INSTITUTO DE COMPUTACIÓN - FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY

SISTEMA PARA EL ANÁLISIS DE DATOS DE SENSORES EN EL AGRO

Informe de Proyecto de Grado

Andrés Vera

Maite Ibarburu

CLIENTES:

Fernando Silveira

Leonardo Steinfeld

Javier Schandy

Tutora: Msc. Ing. Raquel Sosa

Resumen

El Internet de la cosas (Internet of Things, IoT) es un concepto cada vez más escuchado en el mundo del software, y es de interés aplicarlo en el medio agrario. En dicha área es importante poder monitorear los cultivos, para lo cual es necesario medir los factores climáticos que pueden afectarlos y a partir de esta información, determinar diferentes acciones y caminos a seguir.

Es por esto que en el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República, se está llevando a cabo un proyecto de investigación –Gervasio- cuyo objetivo general es el monitoreo de las características ambientales de los agro cultivos a través de una red de sensores. Este proyecto de grado surge como una necesidad del proyecto Gervasio, para proveer un sistema que permita el monitoreo de los cultivos mediante la comunicación con esta red a través del uso del protocolo CoAP.

Es por esto que se diseña un sistema de software que se comunica con una red de sensores inalámbricos ubicados físicamente en el terreno de los cultivos; que permite obtener y almacenar la información de los sensores, las medidas registradas y el estado de la red. Y luego esta información se visualiza y gestiona desde un portal web. A través de este portal los ingenieros agrónomos acceden a la información de las medidas que registran los sensores, tanto para su análisis como también para la creación de alarmas que notifican sobre medidas que requieran una rápida intervención. También desde el portal web, los técnicos administradores de la red son capaces de controlar el estado de la misma y los sensores allí dispuestos.

En este proyecto de grado se logra validar el funcionamiento de este sistema en un ambiente de pre-producción instalado en la Facultad de Ingeniería en conjunto con integrantes del proyecto Gervasio.

Palabras Clave: *Internet de las cosas*, *Red de Sensores inalámbricos*, *CoAP*, *Monitoreo de cultivos*

Contenido

1. Introducción

En este documento se presenta el proyecto de grado denominado "Sistema para el análisis de datos de sensores en el agro" el cual se enmarca en el área de Sistemas de Información. En este capítulo se presenta el contexto en el que se realizó el proyecto, la motivación del mismo, así como los objetivos y los aportes realizados.

1.1 Contexto y Motivación

El concepto de IoT se refiere al uso de internet para la comunicación de las personas con los objetos en la vida cotidiana. Por ejemplo se puede aplicar a una cafetera que notifique mediante una aplicación de Smartphone la cantidad de café que queda en la misma, o a una aplicación mobile que permite encender los dispositivos de aire acondicionado antes de llegar al hogar.

Un informe publicado por la empresa multinacional de IT Cisco [1], indica que apenas un 1 % (14.000 millones) de los objetos del mundo se encuentran conectados a la red y este mismo informe estima que la cifra crecería hasta 50.000 millones en el 2020.

El mundo del agro no está ajeno a esta realidad, es creciente el interés por conocer qué factores afectan la producción agrícola y poder generar así información relevante para la toma de decisiones sobre los cultivos. Por lo tanto, resulta de suma importancia realizar el seguimiento de los mismos, lo que implica conocer las características medioambientales que lo rodean, temperatura, humedad del suelo o del aire, entre otras. También es importante identificar aquellas características que pueden afectar el adecuado crecimiento del cultivo. Para alcanzar el conocimiento de los factores antes mencionados, es necesario aplicar en el entorno agropecuario el concepto de IoT.

En el Instituto de Ingeniería Eléctrica (IIE) [3] de la Facultad de Ingeniería de la Universidad de la República (UdelaR) [4], se está llevando a cabo un proyecto de investigación –Gervasio [2]- cuyo objetivo general es el monitoreo de las características ambientales de los agro cultivos. El proyecto Gervasio está enfocado en el diseño e implementación de una red de sensores, es por esto, que este trabajo de grado surge por la necesidad de realizar la implementación de un sistema capaz de comunicarse con esta red, mediante el protocolo de aplicación CoAP, y permitir el almacenamiento y gestión de los datos obtenidos a través de un portal web.

Más concretamente se necesita construir un software capaz de almacenar, procesar y dar acceso a la información generada por las redes de sensores, implantados físicamente en los cultivos de producción agrícola.

1.2 Objetivos

El objetivo general de este proyecto es el diseño y la implementación de un sistema de software que realice la comunicación con la red de sensores ubicados en el campo. Este debe almacenar la información recibida y permitir a los usuarios visualizar los datos y gestionar las redes a través de un portal web

Para cumplir con el objetivo general se plantean los siguientes objetivos específicos:

- Relevar y conocer los conceptos que se manejan en el protocolo CoAP, para la comunicación desde y hacia los sensores de la red.
- Diseñar una arquitectura distribuida, que permita la comunicación con la red de sensores, asegurando que no se pierda información.
- Validar la arquitectura planteada mediante la implementación de la misma.
- Instalar el sistema en un ambiente de prueba o de pre-producción.
- Validar con integrantes del proyecto Gervasio la solución realizada mediante pruebas del sistema.

1.3 Resultados Esperados

Los resultados que se esperan alcanzar son los siguientes:

- Profundizar en el estudio del protocolo CoAP y sus aplicaciones.
- Conocer las variables y características que se desean medir y monitorear en el ciclo de vida de un cultivo.
- Diseñar e implementar un sistema distribuido con las funcionalidades requeridas por el cliente.
- Abordar y profundizar en tecnologías de front-end para el desarrollo de aplicaciones web.
- Completar un ciclo de desarrollo de software identificando las diferentes etapas del mismo.
- Implementar un sistema que permite almacenar, gestionar y visualizar los datos obtenidos de factores climáticos proporcionados por una red de sensores.
- Instalar y probar el sistema desarrollado en un ambiente de pre-producción.

1.4 Aportes del Proyecto

A continuación se detallan los principales aportes del proyecto:

- Se profundizó en el protocolo CoAP para la comunicación con dispositivos de bajos recursos (nodos de la red).
- Se investigó acerca de ordenadores de placa reducida, en particular se trabajó con Raspberry Pi. Se pudo conocer cómo funcionan y poner en práctica la instalación de aplicaciones de prueba sobre el mismo, para conocer más a fondo su funcionamiento y estructura.
- Se obtuvieron conocimientos acerca de parámetros de monitoreo de una red de sensores, las diferentes magnitudes que se desean conocer y cuáles son los significados de las mismas.
- Se implementó un sistema siguiendo la arquitectura definida, con las funcionalidades requeridas.
- Se puso en práctica la implementación de una aplicación distribuida en un entorno de pre-producción.

1.5 Estructura del Documento

En el capítulo 2 se presenta el marco conceptual, identificando y describiendo los principales conceptos utilizados en el proyecto. En el Capítulo 3 se presenta el análisis de los requerimientos del sistema. En el Capítulo 4 se detalla la solución propuesta, los pasos seguidos en la implementación y las herramientas utilizadas. En el Capítulo 5 se presenta el esquema de pruebas y validación utilizados durante el desarrollo el proyecto. En el Capítulo 6 se muestra el plan de gestión del proyecto y la evolución del mismo. Para finalizar, en el Capítulo 7 se presentan las conclusiones del proyecto así como posibles mejoras y trabajo a futuro.

2. Marco Teórico

En este capítulo se presentan los conceptos, protocolos y tecnologías utilizadas, con el objetivo de brindar un marco conceptual para una mejor comprensión del documento.

Los conceptos mencionados, se estudiaron para conocer las diferentes tecnologías y protocolos involucrados y relacionados con el problema a resolver.

2.1 Wireless Sensor Network

Wireless Sensor Network (WSN) es una red inalámbrica de nodos (autónomos) equipados con sensores que se utilizan para el monitoreo y control de fenómenos ambientales y/o físicos, permitiendo la interacción del entorno con las personas [5].

Gracias al avance de la microelectrónica estos nodos son dispositivos pequeños, con procesador y memoria de bajos recursos, capaces de obtener información de su entorno y comunicarla a través de la red inalámbrica. Esto es realizado mediante un bajo consumo de energía debido al manejo de hardware y software adecuado.

Típicamente las redes de sensores están conformadas por nodos (también denominados motes) de sensores, un nodo base y un gateway, como se muestra en la figura 1.

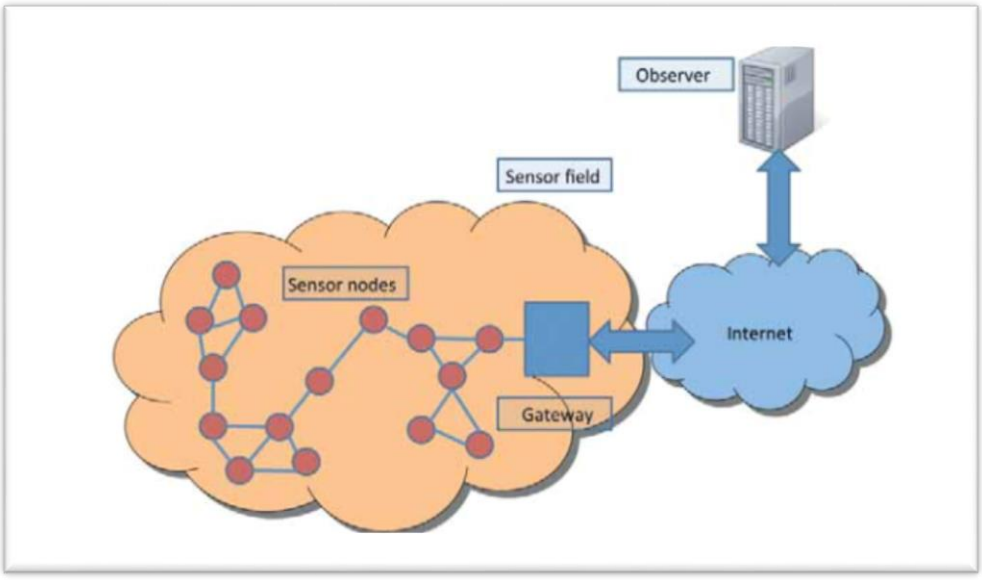

FIGURA 1. RED DE SENSORES. IMAGEN EXTRAÍDA DE [5]

En la actualidad, la construcción de una WSN es cada vez más sencilla de alcanzar debido a la rápida evolución tecnológica, los bajos costos y el foco de desarrollo de estándares para WSN (protocolos, sistemas operativos y software para nodos de red, etc.).

En un principio las WSN eran utilizadas principalmente en el ámbito militar, pero paulatinamente también se fueron utilizando en ámbitos industriales y comerciales, y se espera a futuro que estas redes lleguen a ser de tan fácil acceso que se utilicen en aplicaciones caseras.

En las WSN los nodos son distribuidos en el área que se desea monitorear, la misma puede ser de gran magnitud debido a su característica de red ad hoc, por lo tanto no depende de infraestructura ajena a sí misma para el ruteo de datos. El ruteo lo realiza cada uno de los nodos de la red y es dinámico, basado en la conectividad de cada uno. Esto hace posible mover los nodos de la red sin mayores dificultades, ya que por sí misma ajusta el ruteo de mensajes para que se alcance correctamente el destino.

Este ruteo dinámico también permite que se agreguen o quiten nodos de la red “on the fly” sin afectar su correcto funcionamiento. En particular, ser una red auto configurable la hace tolerante al fallo de nodos, por ejemplo, si un nodo falla y otro requiere del mismo para el ruteo de datos, estos pasarán a utilizar otro nodo.

Este tipo de redes tienen la capacidad de autodiagnóstico, autoconfiguración y auto restauración, de forma tal que la red (en su conjunto) se mantenga estable, a pesar de la falla de nodos particulares.

2.1.1 Nodo sensor

Los nodos sensores son dispositivos autónomos que constan de un procesador, memoria, una fuente de energía (generalmente una batería), un radio trasmisor y receptor, y sensores (figura 2) [6].

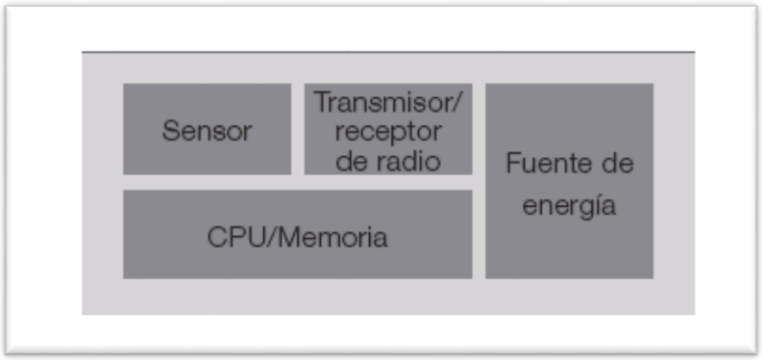

FIGURA 2. ESTRUCTURA NODO SENSOR. IMAGEN EXTRAÍDA DE [6]

Las redes de sensores apuntan a que sean redes de manejo desatendido, con la mínima intervención humana. Por esta razón, se desea que la batería del nodo tenga la mayor vida útil posible, y es así que su hardware y software se construyen teniendo en cuenta la conservación de la energía, con el objetivo de un bajo consumo de recursos.

Al ser tan dependiente de la vida de la batería, es altamente probable que algún nodo de la red se apague en cierto momento, por lo que es de gran importancia que este tipo de redes sean tolerante al fallo de nodos particulares. De forma que si alguno queda sin batería, el transporte de información pueda encontrar otro camino a destino.

2.1.2 Contiki OS

Contiki OS es un sistema operativo de código abierto, desarrollado para ser utilizado en dispositivos de pequeño porte [7]. Fue diseñado para utilizarse en sistemas embebidos de escasa memoria y recursos, como es el caso de los nodos de una red de sensores, que constan típicamente de un microcontrolador de 8-bits y 20 KB de RAM.

En ambientes de poca memoria, una operación que requiere varios hilos consume gran parte de la misma. Esto se debe a que cada hilo debe utilizar su propio stack y como en general es difícil conocer la cantidad exacta de memoria necesaria para correr un proceso, cada hilo asegura su stack mediante la asignación de mayor memoria de la que precisaría para correr. Una vez asignada esta memoria a un hilo, no es posible que otro la utilice.

Contiki, para brindar la concurrencia de hilos, sin necesidad de stacks por cada uno, optó por un diseño de manejo de eventos. En este sistema, los procesos son implementados como controladores de eventos. Como estos no se bloquean, todos los hilos hacen uso del mismo stack, compartiendo entre ellos, la escasa memoria del dispositivo [8].

2.2 Web Service REST

Un web service (o servicio web) es un software cuyo propósito es ofrecer funcionalidades o servicios, diseñado para la interoperabilidad de aplicaciones a través de la red [9]. El proveedor del web service, ofrece su servicio y el cliente es capaz de consumirlo de forma remota a través de la web.

Para mejorar la interoperabilidad entre aplicaciones a través de servicios, existen organizaciones como OASIS y W3C, que son responsables de la arquitectura y reglamentación de estos. REST (Representational State Transfer) fue definido en el 2000 por Roy Fielding, también coautor principal de la especificación HTTP. Se puede considerar REST como un framework para construir aplicaciones web, respetando HTTP.

REST es un estilo de arquitectura que especifica estándares (tales como interfaces uniformes), que provee al web Service de cualidades deseables, por ejemplo, escalabilidad, buena performance y portabilidad [10].

En un RESTful web service, las funcionalidades y los datos son considerados como recursos, accesibles a través de una URI (Uniform Resource Identifiers). Estos recursos son manipulados con un conjunto de operaciones: PUT, GET, POST, DELETE, donde cada

uno equivale respectivamente a una operación CRUD (Create, Read, Update y Delete). La interacción en este tipo de servicios es sin estado, de forma que cada mensaje es independiente y contiene la información necesaria para responder su pedido.

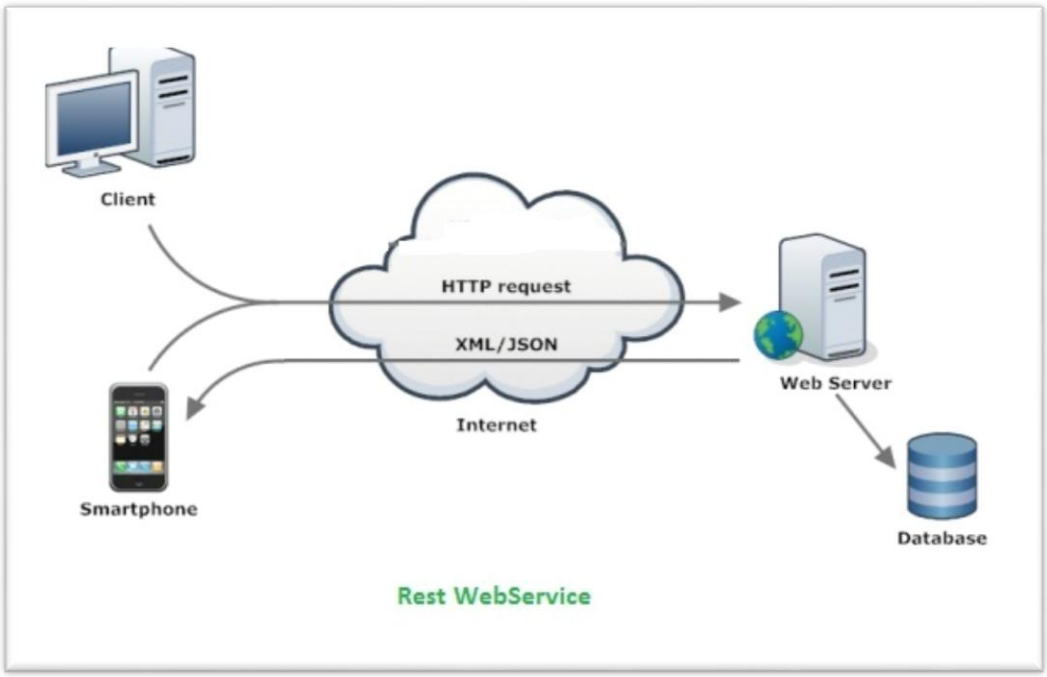

FIGURA 3. FLUJO DE COMUNICACIÓN RESTFUL WEB SERVICE. IMAGEN EXTRAÍDA DE [11]

El proveedor de servicios, define los diferentes recursos que maneja y como se debe acceder a ellos, para que luego los diferentes clientes puedan conectarse con los mismos.

Dicho protocolo, es muy utilizado en la actualidad para definir una API [12] de un determinado sistema. Aplicaciones mundialmente conocidas como son Twitter y Facebook son claros ejemplos de esto. Ellos brindan funcionalidades a través de APIs REST, para la integración con los mismos.

2.3 Middleware

2.3.1 Middleware

Se le llama Middleware a componentes de software que brindan a otros sistemas que fueron diseñados e implementados de forma independiente, la capacidad de interactuar entre ellos. Es decir, un middleware es un intermediario que conecta dos sistemas que se ejecutan en forma independiente uno del otro. [13]

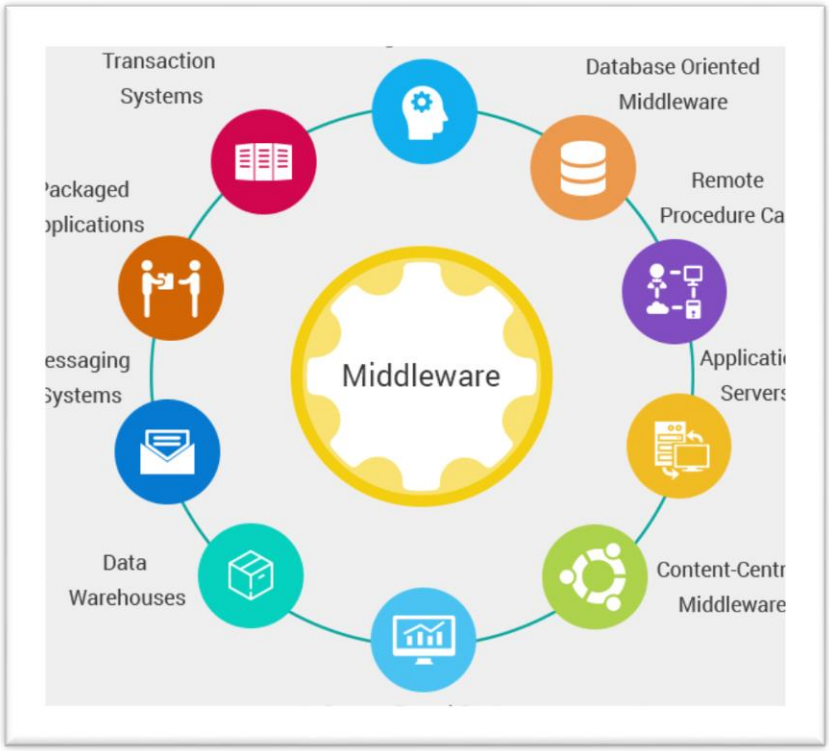

FIGURA 4. ILUSTRACIÓN DE MIDDLEWARE. EXTRAÍDA DE [14]

2.3.2 Message Oriented Middleware

Message Oriented Middleware (MOM) es un middleware que permite a aplicaciones distribuidas comunicarse e intercambiar información a través de mensajes [15].

MOM es una categoría dentro de los posibles middleware que existen y este realiza la comunicación entre sistemas distribuidos a través de mensajería. Los elementos que componen un sistema MOM son por un lado los clientes, por otro, los mensajes y finalmente un proveedor MOM (que brinda una API y una herramienta administrativa).

Al utilizar un proveedor MOM, el cliente hace uso de su API para enviar un mensaje, donde el destino de este es gestionado por dicho proveedor. Una vez enviado el mensaje es responsabilidad del proveedor almacenarlo, hasta que otro cliente lo recupera.

De esta forma, con el uso de un MOM se puede realizar la implementación de un sistema cuyos componentes estén poco acoplados y se comuniquen asincrónicamente mediante mensajes enviados y recibidos a través de este.

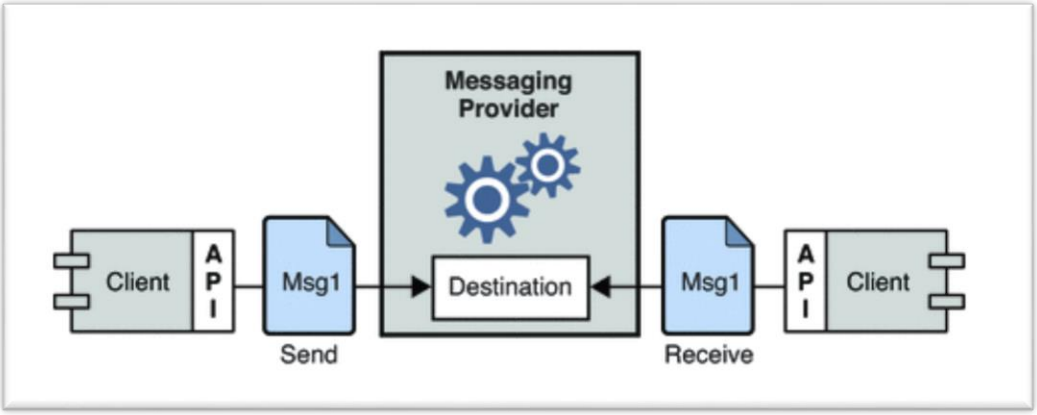

FIGURA 5. COMUNICACIÓN BASADA EN MOM. EXTRAÍDA DE [15]

2.4 Servidor de Mapas

Un servidor de mapas es el encargado de proveer datos tanto vectoriales como raster en proyecciones cartográficas, cumpliendo con el estándar de Web Map Service (WMS) y Web Feature Service (WFS) de Open Geospatial Consortium.

Open Geospatial Consortium (OGC) [16] es una organización internacional encargada de construir estándares abiertos de calidad sobre información geográfica para la comunidad. Estos son estandarizados a través de un proceso consensual, y están disponibles para su uso libre, con el objetivo de mejorar el intercambio mundial de datos geoespaciales.

A continuación se describen dos de los principales estándares que fueron investigados, el Web Map Service y el Web Feature Service.

2.4.1 Web Map Service

El estándar Web Map Service (WMS) [17] proporciona una interfaz HTTP para solicitar imágenes de mapas. Este define un mapa como una representación geográfica en forma de archivo de imagen, estos archivos son generalmente PNG, GIF o JPEG o pueden ser también gráficos vectoriales como SVG.

Un WMS básico clasifica su información geográfica en capas y ofrece un número finito de estilos predefinidos para poder mostrarlas.

