Arquitecturas de la web de las cosas (WoT) para la interoperabilidad en entornos inteligentes

Universidad de Oviedo

Daniel Ibaseta Rodríguez

Director: Julio Molleda Meré

Tesis doctoral
Programa de Doctorado en Informática

Septiembre 2022

Universidad de Oviedo

RESUMEN DEL CONTENIDO DE TESIS DOCTORAL

1.- Título de la Tesis	
Español: Arquitecturas de la web de las cosas (WoT) para la interoperabilidad en entornos inteligentes	Inglés: Web of Things (WoT) architectures for interoperability in smart environments

2.- Autor	
Nombre: Daniel Ibaseta Rodríguez	
Programa de Doctorado: Informática	
Órgano responsable: Centro Internacional de Postgrado	

RESUMEN (en español)

La creciente demanda de entornos inteligentes capaces tomar decisiones de forma autónoma en función de las condiciones muestreadas del ambiente, especialmente en el campo de la eficiencia energética, hace que la interconexión de dispositivos heterogéneos, tales como sensores, actuadores y sistemas de computación, concentre un elevado interés desde el punto de vista de la investigación, el desarrollo y la innovación.

Los avances recientes en tecnologías informáticas, electrónicas y de comunicaciones brindan la oportunidad de construir edificios inteligentes, o modernizar los existentes, dotándolos de inteligencia. El Internet de las Cosas, o Internet of Things (IoT), es un paradigma que combina sistemas empotrados y redes de comunicación inalámbricas de muy bajo consumo con el objetivo de proporcionar conexión a Internet a objetos físicos cotidianos. IoT resulta crucial en el análisis y la optimización del rendimiento energético de edificios. Sin embargo, hay varios aspectos que limitan la utilización de soluciones IoT en este ámbito, como la falta de estándares que garanticen la interoperabilidad entre los diferentes tipos de sistemas y dispositivos. Incluso cuando se utilizan estándares IoT, el amplio abanico de protocolos disponible hace que lograr su interoperabilidad en toda la solución resulte extremadamente complicado y costoso.

Por ello, en esta tesis doctoral se propone y se evalúa una arquitectura basada en el paradigma Web de las Cosas, o Web of Things (WoT), que extiende la interconexión de dispositivos heterogéneos que se logra mediante IoT proporcionando interoperabilidad a nivel de la capa de aplicación. Los propósitos concretos son, en primer lugar, investigar el concepto de interoperabilidad de dispositivos físicos heterogéneos a través de Internet, con especial hincapié en las recomendaciones y estándares propuestas en este campo. Para lograr este primer propósito se ha llevado a cabo una revisión exhaustiva del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con IoT y con WoT. Y, en segundo lugar, evaluar la viabilidad de una arquitectura para la interconexión de dispositivos heterogéneos que sirva de soporte para sistemas de gestión de energía de edificios, implementada en base a estándares Web y las recomendaciones del World Wide Web Consortium (W3C). La arquitectura propuesta en esta tesis doctoral se despliega en dos casos de uso: uno en un edificio de oficinas situado en Gijón y otro en un edificio de viviendas situado en Bagnolo.

Si bien la solución basada en recomendaciones del W3C para arquitecturas WoT añade más latencia que una basada en el uso directo de protocolos más simples de capas inferiores, o protocolos IoT, aporta como principal beneficio la interoperabilidad, independientemente del dispositivo y de su fabricante, y proporciona metadatos que enriquecerán las aplicaciones finales de usuario.

Esta tesis doctoral se enmarca en un proyecto de investigación industrial en CTIC Centro Tecnológico, que se define a partir de un subconjunto de tareas relacionadas con la interoperabilidad de dispositivos heterogéneos dentro del marco de un proyecto de

investigación europeo centrado en la rehabilitación de edificios bajo criterios de eficiencia energética.

Las contribuciones de este trabajo propician varias líneas de investigación, relacionadas con seguridad, redes y analítica de datos, que podrían ser abordadas en trabajos posteriores para ampliar los objetivos iniciales de esta tesis.

RESUMEN (en inglés)

The growing demand for intelligent environments capable of making decisions autonomously based on the conditions sampled from the environment, especially in the field of energy efficiency, makes really interesting, from the point of view of research, development, and innovation, the interconnection of heterogeneous devices such as sensors, actuators, and computing systems.

Recent advances in computing, electronic and communication technologies offer the opportunity to build smart buildings or modernize existing ones. The Internet of Things (IoT) is a paradigm that combines embedded systems and very low-power wireless communication networks with the aim of providing Internet connection to everyday physical objects. IoT is crucial in analysing and optimizing the energy performance of buildings. However, there are several aspects that limit the use of IoT solutions in this field, such as the lack of standards that guarantee interoperability between different types of systems and devices. Even when IoT standards are used, the wide range of protocols available makes achieving interoperability extremely difficult and costly.

For this reason, this doctoral thesis proposes and evaluates an architecture based on the Web of Things (WoT) paradigm. WoT extends the interconnection of heterogeneous devices achieved through IoT, providing interoperability through the application layer. This thesis defines two specific purposes. Firstly, investigating the concept of interoperability of heterogeneous physical devices through the Internet, with special emphasis on the recommendations and standards proposed in this field. To achieve this purpose, a comprehensive review has been carried out about the state of the art of standards, recommendations, architectures, and technologies related to the Internet of Things and the Web of Things. Secondly, evaluating the feasibility of an architecture for the interconnection of heterogeneous devices that serves as support for building energy management systems implemented based on Web standards and the recommendations of the World Wide Web Consortium (W3C). The architecture proposed in this this doctoral thesis is deployed in two use cases: one in an office building located in Gijón and another one in a residential building located in Bagnolo.

Although the solution based on W3C recommendations for WoT architectures adds more latency than one based on the use of simpler lower layer protocols, or IoT protocols, the main benefit is interoperability, which is agnostic to the device and its manufacturer, and provides metadata that will enrich end user applications.

This doctoral thesis is part of an industrial research project at CTIC Technological Centre, which is defined from a subset of tasks related to the interoperability of heterogeneous devices within the framework of a European research project focused on the retrofit of buildings under energy efficiency criteria.

The contributions of this research work favour several lines of research, related to security, networks and data analytics, which could be addressed in later works to expand the initial objectives of this thesis.

En Gijón, a fecha de la firma digital.

Fdo.: Daniel Ibaseta Rodríguez

Daniel Ibaseta Rodríguez

Firmado digitalmente por Daniel Ibaseta Rodríguez
Fecha: 2022.09.22 10:45:05 +02'00'

SR. PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO EN INFORMÁTICA

A mis padres, sin los cuales nada de esto sería posible.
A Julio, por su implicación y apoyo constantes.
A CTIC, por el espacio y facilidades aportadas.

Podemos juzgar nuestro progreso por el coraje de nuestras preguntas y la profundidad de nuestras respuestas, nuestra voluntad de aceptar lo que es cierto por encima de lo que es deseable.

Carl Sagan, Cosmos

Comenzar es la mitad del trabajo, comienza nuevamente con la mitad restante, y habrás terminado.

Marco Aurelio, Meditaciones

La vida no es principalmente una búsqueda del placer, como creía Freud, ni una búsqueda de poder, como lo enseñó Alfred Adler, sino una búsqueda de sentido. La mejor tarea para cualquier persona es encontrarle sentido a su propia vida.

Victor Frank, El Hombre en Busca de Sentido

Resumen

La creciente demanda de entornos inteligentes capaces tomar decisiones de forma autónoma en función de las condiciones muestreadas del ambiente, especialmente en el campo de la eficiencia energética, hace que la interconexión de dispositivos heterogéneos, tales como sensores, actuadores y sistemas de computación, concentre un elevado interés desde el punto de vista de la investigación, el desarrollo y la innovación.

Los avances recientes en tecnologías informáticas, electrónicas y de comunicaciones brindan la oportunidad de construir edificios inteligentes, o modernizar los existentes, dotándolos de inteligencia. El Internet de las Cosas, o *Internet of Things* (IoT), es un paradigma que combina sistemas empotrados y redes de comunicación inalámbricas de muy bajo consumo con el objetivo de proporcionar conexión a Internet a objetos físicos cotidianos. IoT resulta crucial en el análisis y la optimización del rendimiento energético de edificios. Sin embargo, hay varios aspectos que limitan la utilización de soluciones IoT en este ámbito, como la falta de estándares que garanticen la interoperabilidad entre los diferentes tipos de sistemas y dispositivos. Incluso cuando se utilizan estándares IoT, el amplio abanico de protocolos disponible hace que lograr su interoperabilidad en toda la solución resulte extremadamente complicado y costoso.

Por ello, en esta tesis doctoral se propone y se evalúa una arquitectura basada en el paradigma Web de las Cosas, o *Web of Things* (WoT), que extiende la interconexión de dispositivos heterogéneos que se logra mediante IoT proporcionando interoperabilidad a nivel de la capa de aplicación. Los propósitos concretos son, en primer lugar, investigar el concepto de interoperabilidad de dispositivos físicos heterogéneos a través de Internet, con especial hincapié en las recomendaciones y estándares propuestas en este campo. Para lograr este primer propósito se ha llevado a cabo una revisión exhaustiva del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con IoT y con WoT. Y, en segundo lugar, evaluar la viabilidad de una arquitectura para la interconexión de dispositivos heterogéneos que sirva de soporte para sistemas de gestión de energía de edificios, implementada en base a estándares Web y a las recomendaciones del *World Wide Web Consortium* (W3C). La arquitectura propuesta en esta tesis doctoral se despliega en dos casos de uso: uno en un edificio de oficinas situado en Gijón y otro en un edificio de viviendas situado en Bagnolo.

Si bien la solución basada en recomendaciones del W3C para arquitecturas WoT añade más latencia que una basada en el uso directo de protocolos más simples de capas inferiores, o protocolos IoT, aporta como principal beneficio la interoperabilidad, independientemente del dispositivo y de su fabricante, y proporciona metadatos que enriquecerán las aplicaciones finales de usuario.

Esta tesis doctoral se enmarca en un proyecto de investigación industrial en CTIC Centro Tecnológico, que se define a partir de un subconjunto de tareas relacionadas con la interoperabilidad de dispositivos heterogéneos dentro del marco de un proyecto de investigación europeo centrado en la rehabilitación de edificios bajo criterios de eficiencia energética.

Las contribuciones de este trabajo propician varias líneas de investigación, relacionadas con seguridad, redes y analítica de datos, que podrían ser abordadas en trabajos posteriores para ampliar los objetivos iniciales de esta tesis.

Abstract

The growing demand for intelligent environments capable of making decisions autonomously based on the conditions sampled from the environment, especially in the field of energy efficiency, makes really interesting, from the point of view of research, development, and innovation, the interconnection of heterogeneous devices such as sensors, actuators, and computing systems.

Recent advances in computing, electronic and communication technologies offer the opportunity to build smart buildings or modernize existing ones. The Internet of Things (IoT) is a paradigm that combines embedded systems and very low-power wireless communication networks with the aim of providing Internet connection to everyday physical objects. IoT is crucial in analysing and optimizing the energy performance of buildings. However, there are several aspects that limit the use of IoT solutions in this field, such as the lack of standards that guarantee interoperability between different types of systems and devices. Even when IoT standards are used, the wide range of protocols available makes achieving interoperability extremely difficult and costly.

For this reason, this doctoral thesis proposes and evaluates an architecture based on the Web of Things (WoT) paradigm. WoT extends the interconnection of heterogeneous devices achieved through IoT, providing interoperability through the application layer. This thesis defines two specific purposes. Firstly, investigating the concept of interoperability of heterogeneous physical devices through the Internet, with special emphasis on the recommendations and standards proposed in this field. To achieve this purpose, a comprehensive review has been carried out about the state of the art of standards, recommendations, architectures, and technologies related to the Internet of Things and the Web of Things. Secondly, evaluating the feasibility of an architecture for the interconnection of heterogeneous devices that serves as support for building energy management systems implemented based on Web standards and the recommendations of the World Wide Web Consortium (W3C). The architecture proposed in this this doctoral thesis is deployed in two use cases: one in an office building located in Gijón and another one in a residential building located in Bagnolo.

Although the solution based on W3C recommendations for WoT architectures adds more latency than one based on the use of simpler lower layer protocols, or IoT protocols, the main benefit is interoperability, which is agnostic to the device and its manufacturer, and provides metadata that will enrich end user applications.

This doctoral thesis is part of an industrial research project at CTIC Technological Centre, which is defined from a subset of tasks related to the interoperability of heterogeneous devices within the framework of a European research project focused on the retrofit of buildings under energy efficiency criteria.

The contributions of this research work favour several lines of research, related to security, networks and data analytics, which could be addressed in later works to expand the initial objectives of this thesis.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

API	*Application Programming Interface.*
BEMS	*Building Energy Management System.*
BGC	*Building Gateway and Controller.*
BMS	*Building Management System.*
BPM	*Business Process Manager.*
CoAP	*Constrained Application Protocol.*
CRUD	*Create, Read, Update, Delete.*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance.*
DHW	*Domestic Hot Water.*
DSM	*Demand-Side Management.*
DTLS	*Datagram Transport Layer Security.*
ETSI	*European Telecommunications Standards Institute.*
FC	*Fan Coil.*
FCAPS	*Fault, Configuration, Accounting, Performance, Security.*
FEDER	Fondo Europeo de Desarrollo Regional.
HDFS	*Hadoop Distributed File System.*
HEART	*Holistic Energy and Architectural Retrofit Toolkit.*
HEMS	*Home Energy Management System.*
HP	*Heat Pump.*
HTML	*HyperText Markup Language.*
HTTP	*HyperText Transfer Protocol.*
IEEE	*Institute of Electrical and Electronic Engineers.*

IEC *International Electrotechnical Commission.*

IoT *Internet of Things.*

IP *Internet Protocol.*

ISO *International Standarization Organization.*

ITU *International Telecommunication Union.*

JSON *JavaScript Object Notation.*

LAN *Local Area Network.*

LLN *Low-Power Lossy Network.*

LPWAN *Low Power Wide Area Network.*

M2M *Machine to Machine.*

MIMO *Multiple Input, Multiple Output.*

MQTT *Message Queuing Telemetry Transport.*

NAT *Network Address Translation.*

OGC *Open Geospatial Consortium.*

OCF *Open Connectivity Foundation.*

OMA *Open Mobile Alliance.*

OTA *Over The Air.*

PCM *Phase-Change Material.*

PLC *Programmable Logic Controller.*

RDF *Resource Description Framework.*

REST *REpresentational State Transfer.*

RIS3 Estrategia Regional de Investigación e Innovación para una Especialización Inteligente.

RPL *Routing Protocol for Low-Power and Lossy Networks.*

RSSI *Received Signal Strength Indicator.*

SBC *Single-Board Computer.*

SBCT *Smart Building Controller.*

SDC *Standards Developing Organizations.*

SMS *Short Message Service.*

SoC *System on Chip.*

SSID *Service Set Identifier.*

TCP *Transmission Control Protocol.*

TD *Thing Description.*

TSDB *Time Series Database.*

UDP *User Datagram Protocol.*

URI *Uniform Resource Identifier.*

VPN *Virtual Private Network.*

W3C *World Wide Web Consortium.*

WMN *Wireless Mesh Network.*

WPAN *Wireless Personal Area Network.*

WoT *Web of Things.*

XML *eXtensible Markup Language.*

Capítulo 1

Introducción

La creciente demanda de entornos inteligentes capaces de tomar decisiones de forma autónoma en función de las condiciones muestreadas del ambiente, especialmente en el campo de la eficiencia energética, hace que la interconexión de dispositivos heterogéneos tales como sensores, actuadores y sistemas de computación concentre un elevado interés desde el punto de vista de la investigación, el desarrollo y la innovación [1, 2].

Los edificios destinados a viviendas suponen una demanda de casi el 40 % de la energía consumida en los países desarrollados, siendo, además, la mayor fuente de emisión de gases de efecto invernadero en los entornos urbanos [3, 4]. En la Unión Europea, se estima que el 50 % de la energía consumida se debe a sistemas de calefacción y de refrigeración, y de este, entre el 50 % y el 80 % se consume en edificios. Por ello, tanto gobiernos como la comunidad científica son conscientes de la necesidad crucial de rehabilitar y modernizar el parque de edificios existentes actualmente con el objetivo de reducir el consumo de energía y, de esta forma, reducir también las emisiones [5, 6]. Estos edificios tienen un potencial enorme en cuanto a reducción de consumo energético, con lo que su modernización en términos de energía, o *energy retrofit*, [7] tiene un rol importante en la optimización de su rendimiento energético [8, 9]. Además de los efectos beneficiosos en cuanto a la reducción de emisiones, la modernización de edificios contribuye al desarrollo de ciudades inteligentes, o *smart cities*, [10, 11] al tiempo que resulta esencial para abordar el problema de la pobreza energética [12].

A nivel regulatorio, la Directiva (UE) 2018/844[1], por la que se modifica la Directiva 2010/31/UE relativa a la eficiencia energética de los edificios y la Directiva 2012/27/UE relativa a la eficiencia energética, establece un acceso igualitario a la financiación para la renovación de edificios, de tal forma que los edificios existentes puedan ser rehabilitados para alcanzar estándares de consumo casi nulo de energía. Para ello, esta Directiva adopta medidas que posibilitan la renovación de edificios en todas las condiciones, incluso en las situaciones más desfavorecidas, tales como edificios con muy bajo rendimiento energético, consumidores que sufren pobreza energética o viviendas sociales, entre otras.

[1] https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32018L0844

En España hay varias vías de financiación dentro del Plan de Recuperación, Transformación y Resiliencia financiado por la Unión Europea a través de *NextGenerationEU* para la rehabilitación energética de edificios, como por ejemplo el Programa PREE[2], dotado con 300 millones de euros, como impulso a la rehabilitación energética y a la disminución del consumo de energía final y de las emisiones de CO_2 en el parque de edificios, y el Programa PREE 5000[3], dotado con 50 millones, para actuaciones de rehabilitación energética en edificios en municipios de reto demográfico, ambos del Ministerio para la Transición Ecológica y el Reto Demográfico.

Los avances recientes en tecnologías informáticas, electrónicas y de comunicaciones brindan la oportunidad de construir edificios inteligentes o modernizar los existentes dotándolos de inteligencia [13–15]. Para ello, lo habitual es que un sistema de gestión del edificio, o *Building Management System* (BMS), se encargue de la monitorización y el control de su equipamiento eléctrico y mecánico. Estos avances permiten reducir aún más el consumo de energía en edificios. Durante la fase de construcción, o en las labores de rehabilitación, se pueden desplegar redes inalámbricas a las que conectar sensores, actuadores y controladores que conformen un sistema de gestión de energía, o *Building Energy Management System* (BEMS) [16–19], que involucra a los componentes del BMS relacionados con la gestión de energía. De forma general, un BEMS se compone de varios subsistemas hardware y software heterogéneos tanto en la función como en la implementación. El BEMS se encarga de monitorizar parámetros del entorno, permitiendo evaluar en tiempo real el consumo de energía del edificio, así como definir estrategias de alto nivel para minimizarlo. Además, el conocimiento inferido del entorno mediante técnicas de ciencia e ingeniería de datos [20] permite conocer las condiciones del edificio, de tal forma que facilita a los expertos tanto el desarrollo de estrategias para lograr un consumo eficiente como la definición de planes de actuación para una futura rehabilitación [21, 22].

El Internet de las Cosas, o *Internet of Things* (IoT), resulta crucial en el análisis y la optimización del rendimiento energético de edificios [23], dado que ha conseguido aportar soluciones de coste reducido y no intrusivas en la rehabilitación de edificios existentes. Sin embargo, hay varios aspectos que limitan la utilización de soluciones IoT en este ámbito. En primer lugar, la falta de estándares que garanticen la interoperabilidad entre los diferentes tipos de sistemas y dispositivos que se requieren para la implementación de estas soluciones [24]. Además, los edificios antiguos imponen un elevado número de barreras arquitectónicas, con lo que el despliegue de soluciones IoT en estos escenarios supone un reto tecnológico [25]. Otro reto es la distribución de sensores de tal forma que se cubran áreas relevantes y con distintas características físicas en el edificio, para lo que se suelen utilizar algoritmos de optimización para determinar su ubicación [26, 27]. Los sensores, así como actuadores, controladores y

[2]https://www.idae.es/ayudas-y-financiacion/para-la-rehabilitacion-de-edificios/convocatorias-cerradas/programa-pree

[3]https://www.idae.es/ayudas-y-financiacion/para-la-rehabilitacion-de-edificios/programa-pree-5000-rehabilitacion

el resto de los dispositivos del sistema, pueden ser de fabricantes diferentes, cada uno con protocolos de comunicación específicos y no interoperables directamente entre ellos. Incluso cuando los dispositivos utilicen estándares IoT, el amplio abanico de protocolos disponible hace que lograr su interoperabilidad en todo el sistema resulte extremadamente complicado y costoso.

1.1. Motivación

La Web de las Cosas, *Web of Things* (WoT), es un paradigma que extiende la interconexión de dispositivos heterogéneos que se logra mediante IoT, proporcionando interoperabilidad a nivel de la capa de aplicación. El crecimiento exponencial experimentado en la última década en el número de dispositivos que se conectan a Internet ha acelerado la fragmentación en el ámbito IoT y, por tanto, la necesidad de proporcionar soluciones interoperables. El término WoT apareció a principios de los años 2000 con la idea de reutilizar las tecnologías y estándares Web propuestos por el *World Wide Web Consortium* (W3C) para interconectar cualquier tipo de dispositivo empotrado.

La Fundación Centro Tecnológico de la Información y la Comunicación[4] (CTIC) es una institución privada y sin ánimo de lucro que está configurada por empresas y organismos públicos y privados del ámbito de las Tecnologías de la Información y la Comunicación, así como por el Gobierno del Principado de Asturias. Es, además, la Oficina Española del W3C desde 2003, y la Oficina de Latinoamérica del W3C desde 2019. Esta fundación tiene por objeto ser un agente de innovación del Principado de Asturias, y tiene como misión principal promover la adopción de las recomendaciones del W3C entre los desarrolladores, creadores de aplicaciones y la comunidad Web en general, propiciando a su vez la inclusión de organizaciones que apuesten por la creación de futuras recomendaciones uniéndose al W3C. De acuerdo con sus Estatutos, CTIC Centro Tecnológico tiene como fines fundacionales: la contribución al beneficio general de la sociedad y a la mejora de la competitividad de las empresas mediante la generación de conocimiento tecnológico TIC, realizando actividades de I+D+i y desarrollando su aplicación; y la promoción y desarrollo de la sociedad de la información en general, y de las tecnologías de la información y de la comunicación en particular.

CTIC Centro Tecnológico se divide en dos áreas de negocio: un área de I+D+i y un área de promoción y desarrollo de la sociedad de la información. El área específica de I+D+i está compuesta por diferentes grupos de especialización: *Advanced Analytics & AI*, *Emerging Technologies*, *Inmersive Technologies*, *Web of Things* e *Internet of Things*, y *Blockchain*. El grupo de especialización sobre WoT e IoT ha participado recientemente en proyectos de investigación y desarrollo relacionados con la adquisición, filtrado, procesamiento y analítica de datos adquiridos mediante redes de sensores.

[4]https://www.fundacionctic.org

En los años 2016 y 2017, el grupo participó en el proyecto *iWhere*[5], financiado por la Unión Europea, mediante FEDER, a través del Plan de Ciencia, Tecnología e Innovación 2013-2017 (ref.: IDI/2016/000176). Las labores de investigación desarrolladas en este proyecto se centraron en el diseño de algoritmos avanzados de localización en interiores basados en intensidad de señal. El objetivo principal del proyecto fue tratar de superar las limitaciones existentes en las soluciones disponibles en aquel entonces y que eran válidas exclusivamente para geolocalizaciones en exteriores. El proyecto planteó una solución basada en el indicador de fuerza de la señal recibida, o *Received Signal Strength Indicator* (RSSI), a partir de nodos fijos con posiciones conocidas en el entorno. La dificultad de la solución radicaba en localización de trabajadores en instalaciones industriales con un elevado número de elementos metálicos, con lo que debían proporcionarse métodos para evitar posibles errores inducidos por interferencias o por pérdida de calidad de las señales.

También en los años 2016 y 2017, el grupo participó en el proyecto *Smart Air Quality*[6], financiado por la Unión Europea, mediante FEDER, a través del Plan de Ciencia, Tecnología e Innovación 2013-2017 (ref.: IDI/2016/000169). Este proyecto abordó la problemática de la gestión de la calidad del aire en interiores en tiempo real mediante el diseño y posterior desarrollo de un sistema IoT y WoT. El sistema se desplegó en un caso de ejemplo en un hospital, y para ello se instalaron sensores en los carros de limpieza, que seguían recorridos predefinidos. Los datos adquiridos por esta red de sensores móviles sirvieron de soporte para la toma de decisiones en tiempo real sobre los sistemas de control del aire del edificio.

Posteriormente, en los años 2018 y 2019 el grupo desarrolló el proyecto SCOUT (Sistema de monitorizaCión de fauna no intrusivO basado en visión por compUTador)[7], financiado por el Gobierno del Principado de Asturias a través de la convocatoria Innova-IDEPA (Programa RIS3-Empresa) para el desarrollo de proyectos de I+D+i, y desarrollado en colaboración con Biosfera Consultoría Medioambiental, S.L. y Táctica TIC. El objetivo general de este proyecto fue la construcción de un sistema de monitorización no intrusivo para la vigilancia de fauna en espacios abiertos, que se alcanzó mediante el diseño y desarrollo de un dispositivo que aprovechaba las capacidades ofrecidas por las nuevas tecnologías de sensórica y comunicación, junto con técnicas de visión artificial y aprendizaje automático, para la captura de imágenes relevantes en campo que facilitaron el seguimiento y el control de la fauna en un área determinada. Dichas imágenes se enviaban a una plataforma central que combinaba los datos de distintas zonas vigiladas y permitía monitorizar todos los dispositivos desplegados, alertando a los operarios de cualquier irregularidad, tanto en el entorno natural como en los dispositivos de detección de fauna.

[5]https://www.fundacionctic.org/es/proyectos/iwhere
[6]https://www.fundacionctic.org/es/proyectos/smart-air-quality
[7]https://www.fundacionctic.org/es/proyectos/scout-sistema-de-monitorizacion-de-fauna-no-intrusivo-basado-en-vision-por-computador

En un período similar, entre los años 2018 y 2020, este grupo de especialización participó en el proyecto WoRMS (*Web of Things Reference-implementation for a Microservices-based Servient*)[8], proyecto financiado por la Unión Europea, a través del FEDER, y por el Principado de Asturias, a través del Plan de Ciencia, Tecnología e Innovación 2018-2022 (ref.: IDI/2018/000101). El objetivo principal de este proyecto consistió en investigar un conjunto de tecnologías, componentes y herramientas software de código abierto para diseñar y desarrollar una implementación de referencia que facilitase a la industria el desarrollo de sistemas y aplicaciones compatibles con los estándares WoT definidos por el W3C [28].

Relacionada con tecnologías Web y estándares promovidos por el W3C, el grupo está desarrollando actualmente el proyecto SLOW (Oportunidades en la misión regional de reducción de emisiones y gases de efecto invernadero habilitada por la investigación en tecnologías W3C)[9]. Este proyecto, financiado por la Unión Europea, a través del FEDER, y por el Principado de Asturias, a través del Plan de Ciencia, Tecnología e Innovación 2021-2023 (ref.: IDI/2021/000306), planeta una metodología analítica de estimación (emisión y absorción) de gases de efecto invernadero mediante la investigación y aplicación de tecnologías avanzadas en un entorno hipersensorizado bajo estándares del W3C.

Esta amplia experiencia en investigación en sistemas IoT y WoT, así como la problemática generada por la heterogeneidad de los dispositivos físicos que se conectan a Internet, motivan la realización de esta tesis doctoral.

1.2. Ámbito y alcance

Esta tesis doctoral se enmarca en un proyecto de investigación industrial en CTIC Centro Tecnológico, que se define a partir de un subconjunto de tareas relacionadas con la interoperabilidad de dispositivos heterogéneos dentro del marco de un proyecto de investigación europeo centrado en la rehabilitación de edificios bajo criterios de eficiencia energética.

1.2.1. Proyecto HEART

El proyecto HEART (*Holistic Energy and Architectural Retrofit Toolkit – The Sum of All Things*) [29, 30] fue un proyecto financiado a través del programa *Horizon 2020* de la Unión Europea (*Grant agreement ID 768921*), con un importe de 6.638.687,50 €, a través de H2020-EU.2.1.5 *Industrial Leadership - Leadership in enabling and industrial technologies - Advanced manufacturing and processing*[10], y más concretamente de H2020-EU.2.1.5.2 *Technologies enabling energy-efficient systems and energy-efficient buildings with a low environmental*

[8]https://www.fundacionctic.org/es/proyectos/worms
[9]https://www.fundacionctic.org/es/proyectos/slow-oportunidades-en-la-mision-regional-de-reduccion-de-emisiones-y-gases-de-efecto
[10]https://cordis.europa.eu/programme/id/H2020-EU.2.1.5.

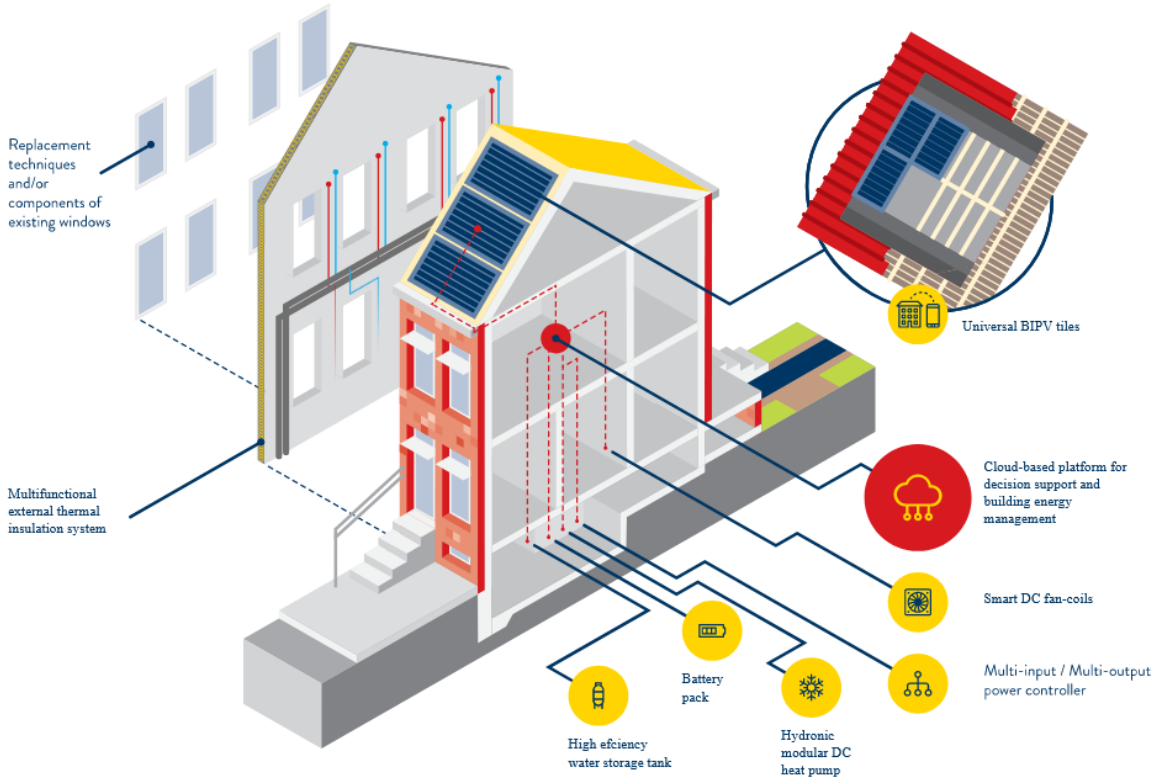


Figura 1.1 Visión general del proyecto HEART [30].

impact[11]. Con una planificación inicial de cuatro años, el desarrollo del proyecto se extendió, con una prórroga incluida, desde el 1 de octubre de 2017 al 31 de julio de 2022. Involucró a 16 socios de 10 países y estuvo coordinado desde el Politécnico de Milán por el Prof. Niccolò Aste y por el Dr. Claudio Del Pero. La idea central del proyecto era mejorar la eficiencia energética de edificios residenciales existentes actualmente en la Unión Europea mediante su conversión en edificios inteligentes, *smart buildings*, que controlasen de forma autónoma, entre otros, sus sistemas de calefacción, aire acondicionado y agua caliente. El núcleo del proyecto HEART se concibió como una plataforma informática basada en la nube, tal y como se muestra en la Figura 1.1, desarrollada para abordar y respaldar la toma de decisiones relacionadas con la gestión de la energía en edificios.

La información y el conocimiento inferido por esta plataforma servirá para desarrollar un modelo de consumo energético predictivo y adaptativo, tal y como se muestra en el esquema de la Figura 1.2.

Los objetivos estratégicos del proyecto HEART incluyen:

- Simplificar el proceso de renovación de edificios residenciales, maximizando su eficiencia energética.

[11] https://cordis.europa.eu/programme/id/H2020-EU.2.1.5.2.

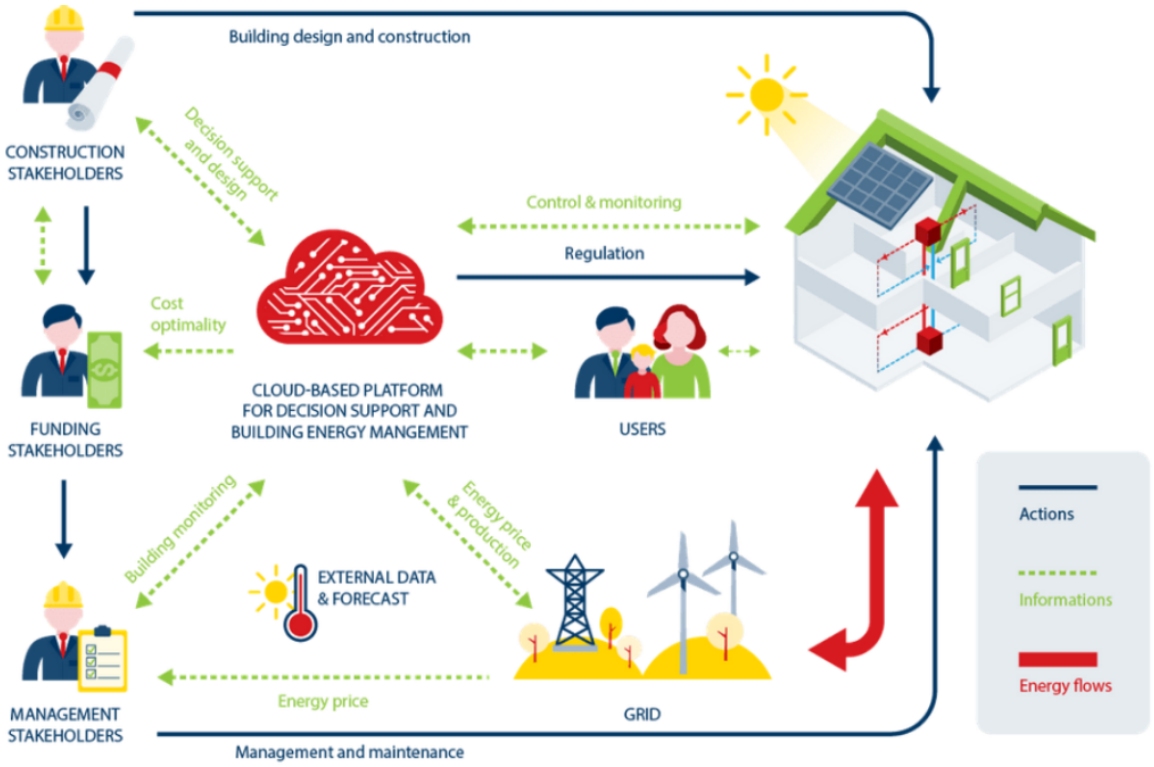


Figura 1.2 Modelo energético predictivo y adaptativo del proyecto HEART [30].

- Colaborar con todos los entes que participan en el proceso de renovación energética de edificios residenciales.
- Reducir el consumo de energía de edificios residenciales, introduciendo, entre otras soluciones, energías renovables.
- Reducir los costes de operación y la factura energética de los residentes, a la vez que se incrementa el bienestar térmico en los hogares.
- Extrapolar los conceptos del proyecto a edificios residenciales de nueva construcción además de a los edificios existentes.

Uno de los socios de este proyecto fue CTIC Centro Tecnológico, que recibió una aportación de 243.125 € para desarrollar las tareas asignadas en dicho proyecto.

1.2.2. Proyecto Web of Things para entornos inteligentes y energéticamente eficientes

Para el desarrollo de las tareas relacionadas con el estudio y la implementación de una arquitectura de interconexión de dispositivos heterogéneos que diera soporte al proyecto HEART, CTIC definió un proyecto de investigación industrial denominado "*Web of Things*

para entornos inteligentes y energéticamente eficientes", que constituye el marco en el que se desarrolla esta tesis doctoral. El objetivo principal de este proyecto es la integración de diferentes componentes hardware y software en edificios existentes de tal forma que permitan su conversión en edificios inteligentes y energéticamente eficientes. Las principales tareas de este proyecto están encaminadas a analizar la viabilidad de utilizar WoT como paradigma para la construcción de entornos inteligentes, integrando diferentes subsistemas heterogéneos de climatización y refrigeración en edificios residenciales.

La dirección de este proyecto de investigación industrial ha recaído en D. Fidel Díez Díaz, Director de I+D en CTIC, y Dª. Blanca Rosario Campomanes Álvarez, *R&D researcher* y *senior data scientist* en CTIC, y el equipo de desarrollo está formado por D. Daniel Ibaseta Rodríguez.

