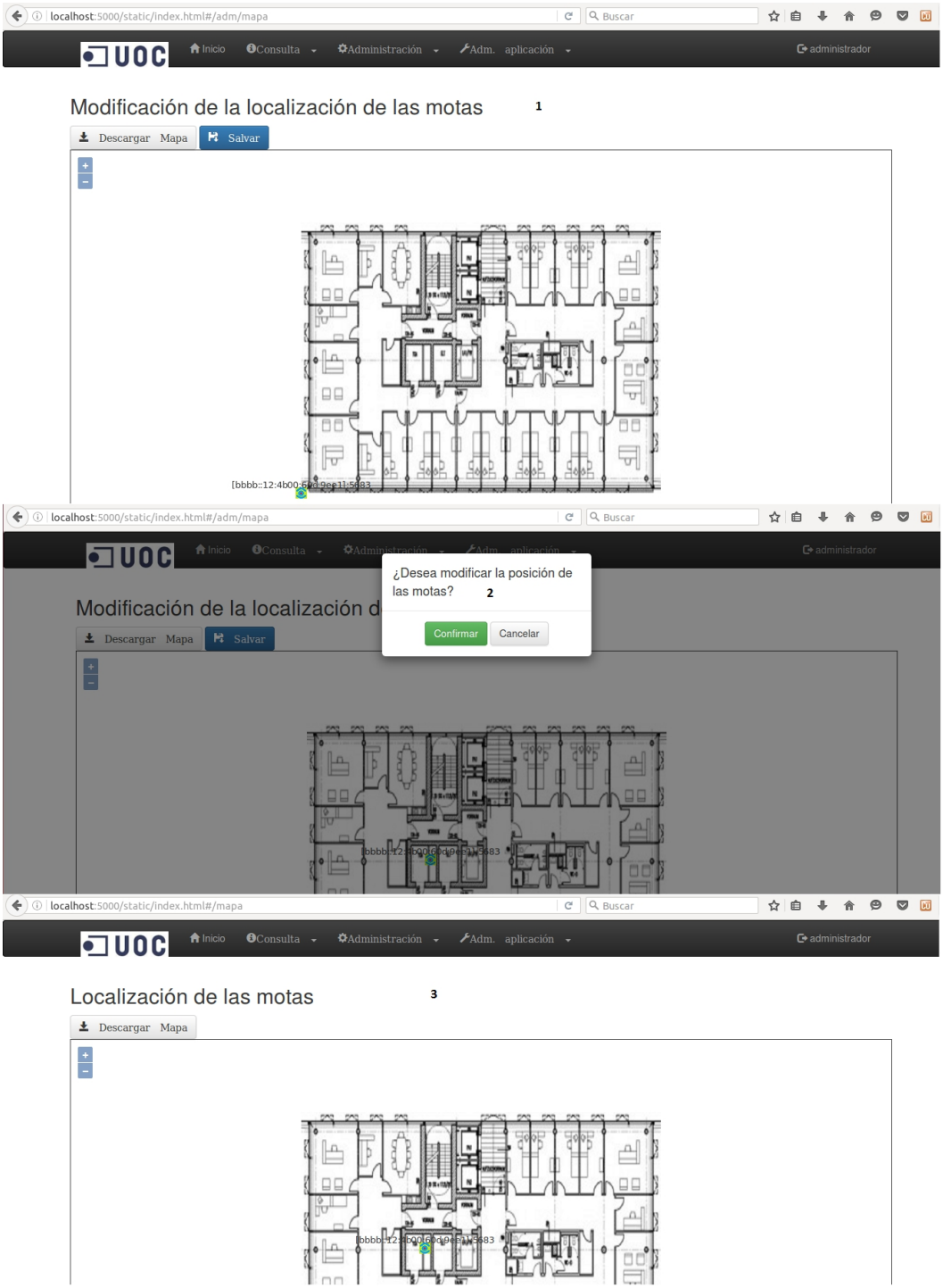


5

6

7

7.2.10 Prueba de administración de la configuración de la mota (II)
[5] Comprobación en consulta (cambia la descripción) [6] Comprobación con COPPER (parámetro 2 es 30000 ms.) [7] Comprobación en el histórico de valores de la mota (valores cada 30 segundos como se ha indicado)