El estándar de OGC define tres posibles operaciones. La primera de ellas, denominada “Get Capabilities”, retorna metadatos a nivel de servicio, tales como los formatos de imagen que ofrece o la lista de una o más capas disponibles en el servicio. La segunda, se denomina “Get Map” y retorna un mapa cuya geografía y parámetros dimensionales están bien definidos. En la misma se puede indicar el formato que se quiere, el mapa,

las capas que se desean, así como también el sistema de referencia a utilizar. La tercera y última operación denominada "Get Feature Info" es de carácter opcional. La misma devuelve toda la información sobre las características particulares mostradas en el mapa. Estas operaciones pueden ser invocadas mediante un navegador Web, realizando peticiones a una determinada URL. En particular cuando se solicita un mapa, en dicha URL se colocan algunos parámetros como por ejemplo, qué información mostrar, sistema de referencia de coordenadas y coordenadas a mostrar inicialmente, entre otros parámetros.

2.4.2 Web Feature Service

El estándar Web Feature Service (WFS) [18] define operaciones web de interface para la consulta y edición de entidades geográficas vectoriales, como pueden ser las carreteras.

Este estándar define cuatro posibles operaciones que se pueden realizar. La primera es descubrir colecciones de entidades disponibles en el servidor (Get Capabilities), la segunda, brinda la descripción de los atributos disponibles para cada entidad (Describe Feature Type). La tercera, permite consultar una colección sobre un subconjunto de entidades filtrando la información de manera deseada (Get Feature) y la última, permite agregar, editar o eliminar entidades existentes (Transaction).

Todas las operaciones definidas en el estándar WFS soportan entrada y salida de datos utilizando lenguaje Geography Markup Language (GML).

2.4.3 Geography Markup Language

Geography Markup Language (GML) [19] es una gramática sobre XML para expresar y comunicar características geográficas. GML constituye por tanto, un lenguaje de modelado para sistemas geográficos, así como un formato de intercambio abierto para transacciones de información geográfica a través de Internet.

La capacidad de integrar formas de información geográfica es la clave del formato GML. Es principalmente utilizado por el estándar WFS, para la operación (Transaction y GetFeature particularmente).

2.5 Bases de datos no relacionales

Las bases de datos no relacionales son cada vez más utilizadas en sistemas donde no se requiere que la información esté estructurada y siguiendo algún tipo de modelo. Esto se debe a que las mismas no utilizan esquemas y entidades como lo hacen las bases de datos relacionales. Estas bases de datos, son también denominadas NoSQL. Como el

propio nombre lo indica, no están orientadas al lenguaje de consulta SQL, donde se realizan JOINS de tablas de datos [21].

Estas bases surgen a raíz del crecimiento de la información a nivel mundial, debido principalmente a los servicios en la nube. La necesidad de almacenar grandes cantidades de datos (big data [22]) y soportar su procesamiento posterior, se fue tornando cada vez más difícil con las base de datos tradicionales, por lo que surgen las no relacionales o NoSQL. Una de las principales ventajas que tienen, es que al no poseer ningún tipo de esquema, las operaciones de lectura y escritura son por lo general muy rápidas.

El hecho de no tener esquema, permite almacenar información en múltiples esquemas y formatos, lo que ayuda a abstraerse y definir un modelo de datos genérico en caso de necesitarlo.

A continuación, se describen los cuatro principales formatos que se utilizan para implementar este tipo de base de datos [21].

Orientada a documentos

Son aquellas que gestionan datos semiestructurados, es decir documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON.

Orientada a columnas

Este tipo de bases de datos están diseñadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenan columnas de datos en lugar de registros.

De clave-valor

Este tipo es el más sencillo de comprender. Se almacenan tuplas que contienen una clave y su valor. Cuando se quiere recuperar un dato, se busca por su clave y se recupera el valor.

Orientada a Grafos

Estas bases de datos están basadas en la teoría de grafos, utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales.

2.6 Raspberry Pi

Un Raspberry Pi [23] es una computadora de placa simple, siendo su sistema operativo una versión adaptada de Debian. El objetivo principal de su fabricación fue la creación de una computadora de bajo costo para la enseñanza de programación a escolares.

En la actualidad debido a su relación costo/beneficio, el Raspberry Pi es utilizado además en la realización de proyectos a lo largo del mundo [24]. Estos proyectos varían desde su utilización para emular consolas de videojuegos hasta aplicaciones de domótica (robótica en el hogar).

Estos dispositivos poseen un procesador central y memoria RAM pero no incluyen disco duro, ya que el almacenamiento lo realiza en una tarjeta SD.

El último de los modelos (Raspberry Pi 3), tiene las siguientes especificaciones:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- 1 GB RAM
- 4 puertos USB
- Puerto HDMI
- Puerto Ethernet
- 3.5mm audio jack
- Camera interface (CSI)
- Display interface (DSI)
- Soporta Micro SD

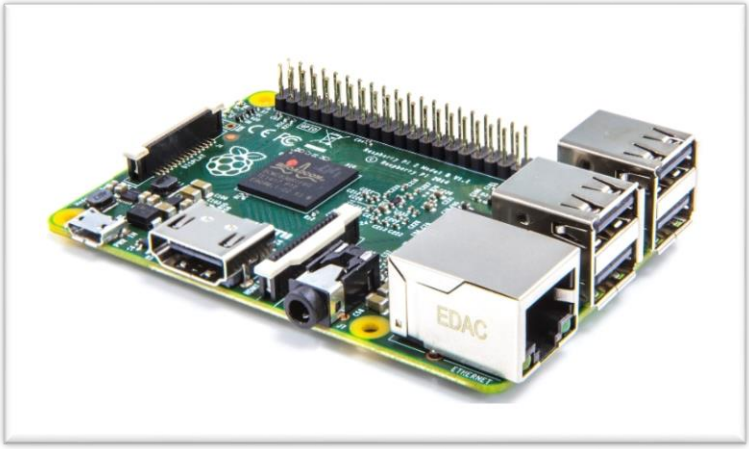

FIGURA 6. RASPBERRY PI [23]

2.7 Constrained Application Protocol

Constrained Application Protocol (CoAP) es un protocolo de capa de aplicación destinado a ser utilizado en dispositivos electrónicos simples (o con recursos limitados), que se desea sean supervisados y controlados de forma remota, a través de Internet [25].

Su uso en dispositivos simples se hace posible ya que el protocolo está diseñado para utilizar un mínimo de recursos, tanto en el dispositivo, como en la red.

2.7.1 Comunicación CoAP

Está basado en la arquitectura REST. Los servidores CoAP publican recursos a través de una URI, y los clientes CoAP acceden a estos recursos utilizando los métodos GET, PUT, POST y DELETE.

La comunicación se realiza a través del protocolo de transporte UDP. Esto, entre otras ventajas, optimiza el número de paquetes necesarios para transmitir en la red, ya que en comparación con TCP, este no necesita establecer previamente la conexión entre las partes. Mientras que en TCP debe hacerse el three-way handshake [26] antes de enviar cualquier dato, en UDP el primer paquete enviado contiene tanto los datos que se desea enviar, como la información del destinatario y no es necesaria la conexión previa entre origen y destino.

Otra de las ventajas del uso de UDP como protocolo de transporte, es el soporte de multicast. Por lo que, de ser necesario, se podrían enviar pedidos a un grupo multicast en lugar de a un sólo cliente por vez.

CoAP está organizado en dos capas [27]. La primera capa denominada transaccional, es aquella que maneja la transmisión del mensaje entre endpoints, el cual puede ser de los siguientes tipos:

- **Confirmable**: requiere un ack (acknowledgment).
- **No-confirmable**: no requiere un ack.
- **Ack**: acknowledgment (reconocimiento).
- **Reset**: indica que se recibió un mensaje confirmable, pero no es posible su procesado.

Por otro lado, está la capa request/response, donde es aplicado REST que maneja la transmisión de un pedido y su correspondiente respuesta, utilizados en la comunicación desde y hacia los recursos (URIs) publicados.

Un pedido REST es transportado por un mensaje Confirmable o No-confirmable, mientras que un response REST es transportado por su correspondiente Ack.

Este enfoque, permite brindar fiabilidad, a pesar de utilizar UDP, ya que un mensaje Confirmable es reenviado (utilizando un timeout), hasta que se reciba su Ack correspondiente.

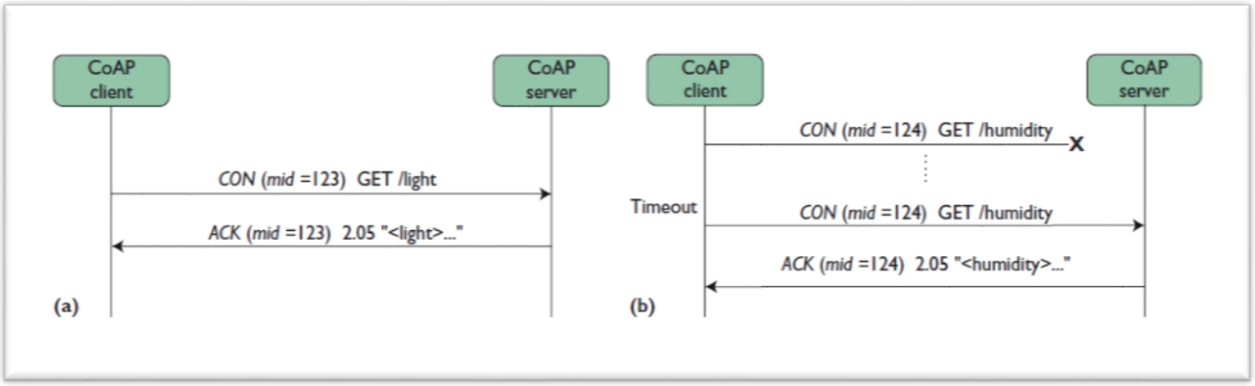

FIGURA 7. INTERCAMBIO DE MENSAJES COAP. EXTRAÍDA DE [25].

La estructura del mensaje CoAP fue diseñada para optimizar su overhead, es decir, el mínimo tiempo de procesamiento posible. Un cabezal de un mensaje CoAP promedio contiene entre 10 y 20 bytes de información, donde se puede encontrar una sección de 4 bytes fija y luego la codificación de opciones.

En cuanto al payload, al igual que HTTP, distintos tipos de formatos son soportados (por ejemplo XML, JSON, CBOR, etc) y allí se permite transferir una gran cantidad de información. Para esto, CoAP lo separa en distintos paquetes y este se transmite en bloques separados. Utilizando "opciones de bloque" en el cabezal del mensaje, se proporciona información del número de bloque que corresponde, por lo que el cliente puede interpretar que los bloques forman un mensaje único. Este manejo de bloques permite una sesión sin estado, ya que alcanza con completar las opciones adecuadas para que se interpreten de forma correcta los mensajes recibidos (sin ser necesario mantener información de la sesión).

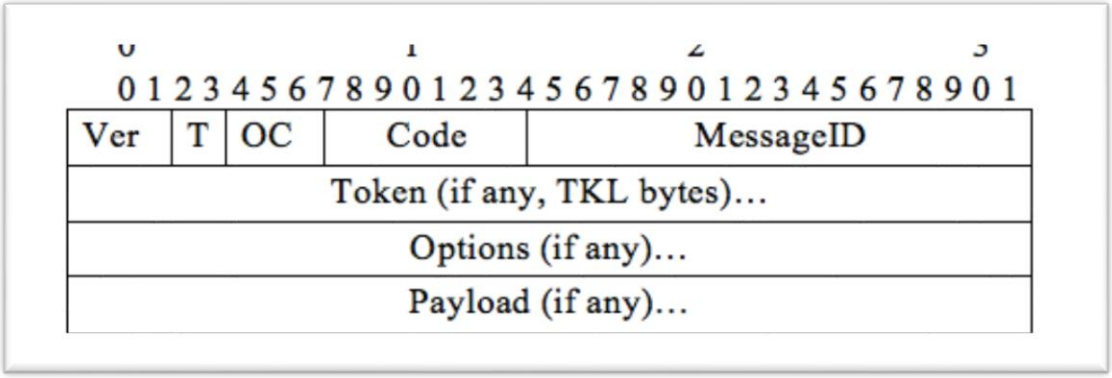

FIGURA 8. FORMATO DE MENSAJE COAP. EXTRAÍDA DE [29].

A continuación se describen los campos del mensaje [29]:

- **Ver (versión):** Versión CoAP.
- **T (tipo):** CON (Confirmable), NON (No-confirmable), ACK (Acknowledgment), RST (Reset).
- **OC (contador de opciones**): número de opciones luego del cabezal.
- **Code (código):** indica si es un mensaje de request o response.
- **MessageID:** utilizado para detectar duplicados.
- **Token:** utilizado para asociar el response con su correspondiente request.

2.7.2 CoAP URIs

El protocolo utiliza *coap* y *coaps* como URI schema [30] para la identificación de recursos CoAP. Éstos son comparables a http y https en el protocolo HTTP.

La sintaxis de las URIs CoAP es la siguiente:

coap://[mote_ip]:puerto/recurso(?query_params)

Donde:

- ***mote_ip:*** es la dirección IPv6 del mote donde está publicado el recuso.
- ***puerto:*** refiere al puerto donde está publicado el recurso.
- ***recurso:*** es la URI del mote sobre el cual se solicita el pedido.
- ***query_params:*** opcionalmente puede haber parámetros en la URI.

Un ejemplo de URI CoAP, puede ser el siguiente:

coap://[aaaa::200:0:0:2]:5683/node/sensors/battery/config?param=param

2.7.3 Discovery

Cuando se trabaja con este protocolo, es de interés de los clientes conocer los recursos que tiene publicado un servidor CoAP, para luego conocer cómo acceder a ellos. Para esto, el protocolo contempla un proceso de descubrimiento de recursos de un servidor específico [31].

El protocolo CoAP ofrece esta posibilidad, de gran importancia para las aplicaciones M2M (machine to machine) que sobre él se desarrollan y establece que los nodos deben soportar el formato Constrained RESTful Environment Link (CoRE Link) de recursos "descubribles" [31]. En este protocolo, se establece que una petición de tipo GET

dirigida sobre el recurso "/.well-known/core" debe ser respondida por el servidor con un mensaje cuyo payload contiene una descripción de los recursos, siguiendo el formato CoRE Link. Este formato contiene ciertos atributos que describen sus recursos, de forma tal que el cliente puede conocer qué recursos tiene a su disposición.

Se puede ver a continuación un ejemplo de respuesta, de un pedido de descubrimiento:

</.well-known/core>;ct=40,
</node>;title="Nodo";obs,
</node/sensors/temperature>;title="Temperatura";obs,
</node/sensors/humidity>;title="Humedad del aire";obs,
</node/sensors/soil_humidity>;title="Humedad del suelo";obs,
</node/sensors/battery>;title="Battery status";rt="Battery",
</network/status>;title="Network Status";obs;rt="Control",
</network/routetable>;title="Network Routetable";obs;rt="Control",
</network/neighbours>;title="Network Neighbors";obs;rt="Control"200",
</network/stats>;title="Network Status";obs;rt="Control"

En el mismo, se encuentran los recursos publicados junto con sus propiedades:

<URI_RECURSO>;propiedad1=valor,propiedad2=valor,

Adicionalmente, se indica con la propiedad "obs" si el recurso es observable.

2.7.4 Observación de recursos

CoAP hace uso del patrón de diseño "observer". Esto implica la suscripción por parte de un cliente a un recurso (en un servidor) y una vez que el recurso se modifica, el servidor envía el nuevo valor del recurso a todos sus clientes suscriptos. De esta manera se evita el envío de paquetes innecesarios en la red cuando se desea conocer si un recurso fue modificado. En lugar de realizar un polling (enviar GETs al recurso cada cierto tiempo) el servidor avisa, en forma asincrónica, cuándo el recurso se modificó [31].

El servidor debe mantener y actualizar la lista de clientes que quieren recibir notificaciones en función de los mensajes intercambiados con los mismos. Cuando un cliente desea observar un recurso, debe enviar una petición de tipo GET y en esta añadir la opción OBSERVE. En el momento en que el servidor recibe esta petición y encuentra esta opción, el mismo entiende que es una petición de registro del cliente (para el recurso indicado en la URI) y responde con una representación del recurso. Además añade a dicho cliente a la lista de clientes registrados para notificación.

2.7.5 CoAP para WSN

En estudio realizado por W. Colitti, K. Steenhaut, N. De Caro [33] se implementan dos motes servidores, uno con recursos publicados CoAP y otro HTTP. Ambos responden a pedidos GET, las medidas tomadas por los sensores presentes son enviadas en paquetes con payload en formato JSON, dentro del paquete correspondiente a cada protocolo.

Cada mote servidor cuenta con 2 baterías AA y este estudio compara los bytes transferidos por transacción, el consumo de batería y el tiempo de vida de la misma.

En la figura 9 se muestran los resultados obtenidos. Para la misma cantidad de pedidos, utilizando protocolo HTTP, el servidor tuvo 84 días de vida, mientras que al utilizar el protocolo CoAP, el servidor tuvo 151 días.

	Bytes per-transaction	Power	Lifetime
CoAP	154	0.744 mW	151 days
HTTP	1451	1.333 mW	84 days

FIGURA 9. COMPARATIVA COAP Y HTTP. EXTRAÍDA DE [33]

Por lo tanto, el uso de UDP con las ventajas que conlleva y el reducido tamaño de sus paquetes, hacen de CoAP un protocolo adecuado para las WSNs, ya que permite hacer uso óptimo de la batería de los nodos de la WSN, que es uno de los principales problemas a resolver a la hora de implementar una red de sensores.

2.8 Antecedentes

2.8.1 Proyecto Gervasio

El título del proyecto es “GERVASIO: Generalización de las redes de sensores inalámbricos como herramienta de valorización en sistemas vegetales intensivos” [2]. El mismo nace como un proyecto de investigación del Instituto de Ingeniería Eléctrica de la Universidad de la República, y lo integran docentes pertenecientes a dicho instituto.

Este proyecto generaliza la aplicación de la tecnología de redes de sensores inalámbricos en la agricultura, que se utilizó para dos aplicaciones productivas y que sirvieron de ejemplo para demostrar la potencialidad de esta tecnología [34]. A continuación se detallan ambos proyectos:

1. La adquisición y transmisión de las imágenes de trampas adhesivas de insectos, usadas para el monitoreo del nivel de plagas que afectan a frutales. De esta manera se evitan errores humanos en la recolección de estos datos, se hacen disponibles con mayor frecuencia y facilidad (en Internet) los datos permitiendo su uso regional y una mejor generación de alertas tempranas.

2. El monitoreo de condiciones micro-climáticas, humedad de suelos y diámetro de tronco, particularmente orientado a cítricos, pero aplicable también a otros cultivos, para la detección del impacto de heladas y optimización de riego, entre otros. La información se adquiere por una red de sensores inalámbricos de bajo consumo de energía y es transmitida a un servidor accesible vía web a través de un concentrador alimentado por energía solar y conectado a la red celular. El proyecto genera productos tecnológicos, conocimiento y formación de recursos humanos en las áreas técnicas vinculadas a redes de sensores inalámbricos aplicadas al agro.

2.8.1.1 Estrategia del proyecto

La acción de Gervasio se basa en el adecuado uso de los recursos hardware y software disponibles y de la experiencia del grupo de trabajo en implementaciones de redes de sensores inalámbricos.

En el predio se dispone de nodos que se comunican entre sí, formando redes distribuidas alimentadas con pilas pequeñas y en donde se pueden agregar nuevos sensores para monitorear las variables solicitadas. Para monitorear poblaciones de insectos, se equipara cada trampa con un sensor de imágenes adecuadamente seleccionado que toma imágenes de los insectos capturados con una periodicidad de 1 a 2 imágenes por día y las transmite a una base que las retransmite vía enlace celular, a un servidor accesible desde Internet. Para el monitoreo de condiciones agroclimáticas, contenido de volumen de agua en suelo y medición de variaciones de perímetro en troncos, se usa la misma tecnología WSN que en el caso anterior, con diferencias en los protocolos y plataformas hardware y software en función de las diferencias en volúmenes y frecuencia de datos a transmitir.

2.8.2 Proyecto de Ingeniería de Software 2014

En el marco de la asignatura Proyecto de Ingeniería de Software en el año 2014 [35] dos grupos de estudiantes realizaron una prueba de concepto del sistema que se plantea implementar en este proyecto.

Esta asignatura se centra en la ingeniería de software, es decir, en gestionar de forma adecuada el proceso de desarrollo de un equipo con el fin de llevar a completitud un producto. Para este conjunto de estudiantes, el producto implementado tuvo como objetivo: validar la posibilidad de implementación de un sistema, que cumpla con las

necesidades del cliente, utilizando un protocolo de comunicación entre sensores recientemente liberado en el mercado (como lo era CoAP en el 2014).

Dichos proyectos tuvieron como alcance la construcción de un software que se comunique con una única red de sensores (mediante el protocolo CoAP), almacene y procese esta información para luego visualizarla y manejarla desde una interfaz web [36].

Debido a que se trató de una prueba de concepto, la red de sensores que se utilizó no fue una red física con motes y sensores reales, sino que se utilizó un simulador. Dicho prototipo cuenta con un componente web, que muestra la información obtenida desde los nodos de la red. Esta web, cuenta con múltiples tipos usuarios, que en función de ello, pueden realizar diferentes acciones.

Por otra parte, cuando una red es reportada al sistema, el usuario con rol administrador o técnico de la red, es capaz de inicializarla, indicando por ejemplo la ubicación de cada uno de los motes. Una vez realizado esto, es posible visualizar los motes de la red en un mapa, donde se indican algunas variables correspondientes a los mismos, como ser el indicador de batería o los vecinos que tiene cada uno. Además permite graficar la información de las medidas recibidas, de forma tal de poder comparar las mismas provenientes de distintos motes. Si corresponde, cuando una medida es recibida desde la red, el sistema envía una alarma vía SMS o mail [36].

La arquitectura diseñada para este sistema está compuesta por tres componentes. El primero encargado de la comunicación con la red (denominado Demonio), el segundo realiza el procesado y almacenamiento de los datos y es denominado Servidor Central y por último, se encuentra una aplicación web que interactúa con el usuario final. Dichos componentes son independientes entre sí y podrían implantarse en distintos servidores [37].

Ambos proyectos, además de realizar la implementación de dicho sistema lograron mitigar el riesgo de utilizar el protocolo de comunicación CoAP.

2.8.2.1 Limitaciones

Las principales limitaciones que tiene el prototipo construido y que ha motivado la reconstrucción del mismo son las siguientes:

- **Red de sensores simulada**
  La red de sensores con la cual se trabajó fue siempre simulada.

- **Pérdida de conexión y buffer de datos**
  En el componente de recepción de datos de la red no se contempla la pérdida de conexión a internet. Dicho componente se encuentra cerca de la red y tiene conexión limitada a internet, por lo tanto al no contar con un almacenamiento de datos temporales cuando no hay conexión, éstos datos se pierden.

- **Configuración de nodos mediante la web**
  En el prototipo planteado, la comunicación desde el servidor central hacia los nodos de la red, es limitada. Únicamente permite la observación de recursos y no es posible realizar cambios en la configuración de los mismos, como ser el período de muestro o sincronizar la hora de los nodos de la red.

- **Única red**
  El prototipo se diseñó para la comunicación y soporte de una única red de sensores. Por lo que no permite el manejo de más de una red a la vez y todos los usuarios del sistema pueden gestionar esa red.

- **Alarmas**
  El sistema de alarmas permite definir solo alarmas simples. Se indican los usuarios que reciben la notificación, se especifica el mote, el tipo de sensor y el rango de medida deseado.

- **Estandarización de mensaje CoAP**
  En el prototipo construido, los mensajes CoAP intercambiados entre el componente demonio y la red de sensores no cumplen con un estándar de formato conocido. Estos son enviados mediante texto plano con un formato acordado entre los integrantes del proyecto del PIS y los integrantes de Gervasio.

- **Tabla de ruteo**
  El prototipo no maneja como recurso de los motes el concepto de tabla de ruteo.

- **Historiales**
  El prototipo del proyecto de PIS contempla únicamente al recurso medida para el almacenamiento histórico de la información, no tiene en cuenta otros recursos del mote.

2.9 Trabajos relacionados

En esta sección se describen proyectos de similares características a este, donde se realiza la implementación e instalación de una red de sensores inalámbrica, que no sólo toma medidas sino que también se transmiten a un servidor central para su posterior análisis.

En este marco de búsqueda se encuentra el trabajo de Yingli Zhua, Jingjiang Songa y Fuzhou Donga [38] en el cual se implementa una red de sensores en un contexto agropecuario.

Esta red consiste en nodos sensores, encargados de obtener medidas de temperatura y humedad. Estos nodos envían su información recabada al llamado "sink node" o base,

donde los datos pueden ser almacenado hasta que sean enviados a la terminal cliente vía WIFI, trasmisión de radio o directo mediante cable, para que la terminal realice el procesado de los datos.

Los nodos sensores consisten en un sensor que obtiene medidas de una determinada magnitud climática, un micro controlador y un módulo nRF2401 [39] y un chip radio receptor y transmisor, encargado de la comunicación entre nodos.