1.3. Metodología de investigación

En general, todo proceso de investigación trata de resolver cinco aspectos fundamentales: identificar el objetivo de la investigación, comprender las teorías relevantes para el proceso, proponer las preguntas de investigación a responder, definir el método para recopilar los datos necesarios y proponer el método para muestrear estos datos [31].

Dado que esta tesis se desarrolla en el marco de un proyecto de investigación industrial que pretende resolver un problema práctico, la metodología que se empleará para guiar el proceso de investigación, así como para alcanzar los objetivos que se plantean en la tesis, se basa en un enfoque de investigación constructivo [32], del que se puede observar un esquema en la Figura 1.3. El principal objetivo de esta metodología es solucionar problemas prácticos a la vez que se amplía el conocimiento en el área de estudio mediante contribuciones tales como métodos, procesos, prácticas o herramientas, entre otras.

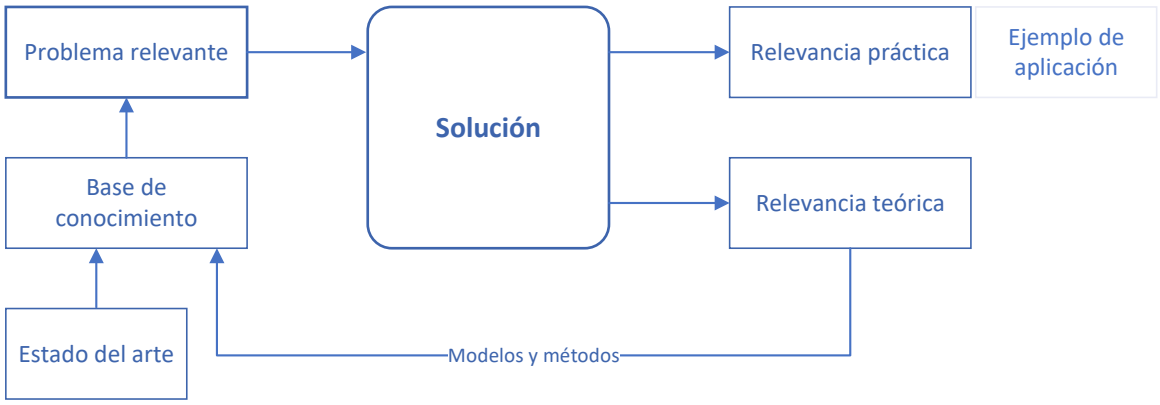


Figura 1.3 Enfoque de investigación constructivo.

Esta metodología divide el proceso de investigación en tres fases principales. La primera fase consiste en identificar un problema relevante desde un punto de vista práctico basado en un

conocimiento en profundidad de la literatura científica relacionada. Concretamente, las tareas de investigación desarrolladas en esta tesis comienzan con un análisis exhaustivo de la literatura científica relacionada con IoT y con WoT. La segunda fase de esta metodología involucra el diseño y la implementación de una solución que pueda ser aplicable para resolver el problema identificado en la fase anterior, y que responda a las preguntas de investigación planteadas para resolver el problema. Finalmente, la tercera fase de la metodología de investigación constructiva se centra en consolidar la contribución al estado del arte, así como en analizar los resultados obtenidos. Para ello, en el contexto de esta tesis, en esta fase se desarrolla la validación de las contribuciones mediante casos de estudio desplegados en escenarios reales.

Además de la metodología constructiva para guiar el proceso de investigación, las preguntas de investigación que trata de responder el desarrollo de esta tesis han sido formuladas por influencia de la metodología de *problematization* expresada en [33], en lugar de la ampliamente extendida *gap spotting* [34], dado que se entiende que es la que mejor encaje tiene en un proyecto de investigación industrial.

1.4. Objetivos

Conocidas la motivación y el alcance del trabajo que se pretende realizar, así como la metodología que dirigirá el proceso de investigación, esta tesis se fija dos propósitos fundamentales. En primer lugar, investigar el concepto de interoperabilidad de dispositivos físicos heterogéneos a través de Internet, con especial hincapié en las recomendaciones y estándares propuestas en este campo. Y, en segundo lugar, evaluar la viabilidad de una arquitectura para la interconexión de dispositivos heterogéneos que sirva de soporte para sistemas de gestión de energía de edificios, BEMS, implementada en base a las recomendaciones y estándares revisados con anterioridad.

Se espera alcanzar estos propósitos generales mediante la consecución de los siguientes objetivos:

- Revisar el estado del arte relacionado con la interconexión de dispositivos heterogéneos en redes inalámbricas de sensores y actuadores.
- Proponer una arquitectura basada en el paradigma *Web of Things* para el despliegue de una red de sensores y actuadores, así como su integración en una red de dispositivos heterogéneos de forma transparente.
- Validar la arquitectura propuesta utilizando diferentes casos de estudio, tanto en laboratorio como reales en entornos residenciales.

La consecución de estos objetivos y, por tanto, de los propósitos fundamentales de esta tesis, se alcanzará al responder las siguientes preguntas de investigación:

- **RQ1** ¿Cuáles son los estándares que permiten la interconexión, de forma transparente, de dispositivos físicos heterogéneos en redes inalámbricas?
- **RQ2** ¿En qué estándares se basan las arquitecturas de software para el soporte de sistemas de gestión de energía en edificios?
- **RQ3** ¿Cómo se puede dar soporte a sistemas de gestión de energía en edificios mediante el paradigma *Web of Things*?

1.5. Casos de estudio

La metodología de investigación que guía a esta tesis, descrita en la Sección 1.3, recoge el diseño y la implementación de una solución que pueda ser aplicable para resolver el problema identificado, así como contribuir al desarrollo del estado del arte. Para ello, se definen dos casos de estudio, que se tratan en detalle en el Capítulo 4, que permitirán validar las contribuciones de esta tesis:

1. El primer caso de estudio se llevará a cabo en un edificio de oficinas situado en Gijón (España), y se considerará como un experimento de laboratorio en un entorno reducido y controlado en el que, además de dispositivos hardware reales, varios dispositivos del sistema serán simulados mediante dispositivos virtuales. El objetivo de este caso de estudio es obtener datos preliminares sobre la viabilidad de la arquitectura *Web of Things* propuesta.
2. El segundo caso de estudio se llevará a cabo en un edificio de viviendas situado en Bagnolo (Italia), y constituirá un escenario de tamaño medio-grande para la solución del problema descrito en esta tesis. El objetivo de este caso de estudio es recabar datos reales con los que se pueda analizar la viabilidad de la arquitectura propuesta para proporcionar interoperabilidad de forma transparente a los dispositivos involucrados en la gestión energética conectados a la red de un edificio.

1.6. Organización de la tesis

La documentación de esta tesis está organizada de la siguiente forma.

En el Capítulo 2 se describe el contexto relacionado con el presente trabajo, a través de la revisión del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con *Internet of Things* y con *Web of Things*, así como su evolución histórica.

En el Capítulo 3 se presenta un análisis de diferentes arquitecturas hardware y software para la construcción de edificios energéticamente eficientes, centrándose particularmente en la implementación que da soporte a su gestor de energía, BEMS, así como la arquitectura

propuesta como soporte para la construcción de un BEMS basada en las recomendaciones del W3C.

En el Capítulo 4 se presenta la validación de los resultados obtenidos en los dos casos de estudio en los que se ha desplegado la arquitectura propuesta en esta tesis como soporte para sistemas de gestión energética en edificios.

Esta tesis finaliza con el Capítulo 5, que recoge las conclusiones más relevantes a las que se llega tras el desarrollo del proceso de investigación, así como las principales aportaciones a las que ha dado lugar, y se presentan las líneas de trabajo futuras que podrían derivarse de la misma.

Capítulo 2

Estándares y arquitecturas IoT y WoT

En este capítulo se describe el contexto relacionado con los objetivos planteados en la tesis a través de la revisión del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con *Internet of Things* y *con Web of Things*, así como su evolución histórica. Además de las referencias a la literatura científica que se citan en este capítulo, un excelente punto de partida introductorio a este contexto se encuentra en Tsiatsis *et al.* [35].

2.1. Internet of Things

El Internet de las cosas, más comúnmente conocido por su denominación en lengua inglesa como *Internet of Things* (IoT), es un paradigma que combina sistemas empotrados y redes de comunicación inalámbricas de muy bajo consumo con el objetivo de proporcionar conexión a Internet a objetos físicos cotidianos [36–40]. El principal cometido de este paradigma es extender el mundo real mediante la conexión de dispositivos, capaces de medir y actuar sobre el entorno, a través de Internet [41].

Si bien en la literatura científica se encuentran varias definiciones para IoT, las más comúnmente utilizadas son las propuestas por la *International Telecommunication Union* (ITU), que define IoT como:

> «una infraestructura global para la sociedad de la información, que permite servicios avanzados interconectando elementos (físicos y virtuales) basada en tecnologías de la información y la comunicación interoperables, tanto existentes como en evolución» [42],

y por el *Institute of Electrical and Electronic Engineers* (IEEE), que propone la siguiente definición, muy completa, para IoT:

> «una red compleja, autoconfigurable y adaptativa que interconecta "cosas" a Internet mediante el uso de protocolos de comunicación estándar. Las cosas interconectadas tienen representación física o virtual en el mundo digital, capacidad

> de detección y/o actuación, son programables e identificables de forma única. La representación contiene información que incluye la identidad, el estado, la ubicación o cualquier otra información comercial, social o privada de la cosa. Las cosas ofrecen servicios, con o sin intervención humana, mediante la explotación de la identificación única, la captura y comunicación de datos y la capacidad de actuación. El servicio se explota mediante el uso de interfaces inteligentes y está disponible en cualquier lugar, en cualquier momento y para cualquier cosa que tenga en cuenta la seguridad»[1] [43].

La conexión a Internet de objetos empotrados introduce muchos retos, dado que las tecnologías y protocolos desarrolladas originalmente para Internet no fueron diseñadas teniendo en cuenta este tipo de objetos. En este paradigma, los objetos a los que se les proporcionan capacidades de comunicación reciben el nombre de dispositivos. Un dispositivo IoT puede intercambiar datos con otros dispositivos conectados y con aplicaciones, puede adquirir, almacenar y procesar datos, así como actuar sobre el entorno. Para gestionar y explotar estos datos, IoT ofrece multitud de aplicaciones que dan servicio a los usuarios. Estas aplicaciones pueden ser locales, estar alojadas en servidores centralizados o distribuidas en la nube. Los servicios ofrecidos por IoT, tales como el descubrimiento y el control de dispositivos o la analítica de datos, constituyen el núcleo de los entornos inteligentes, o *smart environments*. Ejemplos de estos entornos son casas (*smart homes*), industrias (*smart industries*) o ciudades inteligentes (*smart cities*), entre otros muchos.

Además de los dispositivos, los mecanismos de comunicación y los servicios, un sistema IoT cuenta con otros bloques funcionales organizados en capas, tales como la capa de gestión, que permite definir y controlar la política de operación del sistema; la capa de seguridad, que proporciona funciones de autenticación, encriptación, privacidad, integridad de mensajes y seguridad de los datos; y la capa de aplicación, que proporciona la interfaz con los usuarios del sistema a través de módulos de control y visualización. La forma en la que se define cada uno de estos bloques funcionales, la relación entre cada uno de ellos, y el funcionamiento del sistema, determina lo que se conoce como arquitectura IoT.

Inicialmente, en ausencia de estándares y protocolos específicos, el acceso a los dispositivos IoT se realizaba a través de *gateways* que traducían entre protocolos de Internet y protocolos propietarios de fabricantes de dispositivos, tal y como se muestra en la Figura 2.1. En la última década varias organizaciones han propuesto estándares para el intercambio de información entre máquinas remotas, *machine-to-machine* (M2M), y para IoT [44–46] basados en protocolos IP en los que se sustentan estas arquitecturas de referencia. En la literatura científica se encuentran varios trabajos que revisan en profundidad el estado del arte sobre estándares para interoperabilidad en *Internet of Things*, entre los que destacan Lee *et al.* (2021) [47] y Derhamy *et al.* (2015)[48].

[1] https://iot.ieee.org/definition.html

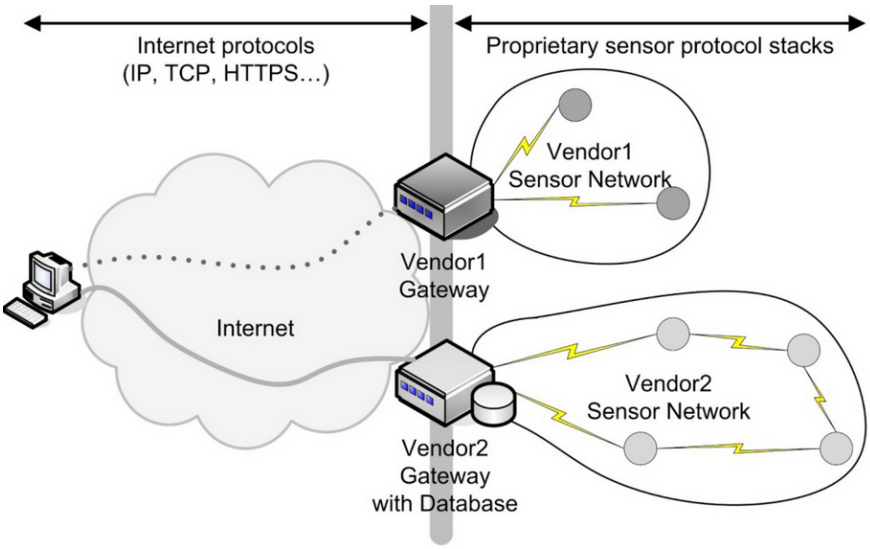


Figura 2.1 Conexión de dispositivos a Internet utilizando *gateways* [46].

A continuación se revisa el estado del arte relacionado con arquitecturas de alto nivel, o modelos de referencia, y estándares para IoT agrupados por las organizaciones que las han propuesto. El orden en que se presentan no sigue ningún criterio específico.

2.1.1. International Telecommunication Union

El grupo de telecomunicaciones de la *International Telecommunication Union* (ITU-T) creó en 2011 la *IoT Global Standards Initiative* (IoT-GSI) para tratar todos los aspectos relacionados con recomendaciones sobre IoT. Esta iniciativa finalizó sus trabajos en 2015 y se creó entonces el *Study Group 20* (SG20) sobre IoT y aplicaciones relacionadas, cuya recomendación Y.2060[2], renombrada a Y.4000 en 2016, proporciona una visión de alto nivel de IoT que constituye la base del resto de recomendaciones relacionadas del ITU-T y pretende ser un paraguas para el desarrollo de estándares IoT a nivel mundial.

El modelo de IoT definido por el ITU-T engloba objetos físicos que se conectan directamente entre sí, o a través de otros dispositivos tales como *gateways*, a redes de comunicaciones mediante las cuales pueden intercambiar información con otros dispositivos, servicios o aplicaciones. En este modelo, que también seguirán otros organismos de estandarización, o *Standards Developing Organizations* (SDO), los dispositivos están obligados a proporcionar mecanismos de comunicación, y pueden implementar de forma opcional mecanismos de adquisición, de actuación o de procesamiento de datos.

La Figura 2.2 muestra un esquema de este modelo de referencia, donde se puede observar que la capa de aplicación es la que proporciona la interfaz al usuario. Esta capa se sustenta sobre una capa de soporte que proporciona servicios genéricos para todas las aplicaciones

[2]https://www.itu.int/rec/T-REC-Y.2060-201206-I

IoT, como por ejemplo adquisición y procesamiento de datos. Esta capa de soporte también engloba servicios específicos para dominios de aplicación concretos (salud, industria, ocio, etc.). Bajo esta capa, la capa de red proporciona funcionalidades de autenticación y autorización, así como de movilidad y de conectividad para los servicios IoT. Finalmente, la capa de dispositivos identifica las capacidades de los dispositivos y de los *gateways*. En el primer grupo se engloban capacidades de interacción directa con las redes de comunicaciones, mientras que en el segundo se tienen en cuenta aspectos relacionados con comunicaciones indirectas a través de *gateways*, como soporte para múltiples protocolos o conversión de protocolos, entre otros. De forma transversal a las capas citadas, la capa de gestión incluye modelos como FCAPS (*Fault, Configuration, Accounting, Performance, Security*) y aspectos relacionados con la gestión de dispositivos, como provisión, activación y desactivación, actualización de software. La otra capa transversal engloba aspectos de seguridad requeridos por el resto de capas, tales como autenticación y autorización, integridad y confidencialidad de mensajes.

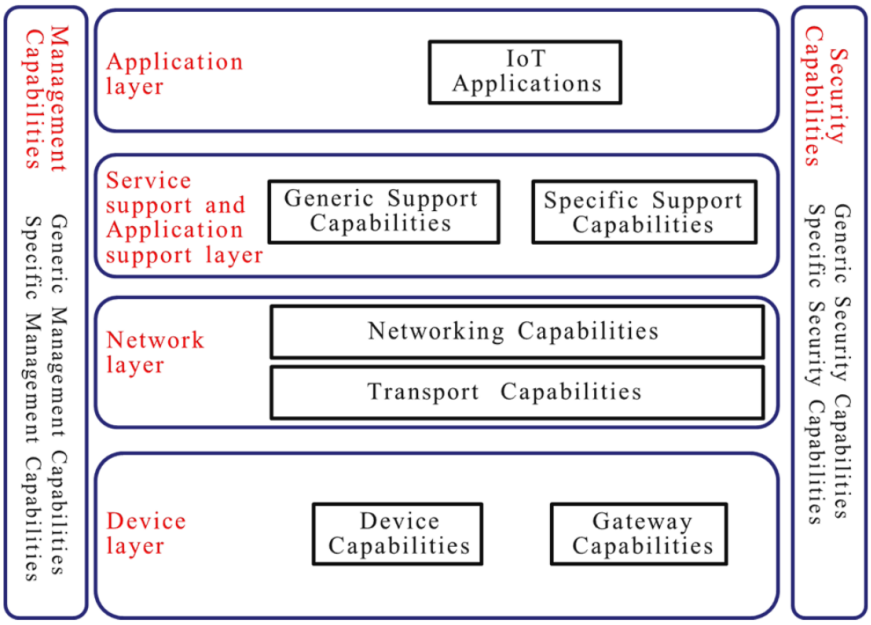


Figura 2.2 Modelo de referencia de IoT del ITU-T [42].

2.1.2. Internet Engineering Task Force

La *Internet Engineering Task Force* (IETF) es una organización para el desarrollo de estándares abiertos relacionados con tecnologías utilizadas en Internet, como por ejemplo los protocolos IP, TCP y UDP. Los participantes de cualquier organización pueden transmitir sus intereses en tecnologías de Internet a IETF, dirigidos al grupo o grupos de trabajo correspondientes. Cuando el trabajo desarrollado por un grupo llega a una fase consolidada y

comienza la difusión se publica a través de un documento denominado *Request For Comments* (RFC).

En los últimos años, IETF ha constituido varios grupos relacionados con el desarrollo de estándares para IoT [46]. Entre ellos cabe mencionar el grupo *IPv6 over Low power WPAN* (6lowpan), activo desde los años 2004 a 2014, cuyo objetivo principal era permitir la utilización del protocolo IP a cualquier dispositivo en Internet, independientemente de su tamaño y su capacidad computacional. El trabajo relacionado con la transmisión sobre IPv6 dio lugar a la publicación del RFC 4919 "*IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*"[3] y del RFC 4944 "*Transmission of IPv6 Packets over IEEE 802.15.4 Networks*"[4] que, entre otros, constituyen la base de la capa de adaptación que IETF propone para IoT.

El grupo *IPv6 over Networks of Resource-constrained Nodes* (6lo), constituido en 2013, se centra en facilitar la conectividad mediante IPv6 sobre redes y dispositivos con recursos limitados, tanto de potencia, como de computacionales y de memoria. Este grupo ha publicado, entre otros, el RFC 7388 "*Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*"[5] y el RFC 7668 "*IPv6 over Bluetooth Low Energy*"[6].

El grupo *IPv6 over the TSCH mode of IEEE 802.15.4e* (6tisch), constituido también en el año 2013, centra su esfuerzo en la transmisión en redes de baja potencia con pérdida o *Low-Power Lossy Networks* (LLN) y define protocolos para varias capas incluyendo la adaptación y el enrutamiento sobre IEEE 802.15.4, publicando, entre otros, el RFC 7554 "*Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement*"[7] y el RFC 8180 "*Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration*"[8].

Por su parte, el grupo de trabajo *Routing Over Low power and Lossy networks* (roll), constituido en el año 2017, se encarga de la resolución de problemas relacionados con el enrutamiento en entornos formados por redes de sensores, con especial énfasis en lograr una alta fiabilidad en redes de muy baja potencia y con probabilidad de pérdida de paquetes. Como resultado de este trabajo, el grupo ha publicado el RFC 6550 "*RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks'*'[9], que recoge las especificaciones de un protocolo de enrutamiento para la capa de red en el modelo IoT de IETF.

Otro grupo de trabajo con implicación directa en IoT es *Constrained RESTful Environments* (CoRE). Constituido en 2010, este grupo centra sus esfuerzos en definir protocolos para aplicaciones basados en arquitecturas REST [49]. Los servicios que implementan un

[3]https://datatracker.ietf.org/doc/html/rfc4919
[4]https://datatracker.ietf.org/doc/html/rfc4944
[5]https://datatracker.ietf.org/doc/html/rfc7838
[6]https://datatracker.ietf.org/doc/html/rfc7668
[7]https://datatracker.ietf.org/doc/html/rfc7554
[8]https://datatracker.ietf.org/doc/html/rfc8180
[9]https://datatracker.ietf.org/doc/html/rfc6550

modelo RESTful proporcionan interfaces accesibles mediante URIs a través de cuatro métodos HTTP: GET, PUT, POST y DELETE. La publicación más destacada de este grupo es el RFC 2752 "The Constrained Application Protocol (CoAP)"[10], que incluye la especificación de un protocolo de paso de mensajes con arquitectura cliente-servidor basado en UDP. Este protocolo permite el acceso a dispositivos con recursos limitados a través de los métodos GET, PUT, POST y DELETE mencionados anteriormente.

La Figura 2.3 muestra una comparativa entre la pila de protocolos propuesta por IETF como soporte para IoT y la pila de protocolos TCP/IP utilizada en Internet. Y la Figura 2.4 muestra la propuesta de extensión de los protocolos de Internet a dispositivos IoT realizada por IETF.

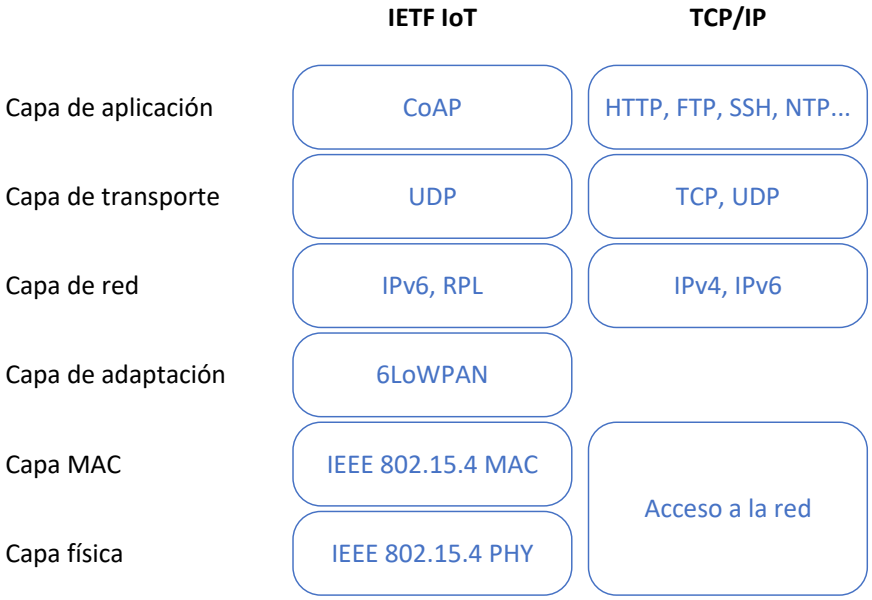


Figura 2.3 Especificaciones de IETF para IoT: comparativa con la pila de protocolos TCP/IP.

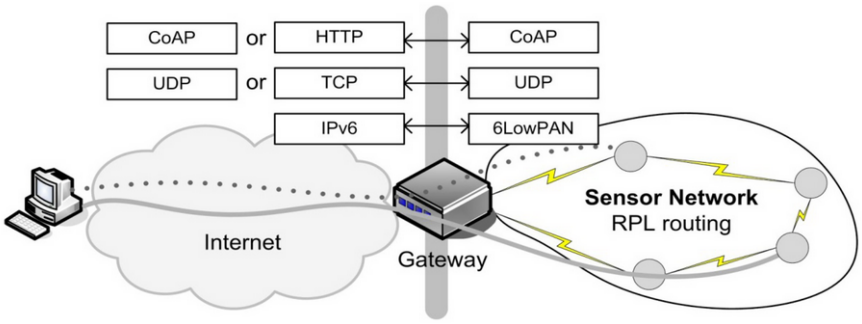


Figura 2.4 Extensión de protocolos de Internet a dispositivos IoT propuesta por IETF [46].

[10]https://datatracker.ietf.org/doc/html/rfc7252

2.1.3. OMA SpecWorks

La organización *Open Mobile Alliance* (OMA), creada en 2002, se fusionó con la *Internet Protocol for Smart Objects Alliance* (IPSO) en 2018 para formar la OMA *SpecWorks*, una organización para el desarrollo de estándares técnicos abiertos en el ámbito de la industria de la telefonía móvil. El principal cometido de la organización es fomentar la interoperabilidad, entre diferentes países, tanto de operadores, como de servicios y terminales móviles.

En el ámbito más relacionado con IoT, la OMA definió una arquitectura cliente-servidor para la comunicación de directa de dispositivos en redes de sensores, siguiendo el patrón REST, y un protocolo para la gestión de dispositivos a nivel de capa de aplicación, denominado Lightweight M2M (LwM2M) [50]. LwM2M se centra en la gestión de dispositivos de muy bajo consumo, lo que resulta particularmente útil en IoT.

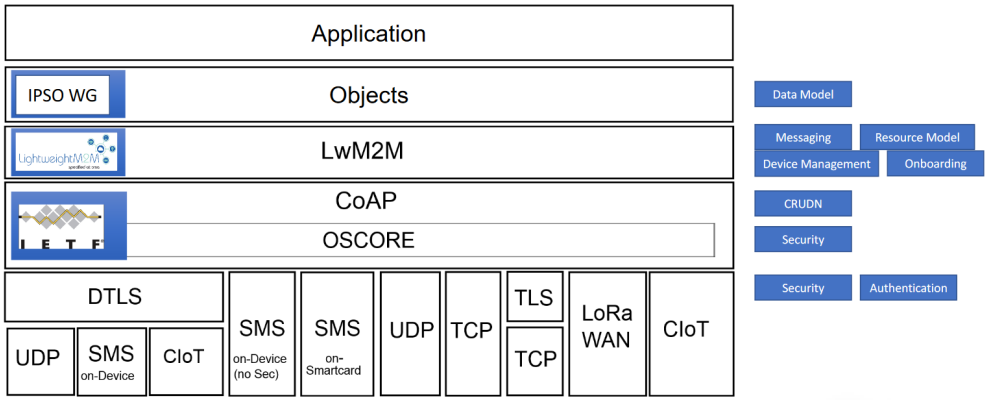


Figura 2.5 Arquitectura OMA SpecWorks LwM2M [50].

Como se puede observar en la Figura 2.5, la pila de protocolos de esta arquitectura utiliza CoAP a nivel de transferencia, mientras que a nivel de transporte puede utilizar varios protocolos, como UDP o SMS, entre otros, y las transferencias pueden incrementar su nivel de seguridad utilizando Datagram Transport Layer Security (DTLS). La capa LwM2M describe los objetos, los recursos y las operaciones que permiten la implementación de las interfaces entre un cliente y un servidor LwM2M. Cada uno de los objetos o dispositivos en esta arquitectura puede incluir sensores, actuadores, firmware y un cliente LwM2M. Los objetos se agrupan en recursos, sobre los que se pueden realizar operaciones de lectura, escritura o ejecución, así como asignar características, tales como solo-lectura o lectura/escritura.

La OMA mantiene un registro de objetos y recursos[11] en el que, además de los definidos por la propia OMA, se encuentran objetos y recursos definidos por otras organizaciones, tales como *Global System for Mobile Communications* (GSMA) y oneM2M, así como por otras corporaciones y empresas, entre las que cabe destacar ARM, Huawei y Vodafone.

[11]https://technical.openmobilealliance.org/OMNA/LwM2M/LwM2MRegistry.html

2.1.4. oneM2M

oneM2M es una iniciativa global constituida en 2012 tras la agrupación de organizaciones de estandarización como la *Association of Radio Industries and Businesses* (ARIB), el *European Telecommunications Standards Institute* (ETSI) y el *Telecommunication Technology Committee* (TTC), entre otras. El objetivo que persigue esta iniciativa es el desarrollo de estándares que permitan crear servicios IoT interoperables, seguros y sencillos de desplegar, es decir, especificar una capa de servicios para comunicaciones M2M.

La arquitectura propuesta por oneM2M es similar a un sistema operativo distribuido para IoT. Se comporta como una capa de servicio *middleware* constituida por varias funciones, denominadas *common service functions* (CSF), que se exponen a las aplicaciones y dispositivos IoT mediante APIs RESTful, tal como se muestra en la Figura 2.6.

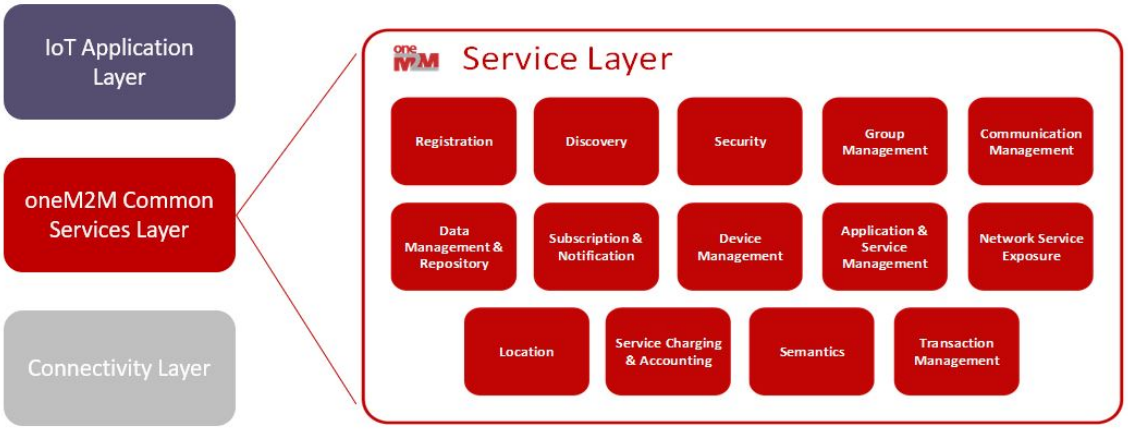


Figura 2.6 Arquitectura oneM2M [51].

Los estándares que desarrolla oneM2M, denominados especificaciones técnicas o *technical specifications* (TS), son abiertos y de ámbito internacional. Varias especificaciones promovidas por oneM2M se han incluido en recomendaciones ITU-T, lo que les da mayor visibilidad y adopción a nivel global, como por ejemplo las especificaciones técnicas TS-0001 "*oneM2M - Functional architecture*"[12] a TS-0032 "*oneM2M - MAF and MEF interface specification*"[13] que han sido traspuestas a las recomendaciones Y.4500.1[14] a Y.4500.32[15]. Estas especificaciones describen la arquitectura oneM2M así como la interconexión entre oneM2M y HTTP, CoAP, MQTT y LwM2M.

[12]https://www.onem2m.org/images/files/deliverables/Release3/TS-0001-Functional_Architecture-V3_15_1.pdf
[13]https://www.onem2m.org/images/files/deliverables/Release3/TS-0032-MAF_and_MEF_Interface_Specification-V3_0_0.pdf
[14]https://www.itu.int/rec/T-REC-Y.4500.1
[15]https://www.itu.int/rec/T-REC-Y.4500.32

Además de promover estándares, oneM2M proporciona, a través de la plataforma OCEAN (*Open Alliance for IoT Standards*)[16], implementaciones de código libre para la comunicación de dispositivos, *gateways*, servidores y aplicaciones, así como herramientas de desarrollo y de verificación de cumplimiento de estándares, todo ello bajo licencia BSD.

2.1.5. Open Connectivity Foundation

La *Open Connectivity Foundation* (OCF) es una organización que desarrolla estándares para IoT surgida de la evolución del *Open Interconnect Consortium* (OIC), constituido en 2014. Este consorcio comenzó a desarrollar el proyecto IoTivity[17], que perseguía desarrollar una implementación de referencia de estándares propuestos por el OIC mediante la elaboración de normas de interoperabilidad y programas de certificación para IoT. En 2015 adquirió los activos de *Universal Plug and Play* (UPnP), un conjunto de protocolos que permiten el descubrimiento y la configuración de dispositivos de red.

Posteriormente se unió con la *AllSeen Alliance*, que desarrollaba un framework de código abierto para el descubrimiento y la comunicación de dispositivos IoT, denominado AllJoyn[18]. Actualmente OCF mantiene el desarrollo de estos dos proyectos de código abierto.

La arquitectura que plantea OCF se basa en una arquitectura orientada a recursos, siguiendo el modelo cliente-servidor con el patrón REST, de forma similar a las arquitecturas OMA LwM2M y oneM2M. A nivel de aplicación utiliza el protocolo CoAP, con encriptación mediante DTLS cuando se transporta sobre UDP y con TLS cuando se transporta sobre TCP. De manera opcional también puede utilizar a nivel de aplicación HTTP.

2.1.6. Institute of Electrical and Electronic Engineers

El *Institute of Electrical and Electronic Engineers* (IEEE) es una de las mayores organizaciones profesionales del ámbito técnico y tecnológico, que desempeña un papel importante en la estandarización de diferentes aspectos relacionados con IoT y con los protocolos de comunicación que sientan las bases de este paradigma.

El IEEE creó en 2014 el grupo de trabajo 2413 con el objetivo de definir una arquitectura de referencia para IoT, recogido en el estándar IEEE 2413-2019 "*IEEE Standard for an Architectural Framework for the Internet of Things (IoT)*"[19]. También en 2014 se creó el grupo de trabajo Devices and Systems Harmonization Working Group (DASH) en el que se desarrolla el proyecto P1451-99 "*Standard for Harmonization of Internet of Things (IoT) Devices and Systems*", que persigue la definición de un método interoperable que permita compartir datos de forma segura a través de una red, donde los sensores, actuadores y otros dispositivos pueden interoperar, independientemente de la tecnología de comunicación subyacente.

[16]http://developers.iotocean.org/
[17]https://iotivity.org
[18]https://openconnectivity.org/technology/reference-implementation/alljoyn/
[19]https://standards.ieee.org/standard/2413-2019.html

2.1.7. International Standarization Organization

La *International Standarization Organization* (ISO) es una organización para la creación de estándares internacionales que está compuesta por varias organizaciones nacionales de estandarización. En conjunción con la *International Electrotechnical Commission* (IEC) creó, en el año 2015, el grupo de trabajo ISO/IEC JTC/WG 10[20], con el objetivo de proponer una arquitectura de referencia para IoT. Posteriormente, en 2017, el grupo de trabajo se reconvirtió a un subcomité, ISO/IEC JTC 1/SC 41, de forma que guiara el proceso de estandarización de IoT y tecnologías relacionadas con redes de sensores. En 2018 publicó una arquitectura de referencia en la norma ISO/IEC 30141:2018 *Internet of Things (IoT) — Reference Architecture*[21].

La arquitectura de referencia ISO/IEC proporciona un modelo conceptual en el que se definen las entidades IoT, así como sus relaciones, y un modelo de referencia basado en entidades. El modelo de referencia incluye todas las entidades físicas que se pueden monitorizar y controlar, así como los dispositivos de instrumentación (sensores y actuadores), de comunicación y de computación (*gateways* y servidores).

2.2. Web of Things

De la misma forma que el éxito de Internet se ha basado, no en la interconexión global de computadores, sino en una capa que proporciona servicios Web a nivel global, se busca una capa Web análoga para IoT, que convierta el Internet de las Cosas en la Web de las Cosas, también denominada *Web of Things* (WoT) [52]. Desde el principio, WoT se ha considerado como una extensión de IoT. Aunque no existe una definición estándar, IoT es considerado como el paradigma de interconexión de dispositivos heterogéneos, mientras que WoT se encarga de gestionar aspectos relacionados con la interoperabilidad desde la capa de aplicación.

Esta capa tiene como objetivo facilitar el acceso a dispositivos tales como sensores y actuadores, evitando las limitaciones impuestas por IoT debido a soluciones verticales y sistemas cerrados que crean un problema de interoperabilidad comúnmente conocido como silos IoT. La característica principal de esta capa consiste en utilizar estándares y tecnologías Web ampliamente extendidas que permitan crear aplicaciones y servicios que de otra forma no podrían ser accesibles mediante IoT.

En el mundo real, las *Things* se pueden considerar como *proxies* de objetos físicos o entidades abstractas, que se comunican mediante tecnologías y servicios Web [53], como por ejemplo, a través del protocolo HTTP.