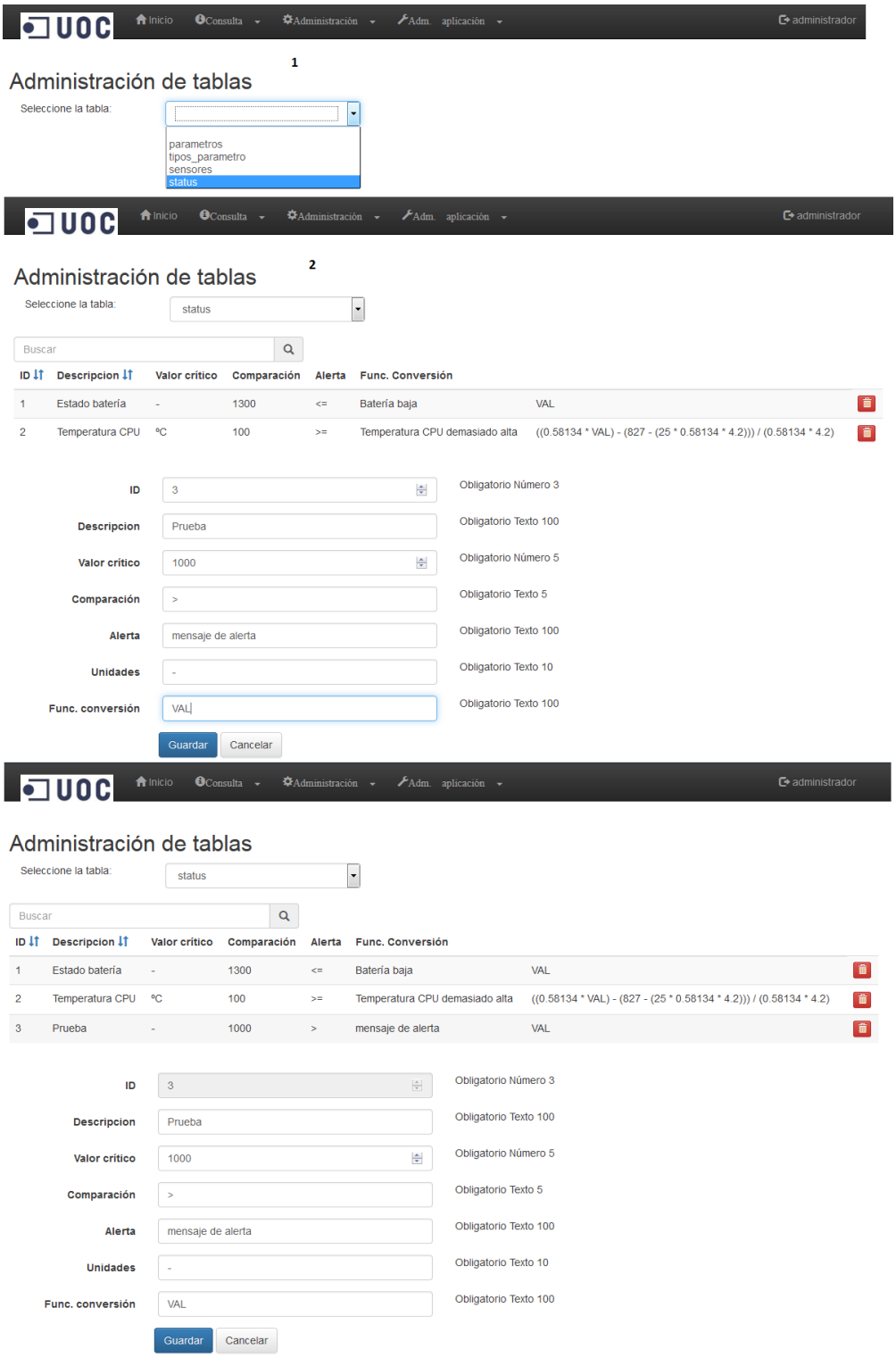


7.2.11Prueba de administración de las mota en el mapa
Casos de prueba CP-A12 CP-A13
[1] Mapa de las motas [2] Modificación de la posición [3] Comprobación en la pantalla de consulta