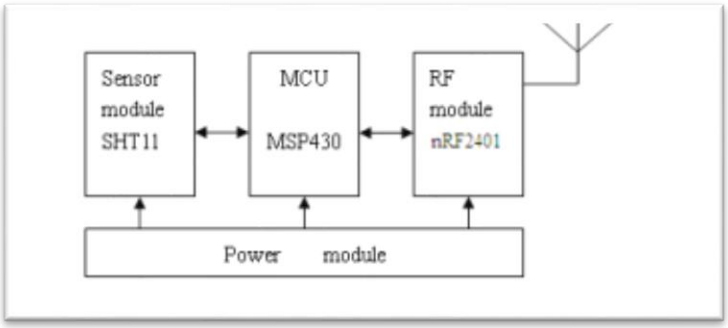

FIGURA 10. NODO SENSOR. EXTRAÍDA DE [38].

Estos nodos sensores no utilizan batería para su consumo de energía, sino que mediante paneles solares, control de carga y un acumulador, utilizan energía solar. La energía almacenada del panel solar es suficiente debido al bajo consumo de este tipo de nodos, por lo que se garantiza que los nodos de la red trabajen sin problemas de batería.

Para la comunicación entre los nodos se utiliza el módulo nRF en modo ShockBurst [40]. ShockBurst es un protocolo de comunicación que provee "buffereo" de paquetes, "acknowledgment" de paquetes y reenvío de paquetes perdidos. Permitiendo a esta red de sensores ser ad hoc y que el ruteo de paquetes lo realice la misma red de nodos sin necesidad de agentes externos.

Este proyecto realiza la instalación de la red de sensores en un terreno y compara las medidas de temperatura obtenida por los sensores con la temperatura real del ambiente, llegando a la comparativa que se muestra en figura 11.

Node Nr.	Real Temperature(℃)	Measured Temperature(℃)	Error(℃)
1	-15.1 0	-15.80	0.70
2	-9.30	-9.00	0.30
3	0	0.30	0.30
4	14.30	14.10	0.20
5	22.50	22.80	0.30
6	38.65	39.15	0.50

FIGURA 11. COMPARATIVA TEMPERATURAS. EXTRAÍDA DE [38].

Se concluye que es posible instalar una red de sensores de bajo consumo de funcionamiento estable, donde se puede realizar el monitoreo del entorno en tiempo real en un contexto agropecuario sin interacción humana directa.

También se concluye que la utilización de redes de sensores provee información fiable, de alta precisión, y sería posible, cambiando el tipo de sensor, monitorear ambientes con distintos fines. Por ejemplo, con un sensor adecuado determinar incendios forestales, lo que conduciría a incrementar la protección del medio ambiente.

Por otro lado, Jiménez, Ravelo y Gómez [41] realizaron un estudio de plagas y enfermedades que afectan a un duraznero, mediante la adquisición de muestras de una red de sensores en Colombia.

Ellos establecen que en primer lugar se debe realizar un muestreo del comportamiento de las plagas, es decir determinar e identificar los parámetros que intervienen en la fenología del cultivo (relación entre factores climáticos y los ciclos de seres vivos, como ser las plagas), para luego realizar el monitoreo de estos factores climáticos y así tenerlos en cuenta a la hora del control y prevención de plagas en cultivos.

La red de sensores utilizada consiste en nodos que miden variables climáticas y también de "trampas electrónicas" utilizadas para la adquisición de información de la población de la mosca de la fruta. Estas trampas consisten en dos sensores ubicados a 5mm uno del otro que se ubican en el orificio de un atrampa McPhail para moscas de la fruta.

En México, también se realizó un monitoreo de la humedad en suelo a través de una red de sensores inalámbrica [42]. El objetivo de este sistema es proporcionar mediante una red de sensores información, para el posterior análisis, de factores del ambiente, como por ejemplo la humedad de suelo, para realizar un consumo eficaz de agua en aquellos territorios donde este bien escasea.

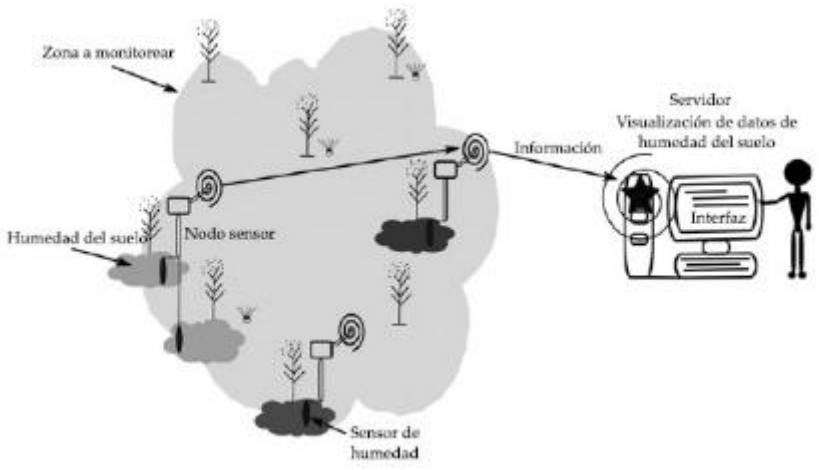

FIGURA 12. COMPARATIVA TEMPERATURAS. EXTRAÍDA DE [42].

Para realizar el primer caso de estudio, la red de sensores es instalada en un invernadero de la ciudad de Torreón donde se cultiva una planta ornamental.

Se utilizan sondas de humedad, para medir la humedad de suelo y se integra el sensor SHT11 para obtener medidas de temperatura, humedad relativa y punto de rocío.

Se monitorea dicha humedad de forma continua y estable las 24 horas del día, con período de muestreo cada 15 minutos durante 6 días.

Este sistema proporciona datos de medidas de humedad de suelo confiable y en tiempo real en condiciones controladas (dentro de un invernadero), dejando abierta la posibilidad de traslado a campo abierto y de ser utilizada en la toma de decisiones para el manejo del agua en los cultivos.

3. Análisis

En esta sección se presenta el análisis realizado para la construcción del sistema final. Se parte del análisis del sistema construido previamente por estudiantes de la facultad de ingeniería en el marco de la asignatura Proyecto de Ingeniería de Software en el año 2014. Esto implica que inicialmente se tomaron gran parte de los requerimientos de dicho análisis y posteriormente se especificaron nuevos requerimientos.

3.1 Red de Sensores

Para la construcción del sistema, se toma como base, una red de sensores previamente construida por integrantes del proyecto Gervasio. Esta red está implementada bajo los estándares del protocolo COAP, donde cada uno de sus motes se comunican entre sí, enviando mutuamente la información. Cuando un sensor genera un dato, este dato sigue una ruta entre los distintos motes hasta llegar al mote principal, denominado base de la red.

A continuación se describen los diferentes conceptos que se ven involucrados en una red de sensores inalámbrica.

- **Red**
  La red es el concepto base que contiene a todo el resto, está formada por un conjunto de motes que se comunican entre sí.

- **Mote**
  Es el dispositivo físico que contiene los sensores, es llamado también nodo y tiene una IPv6 de red que lo define.

- **Sensor**
  Los sensores están ubicados dentro de los motes. Un mote puede contener varios sensores. Este dispositivo se encarga de capturar las medidas correspondientes a una magnitud climática, por lo que cada sensor tiene un tipo definido.

- **Tipo de Sensor**
  Es la clasificación de un sensor y refiere al tipo de medida que el mismo captura, como puede ser humedad, temperatura o humedad suelo.

- **Mote base**
  El mote base es encargado de recibir toda la información generada por diferentes sensores y motes de la red.

- **Tabla ruteo**
  Cuando un mote recibe un paquete de otro mote debe determinar el camino a seguir hasta llegar a la base. Cada mote contiene una tabla de ruteo que indica el siguiente mote en este camino.

- **Tabla de vecinos**
  Cada mote conoce con qué otros motes de la red tiene conectividad. Esta información se almacena en la tabla de vecinos del mote. Además esta tabla contiene el ETX entre un mote y su vecino que es el número esperado de veces que el mote le tiene que enviar un mensaje a su vecino para que lo reciba con éxito. Como consecuencia, podemos definir el RANK de un mote como la suma de los ETX desde el mote hacia la base, siguiendo el camino de los vecinos preferidos.

- **Ciclo de trabajo**
  El ciclo de trabajo es el porcentaje del tiempo en el que un mote está en cada estado del microprocesador. Los diferentes estados son:
  - CPU - microcontrolador encendido y operando. Radio apagada.
  - RX - microcontrolador encendido y operando. Radio recibiendo paquetes o escuchando el canal.
  - TX - microcontrolador encendido y operando. Radio transmitiendo paquetes.
  - LPM - microcontrolador en modo “bajo consumo” y radio apagada.

En la figura 13 se muestra un modelo conceptual con los conceptos mencionados. En el mismo puede observarse la relación entre ellos.

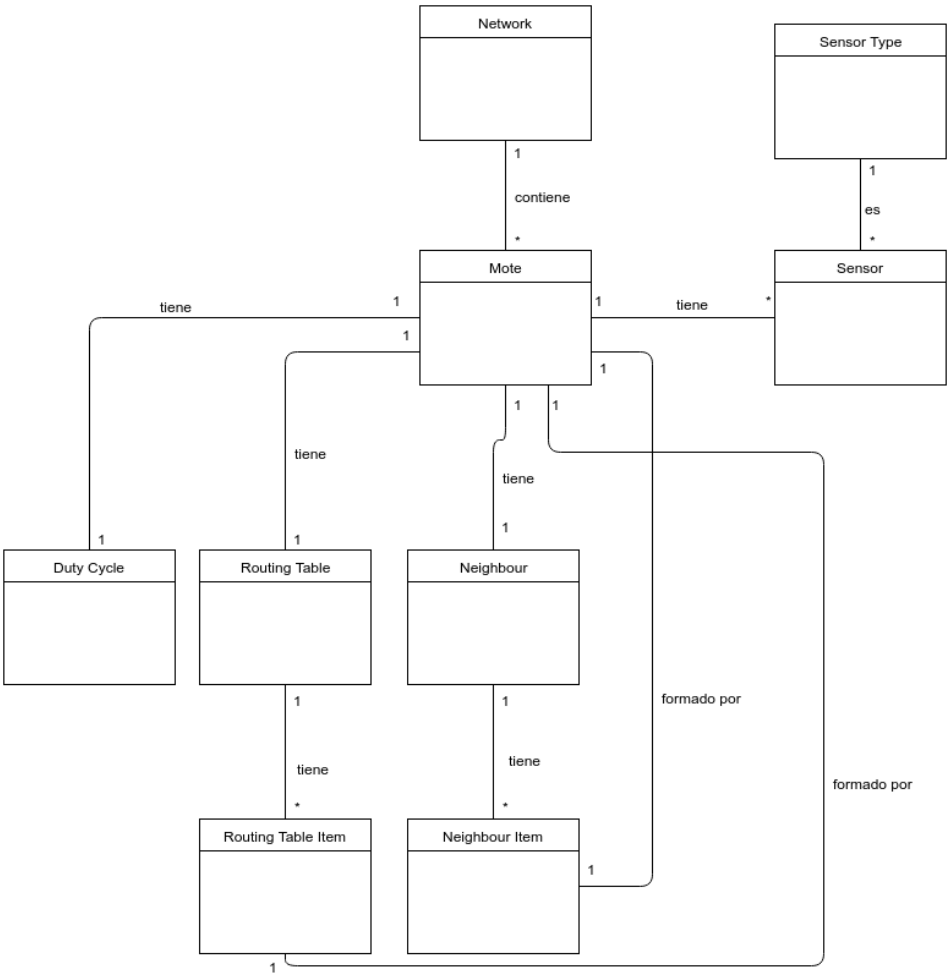

FIGURA 13. MODELO DE DOMINIO RED DE SENSORES

3.2 Requerimientos

En base al objetivo definido en el proyecto se definieron los siguientes requerimientos.

Construir un sistema que permita visualizar la información asociada a una red de sensores. Es decir, dejar disponibles las medidas de los diferentes sensores así como también la información del estado de la red.

El sistema debe tener tres perfiles de usuarios: un perfil de Agrónomo al que se le permita visualizar y realizar acciones sobre las medidas obtenidas por los sensores y un perfil Técnico al que se le permita configurar parámetros sobre dicha red. También se debe tener un perfil de Administrador que tiene acceso a todas las funcionalidades del sistema.

Por último, se debe garantizar que no se pierda la información obtenida por la red de sensores, para lo cual se debe tener en cuenta posibles fallas en la conexión entre la red y la fuente de almacenamiento del sistema.

3.2.1 Requerimientos Funcionales

A continuación se describen los requerimientos funcionales que se mantuvieron del proyecto original de la asignatura Proyecto de Ingeniería de Software.

RF-1 Crear / Modificar / Eliminar Usuarios

El sistema debe permitir a los usuarios administradores, crear usuarios de distintos roles. Para la creación del usuario, se deberá ingresar un nombre de usuario y contraseña, junto con los datos personales del mismo. Por último, se debe permitir seleccionar las redes a las cuales tiene acceso. Una vez creado se podrá visualizar su información, así como modificarla y por último eliminar el usuario registrado.

RF-2 Crear / Eliminar Usuarios Técnico y Agrónomo

El sistema debe permitir a los usuarios técnicos crear diferentes usuarios técnicos y agrónomos. Para la creación del usuario se deberá ingresar un nombre de usuario y contraseña, y otros datos personales. Por último, se debe permitir seleccionar las redes a las cuales el mismo tiene acceso.

RF-3 Visualizar Red

El sistema debe permitir visualizar los datos de una red si el usuario tiene permiso sobre ella. Se debe mostrar un mapa donde se muestre la ubicación de cada mote de la red, junto con su nombre. Adicionalmente, los usuarios administradores y técnicos tendrán acceso a la información de monitoreo de la red, esto significa que pueden visualizar las IPs de los motes, la tabla de ruteo actual y la información de los vecinos de cada mote que aparece en el mapa.

RF-4 Editar Red

Se deberán poder editar ciertos parámetros de la red. Cualquier tipo de usuario podrá modificar el nombre de la red, pero solo los administradores y técnicos podrán modificar la ubicación de los motes en el mapa.

RF-5 Cargar capa

Todos los usuarios podrán cargar capas extra sobre el mapa de ubicación de motes de una red, para posteriormente poder visualizarla. Para esto se requiere que se pueda cargar un archivo con dicha información.

RF-6 Observación de los recursos

Se deberá poder indicar qué recursos se desean observar para cada mote y qué información se desea recibir de cada uno, por ejemplo tabla de ruteo, tabla de vecinos, medidas de un cierto sensor, etc. Esta acción la llevan a cabo los usuarios administrador y técnico.

RF-7 Ver / Eliminar mote

Los usuarios administradores y técnicos pueden visualizar información de un mote en particular, esto incluye la información de sus sensores, la información de batería o el RANK.

RF-8 Generar gráfico Medidas

Todos los usuarios pueden visualizar gráficas de tiempo de las medidas de un sensor, seleccionando el rango de fecha, los motes a graficar y por último el tipo de sensor que se desea.

RF-9 Generar reporte Medidas

Todos los usuarios pueden generar reporte de las medidas de un sensor, seleccionando el rango de fecha, los motes y el tipo de sensor que se desea graficar. El reporte se genera en un archivo Excel.

RF-10 Alta / Modificación / Baja Tipo de Sensor

El sistema debe permitir a los usuarios administradores y técnicos, agregar nuevos tipo de sensores. Para esto se deberá agregar una descripción (para ser mostrada), el tipo y su URI CoAP asociada. Luego, se podrán modificar y eliminar estos si se desea.

3.2.1.1 Requerimientos Funcionales Agregados

A continuación se explicitan los requerimientos funcionales que fueron agregados a los requerimientos originales del Proyecto de Ingeniería de Software.

RF-11 Ver / Eliminar capa

El sistema debe permitir a cada usuario consultar las redes sobre las que tiene permiso, así como también visualizar las capas que están cargadas en su mapa y además que pueda eliminarlas.

RF-12 Configurar Parámetros de Muestreo

El sistema debe permitir que los usuarios administradores y técnicos puedan configurar los parámetros de muestreo de las medidas de los motes. Para cada mote se podrá configurar el período de muestreo (el tiempo que transcurre entre una medida y la siguiente) y el período de observación (cada cuánto tiempo se envían datos a la base de la red).

RF-13 Reporte Histórico de Vecinos

El sistema debe permitir que los usuarios administrador y técnico puedan generar reportes históricos de la tabla de vecinos de cierto mote. Dicho reporte deberá mostrar los diferentes vecinos de un mote incluyendo también el ETX correspondiente. Para el reporte se debe seleccionar un rango de fecha y generar un archivo Excel.

RF-14 Reporte Histórico de Tabla de Ruteo

El sistema debe permitir que los usuarios administrador y técnico puedan generar reportes históricos de la tabla de ruteo de cierto mote, seleccionando un rango de fecha. Este reporte debe generarse en un archivo PDF.

RF-15 Ver Ciclo de Trabajo

El sistema debe permitir que los usuarios administrador y técnico puedan visualizar los ciclos de trabajo de cada mote.

RF-16 Alta de alarma

El sistema debe permitir que todos los usuarios del sistema puedan crear alarmas sobre los sensores de los diferentes motes de una red. Para ello se debe indicar el tipo de sensor en el que se configura la alarma, el rango de valores que deben chequearse, el tipo de notificación (mail/SMS), los usuarios que se desean alertar y los motes sobre los cuales se aplica la alarma.

Se necesitan tres tipos de alarmas:

1. *Simple:* para un único mote dado, se deberá indicar un tipo de sensor y un umbral de medidas. Se envía notificación si se cumple que la medida recibida de la red de sensores, está fuera del umbral de medidas definido.

2. *Contador:* para un único mote dado, se deberá seleccionar un tipo de sensor, un umbral de medidas, un contador y un período de tiempo. Se envía notificación si dentro del período de tiempo indicado, se reciben una cantidad de medidas igual al contador seleccionado que se encuentran fuera del umbral definido.

3. *Mixta:* para un conjunto de motes dados, se deberá seleccionar un tipo de sensor, un umbral de medidas, un contador y un período de tiempo. Se envía notificación si dentro del período de tiempo indicado, se reciben una cantidad de medidas igual al contador seleccionado, que se encuentran fuera del umbral definido para cada uno de los motes indicados.

RF-17 Editar / Eliminar Alarma

El sistema debe permitir que todos los usuarios puedan editar alarmas previamente configuradas y que permita cambiar los valores del umbral, el nombre, el tipo de notificación y los usuarios a notificar. El tipo de alarma no se puede editar, pero sí se puede eliminar alarmas.

RF-18 Alta alarmas desde archivo

El sistema debe permitir que sea posible cargar alarmas de manera masiva en el sistema, para lo que se debe ingresar un archivo de Excel conteniendo la configuración para varias alarmas y que se carguen en el sistema.

RF-19 Pedidos CoAP

El sistema debe permitir que se disponga de un mecanismo para enviar mensajes CoAP a los sensores.

3.2.2 Requerimientos No Funcionales

A continuación se detallan los requerimientos no funcionales relevados en este proyecto. Estos no fueron contemplados en el relevamiento realizado por los proyectos de la asignatura Proyecto de Ingeniería de Software.

RNF-1 Componente recepción datos en procesador Embebido

El componente de software que recibe los datos provenientes de la red de sensores, debe poder ejecutarse en un procesador embebido ya que estará ubicado cerca de la red de sensores.

RNF-2 Componente recepción datos tolerante a fallos

El componente de software que recibe los datos desde la red de sensores, debe ser tolerante a fallos de conexión con el componente que almacena y procesa los datos. Esto implica, que debe contener un mecanismo de almacenamiento temporal de los mensajes recibidos desde la base de la red.

RNF-3 Soportar comunicación de Internet por varios medios

El componente de software que recibe los datos de la red de sensores, debe soportar diferente medios de conexión a internet con el componente que almacena los datos: 3G, WiFi o red cableada.

RNF-4 Almacenamiento de medidas por tiempo indeterminado

Las medidas generadas por la red de sensores se deben almacenar por tiempo indefinido en el sistema.

RNF-5 Manejar Eficientemente Conexión

El componente de recepción de los datos debe estar conectado con una fuente de energía (batería) en el terreno donde se encuentre la red de sensores. Para el uso eficiente de la batería se deberá optimizar el tiempo de conexión a internet.

RNF-6 Actualizar tiempo en motes

El sistema debe actualizar periódicamente la hora de los motes.

RNF-7 Atender múltiples Redes

El sistema debe ser capaz de visualizar diferentes redes independientes, montadas en lugares físicos distintos.

3.4 Vista Casos de Uso

A continuación se detallan los actores del sistema junto con los principales casos de uso.

3.4.1 Actores del sistema

Agrónomo

Este actor tiene acceso a la visualización de las redes, monitoreo de medidas de los sensores, creación de alarmas sobre la red. Este actor se limita a visualizar las medidas de los sensores y sus funcionalidades relacionadas: gráficas, informes y alarmas.

Técnico

Es el que administra la red de sensores, tiene acceso a las mismas funcionalidades que el Agrónomo y además puede acceder a opciones de monitoreo y configuración de la red, crear nuevos usuarios y nuevos tipos de sensores.

Administrador

El actor Administrador tiene acceso a todas las funcionalidades definidas para el Técnico y el Agrónomo.

Tiempo

Este actor representa todos los procesos que son automáticos y se ejecutan de manera periódica en el sistema, por ejemplo la sincronización de los motes de la red con el sistema o el envío pendiente de datos desde el procesador embebido hacia el servidor central.

Red de sensores

La principal acción que realiza este actor es el envío de nuevos datos generados por sus motes.

3.4.2 Casos de uso críticos

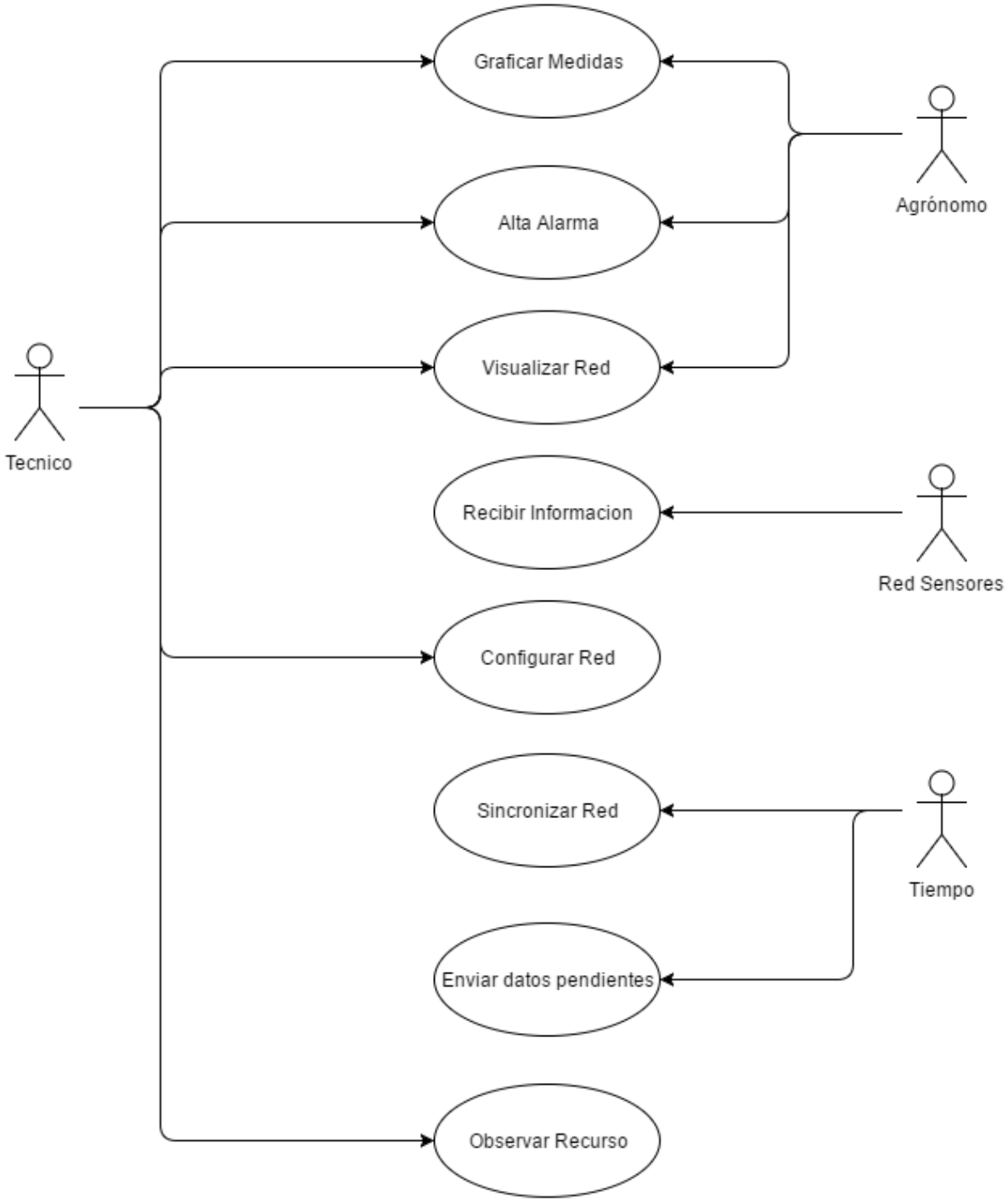

FIGURA 14. CASOS DE USO CRÍTICOS

La figura 14 muestra una vista de los casos de uso críticos del sistema. Se tomaron como casos de uso crítico a las funcionalidades que presentan mayor riesgo técnico, así como las que el cliente les asignó mayor prioridad.