El término *Web of Things* apareció a principio de los años 2000, a través de la idea *Web presence*, en el laboratorio de Internet y Tecnologías Móviles de HP [54]. En 2007 se extendió

[20] https://www.iec.ch/dyn/www/f?p=103:14:0::::FSP_ORG_ID,FSP_LANG_ID:12726,25#

[21] https://www.iso.org/standard/65695.html

con la idea de asignar un URI (*Uniform Resource Identifier*) a cada objeto, a la vez que se podía interactuar con él mediante los métodos más básicos de HTTP: PUT, GET, POST y DELETE [55]. Cada objeto puede por tanto representarse como un recurso Web mediante una descripción en HTML, XML o JSON, por ejemplo. La idea fundamental de WoT es integrar *Smart Things* no solo en la capa de red de Internet, sino en la capa de arquitectura Web [56], tal y como se muestra en el modelo por capas propuesto por Guinard y Trifa [57] representado en la Figura 2.7.

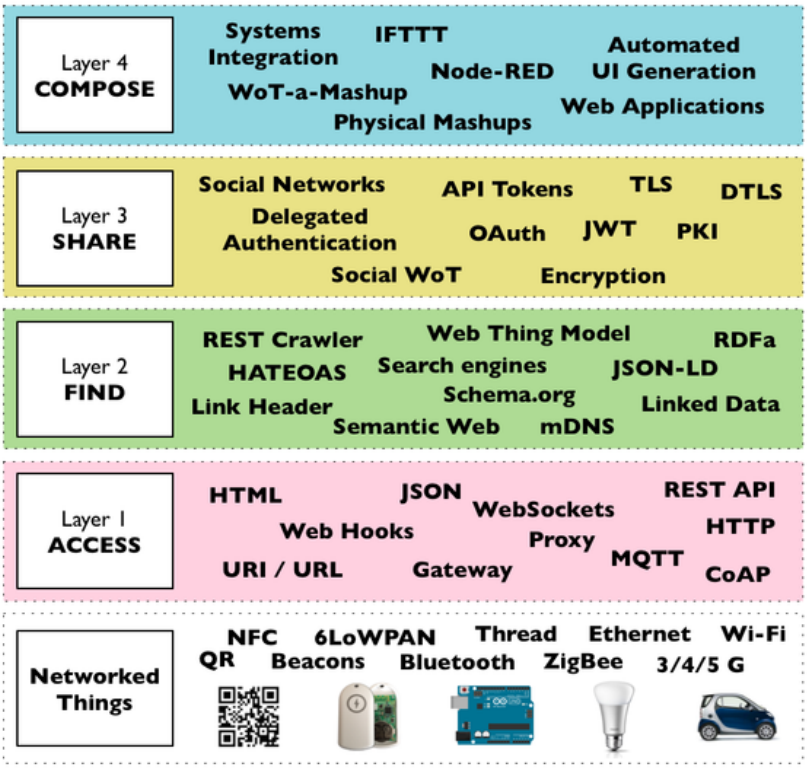


Figura 2.7 Modelo WoT propuesto por Guinard y Trifa [57].

WoT persigue la interoperabilidad entre diferentes dispositivos y soluciones IoT. Se deben tener en cuenta dos requisitos muy importantes para lograr esta interoperabilidad. En primer lugar, WoT está orientada a reducir la heterogeneidad del ecosistema IoT y, por tanto, necesita escalar de manera acorde, adaptándose a los múltiples requisitos que exigirá una enorme variedad de dispositivos [58]. Y, en segundo lugar, la interoperabilidad deseada se debe obtener con un consumo de energía muy bajo, dado que los diferentes sistemas pueden utilizar conexiones basadas en técnicas de radio de muy bajo consumo en la capa de enlace o en la subcapa MAC.

Las arquitecturas WoT se configuran como una capa que se superpone a las arquitecturas IoT, y se basan en definir un sistema capaz de representar y exponer los recursos de los dispositivos del sistema IoT de una manera que sean accesibles como recursos Web. A

continuación, se presentan algunos de los esfuerzos de estandarización más reconocidos para las tecnologías WoT. Estos estándares, o recomendaciones en algunos casos, proponen arquitecturas basadas en WoT que cubren un abanico genérico de necesidades a la hora de desplegar un ecosistema IoT.

2.2.1. Open Geospatial Consortium

El *Open Geospatial Consortium* (OGC) es un consorcio internacional formado por más de quinientas empresas, agencias gubernamentales, organismos de investigación y universidades con el objetivo de resolver problemas relacionados con la creación, la comunicación y el uso de información de localización, es decir, de obtener el máximo rendimiento y aprovechamiento a partir de información geoespacial. Para ello, OGC hace de interlocutor entre diferentes dominios, desarrolla proyectos piloto y realiza experimentos de interoperabilidad para poder identificar interfaces y estándares. Para lograr este último punto, se organizan en comités y en grupos de trabajo y, entre otras actividades, desarrollan estándares abiertos, gratuitos y de uso libre.

Uno de los principales conjuntos de estándares que define OGC se denomina *Sensor Web Enablement* (SWE)[22] que persigue el desarrollo de interfaces abiertas para el desarrollo de sensores que se conecten mediante aplicaciones Web, el descubrimiento de sensores, la suscripción y la publicación de alertas por parte de sensores, etc. En base a este estándar, OGC define una arquitectura WoT distribuida mediante el modelado y la representación de datos observados o adquiridos por sensores y un conjunto de servicios Web para controlar los sensores y actuadores en cada solución[23]. En 2016, OGC publicó otro estándar dentro de SWE, denominado *SensorThings*, que trata de incorporar interoperabilidad semántica en sistemas IoT. Este estándar define una API dividida en un bloque de gestión de sensores IoT [59] y un bloque de operación de otros dispositivos [60]. A través de una API REST utilizando HTTP (u opcionalmente MQTT), un servidor *SensorThings* gestiona la comunicación mediante operaciones CRUD (*Create, Read, Update, Delete*) de los dispositivos en el sistema.

Los estándares OGC SWE se centran más en el flujo de información dentro del sistema que en la comunicación que se establece entre los dispositivos, al contrario de lo que sucede, por ejemplo, con los estándares propuestos por IEEE e IETF, que se centran más en los aspectos de comunicación.

2.2.2. International Telecommunication Union

La *International Telecommunication Union* (ITU) define un *framework* para WoT, ITU-T Y.4400/Y.2063 [61], con el objetivo de facilitar el acceso a dispositivos utilizando métodos unificados mediante servicios Web. Esta recomendación demuestra cómo pueden interactuar

[22] https://www.ogc.org/node/698
[23] http://www.opengeospatial.org/standards/

dispositivos físicos con servicios Web. Plantea, además, que WoT debe dar soporte a la interoperabilidad entre redes y sistemas operativos diferentes.

La arquitectura WoT propuesta por la ITU define tres capas, tal y como se puede observar en la Figura 2.8: servicio, adaptación y física. La capa de servicio proporciona acceso a la funcionalidad del sistema y es la responsable de hacer que el servicio esté disponible. La capa de adaptación está compuesta por agentes WoT que traducen los mensajes de diferentes protocolos e interaccionan con los objetos físicos. Y la capa física es donde se sitúan los objetos físicos del entorno. Cada uno de estos objetos puede ser accesible por un agente de la capa de adaptación.

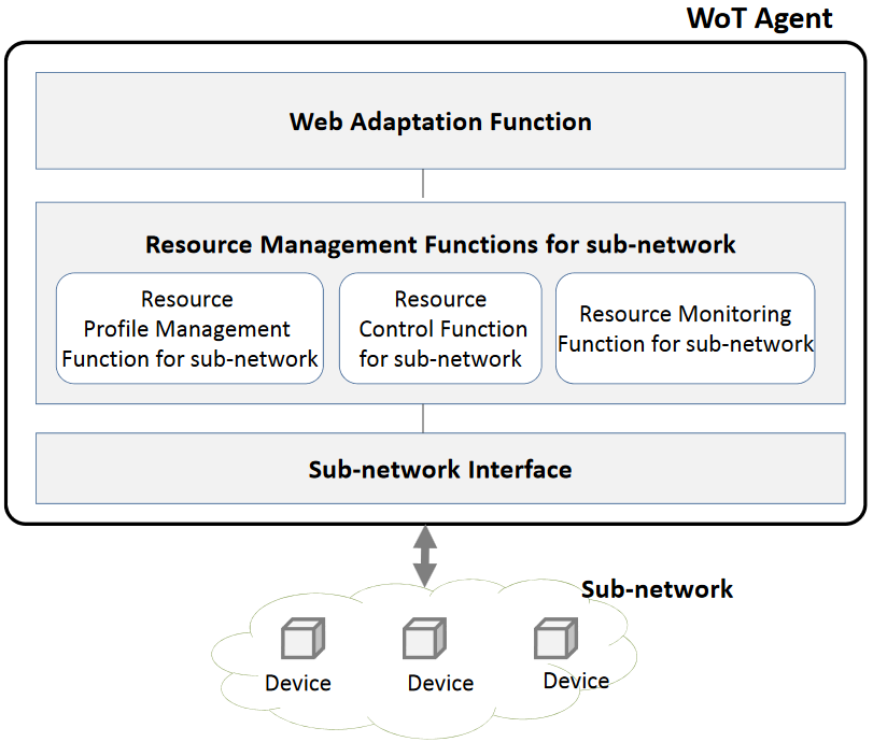


Figura 2.8 Agente WoT propuesto por ITU-T [61].

La Figura 2.9 muestra una representación de la arquitectura WoT propuesta por la recomendación ITU-T Y.4400/Y.2063. Las capas de servicio y adaptación conforman el denominado *broker* WoT, que implementa entidades funcionales como los agentes, y permite descubrir, registrar, ejecutar y componer tanto servicios como agentes.

2.2.3. World Wide Web Consortium

El *World Wide Web Consortium* (W3C) es el principal organismo de estandarización internacional para la *World Wide Web*. Con casi quinientos miembros, el W3C se centra en el desarrollo de estándares, directrices y protocolos, todos ellos abiertos y gratuitos, que garanticen el mantenimiento y crecimiento de la Web. El W3C basa sus principios de diseño

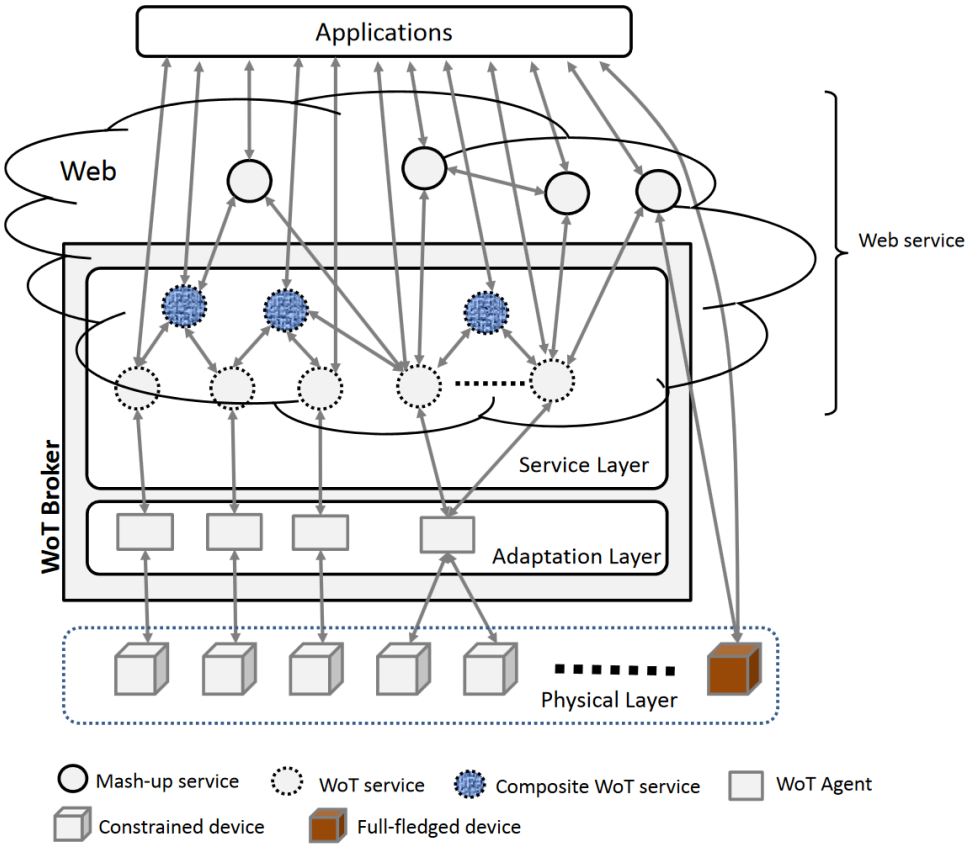


Figura 2.9 Modelo de referencia de WoT del ITU-T [61].

en dos conceptos muy claros: *Web for All* y *Web on Everything*. Con el primero pretende que los beneficios de la Web lleguen a todas las personas, independientemente del hardware, el software, la infraestructura de red, el idioma, la cultura o la localización geográfica desde la que accedan a la Web. Para ello, este primer concepto explota aspectos tales como la accesibilidad y la internacionalización. El segundo concepto se refiere a que la Web pueda ser accedida desde cualquier tipo de dispositivo, desde teléfonos móviles o televisores, a cualquier dispositivo doméstico o industrial.

En 2014, el W3C muestra su interés en WoT mediante la creación del WoT IG[24], grupo que definió, en 2015, el primer modelo del W3C para WoT [62]. Posteriormente, en 2017, el W3C creó el grupo de trabajo sobre *Web of Things*, denominado WoT WG[25], con el propósito de contrarrestar la fragmentación en IoT mediante el uso de tecnologías Web estándar, o mediante la ampliación de las tecnologías Web existentes. El camino propuesto para lograr una integración sencilla entre plataformas y dominios de aplicación de IoT es la utilización metadatos estandarizados y de componentes tecnológicos (o *building blocks*) también estandarizados y que además sean reutilizables. La principal aportación de este grupo

[24]https://www.w3.org/WoT/ig/
[25]https://www.w3.org/WoT/wg/

de trabajo hasta el momento es la especificación de una arquitectura para la *Web of Things*, denominada *W3C Web of Things Architecture* [63, 64]. Esta especificación, que representa un estándar *de facto* a nivel de implementación para cada componente de un sistema WoT, define un *framework* conceptual que se puede mapear en el despliegue de soluciones creadas para diversos escenarios.

Dado que en este trabajo se siguen las recomendaciones de la arquitectura WoT propuesta por el W3C, esta arquitectura se describe en detalle en el siguiente apartado.

2.3. W3C WoT

La *Web of Things* incrementa la flexibilidad y la interoperabilidad de los diferentes dispositivos en una solución IoT, a la vez que facilita la reutilización de los componentes desarrollados entre diferentes soluciones. Además, dado que los estándares propuestos son gratuitos y de acceso libre, se eliminan las limitaciones que se imponen en el uso de patentes en otras arquitecturas comerciales.

El modelo de WoT del W3C define una abstracción de la interacción entre componentes basada en enlaces, propiedades, eventos y acciones. A través de esta abstracción, las aplicaciones pueden acceder a servicios y a datos en cualquier red IoT que implemente el modelo, siguiendo el paradigma propiedades-acción-evento. Además, los conceptos de este modelo WoT se pueden aplicar a todos los niveles de soluciones y sistemas IoT, desde el dispositivo individual a los niveles de *edge* y *cloud*. La Figura 2.10 muestra un esquema de la arquitectura de referencia WoT propuesta por el W3C, que se describe en detalle a continuación, y en la que se pueden observar todos los niveles a los que se puede aplicar.

Los componentes de una arquitectura WoT son los objetos o *Things*, los intermediarios y los consumidores. En la arquitectura propuesta por el W3C, se denomina *servient* (palabra compuesta por *server* y *client*) a la pila software que implementa estos componentes.

2.3.1. Objetos, consumidores e intermediarios

Una *Web Thing*, o simplemente *Thing*, es una abstracción de una entidad física o virtual que se describe mediante metadatos en base a algún tipo de estandarización. La arquitectura WoT del W3C no especifica dónde se debe implementar una *Thing*, más allá de que deben implementarse en algún componente conectado a la red que proporcione una pila de protocolos software que le permitan interactuar con el entorno. Por ejemplo, se podría implementar en un dispositivo IoT empotrado, conectado con sensores y actuadores, que a su vez contenga un servidor HTTP embebido. Aunque también podría estar almacenado en la nube o en un dispositivo intermedio como un *gateway*.

Una *Thing* se puede implementar mediante un *servient*, que contiene la representación de la *Thing*, denominada *Exposed Thing*. El *servient* permite a los consumidores acceder a la interfaz

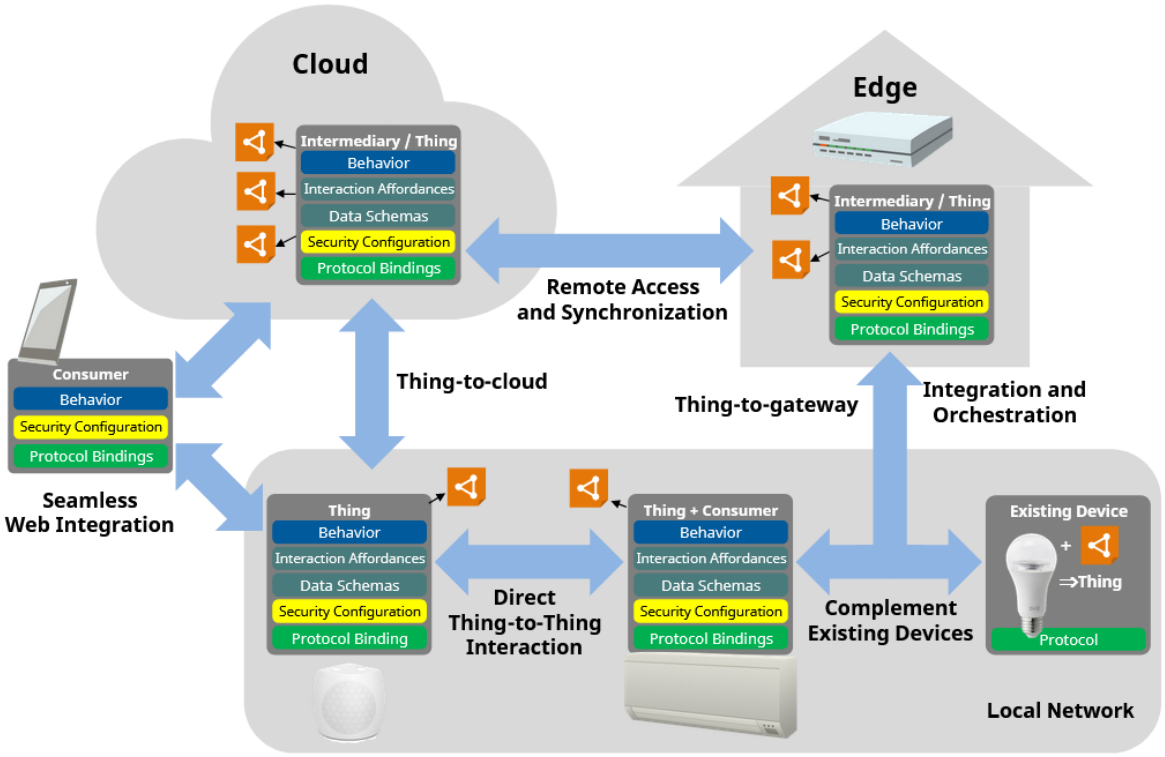


Figura 2.10 Arquitectura WoT de referencia propuesta por el W3C [63].

WoT de la *Thing*. Por su parte, un consumidor debe implementarse obligatoriamente mediante un *servient*, que debe ser capaz de procesar una *Thing Description* e implementar la pila de protocolos que esta última especifique. Finalmente, un intermediario es otro componente de la arquitectura WoT que se implementa mediante un *servient*, y que realiza las tareas de una *Thing* desde el punto de vista de consumidor (para ello el *servient* realiza la labor de un *proxy* de la *Thing* expuesta), y las de un consumidor desde el punto de vista de una *Thing*. Los intermediarios se precisan cuando los *servients* utilizan diferentes protocolos o si se encuentran en redes distintas que requieran control de acceso o autorización, por ejemplo, a través de un *firewall*. De esta forma, la aplicación ejecutada por un consumidor que utilice HTTP podría acceder a una *Thing* que utiliza CoAP, MQTT o Modbus, por ejemplo, definiendo los mensajes específicos de cada protocolo en *Protocol Bindings*. Además del reenvío de mensajes, pueden implementar técnicas de almacenamiento temporal de datos, a modo de caché.

2.3.2. Modelo de interacción

Un recurso Web representa generalmente un documento en la *World Wide Web* que puede ser accedido mediante un cliente Web. Con la aparición de los servicios Web, los recursos se convierten en entidades de interacción más genéricas, que pueden implementar cualquier tipo de comportamiento. Aunque se incrementan las posibilidades de interacción, con un nivel tan

alto de abstracción se hace difícil proporcionar un acoplamiento flexible entre aplicaciones y recursos. De ahí que el modelo de interacción propuesto por el W3C para WoT introduzca un nivel intermedio de abstracción que mapea aplicaciones a operaciones concretas de un determinado protocolo.

Además de la interacción en base a enlaces Web, las *Things* pueden ofrecer otros tres tipos de interacción: propiedades, acciones y eventos. Estos cuatro tipos de interacción pueden modelar prácticamente todas las posibilidades de interacción que se encuentran en los dispositivos y servicios de IoT [63]. Como se verá más adelante, las *Things* describen su funcionalidad a través de *Thing Description* y la exponen a través de un *WoT interface*.

2.3.2.1. Propiedades

Las propiedades son interacciones que representan valores o estados expuestos por una *Thing*, que deben poder ser leídos y que, opcionalmente, pueden ser modificados. Por ejemplo, valores que se obtienen de sensores (de tipo solo lectura), parámetros de configuración de dispositivos hardware (de tipo lectura y escritura), valores de estado (que pueden ser de solo lectura o de lectura y escritura) y resultados de operaciones concretas (que normalmente son de solo lectura).

2.3.2.2. Acciones

Las acciones encapsulan el mecanismo para invocar a una función en una *Thing*. La principal característica de una acción es que puede modificar un estado dentro de una *Thing*, por ejemplo una propiedad, que no está expuesta. Además, la invocación de una acción puede modificar el estado físico, a través de actuadores, del entorno de la propia *Thing*.

2.3.2.3. Eventos

Por su parte, los eventos encapsulan típicamente una comunicación asíncrona entre la *Thing* y el usuario, donde lo que se transmite no son datos relacionados con estados sino con transiciones de estados de la propia *Thing*. Normalmente se utilizan para comunicar alarmas.

2.3.3. Bloques fundamentales

La propuesta de estandarización desarrollada por el W3C propone cuatro bloques fundamentales, o bloques constructivos, denominados *building blocks*, que son:

- *WoT Thing Description*.
- *WoT Binding Templates*.
- *WoT Scripting API*.
- *WoT Security and Privacy Guidelines*.

2.3.3.1. WoT Thing Description

En la arquitectura propuesta por el W3C, los metadatos deben constituir obligatoriamente lo que se denomina *WoT Thing Description* (TD) [65], que se expone como la interfaz de red de la *Thing* (*WoT Interface*). La TD se representa en formato JSON, o JSON-LD, y constituye una representación Web textual de la *Thing*, aunque esta representación no tiene por qué ser única y una *Thing* puede proporcionar otras representaciones, por ejemplo, basadas en HTML. La TD permite a los consumidores descubrir y conocer las características de una *Thing*, de tal forma que puedan determinar, entre otros, los protocolos y estructuras de datos a utilizar para interactuar con ella.

Para el caso de representaciones basadas en JSON, el W3C proporciona un esquema JSON que permite validar sintácticamente una *Thing Description*.[26]

En el listado siguiente se muestra un ejemplo de TD del objeto *MyLampThing*, que expone una propiedad, una acción y un evento.

```
"title": "MyLampThing",
"securityDefinitions": {
  "basic_sc": {"scheme": "basic", "in":"header"}
},
"security": ["basic_sc"],
"properties": {
  "status" : {
    "type": "string",
    "forms": [{"href": "https://mylamp.example.com/status"}]
  }
},
"actions": {
  "toggle" : {
    "forms": [{"href": "https://mylamp.example.com/toggle"}]
  }
},
"events":{
  "overheating":{
    "data": {"type": "string"},
    "forms": [{
      "href": "https://mylamp.example.com/oh",
      "subprotocol": "longpoll"
    }]
```

[26] https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/#json-schema-for-validation

```
    }
}
```

La propiedad, denominada `status`, proporciona acceso a un valor de tipo `string` a través del URI `https://mylamp.example.com/status`. Por defecto, una propiedad es accesible mediante el método GET, de ahí que no se indique específicamente en la TD. Por su parte, la acción `toggle`, accesible a través del URI `https://mylamp.example.com/toggle` requiere utilizar el método POST, aunque al igual que en el caso anterior, no se indica de forma explícita en la TD. El evento `overheating` define un mecanismo de comunicación a través de mensajes asíncronos, de tipo `string`, mediante una suscripción utilizando *HTTP Long Polling* a través del URI `https://mylamp.example.com/oh`. En esta se TD especifica, además, el mecanismo de seguridad que debe implementar el consumidor para poder acceder al objeto. En este caso se requiere un esquema de seguridad básico, `basic_sc`, que solicita la identificación mediante usuario y contraseña a través de *HTTP Basic Authentication*.

La interacción con una *Thing* se podría realizar, por ejemplo, utilizando el protocolo HTTP. Para ello, un consumidor puede utilizar las operaciones GET, para leer, PUT, para escribir, o POST para realizar una invocación.

La Figura 2.11 muestra un diagrama básico de interacción entre un consumidor y una *Thing*, así como el papel de la TD.

Figura 2.11 Interacción consumidor-*Thing*.

La TD es el bloque constructivo fundamental de la arquitectura propuesta por el W3C, y puede considerarse la puerta de entrada al sistema. En un símil con la Web, es como el recurso `index.html` de un sitio Web.

En el caso de que la configuración de la red no permita que el consumidor acceda directamente a la *Thing*, la arquitectura W3C WoT define el concepto de intermediario, que puede actuar como un *proxy* para las *Things* ocultas, por ejemplo, tras un *firewall* o una configuración de NAT (*Network Address Translation*). Desde el punto de vista de los consumidores, los intermediarios se comportan como las *Things*, dado que poseen una TD y proporcionan una interfaz. La Figura 2.12 muestra este concepto.

2.3.3.2. WoT Binding Templates

La propuesta del W3C no especifica qué protocolo se debe utilizar para la interacción entre los consumidores y las *Things*. Por tanto, el segundo de los bloques constructivos de la

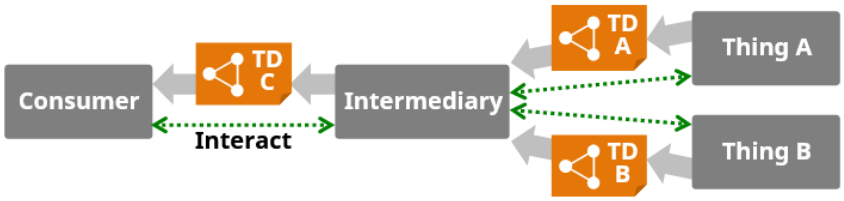


Figura 2.12 Intermediario entre consumidor y *Thing*.

arquitectura está constituido por plantillas de enlace, denominadas comúnmente *binding templates* [66]. Estas plantillas proporcionan mecanismos para definir cómo protocolos específicos tales como HTTP, CoAP, MQTT o Modbus se pueden mapear en el modelo de interacción propiedades-acción-evento. En suma, cada una de estas plantillas define cómo un cliente puede comunicarse con el modelo de interacción a través de la interfaz de red utilizando un protocolo específico.

Cada plantilla de enlace se compone de una serie de metadatos necesarios para llevar a cabo la comunicación:

- Plataforma: qué requisitos específicos tiene esta plantilla para comunicarse con una plataforma IoT concreta.
- Formato: qué tipo de formato siguen los datos que se envían y reciben en las transmisiones.
- Protocolo: qué protocolo de capa de aplicación se usará para establecer las comunicaciones.
- Subprotocolo: qué tipo de extensión es necesario usar con el protocolo definido.
- Seguridad: qué mecanismos de seguridad se aplican a dichos protocolos y la información necesaria para aplicarlos.

La Figura 2.13 muestra la relación entre *binding templates* y diferentes protocolos, o pilas de protocolos, que se pueden utilizar en soluciones WoT.

2.3.3.3. WoT Scripting API

El tercer bloque constructivo, que recibe el nombre de *scripting API* [67], es un componente opcional que se utiliza típicamente en *gateways* o navegadores Web para permiten extender la funcionalidad de la arquitectura mediante la integración de *scripts* en tiempo de ejecución, que permiten modificar el comportamiento de las *Things*. De esta forma, por ejemplo, se pueden mover tareas de procesamiento intensivo a un *gateway* local más potente o lógica crítica que requiera actuación en tiempo real hacia la propia *Thing*.

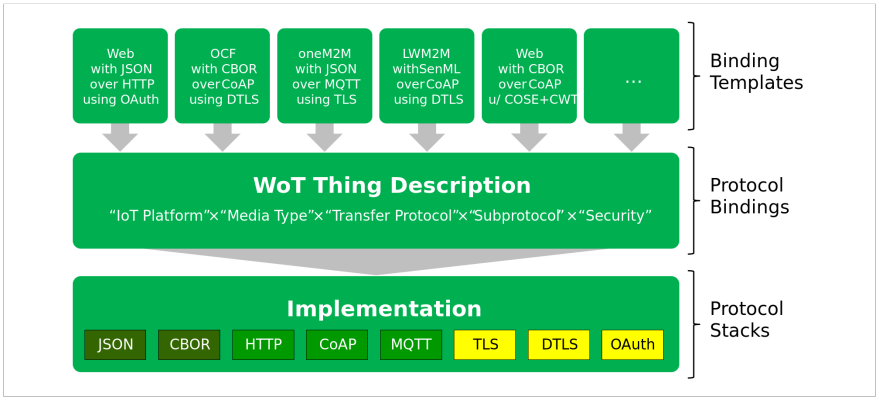


Figura 2.13 *Binding templates* para diferentes protocolos WoT [63].

2.3.3.4. WoT Security and Privacy Guidelines

El cuarto bloque constructivo de la arquitectura está relacionado con mecanismos de seguridad. Se denomina *WoT Security and privacy* [68] y recoge una serie de pautas para la implementación y la configuración de *Things* de forma segura. Aunque estas pautas son opcionales, la recomendación del W3C es que deben ser consideradas en todas las soluciones WoT. Estas pautas no describen nuevos mecanismos de seguridad, sino que se utilizan los ya existentes para otros modelos en la Web, a la vez que se especifica cómo aplicarlas a activos específicos de WoT tales como *Thing Descriptions* o *Scripting APIs*.

Capítulo 3

Arquitecturas IoT y WoT para edificios energéticamente eficientes

En el campo de IoT se han planteado diferentes arquitecturas, protocolos y aplicaciones con el objetivo de proporcionar interoperabilidad en soluciones que utilizan sistemas y dispositivos heterogéneos [69, 70]. Estas arquitecturas se alinean hacia la obtención de datos de dispositivos imbricados en el mundo físico, organizarlos, almacenarlos y proveerlos de la manera más conveniente posible a las aplicaciones finales de usuario o a aplicaciones de control inteligentes.

En este capítulo se revisará el estado del arte en cuanto a arquitecturas IoT y WoT relacionadas con el ámbito de este trabajo, definido en la Sección 1.2, y se describirá la arquitectura propuesta en esta tesis para dar soporte a sistemas de gestión de energía en edificios (BEMS) utilizando WoT como paradigma para la interoperabilidad de todos los dispositivos que lo componen.

3.1. Trabajo relacionado

A la hora de plantear una solución IoT para un sistema de control inteligente en un edificio existen varios enfoques. Cada edificio supone un reto distinto ya sea por diferencias de planos, dimensiones, uso, altura, número de espacios habitables (si es que hay alguno), etc. Por ejemplo, la sensorización y control de un edificio de oficinas, que solo se usará en las horas centrales del día, puede diferir de aquella necesaria para un edificio de viviendas, que tiene un uso continuado. En la literatura científica se puede encontrar información sobre cómo se estructuran estos sistemas habitualmente. A continuación, se revisan los trabajos más relevantes en este campo, en orden cronológico.

En Alwan *et al.* (2019) [71] se presenta una arquitectura software, de tipo *open source*, para el desarrollo de controladores de edificios inteligentes, o SBCT (*Smart Building Controller*). La implementación de esta arquitectura se basa, a su vez, en un proyecto *open source* denominado

openHAB[1]. La arquitectura de SBCT propuesta coordina diferentes unidades openHAB; cada una de estas unidades controla y monitoriza viviendas individuales (además, espacios singulares como tejados o jardines se pueden controlar también mediante unidades openHAB adicionales). El proyecto openHAB se basa en la arquitectura modular OSGi[2] y proporciona un bus de eventos para la comunicación entre dispositivos. Mediante la implementación de componentes adicionales, o *add-ons*, el proyecto permite integrar dispositivos, interfaces e incluso plataformas de terceros. Por tanto, el número de protocolos soportados en la arquitectura propuesta en base a esta solución es, potencialmente, ilimitado.

Esta arquitectura se construye en base a tres capas, tal y como se muestra en la Figura 3.1:

1. Sensores/actuadores: los diferentes sensores y actuadores repartidos en las distintas estancias del edificio. Son los responsables de enviar datos al sistema openHAb y este actúa sobre ellos en consecuencia.

2. *Middleware* openHAB: openHAB es un software de código abierto para la operación de múltiples dispositivos en un hogar inteligente. Provee reglas de actuación e interfaces gráficos para los usuarios finales, que también pueden ser gestionados desde las aplicaciones de capas superiores.

3. Capa de aplicación: desde la que el SBCT controla el almacenamiento de datos y el sistema de gestión de procesos, o BPM (*Business Process Manage*). Para ello, proporciona un motor de reglas de alto nivel que combina los resultados de las reglas de actuación enviadas por las unidades openHAB de la capa inferior.

En Chevallier *et al.* (2020) [72] se presenta un sistema de monitorización de edificios orientado a la creación de un gemelo digital cuyo objetivo es facilitar el desarrollo de aplicaciones de alto nivel, solucionando las limitaciones impuestas por silos IoT al utilizar soluciones parciales de diferentes fabricantes o proveedores. Mediante este gemelo digital se describen las estructuras semánticas estáticas y las series de datos temporales proporcionadas por los sensores. Esta arquitectura propone el uso de una base de datos *triplestore*, o almacén de RDF (*Resource Description Framework*), a la que se accede mediante consultas semánticas. Además, se presenta un caso de uso en el que el servidor se implementa en Node-RED[3] (que tiene la particularidad de soportar conectores para múltiples protocolos IoT así como una API basada en WebServices que permite modificar los flujos de datos entre los sensores y las capas superiores) y la base de datos mediante Apache Cassandra[4].

Esta arquitectura de gemelo digital, que además permite realizar simulaciones en el edificio, se basa en cinco capas o niveles, tal y como se puede observar en la Figura 3.2:

[1]https://www.openhab.org/
[2]https://www.osgi.org/
[3]https://nodered.org/
[4]https://cassandra.apache.org/

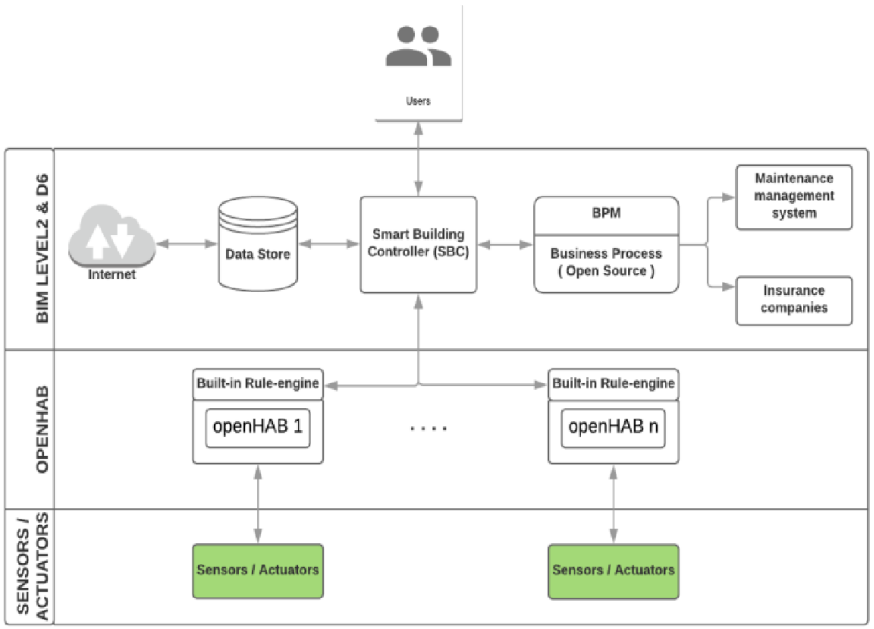


Figura 3.1 Arquitectura de un SBCT basada en openHAB [71].

1. Construcción de un modelo de datos basado en RDF.
2. Enriquecimiento y consistencia de datos a la hora de integrar los múltiples sensores que componen el sistema.
3. Flujo y almacenamiento de datos, o cómo se obtienen y almacenan los datos.
4. Sistema y tiempo de ejecución, o el proceso que se sigue para obtener la información.
5. Aplicaciones para los usuarios finales.