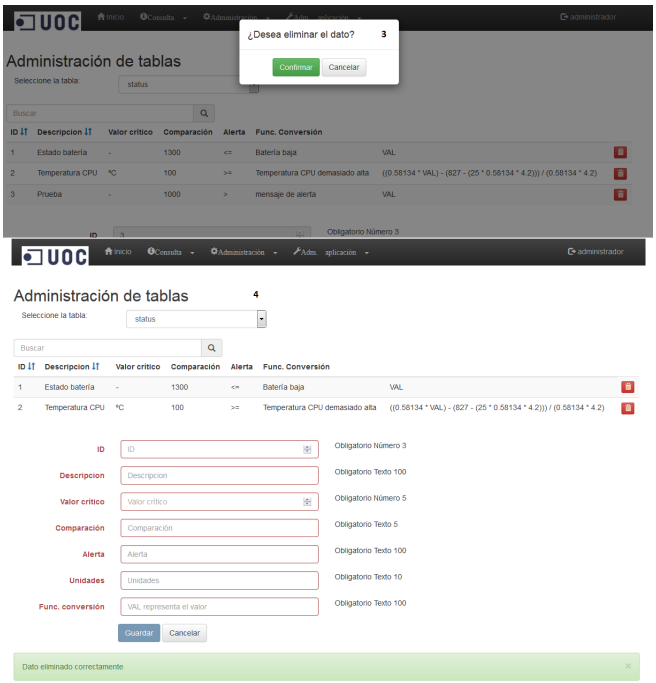
7.2.3 Administración de la aplicación

Caso de Uso	Prueba	Descripción
AP01 AP02 AP03	CP-P01	Acceder a la administración de los elementos de configuración, seleccionar una tabla, se muestran los elementos de esa tabla, con un botón de eliminar al lado de cada uno de ellos.
	CP-P02	Completar todos los valores del formulario, pulsar guardar. Se crea un nuevo elemento de configuración.
	CP-P03	Volver a acceder, el elemento creado aparece en la lista. Seleccionar el elemento, sus valores se cargan en el formulario.
	CP-P04	Modificar algunos de los valores del formulario, pulsar guardar. Se modifica el elemento de configuración.
	CP-P05	Pulsar el botón de eliminar, se elimina el elemento de configuración. Volver a acceder, el elemento eliminado no aparece en la lista.
AP04	CP-P06	Acceder a la administración de los usuarios, se muestran los usuarios, con un botón de eliminar al lado de cada uno de ellos si hay más de uno.
	CP-P07	Completar todos los valores del formulario, pulsar guardar. Se crea un nuevo usuario, con la contraseña en modo HASH en la tabla. Logarse con este nuevo usuario en la aplicación, el login es correcto.
	CP-P08	Volver a acceder, el usuario creado aparece en la lista. Seleccionarlo, sus valores se cargan en el formulario.
	CP-P09	Modificar la contraseña. Se modifica el elemento usuario. Logarse con este usuario y la contraseña modificada en la aplicación, el login es correcto.
	CP-P10	Pulsar el botón de eliminar, se elimina el usuario. Volver a acceder, el usuario eliminado no aparece en la lista.
AP05	CP-P11	Acceder a la administración del mapa, seleccionar una imagen del disco y pulsar guardar. Acceder a la consulta de posición de las motas, el mapa se ha cambiado por el escogido.

7.2.12 Tabla de casos de prueba administración de la aplicación



7.2.13 Prueba de administración de tablas maestras (I)
Casos de prueba CP-P01 CP-P02 CP-P03 CP-P04 CP-P05
[1] Selección de la tabla [2] Crear nuevo elemento [3] Selección del elemento creado



7.2.14 Prueba de administración de tablas maestras (II)
Casos de prueba CP-P01 CP-P02 CP-P03 CP-P04 CP-P05
[4] Eliminación del elemento creado

7.2.4 Información de la mota

Caso de Uso	Prueba	Descripción
U01	CP-M01	La mota envía cada (tiempo entre muestras) a los servidores de (servidores) la información de los valores de los sensores, los valores de estado y la fecha de última sincronización de los parámetros.