Los casos de uso que realizan la conexión con la red de sensores: recibir información, configurar red, sincronizar red, enviar datos pendientes y observar recursos fueron

elegidos por su complejidad técnica. Además, visualizar red, alta alarma y graficar medidas fueron seleccionados, ya que estas funcionalidades son el núcleo del sistema, por lo que también son consideradas críticas.

3.5 Arquitectura propuesta

Teniendo en cuenta los casos de uso y los requerimientos no funcionales ya planteados, en esta sección se realiza una introducción sobre la arquitectura propuesta para resolver el problema planteado.

En la figura 15 se puede visualizar de forma gráfica la arquitectura propuesta.

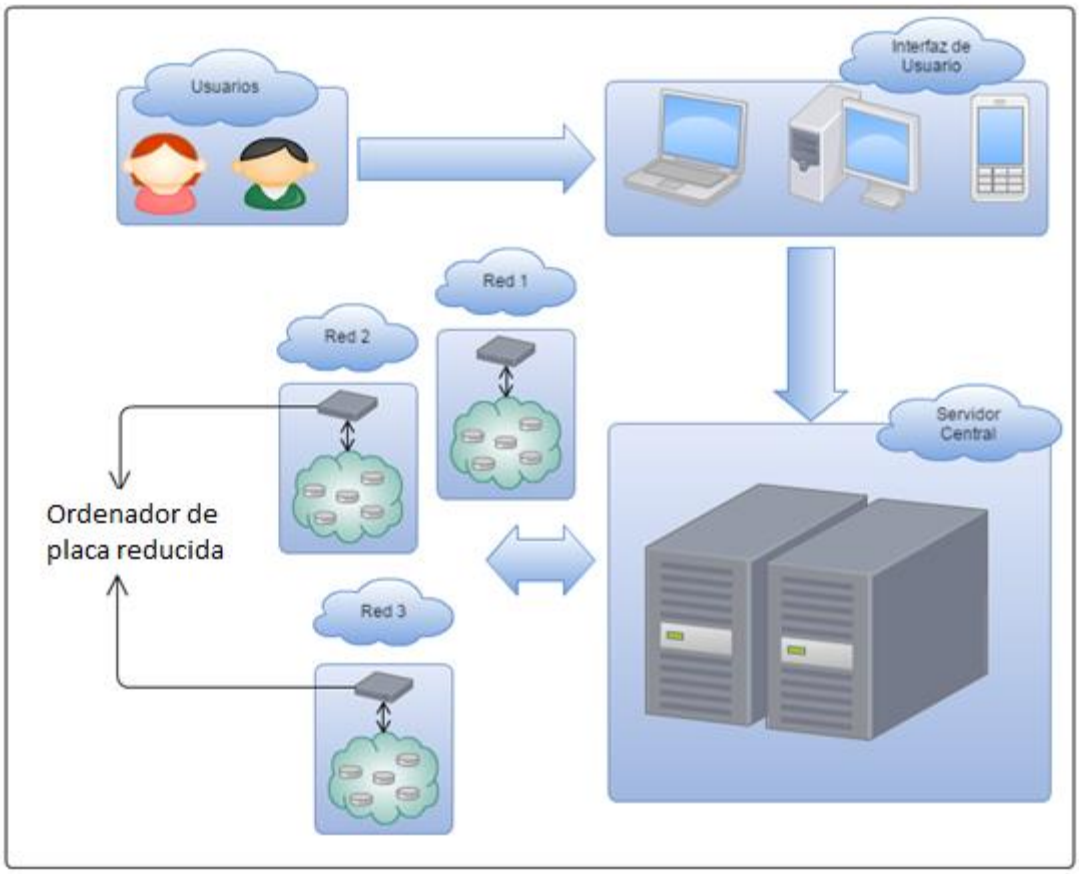

FIGURA 15. ARQUITECTURA PROPUESTA

Para solucionar una de las principales restricciones no funcionales planteadas, se establece que el componente de comunicación con la red de sensores pueda ejecutarse en un ordenador de placa reducida, ubicado físicamente cerca de la red de sensores. El sistema requiere por lo menos dos módulos independientes; el primero, encargado de la interacción con los motes de la red y el segundo, el módulo principal encargado de almacenar la información recabada y de ejecutar los procesos correspondientes.

Por otro lado, se acuerda construir una interfaz gráfica para visualizar los datos. Con el fin de poder desacoplar la lógica del negocio de la lógica de la presentación, se agrega un tercer módulo que se encargue de la interfaz web solicitada.

La figura 16 ilustra los tres módulos que se propone implementar.

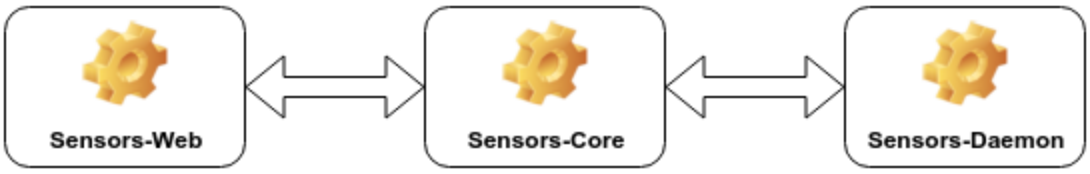

FIGURA 16. MÓDULOS PROPUESTOS

3.5.1 Módulos del sistema

3.5.1.1 Sensors-Daemon

Este módulo tiene la responsabilidad de establecer la comunicación con la red de sensores instalada. Es denominado "demonio" ya que principalmente ejecuta procesos de tipo batch, sincronizando la red con los datos del sistema. Así mismo, implementa un mecanismo de almacenamiento temporal de los datos recibidos en caso que estos no puedan ser enviados al módulo Sensors-Core.

Su despliegue está diseñado para realizarse en un ordenador de placa reducida, optimizando el uso de sus recursos. Dicho procesador tiene acceso a internet mediante un modem 3G y se desea minimizar el uso de la conexión a internet, debido a su alto consumo de batería. Como consecuencia, el componente en cuestión implementa un mecanismo de conexión a internet periódico, es decir, cada cierto tiempo establece conexión y dentro de esta ventana de tiempo se envían los datos pendientes.

Las principales tareas que realiza este módulo son las siguientes:

- Sincronización de la información de los motes de la red con el sistema.
- Recepción de datos provenientes desde los motes.
- Envío de información hacia el módulo Sensors-Core.
- Almacenamiento temporal de información.
- Manejo de la conexión 3G de manera eficiente.

Por último, en la figura 15, se puede observar la coexistencia de múltiples instancias de dicho modulo, ya que cada uno atiende y sincroniza a una red diferente.

3.5.1.2 Sensors-Core

El módulo Sensors-Core maneja toda la información del sistema y es el responsable de ejecutar los procesos de negocio sobre los datos. También se encarga de recibir la información enviada por Sensors-Daemon y almacenarla.

Uno de los principales procesos que lleva a cabo es el chequeo de ocurrencias de las alarmas. Cuando una nueva medida es recibida desde el Sensors-Daemon se chequean las alarmas configuradas y se notifica a quien corresponda en caso de cumplirse las condiciones de esta.

Por otro lado, Sensors-Core brinda una capa de servicios de forma tal que otros módulos puedan consultar por los datos persistidos.

Al ser el gestor de los datos, es también el encargado de construir los reportes en PDF y Excel con la información almacenada.

3.5.1.3 Sensors-Web

El tercer módulo de la arquitectura es denominado Sensors-Web. Tiene la responsabilidad de implementar la interfaz gráfica web para visualizar de forma amigable la información del sistema. Consume principalmente los servicios brindados por Sensors-Core y adapta el formato de los datos para ser mostrados.

Este módulo contiene todo el contenido estático para ser visualizados en el navegador web. Es en su totalidad un componente de front-end.

4. Implementación de la Solución

En el presente capítulo se detalla la solución propuesta para resolver el problema planteado. En primer lugar se describe la arquitectura haciendo foco en los módulos detallados en el capítulo anterior y cómo se comunican entre sí. En segundo lugar, se muestra el despliegue propuesto para cada uno de ellos y por último, se presentan algunas características particulares de la implementación del sistema.

4.5 Diseño de la Arquitectura

En esta sección se describe la implementación y diseño de la arquitectura propuesta en el capítulo 3. Para esto se detallan los diferentes mecanismos de comunicación entre los módulos y como los mismos pueden ser desplegados.

Para este proyecto se optó por una arquitectura basada en capas. Las capas son las siguientes:

- **Capa de comunicación**
  Esta capa es la encargada de la comunicación con la red de sensores inalámbricos.

- **Capa de lógica**
  Es la capa que contiene la lógica de negocio del sistema y tiene acceso a sus datos.

- **Capa de presentación**
  Esta capa implementa la interfaz de usuario del sistema.

Cada una de estas capas se ve reflejada en un módulo del sistema construido.

Uno de los principales motivos por lo que se definió esta arquitectura, se debe a la necesidad de que el módulo que interactúa con la red de sensores debe estar cercano a la propia red, y físicamente ubicado en la misma zona. Esto obligó a definir una arquitectura distribuida, donde se propone colocar el módulo de lógica de negocio por un lado, y el módulo de comunicación desplegado cerca de la red de sensores.

Por otro lado, se decidió desacoplar el componente web de la capa lógica para poder mantener un despliegue independiente. Esto brinda la posibilidad de migrar rápidamente la capa de presentación del sistema de ser necesario, puesto que la capa lógica brinda todos los servicios necesarios.

4.5.1 Diagrama de Arquitectura

Se presenta a continuación la propuesta de implementación de la arquitectura mencionada.

Cada uno de los módulos se compone de uno o varios componentes. En la figura 17 se muestra el mapa completo de componentes de la arquitectura.

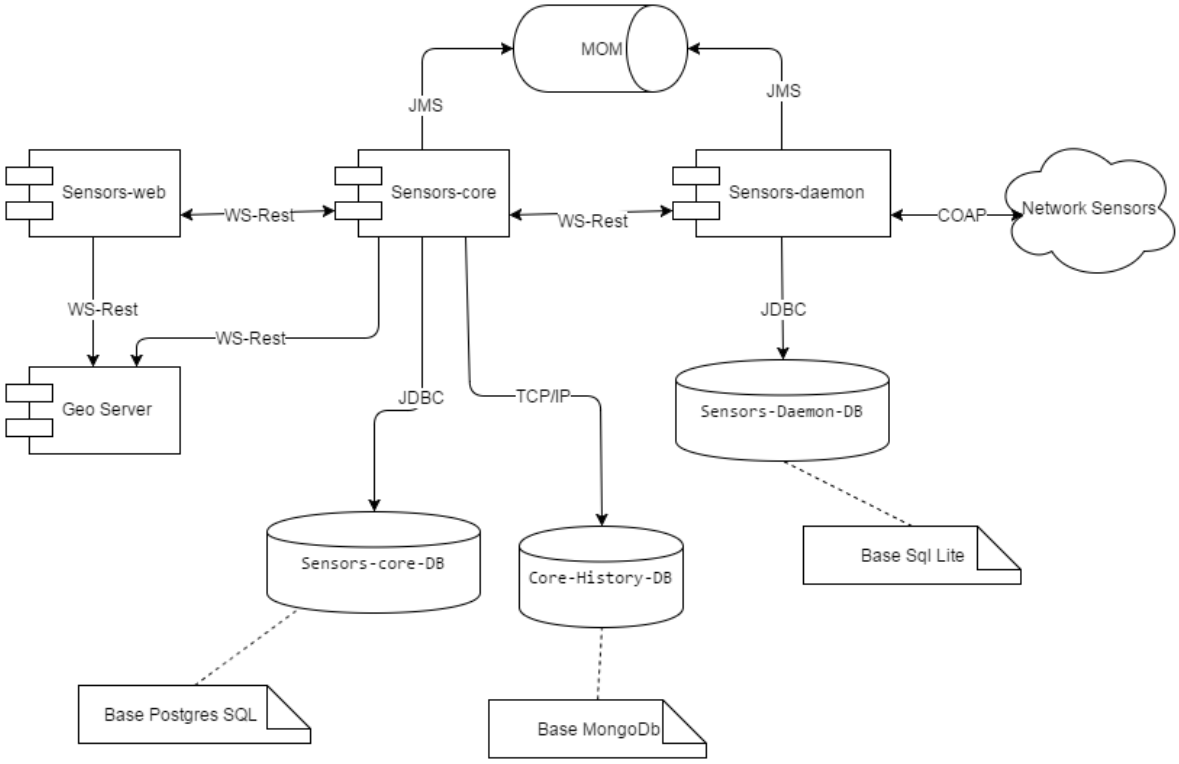

FIGURA 17. COMPONENTES DE ARQUITECTURA PROPUESTA

4.5.1.1 Sensors-Daemon

El componente *Sensors-Daemon* se ejecuta sobre un ordenador de placa reducida. Esto implica que el componente no debe consumir una cantidad importante de recursos ya que el ordenador no los posee. Por este motivo el componente implementado está pensado para ser un servidor liviano y de poco procesamiento.

Su tarea principal es sincronizar el sistema con la red de sensores inalámbricos. Para esto intercambia mensajes CoAP con la base de la red para obtener periódicamente el estado de la misma. Esto implica conocer los motes que la componen y con esto la información de los sensores que contiene cada mote.

Además de sincronizar la red recibe los diferentes datos de cada uno de los sensores. Esto lo realiza mediante el mecanismo de observación de recursos brindado por el

protocolo CoAP. Esta información incluye tanto las medidas de cada uno de los sensores como también la tabla de ruteo o vecinos de los motes.

Cuando un dato nuevo es recibido desde la red, el componente debe informar al Sensors-Core. Para esto, Sensors-Daemon implementa dos estrategias distintas dependiendo del tipo de mensaje que se haya recibido desde la red, y decide si mandar la información de forma sincrónica o asincrónica.

- **Envío a través de servicios**
  Cuando un mensaje debe ser enviado de forma sincrónica al Sensors-Core, se envía a través de un Web Servicie REST. Esto garantiza que el mensaje se haya recibido correctamente. Este tipo de estrategia es la utilizada por ejemplo cuando un nuevo mote ha sido reportado por la red de sensores.

- **Envío a través de MOM**
  En el caso de que los mensajes deban ser enviados de forma asincrónica, se utiliza un MOM para su envío. Esto garantiza que el mensaje será enviado al Sensors-Core y que este efectivamente lo recibirá.

Como consecuencia del requerimiento no funcional RNF-5, el ordenador de placa reducida en el cual se ejecuta el *Sensors-Daemon*, estará conectado a internet de manera periódica a través de una red 3G. Esto implica que se deberán almacenar los datos recibidos de la red de sensores de manera temporal, hasta lograr conectividad.

Por este motivo se decidió agregar a este componente, una base de datos relacional y liviana que pueda realizar esta tarea. La misma, denominada *Sensors-Daemon-DB* emplea un servidor de base de datos SQL liviano que almacena su información de manera local.

Un último aspecto a destacar es la existencia de múltiples instancias de este componente. Dado que pueden existir diferentes redes de sensores independientes, para cada una de ellas tendremos un componente Sensors-Daemon dedicado, que se sincroniza con la red y se encarga de la comunicación exclusiva con ella.

4.5.1.2 Sensors-Core

El componente *Sensors-Core* es considerado el servidor central, ya que contiene toda la información y procesos de negocio del sistema. Implementa las principales funcionalidades brindadas y es el responsable de almacenar los datos del negocio.

Este componente brinda una API REST que permite consultar, modificar y eliminar datos del sistema, implementando de esta manera la mayor parte de las funcionalidades requeridas.

Por otra parte, este componente implementa servicios REST dedicados a recibir datos desde el componente *Sensors-Daemon*. Estos servicios son los que se utilizan cuando el *Sensors-Daemon* debe enviar mensajes en forma sincrónica.

Para el almacenamiento de los datos, se manejan dos bases de datos diferentes. La primera, *Sensors-Core-DB,* es una base de datos relacional que contiene información del estado actual de cada una de las redes, por ejemplo de los motes que la forman, la tabla de vecinos, la tabla de ruteo, los sensores de cada mote y las alarmas configuradas. Por otro lado, se encuentra la base de datos *Core-History-DB*. Como indica su nombre, en ella se almacenan datos históricos de las redes: tablas de ruteo, tablas de vecinos y las propias medidas de cada uno de los sensores. Se trata de una base de datos NoSQL orientada a documentos, lo que implica que no existe un modelo de datos relacional. Se decidió por esta solución ya que se espera que la base de datos va a almacenar gran cantidad de información (al tratarse del almacenamiento de datos históricos) y de esta forma se disminuye considerablemente el tiempo de lectura y escritura de los mismos.

4.5.1.1 Sensors-Web

Este componente implementa la interfaz de usuario del sistema. Está formado por contenido estático para ser mostrado en el navegador web y es en su totalidad un componente front-end.

Además contiene la lógica necesaria para obtener los datos desde el *Sensors-Core* y aplicarles las transformaciones necesarias para que los mismos puedan ser mostrados en las diferentes páginas web.

Otra funcionalidad implementada en este componente es el manejo de permisos sobre los datos que pueden visualizarse en las diferentes vistas. Según el rol del usuario que se encuentre autenticado en el sistema, se muestra u oculta diferente información.

4.5.1.2 Geo Server

Geo server es un servidor de mapas [43]. En la arquitectura propuesta, dicho componente almacena las capas cargadas por los usuarios de los mapas de la red.

Cuando un usuario desea cargar una nueva capa en el mapa de una red, la capa cargada es enviada al servidor y almacenada allí.

4.5.2 Despliegue de la Arquitectura

En esta sección se presenta el diagrama de despliegue físico utilizado para las pruebas del sistema y se muestran además otras alternativas.

4.5.2.1 Despliegue implementado

La figura 18 muestra el diagrama de despliegue alcanzado en las pruebas del sistema.

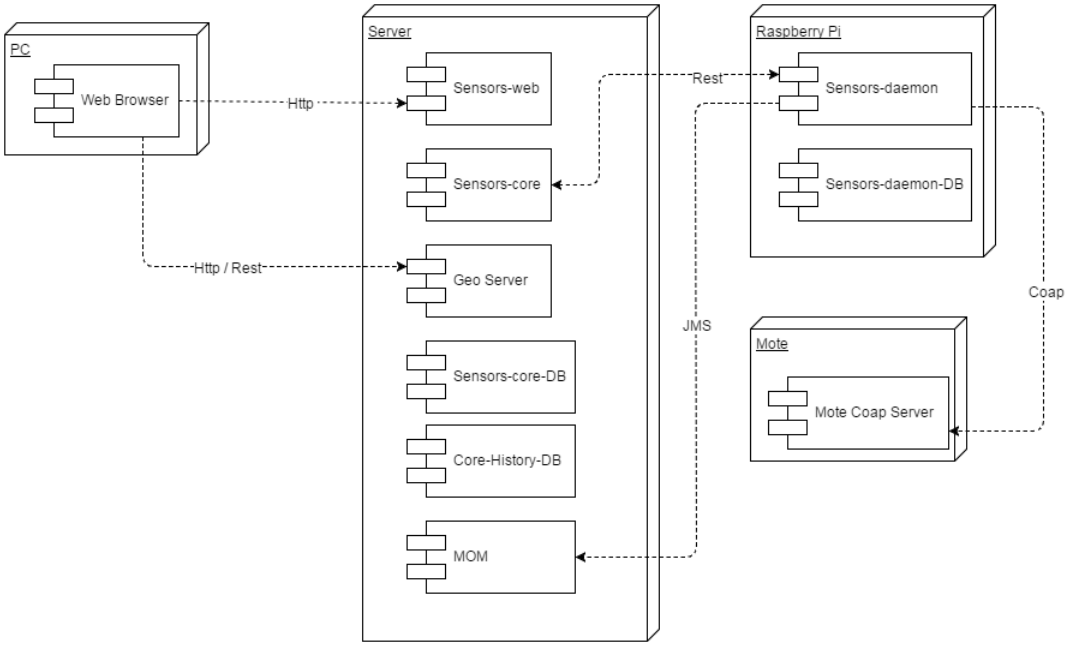

FIGURA 18. DIAGRAMA DESPLIEGUE IMPLEMENTADO

Por un lado se encuentra el cliente que accede desde su navegador web. Luego, desplegado en un servidor se tienen 6 componentes instalados: Sensors-Web, Sensors-Core, Geo Server, el MOM y las dos bases de datos utilizadas por el componente Core: Sensors-Core-DB y Core-History-DB.

Por último, se encuentra el componente Sensors-Daemon (con su respectiva base de datos) desplegado en un dispositivo Raspberry Pi y cada uno de los motes de la red.

La decisión de este despliegue fue tomada en base a los recursos que el equipo de Gervasio tiene disponibles. En este escenario se dispone de una máquina servidor y de un Raspberry Pi para el despliegue. Por este motivo se decidió desplegar el resto de los componentes en un mismo servidor físico.

A pesar de esto, se ha diseñado el sistema de manera tal que se puedan distribuir todos los restantes componentes en servidores físicos diferentes. Esto lograría una mejor performance ya que no se estarían compartiendo recursos de memoria y CPU entre los diferentes componentes.

4.5.2.1 Despliegues alternativos

Como se menciona en la sección anterior, las pruebas fueron realizadas sobre el despliegue físico mostrado en la figura 18, cabe destacar que el diseño del sistema permite desacoplar aún más los diferentes componentes. Esto facilita distribuir los componentes optimizando el uso de recursos de un servidor y dedicando el mismo a tareas más específicas.

A continuación se mencionan dos variantes al despliegue original determinando dos niveles de distribución.

DESPLIEGUE ALTERNATIVO 1

En la figura 19 se ilustra el primer nivel de distribución posible para el despliegue del sistema.

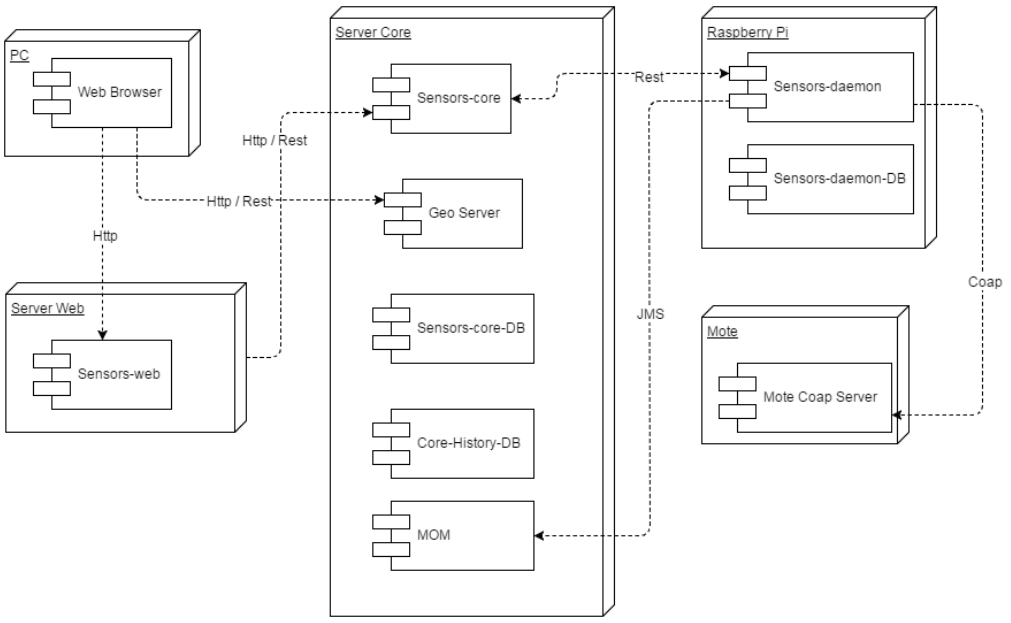

FIGURA 19. DIAGRAMA ALTERNATIVA DESPLIEGUE 1

En este escenario se separa el Sensors-Web del Sensors- Core, de forma de separar lo relativo a la interfaz de usuario.

DESPLIEGUE ALTERNATIVO 2

En la figura 20 se ilustra el segundo nivel de distribución posible para el despliegue del sistema.