En Lagsaiar *et al.* (2021) [73] se presenta una arquitectura para un sistema de control inteligente de edificios, cuya implementación se puede ejecutar en un servidor local de tipo SBC (*Single-Board Computer*), que se puede conectar a una plataforma central en la nube. Utiliza una base de datos relacional, en la que las relaciones entre las tablas se basan en una clave especial con el objetivo de organizar los datos según una estructura semántica. El servidor propuesto se implementa en Node.js[5], seleccionado por sus posibilidades de escalabilidad. Para demostrar la viabilidad de esta arquitectura, se presenta además un caso de estudio en un edificio de 15 viviendas, en las que se dispone de sensores de temperatura y humedad, medidores de energía y medidores de caudal de agua. Esta arquitectura se organiza en cuatro capas, tal y como muestra la Figura 3.3:

1. Capa física: es la capa en la que se alojan los sensores y actuadores. Esta arquitectura soporta tres tipos de sensores: confort (que monitorizan, entre otros parámetros, la

[5]https://nodejs.org/

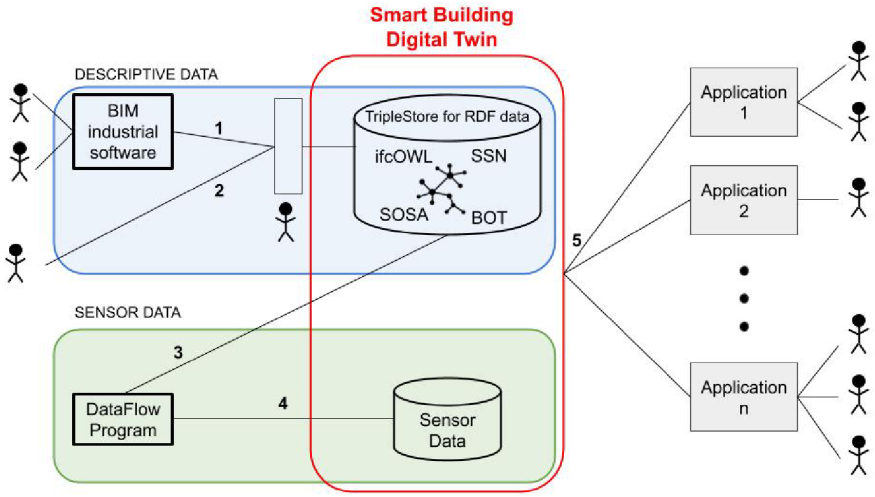


Figura 3.2 Arquitectura IoT para un edificio inteligente, basada en un gemelo digital [72].

calidad del aire), consumo (de electricidad y de volumen de agua) y seguridad (presencia, apertura de puertas y ventanas, y detección de humos). Y tres tipos de actuadores: control del flujo de agua en radiadores, control de iluminación y control de cargas en enchufes.

2. Capa de transmisión de datos: esta capa se encarga de las comunicaciones, o cómo el dato obtenido de un sensor o la orden enviada a un actuador llega a su destino, garantizando la conexión entre los sensores y actuadores con el servidor local. Para esta comunicación de corto alcance, esta arquitectura soporta varios protocolos, tales como NFC, BLE, Z-Wave, Wi-Fi y Zigbee. También se encarga de gestionar la comunicación entre el servidor local y la plataforma central en la nube, para lo que soporta protocolos como LTE, NB-IoT, LoRA y Sigfox.

3. Capa de gestión de datos: compone los sistemas, programas o bases de datos que se encargan de gestionar y almacenar los datos adquiridos o enviados a los sensores/actuadores a través de la capa de transmisión. Constituye el núcleo de la arquitectura al ser la pasarela entre los dispositivos sensores y actuadores y los servicios proporcionados por el sistema.

4. Capa de servicios: compuesta por todos aquellos servicios que permiten a usuarios finales, gestores u otras máquinas acceder a la información proporcionada por la capa de gestión de datos. Es la capa responsable de la visualización, tanto a nivel histórico como en tiempo real, así como del control de todos los dispositivos del edificio. Por último, gestiona el acceso al servidor local por parte de dos tipos de usuarios, con roles claramente diferenciados: los ocupantes de las viviendas y los administradores del sistema.

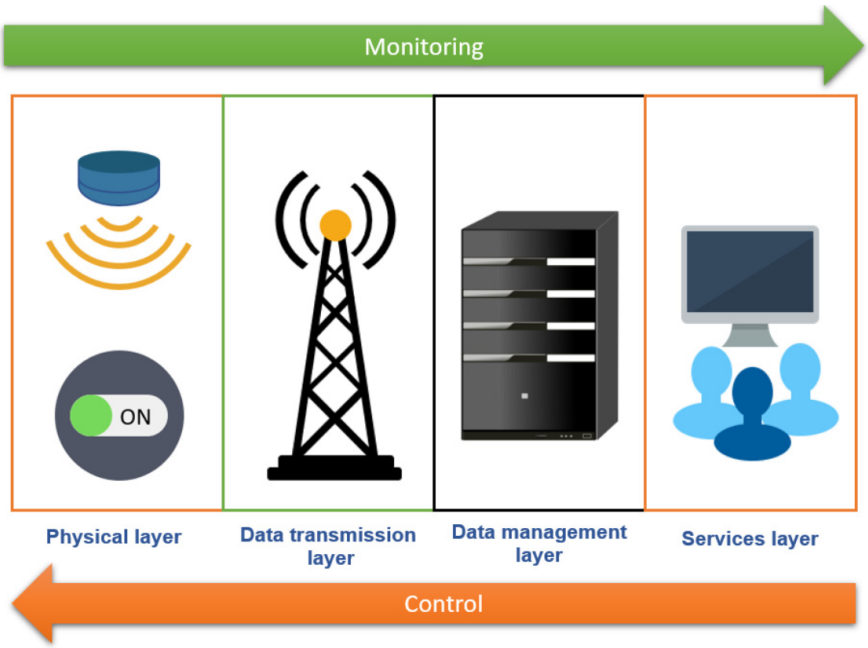


Figura 3.3 Arquitectura de un BEMS basada en un diseño modular [73].

En Ruiz *et al.* (2022) [20] se presenta una arquitectura para la implementación de un BEMS basado en el paradigma *Big Data*. Este paradigma se usa, además de para la fase de adquisición de datos, en la implementación de algoritmos capaces de analizar cantidades masivas de datos provenientes de dispositivos heterogéneos. Esta arquitectura tiene como objetivo gestionar de forma eficiente las operaciones en instalaciones tales como edificios, centros comerciales, o aeropuertos, para los que además presentan varios casos de uso en escenarios reales. El despliegue se realiza utilizando la distribución *Cloudera Manager*[6], que permite la instalación y posterior configuración del sistema mediante una interfaz Web, proporciona soporte para almacenamiento a través de HDFS y MongoDB[7], e implementa analíticas de datos utilizando Apache Spark[8]. Esta arquitectura se basa en tres capas, como se puede observar en la Figura 3.4:

1. Adquisición y transporte de datos: esta capa se encarga de gestionar todos los datos que serán utilizados en el resto de los módulos de la arquitectura. Con el objetivo de proporcionar una arquitectura lo más genérica posible, en esta capa el almacenamiento de datos se puede realizar en bases de datos relacionales, noSQL, o RDF. Aunque de forma particular, el almacenamiento de los datos provenientes de sensores se realiza típicamente en MongoDB y los datos ya procesados se almacenan utilizando HDFS

[6]https://www.cloudera.com/
[7]https://www.mongodb.com/
[8]https://spark.apache.org/

(*Hadoop Distributed File System*), dado que se adapta mejor a las técnicas de minería de datos o de aprendizaje automático que se utilizarán en capas superiores.

2. Unidad de control: se encarga de gestionar la operación del día a día del sistema, principalmente la que tiene que ver con los sistemas de calefacción y aire acondicionado. Se compone de dos estrategias diferenciadas: un modelo de control predictivo y un modelo de control bajo demanda. El modelo de control predictivo trata de obtener un funcionamiento óptimo del sistema en base a predicciones meteorológicas, nivel de ocupación de las estancias del edificio, del estado de diferentes componentes del edificio y de las tarifas energéticas, entre otros datos. Por otro lado, el modelo bajo de manda trata de optimizar el funcionamiento teniendo en cuenta las configuraciones particulares del usuario.

3. Análisis de datos y toma de decisiones: en esta capa se utiliza Apache Spark como plataforma para la implementación de las técnicas de minería de datos, tanto de datos históricos como de datos adquiridos en tiempo real, soportadas por la solución.

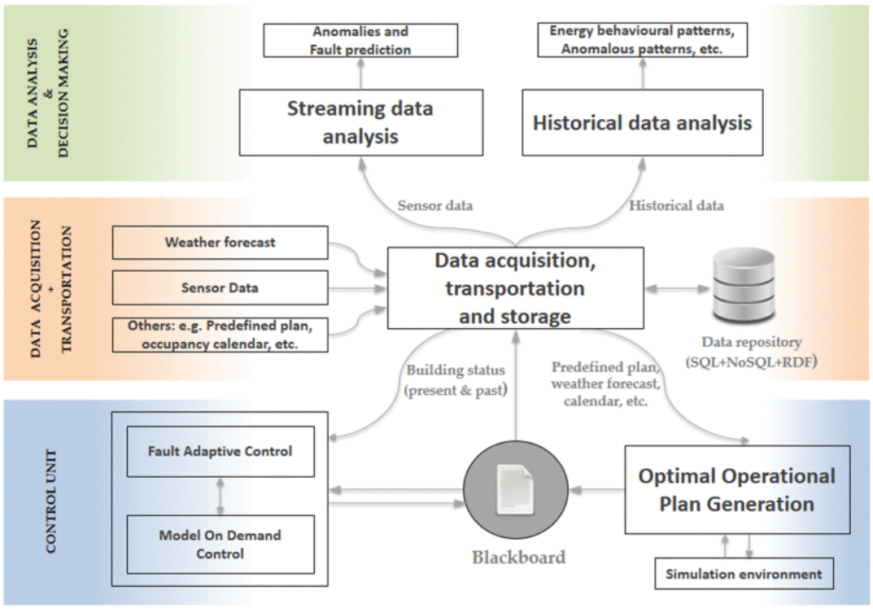


Figura 3.4 Arquitectura de un BEMS basada en el paradigma *Big Data* [20].

Si bien en la literatura científica se recogen más propuestas de arquitecturas de redes de sensores y actuadores, algunas de ellas utilizando estándares IoT, estas se suelen centrar en espacios más reducidos, tales como viviendas, siendo soluciones apropiadas para el ámbito de *smart homes* en lugar de para la monitorización y el control en edificios completos. Este tipo de sistemas suelen recibir el nombre de HEMS, o *Home Energy Management System*. En Beaudin y Zareipour (2015) [74] se presenta un análisis comparativo de las soluciones HEMS propuestas en la literatura científica en el que se revisan aspectos tales como el manejo de dispositivos

heterogéneos y el uso de recursos computacionales. Con posterioridad a este análisis se han desarrollado nuevas soluciones, entre las que destacan la de Al-Ali *et al.* (2017) [75], en la que se propone una plataforma IoT combinada con *frameworks* de analítica de datos de terceros para la gestión del consumo de sistemas de climatización en una vivienda; la propuesta por Balikhina *et al.* [76], en la que se presenta la arquitectura para la monitorización de los dispositivos conectados en una vivienda así como un caso de uso desplegado en la plataforma AWS IoT (*Amazon Web Services*); y la arquitectura propuesta por Lin (2019) [77], que aplica técnicas de DSM (*Demand-Side Management*) junto con el paradigma IoT para controlar el consumo de energía de todos los electrodomésticos de una vivienda.

En las arquitecturas relacionadas con el trabajo de esta tesis, independientemente del número de capas o niveles que presenten, existen dos sentidos en los que puede discurrir el flujo de datos:

- Monitorización: permite la consulta y obtención de datos de los sensores de la capa física. Estos datos se envían a través de módulos de transmisión de datos a un ente local o remoto, en el que se alojan o al que se conectan módulos de gestión de datos. Finalmente, estos datos se exponen desde el citado ente, ya sea de manera gráfica para un usuario final o de manera programática para que sean consumidos por otros sistemas.
- Control: permite a los usuarios finales, o a un sistema de control (inteligente), controlar los distintos actuadores presentes en la capa física. Para ello se otorga, a través de módulos de servicios, un control sobre los elementos actuables que componen el sistema. Estas órdenes se gestionan en los módulos de gestión de datos, donde se comprueba que la orden invocada va asociada al actuador correcto. Finalmente, se envían a través de módulos de transmisión o comunicación hasta la capa física, donde se ejecuta la acción invocada.

De las arquitecturas propuestas en el trabajo relacionado se puede identificar que una arquitectura para el control inteligente de un edificio debe estar constituida, al menos, por las tres siguientes capas:

1. Capa física: compuesta por los elementos de muestreo y actuación del sistema.
2. Capa de comunicaciones: compuesta por los elementos que permiten comunicar los sensores y actuadores con los controladores.
3. Capa de gestión de datos: compuesta por los elementos que exponen y/o gestionan los datos adquiridos en la capa física.

3.2. Arquitectura propuesta

En base a la revisión del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con IoT y con WoT desarrollada en el Capítulo 2, que proporciona

una visión del estado actual en cuanto a las posibilidades de interconexión de dispositivos heterogéneos, así como al análisis del trabajo relacionado sobre arquitecturas propuestas para la implementación de soluciones en el ámbito del control eficiente en edificios realizada en la Sección 3.1, se puede concluir que, hasta donde alcanza nuestro conocimiento, no existe en la actualidad una propuesta de arquitectura de este tipo que cumpla con la recomendación *WoT Architecture* del W3C [63]. Por ello, en esta sección se presenta la propuesta de una arquitectura para el soporte de sistemas de gestión de energía en edificios mediante el paradigma *Web of Things*, con el objetivo de proporcionar interoperabilidad a todos los dispositivos del sistema a través de la capa de aplicación mediante el uso de estándares y tecnologías Web siguiendo las recomendaciones del W3C relacionadas con WoT. Es preciso mencionar que dado que el diseño y la implementación de mecanismos de seguridad *end-to-end* en entornos IoT supone un reto de investigación en sí mismo, esta tarea será incluida como línea de trabajo futura (ver Sección 5.3) y no se abordará en la propuesta actual.

El gestor de energía de un edificio es un sistema que se encarga de las operaciones lógicas que controlan y distribuyen el flujo de energía eléctrica y térmica, así como el flujo de datos, coordinando todos los dispositivos del edificio. Entre otros, estos dispositivos pueden ser: dispositivos de tipo MIMO (*Multiple Input Multiple Output*); bombas de calor, o HP (*Heat Pump*); ventiloconvectores, o FC (*Fan Coil*); tanques de almacenamiento de agua caliente, o DHW (*Domestic Hot Water*); dispositivos de almacenamiento por materiales de cambio de fase, o PCM (*Phase-Change Material*); y placas fotovoltaicas. Como se muestra en la Figura 3.5, los principales componentes de un BEMS interaccionan entre ellos en términos de energía eléctrica, energía térmica y datos. Esta representación es lo suficientemente genérica como para incluir cualquier BEMS, no sólo los que serán objeto de estudio en esta tesis. En esta sección se describen los dispositivos de comunicación y control involucrados en la operación del sistema, con un enfoque basado en el flujo de datos.

La idea principal en un BEMS es que toda la información que pueda ser obtenida de los aparatos de climatización en un edificio se puede combinar con los datos adquiridos de los sistemas de energía, ventilación y bombas de agua para la creación de un modelo de consumo energético con el que se pueda optimizar, a través de tareas automatizadas en el sistema, la gestión energética del edificio. En Ren y Cao (2019) [78] y en Zhang *et al.* (2019) [79] se trata en profundidad el potencial que tiene la obtención de los datos ambientales recabados en el interior de edificios para la consecución de estas tareas. Aunque el propósito de los datos adquiridos en base a la arquitectura propuesta es, efectivamente, servir estos objetivos, los esfuerzos en esta tesis se centran en la recolección de los datos, su agregación y su exposición, todo ello mediante tecnologías Web estándares siguiendo las recomendaciones del W3C, para que otros subsistemas, u otros sistemas externos, puedan consumirlos y explotarlos.

La arquitectura propuesta en esta tesis trata de proporcionar una solución basada en WoT a un problema de sensorización de un edificio para dotarlo de capacidades de monitorización y control de parámetros ambientales. Los dispositivos presentes en el mismo, como pueden

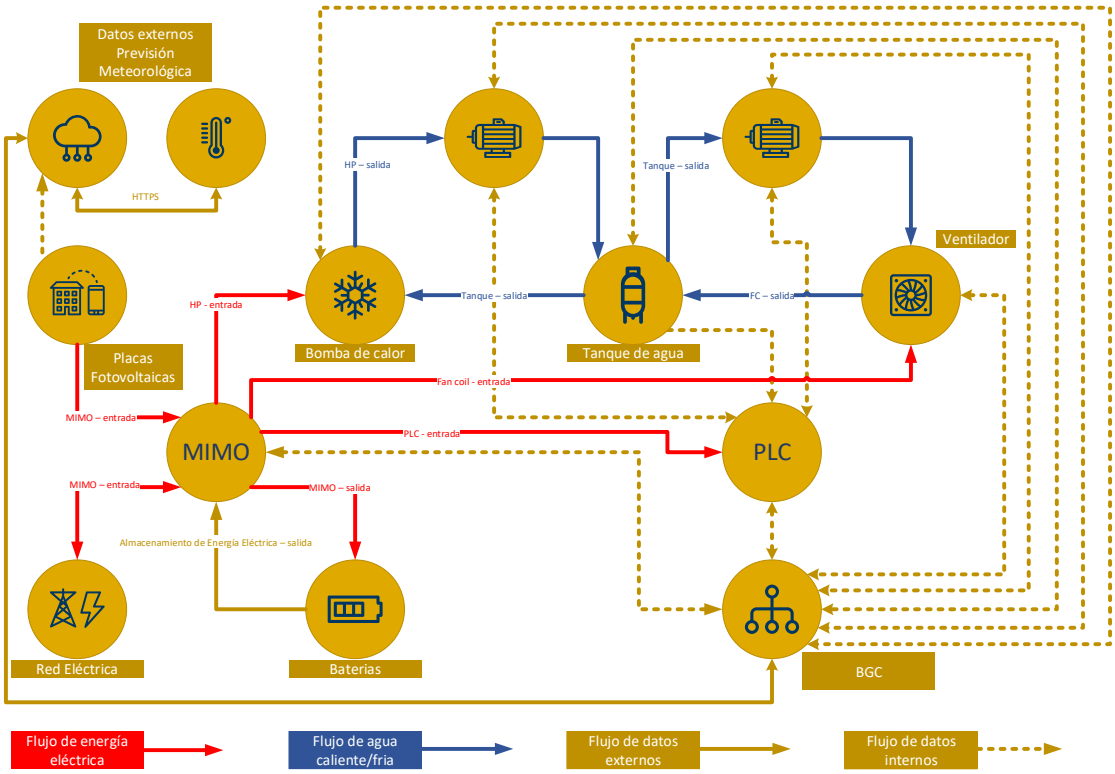


Figura 3.5 Arquitectura genérica de un BEMS.

ser bombas de calor, medidores de energía, aparatos de aire acondicionado o calefactores, serán sensorizados y monitorizados en base a esta arquitectura. A raíz del análisis realizado en la Sección 3.1, se propone una arquitectura de tres niveles, de la que se puede observar un ejemplo simplificado en la Figura 3.6. En cada nivel se define una serie de funcionalidades, requisitos e interfaces que enlazan con las recomendaciones WoT del W3C, de la siguiente manera:

- Nivel 1: se asocia con el modelo de interacción, dotándolo de las herramientas oportunas para la comunicación con los sensores y actuadores así como con el mundo real.
- Nivel 2: se relaciona tanto con el modelo de interacción, definiendo las propiedades de las *Things*, como con los *Protocol Bindings*, definiendo cómo interactuar con esas propiedades.
- Nivel 3: está ligado con la parte cliente de las recomendaciones W3C WoT, y utiliza las interacciones ya definidas para acceder a los datos.

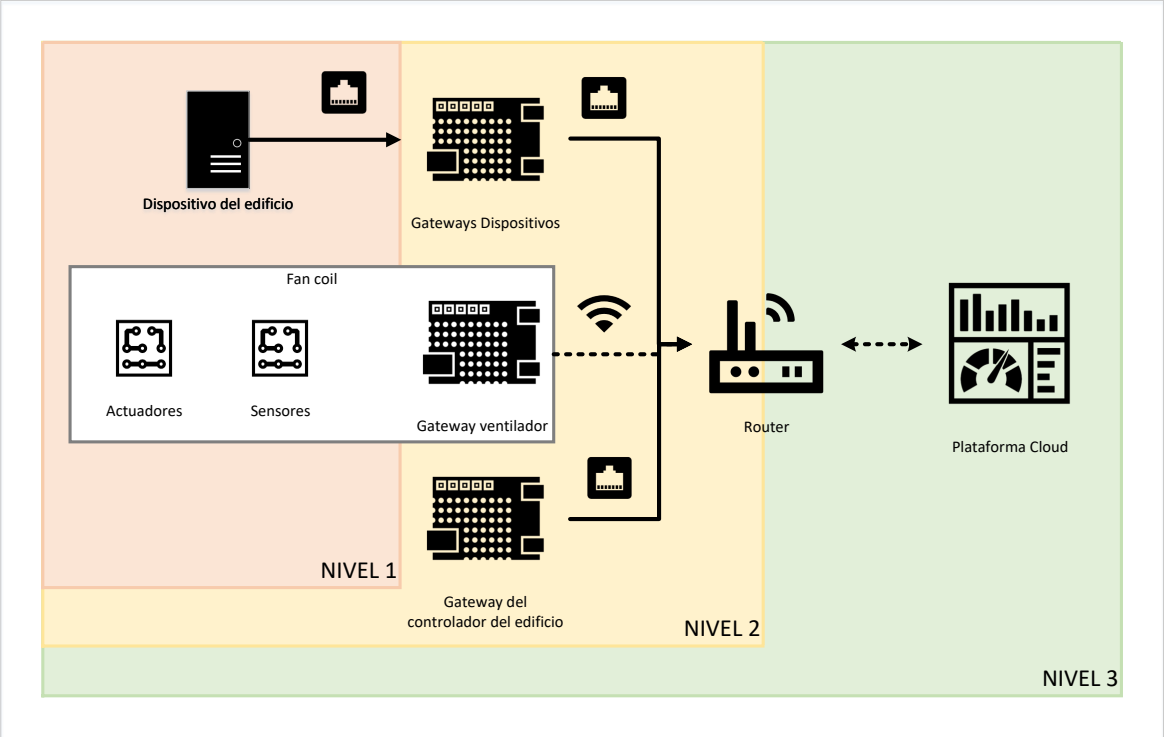


Figura 3.6 Arquitectura propuesta de tres niveles: ejemplo simple.

3.2.1. Arquitectura por niveles

En esta sección se describirá en detalle cada uno de los tres niveles que componen la arquitectura propuesta en esta tesis.

3.2.1.1. Nivel 1: sensores y actuadores

En este nivel se alojan los sensores que capturan los datos de los diferentes elementos del sistema, muestreando variables del entorno, así como de los actuadores que las modifican. Estos dispositivos, sensores y actuadores, se verán representados por el modelo de interacción, que incluye las propiedades y acciones que define el W3C en sus recomendaciones para WoT. Cada uno de estos dispositivos se representará mediante una *Thing* WoT, en la que cada propiedad es una representación del estado de la *Thing* que puede ser leída y, en ciertos casos, modificada. También es donde se tratan las acciones, que permiten activar ciertos comportamientos en la *Thing*. Estas acciones son interacciones que permiten ejecutar un proceso dentro de la *Thing*, usando propiedades que quizás no están directamente expuestas, como se ha visto previamente. La invocación de una acción o la escritura de una propiedad puede suponer cambios físicos en la *Thing* objetivo y en su entorno.

En este nivel se definen los componentes hardware que serán necesarios tanto para la toma de datos, como para la lectura y exposición de los mismos. Para ello se requeire de

unos dispositivos hardware capaces de interactuar con el entorno, que deben cumplir unas características determinadas.

Los dispositivos han de disponer de interfaces capaces de comunicarse con todo tipo de sensores y actuadores mediante conexiones analógicas y/o digitales. En este nivel, los datos adquiridos pueden ser filtrados y procesados previamente, utilizando por ejemplo métodos estadísticos para la eliminación de espurios, con el objetivo de proporcionar lecturas precisas. Además de obtener y enviar los datos de los sensores y actuadores, los dispositivos en este nivel han de ser capaces de procesarlos y almacenarlos de manera ordenada. Estas funcionalidades se integran con las recomendaciones WoT del W3C siendo parte del modelo de interacción.

Tal y como se ha descrito en la Sección 2.3.2, el modelo de interacción se compone de tres elementos básicos: propiedades, acciones y eventos. Las propiedades enlazan con lo descrito en este Nivel 1, en el que se asocian directamente o mediante transformaciones a los datos leídos de los sensores y actuadores. Las acciones se refieren normalmente a una interacción con los actuadores, pero también pueden implicar la lectura de sensores o el cambio de parámetros o funciones internas. Los eventos se asocian con cambios, umbrales o alarmas que acontecen en la *Thing*, provocando un evento asociado que es capturado por un cliente.

En la Figura 3.7 se observa este nivel, asociado a los dispositivos sensores y actuadores del edificio. Estos se comunican con sus respectivos *gateways* mediante protocolos de comunicación internos, es decir, cercanos, no protocolos de red. Esta interfaz ofrece lectura y actuación sobre los dispositivos, además de requerir, en muchos casos, un sistema de alimentación eléctrica.

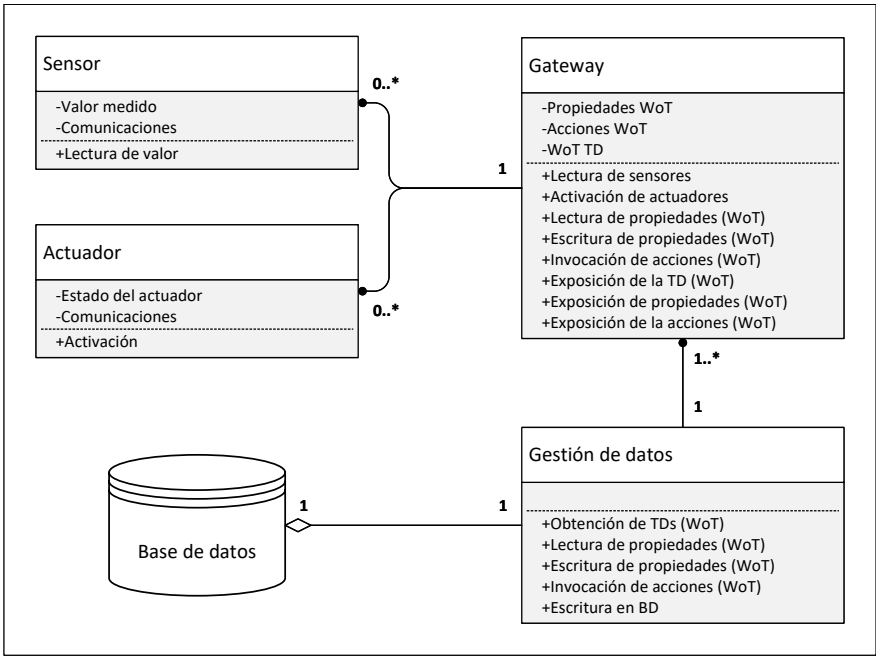


Figura 3.7 Diagrama lógico de la arquitectura propuesta.

3.2.1.2. Nivel 2: comunicaciones

El Nivel 2 se encarga de las interacciones con las *Things* y con los protocolos que permiten el acceso a los distintos recursos. Este nivel se ve representado dentro de las recomendaciones WoT del W3C en los *Protocol Bindings* que definen los mensajes, el protocolo de red, el tipo de mensaje y su contenido. De manera genérica se define dentro de las recomendaciones del W3C que las interacciones deben tener un conjunto de métodos predefinidos y conocidos, como los que se encuentran en el protocolo HTTP REST [80]. Esta serie de protocolos se definen dentro de las *Things*, y se usan a la hora de obtener los datos. Los diferentes dispositivos, que exponen varias propiedades y acciones, tendrán un protocolo de comunicaciones en común que permite un método de acceso uniforme entre todos los dispositivos que componen la red del edificio.

En este nivel aparece un actor denominado *Building Gateway and Controller* (BGC). Este dispositivo provee la monitorización, el control en tiempo real y otras funciones básicas que enlazan con el siguiente nivel.

El software de los componentes principales del edificio que se han revisado hasta el momento (BGC, sensores y actuadores), implementa lo que en las recomendaciones del W3C se denomina *servient* (ver Sección 2.3). Los *servients* tienen la capacidad de exponer *Things*, en modo servidor, y de consumirlas, en modo cliente. Las recomendaciones del W3C definen una serie de *Application Programming Interfaces* (APIs) que permiten acceder a los diferentes dispositivos mediante HTTP REST implementado en sus interfaces de comunicación.

La integración de las recomendaciones WoT del W3C requiere de la capacidad por parte de los componentes del edificio de implementar ciertas lógicas y proporcionar soporte a varios protocolos de comunicación. El W3C se refiere a esto como *Protocol Bindings*, que establecen los canales de comunicación definidos en la TD, especificando el formato y contenido de los mensajes que circularán entre los dispositivos. Los dispositivos seguirán tales normas a la hora de comunicarse entre sí y con otros clientes a través de una interfaz de red.

Para cumplir las recomendaciones del W3C en relación con una arquitectura WoT, los protocolos utilizados deben tener asociado un esquema de URIs (*Universal Resource Identifiers*), que permita la identificación unívoca de los recursos dentro de la red. Estos se enlazan con los controles de hypermedia para identificar enlaces y destinos de envío de mensajes. Los protocolos, que se definen como de transferencia, deben basarse en un conjunto de métodos conocidos a priori, para que los mensajes sean autodescriptivos, permitiendo así el uso de intermediarios (*proxies*) y reutilizando protocolos ampliamente conocidos, como por ejemplo HTTP, MQTT o CoAP.

En esta arquitectura, todas las opciones de interacción con una *Thing* han de ir acompañadas de un control de hypermedia. Esto se refiere a un enlace o un formulario, (*form*), mediante el cual sea posible interactuar con la *Thing*. Estos han de ser descriptivos, de forma que se identifique claramente cómo activar las posibles interacciones. Las opciones se crearán

dentro de la TD por parte de la propia *Thing*, basándose en su estado actual e incluso en parámetros de la red, como por ejemplo su dirección IP. También pueden emanar de un componente externo que conozca las propiedades de la *Thing* y genere una TD por su cuenta para representarla y proporcionarle un modelo de interacción.

En las Figuras 3.6 y 3.7 se observa este nivel, así como los dispositivos o *gateways* del edificio que, de manera genérica, implementan las distintas partes mencionadas anteriormente. También se asocia con estos dispositivos la interfaz WoT, que proveerá de las herramientas necesarias para interactuar de manera acorde, ofreciendo una interfaz HTTP conforme a las recomendaciones del W3C. Esta interfaz hace uso de la interfaz descrita en el Nivel 1 para acceder a los sensores y actuadores asociados con las distintas opciones de interacción.

3.2.1.3. Nivel 3: gestión de datos

El nivel de gestión de datos tiene la función de recabar todas las propiedades expuestas por las *Things* que componen el sistema y gestionarlas para su posterior envío a otros servicios ya sean tanto locales como remotos. La recolección de datos se realiza de manera directa hacia las diferentes *Things* presentes en el sistema y definidas en el Nivel 2, usando las interacciones definidas en ese mismo nivel. El Nivel 3 se asocia en las arquitecturas IoT con la gestión de datos o capa de servicios, y es la que más cerca se encuentra del usuario final.

El almacenamiento en la nube de los datos obtenidos es una tarea clave, ya que permite su acceso y modificación desde cualquier dispositivo al que se le facilite acceso. Los servicios implementados en este nivel recaban los datos de las *Things* y también otros datos externos, como por ejemplo datos meteorológicos, integrándose de esta manera en la arquitectura. Además, permite conectar servicios externos de minería y analítica de datos, tanto de datos adquiridos en tiempo real como de datos históricos. Este tipo de servicios permite dotar de inteligencia al sistema, utilizando por ejemplo motores de reglas que actúen sobre las *Things* en función de los datos almacenados o cualquier otra estrategia heurística de más alto nivel.

El proceso de obtención de los datos se realiza a través de las interacciones definidas en el Nivel 2, accediendo a través del modelo de interacción proporcionados por la TD de cada *Thing* a los distintos recursos. Para ello, se dispondrá de una interfaz WoT capaz de interactuar con las TD y con los modelos de interacción de las *Things* distribuidas por el edificio.

Estos datos se almacenan en una base de datos de series temporales, o TSDB (*Time Series Database*), es decir, una base de datos optimizada para el almacenamiento y la recuperación de pares de tipo *tiempo-valor*. Estos pares están agrupados por categorías y con un *timestamp* de entrada que permite ordenar los datos en el tiempo. En esta base de datos, la URI que identifica cada una de las propiedades de una *Thing* se utiliza como etiqueta. Para cada medida se almacenará el instante temporal en que se realiza la medida, el valor de esta, su unidad y cualquier otro valor relevante que pudiese ser preciso para cualquier tarea de

automatización, mantenimiento o ajustes predictivos, entre otros, que permitan una gestión eficiente del edificio en el que se despliegue el sistema. La arquitectura propuesta soporta que la base de datos se encuentre alojada en un servicio externo (nube pública) o en un servidor en el propio edificio (a modo de nube privada).

Además, como interacción con el usuario final, en la arquitectura propuesta se pueden conectar paneles de control a los que usuarios y administradores del sistema podrán acceder para visualizar mediciones en tiempo real, históricos de datos, así como obtener resúmenes estadísticos y para configurar la operación del propio sistema.

En la Figura 3.7 se puede observar cómo este nivel de gestión de datos interactúa con el Nivel 2 utilizando las interfaces de las que dispone para obtener las TDs, las propiedades y ejecutar las acciones desde una perspectiva de usuario. Además, incluye la persistencia en la base de datos de la información recogida. Será a través de estas interfaces desde las que se podrá enviar y obtener datos hacia y desde la nube, aumentando así la capacidad de computación y almacenamiento del sistema.

3.2.2. Arquitectura de red

Esta sección especifica el hardware necesario para desplegar una red privada local para un BEMS, así como la comunicación con el resto de los componentes del edificio. Esta red local se basa en el estándar IEEE 802.11 (wifi) para conexiones inalámbricas y en el estándar IEEE 802.3 (Ethernet) para las conexiones cableadas. El BGC se encarga de activar y monitorizar el tipo de conectividad física de los dispositivos, ya sea cableado o inalámbrico. Para evitar vulnerabilidades y maximizar la confiabilidad y el tiempo de actividad del sistema, la red local debe implementar las siguientes características:

- SSID: el identificador de servicio de la red, o *Service Set Identifier*, se de configurar como oculto para dificultar que se pueda descubrir. Además, debe ser fijo para proteger el acceso no autorizado a la red mediante el uso de una contraseña previamente compartida.
- Los componentes centrales de la red del edificio deben soportar, al menos, el estándar inalámbrico IEEE 802.11b/g/n, y se conectarán a la red de área local inalámbrica en la banda de 2,4 GHz para garantizar el máximo rendimiento de la señal.

Para diseñar la mejor configuración para la red local inalámbrica se deben considerar tres aspectos clave: cobertura, capacidad, rendimiento e instalación.

- La cobertura es el área donde los dispositivos wifi podrán conectarse a la red. Dado que la red local pretende cubrir todos los espacios de trabajo o residenciales del edificio, esta variable depende de la configuración física del edificio (dimensiones y plano de las oficinas y/o viviendas, espacios comunes, paredes, etc.) Por este motivo, y para minimizar el número de puntos de acceso, la arquitectura propone, aunque no limita, el uso de la banda de 2,4 GHz.

- La capacidad es la habilidad de cada punto de acceso inalámbrico para manejar cierta cantidad de dispositivos conectados a la red. El diseño de la red debe tener en cuenta un número mínimo de sensores, actuadores y dispositivos distribuidos dentro del edificio.
- El rendimiento de la red implica un mínimo de ancho de banda disponible suficiente para que los dispositivos y aplicaciones realicen sus tareas y no se vean penalizados por la latencia o la saturación de la red. Es importante seleccionar tecnologías y sistemas que ofrezcan los más altos niveles de rendimiento y escalabilidad.

En la Figura 3.8 se puede observar la arquitectura de red propuesta, de forma genérica, para el BEMS de un edificio.

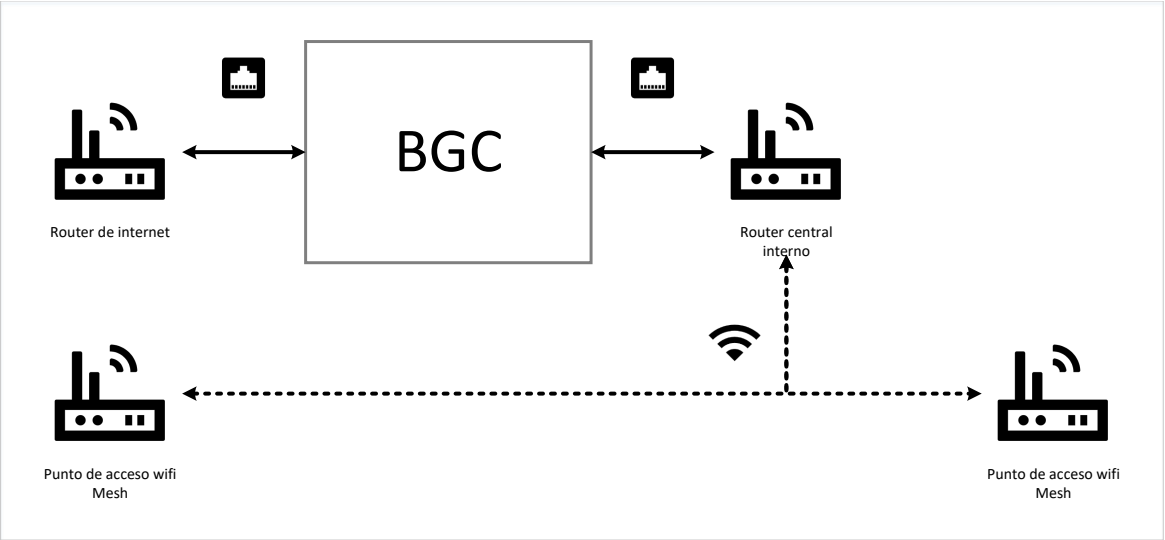


Figura 3.8 Arquitectura genérica de red propuesta para el BEMS.