7.2.15 Tabla de casos de prueba información de la mota

8. Conclusiones

Las redes inalámbricas de sensores WSN son cada vez más habituales y se utilizan en cada vez mayor número de escenarios. Su complejidad y dispersión hace que dotarlas de una interfaz remota de administración sea cada vez más deseable. El estándar IEEE 802.15.4 cubre la necesidad de estándares inalámbricos de baja tasa y bajo consumo para aplicaciones IoT.

Las soluciones tradicionales en materia de gestión remota de redes WSN se han limitado a un sistema operativo, exigiendo en algunos casos la uniformidad de toda la red WSN en cuanto a hardware y software.

El presente Trabajo Fin de Máster ha pretendido una solución más abierta, basada en estándares definidos específicamente para IoT como CoAP, con el fin de reducir las limitaciones de otras soluciones. Este proceso de eliminación de limitaciones se basa en el empleo de estos estándares y en el uso de configuración a nivel de servidor para permitir que cualquier mota, con sus especificaciones HW propias, sobre cualquier sistema operativo, pueda integrarse en la herramienta de gestión, adaptando la aplicación creada para OpenWSN y OpenMote a la plataforma y S.O. correspondiente.

Para lograr este objetivo se ha realizado un estudio de los estándares IoT para una mejor compresión de las redes WSN y del trabajo a realizar, y de OpenWSN y OpenMote para conocer y en algún caso ampliar las funcionalidades existentes y buscar una solución que refleje en mejor medida las capacidades que se pueden requerir a una interfaz remota de una res WSN.

En el aspecto de la interfaz de usuario y de la aplicación del lado del servidor, se ha buscado una arquitectura y un desarrollo que siga las tendencias actuales de aplicaciones web, con aplicaciones de una sola página SPA, que delegan parte de la lógica al cliente web, y comunicaciones REST. La decisión inicial de desplegar el servidor en una Raspberry PI ha impuesto una serie de decisiones de diseño adicionales que se han respetado a lo largo del proyecto.

Ha quedado fuera de este Trabajo Fin de Master la modificación dinámica del software de las motas o programación over-the-air OTA. Es en este punto donde se puede realizar un mayor avance futuro, siguiendo por ejemplo el ELF-loader de Contiki.

También es un punto de avance posible la adaptación de la aplicación en la mota a otras plataformas HW y otros S.O. (como Contiki) para generalizar su aplicabilidad.

9. Glosario

AJAX: Asynchronous JavaScript And XML

BBDD: Base de datos

CoAP: Constrained Application Protocol

CSS: Cascading Style Sheets

HTML: HyperText Markup Language

IoT: Internet Of the Things

ISM: Industrial, Scientifical, Medical

JSON: JavaScript Object Notation

M2M: Machine to Machine

NesC: Network Embedded Systems C

ORM: Object-Relational Mapping

REST: REpresentational State Transfer

S.O.: sistema operativo

SQL: Structured Query Language

SPA: Single Page Application

SVG: Scalable Vector Graphics

WPAN: Wireless Personal Area Network

WSN: Wireless Sensor Network

10. Bibliografía

[1] An Open and Integrated Management Platform for Wireless Sensor Networks

Marzo 2009 *(Biblioteca de la UOC)*

M. Kalochristianakis, V. Gkamas, G. Mylonas, S. Nikoletseas, E. Varvarigos

http://0-ieeexplore.ieee.org.cataleg.uoc.edu/xpls/abs_all.jsp?arnumber=5207348&tag=1