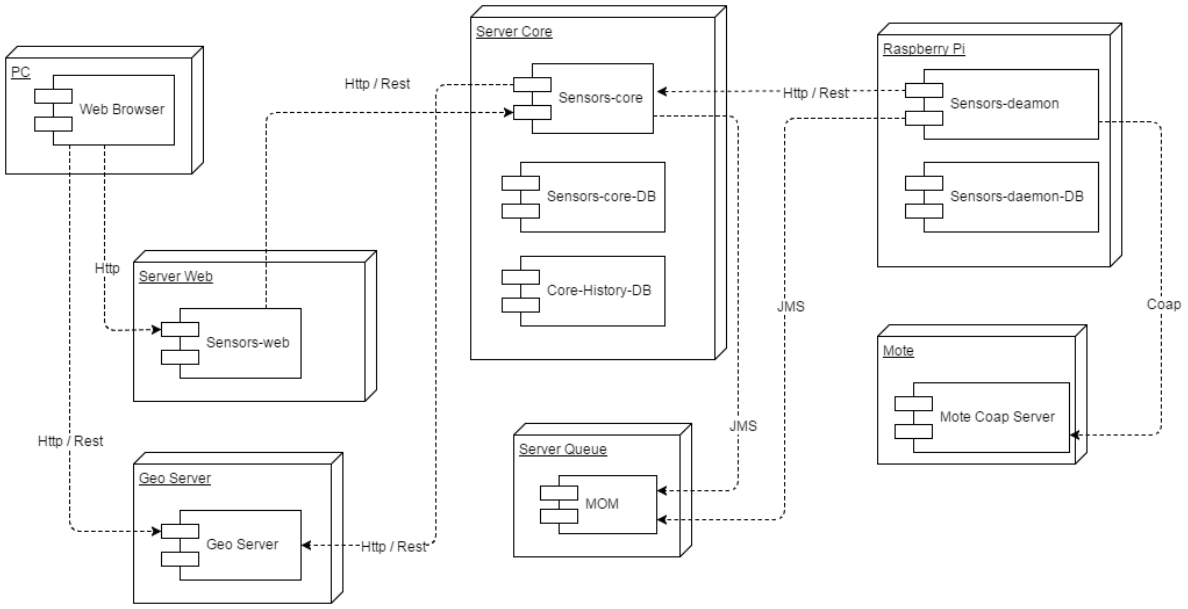

FIGURA 20. DIAGRAMA ALTERNATIVA DESPLIEGUE 2

En este segundo escenario se distribuye aún más el sistema, lo que constituye el escenario ideal de despliegue. Como muestra la figura 20, se desacopla el MOM y el servidor de mapas en servidores diferentes.

Para lograr la distribución total de los componentes sería necesario separar las base de datos Sensors-Core-DB y Core-History-DB en servidores diferentes.

4.2 Implementación

En esta sección se presentan los distintos aspectos de la implementación de la solución propuesta. Detallando las tecnologías y herramientas utilizadas, para luego destacar características propias de la implementación realizada.

4.2.1 Herramientas Utilizadas

En la figura 21 se puede observar el diagrama de componentes del sistema junto con las tecnologías más importantes utilizadas en cada uno.

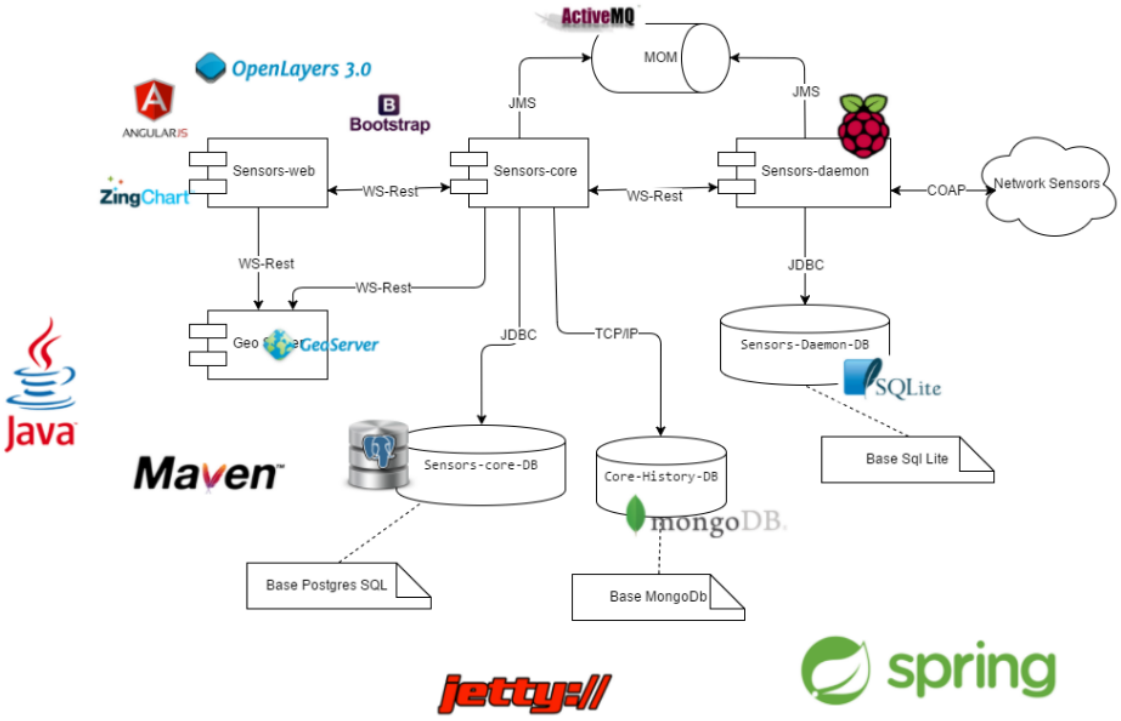

FIGURA 21. DIAGRAMA TECNOLOGÍAS EN COMPONENTES

Todos los componentes del sistema: Sensors-Daemon, Sensors-Core y Sensors-Web se desarrollaron en Java, utilizando Maven y el framework Spring, y se ejecutan sobre el servidor Jetty.

En el componente Sensors-Daemon se utiliza la base de datos SQLite debido a que el mismo se ejecuta sobre un Raspberry Pi de escasa capacidad y necesita almacenar datos de temporales. Para la comunicación con la red de sensores se utiliza Californium, que brinda una API para el uso del protocolo CoAP, mientras que para la comunicación con el Sensors-Core se implementa el estándar JMS, utilizando el broker ActiveMQ, y se utilizan también web services REST.

El componente Sensors-Core almacena las medidas de los sensores y los valores históricos, de tablas de ruteo y vecinos de los motes de la red, en una base de datos no relacional MongoDB. Además utiliza otra base de datos PostgreSQL para el almacenamiento de el resto de los datos del sistema: usuarios, sensores, alarmas, etc.

En la implementación del componente Sensors-Web se utiliza el framework AngularJS, junto con Bootstrap permitiendo a la interfaz web ser responsive. La visualización de

mapas se realiza a través de Openlayers 3 y para la visualización de gráficas es utilizado Zing Chart.

El almacenamiento de las capas de los mapas, se realiza mediante el servidor Geo Server y la comunicación con este se implementa a través de su REST API.

Para más detalle de estas tecnologías ver Apéndice 1. Descripción de las herramientas utilizadas.

4.2.3 Detalles de implementación

En esta sección se describen algunas particularidades de la implementación de las funcionalidades que se consideran de mayor complejidad y riesgo técnico.

4.2.3.1 Multi red

Para el soporte de múltiples redes en el sistema, cuando un Sensors-Daemon es instalado e inicializado, se le indica la IP de la base de la red de sensores que atiende para poder realizar la sincronización con ella.

Sensors-Daemon debe enviar la IP de la base de la red que atiende junto con su propia IP al Sensors-Core. La primera es enviada ya que es el identificador de la red y la segunda IP es utilizada para realizar las invocaciones a los servicios expuestos por Sensors-Daemon.

4.2.3.2 Formato de URIs y mensajes CoAP

Uno de los principales desafíos en el uso del protocolo CoAP en este proyecto, fue definir el formato de los mensajes que se intercambian entre Sensors-Daemon y la red de sensores. Esta definición fue realizada en conjunto con los integrantes del proyecto Gervasio debido a que fueron ellos quienes realizaron el desarrollo correspondiente al firmware de los motes. Además, se definieron lineamientos generales para la construcción de las URIs donde serían publicados los recursos de los motes.

Para la mayoría de los mensajes intercambiados se acordó el uso del formato JSON [61] mientras que para los mensajes restantes el formato utilizado es texto plano ya que son contenidos más sencillos.

Los recursos pueden ser categorizados dentro de tres grupos:

- **Recurso de Mote**
  Los recursos de Mote son las magnitudes y configuraciones particulares de un mote. En esta categoría se puede incluir la hora Unix del mote o la información de la batería. Los recursos de esta categoría comienzan con el prefijo de URI */node***.**

- **Recurso de Red**
  Estos recursos describen características de las relaciones de un mote con los demás. En esta categoría se incluyen por ejemplo: la tabla de Ruteo y la tabla de vecinos. Estos recursos tienen como prefijo en la URI */network*.

- **Recurso de Sensor**
  Por último se encuentran los recursos de sensor. Estos son las características de los sensores, como las medidas, información del sensor o período de muestro y período de observación. Los recursos de sensor tienen como prefijo de URI /node/sensor.

Otro aspecto importante a destacar dentro de esta categoría, es que cada sensor tiene asociado un nombre de URI único, por ejemplo la temperatura es temperature, que no puede repetirse dentro del mismo mote.

A continuación, en la tabla 1 se muestran los diferentes recursos acordados. Para cada elemento de la tabla se indica el significado del recurso, la URI acordada y su formato de mensaje.

Descripción	*Método CoAP*	*URI*	*Cuerpo del Mensaje*
Configuración de la hora Unix del mote	GET / PUT	/node?type=time	54654545
Datos de consumo de energía	GET	/node?type=charge	1200
Datos de los tiempos (acumulados) que el nodo permanece en cada estado	GET	/node?type=duty-cycle	{ "tx": "150", "rx": "120", "cpu": "142", "lpm": "110" }
Medida de un sensor determinado	GET / OBSERVE	/node/sensors/{sensorType}?type=measure	{ "value": "91.5", "time": "12345678901", "raw": "515" }
Información sobre un Sensor determinado	GET	/node/sensors/{sensorType}?type=info	{ "model": "SHT25", "brand": "Sensirion", "otherInfo": "Más info", "unit": "%" }
Período de muestreo de medidas	GET / PUT	/node/sensors/{sensorType}?type=sample_period	52
Período de envío de medidas a la base de la red	GET / PUT	/node/sensors/{sensorType}?type=observe_period	52
Tabla de vecinos de un nodo determinado	GET / OBSERVE	/network/neighbours	[{ "IP": "aaaa::10.1.1.2", "ETX": "4", "rssi": "-40" }, { "IP": "aaaa::10.1.1.4", "ETX": "1", "rssi": "-15" }]
Estado de la red / Estado de un nodo	GET / OBSERVE	/network/status	{ "rank": "534" } ["aaaa::10.1.1.1", "aaaa::10.1.1.2", "aaaa::10.1.1.3", "aaaa::10.1.1.4", "aaaa::10.1.1.5"]
Tabla de ruteo de un nodo determinado	GET / OBSERVE	/network/routetable	[{ "dest": "aaaa::10.1.1.1", "via": "aaaa::10.1.1.4" }, { "dest": "aaaa::10.1.1.2", "via": "aaaa::10.1.1.5" }]
Recurso libre	GET / OBSERVE	/network/stats	"informacion en formato texto"

TABLA 1. URIS Y FORMATOS DE MENSAJES COAP

Respecto a la tabla anterior cabe realizar las siguientes observaciones:

- El recurso /network/status tiene dos posibles formatos de respuesta:
  - Si el mote corresponde a la base de la red, la respuesta es la lista de IPs correspondiente a los motes que conforman la red.
  - Si el mote no es la base, la respuesta es la información del estado del mote.

4.2.3.3 API REST para CoAP

Se dispone de una API REST sobre CoAP implementada en Sensors-Daemon. Esta API implementa la llamada a un recurso CoAP a través de un pedido REST y facilita la tarea de acceder a los recursos de los motes.

La imagen 30 ilustra la arquitectura de esta API.

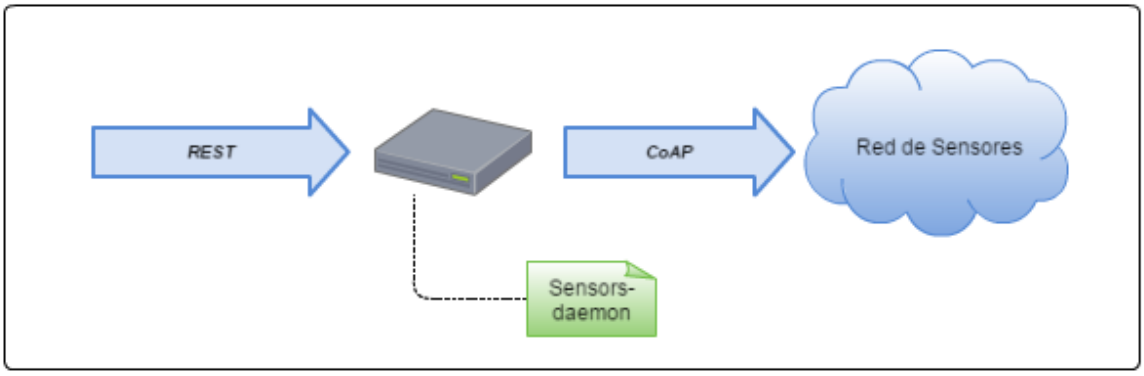

FIGURA 30. ILUSTRACIÓN API PARA COAP

En la tabla 2 se puede observar la API implementada y los servicios que esta ofrece.

Descripción	*Método HTTP*	*URI*
Ejecuta el método CoAP OBSERVE	GET	/resources/observe?uri={uri_coap}
Ejecuta el método CoAP STOP	GET	/resources/stop?uri={uri_coap}
Ejecuta el método CoAP PUT	PUT	/resources/put?uri={uri_coap}
Ejecuta el método CoAP POST	POST	/resources/post?uri={uri_coap}
Ejecuta el método CoAP GET	GET	/resources/get?uri={uri_coap}

TABLA 2. API COAP

4.2.3.4 Almacenamiento local en Sensors-Daemon

En este punto se explican los procedimientos y estructuras utilizadas para el almacenamiento local en Sensors-Daemon de los mensajes provenientes desde la red de sensores.

Cuando un mensaje proveniente de la base de la red es recibido por Sensors-Daemon, este lo categoriza según el recurso del que se trate y lo almacena de manera local en una base de datos SQLite. La misma contiene dos estructuras, la primera de ellas se utiliza para almacenar los mensajes recibidos de los sensores de la red y la segunda destinada a mensajes que contengan información del estado de los motes.

En la figura 31 puede verse la representación de las estructuras mencionadas.

```
public class SensorMessage {

	private String networkId;
	private String moteIp;
	private String message;
	private SensorMessageCategory messageCategory;
	private String sensorType;
	private long date;
	private boolean sendToQueue;

	public String getNetworkId() {
		return networkId;
	}

	public void setNetworkId(String networkId) {
		this.networkId = networkId;
	}

	public String getMoteIp() {
		return moteIp;
	}
```

```
public class NetworkMessage {

	private String networkId;
	private String moteIp;
	private String message;
	private NetworkMessageCategory messageCategory;
	private long date;
	private boolean sendToQueue;

	public String getNetworkId() {
		return networkId;
	}

	public void setNetworkId(String networkId) {
		this.networkId = networkId;
	}

	public String getMoteIp() {
		return moteIp;
	}
```

FIGURA 31. ESTRUCTURAS DE ALMACENAMIENTO DE SENSORS-DAEMON

Ambas estructuras almacenan los datos del mote y de la red de la cual proviene el mensaje. Además se indica el medio por el cual el mensaje debe ser enviado al Sensors-Core (web service o MOM), el timestamp en el cual fue recibido y su categorización. El campo "message" en ambas estructuras contiene en texto plano el mensaje recibido desde la base de la red. Adicionalmente, si el mensaje proviene de un recurso sensor, se indica el tipo de sensor mediante el campo "sensorType".

La figura 32 ilustra el flujo de recepción de un mensaje proveniente desde la red.

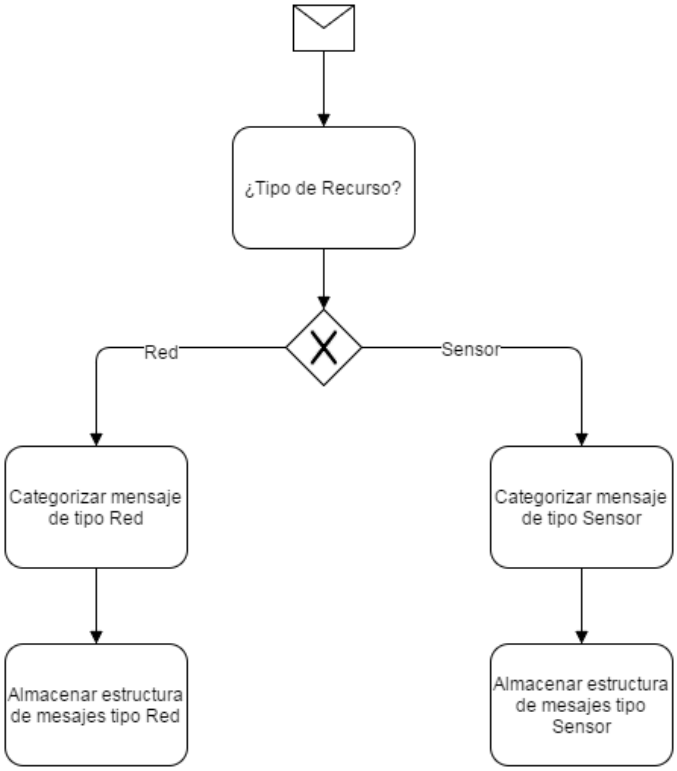

FIGURA 32. FLUJO DE RECEPCIÓN DE MENSAJES

En primer lugar se identifica si el mensaje proviene de un recurso Sensor o no. Esta categorización se realiza siguiendo la siguiente regla: si el recurso contiene en su URI el prefijo "/node/sensors", el mismo se categoriza como un mensaje de sensor, de lo contrario se categoriza como mensaje de red.

Luego se realiza una sub-categorización, indicada a continuación:

- Mensajes de tipo sensor:
  - Medida (*MEASURE*): medida del Sensor correspondiente.
  - Configuración (*CONFIG*): configuraciones del sensor (períodos de muestro u observación del sensor).
  - Información (*INFO*): información del sensor.

- Mensajes de tipo red:
  - Configuración del mote (NODE_CONFIG): configuraciones del nodo.
  - Carga del mote (NODE_CHARGE): información sobre el estado de carga de la batería del mote.
  - Ciclos de trabajo (NODE_DUTY_CYCLE): información sobre estados del procesador del mote.
  - Tabla de ruteo (NETWORK_ROUTE_TABLE): información sobre la tabla de ruteo del mote.
  - Información de vecinos (NETWORK_NEIGHBOUR): información sobre la tabla de vecinos del mote.
  - Información de estado de la Red/Mote (NETWORK_STATUS).
  - Mensaje Genérico (NETWORK_STATS): información extra del mote.

```
public enum SensorMessageCategory {

        MEASURE,
        CONFIG,
        INFO;

}
```

```
public enum NetworkMessageCategory {

        NODE_CONFIG,
        NODE_CHARGE,
        NODE_DUTY_CYCLE,

        NETWORK_ROUTE_TABLE,
        NETWORK_STATUS,
        NETWORK_NEIGHBOUR,
        NETWORK_STATS;

}
```

FIGURA 33. MENSAJES DE TIPO SENSOR Y RED

4.2.3.5 Pedidos CoAP hacia Sensors-Daemon

Algunas de las funcionalidades realizan cambios sobre la red de sensores. Para esto, es necesario contemplar la falta de conexión del Sensors-Daemon.

Cuando Sensors-Core desea enviar un pedido hacia una determinada red de sensores, en primer lugar este intenta enviarlo vía Web Service (mediante la API REST descripta en el punto 4.2.3.3) hacia el Sensors-Daemon correspondiente de dicha red. Si el pedido puede enviare correctamente, esto implica que el Sensors-Daemon se encontraba en su ciclo de conexión en el momento del envío y no ocurre ningún inconveniente. Si el pedido no puede enviarse correctamente, el mismo es almacenado en la base de datos PostgreSQL para su posterior re-envío. En el punto "Sincronización de datos entre Sensors-Core y Sensors-Daemon" se describe el proceso de re-envío.

La figura 34 muestra la estructura de la base de datos en la que se almacenan los datos del pedido a re-enviar.

```
@Entity
@Table(name = "daemon_pending_request")
public class DaemonPendingRequest extends AbstractEntity<Long> {

        /**
         *
         */
        private static final long serialVersionUID = 1L;

        @Column(name = "uri")
        private String uri;

        @Column(name = "request_method")
        private String action;

        @Column(name="put_value")
        private String putValue;

        @Column(name = "error_message")
        private String errorMessage;

        @Column(name = "retries")
        private int retries = 3;

        @Column(name="network_base_ip")
        private String networkBaseIp;
```

FIGURA 34. ESTRUCTURA DE ALMACENAMIENTO TEMPORAL DE MENSAJES COAP

Los principales campos que almacena la estructura son los siguientes:

- **Uri**
  Contiene la URI CoAP a ser invocada.

- **Action**
  Método CoAP que se desea invocar (PUT, GET, OBSERVE).

- **Put Value**
  Valor con el cual se desea realizar la acción PUT. Este campo tiene sentido únicamente si el pedido es de dicho tipo.

- **Network Base IP**
  Corresponde a la IP de la base de la red sobre la cual se desea realizar el pedido. Este campo es necesario para consultar los mensajes pendientes de una red en particular.

4.2.3.6 Tareas programadas en Sensors-Daemon

El componente Sensors-Daemon está compuesto casi en su totalidad por tareas asincrónicas o procesos batch (también llamadas tareas programadas o Jobs). Esto se debe a que el procesamiento que hace este componente no es controlado por el usuario y su tarea principal es la sincronización con la red.

TAREAS PROGRAMAS CON SPRING

Para la implementación de dichos procesos se utilizaron las tareas programadas que brinda Spring. Para esto se implementa un método Java y mediante una serie de anotaciones se indica cada cuanto tiempo se debe ejecutar el mismo. Para configurar este tiempo se utilizaron las denominadas expresiones cron [62] mediante un archivo de configuración [63]. La figura 35 muestra un ejemplo de configuraciones de estas expresiones.

```
# Scheduled Jobs
networkDiscoverJob.cron = 0 0 0/1 * * ?
connectJob.cron = */120 * * * * ?
motesRefresherJob.cron = 0 0 0/1 * * ?
```

FIGURA 35. CONFIGURACIÓN DE JOBS CON ARCHIVO DE PROPIEDADES

Tareas programadas en Sensors-Daemon

Sensors-Daemon está formado por tres Jobs. Las funcionalidades de cada uno de ellos son las siguientes.

- **Job de conexión**
  Este Job se encarga de obtener la conexión a internet, enviar los datos pendientes que estén almacenados localmente y esperar hasta que Sensors-Core envíe sus pedidos pendientes.

- **Job de sincronización**
  Este Job tiene como tarea la sincronización de los motes de la red, ya que eventualmente, podrían agregarse nuevos motes y estos deben ser reportados a *Sensors-Core*. En este proceso, para cada nuevo mote encontrado se ejecuta el descubrimiento de todos sus recursos, utilizando el método de descubrimiento CoAP.

- **Job de actualización de la fecha-hora de motes**
  Este Job envía periódicamente la fecha y hora actual a todos los motes de la red para que estos se mantengan sincronizados. Esto se realiza con el fin de evitar desfasajes de tiempos en las medidas reportadas.

Control de tareas programas en Sensors-Daemon

En la solución construida es posible controlar la ejecución de las tareas previamente mencionadas. Para esto, cuando cada Job inicia un ciclo de ejecución chequea si está habilitado. Esta configuración es almacenada en la base de datos local de Sensors-Daemon.

4.2.3.7 Descubrimiento inicial de una red

Este proceso es ejecutado cuando se inicia el servidor de Sensors-Daemon. Como fue mencionado en el punto 4.2.3.1, Sensors-Daemon conoce la IP de la base de la red de sensores que atiende y esta es utilizada en este proceso para el descubrimiento inicial de la red.

Una vez construido el contexto de Spring de Sensors-Daemon, se ejecuta un método que sincroniza inicialmente la red de sensores. Este mecanismo es útil ya que de otra forma se debería esperar a la primera ejecución del Job de sincronización.