El *router* de Internet se vinculará directamente con el BGC, actuando como un *proxy* de comunicaciones para todos los dispositivos del edificio (es decir, sensores y actuadores finales). Este dispositivo es el único que tiene acceso a Internet, para minimizar el número de vectores de ataque del sistema aumentando con ello la seguridad de los dispositivos del edificio. Para gestionar la red y la configuración de los dispositivos del edificio, se propone la utilización de un *router* interno y un *switch*, que pueden estar implementados en el mismo dispositivo. El *router* debe implementar un servidor DHCP con el objetivo de asignar recursos y proporcionar direcciones IP a todos los dispositivos distribuidos por el edificio. La misión del *switch* es proporcionar conectividad cableada suficiente para todos los dispositivos que lo requieran. La red inalámbrica se deberá implementar mediante una topología en malla, de tipo wifi *mesh*. El número de puntos de acceso wifi *mesh* vendrá determinado por la configuración del edificio.

3.2.3. Building Gateway and Controller

El *Building Gateway and Controller* (BGC) es el componente central del BEMS del edificio. Está conectado con el resto de los dispositivos y se encarga de implementar, en el Nivel 2, las operaciones lógicas de control. Su configuración es dinámica, ya que recibirá información de configuración de servicios externos. Actúa como *proxy* entre los dispositivos sensores y actuadores desplegados en el edificio y los servicios externos, como la plataforma de control en la nube. También actúa como puerta de enlace entre el edificio e Internet, permitiendo un acceso de banda ancha directo y confiable a la plataforma en la nube, convirtiendo protocolos, adaptando formatos de paquetes y traduciendo mensajes entre redes.

3.2.4. Ventiladores

Los electroventiladores, o *fan coils* (FC), son los dispositivos que permiten el control de la temperatura ambiental en los espacios controlados del edificio. En esta arquitectura se propone el uso de FCs con capacidades de interconexión inalámbrica mediante wifi, y se ímplementan interfaces wifi específicas para cuando no las tengan. Tal y como se muestra en la Figura 3.9, los FCs operan con el BEMS de la siguiente forma:

- Intercambiando datos: los FCs envían el estado de parámetros internos y reciben órdenes de control, a través de un dispositivo denominado FC *gateway*.
- Obteniendo energía: los FCs reciben energía eléctrica de entrada constante a través del MIMO.
- Compartiendo energía térmica: los FCs reciben un flujo de agua de la sala de calderas.

A partir de los datos proporcionados por los FCs, el sistema puede obtener la energía térmica instantánea. La energía térmica es la diferencia entre la temperatura del agua de entrada y la temperatura del agua de salida en correlación con el flujo de agua, lo que se traduce en una cantidad total de energía utilizada en el FC. Se puede obtener el valor de la energía térmica utilizada en los FC expresado como un valor instantáneo. El consumo de energía térmica, en kWh, para el circuito de agua caliente, E_w^h, y para el circuito de agua fría, E_w^c, se puede calcular mediante las expresiones Eq. 3.1 y Eq. 3.2, respectivamente, donde $FlowR_w$ es el caudal total de agua a través del FC durante un cierto período (l/min), SH es la constante de calor específico del agua (0,0011627), Tin_w es la temperatura de entrada del agua en el FC (°C) y $Tout_w$ es la temperatura de salida del agua en el FC.

$$E_w^h = FlowR_w \times SH \times (Tin_w - Tout_w) \tag{3.1}$$

$$E_w^c = FlowR_w \times SH \times (Tout_w - Tin_w) \tag{3.2}$$

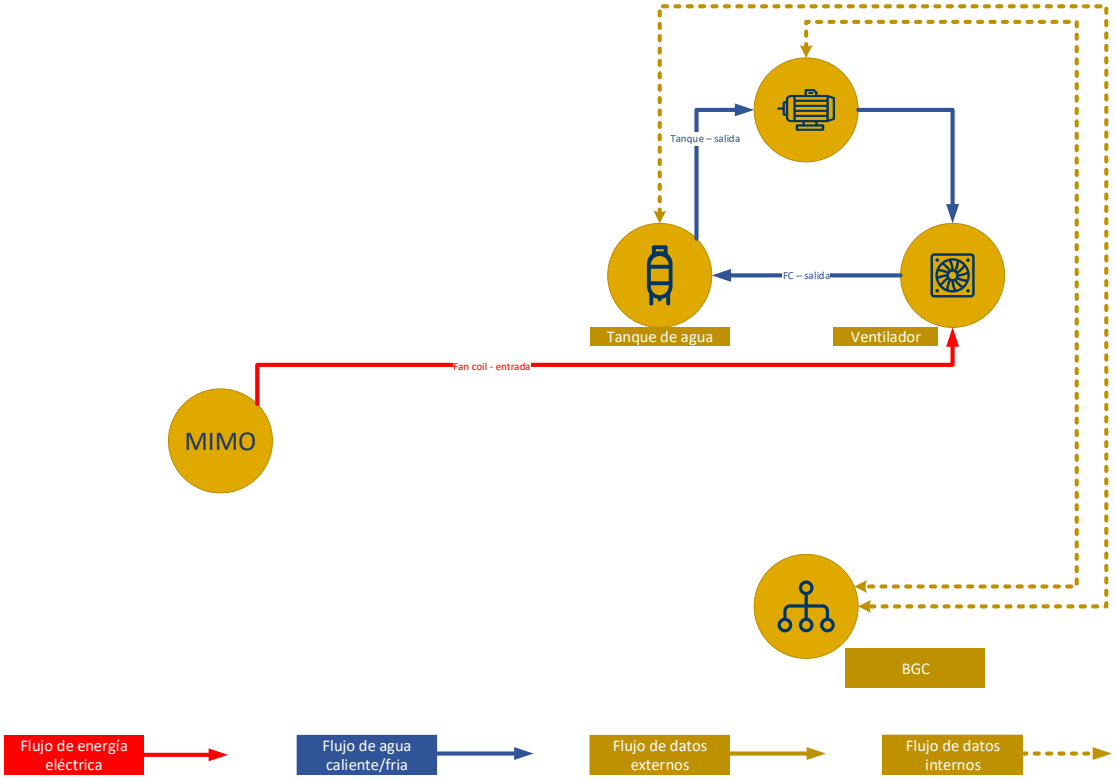


Figura 3.9 Interacción de los FCs en la arquitectura genérica del BEMS.

3.2.4.1. Controles a bordo

Los FCs deberán tener un sistema de control incorporado que permita medir las variables ambientales e internas útiles para los procesos del BEMS. Los FCs recibirán información de control del BEMS para ajustar su funcionamiento mediante un módulo de interfaz. El sistema de control a bordo controlará las operaciones básicas del aparato (es decir, encendido/apagado, situación crítica, cambio de estación y velocidad del ventilador, entre otros que pudieran estar disponibles).

3.2.4.2. Interfaz de comunicaciones y control

El sistema de control a bordo deberá estar conectado a un componente de interfaz de comunicaciones, controlado por un microcontrolador o similar, que admita comunicaciones inalámbricas mediante wifi. Cada FC debe implementar un servidor WoT basado en las recomendaciones del W3C que le permita interactuar con el sistema. A través de estos servicios Web, los FCs recibirán los comandos de control remitidos por el BEMS. Estos servicios se expondrán, como marcan las recomendaciones, a través de HTTP, permitiendo que los FCs se integren con el resto de los dispositivos. El módulo software de estos dispositivos

gestionará todos los sensores y actuadores instalados en los FCs, enviando y tomando valores, procesando datos, realizando conversiones y exponiéndolos a los dispositivos conectados a la red interna del edificio para su uso y consumo. La interfaz identificará y publicará las descripciones de las propiedades del FC en una URL privada.

3.2.5. Bomba de calor

La bomba de calor, o *heat pumps* (HP), se encarga de controlar y distribuir el agua a través del edificio. En la arquitectura propuesta, la HP deberá proporcionar valores de su telemetría interna y permitir actuar sobre sus consignas de control. En esta arquitectura, tal y como se muestra en la Figura 3.10, la HP interactúa con el resto de los componentes del BEMS de varias maneras:

- Intercambiando datos: la HP debe comunicar datos tales como modos de funcionamiento o temperaturas de consigna.
- Obteniendo energía: la HP recibe energía eléctrica de entrada constante a través del MIMO.
- Compartiendo energía térmica: la HP estará físicamente conectada a un tanque de agua, con lo que controlará los flujos de energía entre acumuladores y depósitos del edificio.

La HP se conectará a un dispositivo *gateway* que actuará de controlador, y que cumplirá la función de interfaz de comunicación con el resto del BEMS. Internamente, la HP deberá implementar un servidor que permita exponer las propiedades accesibles. El *gateway* se encargará de exponer los comandos y las variables de la HP siguiendo las recomendaciones sobre WoT del W3C, en forma de propiedades, y descripciones que hace que la HP se integre de manera transparente con el resto de los dispositivos. Cuando sea posible, la HP y su *gateway* se conectarán a la red a través de Ethernet.

3.2.6. MIMO

El dispositivo de tipo múltiples entradas y múltiples salidas, o *Multiple Input, Multiple Output* (MIMO), interoperará con el BEMS distribuyendo la energía eléctrica desde y hacia los dispositivos del edificio, como se puede apreciar en la Figura 3.11. Del mismo modo que la HP, el MIMO estará conectado a un *gateway* que cumplirá la función de interfaz de comunicación con el resto de los componentes del BEMS, exponiendo sus comandos y variables internas siguiendo las recomendaciones del W3C. Para lograr esta función, el MIMO deberá implementar un servidor que le permita interactuar con el resto de los dispositivos del sistema. Cuando sea posible, el MIMO y su *gateway* se conectarán a la red a través de Ethernet.

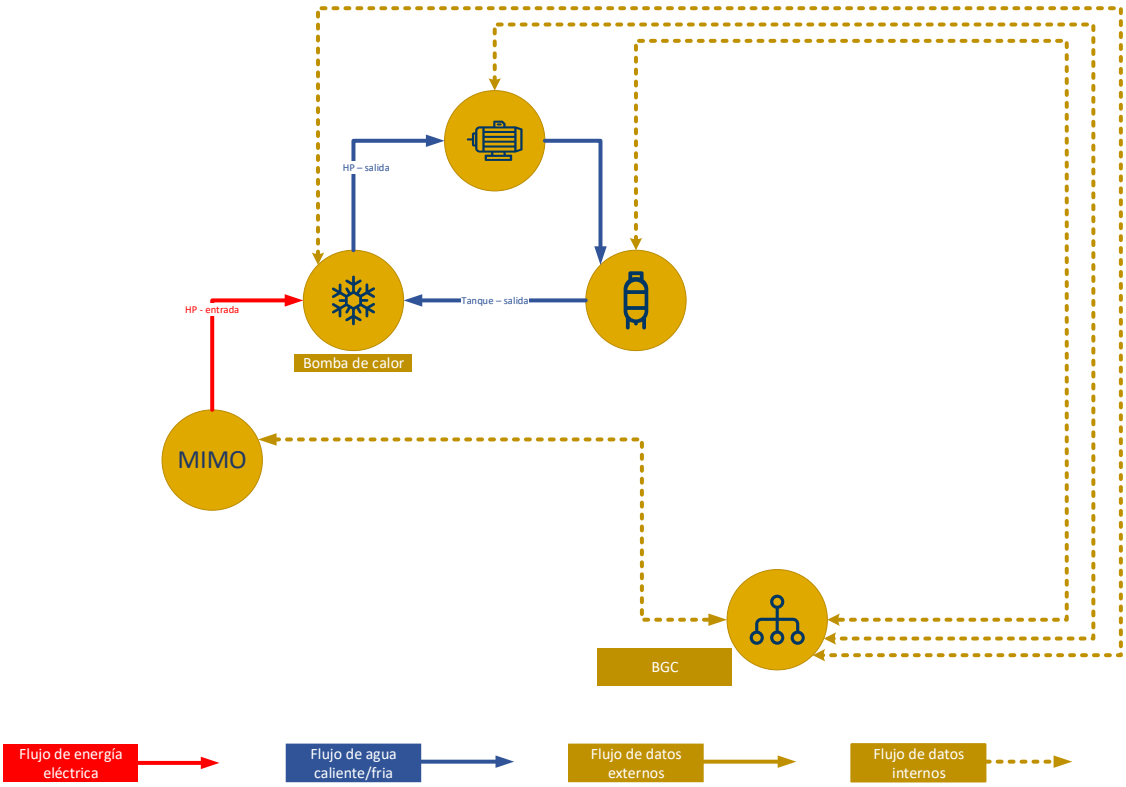


Figura 3.10 Interacción de la HP en la arquitectura genérica del BEMS.

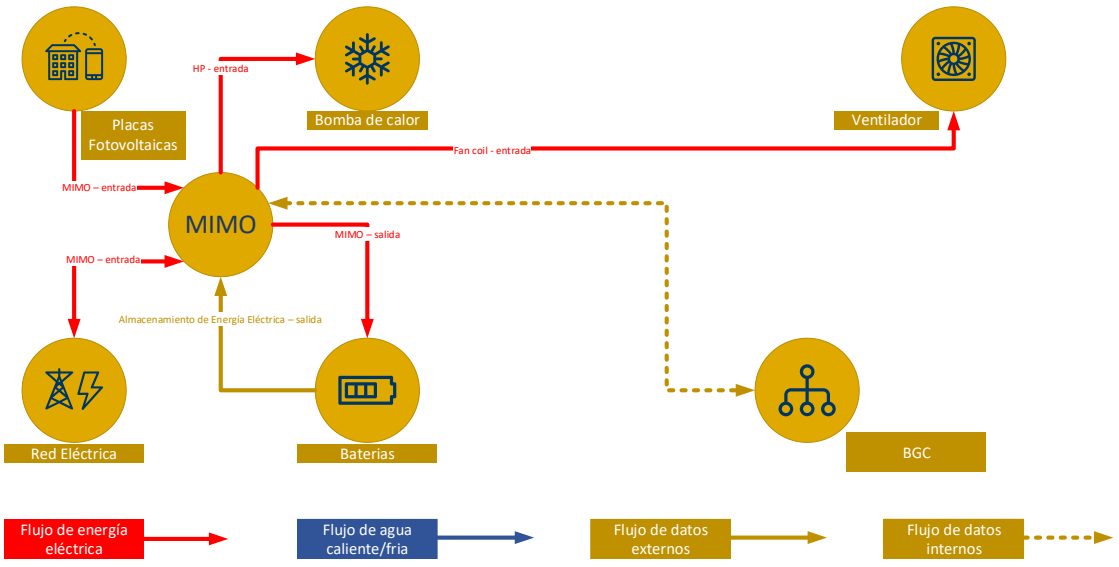


Figura 3.11 Interacción del MIMO en la arquitectura genérica del BEMS.

3.2.7. Autómata programable

En la arquitectura propuesta se incluye un autómata programable, o *Programmable Logic Controller* (PLC), que interoperará con el BEMS con el objetivo de controlar el hardware crítico de la instalación, como son las bombas de calor y los tanques de almacenamiento de agua. Mediante el PLC se deberá poder acceder a valores tales como el nivel y la temperatura del aqua de los tanques y el consumo de energía anotado en los contadores de la instalación. La función del PLC será proporcionar un acceso a estos datos, permitiendo la lectura y modificación de los parámetros que así lo permitan. El PLC también podrá incluir una capa lógica que le permita decidir sobre la operación de las bombas de agua en función de los sensores y parámetros medidos. La Figura 3.12 muestra cómo el PLC se conecta para obtener la telemetría de diversos dispositivos del sistema de agua del edificio.

La integración con el resto de los componentes del BEMS se hace a través de un *gateway*, que será capaz de transformar el protocolo de comunicaciones con el PLC y adaptarlo a las recomendaciones WoT del W3C. Siempre que sea posible, el PLC estará conectado a la red del edificio a través de Ethernet.

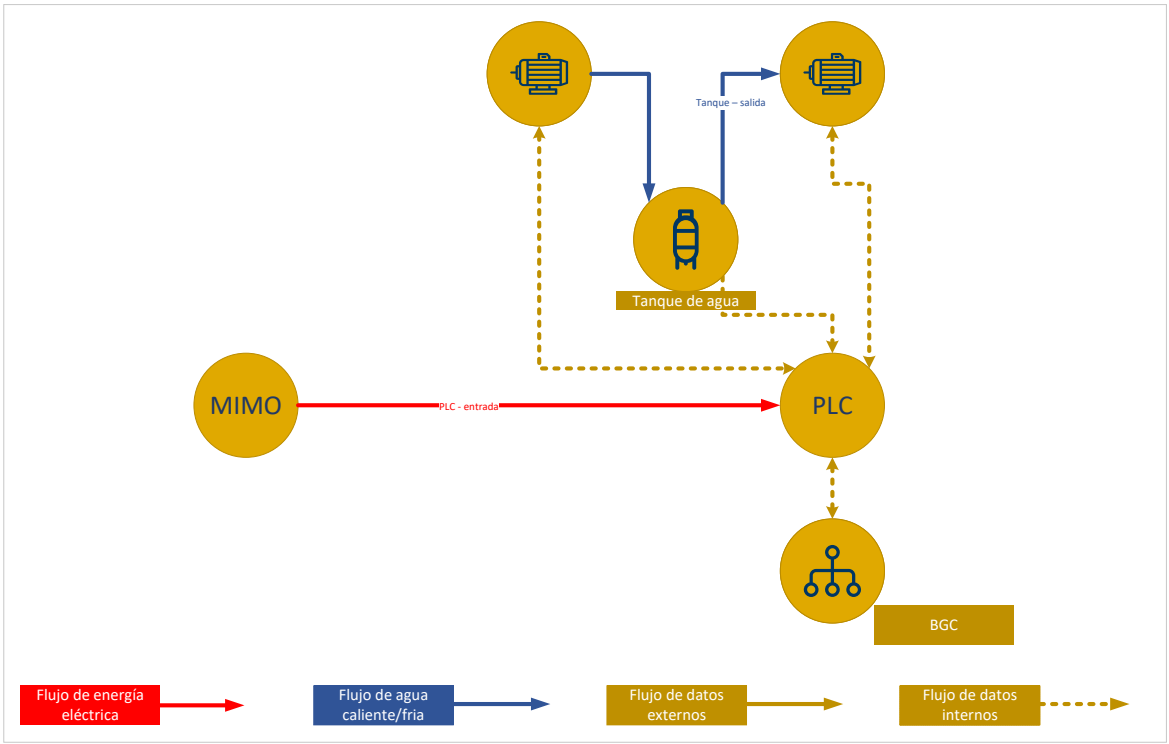


Figura 3.12 Interacción del PLC en la arquitectura genérica del BEMS.

Capítulo 4

Construyendo sistemas de control para edificios mediante WoT

En este capítulo se presenta un conjunto de experimentos orientados a evaluar, y validar en su caso, la arquitectura propuesta en el Capítulo 3 para el despliegue de una red de sensores y actuadores, así como su integración en una red de dispositivos heterogéneos en un edificio, que permitan controlar parámetros ambientales de forma eficiente, reduciendo por tanto el consumo energético. Los experimentos se llevan a cabo mediante dos casos de estudio, en los que se construirán y desplegarán sendos BEMS utilizando WoT como paradigma para la interconexión de todos los dispositivos y siguiendo las recomendaciones del W3C para arquitecturas WoT.

El primer caso de estudio se desarrolla, a modo de prueba de laboratorio, en un edificio de oficinas, y constituye un escenario de tamaño pequeño para los sistemas descritos en esta tesis. El segundo caso de estudio se desarrolla en un edificio de viviendas, que define un escenario de tamaño medio o grande para la arquitectura propuesta en esta tesis.

En las siguientes secciones se describen en detalle los dos casos de estudio, así como los resultados experimentales obtenidos y las conclusiones relativas a la validación de la arquitectura propuesta.

El despliegue de la arquitectura propuesta en cada caso de estudio parte de una implementación base que resulta común a cualquier edificio en el que se pretenda utilizar el sistema. En el Apéndice A se describen los módulos software que componen esta implementación base y que se desplegarán en los diferentes *gateways* del sistema. Esta base utiliza dos librerías en las que se implementan las recomendaciones del W3C para arquitecturas WoT: uWoT, que permite integrar en una arquitectura WoT dispositivos con recursos computacionales muy limitados (ver Apéndice B), y WoTPy [28].

4.1. Caso de estudio A: edificio de oficinas

El primer caso de estudio se considera como un experimento de laboratorio, y se lleva a cabo en un edificio de oficinas, un entorno de tamaño reducido como paso previo al despliegue en un edificio de viviendas. Este caso de estudio se desarrolla en las oficinas de Fundación CTIC Centro Tecnológico, en el Parque Científico Tecnológico de Gijón, Asturias[1]. Se trata de un edificio de cinco plantas, de las cuales dos están soterradas. El caso de estudio se desarrolla en las dos primeras plantas que se encuentran sobre el nivel del suelo, cada una de aproximadamente $100m^2$. La Figura 4.1 muestra una imagen del edificio de CTIC en el que se desarrolla este caso de estudio.

Figura 4.1 Edificio de oficinas del caso de estudio A: CTIC Centro Tecnológico; Parque Científico Tecnológico de Gijón.

El objetivo fundamental que persigue este caso de estudio es recabar datos preliminares sobre la viabilidad de la arquitectura propuesta y la solución diseñada en el Capítulo 3, así como de los dispositivos utilizados, en un entorno reducido y controlado. Para ello, se implementa una solución adaptada a los sistemas de aire acondicionado presentes en este edificio de oficinas. Una vez desplegada la solución, se debe validar la operación correcta e ininterrumpida de los sistemas, monitorizando los parámetros ambientales de los aparatos de aire acondicionado existentes.

Con este caso de estudio se espera cubrir todos los escenarios potenciales que se puedan dar en despliegues posteriores en edificios de viviendas de mayor tamaño, identificando y subsanando los posibles problemas o limitaciones de la solución propuesta. Estos escenarios incluyen una red inalámbrica compleja, con cobertura limitada y comunicaciones a través de paredes y suelos; múltiples dispositivos que deben compartir espacio en la red; un BGC que

[1]https://goo.gl/maps/nCb9FXwL6mfXsGAJ7

debe gestionar y exponer todas las *Things* del edificio, además de servir de enlace entre la red interna y la externa para poder acceder a servicios en la nube; etc.

4.1.1. Definición del hardware

En este caso de estudio, la implementación y el despliegue de la arquitectura propuesta hará uso de los sistemas de ventilación y aire acondicionado existentes en el edifico de oficinas de CTIC, cubriendo un espacio de dos plantas del edificio. Estos sistemas son, en base, similares a cualquier sistema de aire acondicionado que se pueda disponer en edificios residenciales. La principal diferencia radica en que la entrada de agua caliente y fría está dividida en dos sistemas distintos de tuberías. El sistema de aire acondicionado incluye un motor que mueve el agua, un ventilador que hace pasar el aire por el sistema y un radiador por el que pasa el agua caliente o fría.

En este despliegue, el MIMO y la HP son dispositivos virtualizados, dado que la instalación del edificio de oficinas no dispone de estos dispositivos. Ambos contienen un servidor Modbus TCP que simula a los servidores que podrían estar embebidos en estos dispositivos en edificios con infraestructuras de climatización más complejas. Se propone Modbus TCP porque es el protocolo más común en este tipo de dispositivos, aunque podría utilizarse cualquier otro. Estos dos dispositivos proveen datos estáticos con el propósito de validar los métodos de lectura y escritura contra un servidor Modbus que contenga las variables que se esperará encontrar en otros edificios. El MIMO y la HP se controlan de manera similar, ya que ambos dispositivos proporcionan una interfaz Modbus TCP como canal de comunicaciones principal. Estos dispositivos se conectan a sus *gateways* a través de la red cableada interna. Los *gateways* se encargan de conectarse, leer y exponer los datos que residen en estos dispositivos.

Al sistema de gestión de energía en este edificio se conectan también los FC, que estarán sensorizados cada uno de ellos por una placa personalizada (diseñada y construida *ad hoc* para los ventiladores del edificio) que se conecta de forma inalámbrica a la red. Los FC implementan un sistema de control que actúa de manera directa sobre ellos, además de varios sensores de temperatura y humedad.

El nodo central del sistema lo implementa el BGC, que actúa como un *proxy* entre los dispositivos desplegados en el edificio y los servicios externos, implementados en la plataforma de control en la nube. Este *proxy* agrega los datos adquiridos por los demás dispositivos del sistema (MIMO, HP y FC) a través de una única interfaz WoT mediante la cual es posible leer, escribir y actuar sobre dichos dispositivos. El BGC también actúa como controlador *offline*, siendo capaz de gestionar los aspectos básicos de operación del sistema cuando, por alguna circunstancia sobrevenida, no se posible el acceso a los servicios de control en la nube.

Los elementos hardware que se utilizan en el despliegue de la arquitectura propuesta en este caso de estudio son los siguientes. Las Tablas 4.1 a 4.9 muestran las especificaciones de estos dispositivos.

- 1× BGC, ejecutado en un Up Squared Gateway[2] (ver Tabla 4.1).
- 1× MIMO *gateway*, ejecutado en un Up Board[3] (ver Tabla 4.2).
- 1× HP *gateway*, ejecutado en un Up Board[4] (ver Tabla 4.2).
- 2× FC, controlados mediante una Pycom GPy[5] con una placa Pysense[6] (ver Tabla 4.3).
- 1× FC, controlado mediante un DOIT ESP32[7] (ver Tabla 4.4).
- 2× simuladores Modbus, ejecutados en sendas Raspberry Pi[8] (ver Tabla 4.5).
- 1× *router* wifi *mesh* AmpliFi Router HD[9] (ver Tabla 4.6).
- 2× puntos de acceso wifi *mesh* AmpliFi MeshPoint HD[10] (ver Tabla 4.7).
- 18× sensores DS18B20 de temperatura para el agua fría y/o caliente a la entrada y a la salida del FC y de temperatura del aire a la entrada y a la salida[11] (ver Tabla 4.8).
- 6× caudalímetros YF-B1 para tuberías de agua tanto caliente como fría.[12] (ver Tabla 4.9).

Modelo	UP Squared Gateway (UPS-GWS01)
Procesador	Intel Pentium™ N4200 2.5 GHz
Memoria	8 GB RAM LPDDR4
Almacenamiento	128 GB SSD
Conectividad	2x Gigabit Ethernet LAN y Wi-Fi
Consumo	18W

Tabla 4.1 Especificaciones hardware del BGC en el caso de estudio A.

4.1.2. Definición del software

La adquisición de datos, tanto de los FCs como de las HPs y del MIMO, sigue el esquema propuesto en la Sección 3.2 y utiliza la implementación base descrita en el Apéndice A,

[2]https://up-shop.org/up-squared-gateway-pentium-n4200-w-8g-memory-64g-emmc-board-w-ovesa-plate.html
[3]https://up-board.org/
[4]https://up-board.org/
[5]https://pycom.io/product/gpy/
[6]https://pycom.io/product/pysense-2-0-x/
[7]https://www.espressif.com/en/products/socs/esp32
[8]https://www.raspberrypi.org/
[9]https://store.amplifi.com/products/amplifi-hd-mesh-router
[10]https://store.amplifi.com/products/amplifi-meshpoint-hd
[11]https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
[12]https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/114991171_Web.pdf

Modelo	UP board
Procesador	Intel Atom x5-z8350 (hasta 1.92 GHz)
Memoria	4GB RAM
Almacenamiento	32GB
Conectividad	Ethernet, wifi
Consumo	13W

Tabla 4.2 Especificaciones hardware de los emuladores MIMO y HP en el caso de estudio A.

Modelo	Pycom GPy
Procesador	ESP32 de doble nucleo
Memoria	4MB + 520KBytes
Almacenamiento	8MBytes
Conectividad	wifi, BLE, LTE CAT M1/NB1
Consumo	1.5W

Tabla 4.3 Especificaciones hardware del controlador Pycom del FC en el caso de estudio A.

Modelo	Doit ESP32 DevKit v1
Procesador	MCU Tensilica Xtensa LX6, 240 MHz, 32 bits
Memoria	520 KB SRAM
Almacenamiento	4 MB flash
Conectividad	Wi-Fi IEEE 802.11 b/g/n/e/i 2.4 GHz, BLE 4.2
Consumo	1W

Tabla 4.4 Especificaciones hardware del controlador DOIT del FC en el caso de estudio A.

Modelo	Raspberry Pi 3
Procesador	ARM Cortex-A53, 64 bits, cuádruple núcleo
Memoria	1GB
Almacenamiento	MicroSD
Conectividad	Bluetooth y wifi
Consumo	12.5 W

Tabla 4.5 Especificaciones hardware del emulador Modbus en el caso de estudio A.

Modelo	AmpliFi Router HD
Conectividad	Ethernet y wifi
Consumo	6 W

Tabla 4.6 Especificaciones hardware del *router* wifi en el caso de estudio A.

exceptuando el PLC, ya que al no haber sistemas críticos para controlar en el edificio de oficinas, no se requiere su uso. Por tanto, en esta sección se describe el software que implementa específicamente la arquitectura WoT, detallando cada componente y su función dentro del sistema.

Modelo	AmpliFi MeshPoint HD
Conectividad	Wifi
Consumo	11 W

Tabla 4.7 Especificaciones hardware de los puntos de acceso wifi en el caso de estudio A.

Precisión	$\pm 0,5°C$
Rango de medida	$-55°C$ a $+125°C$
Consumo	1 mA
Voltaje	de 3 V a 5,5 V
Interfaz	1-Wire

Tabla 4.8 Tabla de características del sensor DS18B20 en el caso de estudio A.

Rango de medida	$1 - 30$ L/min
Temperatura máxima del líquido a medir	$120°C$
Presión de operación	1,75 MPa
Voltaje	de 5 V a 24 V
Consumo	15 mA
Interfaz	Pulsos digitales

Tabla 4.9 Tabla de características del caudalímetro YF-B1 en el caso de estudio A.

4.1.2.1. MIMO y HP

Ante la imposibilidad de disponer de un MIMO y HP reales en laboratorio, cada uno de estos elementos se sustituye por un servidor Modbus genérico, que simula las variables que proporcionan dispositivos reales instalados en edificios con infraestructuras de climatización más complejas. El simulador se ejecutará en un dispositivo de tipo SBC, en este caso en una Raspberry Pi, desplegando un servidor Modbus que simule las variables de lectura y escritura que se procesarán en entornos reales en edificios otros edificios. El *gateway* se encarga entonces de leer este MIMO o HP virtual y de exponer las variables contenidas en él mediante una librería WoT, siguiendo el esquema detallado en la Sección A.1.

4.1.2.2. FCs

El software que se desarrolla para los FCs desplegados en este caso de uso, tanto con la Pycom GPy como en el DOIT ESP32, se basa en la descripción realizada en la Sección A.3. A este esqueleto de implementación se le añaden una serie de métodos y funciones necesarios para la obtención de los datos de los sensores, como se puede observar en la Tabla 4.10.

Los métodos que obtienen datos de los sensores, como por ejemplo *read_Air_Humidity()*, se utilizarán en un bucle recurrente que, cada cierto tiempo, leerá todos los sensores del controlador. Este proceso es bloqueante, y supondrá que el dispositivo ponga en espera peticiones entrantes durante unos momentos mientras este se ejecuta. Otras funciones, como

Método	**Descripción**
async DS18b20_Read(sensor)	Conecta con un sensor DS18b20 específico y obtiene el valor actual de la temperatura. Aplica un filtro de mediana de `n` medidas instantáneas para evitar espurios derivados de errores en la medida.
async read_Air_Humidity()	Lee el porcentaje de humedad del aire y almacena el valor en memoria.
async read_Air_In_Probe()	Lee la temperatura del aire a la entrada del FC y almacena el valor en memoria.
async read_Air_Out_Probe()	Lee la temperatura del aire a la salida del FC y almacena el valor en memoria.
async get_Humidity_Air()	Retorna el último valor medido de humedad del aire.
async get_Temp_Air_In()	Retorna el último valor medido de temperatura del aire a la entrada del FC.
async get_Temp_Air_Out()	Retorna el último valor medido de temperatura del aire a la salida del FC.
async read_DHW_In_Temp_Probe()	Lee la temperatura del agua caliente a la entrada del FC y almacena el valor en memoria.
async read_DHW_Out_Temp_Probes()	Lee la temperatura del agua caliente a la salida del FC y almacena el valor en memoria.
async read_Water_In_Temp_Probes()	Lee la temperatura del agua fría a la entrada del FC y almacena el valor en memoria.
async read_Water_Out_Temp_Probes()	Lee la temperatura del agua fría a la salida del FC y almacena el valor en memoria.
async get_Temp_DHW_In()	Retorna el último valor medido de temperatura del agua caliente a la entrada del FC.
async get_Temp_DHW_Out()	Retorna el último valor medido de temperatura del agua caliente a la salida del FC.
async get_Temp_Water_In()	Retorna el último valor medido de temperatura del agua fría a la entrada del FC.
async get_Temp_Water_Out()	Retorna el último valor medido de temperatura del agua fría a la salida del FC.
async read_Water_Flow()	Lee el sensor de caudal de agua y almacena el valor en memoria.
async get_Water_Flow()	Retorna el último valor medido del caudal de agua.

Tabla 4.10 Métodos y funciones de los FCs en el caso de estudio A.

por ejemplo *get_ Humidity_ Air()*, se pasarán a la librería uWoT, concretamente al método *ExposedThing.set_ property_ read_ handler(property_ name, read_ handler)* como *read_ handler*, ya que permitirá obtener el valor de la propiedad asociada y se invocará cuando llegue una petición a la URL correspondiente. La ventaja de no hacer la lectura del sensor bajo demanda, es decir, cada vez que llega una petición, es que el dato está disponible de manera inmediata, en tanto que si se hiciere así, el tiempo de respuesta vendría determinado por la velocidad de respuesta del sensor así como por las tareas de prefiltrado y preprocesamiento que se hubiesen definido para obtener el dato en cuestión. En cualquier caso, la lectura de datos bajo demanda se podría configurar en caso de requerir el cumplimiento de restricciones de tiempo real en con algún tipo de sensor.

4.1.2.3. BGC

El BGC se encarga de agregar los datos de todos los *gateways* presentes en este despliegue, proporcionándoles visibilidad al exterior y ofreciendo una interfaz unificada de acceso al BEMS del edificio. El software que implementa este dispositivo se compone de una interfaz WoT que actúa como *proxy*, es decir, un acceso indirecto a las propiedades y acciones disponibles en todos y cada uno de los componentes del sistema.

Este *proxy* contiene la lista de elementos de la red a exponer (MIMO, HP y FCs) y se encargará de crear las redirecciones oportunas para hacerlos accesibles desde un punto de acceso único. Esta filosofía extiende la funcionalidad expuesta en la sección A.4. La manera en que es capaz de agregar todas estas fuentes de datos es a través de un *proxy* WoT, que permite enlazar TDs y exponer sus recursos como si fueran propios. Esto quiere decir que el BGC actúa como una puerta de acceso a todos los dispositivos. Para generarla se utiliza un fichero de configuración en el que se indican las URLs en las que se encuentran las TDs objetivo. Este fichero contiene un diccionario como el siguiente, donde se almacenan las direcciones:

```
"CATALOG": {
  "BASE_URL": "http://192.168.1.100:9191",
  "REQUEST_TIMEOUT_SECONDS": 3,
  "DEVICES" : {
    "FAN_COILS": {
      "FLOOR_01" :
        {
          "APARTMENT_65_02":
            {
              "ROOM_65_02_01": [ "http://192.168.1.220:80" , "20"],
              "ROOM_65_02_02": [ "http://192.168.1.222:80" , "22"],
              "ROOM_65_02_03": [ "http://192.168.1.224:80" , "24"]
            },
```

```
            "APARTMENT_67_01":
              {
                "ROOM_67_01_01": [ "http://192.168.1.205:80" , "05"],
                "ROOM_67_01_02": [ "http://192.168.1.207:80" , "07"],
                "ROOM_67_01_03": [ "http://192.168.1.228:80" , "28"]
              },
            "APARTMENT_67_02":
              {
                "ROOM_67_02_01": [ "http://192.168.1.230:80" , "30"],
                "ROOM_67_02_02": [ "http://192.168.1.200:80" , "00"]
              },
          },
        "FLOOR_02" :
          {
            ...
          },
      },

      "HEAT_PUMPS":
        {
          "HEAT_PUMP_01": ["http://192.168.1.106:9090", "HP01"],
          "HEAT_PUMP_02": ["http://192.168.1.107:9090", "HP02"]
        }

      "MIMOS":
        {
          "MIMO_01": ["http://192.168.1.101:9090", "MIMO01"]
        }

      "PLC":
        {
          "http://192.168.1.10", "PLC", "OFF"
        }
    }
}
```

4.1.3. Definición de la red

Para la conexión de los dispositivos del BEMS se utiliza, tal y como se ha descrito en la Sección 3.2.2, una red wifi en malla distribuida en el edificio a través de un *router*, que hace de punto de acceso principal, y dos puntos de acceso *mesh*, que pretenden cubrir todos los espacios donde sea necesario proporcionar cobertura de red. Además, se proporciona conexión mediante Ethernet para los dispositivos que no posean conectividad inalámbrica o se requiera un nivel de disponibilidad elevado, como por ejemplo el BGC.