[2] TinyDB

Abril 2016

http://telegraph.cs.berkeley.edu/tinydb/

[3] Crossbow: MoteWorks Getting Started Guide

Noviembre 2010

Catherine Greene, Bretny Khamphavong, Chloe Norris, Nancy White

http://www.radford.edu/nsrl/creu1011/PowerPoints/MoteWorks.pdf

[4] Cougar

Abril 2016

http://www.cs.cornell.edu/bigreddata/cougar/index.php

[5] RESTful SensorWeb Enablement Services forWireless Sensor Networks

Junio 2012

Mohsen Rouached, Sana Baccar, Mohamed Abid

https://www.researchgate.net/publication/236658477_RESTful_Sensor_Web_Enablement_Services_for_Wireless_Sensor_Networks

[6] Sensor Web Enablement (SWE)

Abril 2016

http://www.opengeospatial.org/ogc/markets-technologies/swe

[7] TWIST

Abril 2016

http://www.twist.tu-berlin.de/testbeds/sensor.html

[8] OpenMote

Abril 2016

http://www.openmote.com/

[9] Implementación del Mecanismo de Acceso al Medio (MAC) IEEE 802.15.4e CSL (Coordinated Sampled Listening) sobre OpenWSN y Plataforma OpenMote

Enero 2015 *(Trabajo Fin de Master, Biblioteca de la UOC)*
Sergio Gonzalo San José
http://openaccess.uoc.edu/webapps/o2/handle/10609/40043

[10] IoT: TECNOLOGÍAS, usos, tendencias y desarrollo futuro
Noviembre 2014 *(Trabajo Fin de Master, Biblioteca de la UOC)*
David Bliznakoff del Valle
http://openaccess.uoc.edu/webapps/o2/handle/10609/40044

[11] Terminology in IPv6 over the TSCH mode of IEEE 802.15.4e
Marzo 2016
P. Thubert, T. Watteyne, Q. Wang
http://www.potaroo.net/ietf/ids/draft-ietf-6tisch-terminology-07.txt

[12] Internet of Things: 802.15.4, 6LoWPAN, RPL, COAP
Octubre 2010
Jürgen Schönwälder
https://www.utwente.nl/ewi/dacs/colloquium/archive/2010/slides/2010-utwente-6lowpan-rpl-coap.pdf

[13] RFC7252 The Constrained Application Protocol (CoAP)
Junio 2014
Z. Shelby, K. Hartke, C. Bormann
https://tools.ietf.org/html/rfc7252

[14] Understanding The Internet Of Things
Mayo 2013 *(Requiere registro)*
Ronak Sutaria, Raghunath Govindachari
http://electronicdesign.com/print/communications/understanding-internet-things

[15] OpenWSN
Abril 2016
https://openwsn.atlassian.net/wiki/display/OW/Home

[16] Wikipedia Raspberry PI
Abril 2016
https://en.wikipedia.org/wiki/Raspberry_Pi

[17] Wikipedia Python
Abril 2016
https://es.wikipedia.org/wiki/Python

[18] Django vs Flask vs Pyramid: Choosing a Python Web Framework
Abril 2016

Ryan Brown
https://www.airpair.com/python/posts/django-flask-pyramid

[19] AngularJS
Abril 2016
https://angularjs.org/

[20] Bootstrap
Abril 2016
http://getbootstrap.com/

[21] SQLite
Abril 2016
https://www.sqlite.org

[22] Jasmine
Abril 2016
http://jasmine.github.io/2.0/introduction.html

[23] Plot.ly
Abril 2016
https://plot.ly/javascript/

[24] OpenLayers
Abril 2016
http://openlayers.org/

[25] Wikipedia TinyOS
Mayo 2016
https://en.wikipedia.org/wiki/TinyOS

[26] CC2538 Peripheral Driver Library User's Guide
Mayo 2016
http://www.ti.com/lit/ug/swru325a/swru325a.pdf

[27] The dynamic loader
Mayo 2016
https://github.com/contiki-os/contiki/wiki/The-dynamic-loader

11. Anexos

I. Entorno de desarrollo cliente WEB

No es necesario un entorno de desarrollo específico, basta con un editor de texto y un navegador web.