En la figura 36 se puede observar el método que se ejecuta para inicializar la red de sensores.

```
@PostConstruct
public void initializeNetowrk() {

    boolean jobEnable = this.jobConfigService.isActive(JobName.INITIAL_DISCOVER);

    if (jobEnable) {
        LOGGER.info("Initial Discovery is Running");
        //Manda la ip de la base como mote y lo agrega a la lista que se mantiene local de motes
        this.discoverNetworkBean.addBaseToMotes();
        this.discoverNetworkBean.discoverNetwork();
        LOGGER.info("Initial Discovery is Finished");
    }
    else{
        LOGGER.info("Initial Discover is not Enabled");
    }

}
```

FIGURA 36. MÉTODO DE INICIALIZACIÓN DE UNA RED

La anotación @PostConstruct de Spring indica que se desea que el método "initializeNetwork" sea ejecutado una vez instanciado el objeto que lo contiene y no es necesario esperar a que se ejecute el Job de sincronización de la red.

4.2.3.8 Sincronización de datos entre Sensors-Core y Sensors-Daemon

El componente Sensors-Daemon está diseñado para disponer de conexión a internet de manera intermitente mediante conexión 3G. Para realizar esta tarea el mismo controla la conexión y desconexión del modem a demanda según sea necesario. Una vez conectado, se debe realizar la sincronización de datos entre Sensors-Daemon y Sensors-Core, esto implica que Sensors-Daemon envíe todos los mensajes recibidos de la red y que Sensors-Core envíe todos los pedidos pendientes. En este punto se describe el proceso de sincronización entre el componente mencionados.

El equipo de Gervasio se encargó del diseño del instrumento físico que controla el encendido y apagado del modem. Además proporcionó una serie de scripts implementados en Python [64] con funciones para el manejo de la conexión y la desconexión del modem.

El encargado de realizar la tarea de sincronización de datos es el Job de conexión y el mismo utiliza las herramientas brindadas por Gervasio para el manejo de conexión del modem. A continuación se describen los pasos que ejecuta el mismo.

Ejecución de Job de conexión

La figura 37 muestra un diagrama de secuencia donde se detallan los pasos ejecutados por el Job y como participan los diferentes componentes.

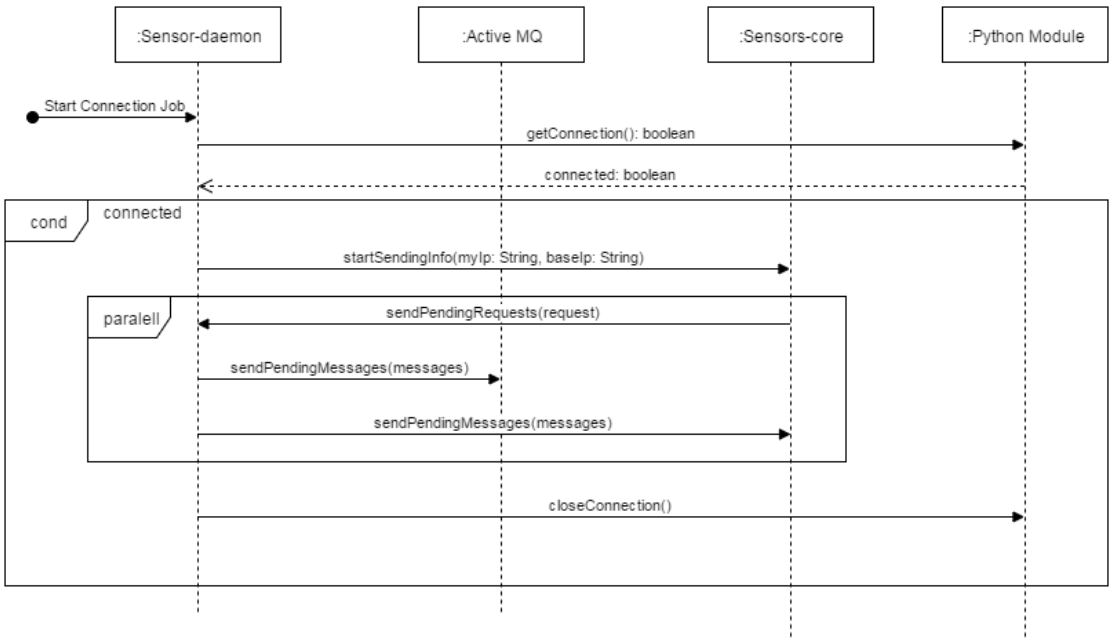

FIGURA 37. DIAGRAMA SECUENCIA JOB CONEXIÓN

En este proceso participan cuatro componentes: Sensors-Daemon, Sensors-Core, ActiveMQ y el módulo de Python para el manejo de la conexión.

La primera tarea del Job es solicitar la conexión a internet y una vez lograda la misma enviar todos los datos pendientes. Posteriormente, Sensors-Daemon envía a Sensors-Core un pedido para que este le envíe las solicitudes pendientes que están asociadas a la red que el primero atiende. Finalmente, se cierra la conexión de internet y termina el ciclo.

La figura 38 muestra un diagrama de procesos donde puede verse el proceso completo que se ejecuta en cada ciclo del Job.

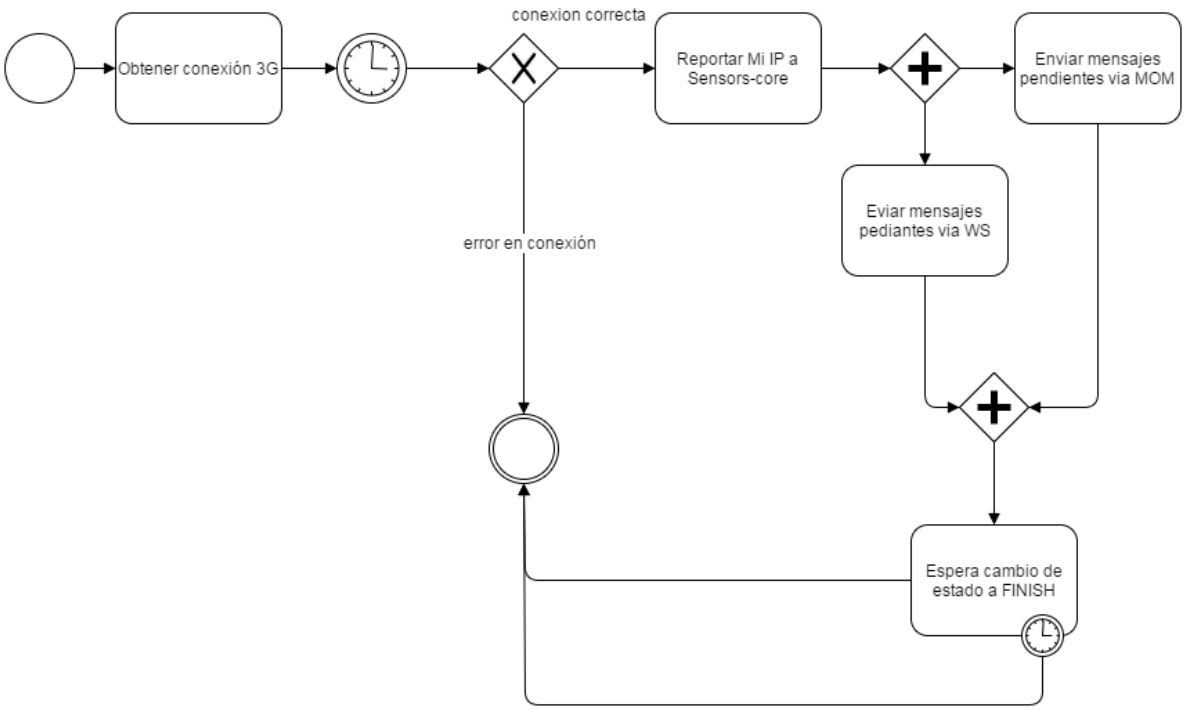

FIGURA 38. PROCESO EJECUTADO POR JOB DE CONEXIÓN

A continuación se detalla cada uno de los pasos realizados por el Job:

1) Sensors-Daemon solicita la conexión hacia el módulo Python. Luego se espera un lapso predefinido para dar tiempo al modem de generar la conexión. Si fue posible generar la conexión se prosigue con el ciclo, de lo contrario el ciclo se finaliza.

2) Si se logró establecer la conexión a internet, se procede a actualizar una bandera que indica que el Sensors-Daemon se encuentra en estado de recibir pedidos desde Sensors-Core.

   Además se informa a Sensors-Core:

   - La IP que tiene Sensors-Daemon ya que en cada ciclo de conexión esta puede cambiar. Es necesario que Sensors-Core conozca dicha IP para poder realizar el envío al Sensors-Daemon de las solicitudes pendientes.
   - La IP de la base de la red que atiende, de modo que Sensors-Core envíe solamente las solicitudes de esa red.

3) A continuación se comienzan a ejecutar en paralelo dos tareas:

   3.1) Por un lado Sensors-Daemon envía la información pendiente a Sensors-Core, almacenada de manera local en la base de datos SQLite. Para esto se realizan dos procedimientos también en paralelo: el primero envía los mensajes marcados con canal Web Service, mientras que el segundo envía los mensajes correspondientes a envío por MOM.

3.2) Por otro lado Sensors-Core envía sus pedidos pendientes a Sensors-Daemon.

Una vez terminada la tarea de 3.1) se chequea la bandera de estado mencionada en el paso 2. Esta bandera es actualizada por una invocación de Sensors-Core cuando termina el envío de sus datos pendientes de 3.2). Si la bandera no ha cambiado aún, el Job no avanzará hasta que la misma no cambie de estado. Esto garantiza que se reciben todos los pedidos pendientes de Sensors-Core. Existe un timeout de forma que si no llega la invocación mencionada desde Sensors-Core igualmente se continuará con la ejecución del Job.

4) La última tarea es notificar al módulo Python que se desea desconectar el modem de internet.

En la solución propuesta, el algoritmo optimiza el tiempo de uso del modem 3G. Es por esto que una vez hecha la conexión a internet, ambos componentes (Sensors-Core y Sensors-Daemon) se envían mutuamente la información en el mismo período de tiempo.

4.2.3.9 Recepción de medidas y chequeo de las alarmas

Recepción de las Medidas

Cuando una medida es recibida desde Sensors-Daemon, la misma es almacenada de manera asincrónica en la base de datos no relacional MongoDB.

Luego de manera asincrónica se ejecuta un algoritmo para detectar si se debe disparar alguna de las alarmas. Para la implementación de este algoritmo se utilizó un patrón de diseño denominado “Strategy”. Siguiendo este patrón, se define una interfaz que contiene la firma de la operación “checkAlarm”. Se definieron también tres clases, una por cada tipo de alarma, que implementan la interfaz mencionada. La operación “checkAlarm” recibe como parámetros la alarma a chequear, el tipo de sensor de la medida recibida, el valor de la misma y el mote sobre el cual fue reportada.

Como muestra la figura 39, una vez obtenidas todas las alarmas de la red, estas son colocadas en una cola de tipo concurrente y luego se ejecuta el chequeo de cada una de estas alarmas, en una serie de hilos en paralelo. Esto colabora en paralelizar y agilizar la tarea del chequeo de las diferentes alarmas.

```
public void checkAlarms(String measureValue, String moteIp, String sensorType) {

    try {
        Mote mote = this.moteDAO.loadWithNetworkByIp(moteIp);
        Sensor sensor = this.sensorBean.findSensorByUriType(mote, sensorType);

        List<Alarm> networkAlarms = this.alarmDAO.findByNetworkId(mote.getNetwork().getId());

        // Aramos una cola donde se irán tomando las alarmas para chequear si aplica

        Queue<Alarm> alarmQueue = new ConcurrentLinkedQueue();
        alarmQueue.addAll(networkAlarms);

        for (int i = 1; i <= MAX_THREADS; i++) {
            this.checkPararelAlarm(measureValue, sensor.getType(), mote, alarmQueue);
        }


    } catch (RuntimeException e) {

    }


}

private void checkPararelAlarm(String measureValue, SensorType sensorType, Mote mote, Queue<Alarm> alarmQueue) {

    while (!alarmQueue.isEmpty()) {

        Alarm alarm = alarmQueue.poll();
        this.alarmCheckers.stream().
                                        filter(ch ->
                                                alarm.getType().equals(ch.checkerAlarmType()))
                                                        .findFirst().get()
                                                        .checkAlarm(alarm, sensorType, mote, measureValue);

    }

}
```

FIGURA 39. CHEQUEO DE ALARMAS EN PARALELO

Mapa de expiración para Alarmas Contador y Mixta

Dado que las alarmas de tipo contador y mixta deben mantener información sobre ocurrencias de eventos en el tiempo, se diseñó un mecanismo para realizar esta tarea.

Para este mecanismo se utilizó una implementación de Mapa denominada Expiring Map [65]. La misma permite definir mapas en memoria asignando un tiempo de vida para cada una de sus entradas.

Tomando como ejemplo las alarmas de tipo contador, la estructura de Expiring Map utilizada para almacenar los eventos ocurridos se muestra en la figura 40.

```
private ExpiringMap<Long,Map<Date,Long>> counterSimpleAlarmInstances = ExpiringMap
        .builder()
        .variableExpiration()
        .build();
```

FIGURA 40. EXPIRING MAP UTILIZADO EN ALARMAS DE TIPO CONTADOR

El Expiring Map utiliza como clave el id de una alarma y como valor un mapa indicando las ocurrencias de la misma. Este segundo mapa utiliza como clave la fecha de la ocurrencia y como valor, el id del mote en donde fue registrada la ocurrencia.

La particularidad del Expiring Map radica en que cuando una entrada es agregada se le indica un tiempo de vida. Por lo tanto, cuando se agrega al mapa una alarma que aún no tiene ocurrencias, se define como tiempo de vida el período configurado para esta alarma. Esto significa que las ocurrencias agregadas para dicha alarma se borrarán cuando pase el tiempo de vida definido.

Para chequear que una alarma debe ser notificada se obtiene el mapa almacenado dentro del Expiring Map y si el tamaño del mismo es igual al valor de ocurrencias configurado por la alarma, debe notificarse.

4.2.3.10 Manejo de capas

Para la gestión y el manejo de capas agregadas por usuarios en el mapa de una red, se mantiene información en dos componentes distintos.

Por un lado en la base de datos Sensors-Core-DB se almacenan los nombres de las capas vinculadas a su red correspondiente. Por otro lado, la información geográfica de las capas se almacenan en el Geo Server en formato Shapefile.

El Geo Server se despliega en un servidor Tomcat [66] con URI de acceso "geoserver". Una vez instalado, se utiliza un workspace particular de forma de mantener agrupada y centralizada la información. Los workspaces se utilizan como contenedores lógicos para organizar elementos dentro del Geo Server y como forma de mantener ordenadas y separadas las capas almacenadas por usuarios del sistema.

Geo Server provee una API REST mediante la cual, a través de llamadas REST permite obtener las distintas capas almacenadas por su nombre correspondiente.

Cuando Sensors-Web solicita a Sensors-Core los datos de una determinada red para ser visualizada, entre estos se pide una lista de nombre de capas asociadas a dicha red. Luego, estas son obtenidas desde Geo Server para su visualización en el mapa correspondiente.

4.2.3.11 Servidor de SMS

Para esta tarea se utilizó un proveedor pago denominado Twilio [67]. El mismo permite el envío de SMS a través de una API REST. Para la utilización de este servicio se debe crear una cuenta en la página web del producto.

La herramienta brinda un dashboard mostrando graficas de la cantidad de mensajes utilizados en los últimos meses, así como también información del usuario tal como los números de teléfono que tiene disponibles y los servicios habilitados para el mismo.

Recent Messages View all Messages Logs

DATE	SERVICE	DIRECTION	FROM	TO	# SEGMENTS	STATUS	MEDIA
20:12:34 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—
20:12:35 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—
20:12:34 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—
20:12:34 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—
20:12:34 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—
20:12:30 UTC 2016-02-08	—	Outgoing API	(201) 884-2952	+59898874886	1	Delivered	—

FIGURA 41. DASHBOARD TWILIO, ÚLTIMOS MENSAJES ENVIADOS

5. Proceso de pruebas

En el presente capítulo se describen las pruebas realizadas para la validación de los componentes del sistema y las herramientas utilizadas para este fin.

5.1 Herramientas utilizadas

5.1.1 Simulador Cooja

Cooja [68] es un simulador de WSN donde cada mote utiliza Contiki OS como sistema operativo.

En Cooja se pueden simular redes, para lo que se proveen las siguientes funcionalidades:

- Motes de la red: se muestran todos los motes de la red y de cada uno se puede visualizar su información.
- Control de simulación: es posible iniciar, pausar y recargar una simulación, así como también controlar el tiempo y velocidad de ejecución.
- Visor de mensajes: se muestran los mensajes y eventos ocurridos en la red de simulación.
- Debug: se brinda la funcionalidad de debug, donde se puede analizar el código que se ejecuta en cada uno de los motes (con la posibilidad de utilizar puntos de interrupción). También es posible obtener el valor de las variables utilizadas durante la ejecución.

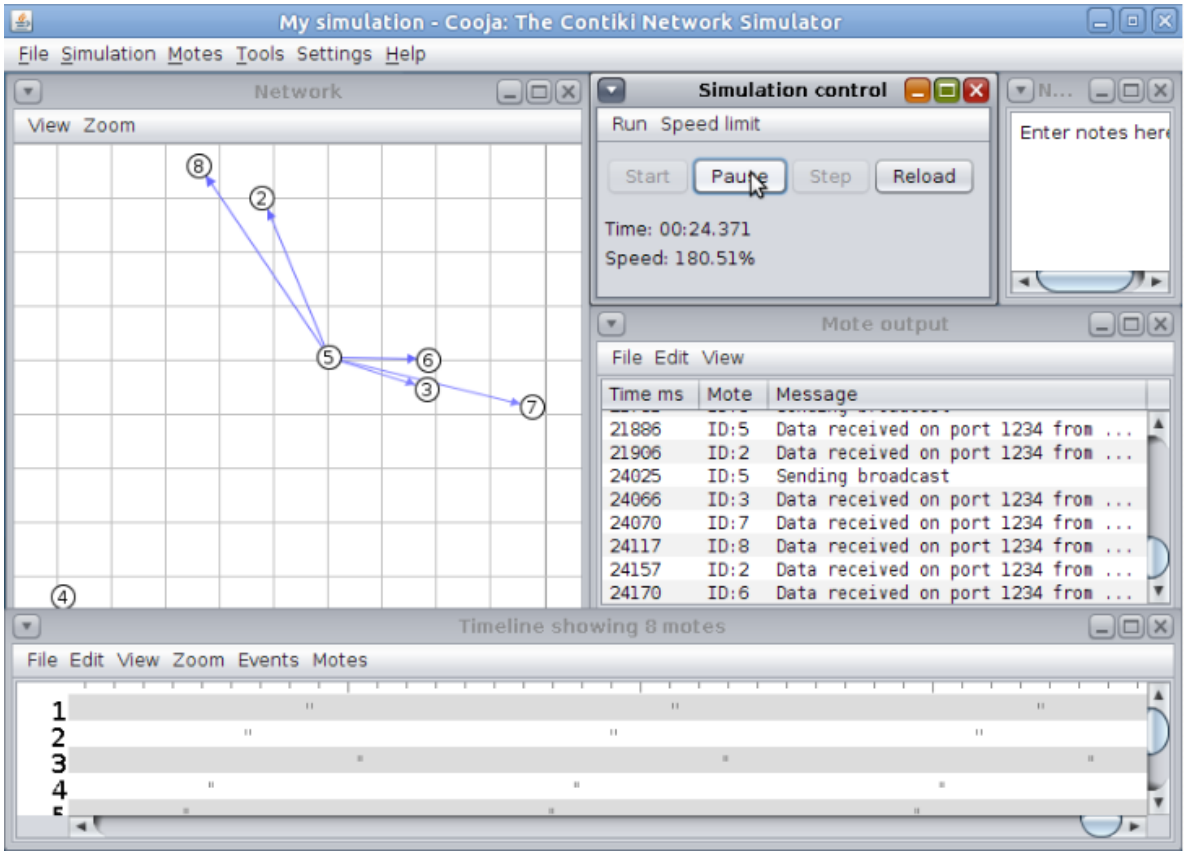

FIGURA 42. SIMULADOR COOJA

Para definir una simulación se debe seleccionar el tipo de mote utilizado en la misma. El simulador Cooja brinda la opción de crear un nuevo tipo de mote, indicando una descripción y el firmware a utilizar, esto es un archivo fuente en lenguaje C que es compilado antes de la creación del nuevo tipo de mote.

Una vez compilado con éxito el firmware es posible agregar motes de este tipo a la simulación creada.

De esta forma se emulan los motes de una red real, y es posible interactuar con ellos vía CoAP.

5.1.2 Copper

Copper [69] es un complemento para el navegador Firefox. El mismo brinda un cliente amigable para interactuar con un servidor CoAP y de esta manera acceder a sus recursos. La herramienta permite generar pedidos CoAP para las solicitudes GET, POST, PUT, OBSERVE, DELETE. Cooper muestra toda la información de los paquetes intercambiados mediante una interfaz sencilla. En la sección izquierda de su pantalla, muestra el árbol de recursos que un servidor CoAP ofrece a los clientes, como se muestra en la figura 43.

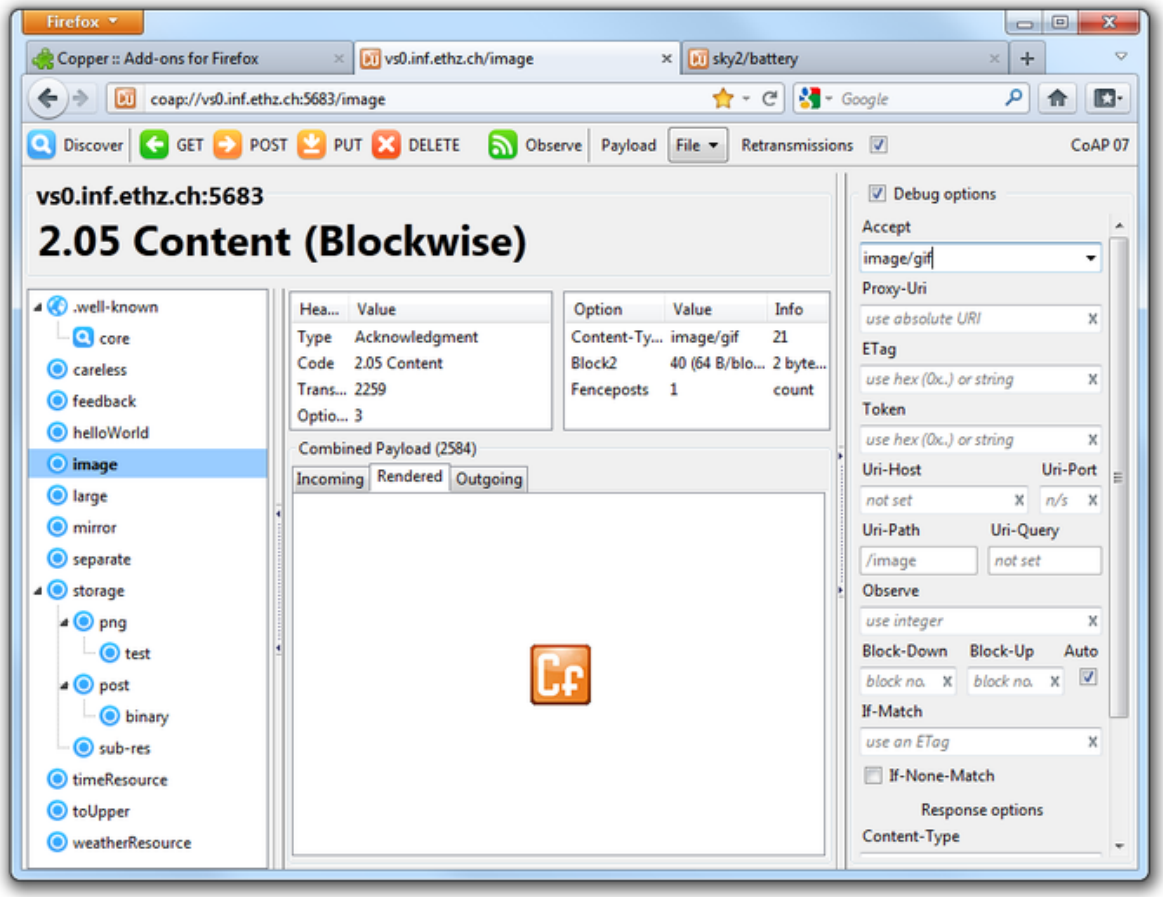

FIGURA 43. COMPLEMENTO COPPER

5.2 Validación del sistema con el simulador

Con el objetivo de validar el sistema implementado se plantean ciertos escenarios para probar el correcto funcionamiento de los siguientes componentes:

1. Formato acordado de mensajes CoAP proveniente de la red de sensores.
2. Funcionalidades de Sensors-Daemon.
3. Comunicación entre Sensors-Daemon y Sensors-Core.
4. Funcionalidades de Sensors-Core.
5. Comunicación entre Sensors-Core y Sensors-Web.
6. Funcionalidades de Sensors-Web.