La red inalámbrica desplegada en el edificio cubre todas las plantas en las que se despliega este caso de uso y los dispositivos contenidos en ellas. Los principales dispositivos de red, como son el BGC y el *router* central interno, están conectados a través de Ethernet (Cat 5 o superior). Un proveedor externo proporciona un módem de banda ancha para conectarlo al *router* de Internet (el módem de banda ancha y el *router* de Internet pueden ser el mismo dispositivo, según el producto suministrado por el proveedor de Internet). La topología de red implementada en este caso de estudio se resume en la Figura 4.2.

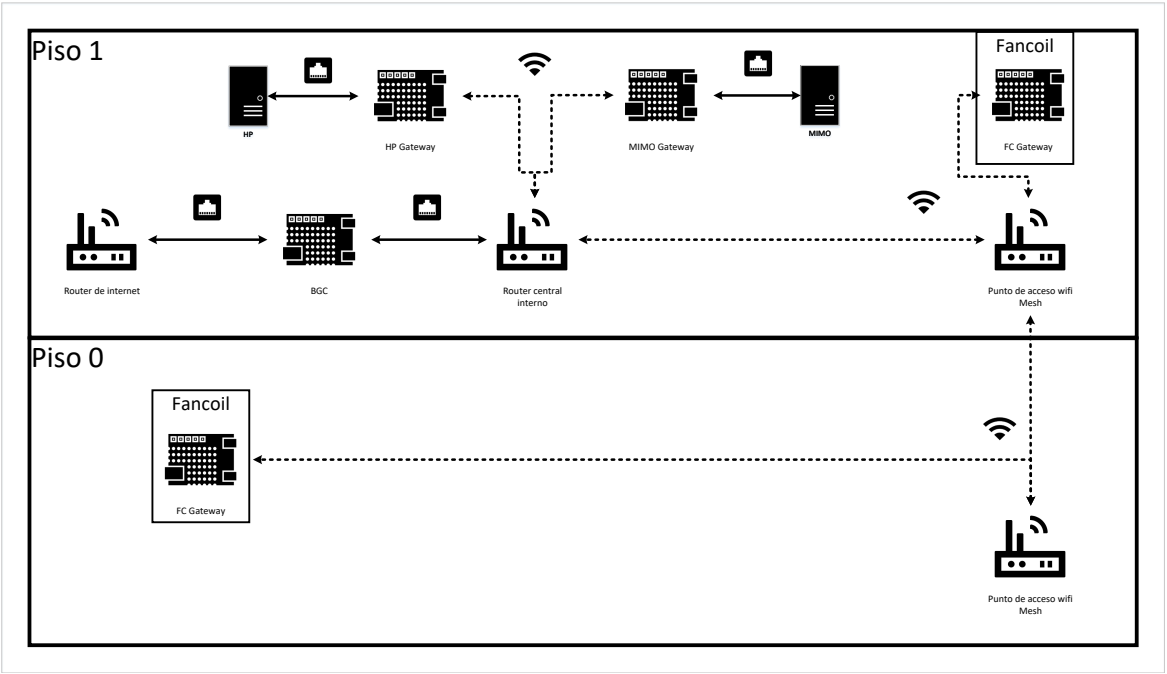


Figura 4.2 Arquitectura de red en el caso de estudio A.

El *router* de Internet se vincula directamente con el BGC, el cual actuará como un *proxy* de comunicaciones para todos los dispositivos del edificio (es decir, sensores y actuadores finales). El BGC es el único dispositivo del sistema que tiene acceso a Internet.

Para gestionar la red local del edificio, así como la configuración de los dispositivos del mismo, se utiliza un *router* interno y un *switch*, implementados en el mismo dispositivo hardware. El *router* implementa un servidor DHCP que asigna recursos y proporciona

direcciones IPv4 a todos los dispositivos. La misión del *switch*, conectado directamente al BGC, es dar soporte a la implementación de la red de área local o LAN del sistema.

El *switch* recibe, procesa y reenvía datos entre el BGC y el resto de los dispositivos del edificio. Como se ha mencionado anteriormente, se implementa una red local wifi en malla en el edificio utilizando *routers* de malla vinculados al *router* central. Los *routers* wifi *mesh* se conectarán a los puntos de acceso wifi *mesh* distribuidos en el edificio para desplegar la red inalámbrica. El número de puntos de acceso wifi *mesh* puede variar según la configuración del edificio, aunque para este caso de estudio se ha estimado que dos puntos son suficientes para proporcionar cobertura a todo el espacio de pruebas.

4.1.4. Despliegue

4.1.4.1. Despliegue de la red

La conexión a la red de los FC, el MIMO y la HP requiere de un acceso wifi a la red interna del edificio. Esta red crea varios puntos de acceso con SSID idénticos, permitiendo el uso de un solo nombre de red y contraseña para todos los dispositivos asociados, que se extiende por, en este caso, las dos plantas a cubrir. La configuración de la red local se complementa con una red cableada que permite conectar ciertos dispositivos de manera confiable. En este caso de estudio solo hay un dispositivo conectado por cable, el BGC, ya que es necesario para su conexión a Internet.

La solución para la red interna del edificio se basa en un sistema wifi en malla denominado AmpliFi, que se ha detallado en la Sección 4.1.1. Se trata de un producto comercial que proporciona capacidades de red de potencia empresarial. El sistema de AmpliFi está compuesto por AmpliFi *routers* y *MeshPoints*, componentes diseñados para trabajar al unísono para eliminar cualquier punto muerto en el edificio. El objetivo principal de esta red es proporcionar conectividad wifi a los FCs y demás dispositivos distribuidos por el edificio. Este sistema aporta tanto la conexión wifi deseada como la conexión cableada y el acceso a Internet, ya que dispone de un *router* central con una serie de puertos Ethernet y la capacidad wifi para conectarse a los puntos de acceso de la red en malla.

Como se observa en la Figura 4.2, se instalan dos puntos de acceso wifi *mesh*, uno en cada planta, donde se ubicarán los principales dispositivos de red. Los puntos de acceso *mesh* están instalados en el interior de las plantas y se comunican de un piso a otro. La parte más desafiante del despliegue de red es la configuración de los puntos de acceso wifi *mesh* para proporcionar una cobertura completa a todos los dispositivos del edificio y suficiente ancho de banda para satisfacer los requisitos impuestos por el BEMS. Además, es importante contar con redundancia de conectividad inalámbrica en caso de un fallo del punto de acceso. Tras la instalación de todos los dispositivos de red se realizaron varias pruebas de campo que confirmaron la cobertura del despliegue del sistema.

En el despliegue realizado se ha medido que en el primer salto *mesh* el ancho de banda es de 7,5 MB/s mientras que el segundo salto proporciona un ancho de banda de 2,6 MB/s.

4.1.4.2. Despliegue de los FC

El despliegue hardware de estos dispositivos requiere de la intervención de personal técnico especializado ya que precisa, por un lado, la instalación de sensores de caudal en las tuberías preexistentes en los FC, además de la instalación de las sondas de temperatura que se alojan en la tubería, en unos receptáculos específicos que también hay que integrar en la propia tubería. Y, por otro lado, el sensor de corriente ha de colocarse en la entrada de la alimentación del FC.

Una vez colocados en posición todos los sensores, se conectan al *gateway* del FC que se encargará, en este caso, de leer todas las variables proporcionadas.

4.1.4.3. Despliegue del MIMO y de la HP

El hecho de que en este caso de estudio tanto el MIMO como la HP sean simulados facilita en gran medida su despliegue, ya que son, eminentemente, componentes software. Por un lado, se despliega en las Raspberry Pi el software que simula el servidor Modbus TCP y se cargan en él todas las variables a leer. Después, se conecta de manera directa la Raspberry Pi con el *gateway* correspondiente, creando una subred privada exclusiva para las comunicaciones Modbus.

A continuación, en el *gateway* se despliega el software que implementa la librería WoTPy, que, al igual que la librería uWoT en los FC, permite exponer todas las variables del dispositivo.

4.1.4.4. Despliegue del BGC

El BGC es la piedra angular del BEMS edificio, ya que se encarga de hacer converger todos los *gateways* WoT en un solo punto. Esto se denomina *proxy*, y se trata de un componente software implementado mediante la librería WoT. Por un lado, se conecta a los *gateways*, lee sus propiedades y las guarda, y, por otro, las expone, permitiendo acceder desde un mismo punto a todos los dispositivos y sistemas del edificio. En el BGC también existe otro componente software, de utilidad en este caso para el desarrollo de las etapas de pruebas y experimentación, que se encarga de aprovechar las capacidades del *proxy* para leer de forma continua todas las variables que expone, almacenándolas en una base de datos local de tipo series temporales, TSDB, que también se encuentra instalada en el BGC. En este caso de estudio, la base de datos utilizada para experimentación e incluida en el BGC es una InfluxDB[13].

[13] https://www.influxdata.com/

4.1.5. Problemas encontrados

Durante el despliegue de este caso de estudio, y en las pruebas realizadas, se ha podido comprobar tanto la dificultad de la instalación de los sensores como el grado de precisión necesario para tomar ciertas medidas. Los fallos o problemas encontrados se han recopilado como base para mejorar el proceso de despliegue en futuras ocasiones y hacerlo más eficiente. Han surgido problemas tanto de hardware como de software y limitaciones a nivel de red.

4.1.5.1. Hardware

- Problemas con los sensores de temperatura que retornaban medidas inconsistentes del flujo a través de las tuberías de entrada y salida de agua. Esta limitación se resolvió utilizando adaptadores modo sonda que permiten un mejor contacto del sensor con el líquido.

4.1.5.2. Software

- Errores en la adquisición de datos: los *gateways* instalados en los FCs pierden frecuentemente la conexión wifi por lo que era necesario reiniciarlos. Para ello se desarrolló un algoritmo ligero capaz de analizar la intensidad de la señal wifi y conectarse a la más potente, para prevenir así la conexión a un punto de acceso demasiado lejano. Además, se desarrolló un sistema de reconexión automático.

- Los *gateways* instalados en los FCs sufrían fallos que detenían su funcionamiento, teniendo que reiniciarlos al cabo de un tiempo. Se solventa minimizando el uso de recursos, monitorizando de forma periódica tanto el estado de ejecución del proceso principal como de la conexión wifi, reiniciando el dispositivo en caso de requerirlo.

4.1.6. Experimentación

La experimentación llevada a cabo en este caso de estudio tiene como objetivo asegurar el correcto desempeño de todos los dispositivos conectados en el despliegue realizado en base a la arquitectura propuesta para el soporte del BEMS. Los experimentos se diseñan de tal forma que se pueda evaluar, en primer lugar, la integración de todos los componentes software, asegurando la interoperabilidad entre ellos mediante el paradigma WoT, y por tanto el funcionamiento del sistema. Y, en segundo lugar, el rendimiento de los dispositivos que conforman el sistema mediante pruebas de rendimiento, divididas en estabilidad y carga. En este caso de estudio las pruebas excluyen a los sistemas implicados con el MIMO y la HP, dado que son elementos virtuales y las pruebas de rendimiento no aportarían resultados concluyentes sobre las capacidades reales tanto del hardware como del software. Los experimentos con estos dispositivos se harán en casos de estudio posteriores.

Las pruebas en este caso de estudio se extendieron por un plazo de dos meses, en los cuales el sistema desplegado estuvo en funcionamiento durante diez horas al día, cinco días a la semana, lo que supone más de 300 horas de funcionamiento de cada dispositivo.

4.1.6.1. Pruebas de integración y del sistema

Las pruebas de integración permiten detectar defectos en la interacción entre todos los componentes software y hardware del sistema. Para ello, el esfuerzo se ha centrado en la comprobación de la comunicación entre todos los módulos, atendiendo a las interfaces definidas previamente, siguiendo una aproximación incremental. Tras las pruebas de integración se llevan a cabo pruebas del sistema, en las que se ejecuta la solución completa con el objetivo de validar que la solución propuesta cumple con las especificaciones definidas.

En estas pruebas, los dispositivos controladores de los FC exponen 15 propiedades, a las que se puede acceder en cualquier momento. Para simular el funcionamiento de un BEMS, las medidas se adquieren con una tasa de una muestra por segundo. Para ello, se realiza una solicitud a cada propiedad y se almacena en la base de datos local. Este proceso se repite en intervalos de diez minutos. Teniendo en cuenta que cada solicitud consume unos 100 bytes, la tasa de transferencia requerida para la operación de lectura completa es de, aproximadamente, 1,5 kB por minuto. Dado que existen dos FC en el sistema desplegado, el ancho de banda total necesario para monitorizar todas las propiedades es de 50 kB/s. En este punto se debe tener en cuenta que el ancho de banda disponible es variable, y depende de la distancia entre el FC y el punto de acceso de la red *mesh*. Cono se ha indicado anteriormente, en el despliegue realizado se ha medido que en el primer salto *mesh* el ancho de banda es de 7,5 MB/s mientras que el segundo salto proporciona un ancho de banda de 2,6 MB/s. En ambos casos el ancho de banda disponible es suficiente para los requisitos de funcionamiento del sistema.

Durante las pruebas se recabaron datos de temperatura del agua en los circuitos de agua fría y caliente, del flujo de agua y del consumo de energía de tres FCs. A partir de estos datos de temperatura, y como se ha descrito en la Sección 3.2.4, se puede calcular la energía térmica instantánea del FC. A modo de ejemplo, se muestran los datos adquiridos por uno de los FCs en cuatro días consecutivos de funcionamiento del sistema. No se muestran datos nocturnos ya que el sistema de FCs se apaga en horario que no es de oficina. Por un lado, se muestra la temperatura del agua caliente, tanto a la entrada como a la salida del FC, representada en la Figura 4.3a. En esta figura se pueden identificar momentos de arranque, en los que la temperatura asciende para adecuarse a la configuración del sistema, y también momentos de estabilidad, en los que la temperatura del agua fluctúa levemente para mantener los espacios a la temperatura seleccionada. Por otro lado, se obtiene el caudal del agua que se representa en la Figura 4.3b. En esta figura también se pueden identificar momentos de arranque, al principio del día. A partir de estos datos adquiridos, y aplicando la expresión 3.1 (ver Sección 3.2.4) para cada dato instantáneo, se puede calcular la energía térmica, que se

	DOIT ESP32	**GPy Pycom**
Número máximo de recursos expuestos	70	460
Tiempo de respuesta medio con el máximo de recursos expuestos	TD: $< 0{,}01$ s Recursos: 0,19 s	TD: $< 0{,}01$ s Recursos: 0,14 s
Número máximo de peticiones concurrentes	> 500	125

Tabla 4.11 Rendimiento de los *gateway* de los FC en el caso de estudio A.

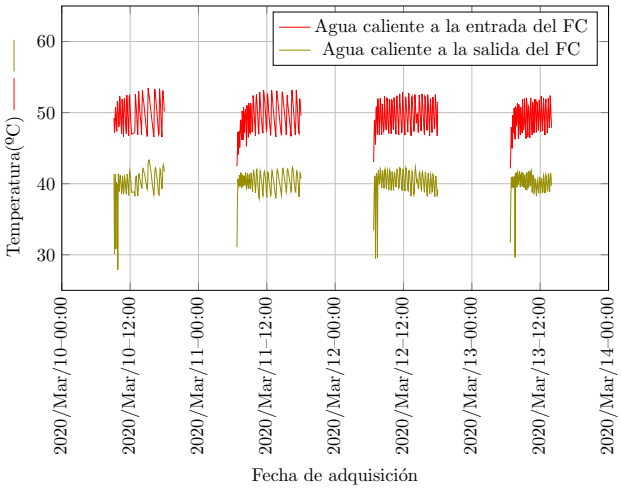
representa en la Figura 4.3c. En esta última figura se puede observar cómo al inicio del día la energía térmica es generalmente mayor, ya que es necesario calentar la estancia y, por tanto, hacer fluir más agua por el sistema, aunque puede haber momentos puntuales del día en que este dato también aumente debido, por ejemplo, a cambios bruscos de temperatura.

4.1.6.2. Pruebas de rendimiento

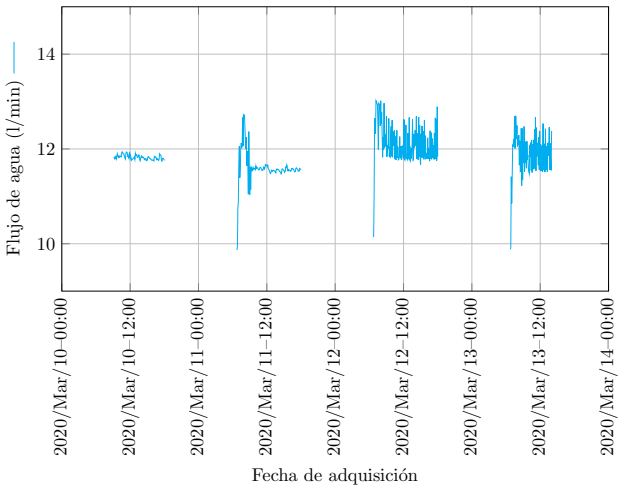
Para determinar la estabilidad del sistema, y evaluar su respuesta cuando aumenta el número de propiedades expuestas, se han diseñado y ejecutado pruebas de carga. En el primer conjunto de pruebas se ha simulado en un dispositivo GPy Pycom y en un dispositivo DOIT ESP32, que componen los *gateways* de los FC, la existencia de un número variable de propiedades. El objetivo de este experimento es poder determinar el tiempo de respuesta de cada dispositivo, incluyendo la sobrecarga que introduce la arquitectura WoT propuesta, tanto para las propiedades como para la TD. Dado que los dispositivos que implementan los *gateways* son hardware con recursos limitados, en primer lugar, se determina la capacidad de cada dispositivo para contener propiedades antes de llenar su memoria RAM. Esta capacidad se muestra en la Tabla 4.11. Como se puede observar, la placa Pycom GPy es capaz de contener un número más elevado de propiedades que la placa DOIT ESP32, debido a que dispone de una memoria RAM de mayor tamaño, aunque ambas placas utilicen de base el mismo SoC: Espressif ESP32.

A continuación, se ejecuta una batería de experimentos con el objetivo de probar la capacidad de respuesta de los FCs del sistema. En el contexto de este trabajo, cada dato se ha tomado mediante el análisis de 100 peticiones usando la herramienta Apache Bench[14]. Los tiempos de respuesta se miden sin concurrencia, es decir, las peticiones llegan al FC de una en una. Las desviaciones de la media oscilan desde unos 500 ms, cuando el número de propiedades es pequeño, hasta 1 segundo, cuando este número es elevado. Además, se deducen ciertas desviaciones puntuales de 2 o más segundos debidas a operaciones internas del dispositivo que retrasan la entrada y salida de datos. Todas las peticiones son peticiones HTTP REST

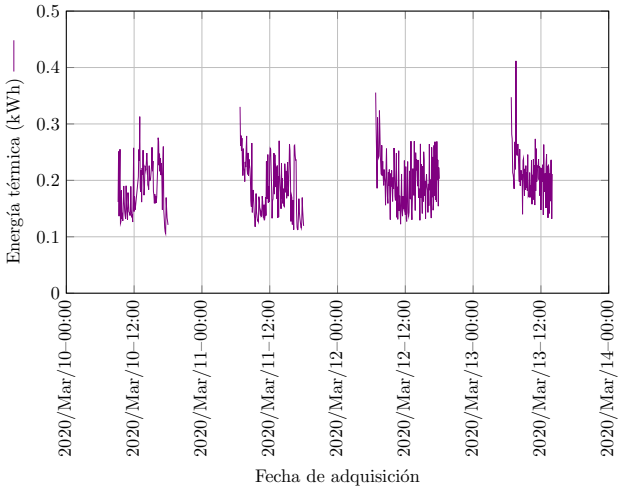
[14]https://httpd.apache.org/docs/2.4/programs/ab.html



(a) Temperatura del agua



(b) Caudal de agua

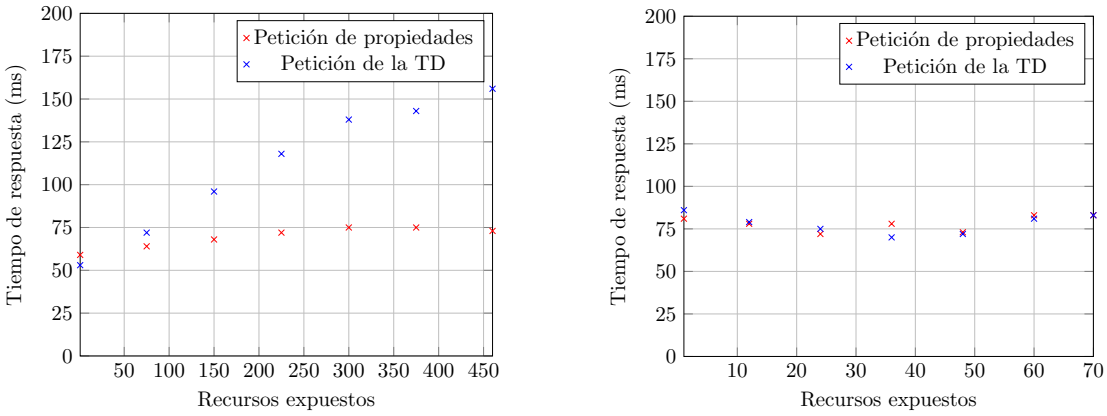


(c) Energía térmica

Figura 4.3 Variables medidas en un FC en el caso de estudio A.

GET hacia una URL que contiene, por un lado, la TD y, por otro, una de las propiedades simuladas del dispositivo.

Los resultados de los experimentos ejecutados sobre la placa Pycom GPy se muestran en la Figura 4.4a, y se puede apreciar cómo el hecho de aumentar el número de recursos, y por consiguiente el tamaño de la TD, provoca un mayor tiempo de respuesta por parte del dispositivo. Por otro lado, los experimentos realizados en la placa DOIT ESP32, mostrados en la Figura 4.4b, brindan resultados constantes debido a la limitada memoria RAM disponible, lo que a su vez limita la cantidad de recursos que se pueden exponer al mismo tiempo. La obtención de la TD es la más costosa en términos de tiempo de respuesta, en caso de tener cientos de propiedades. El tiempo que toma la obtención de la TD se incrementa de manera lineal consecuentemente al aumento del número de propiedades.



(a) FC controlado por Pycom GPy

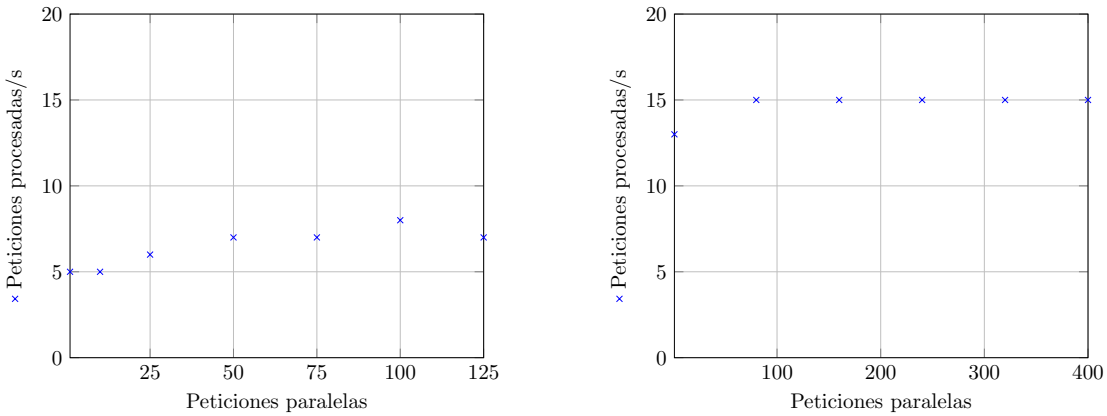
(b) FC controlado por DOIT ESP32

Figura 4.4 Tiempo de respuesta de los FC en base al tipo de *gateway* y al número de propiedades en el caso de estudio A.

Estas pruebas de carga se orientan a determinar la capacidad del sistema y de sus subsistemas para soportar un número elevado de peticiones de lectura/escritura. Con el propósito de medir cómo se comportan los sistemas del edifico en este aspecto se ha decidido medir el número de peticiones por segundo que es capaz de gestionar una *Thing*. Se han diseñado y ejecutado una serie de experimentos que miden este parámetro para los dos tipos de *gateway* que controlan a los FCs. El objetivo es medir el número de peticiones por segundo que es capaz de gestionar una *Thing*, variando el número de peticiones concurrentes que llegan a la misma. Los experimentos se han realizado mediante el análisis de 100 peticiones usando la herramienta Apache Bench, que permite enviar múltiples peticiones en paralelo. El número de propiedades expuestas por los FCs se ha fijado en 15 para estas pruebas.

Los resultados, que se pueden observar en la Figura 4.5, muestran que para ambos *gateways* existe una divergencia en cuanto al número de peticiones que son capaces de procesar por unidad de tiempo, aunque son similares en otro aspecto, ya que se comprueba que ambas

tienen una capacidad estable, aunque limitada, de atender peticiones, independientemente en su práctica totalidad del número de peticiones concurrentes que reciba el dispositivo.



(a) FC controlado por Pycom GPy

(b) FC controlado por DOIT ESP32

Figura 4.5 Productividad de los FC en base al *gateway* en el caso de estudio A.

4.1.7. Conclusiones

En esta sección se ha descrito la implementación y el despliegue de una solución, siguiendo la arquitectura propuesta en la Sección 3.2, para un edificio de oficinas a modo de prueba de laboratorio. En este caso de estudio se han probado, bien de forma real o de forma simulada, los componentes de una arquitectura WoT para el soporte de un sistema BEMS. El diseño y posterior despliegue permite la recopilación de los datos que se relacionarán con la gestión de energía del edificio. Todos los datos se obtienen mediante comandos estándar dirigidos a URLs simples, lo que garantiza la interoperabilidad de todos los dispositivos del sistema. Además, de esta forma es posible agregar en cualquier momento sensores y/o actuadores a diferentes dispositivos del sistema con un bajo impacto en los mismos. Mediante este caso de estudio se ha podido validar el diseño y el despliegue de la arquitectura, cuyo estado de madurez permite plantear un despliegue en escenarios más complejos.

4.2. Caso de estudio B: edificio de viviendas

El segundo caso de estudio se desarrolla en un edificio de viviendas, que constituye un escenario de tamaño medio-grande para el ámbito del trabajo en esta tesis. El edificio está situado en Bagnolo in Piano, en la Emilia Romana, provincia de Italia[15]. Se trata de un edificio de tres plantas en altura más una planta baja a nivel del suelo. La Figura 4.6 muestra una imagen del edificio en el que se desarrolla este caso de estudio. Este edificio es uno de los

[15]https://goo.gl/maps/5owFzuXuhsH6y4fx7

seleccionados para implementar todas las soluciones de rehabilitación y reacondicionamiento energético, identificadas y propuestas en el proyecto HEART en un edificio de viviendas existente (ver Sección 1.2.1), de forma que se pueda convertir en un edificio inteligente y energéticamente eficiente. Entre las soluciones implementadas se encuentra la arquitectura propuesta en esta tesis como soporte para la implementación de un BEMS basada en el paradigma WoT que sigue las recomendaciones del W3C.

Figura 4.6 Edificio de viviendas del caso de estudio B: Bagnolo, Piano (Italia).

El objetivo fundamental de este caso de estudio es recabar datos de uso reales con los que se pueda analizar la viabilidad de la arquitectura propuesta para garantizar la interoperabilidad de los dispositivos conectados en la red del edificio.

4.2.1. Definición del hardware

El hardware para este caso de uso se ha definido en colaboración con los socios del proyecto HEART, el cual dispone una serie de requisitos, variables a monitorizar y metodologías a aplicar para obtener los datos requeridos. El hardware del sistema está compuesto por los siguientes dispositivos distribuidos por el edificio:

- 1× BGC, ejecutado en un Up Squared Gateway[16] (ver Tabla 4.1).
- 1× MIMO Turbo Power Systems.

[16]https://up-shop.org/up-squared-gateway-pentium-n4200-w-8g-memory-64g-emmc-board-w-ovesa-plate.html

- 2× HP Helioterm.
- 1× PLC Siemens Simatic S7-1200[17] (ver Tabla 4.12).
- 28× FC, controlados mediante una Pycom GPy[18] con una placa Pysense[19] (ver Tabla 4.3).
- 112× sensores DS18B20 de temperatura para el agua fría y/o caliente a la entrada y a la salida del FC y de temperatura del aire a la entrada y a la salida[20] (ver Tabla 4.8).
- 28× Caudalímetro YF-B1 para tuberías de agua caliente y fría.[21] (ver Tabla 4.9).
- 1× *router* wifi *mesh* AmpliFi Router HD[22] (ver Tabla 4.6).
- 10× puntos de acceso wifi *mesh* AmpliFi MeshPoint HD[23] (ver Tabla 4.7).
- 1× Switch Zyxel[24] (ver Tabla 4.13).

Modelo	Siemens Simatic S7-1200
Procesador	CPU 1215C
Memoria	125 KB
Almacenamiento	128 GB SSD
Conectividad	1x Ethernet LAN
Consumo	12 W

Tabla 4.12 Especificaciones hardware del PLC en el caso de estudio B.

Modelo	Zyxel GS1900-10HP
Procesador	CPU 1215C
Capacidad de transmisión	14.88 Mbps
Conectividad	10× Gigabit Ethernet LAN
Consumo	95.3 W

Tabla 4.13 Especificaciones hardware del *switch* en el caso de estudio B.

[17] https://new.siemens.com/global/en/products/automation/systems/industrial/plc/s7-1200.html
[18] https://pycom.io/product/gpy/
[19] https://pycom.io/product/pysense-2-0-x/
[20] https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
[21] https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/114991171_Web.pdf
[22] https://store.amplifi.com/products/amplifi-hd-mesh-router
[23] https://store.amplifi.com/products/amplifi-meshpoint-hd
[24] https://www.zyxel.com/es/es/products_services/8-10-16-24-48-port-GbE-Smart-Managed-Switch-GS1900-Series/

4.2.1.1. HP

La bomba de calor se encarga de calentar y distribuir el agua por el edificio, enviándola, por ejemplo, a los FC. La HP mide la energía térmica, la cual fluye desde los tanques de agua que conserva la temperatura gracias a los materiales de cambio de fase, hacia los tanques de agua caliente doméstica. Para alimentarse, la HP recibe energía del MIMO, ya sea una o varias, ya que dentro de un edificio puede haber múltiples HP. En este caso de estudio hay dos HPs desplegadas en el edificio.

Este hardware está construido e instalado por Helioterm, y se compone de una bomba de agua, calentadores de agua, depósitos de agua caliente y un sistema de control. El sistema de control hace las veces de ordenador de a bordo, que a su vez implementa el control de la sensórica del dispositivo. La HP tiene sensores de temperatura, consumo eléctrico, caudal de agua, fallos, etc. Todos estos datos se exponen mediante el controlador de la HP a través de un servidor Modbus TCP que se conecta a la red del edificio mediante Ethernet.

4.2.1.2. MIMO

El MIMO es un sistema de gestión de energía que es capaz de gestionar múltiples entradas y salidas. Por un lado, el MIMO actúa como gestor solar para las placas solares instaladas en el tejado, gestionando la energía recibida y almacenándola o distribuyéndola en caso necesario. Por otro lado, recibe la corriente de la compañía eléctrica en trifásico y la que tenga almacenada en un conjunto de baterías. El MIMO alimenta el resto de los componentes del sistema BEMS del edificio, como son las HPs, los FCs, el GBC, etc.

Este hardware está construido e instalado por Turbo Power Systems, y se compone de los paneles fotovoltaicos, el sistema de gestión de energía, las baterías y un sistema de control, que implementa el control de la sensórica del dispositivo. El MIMO tiene sensores de voltaje, corriente, consumo eléctrico, temperatura o fallos, también dispone de actuadores que permiten abrir y cerrar circuitos, encender y apagar determinados dispositivos, etc. Todos estos datos e interacciones se exponen mediante el controlador del MIMO a través de un servidor Modbus TCP, que se conecta a la red del edificio mediante Ethernet.

4.2.1.3. PLC

El PLC se encarga de gestionar los sistemas de control del agua caliente del edificio, comprendiendo las bombas de agua, sensores de temperatura, contadores de energía calorífica y eléctrica. El PLC, mediante su sistema de gestión, es el responsable de medir, almacenar y proveer estos datos, los cuales son accedidos por el *gateway* correspondiente dentro del BGC. Los componentes que integran el sistema que controla el PLC se detallan en la Figura 4.7.

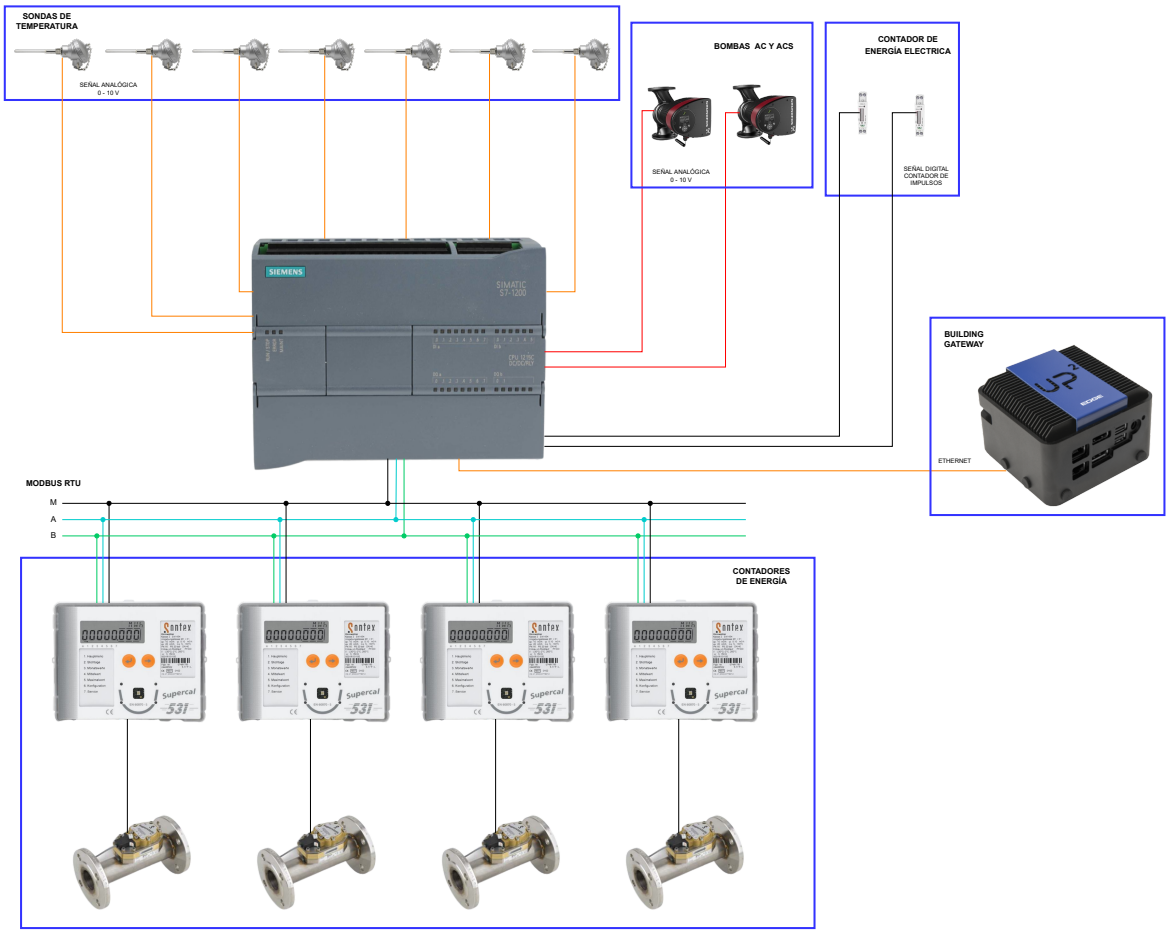


Figura 4.7 Diagrama de conexiones del PLC en el caso de estudio B.

4.2.1.4. FC

Los FC son dispositivos basados en una solución comercial de STILLE para aires acondicionados domésticos. Los FC proveen al BEMS de información ambiental y variables internas del FC, incluyendo la temperatura del aire tanto a la entrada como a la salida, la temperatura del agua a la entrada y a la salida, el caudal y la humedad. Los FCs interactúan con el BEMS compartiendo información al BGC, y también reciben del mismo información y comandos de control. Estos comandos pueden ser: apagar o encender, establecer una temperatura objetivo o cambiar la estación del año.

El sistema de control que se incorpora por parte del fabricante en los FCs es el encargado de actuar sobre los componentes físicos, tuberías, válvulas y ventiladores. Los FC reciben comandos e instrucciones desde el BEMS para ajustar sus modos de operación, usando una interfaz wifi. El sistema de control de STILLE está basado en una placa con un procesador ATmega328P, que controla las operaciones básicas del aparato. Esta información es recibida y procesada por la Pycom GPy que integra cada uno de los FCs en la arquitectura propuesta.

La Pycom se encarga de activar estos comandos básicos sobre el FC, a través de puertos lógicos. Los sensores y actuadores que se incluyen con el FC son los definidos en la Tabla 4.14.

1× Pycom GPy	Procesamiento y conexión wifi.
1× Pyboard Pysense	Placa de conexiones para la GPy (incluye sensor de humedad).
4× DS18B20	Sensores de temperatura para el agua a la entrada y a la salida del FC y de temperatura del aire a la entrada y a la salida.
1× Caudalímetro YF-B1	Medida del caudal de agua caliente o fría.

Tabla 4.14 Componentes de un FC en el caso de estudio B.

Los diferentes sensores y actuadores que se conectan con la Pycom GPy necesitan de una conexión cableada a través de los puertos de la GPy para comunicarse con la misma. El diagrama de conexiones se encuentra detallado en la Figura 4.8. En este diagrama se puede observar cómo los sensores se conectan a la alimentación y a los distintos puertos marcados en la placa GPy. A continuación, se detalla el funcionamiento de cada uno de esos sensores.