No obstante, para trabajar con AngularJS es necesario por cuestiones de seguridad un servidor web para acceder a las páginas. Para este proyecto se opta por el servidor json-server creado en node.js. Este servidor tiene la ventaja de que ofrece un servidor de recursos JSON, lo que facilita el desarrollo por separado de la parte cliente.

Para instalar este servidor es necesario:

1. Instalar el componente con el comando *npm install -g json-server*
2. Crear un fichero (por ejemplo *db.json*) con la base de datos JSON que va a gestionar el servidor (e puede incluir un fichero de rutas para crear rutas adicionales).
3. Crear un subdirectorio llamado *public* donde se despliegan los componentes estáticos
4. Arrancar el servidor en el directorio con el comando *json-server db.json.* Si no se modifica en el comando de arranque, el puerto por defecto es el 3000.

   Los componentes utilizados para el desarrollo del cliente web son

   - AngularJS 1.5.2: https://angularjs.org/
   - UI-Route 0.2.18: https://cdnjs.com/libraries/angular-ui-router
   - UI-BootStrap TPLS 1.2.5: https://angular-ui.github.io/bootstrap/
   - Bootstrap 3.3.6: http://getbootstrap.com/getting-started/#download (sólo hojas de estilos)
   - D3 3.5.16: https://d3js.org/
   - PLOT.LY 1.7.1: https://cdn.plot.ly/plotly-latest.min.js
   - OpenLayers 3.15.1: http://openlayers.org/

Existen URLs fijas para estos componentes pero se ha optado por su descarga para facilitar el desarrollo.

II. Entorno de desarrollo interfaz WEB

El entorno de desarrollo es Python. Puesto que Python es multiplataforma el desarrollo puede realizarse sobre cualquier sistema operativo que soporte Python, aunque el despliegue de la aplicación de desea hacer sobre Raspberry PI con Raspbian como sistema operativo.

Raspbian viene con Python 2.7 y Python 3 instalados por defecto. Si se usa otro entorno se pude descargan Python para la plataforma deseada desde https://www.python.org/downloads/

No es necesario instalar un IDE, se usará IDLE que viene por defecto con la instalación de Python.

El framework web elegido es Flask. Para la instalación es necesario ejecutar el comando *pip install Flask* que descarga Flask y sus dependencias.

Para el cliente/servidor CoAP se va a utilizar la implementación que proporciona OpenWSN, que puede descargarse desde https://github.com/openwsn-berkeley/coap o bien clonado el repositorio Git de openWSN. Puesto que se van a hacer modificaciones en el cliente se despliega junto a la aplicación web sin instalarlo.

Es necesario instalar para que funcione el cliente CoAP el componente PyDispatcher https://pypi.python.org/pypi/PyDispatcher. Se descarga, se descomprime y se ejecuta *python setup.py install*.

Para la ejecución de las pruebas unitarias automáticas es necesario instalar el módulo Pytest http://pytest.org/latest/ mediante el comando *pip install pytest*

III. Entorno de desarrollo de la aplicación en la mota

El entorno de desarrollo será el IDE Eclipse C. Las instrucciones para la instalación del entorno están detalladas en el capítulo 7.1 de [9]. Es necesario tener en cuenta que Eclipse utiliza la máquina virtual de Java; las instrucciones para su instalación pueden encontrarse en http://www.ubuntu-guia.com/2012/04/instalar-oracle-java-7-en-ubuntu-1204.html

Por otro lado, al menos en Ubuntu.14.04 el comando sudo no respeta el PATH del usuario sino que tiene un PATH propio; para trabajar de manera sencilla se puede definir un alias en el fichero .bashrc

```
scons='sudo env PATH=$PATH scons'
eclipse='sudo env PATH=$PATH eclipse'
make='sudo env PATH=$PATH make'
```

De esta manera se mantiene el PATH definido para el usuario donde se ha incluido el cross-compiler para ARM. La compilación desde Eclipse no funciona adecuadamente si no se lanza con sudo, por eso se define un alias.

IV. Instalación de la aplicación

El primer paso es instalar el entorno Python que se va a utilizar. Se deben instalar Flask y PyDispathcher como se indica en el anexo II (no es necesario instalar pytest puesto que no se van a ejecutar las pruebas unitarias).