En una primera etapa, se utilizó el simulador Cooja para validar la correcta sintaxis de los recursos definidos y el formato de la respuesta de los mismos.

Las primeras pruebas fueron únicamente con la herramienta Copper, validando las URIs de los recursos según lo acordado con miembros del equipo Gervasio. Junto con esto, fue posible validar también el formato de la respuesta de cada recurso.

En una segunda etapa se validó el funcionamiento del componente Sensors-Daemon junto con el simulador. Esto implicó el reconocimiento de la red de sensores simulada y de los recursos de cada uno de sus motes, así como también la observación de medidas. Para esto se procedió a desplegar Sensors-Daemon junto con sus dependencias, de manera local, en el mismo equipo que el simulador.

Se logró validar lo siguiente:

- Se identifica la red completa correctamente.
- Se almacena en buffer todo mensaje recibido.
- La comunicación CoAP se realiza correctamente.
- Desde el Sensors-Daemon es posible el envío de pedidos CoAP a los nodos de la red: GET, OBSERVE, PUT.
- Se reciben los mensajes de la red con formato acordado.

Los integrantes del proyecto Gervasio fueron los encargados de proveer y mantener el firmware de los motes del simulador. Por esta razón los errores encontrados se debieron reportar al equipo del proyecto Gervasio, y esperar la corrección de dicho firmware. La ejecución de la primera y segunda etapa se repitieron hasta no encontrarse más errores en la sintaxis de los mensajes CoAP.

Una vez verificado el comportamiento de los motes en comunicación con el componente Sensors-Daemon, el siguiente paso fue la integración entre este y el Sensors-Core. Para esto, además de desplegar el simulador y Sensors-Daemon se desplegó de manera local el Sensors-Core, incluyendo las dependencias y componentes que estos requieren: sus bases de datos correspondientes y el servidor ActiveMQ de mensajería para la comunicación entre ambos.

Una vez desplegado el sistema mencionado, se pudo verificar la comunicación entre estos componentes. Esto se realizó consultando la información almacenada en las base de datos de Sensors-Core y verificando que tanto las medidas como la información de la red se encontraban presentes y eran consistentes. Por otra parte se validó también la API REST provista por Sensors-Core, invocando todos los servicios de manera independiente.

Con la finalización de las pruebas de la comunicación entre ambos componentes, se logró verificar que el flujo de información entre Sensors-Daemon y Sensors-Core se realiza correctamente.

La última etapa de las pruebas consistió en agregarle al despliegue local el componente Sensors-Web. Esto requirió el despliegue de Geo Server en el servidor para almacenar la información geográfica correspondiente.

Con esta última etapa de las pruebas, se logró la validación del sistema completo utilizando la red simulada en un entorno local.

5.3 Validación de Sensors-Daemon en Raspberry Pi

Dado que Sensors-Daemon fue diseñado para ejecutarse sobre un Raspberry Pi, se validó su ejecución en este ambiente. Para esto, se desplegó el componente Sensors-Daemon junto con su base de datos en una Raspberry Pi y se verificó que se iniciara correctamente el servidor y que se conectara a la base de datos.

En el punto 5.4 se detallan las pruebas realizadas junto con el equipo de Gervasio en las que se validó la comunicación del componente Sensors-Daemon con una red real.

5.4 Ambiente preproducción

Una vez alcanzado un estado estable del sistema, se puso en funcionamiento el ambiente de preproducción junto con integrantes del proyecto Gervasio.

El ambiente de preproducción consistió en un Raspberry Pi ejecutando el Sensors-Daemon que se comunica con la base de una red dispuesta en la Facultad de Ingeniería.

Por otro lado se dispuso de un servidor en la nube Microsoft Azure [70]. En este servidor se instaló Sensors-Core y Sensors-Web junto con todas las dependencias de los mismos.

Luego de realizada esta prueba, surgió un problema de conectividad, que obligó a cambiar el ambiente. Este inconveniente se debió a que desde la aplicación Web no es posible enviar pedidos hacia la red de sensores. Esto se debío a que el componente Sensors-Core y Sensors-Web se ejecutaron en la nube pública de Azure, pero no así el componente Sensors-Daemon, que se encontraba conectado a una red privada interna de la facultad, por lo que su IP no era accesible desde fuera.

Teniendo en cuenta estos inconvenientes, se tomó la decisión junto con integrantes de Gervasio de sustituir el servidor en la nube por una máquina virtual ejecutando dentro de un servidor de Facultad de Ingeniería, permitiendo así que todos los componentes se encontraran en la misma red.

Si bien este último escenario es el más próximo al ambiente productivo del sistema, no se realizó la validación de la conexión del Sensors-Daemon a través del modem 3G, sino que se utilizó conexión cableada, lo que asegura que en cada ciclo habrá siempre conexión.

6. Gestión del Proyecto

En este capítulo se describe el proceso seguido a lo largo del proyecto. Se detallan las etapas planificadas en un inicio y luego se describen las desviaciones surgidas a lo largo del proyecto.

Cabe destacar que si bien este proyecto tiene sus objetivos propios, el mismo acompaña al proyecto Gervasio que avanza en forma paralela. Es por esto, que cualquier desvío o cambio en la planificación y el avance de uno, afecta directamente al otro.

6.1 Planificación Inicial

En el marco de la asignatura "Proyecto de Grado" de la Universidad de la República, el proyecto da inicio en el mes de Abril de 2015.

En una primera instancia se efectuó una reunión con la tutora para que se dieran las pautas iniciales. En esta reunión, la tutora explicó el objeto del proyecto, cuáles eran sus antecedentes y el camino a seguir. Además, la tutora proporcionó la solución de una implementación previa, realizada por estudiantes de Proyecto de Ingeniería de Software.

En función de lo mencionado se planificaron los tiempos y etapas del proyecto. En la figura 44 se muestra un diagrama de Gantt con la planificación inicial del proyecto.

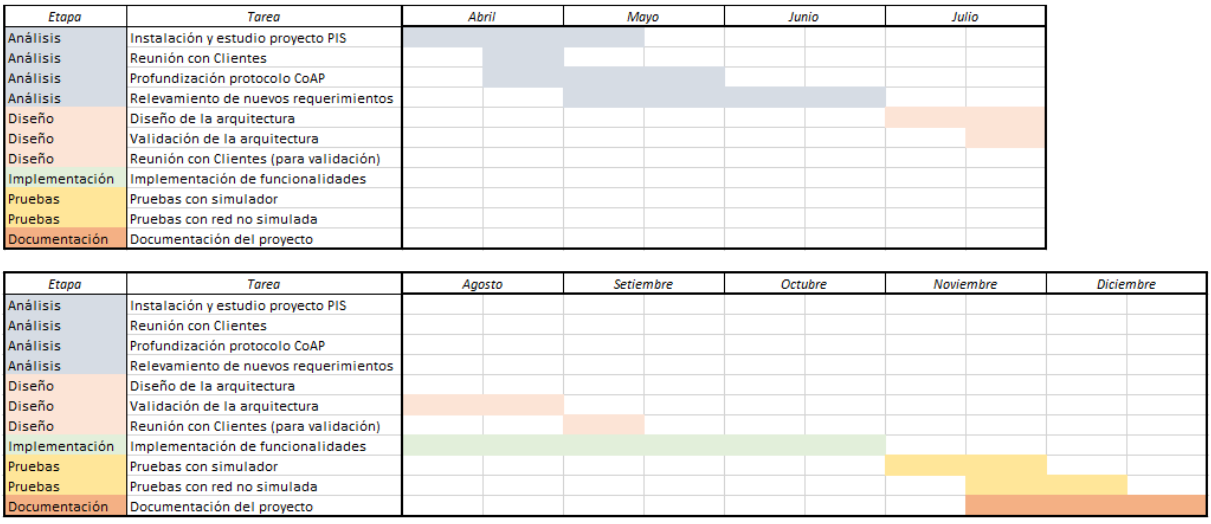

FIGURA 44. PLANIFICACIÓN ORIGINAL DEL PROYECTO

Durante el transcurso del proyecto, ocurrieron desvíos respecto a lo planificado que se detallan en la sección 6.3, donde se exponen también sus posibles causas.

6.2 Etapas desarrolladas

En esta sección se describen las etapas realizadas y las tareas dentro de las mismas.

6.2.1 Análisis del problema

6.2.1.1 Profundización de la temática

En la primera etapa se analizó el problema planteado y se estudiaron los conceptos necesarios. Tuvo lugar una reunión con la tutora del proyecto para conocer en detalle el alcance del mismo.

La tutora proporcionó el código fuente de una solución para el mismo problema realizada por estudiantes de la materia PIS, junto con su documentación. Luego se ejecutó dicho software y se examinaron las funcionalidades del mismo. Además, se realizó una inspección sobre el código para conocer algunos aspectos técnicos.

En una segunda etapa se realizó una reunión con los clientes, integrantes del proyecto Gervasio. En la misma, se pudo conocer más profundamente el trabajo que ese proyecto estaba realizando con redes de sensores y además se relevaron requerimientos adicionales.

Por último, se recibió del equipo de Gervasio la configuración del simulador Cooja para realizar las pruebas iniciales.

6.2.1.2 Profundización en protocolo CoAP

En esta etapa se estudió el protocolo utilizado para la comunicación con los sensores de la red y su especificación. Además se realizaron pruebas de concepto con las herramientas existentes para la integración de dicho protocolo con el lenguaje Java.

6.2.1.3 Especificación de nuevos requerimientos

La última tarea dentro de la etapa de Análisis, fue el relevamiento de los nuevos requerimientos planteados por integrantes de Gervasio. Como resultado se acordó reformular requerimientos ya existentes así como generar nuevos.

6.2.2 Diseño de la solución

En esta etapa se comenzaron a investigar las tecnologías para realizar la solución y se planteó una arquitectura para el problema a resolver.

6.2.2.1 Pruebas con Raspberry

Una vez seleccionadas las tecnologías, se comenzaron a realizar pruebas de concepto con las mismas para validar que eran apropiadas para la solución planteada.

En particular, se comenzó a trabajar con las tecnologías consideradas de mayor riesgo técnico, como por ejemplo el Raspberry Pi. Se realizaron pruebas de pequeño porte y esto permitió evitar ciertos riesgos técnicos.

6.2.2.2 Diseño de Recursos CoAP

Un proceso crítico que comenzó en esta etapa, fue el diseño y formulación de un estándar para la sintaxis de los mensajes CoAP. Inicialmente se realizó una propuesta a integrantes de Gervasio que fue finalmente aceptada luego de ciertas modificaciones.

Esta etapa fue de gran importancia ya que fue uno de los puntos de mayor desvío del proyecto.

6.2.2.3 Construcción de prototipo

Luego de diseñar el esquema de la arquitectura se realizó un prototipo de cada uno de los componentes planteados. Se implementó en cada uno de los componentes las funcionalidades críticas integrando los mismos mediante una comunicación básica de punta a punta.

La construcción de dicho prototipo insumió más tiempo del estimado ya que la inclusión del simulador fue indispensable y lograr ejecutarlo sin impedimentos implicó más tiempo del esperado.

6.2.2.3 Validación de prototipo

Una vez finalizado el prototipo, se realizó una reunión con los clientes para su validación. Allí se les mostró el avance del proyecto y las ideas sobre la arquitectura planteada.

Por otra parte, fue posible mostrar las funcionalidades implementadas con datos de prueba generados.

6.2.3 Implementación

Con la arquitectura validada, en esta etapa se desarrollaron las funcionalidades definidas. La implementación de las funcionalidades se realizó con los dos integrantes de este proyecto trabajando en paralelo.

En primera instancia se implementaron los componentes correspondientes al back-end del sistema, estos son, Sensors-Daemon y Sensors-Core. Con la estructura del prototipo desarrollada, esta tarea no presentó gran dificultad.

Por último se desarrolló el componente de front-end. Este demandó un tiempo importante, dado que las tecnologías elegidas eran novedosas y de poco conocimiento para el equipo.

En esta etapa de desarrollo se fueron realizando pruebas de integración entre los diferentes componentes implementados, a modo de detectar errores en forma temprana.

6.2.4 Pruebas y Validación del Sistema

Esta fue la etapa final del proceso de construcción. Una vez finalizado el desarrollo, se hicieron pruebas locales para validar el correcto funcionamiento del sistema completo.

En esta etapa, fue de interés de los clientes realizar una reunión para conocer la interfaz y el funcionamiento del sistema. Además, fue solicitada la instalación de los componentes Sensors-Web y Sensors-Core en servidores de Azure. De esta forma se comenzó el proceso de corrección de errores y se agregaron mejoras propuestas.

6.2.5 Documentación de la Solución

La etapa final del proyecto fue la elaboración de la documentación para la solución planteada.

La documentación presenta tanto el marco teórico y el contexto del proyecto como las tecnologías aplicadas y la arquitectura propuesta para la solución del problema.

6.3 Desviaciones

Durante el transcurso del proyecto surgieron desviaciones importantes en tiempo, con respecto a lo planificado.

Originalmente el proyecto se planificó para realizarse en 9 meses, y finalmente se implementó y finalizó en 16 meses. En el diagrama de Gantt de la figura 45, se muestra el tiempo insumido en cada tarea, y posteriormente las causas que determinaron que el proyecto se extendiese por 7 meses.

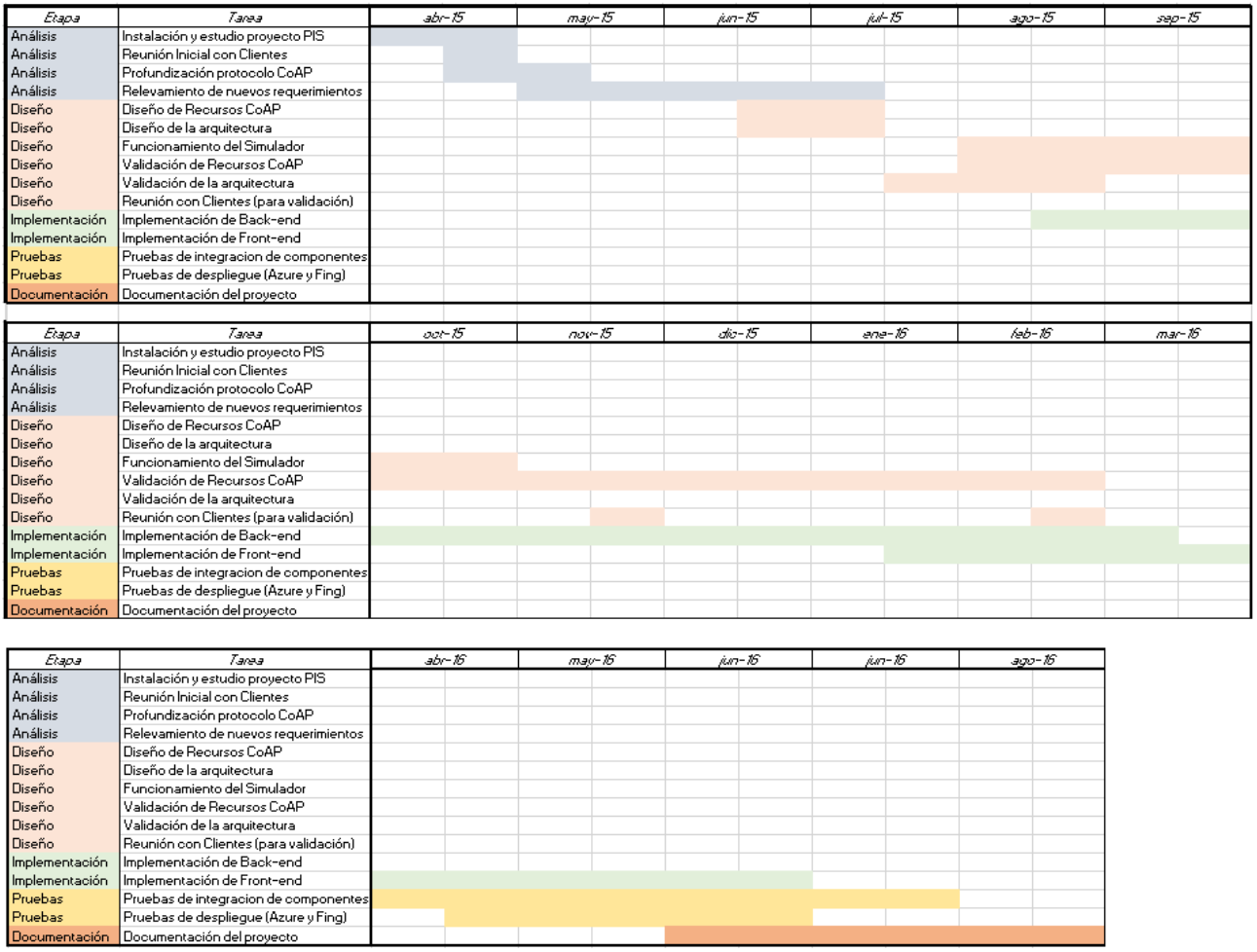

FIGURA 45. TIEMPOS INSUMIDOS POR EL PROYECTO

A continuación se describen las desviaciones del proyecto respecto a la planificación original:

- **Relevamiento de Requerimientos**
  Desde el comienzo fue planteada la posibilidad de una reunión con un Ingeniero Agrónomo, para relevar requerimientos funcionales nuevos pero esta pudo ser efectuada recién a mitad de Junio de 2015. Esto atrasó la tarea de análisis y definición de los requerimientos.

- **Funcionamiento del Simulador**
  Lograr ejecutar el simulador en las dos terminales donde se estaba realizando el desarrollo no fue una tarea fácil, ya que sus configuraciones dependen del sistema operativo y funcionan exclusivamente con compatibilidad de algunas versiones. Este trabajo tardó aproximadamente 3 meses, hasta que se logró dejar el simulador en un estado estable y funcionando. En el transcurso de este proceso, se mantuvo comunicación constante con un integrante de Gervasio que apoyó en esta tarea.

- **Validación de los recursos CoAP**
  La implementación del firmware de los motes fue provista por integrantes del proyecto Gervasio, por lo que esta tarea consistió en validar los mensajes recibidos desde la red simulada. Cuando un error en la sintaxis acordada era detectado, se notificaba a integrantes de Gervasio, para la modificación y corrección del mismo. Este proceso, fue muy prologando, ya implicaba interacciones repetidas con integrantes de dicho proyecto, por lo tanto, en muchas ocasiones transcurrían varias semanas para recibir las correcciones.

- **Implementación del Front-end**
  Otro aspecto que retrasó la ejecución del proyecto fue el riesgo asumido a la hora de implementar el front-end del sistema. La elección de una tecnología nueva y de poco conocimiento en los integrantes del equipo insumió un tiempo de aprendizaje importante.

7. Conclusiones y trabajo a futuro

Este capítulo presenta las conclusiones del proyecto y las posibles mejoras a ser realizadas como trabajo a futuro.

7.1 Conclusiones

El objetivo general del proyecto fue el diseño e implementación de un sistema de software capaz de comunicarse con las redes de sensores inalámbricas construidas por el equipo Gervasio, que permitiese consultar a través de un portal web la información recibida de los sensores.

Para cumplir con este objetivo fue necesario en una primera etapa realizar un análisis del protocolo CoAP, ya que es el utilizado por el equipo Gervasio para la comunicación con la red de sensores construida.

Se diseñó una arquitectura distribuida en tres componentes de forma tal de dividir las responsabilidades. Uno de los componentes se ejecuta sobre un Raspberry Pi ubicado cercano a la red de sensores, y es quien interactúa directamente con los motes de la red mediante CoAP. Por otro lado, se implementaron un componente de front-end y uno de back-end conteniendo toda la lógica de negocio junto con el almacenamiento de los datos del sistema.

En cuanto a las herramientas utilizadas para el desarrollo, se concluye que el protocolo CoAP es sencillo de comprender, teniendo como base el funcionamiento del protocolo UDP y REST. En este proyecto se utilizó Californium como cliente CoAP que es una herramienta muy sencilla de utilizar con la documentación brindada. Al tratarse de una implementación de un protocolo nuevo, se identifica como desventaja que no presenta una comunidad activa que ayude a evacuar las dudas rápidamente.

Por otra parte, se utilizó AngularJS para el desarrollo web. Esta era desconocida para los integrantes del proyecto pero se destaca que es una tecnología muy madura para el desarrollo de aplicaciones web. Además, posee una comunidad activa que colabora en la resolución rápida de incidencias y existe extensa documentación en internet.

El uso de las tecnologías, Californium y AngularJS, presentó el desafío técnico más grande, principalmente por desconocimiento de las mismas por parte de los integrantes de este proyecto.

Una vez finalizado el desarrollo del sistema, el mismo fue probado en un ambiente local con una red de sensores simulada. También fue posible validar la instalación de los componentes en un ambiente de pre-producción junto con los integrantes del proyecto Gervasio. Este ambiente consistió en la instalación de los tres componentes del sistema en la Facultad de Ingeniería, donde se instaló también una red de sensores real.

7.2 Trabajo a Futuro

Si bien el sistema construido cumple con los requerimientos iniciales, a continuación se mencionan algunas limitaciones de la solución implementada, así como también posibles mejoras que podrían implementarse.

- **Pedidos CoAP desde la web**

Una importante mejora radica en la posibilidad de realizar pedidos CoAP a demanda desde el portal web. Es decir, dar la posibilidad a los usuarios web de ingresar una URI sobre la cual realizar el pedido CoAP y luego mostrar su resultado. Al implementar esta funcionalidad se debería contemplar la posible falta de conexión del componente Sensors-Daemon, ya que su conexión es intermitente.

Se destaca que a pesar de que no se brinda esta funcionalidad en la web actual, es posible realizar pedidos CoAP a demanda mediante solicitudes REST a ciertos recursos (utilizando la API REST implementada).

- **Seguridad**

Un aspecto que se debería mejorar es la seguridad del sistema, tanto la seguridad de los accesos a servicios REST publicados por Sensors-Core, como en el manejo de los usuarios y sus roles. Podrían definirse mecanismos como API KEY, que permitan acceder a los servicios sólo a clientes autorizados.

- **Pruebas de campo**

Una tarea importante pendiente de realizar son pruebas de campo exhaustivas con una red de sensores real desplegada en un terreno cultivado junto con un Raspberry Pi que se comunique con la red para recolectar datos reales y en mayor cantidad.

Además, sería deseable poder probar el ciclo de conexión con el modem 3G para validar que la solución diseñada cumple con lo especificado.

- **Alarmas multi-sensor**

Una funcionalidad interesante que no fue requerida, es la de poder crear alarmas que involucren varios tipos de sensores a la vez. La condición lógica para disparar una alarma tomaría en cuenta las medidas de esos sensores.

- **Aplicación para Móviles**

Sería de interés brindar una aplicación móvil, donde tanto el agrónomo como el técnico pudiesen recibir notificaciones de las alarmas que se disparan. En el caso del técnico, sería interesante visualizar los parámetros de configuración de la red y recibir mediante notificaciones push las alarmas por inactividad de los motes. En el caso del agrónomo, se podrían generar las gráficas de las medidas de los sensores.

8. Referencias

[1] "Internet of Things (IoT)". En línea disponible en: http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html. Último acceso: 20 06 2016.

[2] F. Silveira, L. Barboni, L. Steinfeld, P. Mazzara, A. Gómez. "GERVASIO: Generalización de las redes de sensores inalámbricos como herramienta de valorización en sistemas vegetales intensivos". En línea disponible en: http://iie.fing.edu.uy/publicaciones/2014/SBSMG14. Último acceso: 22 06 2016.

[3] "Instituto de Ingeniería Eléctrica de la Universidad de la República del Uruguay". En línea disponible en: http://iie.fing.edu.uy/. Último acceso: 22 06 2016.

[4] "Facultad de Ingeniería de Universidad de la República del Uruguay". En línea disponible en: http://www.fing.edu.uy/. Último acceso: 22 06 2016.

[5] "Internet of Things: Wireless Sensor Networks". International Electrotechnical Commission. 2014. En línea disponible en: http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf. Último acceso: 03 07 2016.

[6] "Wireless Sensor Network". En línea disponible en: http://www.mfbarcell.es/conferencias/wsn.pdf. Último acceso: 02 07 2016.

[7] "Contiki OS". En línea disponible en: http://www.contiki-os.org/. Último acceso: 06 08 2016.

[8] A. Dunkels, B. Gronvall, T. Voigt. "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors". Swedish Institute of Computer Science. En línea disponible en: http://www.dunkels.com/adam/dunkels04contiki.pdf. Último acceso: 06 08 2016.

[9] "Web Service". En línea disponible en: https://www.w3.org/TR/ws-arch/#whatis. Último acceso: 21 06 2016.

[10] "What Are RESTful Web Services?". En línea disponible en: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. Último acceso: 21 06 2016.

[11] "Types of Web Services SOAP,XML-RPC and Restful". En línea disponible en: http://phpflow.com/php/web-service-types-soapxml-rpcrestful/. Último acceso: 21 06 2016.