Pycom GPy y Pysense

Microcontrolador programado en el lenguaje MicroPython, lo que implica un desarrollo rápido y un despliegue sencillo. Pycom ofrece múltiples versiones de placas; la seleccionada para este despliegue proporciona conexión wifi, Bluetooth Low Energy (BLE) y LTE CAT M1/NB1. Otro punto fuerte de este controlador son sus placas de expansión: se usará la placa Pysense ya que integra, entre otros, un sensor de humedad. Estos sensores se comunican con la GPy a través del protocolo de comunicación I2C. Las características principales de este hardware son las que se muestran en la Tabla 4.3.

Sensor de temperatura aire/agua DS18B20

Todos los FC necesitan medir la temperatura tanto del aire al entrar y al salir como del agua al entrar y al salir. Para ello utilizan los sensores DS18B20, un termómetro digital que provee una temperatura codificada en 9 o 12 bits y calibrado en grados Celsius. El protocolo de comunicación que utiliza es 1-Wire, un protocolo de comunicación que requiere de sólo 2 conductores para proporcionar corriente y obtener el dato. Incluye también alarmas con la capacidad de establecer temperaturas límite tanto superiores como inferiores. Además, cada sensor posee un identificador único de 64 bits, lo que permite conectar múltiples sensores a un mismo bus 1-Wire. Las especificaciones de este sensor son las definidas en la Tabla 4.8.

Caudalímetro YF-B1

Los FC precisan de la medición del caudal de agua que circula a través de ellos para realizar un cálculo preciso de la energía calorífica usada. Para ello se utiliza un sensor de caudal que

se vale de un rotor y un sensor de efecto Hall. La velocidad del rotor se determina por la velocidad del flujo, y el sensor de efecto Hall (que varía su voltaje en función de la intensidad de un campo magnético que se genera al girar el rotor) indica mediante pulsos la cantidad de agua que ha pasado por el sensor. Las especificaciones del sensor YF-B1 se recogen en la Tabla 4.9.

Stille Board

Este componente está diseñado y construido por Stille, la empresa que proporciona los FC en este despliegue, y se encarga de gestionar los componentes internos del FC, así como la energía que lo alimenta. Las comunicaciones con este sistema incluyen:

- Encendido/apagado: permite el encendido y apagado de la placa de Stille en base al valor de un pin de la placa GPy.
- Estación: permite cambiar el modo calefacción/refrigeración, dependiendo de la época del año.
- Compresor: permite obtener el valor del compresor, ya sea si está encendido o apagado.
- Alarma: emite un valor en función de si existe algún problema con la placa Stille.

4.2.1.5. BGC

El BGC es el componente central del sistema dentro del edificio. Se conecta al resto de los dispositivos del mismo y es el encargado de manejar y contener los procesos de lectura y escritura de todos los componentes. Actúa como un *proxy* entre los dispositivos desplegados en el edificio y los servicios externos como la plataforma en la nube.

Este dispositivo actúa como un controlador BEMS, permitiendo monitorizar y aplicar la lógica de control del sistema cuando los componentes no tienen una conexión directa con la nube. También actúa como un *gateway* entre el edificio y la nube, transformando, interpretando y adaptando los mensajes, traduciendo protocolos y demás tareas de enlace. Para asegurar la comunicación con la plataforma *cloud*, el BGC se encuentra conectado a Internet a través de un Modem 4G desplegado in situ.

El BGC se implementa sobre la plataforma hardware UP Squared, una placa de alto rendimiento y bajo consumo que cubre todas las necesidades en cuanto a rendimiento, seguridad y conectividad requeridas. Esta placa basada en un procesador Intel Celeron y con 8 GB de RAM permite la instalación de un sistema operativo basado en Linux, concretamente Ubilinux, sistema operativo específicamente diseñado para esta familia de placas. También provee un disco duro de 128 GB donde almacenar los datos históricos, en caso de ser necesario, y dos interfaces de red RJ45, que pueden usarse para conectarse a ambas redes, tanto la red interna del edificio como la externa que provee Internet. Las características de este hardware coinciden con las del utilizado en el caso de estudio A, con lo que se muestran en la Tabla 4.1.

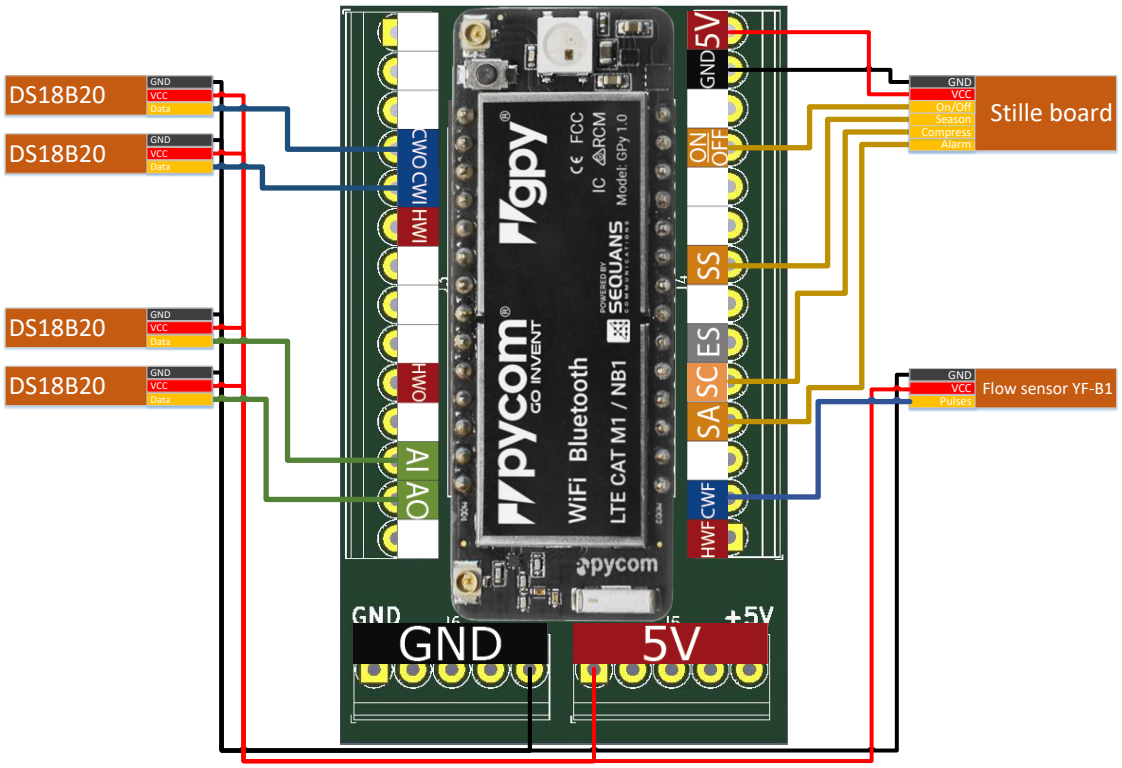


Figura 4.8 Diagrama de conexiones del *gateway* del FC en el caso de estudio B.

4.2.2. Definición del software

El software necesario para interactuar tanto con los FCs como con el MIMO y las HPs sigue el esquema planteado en la Sección 3.2. Por tanto, ahora se define el software que implementa la arquitectura WoT diseñada, detallando cada componente y su función dentro del sistema.

4.2.2.1. FC Gateways

Los dispositivos que contienen los *gateways* de los FC en este despliegue, Pycom GPy, se encuentran distribuidos junto a los propios FC por el edificio, conectados entre sí y con el BGC a través de la red wifi *mesh*. En estos dispositivos se ha implementado una versión de las recomendaciones WoT del W3C adaptada para el lenguaje MicroPython, tal y como se describe en el Apéndice B.

En el marco de esta implementación, y como se ha detallado en la Sección 4.1.2 y en la Sección A.3 del Apéndice A, se utiliza la librería uWoT desarrollada para exponer una serie de variables que cubren los distintos sensores a monitorizar dentro de los FCs, así como las acciones que se pueden tomar a través de ellos. La lista completa de variables a las que se puede acceder se detalla en la Tabla 4.15.

Tabla 4.15 Variables de los FCs en el caso de estudio B.

Tipo	Nombre	Unidades	Acciones posibles
propiedad	Temperatura de la entrada de aire	$°C$	Lectura
propiedad	Temperatura de la salida de aire	$°C$	Lectura
propiedad	Temperatura de la entrada de agua	$°C$	Lectura
propiedad	Temperatura de la salida de agua	$°C$	Lectura
propiedad	Humedad	%	Lectura
propiedad	Acelerómetro	Gs	Lectura
propiedad	Caudal de agua	L/min	Lectura
propiedad	Energía térmica del agua	Kw/m^3	Lectura
propiedad	Temperatura objetivo - Stille	$°C$	Lectura/Escritura
propiedad	Apagado/encendido - Stille	Boolean	Lectura/Escritura
propiedad	Estación - Stille	Boolean	Lectura/Escritura
propiedad	Compresor de agua - Stille	Boolean	Lectura
propiedad	Alarma - Stille	Boolean	Lectura
propiedad	Intensidad señal wifi	dBm	Lectura
acción	Reiniciar	Boolean	Escritura

Por otro lado, se define una serie de métodos que permiten tanto leer los sensores y almacenar sus valores, como recuperar esos valores cuando se recibe una petición de una propiedad. Estas funciones se detallan para este caso de estudio en la Tabla 4.16.

Cabe destacar también la lógica de control que se ha desarrollado para el encendido y apagado de los FCs, y que se utiliza para controlar la temperatura de la vivienda. En la función *stille_Control_Logic()*, definida en la Tabla 4.16, se realiza una monitorización de la variable *Temperature_Air_In* que contiene la temperatura actual de la estancia. En función de si esta variable sobrepasa ciertos límites y teniendo en cuenta si el FC está en modo verano o invierno (enfriar en verano, calentar en invierno), se enciende y apaga el FC para mantener la temperatura deseada. El esquema de la lógica de control se representa en la Figura 4.9.

4.2.2.2. BGC

El BGC es el pilar fundamental en la integración de los múltiples componentes del edificio. Esta pieza de hardware contiene un sistema operativo que permite la ejecución de Docker[25], la cual permite ejecutar múltiples instancias de un programa o servicio de manera paralela con un consumo de recursos muy reducido. De esta forma, y a diferencia del despliegue realizado en el caso de estudio A (ver Sección 4.1), se puede prescindir de *gateways* físicos e implementarlos mediante una solución software. Esto se traduce en que en el despliegue en

[25]https://www.docker.com/

Método	**Descripción**
async DS18b20_Read(sensor)	Conecta con un sensor DS18b20 específico y obtiene el valor actual de la temperatura. Aplica un filtro de mediana de `n` medidas instantáneas para evitar espurios derivados de errores en la medida.
async read_Air_Humidity()	Obtiene el porcentaje de humedad del aire y lo almacena en memoria.
async read_Air_In_Probe()	Lee la temperatura del aire a la entrada del FC.
async read_Air_Out_Probe()	Lee la temperatura del aire a la salida del FC.
async get_Humidity_Air()	Retorna el último valor medido de humedad del aire.
async get_Temp_Air_In()	Retorna el último valor medido de temperatura del aire a la entrada del FC.
async get_Temp_Air_Out()	Retorna el último valor medido de temperatura del aire a la salida del FC.
async read_Water_In_Temp_Probes()	Lee la temperatura del agua a la entrada del FC.
async read_Water_Out_Temp_Probes()	Lee la temperatura del agua a la salida del FC.
async get_Temp_Water_In()	Retorna el último valor medido de temperatura del agua a la entrada del FC.
async get_Temp_Water_Out()	Retorna el último valor medido de temperatura del agua a la salida del FC.
async read_Water_Flow()	Lee el sensor de caudal de agua y almacena la última medida del mismo en memoria.
async get_Water_Flow()	Retorna el último valor medido del caudal de agua.
async set_Stille_On_Off(value)	Cambia el estado del FC de encendido a pagado o viceversa.
async set_Stille_Season(value)	Cambia el modo de funcionamiento de invierno a verano y viceversa.
async set_Stille_Setpoint(value)	Cambia la temperatura objetivo del FC.
async get_Stille_On_Off()	Retorna el valor actual del estado del FC, ya sea encendido o apagado.
async get_Stille_Season()	Retorna la estación en la que se encuentra configurado el FC.
async get_Stille_Setpoint()	Retorna la temperatura objetivo actual del FC.
async stille_Control_Logic()	Decide si encender y apagar el FC en función de la temperatura objetivo, la estación en la que se encuentra y la temperatura del aire a la entrada del FC.

Tabla 4.16 Métodos y funciones de los FCs en el caso de estudio B.

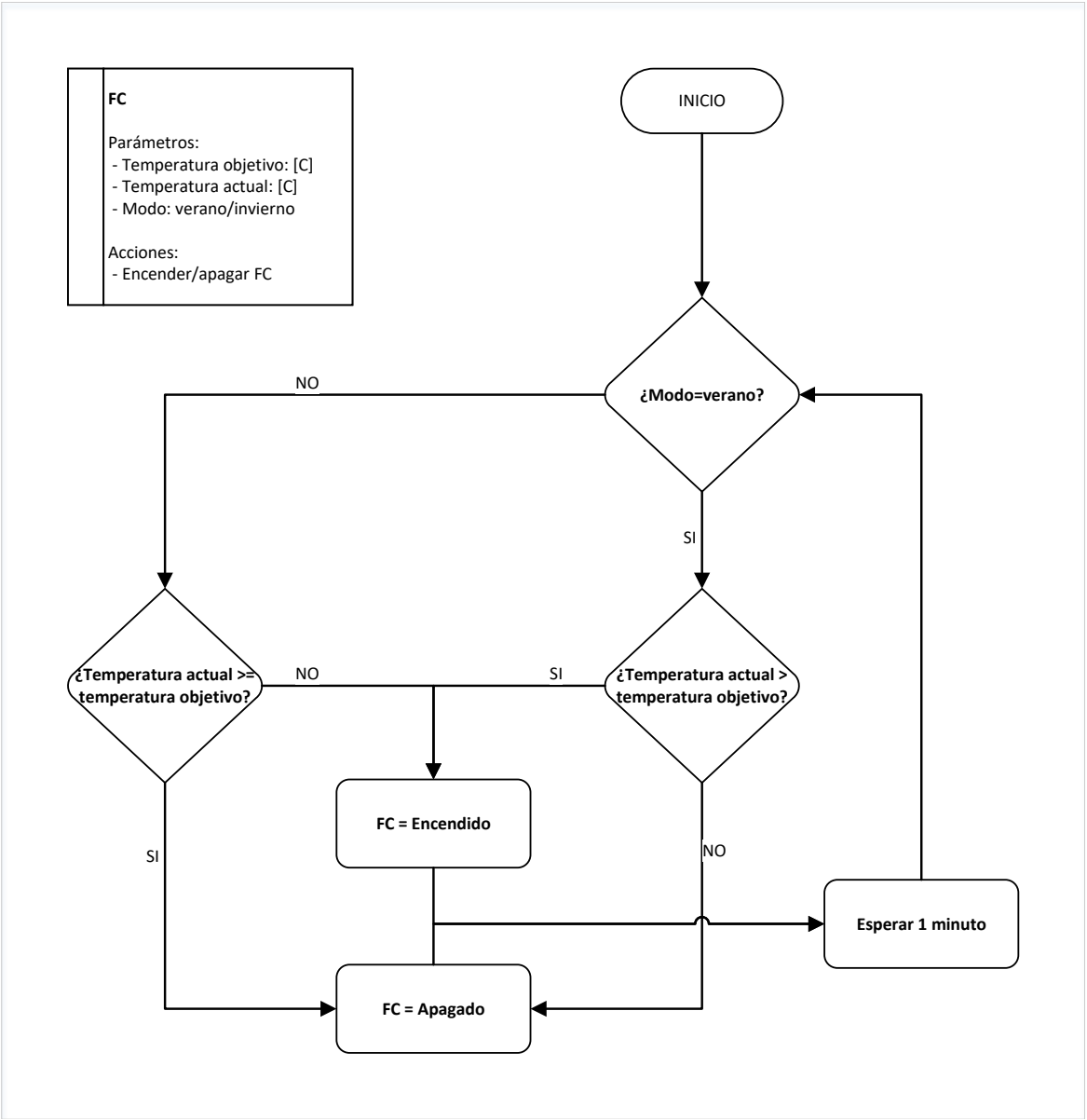


Figura 4.9 Lógica de control de la temperatura del FC en el caso de estudio B.

este caso de estudio los *gateways* son como instancias o contenedores ejecutados y expuestos dentro del BGC y vinculados con los elementos hardware a través de la red interna del edificio.

Estos contenedores se dividen en *gateways* y bases de datos. Los *gateways* contienen el software WoT encargado de leer, escribir e invocar las propiedades de los elementos hardware del edificio, como la HP y el MIMO. Por otro lado, la base de datos se utiliza para almacenar de forma temporal los datos leídos en de los *gateways*.

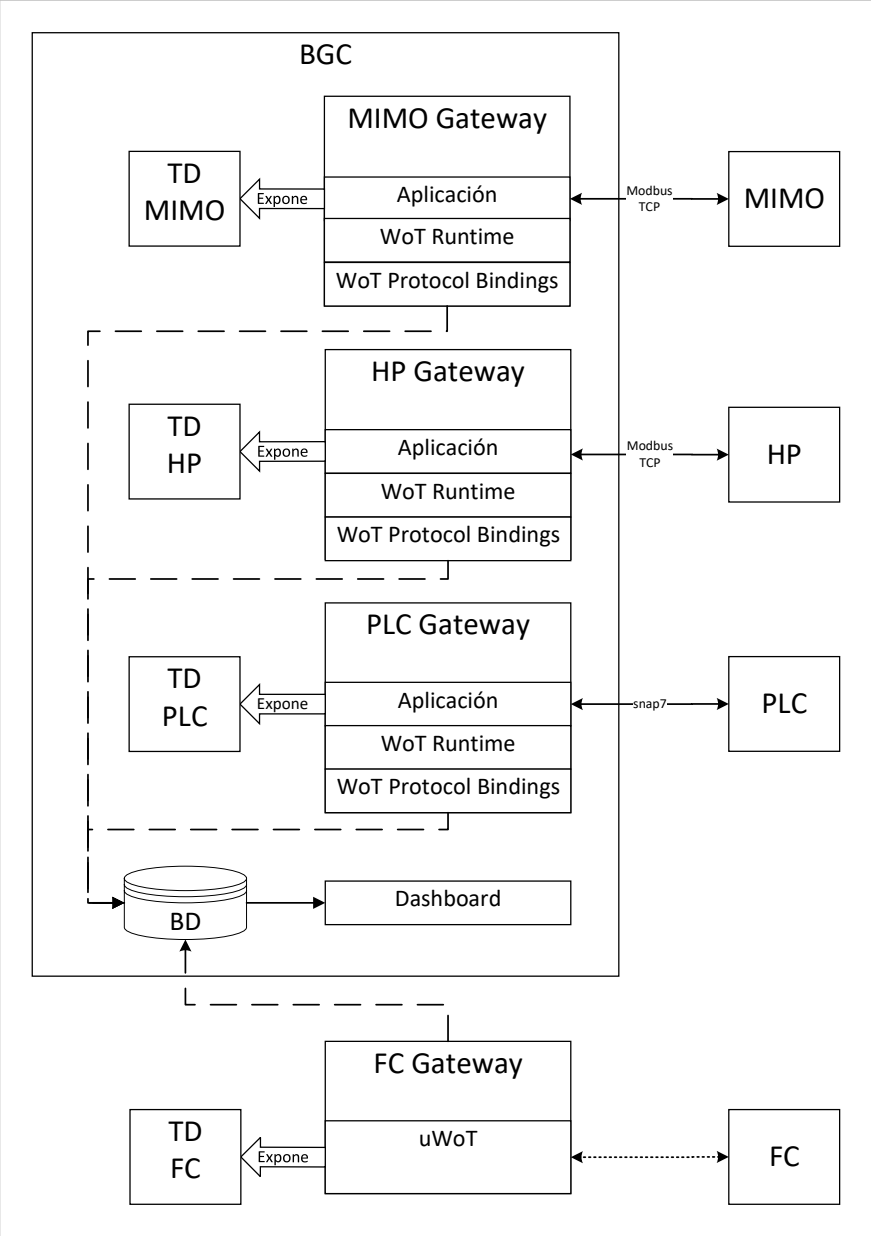


Figura 4.10 Interacciones del BGC con los componentes del edificio en el caso de estudio B.

MIMO *gateway* y HP *gateway*

En este caso de estudio los *gateways* tanto del MIMO como de las HPs están contenidos dentro del BGC. El *gateway* que controla la interfaz Modbus TCP con las HPs y con el MIMO del edificio está alojado en contenedores Docker que integran, como se muestra en la Figura 4.10, sendos *servients* WoT. Internamente, el componente WoT establece una conexión mediante el protocolo Modbus con el servidor dentro de las respectivas máquinas, desde

la cual se provee de los datos de los diferentes registros, los almacena y posteriormente los expone.

El software se compone de una serie de elementos que marcan las propiedades a leer en la HP y/o en el MIMO. Estos elementos contienen la información necesaria para obtener los datos a través de la conexión Modbus TCP. Es necesario tanto un identificador del esclavo Modbus como una dirección de registro, que corresponde con una de las direcciones que se pueden usar en un servidor Modbus para alojar los datos de los sensores. El conjunto de propiedades y direcciones del MIMO se recoge en la Tabla C.1 y de las HPs en la Tabla C.2, en el Apéndice C.

El *servient* WoT utiliza estos registros, de manera respectiva para cada uno de los *gateways*, para exponer una API WoT, representada por una TD en la que se incluyen todas estas variables como recursos accesibles a través de peticiones HTTP. Además, estas propiedades se completan añadiendo información proporcionada por la empresa proveedora de los dispositivos, como por ejemplo las unidades en las que están representados los datos, una descripción más detallada de lo que representan las propiedades, e incluso la declaración de acciones para escribir registros Modbus que, al modificarse, ejecutan tareas concretas.

PLC *gateway*

Este *gateway* se basa en el descrito en la Sección A.2 del Apéndice A, ya que utiliza la misma estructura de *servient* WoT, comunicándose con el PLC utilizando el protocolo *Snap7*[26] que se emplea para las comunicaciones por red con los PLCs de Siemens. Mediante este protocolo es capaz de recopilar los datos de los sensores que se guardan en las diferentes posiciones de memoria dentro del PLC y exponer de igual manera estas a través de la interfaz WoT. Los registros que se almacenan en el PLC y que son leídos y expuestos por este *gateway* se muestran en la Tabla C.3 del Apéndice C.

Dashboard

Esta plataforma de visualización se organiza como un listado de elementos desplegables que representan cada dispositivo WoT del edificio. La lista se carga con la página y los valores que se muestran en cada desplegable, que se actualizan de manera continua, solicitando los datos a todos los dispositivos listados. La página en cuestión se encuentra en un contenedor Docker dentro del BGC y se puede acceder a través de un puerto TCP configurable.

Cada dispositivo WoT tiene en su menú desplegable un listado de las variables que expone a través de su TD. Además, cada dispositivo cuenta con la posibilidad de modificar ciertas variables, si así lo permite la TD, y también invocar acciones. Por ejemplo, en un desplegable de un FC como el que se muestra en la Figura A.1, se puede observar que existen tres paneles, los cuales permiten cambiar la temperatura objetivo del FC, encenderlo y/o apagarlo o cambiar la estación en la que está configurado. La pulsación o cambio de cualquier parámetro

[26] http://snap7.sourceforge.net/

genera un diálogo de confirmación y recarga el valor para confirmar el cambio efectivo de la propiedad en cuestión.

4.2.3. Definición de la red

Para la conexión de los dispositivos del BEMS se utiliza, tal y como se ha descrito en la Sección 3.2.2, una red wifi en malla. Esta red se encuentra distribuida en el edificio a través de un *router*, que hace de punto de acceso principal, y diez puntos de acceso *mesh*, que pretenden cubrir todos los espacios donde sea necesario proporcionar cobertura de red para los FCs (que son los dispositivos que requieren conexión inalámbrica, dado que el resto utilizan una interfaz cableada). Además, se requiere una red cableada que permita conectar los dispositivos como el MIMO, las HPs y el PLC. Esta red se desplegará mediante un *switch* Ethernet que permitirá conectar a la misma red tanto los dispositivos inalámbricos como los cableados.

La red wifi cubre todas las plantas del edificio, con repetidores que proporcionan acceso inalámbrico a la red interna, como se aprecia en la Figura 4.11. Estos repetidores se encuentran en las zonas comunes del edificio, permitiendo, en caso de así requerirse, acceso a los mismos para su mantenimiento sin necesidad de acceder a las viviendas. Por otro lado, la red cableada se encuentra distribuida exclusivamente en el piso inferior, donde se encuentran el BGC, las HPs, el PLC y el MIMO. Ambas redes, inalámbrica y cableada, se encuentran unidas y accesibles entre sí, lo cual permite, desde el BGC, acceder a todos los dispositivos del edificio. También se puede observar que el *router* que proporciona acceso a Internet se encuentra conectado de manera exclusiva al BGC, con lo que se puede acceder al mismo a través de una red VPN segura y gestionarlo en remoto.

4.2.4. Despliegue

El despliegue de la infraestructura en el edificio se ha llevado a cabo por múltiples actores. Por un lado, y previo a las instalaciones de hardware, se ha realizado una tarea de reacondicionamiento del edificio, incorporando una nueva fachada, placas solares y sistemas de distribución de agua. El siguiente paso consistió en integrar las dispositivos y componentes mecánicos que se encargan de calentar, almacenar y distribuir el agua caliente y fría para los FCs. Una vez el edificio estuvo listo, se instalaron las dos HPs, el MIMO, conectándolo a las placas solares, el sistema eléctrico del edificio y unas baterías para el provisionamiento energético nocturno y de emergencia. Posteriormente, los FCs se instalaron en las viviendas, sensorizándolos según los requisitos del sistema. Más adelante, una vez que todo el hardware está disponible y en el marco del trabajo de esta tesis, se despliega la red y se instala el PLC completando el sistema con la conexión al BGC.

En lo que concierne a los FCs, se han realizado tareas de revisión, una vez instalados por parte de la empresa instaladora. En estas laboras se han añadido algunos sensores y

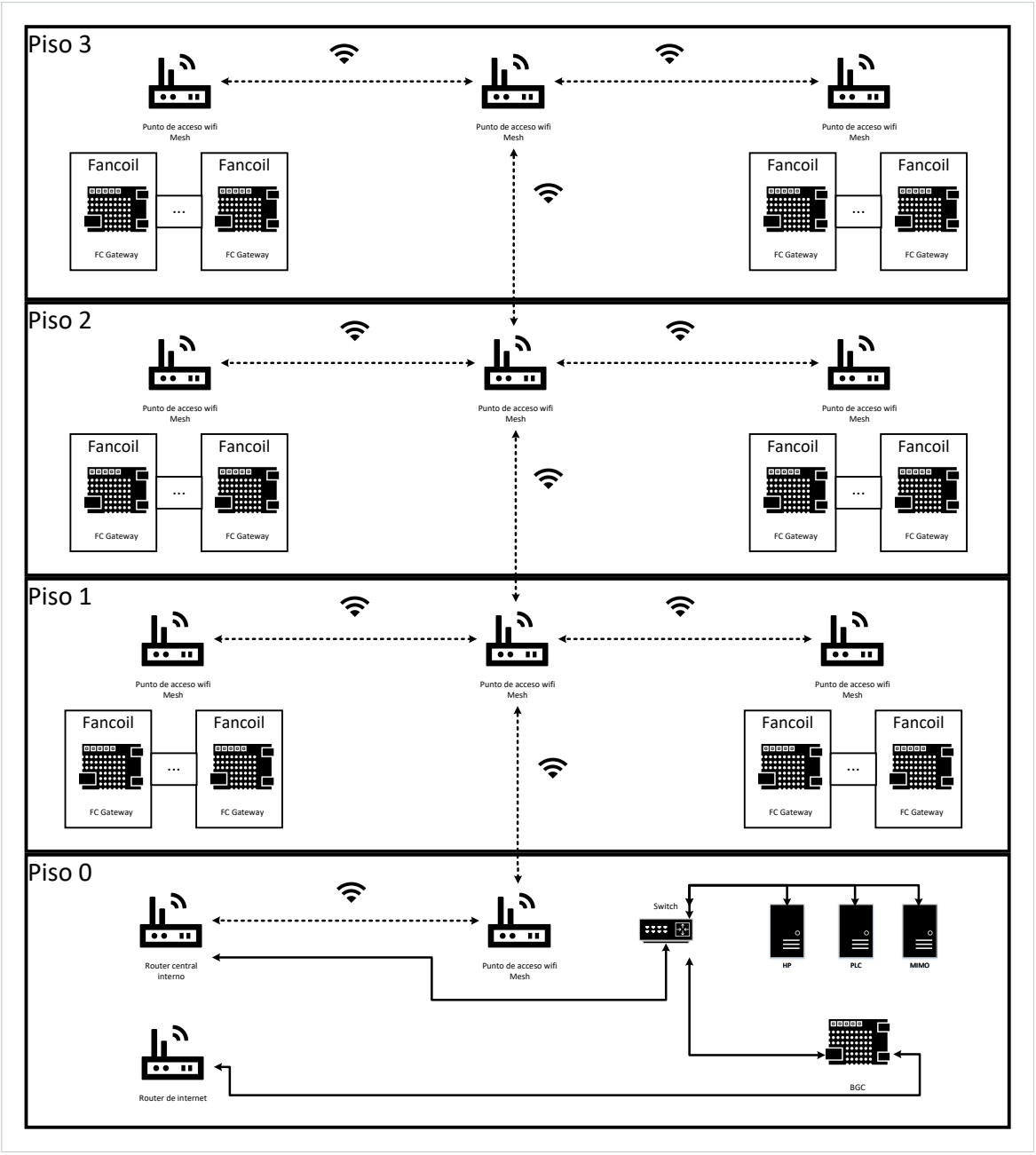


Figura 4.11 Arquitectura de red en el caso de estudio B.

recolocado las antenas wifi para una óptima recepción de la señal. En la Figura 4.12 se puede ver el montaje de uno de los FCs durante la fase de instalación de los sensores.

Para el despliegue software, los dispositivos hardware desarrollados en el contexto de esta tesis para la interfaz con los FC incorporan un sistema de actualización automática, u OTA[27] (*Over The Air*), que permite recibir actualizaciones periódicas de forma automática.

[27]https://docs.pycom.io/updatefirmware/ota/

Figura 4.12 FC durante la instalación de los sensores en el caso de estudio B.

Sin embargo, el BGC requiere una instalación de software personalizada, en función de los dispositivos de los demás actores partícipes del despliegue, los distintos *gateways* e interacciones. El PLC requiere a su vez un despliegue software específico para integrarlo de manera correcta con los múltiples dispositivos que forman el sistema de distribución y almacenaje de agua caliente.

En línea con la instalación y el despliegue de los dispositivos y del software, también se ha desplegado una instancia para pruebas que lee de manera periódica los datos de los sensores. En la Figura 4.13 se puede ver una captura de pantalla del *dashboard* desde el cuál se pueden monitorizar los datos históricos de los sensores.

4.2.5. Problemas encontrados

Durante el despliegue de este caso de uso se ha encontrado una serie de dificultades tanto logísticas como técnicas. Estos problemas han sido tanto hardware como software además de logísticos. A continuación, se detallan los problemas más reseñables y las soluciones aplicadas.

4.2.5.1. Problemas hardware

- Cobertura wifi FCs: La cobertura wifi de los FCs era muy reducida en el primer despliegue; se soluciona más adelante instalando nuevos repetidores a la vista de los mapas de cobertura obtenidos.
- Conexión wifi MIMO y HPs: en las pruebas iniciales del despliegue se determinó que la conexión wifi no era lo suficientemente estable en los bajos del edificio para mantener

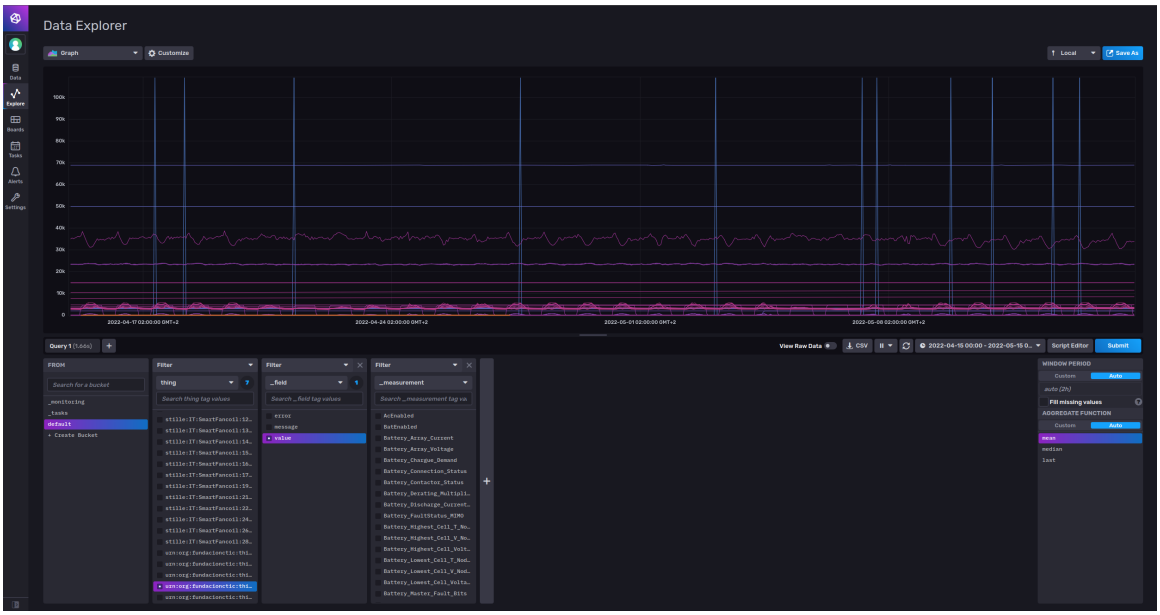


Figura 4.13 Visualización de datos en el entorno de prueba en el caso de estudio B.

una comunicación continua con estos dispositivos. Se opta, por tanto, por una conexión cableada mediante Ethernet.

- Baterías del MIMO: se detectó un problema cuando las baterías de este dispositivo se agotaban, que suponía que algunas de las variables del MIMO reportaran errores hasta su sustitución. Esta condición ya se tiene en cuenta para el resto de las etapas de funcionamiento del sistema.

- Control de la temperatura en los FCs: la lectura de los sensores de temperatura en estos dispositivos genera espurios en algunos casos, con lo que se decide aplicar un filtrado previo a los mismos antes de reportarlos a capas superiores del sistema.

4.2.5.2. Problemas software

- Lectura de las variables Modbus: Las variables de las HPs no se pudieron leer en una primera instancia debido a incidencias con la interfaz del fabricante de las HPs. En la fase de pruebas de integración del sistema se consiguió solucionar este problema mediante actualizaciones del *firmware* de varios dispositivos.

4.2.6. Experimentación

La experimentación llevada a cabo en este caso de estudio tiene como objetivo asegurar el correcto desempeño de todos los dispositivos conectados en el despliegue realizado en base a la arquitectura propuesta para el soporte de un BEMS en un entorno real de tamaño

medio-grande. En primer lugar, se diseña un conjunto de experimentos orientados a comprobar la conectividad wifi en todas las áreas del edificio en las que se van a conectar dispositivos inalámbricos al sistema. A continuación, se diseñan experimentos de tal forma que se pueda evaluar la integración de todos los componentes, asegurando la interoperabilidad entre ellos mediante el paradigma WoT, y por tanto el funcionamiento del sistema. Y, por último, se evalúa el rendimiento de los dispositivos que conforman el sistema mediante pruebas de rendimiento, divididas en estabilidad y carga.

4.2.6.1. Pruebas de cobertura y conectividad

Las pruebas de cobertura y conectividad tienen como objetivo determinar si la red wifi *mesh* que se ha desplegado en el edificio permite cubrir todos los espacios en los que se desplegarán los dispositivos inalámbricos del sistema y si proporciona la intensidad de la señal adecuada para conectarlos.

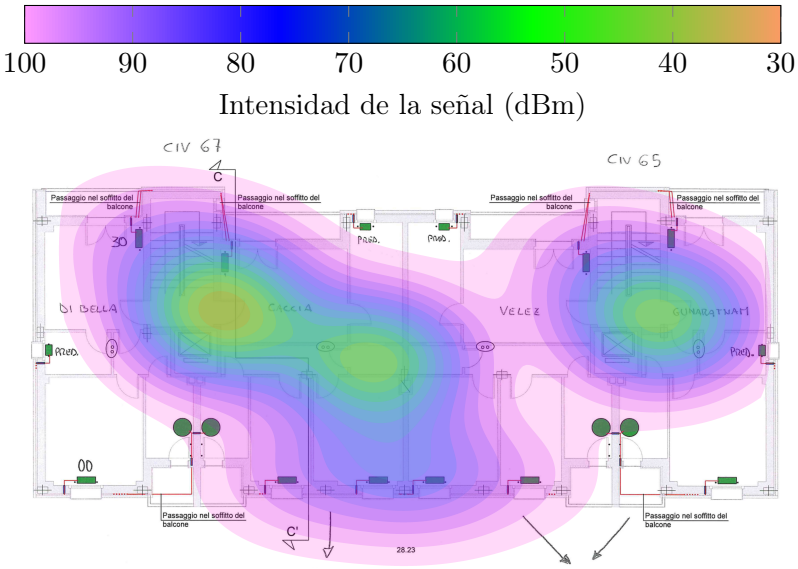
La intensidad de la señal se mide de dos formas diferentes en múltiples zonas de todas las plantas del edificio: con un dispositivo hardware específico que permite medir la intensidad de la señal wifi, y con los propios *gateways* de los dispositivos FC que se conectarán de forma inalámbrica al sistema y que almacenan los datos de este experimento en una base de datos temporal en el BGC. En el primer caso se utiliza un dispositivo idéntico al implementado en el *gateway* de los FC, una Pycom GPy conectada a una pequeña pantalla que muestra en todo momento la intensidad de la señal wifi. Usando este dispositivo se pudieron tomar datos de manera manual, y trazarlos en un plano del edificio. En base a los resultados se han podido añadir nuevos repetidores y recolocar otros ya existentes para asegurar una cobertura total.