Los parámetros de configuración de la aplicación a revisar se encuentran en el fichero cfg/uocapp.cfg:

- DATABASE: fichero que va a contener la BBDD Sqlite3
- SECRET_KEY: clave para la gestión de sesiones
- SERVER_PORT: puerto en que escucha la interfaz web
- UDP_PORT_COAP_SERVER: puerto en el que escucha el servidor CoAP
- UDP_PORT_COAP_CLIENT: puerto
- IMG_MAPA: fichero que va a contener la imagen del mapa

Es necesario iniciar la BBDD; para ello se dispone del comando *python uocapp_init.py*. El fichero que carga la definición de la BBDD que está ubicada en *db/uocapp.sql*.

La parte cliente puede desplegarse junto a la interfaz web, desplegando los ficheros en el directorio *static* (en este caso la URL de acceso será /static/index.html). La configuración a revisar está ubicada en el fichero *uocapp_constantes.js*

- baseURL: si se despliega junto a la interfaz web el parámetro puede contener como dirección "/", si no se despliega conjuntamente debe modificarse con la ip:puerto en la que escuche la interfaz web.
- *RegXPagina*: número de registros por página
- *Xyparams*: ids de los parámetros que representan la posición [X,Y] de la mota en el mapa
- *tiempoInactiva*: tiempo que debe pasar (en ms) para que una mota se considere inactiva y se alerte como tal al usuario

V. Modificación de la implementación CoAP Python de OpenWSN

Para el proceso de descubrimiento tal y como se ha definido es necesario que el servidor, que utiliza la implementación CoAP Python de OpenWSN, a la hora de tratar un mensaje pueda recuperar la dirección de la mota que lo envía.

Para obtener esta funcionalidad ha sido necesario modificar la implementación CoAP en estos puntos:

coapResource.py

Se ha incluido como parámetro de los métodos asociados al tratamiento de los mensajes GET, POST, PUT y DELETE la tupla sender(ipv6,puerto) del que envía el mensaje.

```
def POST(self,options=[],payload=None,sender=None):
```

coap.py

Dentro del método privado *_receive* se ha modificado la invocación al método correspondiente del coapResource para que incluya el sender como parámetro.

```
elif message['code']==d.METHOD_POST:
    (respCode,respOptions,respPayload) = resource.POST(
        options=message['options'],
        payload=message['payload'],
        sender=sender
    )
```

VI. Monitor de baterías en OpenMote y OpenWSN

El SoC CC2358 de Texas Instruments incluye la opción para facilitar la creación de un monitor de las baterías [26].

El valor leído con la opción SOCADC_VDD al leer el ADC mediante la función *SOCADCSingleStart* es, según la documentación, AVDD5/3 (donde AVDD5 corresponde al pin 33 del CC2358; este pin se puede utilizar como referencia externa del ADC, pero no para este caso).

Por tanto leyendo este valor con la referencia interna del ADC (que es aproximadamente 1.19V según el datasheet) podemos obtener la medida del voltaje de las baterías, siempre que el pin 33 esté conectado a VDD (lo que es así según el esquema de OpenMote-CC2538).

Otra alternativa es conectar a alguno de los pines válidos para la entrada del ADC (AIN1 a AIN7 que corresponde a las entradas PA1 a PA7, pines 17 a 23) el voltaje de entrada a través de un divisor de tensión resistivo que baje la entrada en el pin a un máximo de 1.1V aproximadamente y leer esa entrada con el ADC).

Puesto que la primera opción es más sencilla y el conexionado ya está disponible en OpenMote-CC2538 se va a utilizar esta opción.

La modificación en OpenWSN supone:

sensors.h dentro de bsp/boards.

Definir un nuevo sensor y modificar el número de sensores declarados (además en board_info.h se comenta la definición de NUMSENSORS para que el valor de sensors.h sea el que determine este valor).

```
#define NUMSENSORS 9

enum {
  SENSOR_ZERO,              // new sensor types go after this one
  ...
  SENSOR_ADCVDD
  ...
  SENSOR_LAST               // new sensor types go before this one
};
```

adc_sensor.h dentro de bsp/boards/OpenMote-CC2538.