[12] “API - Application Program Interface”. En línea disponible en: https://www.mulesoft.com/resources/api/what-is-an-api. Último acceso: 10 08 2016.

[13] “What is Middleware?”. En línea disponible en: https://www.mulesoft.com/resources/esb/integration-middleware-technology. Ultimo acceso: 22 06 2016.

[14] “Middleware Solutions & Services”. En línea disponible en: http://www.girnarsez.com/middleware . Último acceso: 22 06 2016.

[15] “MOM”. En línea disponible en: https://docs.oracle.com/cd/E19575-01/820-6424/aeraq/index.html. Último acceso: 23 06 2016.

[16] “Open Geospatial Consortium”. En línea disponible en: http://www.opengeospatial.org/ . Último acceso: 23 06 2016.

[17] “Web Map Service”. En línea disponible en: http://www.opengeospatial.org/standards/wms . Último acceso: 23 06 2016.

[18] “Web Feature Service”. En línea disponible en: http://www.opengeospatial.org/standards/wfs . Último acceso: 23 06 2016.

[19] “Geography Markup Language”. En línea disponible en: http://www.opengeospatial.org/standards/kml . Último acceso: 23 06 2016.

[20] “Keyhole Markup Language”. En línea disponible en: http://www.opengeospatial.org/standards/gml. Último acceso: 23 06 2016.

[21] “What is noSQL?”. En línea disponible en: https://www.mongodb.com/nosql-explained. Último acceso: 28 06 2016.

[22] "¿Qué es Big Data?". En línea disponible en: https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/. Último acceso: 11 08 2016.

[23] ”Raspberry Pi”. En línea disponible en: https://www.raspberrypi.org/documentation/ . Último acceso: 23 06 2016.

[24] “Proyectos Raspberry”. En línea disponible en: https://www.raspberrypi.org/forums/viewforum.php?f=15. Último acceso: 23 06 2016.

[25] “CoAP”. En línea disponible en: http://coap.technology/. Último acceso: 25 06 2016.

[26] "TCP Especification". En línea disponible en: https://www.ietf.org/rfc/rfc793.txt. Último acceso 10 08 2016.

[27] "Core Link, RFC 6690". En línea disponible en: https://tools.ietf.org/html/rfc6690 Último acceso: 12 07 2016.

[28] C. Bormann, A. Castellani, Z. Shelby. "CoAP: An Application Protocol for Billions of Tiny Internet Nodes". IEEE Internet Computing. 2012. En línea disponible en: https://www.computer.org/csdl/mags/ic/2012/02/mic2012020062-abs.html. Último acceso: 25 06 2016.

[29] X. Chen. "Constrained Application Protocol for Internet of Things". Wireless and Mobile Networking. 2014. En línea disponible en: http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/. Último acceso: 25 06 2016.

[30] "Uniform Resource Identifier (URI): Generic Syntax". En línea disponible en: https://tools.ietf.org/html/rfc3986#section-1.1.2 . Último acceso: 10 08 2016.

[31] X. Gimenez. "Desarrollo y Estudio del Protocolo Observe para CoAP". En línea disponible en: http://upcommons.upc.edu/bitstream/handle/2099.1/18314/Xavi_Gimeno_Gimenez.pdf. Universitat Polotécnica de Catalunya. Último acceso: 25 06 2016.

[33] W. Colitti, K. Steenhaut, N. De Caro. "Integrating Wireless Sensor Networks with the Web". En línea disponible en: http://web.cs.wpi.edu/~rek/IoT/Papers/Colitti_CoAP_paper.pdf. Worcester Polytechnic Institute. Último acceso: 25 06 2016.

[34] "Resumen Proyecto Gervasio". IEE, Facultad Ingeniería de la República. Último acceso: 22 06 2016.

[35] "Proyecto de Ingeniería de Software". En línea disponible en: https://www.fing.edu.uy/inco/cursos/pis/wikiPIS/field.php/Main.HomePage. Último acceso: 11 08 2016.

[36] Grupo 7, Proyecto de Ingeniería de Software 2014. "Especificación de Requerimientos de Software". En línea disponible en: https://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/dvd02/experiencia2014/material/grupo7/requerimientos/transicion/iter2/RQDRQG7v6.0.docx. Último acceso: 04 03 2016.

[37] Grupo 7, Proyecto de Ingeniería de Software 2014. "Descripción de la arquitectura", En línea disponible en: https://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/dvd02/experiencia2014/material/grupo7/disenio/transicion/iter2/DSARQG7v3.0.docx. Último acceso: 04 03 2016.

[38] Y. Zhua, J. Songa , F. Donga. "Applications of wireless sensor network in the agriculture environment monitoring". SciVerse ScienceDirect Procedia Engineering. 2011. En línea disponible en: http://www.sciencedirect.com/science/article/pii/S1877705811026324. Último acceso: 09 08 2016.

[39] "nRF2401". En línea disponible en: https://www.sparkfun.com/datasheets/RF/nRF2401rev1_1.pdf. Último acceso: 10 08 2016.

[40] "nRF240x ShockBurst™ technology". En línea disponible en: https://www.semiconductorstore.com/pdf/NewSite/nordic/WP_nRF240x_ShockBurst.pdf. Último acceso: 10 08 2016.

[41] A. Jiménez, D. Ravelo, J. Gómez. "Sistema de adquisición, almacenamiento y análisis de información fenológica para el manejo de plagas y enfermedades de un duraznero mediante tecnologías de agricultura de precisión". Tecnura, Tecnología y cultura, afirmando el conocimiento, Universidad Distrital Francisco José de Caldas. 2010. En línea disponible en: http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/6697. Último acceso: 09 08 2016.

[42] M. Flores-Medina, F. Flores-García, V. Velasco-Martínez. "Monitoreo de humedad en suelo a través de red inalámbrica de sensores". Tecnología y ciencias del agua. 2015 En línea disponible en: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-24222015000500006&lng=en&tlng=en. Último acceso: 09 08 2016.

[43] "Geo Server". En línea disponible en: http://geoserver.org/. Último acceso: 09 08 2016.

[44] "Jetty". En línea disponible en: http://www.eclipse.org/jetty/. Último acceso: 21 06 2016.

[45] "Maven". En línea disponible en: https://maven.apache.org/. Último acceso: 27 07 2016.

[46] "Spring". En línea disponible en: https://spring.io/. Último acceso: 27 07 2016.

[47] "AngularJS". En línea disponible en: https://angularjs.org/. Último acceso: 27 07 2016.

[48] "Active MQ". En línea disponible en: http://activemq.apache.org/. Último acceso: 22 06 2016.

[49] Java EE 6 Tutorial, "Basic JMS API Concepts". En línea disponible en: http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html#bnceb. Último acceso: 22 06 2016.

[50] "Bootstrap". En línea disponible en: http://getbootstrap.com/. Último acceso: 22 06 2016.

[51] "Mongo DB". En línea disponible en: https://www.mongodb.com/. Último acceso: 21 06 2016.

[52] "PostreSQL". En línea disponible en: https://www.postgresql.org/about/. Último acceso: 21 06 2016.

[53] "Two phase commit Mechanism". En línea disponible en: https://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_txns003.html. Último acceso: 11 08 2016.

[54] "Point in Time Recovery". En línea disponible en: http://www.postgresql.org.es/node/238. Último acceso: 11 08 2016.

[55] "PostGIS". En línea disponible en: http://postgis.net/. Último acceso: 21 06 2016.

[56] "Sqlite", [En línea]. Disponible en: https://www.sqlite.org/about.html. Último acceso: 21 06 2016.

[57] "Open Layers 3". En línea disponible en: http://openlayers.org/. Último acceso: 21 06 2016.

[58] "Zing Chart". En línea disponible en: https://www.zingchart.com/. Último acceso: 21 06 2016.

[59] "Free Marker". En línea disponible en: http://freemarker.org/. Último acceso: 23 07 2016.

[60] "Californium". En línea disponible en: http://www.eclipse.org/californium/. Último acceso: 23 07 2016.

[61] "JSON". En línea disponible en: http://www.json.org/. Último acceso: 23 07 2016.

[62] "Cron Expressions". En línea disponible en: https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.html. Último acceso: 12 08 2016.

[63] "Properties File". En línea disponible en: https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0204propertiesfileformat01.html. Último acceso: 12 08 2016.

[64] "Pyhton". En línea disponible en: https://www.python.org/. Último acceso: 12 08 2016.

[65] "Expiring Map". En línea disponible en: http://jodah.net/expiringmap/javadoc/net/jodah/expiringmap/ExpiringMap.html. Último acceso: 12 08 2016.

[66] "Tomcat". En línea disponible en: http://tomcat.apache.org/. Último acceso: 12 08 2016.

[67] "Twilio". En línea disponible en: https://www.twilio.com/. Último acceso: 12 08 2016.

[68] "Cooja Simulator". En línea disponible en: http://www.contiki-os.org/start.html. Último acceso: 12 08 2016.

[69] "Cooper". En línea disponible en: https://addons.mozilla.org/es/firefox/addon/copper-270430/. Último acceso: 17 06 2016.

[70] "Microsoft Azure". En línea disponible en: https://azure.microsoft.com/es-es/. Último acceso: 17 06 2016.

Apéndice 1. Descripción de herramientas utilizadas

Jetty

Jetty [44] es un servidor Web y contenedor de servlets open source basado en Java. Está preparado para ser utilizado como servidor independiente o embebido en una aplicación existente.

Este puede ser configurado a través de un archivo de configuración XML o utilizando el propio lenguaje Java embebido dentro de la aplicación y es portátil, ligero, robusto y extensible.

Esta herramienta fue aplicada en los tres componentes Sensors-Daemon, Sensors-Core y Sensors-Web para desplegar los servicios web en los primeros dos, y los contenidos estáticos de la web en el último caso.

Maven

Maven [45] es una herramienta utilizada para la gestión de proyectos Java, que ayuda al empaquetado de las aplicaciones haciendo sencilla esta tarea. Además, maneja las dependencias con librerías externas importando las mismas desde diferentes repositorios. Por otro lado, permite realizar aplicaciones modularizadas sin la necesidad de generar una única aplicación con la totalidad del código.

Su configuración es mediante Project Object Model (pom), un archivo xml donde se describe el proyecto y se realiza de manera centralizada el manejo de las dependencias, tanto de otros módulos como de librerías externas.

En particular, se utilizó Maven para la generación de los módulos del proyecto y administrar las dependencias.

Spring

Spring [46] es un framework open source para Java utilizado en el desarrollo de aplicaciones. Es considerado como una alternativa o incluso un complemento al modelo de EJB (Enterprise JavaBean).

Fue construido, principalmente, para resolver el paradigma de inyección de dependencias, pero con el paso de los años el proyecto fue creciendo y ampliándose en funcionalidades. Hoy en día brinda herramientas, metodologías y soporte para la conexión a variadas bases de datos de forma sencilla y uniforme a través de objetos de configuración, tanto SQL como no SQL.

Además, brinda mecanismos para el manejo de la transaccionalidad con las bases de datos y también implementaciones sencillas, para el manejo de tareas asincrónicas y de tipo batch.

Spring se encuentra implementado en módulos, de manera tal que es posible importar y utilizar únicamente las funcionalidades necesarias en un proyecto.

En particular, en este proyecto se utilizó *spring-context* para el manejo de la inyección de dependencias, *spring-orm* y *spring-data* para el manejo de accesos a la base de datos y por último *spring-web* y *spring-web-mvc* para la implementación de servicios web de tipo REST.

INYECCIÓN DE DEPENDENCIAS

La inyección de dependencias es un patrón que tiene como objetivo el desacoplamiento del código. Utilizando la inyección de dependencias, los componentes declaran sus dependencias pero no se encargan de instanciarlas, sino que es Spring el encargado de obtenerlas e inyectarlas.

Debido a este manejo el desarrollador no es el encargado de generar los objetos que utiliza sino que este framework, basándose en archivos xml o a través de anotaciones, se encarga de construir e instanciar todos los objetos que la aplicación utiliza y necesita.

En este proyecto se utilizó la inyección mediante anotaciones, donde utilizando *@Autowired* se inyecta el objeto para poder utilizarlo luego.

```
@Service
@Transactional(propagation = Propagation.REQUIRED)
public class MoteBean {

    private static final String TIME = "time";

    private static final SimpleDateFormat DATE_FORMATTER = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");

    private static final Logger LOGGER = LoggerFactory.getLogger(MoteBean.class);

    @Autowired
    private MoteDAO moteDAO;
```

FIGURA 46. INYECCIÓN DE DEPENDENCIAS

En la figura 46 se puede apreciar como con @Autowired se inyecta el objeto moteDAO, de forma tal que es posible utilizarlo a lo largo del código sin la necesidad de instanciarlo explícitamente.

Para que un objeto de una clase sea inyectable, se debe declarar esto específicamente en dicha clase. Por ejemplo, para indicar que se desea generar una instancia de una cierta clase, se utiliza @Service, y de esta forma Spring pasa a ser el encargado del "ciclo de vida" de estas instancias.

TRANSACCIONALIDAD

Spring brinda un marco para el manejo transaccional de operaciones y datos, de forma que el desarrollador se puede abstraer y mediante anotaciones, realizar la gestión de transacciones hacia la base de datos.

En la figura 46 se encuentra un ejemplo del manejo de la transaccionalidad, utilizando la anotación @Transactional. En particular se utilizó *propagation=REQUIRED*. Esto implica que, si al ejecutarse un método transaccional, ya existe una transacción abierta con la base de datos, este método utiliza dicha transacción. Mientras que si al ser invocado no existe una transacción creada previamente, se crea una transacción nueva y al finalizar la ejecución del método se intenta realizar el commit correspondiente.

AngularJS

AngularJS [47] es un framework MVC sobre el lenguaje JavaScript (open source) diseñado por Google. Fue construido y diseñado para el desarrollo web front-end, permitiendo crear aplicaciones SPA (Single-Page Application).

AngularJS está principalmente basado en directivas sobre las etiquetas HTML, que facilitan el trabajo a la hora de generar formularios web, así como también las recargas parciales de páginas web.

En consecuencia, gran parte de la carga en el backend se reduce, lo que conlleva a aplicaciones web más ligeras.

En este proyecto fue utilizado en el componente Sensors-Web para la construcción de la interfaz web de usuario.

ActiveMQ

ActiveMQ [48] es un broker de mensajería open source. Este implementa una API REST que lo hace independiente de cualquier lenguaje o plataforma por lo que puede ser utilizado en lenguajes Java, C, C#, Ruby, etc. Esto permite que desde cualquier dispositivo se pueda enviar o recibir mensajes a través de un llamado HTTP POST o GET.

También brinda grupos de mensajería, destinos virtuales y complejos y la óptima persistencia de los mensajes mediante la utilización de JDBC, garantizando la entrega de los mensajes que este recibe.

ActiveMQ es además un proveedor de Java Message Service (JMS) [49].

Su arquitectura se compone de los siguientes elementos:

- Proveedor JMS: sistema de mensajería que implementa interfaces JMS y proporciona herramientas administrativa.
- Cliente: aplicación Java que envía o recibe mensajes JMS. Se le llama productor al que envía y consumidor al que recibe el mensaje.
- Mensaje: elemento mediante el cual se trasmite la información entre productor y consumidor.

JMS brinda dos modelos de mensajería:

- **Punto a punto**
  Se basa en el concepto de colas de mensajes, emisores y receptores. Cada mensaje es enviado a una cola específica y los clientes receptores extraen los mensajes de su respectiva cola. Las colas almacenan los mensajes que reciben hasta que son extraídos o expiran. En este modelo los mensajes son extraídos sólo por un receptor.

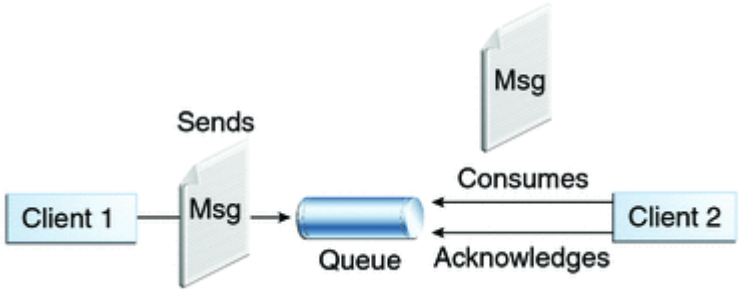

FIGURA 47. MENSAJERÍA PUNTO A PUNTO. EXTRAÍDA de [49]

- **Publicación / suscripción**
  En este modelo los clientes envían los mensajes a un tópico específico del proveedor JMS y este mensaje es recibido por todos los clientes suscriptos a dicho tópico. El sistema es el encargado de distribuir los mensajes y los almacena hasta que sean distribuidos a los suscriptores activos correspondientes.

FIGURA 48. MENSAJERÍA PUBLICACIÓN/SUSCRIPCIÓN. EXTRAÍDA de [49]

En este proyecto se utiliza el modelo de comunicación punto a punto entre Sensors-Daemon y Sensors-Core, utilizando el estándar JMS.

Bootstrap

Bootstrap [50] es un framework open source para el diseño de aplicaciones web. Garantiza con su uso el desarrollo de aplicaciones responsivas, es decir, aplicaciones que se adaptan al tamaño de pantalla del dispositivo utilizado para su visualización. Para esto provee hojas de estilos y archivos JavaScript que implementan componentes predefinidos, dejando también abierta la posibilidad de personalización de estos.

En este proyecto se utiliza en el componente Sensors-Web, para la construcción de la interfaz web de usuario.

Geo Server

Geo Server [43] es un servidor de mapas libre basado en Java que permite el almacenamiento y la gestión de datos geoespaciales. Brinda la funcionalidad de creación de mapas e intercambio de datos mediante el uso de estándares abiertos, establecidos por el Open Geospatial Consortium (OGC).

En particular, a través de los estándares WMS (Web Map Service) y WFS (Web Feature Service) es capaz de crear mapas y hacerlos accesibles públicamente para la edición de sus datos.

En este proyecto se utilizó la comunicación con Geo Server a través de una REST API brindada por el mismo, donde mediante invocaciones con su correspondiente verbo (GET, POST, PUT, DELETE) se realiza la consulta de capas ya existentes, el envío de capas nuevas para su almacenamiento y el borrado de las mismas.

Las capas son manejadas mediante archivos de formato “shapefile” (.shp), el cual representa un formato estándar para el almacenamiento de datos geoespaciales.

MongoDB

Mongo DB [51] es una base de datos NoSQL (no relacional) open source orientada a documentos. No utiliza tablas para almacenar datos (como lo hacen las bases relacionales) sino que guarda documentos en formato JSON dentro de colecciones.

Su uso es recomendado para el almacenamiento de gran cantidad de datos, ya que al no poseer esquemas optimiza las operaciones de lectura y escritura de los elementos, que en una base de datos relacional serían menos eficientes.

Esta base de datos se utiliza para el almacenamiento de medidas de los sensores, y de valores históricos, por ejemplo de tablas de ruteo y de vecinos de cada uno de los motes.

PostgreSQL

PostgreSQL [52] es un sistema de gestión de bases de datos relacional open source. Algunas de sus características de mayor importancia son: su alta concurrencia (permite que un proceso escriba en una tabla, mientras otros acceden a la misma sin necesidad de bloqueos), utiliza multiprocesos, soporte de two-phase commit [53], point in time recovery (PITR) [54] y copias de seguridad.

También permite el almacenamiento de datos espaciales mediante el módulo PostGIS [55]. En un principio se analizó la posibilidad de utilizar dicho módulo para el almacenamiento de datos geográficos (las capas agregadas al mapa de una red), pero finalmente se optó por utilizar el envío de archivos Shapefiles y su almacenamiento local por parte del Geo Server. De esta forma se implementa de manera más sencilla el agregado de capas.

En este proyecto se utiliza PostgreSQL para el almacenamiento de datos del negocio exceptuando las medidas y la información de históricos.

SQLite

SQLite [56] es un sistema de gestión de base de datos embebido y open source. No tiene un proceso servidor independiente y lee y escribe en archivos del “file system” de la máquina host. Es debido a esto que es una base de datos de pequeño tamaño y es recomendado su uso en terminales de recursos limitados.

Particularmente, esta base de datos es aplicada para el almacenamiento en el componente Sensors-Daemon, debido a que el mismo se ejecuta sobre un ordenador de placa reducida, de escasas capacidad y necesita almacenar datos de manera temporal.

OpenLayers 3

Open Layers [57] es una librería JavaScript open source, que permite la visualización de mapas en aplicaciones web. Es ejecutada del lado del cliente.

Es compatible con las distintas cartografías propietarias, como son entre otras Google y Yahoo! y cumple también con los estándares WMS y WFS. Permite la superposición de distintas capas, añadir marcadores, controles y polígonos en el mapa.

En este proyecto es utilizada en el componente Sensors-Web, para la visualización de la red de sensores en el mapa.

Zing Chart

Zing Chart [58] es una librería JavaScript utilizada para la visualización de gráficas del lado del cliente. Provee diversos tipos de gráficas, y brinda la posibilidad de combinarse junto con AngularJS mediante directivas.

En este proyecto es utilizada en el componente Sensors-Web, para la visualización de las medidas en forma de gráfica.

Free Marker

Apache Free Marker [59] permite realizar plantillas que luego generan salida de texto de acuerdo a determinados valores indicados. Con él es posible generar páginas HTML, e-mails, archivos de configuración y hasta código fuente.

Para la implementación de plantillas se utiliza un lenguaje especializado propietario, FreeMarker Template Language (FTL), que se emplea para visualizar los datos. Fuera de la planilla se especifica que datos deben desplegarse.

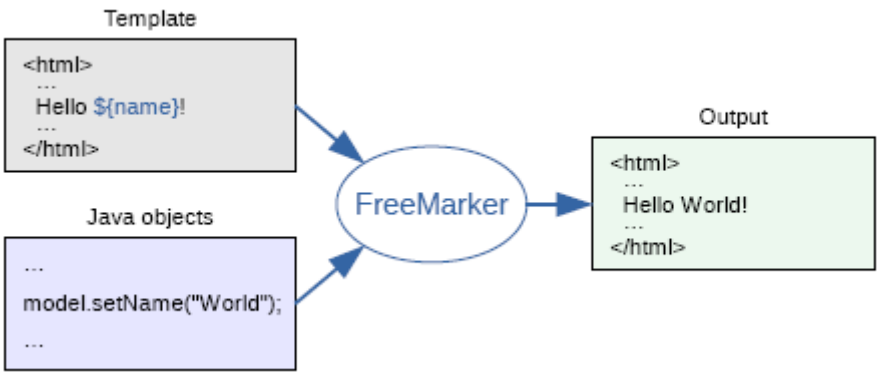

FIGURA 49. GENERADOR DE TEXTO FREEMARKER.

En este proyecto FreeMaker es utilizado en el componente Sensors-Core, para el envío de mails y SMS con formato HTML.

Californium

Californium [60] provee una librería para el uso del protocolo CoAP que es de sencillo uso en proyectos Java. Con esta librería es posible crear tanto clientes como servidores CoAP y realizar el manejo de sus recursos.

Esta librería es utilizada en el Sensors-Daemon para implementar la comunicación entre el componente y la red de sensores.

A continuación se mencionan las funcionalidades utilizadas de Californium.

Cliente CoAP

Antes de realizar una solicitud a cualquier URI de recursos, es necesaria la creación de un cliente CoAP que haga posible esta interacción.

Para ello es necesario indicar la URI del recurso sobre el cual se va a realizar una solicitud.

```
CoapClient clienteRecurso = new CoapClient(coapUrl);
```

FIGURA 50. CREACIÓN CLIENTE COAP

Discovery recursos

Luego de creado un CoapClient es posible realizar el descubrimiento de los recursos que se encuentran en un mote.

```
Set<WebLink> discovery = clienteRecurso.discover();

for (WebLink recurso : discovery) {
    String uriRecurso = recurso.getURI();
}
```

FIGURA 51. DISCOVERY DE RECURSOS

Get de un recurso

Es posible también realizar pedidos GET de recursos.

```
CoapResponse responseGetResource = clienteRecurso.get();
String responseText = responseGetResource.getResponseText();
```

FIGURA 52. GET RECURSO URI

Como se muestra en figura 52 se brinda la funcionalidad de recuperar el mensaje que viaja en el payload en respuesta a la solicitud GET del recurso mediante "getResponseText()".

Observar recurso

Esta funcionalidad permite observar un recurso utilizando el método "observeAndWait". Para esto, a dicho método se le debe indicar como parámetro un manejador CoapHandler, y es dentro de esta instancia de CoapHandler que se debe implementar el código que se ejecuta cada vez que un mensaje es recibido. Esta implementación debe estar contenida dentro del método "onLoad", y a su vez si ocurre un error se invoca el método "onError".

```
CoapObserveRelation obs = clienteRecurso.observeAndWait(new CoapHandler() {
    public void onLoad(CoapResponse response) {

    }
    public void onError() {
    }
});
```

FIGURA 53. OBSERVAR RECURSO URI