Definir el prototipo de la función para leer el nuevo sensor. Solo se va a definir la función de lectura, no la de conversión.

```
uint16_t adc_sens_read_vdd(void);
```

adc_sensor.c dentro de bsp/boards/OpenMote-CC2538.

Definir la implementación para la lectura del sensor. Es similar a la ya existente para el sensor interno de temperatura (y similar al ejemplo incluido en la documentación del CC2538) cambiando la definición del sensor a leer. No se modifica la función de inicialización, respetando declaración de uso de la referencia interna del CC2538 y la precisión de 12 bits.

```
uint16_t adc_sens_read_vdd(void) {
  uint16_t ui16Dummy;

  SOCADCSingleStart(SOCADC_VDD);
  while(!SOCADCEndOfCOnversionGet());
  ui16Dummy = SOCADCDataGet() >> SOCADC_12_BIT_RSHIFT;
  return ui16Dummy;
}
```

sensors.c dentro de bsp/boards/OpenMote-CC2538.

```
callbackRead_cbt sensors_getCallbackRead(uint8_t sensorType) {
  switch (sensorType) {
    ...
    case SENSOR_ADCVDD:
      return &adc_sens_read_vdd;
    default:
      return NULL;
  }
}
```

Si la referencia interna es aproximadamente de 1.19V y la lectura es VDD/3, el valor leído en Voltios corresponde a V = v * 1.19 * 3 / 2047, donde v es el valor leído y 2047 es la resolución en 12 bits (de 2047 a -2048). Por ejemplo en la figura 7.2.8 se puede ver que el valor leído es 1570 (2.738V); es ese momento la tensión leída en batería del OpenBattery con el polímetro era de 2.84V.

VI. Actualización dinámica del software

La actualización dinámica de software supone definir un proceso que va a ser capaz de cargar un un área de memoria que contiene código y pasar a ejecutar dicho código una vez cargado.

Puesto que el código reside en memoria, es necesario dividir la misma de modo que se identifique el área en la que el código actual se está ejecutando, el área del bootloader y el área de código donde se guarda la nueva versión del software. En algunas ocasiones se define una "golden Image", una versión del software cuyo funcionamiento esta garantizado, que se genera en la carga inicial del software junto con el bootloader y que es la que se ejecuta en caso de detectar un error. Esto supone un aumento de la memoria ocupada por el sistema y deja menos memoria libre a las aplicaciones.

En este tipo de procesos se incluyen validaciones para comprobar que el área de memoria con el nuevo software a cargar es valido, basadas en CRC, en funciones Hash o criptografía.

El módulo ELF-loader de Contiki facilita la actualización dinámica de software, tanto de módulos como de una aplicación en su conjunto.

Se base en la notación ELF (Executable Linkable Forma) que es un formato estándar usado en sistemas UNIX (Linux, FreeBSD; etc.) y en sistemas embebidos. El componente "dynamic loader" es el encargado de enlazar y los elementos del ELF en la imagen del sistema.

El api del ELF-loader se basa en la función *elfloader_load*, que se encarga de cargar y realizar el enlazado dinámico del objeto representado por un fichero en la imagen del sistema; esto supone la modificación del fichero origen.

Para poder usar el ELF-loader es necesario al compilar el firmware prepararlo para la carga dinámica, forzando que se genera una tabla de símbolos que pueda utilizarse en el enlazado dinámico, ya que por defecto COntiki incluye una tabla de símbolos vacía.

Cada plataforma HW que admita el uso del ELF-loader de Contiki debe implementar una serie de funciones internas para la carga en memoria y la enlazado de los objetos.

Otra visión es la que utiliza la plataforma Thingsquare, que no se basa en el ELF-loader sino que las aplicaciones se cargan en áreas de memoria pre-configuradas y se compilan contra una versión del firmware como módulos estáticos. La versión del firmware contra el que se compila la aplicación y la que está desplegada en el dispositivo debe ser por tanto la misma.