La Figura 4.14 muestra los datos obtenidos en los experimentos anteriores, en la que se aprecia que la distribución de los puntos de acceso en el despliegue final es la adecuada para proporcionar conectividad a todos los dispositivos inalámbricos del sistema.

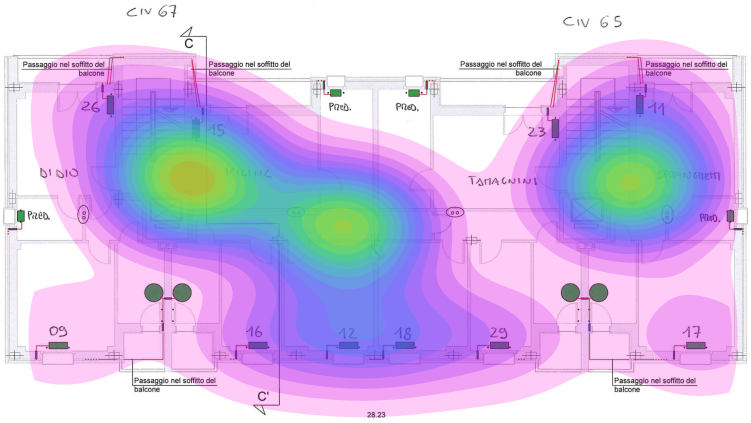
4.2.6.2. Pruebas de integración y del sistema

Las pruebas de integración permiten detectar defectos en la interacción entre todos los componentes software y hardware del sistema. Mediante estas pruebas se valida la comunicación entre todos los módulos, atendiendo a las interfaces definidas en la arquitectura, siguiendo una aproximación incremental. Tras las pruebas de integración se llevan a cabo pruebas del sistema, en las que se ejecuta la solución completa con el objetivo de validar que la solución propuesta cumple con las especificaciones definidas.

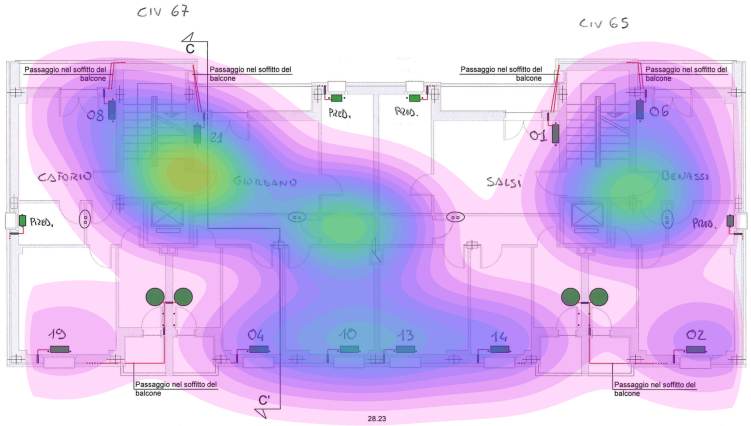
Las pruebas de integración en caso de estudio abarcan una amplia variedad de dispositivos, propiedades y mediciones. En la Figura 4.15 se muestra un ejemplo de dos propiedades (se eligen dos por simplicidad de la representación) muestreadas por el sistema en cada tipo de dispositivo que lo compone (FC, HP, MIMO y PLC) al realizar estas pruebas. La tasa de muestreo es configurable, dado que en función del tipo de variable puede requerir mayor o



(a) Planta primera



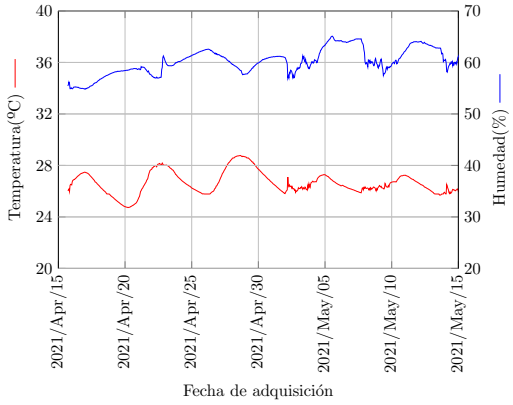
(b) Planta segunda



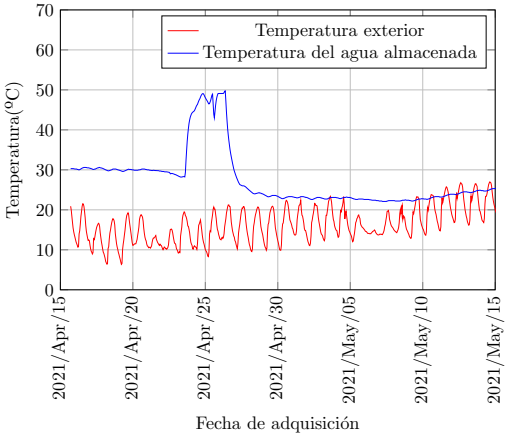
(c) Planta tercera

Figura 4.14 Cobertura e intensidad de la señal de la red inalámbrica wifi *mesh* en el caso de estudio B.

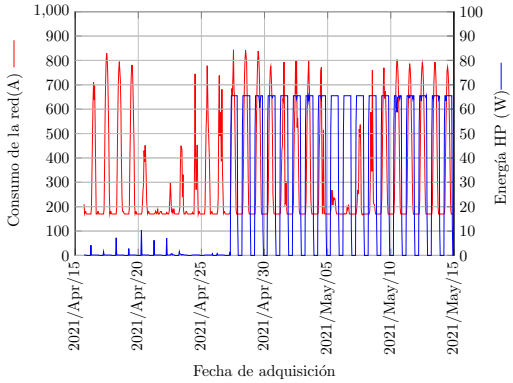
menor granularidad. Los datos de temperatura, en este ejemplo, están muestreados a una tasa de una muestra cada 10 minutos, por ser una variable con una evolución muy lenta. En esta figura se puede observar, por ejemplo, cómo la activación de las HPs el día 24 de abril provoca una subida de la temperatura del agua en el sistema, así como el incremento del consumo de energía de las HP.



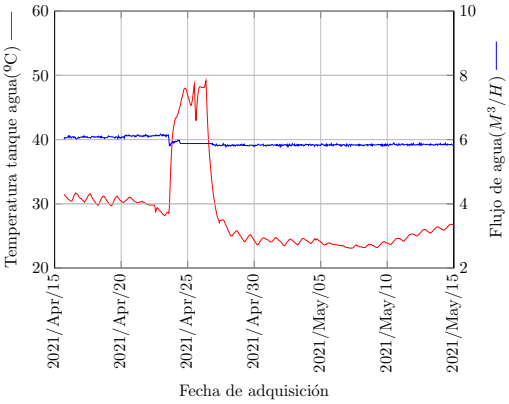
(a) Propiedades medidias en el FC.



(b) Propiedades medidias en la HP.



(c) Propiedades medidas en el MIMO.



(d) Propiedades medidias en el PLC.

Figura 4.15 Ejemplo de propiedades medidas en diferentes componentes del sistema en el caso de estudio B.

En el caso de los FCs, cuya interfaz es inalámbrica, en el intervalo en el que se realizaron las pruebas de integración, de las 1500 peticiones enviadas, 42 de ellas (el 2,9 %) no obtuvo respuesta. En cuanto a la estabilidad del sistema, y en base a los datos recabados, en el caso de la HP en la Figura 4.15b, apenas sufre pérdidas de datos, con solo un 0,13 % de tasa de pérdida en 1500 mediciones. En el caso del MIMO en la figura 4.15c, se obtiene una tasa de pérdida de datos similar a la anteriormente señalada en las HPs, en torno al 0,3 %. Finalmente, en el caso del PLC, en la figura 4.15d, es el más estable de todos, con una pérdida de 0 paquetes en todo el periodo de experimentación. Respecto a la pérdida de datos, esta no se

refiere a una pérdida de conexión, sino a un error 503 o 504, en el que el servidor WoT no es capaz de recuperar el dato solicitado del sensor, ya sea por un reinicio de los mismos o un retraso del hardware a la hora de procesar la señal.

4.2.6.3. Pruebas de rendimiento

Las pruebas de rendimiento se orientan a determinar la capacidad del sistema y de sus subsistemas para responder a las peticiones que reciban, en diversas condiciones, así como para determinar su estabilidad. Con este propósito, se ha determinado medir el tiempo de respuesta percibido cuando se solicitan datos a las diversas *Things* del sistema: FCs, MIMO y HPs. El objetivo es medir el tiempo transcurrido entre que se realiza la petición y el sistema WoT responde a la misma con el dato o datos solicitados.

Los experimentos que se describen a continuación se han llevado a cabo en noviembre de 2021 de manera remota en el despliegue del sistema en Bagnolo. Durante la realización de los experimentos, los dispositivos utilizados estaban desplegados y funcionando, y todas las operaciones secundarias de lectura/escritura habían sido detenidas para que no interfirieran en la determinación de la estabilidad de las *Things*.

FCs

Los FC deben soportar cierta carga a la hora tanto de exponer propiedades como de generar una TD lo suficientemente grande como para definir todas ellas. Esto puede generar, en un dispositivo de capacidades computacionales limitadas, como por ejemplo uno basado en un SoC ESP32, colapsos, cuellos de botella y en definitiva aumentar el tiempo de respuesta de las peticiones, lo que condicionaría el funcionamiento del sistema final.

Los experimentos relacionados con los FC se centran en dos objetivos. Por un lado, determinar el tiempo de respuesta de las propiedades de un FC en función del número de propiedades que publica dicho FC. De igual forma, se pretende determinar el tiempo de respuesta en la obtención de la TD. Se ha diseñado una serie de experimentos en los que se aumenta el número de propiedades de manera paulatina, tanto para la lectura de propiedades como la para la obtención de la TD.

Los experimentos se han realizado desde la propia red interna del edificio, aumentando de 10 en 10 el número de propiedades dentro de un FC. En cada experimento se ha realizado 100 peticiones tanto a una de las propiedades como a la TD. Las peticiones, en formato HTTP GET, se han realizado mediante la herramienta Apache Benchmark, la cual permite, además de tomar las mediciones, analizar los datos y guardarlos de manera íntegra.

La Figura 4.16 recoge los resultados obtenidos. En ella se muestra cómo al aumentar el número de propiedades el tiempo de respuesta sufre cambios casi de manera imperceptible, con una carga de propiedades desde 10 hasta 300. Paralelamente, en la Figura 4.17, se puede observar cómo ese aumento en el número de propiedades sí que afecta al tiempo de respuesta

de la TD, creciendo de manera lineal, como cabe esperar de un fichero que crece linealmente con el número de propiedades que representa. Finalmente, en la Figura 4.18 se puede observar un resumen comparativo los patrones identificados en los tiempos de respuesta.

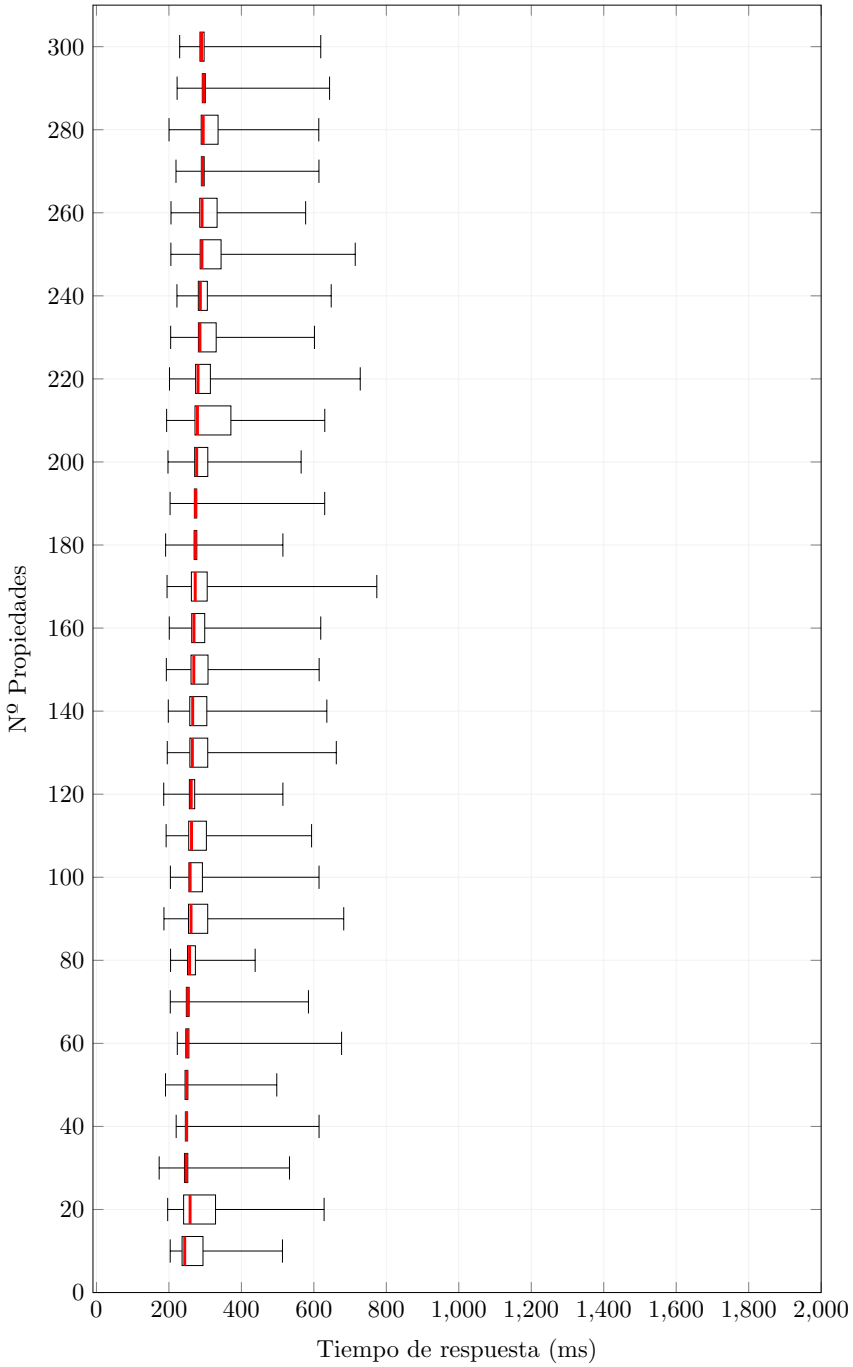


Figura 4.16 Rendimiento del FC ante la solicitud de una propiedad de un FC en el caso de estudio B.

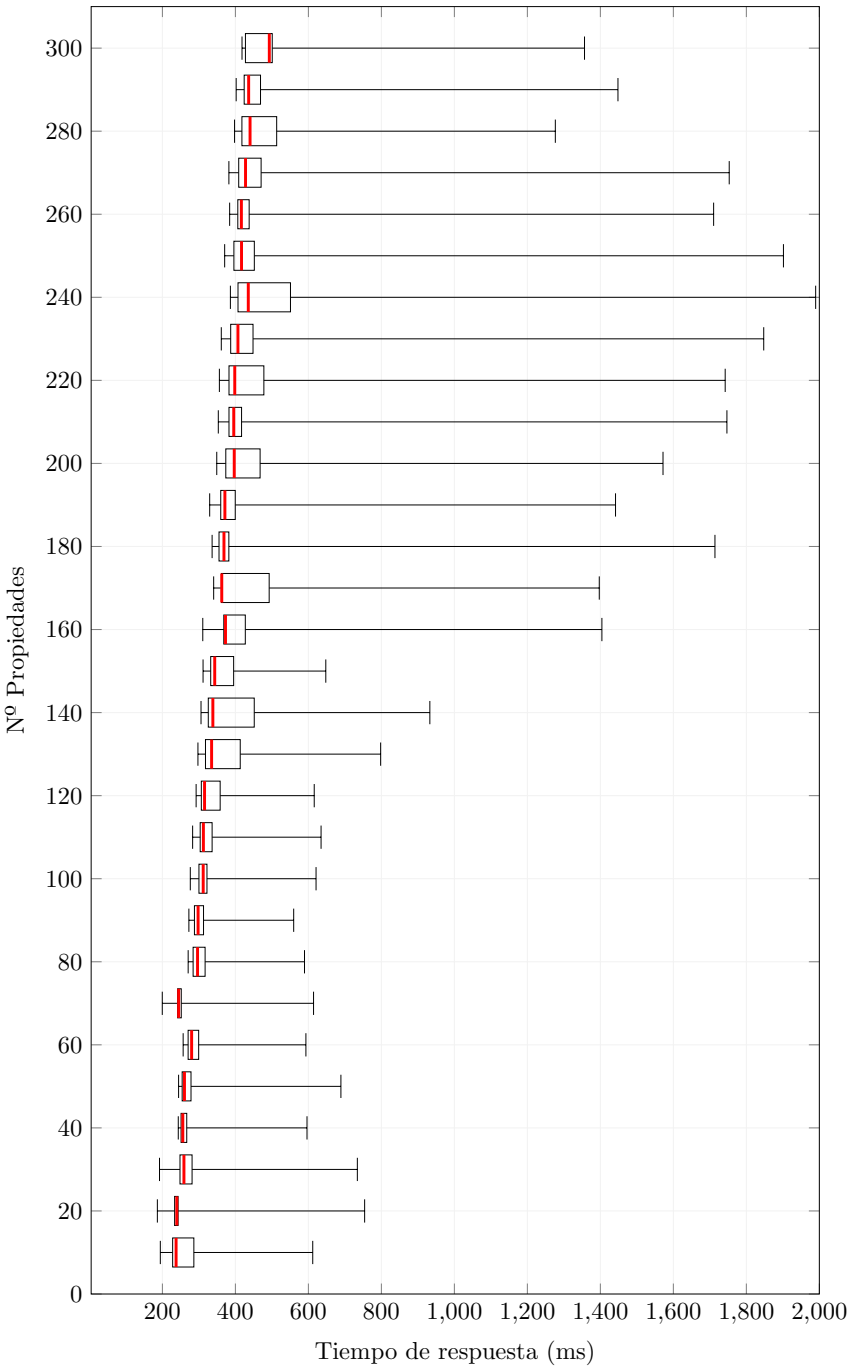


Figura 4.17 Rendimiento del FC ante la solicitud de una TD de un FC en el caso de estudio B.

En el sistema desplegado en este caso de estudio, los FCs proporcionan 15 propiedades, definidas en la Tabla 4.15. Estas propiedades han de estar disponibles en todo momento para su modificación o consulta por parte del resto de componentes del sistema. Por tanto, el

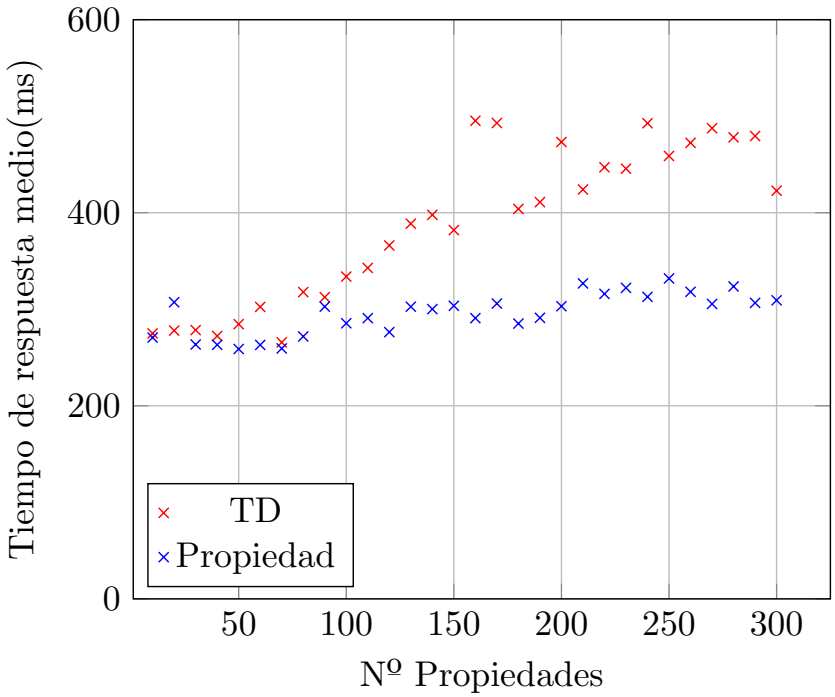


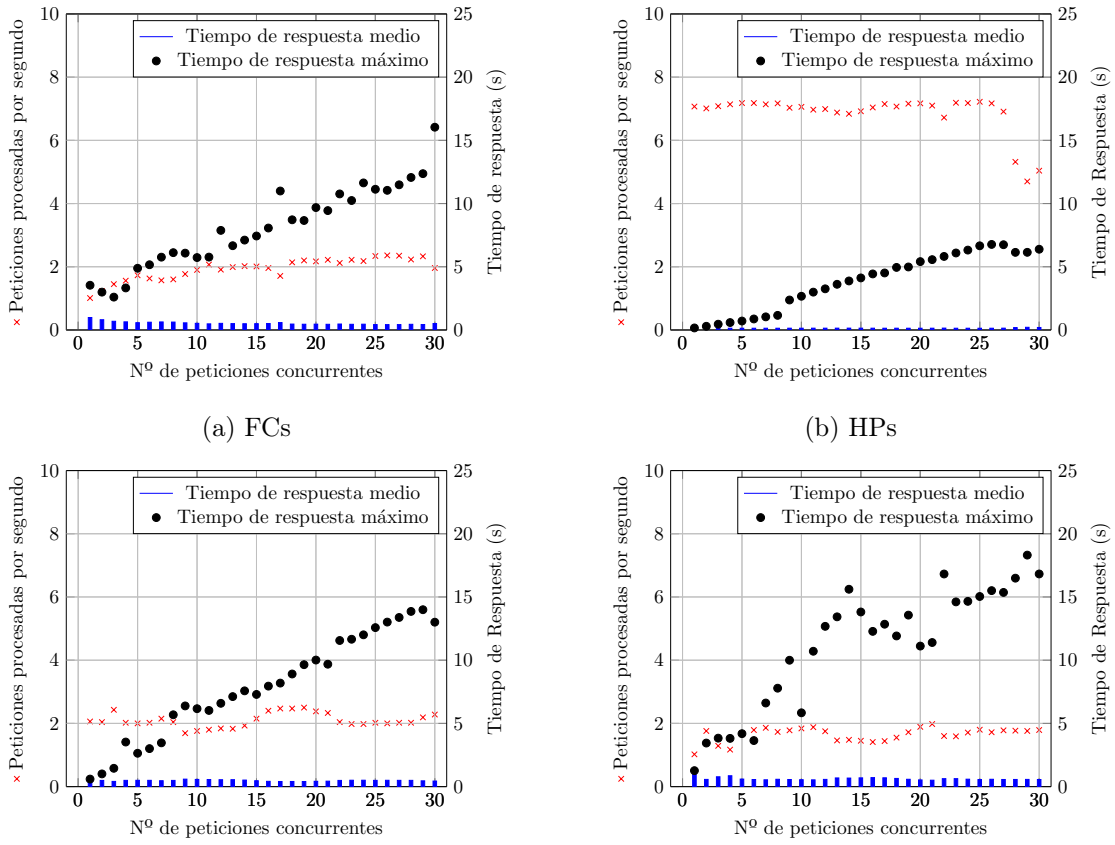
Figura 4.18 Rendimiento del FC ante solicitudes de propiedades y TDs en el caso de estudio B.

siguiente conjunto de experimentos trata de determinar el rendimiento del sistema con este número de variables.

La Figura 4.19a ilustra los resultados obtenidos. En ella se puede observar que el tiempo medio de respuesta para un número creciente de peticiones concurrentes crece de manera lineal. Este comportamiento es el esperado dado que los dispositivos *gateway* de los FC utilizan *firmware* que se ejecuta sobre MicroPython, el cual posee un solo hilo de ejecución, que le obliga a un tratamiento en serie de las peticiones. El número de peticiones que es capaz de procesar el FC, independientemente del número de peticiones recibidas, es aproximadamente de dos peticiones por segundo. Observando los datos concretos para 15 peticiones concurrentes, se puede ver cómo se comportaría el sistema en el caso de que todas las propiedades sean invocadas al mismo tiempo, con un tiempo de respuesta medio para cada petición de 0,5 s, que se mantiene estable en todos los casos.

MIMO y HPs

El MIMO es un dispositivo que debe gestionar múltiples variables y requiere de un funcionamiento rápido y accesible, ya que es crítico para la gestión de la energía del edificio. El objetivo de estas pruebas es determinar el rendimiento del *gateway* del MIMO, evaluando su capacidad de respuesta en diferentes condiciones de carga. Las variables que expone este dispositivo se desglosan en la Tabla C.1 del Apéndice C.



(a) FCs

(b) HPs

(c) MIMO

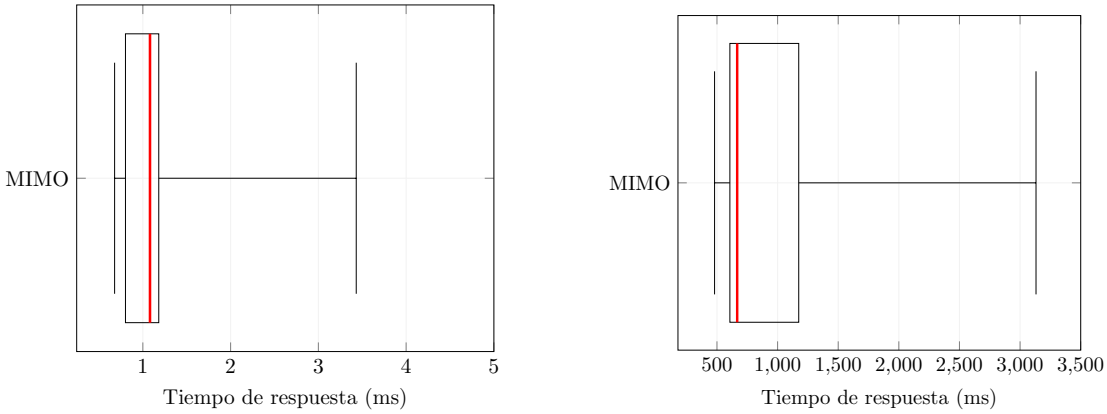
(d) PLC.

Figura 4.19 Pruebas de carga de diferentes componentes del sistema en el caso de estudio B.

Las dos HP que integran el sistema de bombeo de agua caliente en el despliegue realizado en el edificio se encuentran conectadas a la red interna del mismo. Estas máquinas exponen sus variables a través de un servidor Modbus TCP. Las variables expuestas por las HP se recogen en la Tabla C.2 del Apéndice C.

Estos experimentos en las HP pretenden evaluar el rendimiento del *gateway* encargado de transformar las peticiones Modbus TCP a la interfaz requerida por la arquitectura WoT propuesta. Además, se podrá comprobar la sobrecarga introducida por la inclusión de la arquitectura propuesta en comparación con el protocolo IoT genérico Modbus. Es decir, será posible observar una comparativa entre el propio protocolo Modbus, que usa el protocolo TCP de manera directa con las máquinas y obteniendo resultados en crudo, respecto a la solución implementada siguiendo el paradigma WoT, que expone las mismas propiedades, pero con un contexto, meta-información y parametrización que permite interpretar y usar los datos directamente. Mediante esta comparativa se puede determinar el tiempo de cómputo extra que requiere añadir esta capa WoT, y poder así decidir la utilidad de su implementación en este tipo de casos.

Por un lado, se han realizado 1000 peticiones Modbus TCP a uno de los registros del MIMO para medir el tiempo de respuesta de cada petición. Estas peticiones se realizan con un software de medición personalizado desarrollado en Python, utilizando el propio reloj de Python que ofrece una resolución de $\pm 1\mu$s. En la Figura 4.20a se pueden observar los tiempos de respuesta máximo, mínimo y medio, así como los percentiles 25 y 75. Por otro lado, se han realizado 1000 peticiones HTTP al *servient* WoT que aloja la representación de la *Thing* correspondiente al MIMO, utilizando la herramienta Apache Benchmark, obteniendo los resultados mostrados en la Figura 4.20b.



(a) Interfaz Modbus TCP.

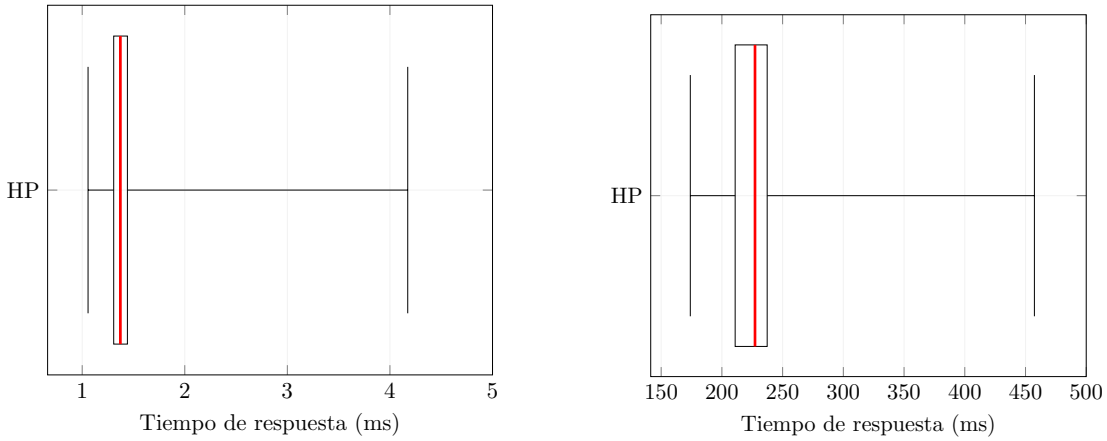
(b) Interfaz WoT.

Figura 4.20 Tiempo de respuesta del *gateway* del MIMO en el caso de estudio B.

De forma análoga, con el objetivo de medir el rendimiento del *gateway* de las HPs, se realizan 1000 peticiones a uno de los registros del servidor Modbus alojado en cada una de las HPs. Estos experimentos se realizan con un software de medición personalizado, desarrollado en Python. Igualmente, debido a su proximidad y similitud, se agregan los resultados de ambas HP, que se muestran en la Figura 4.21a.

Por otro lado, se han realizado 1000 peticiones HTTP al *servient* WoT de cada HP que aloja la representación de la *Thing* correspondiente, utilizando también Apache Benchmark. Al ser ambas máquinas idénticas, fabricadas e instaladas por el mismo fabricante, encontrarse a una distancia física similar del nodo de pruebas y disponer de una cantidad idéntica de propiedades, los datos de ambos experimentos se han agregado en uno sola gráfica, mostrada en la Figura 4.21b.

De los resultados obtenidos de estos experimentos se infiere, como era de esperar, que el tiempo de respuesta ante la solicitud de mediciones de variables concretas, y por tanto, la capacidad de obtener datos en tiempo real, se puede conseguir a través de Modbus TCP sin el retardo introducido por la arquitectura WoT. El tiempo de respuesta medio, obtenido teniendo en cuenta todas las peticiones Modbus TCP realizadas en estos experimentos, es de 1,37 ms, mientras que las mismas peticiones, realizadas a través de todas las capas de la



(a) Interfaz Modbus TCP.

(b) Interfaz WoT.

Figura 4.21 Tiempo de respuesta de los *gateways* de las HPs en el caso de estudio B.

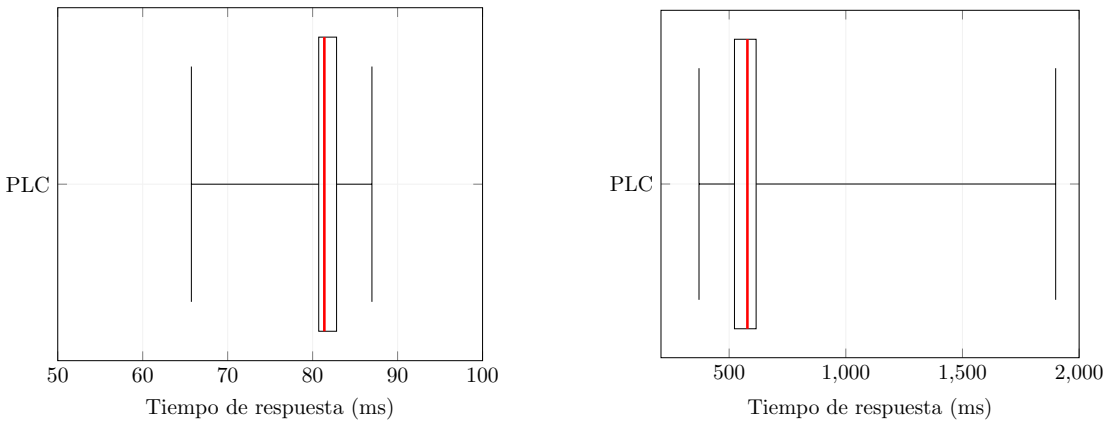
arquitectura WoT propuesta, obtienen un tiempo de respuesta promedio de 666,74 ms en el caso del MIMO y de 227,26 ms en el caso de las HPs.

A continuación, se ejecutan experimentos de carga variando el nivel de concurrencia sobre el MIMO y las HPs. Los resultados obtenidos en estas pruebas, se muestran en la Figura 4.19c y en la Figura 4.19b. En ellas se puede observar cómo el número de peticiones procesadas por unidad de tiempo apenas varía, mientras que el tiempo de respuesta medio y máximo aumentan de manera lineal, tal y como es esperable al aumentar el número de peticiones.

PLC

Como se ha descrito anteriormente, el PLC se encarga de gestionar la infraestructura base del sistema de distribución de agua caliente en el edificio. Para ello, expone una serie de variables que son accesibles a través de un protocolo específico, como se recoge en la Tabla C.3 del Apéndice C. Para su lectura y escritura, así como para la realización de estas pruebas, se ha utilizado la librería *python-snap7* de Python 3, que implementa el protocolo de comunicación Snap7, en combinación con código desarrollado específicamente para estas pruebas y que utiliza un contador de tiempo integrado en Python para medir el tiempo de respuesta de las peticiones Snap7. Los resultados se pueden observar en la Figura 4.22a. Por otro lado, se ha medido el tiempo de respuesta en el acceso a las mismas variables utilizando la solución propuesta mediante la arquitectura WoT a través de Apache Benchmark, cuyos resultados se muestra en la Figura 4.22b. Al igual que en los casos anteriores se pone de manifiesto la sobrecarga introducida por la arquitectura WoT.

A continuación, se realizaron pruebas de concurrencia para determinar el rendimiento ante diferentes niveles de carga del *servient* WoT desarrollado para gestionar el PLC. La Figura 4.19d resume los resultados obtenidos en estas pruebas.



(a) Interfaz Snap7.

(b) Interfaz WoT.

Figura 4.22 Tiempo de respuesta del PLC en el caso de estudio B.

4.2.7. Conclusiones

En esta sección se ha descrito la implementación, el despliegue y la validación de una solución, siguiendo la arquitectura propuesta en la sección 3.2, para un edificio de viviendas recientemente rehabilitado. En este caso de estudio se han probado todos los componentes de una arquitectura WoT, que sigue las recomendaciones del W3C, para el soporte de un sistema BEMS. El diseño y posterior despliegue ha permitido la recopilación de los datos que se relacionarán con la gestión de energía del edificio. Todos los datos se han obtenido mediante comandos estándar dirigidos a URLs simples, lo que garantiza la interoperabilidad de todos los dispositivos del sistema. Mediante este caso de estudio se ha podido validar el diseño y el despliegue de la arquitectura en un entorno real.

Además, cuando ha sido posible se han comparado con el rendimiento producido al utilizar protocolos IoT. En estos casos se determina que existe una gran diferencia en términos de tiempo de respuesta al proporcionar propiedades expuestas por dispositivos en el sistema cuando se utiliza un protocolo asociado al paradigma IoT, respecto a solicitudes similares procesadas por la arquitectura propuesta siguiendo el paradigma WoT. Esta diferencia se debe al contenido extra y al encapsulado de mensajes que requiere una petición WoT, ya que estas se realizan a través de HTTP y contienen metainformación que aporta tanto significado al dato, como localidad a la *Thing* u otras referencias que permiten no solo obtener el propio dato, sino también interpretarlo correctamente sin necesidad de recursos externos. Es decir, de manera muy resumida, la solución implementada siguiendo el paradigma WoT consume recursos, pero aporta mucha más información. Una hipotética solución implementada conectando estos dispositivos exclusivamente mediante protocolos IoT sería más rápida, pero carecería de un contexto intrínseco y requeriría complementarla de metadatos sujetos al modelo de organización que prefiera el programador o usuario que desee obtenerlos.

4.3. Lógica de control

Una vez desplegados todos los componentes del sistema, y verificada tanto la conectividad como la comunicación entre todos ellos, se propone, a modo de ejemplo mínimo para la validación de la arquitectura como soporte a un sistema de tipo BEMS, la implementación de un sistema experto basado en reglas para la monitorización y la gestión energética de edificios. Este sistema experto se basa en la información proporcionada por los sensores y actuadores con los que interactuará a través de tecnologías Web siguiendo el paradigma WoT y, más concretamente, las recomendaciones del W3C para arquitecturas WoT. El sistema experto se elige, en detrimento de otros modelos, ya que es un requisito del proyecto HEART (ver Sección 1.2.1), y será la base sobre la que acometer ampliaciones en el futuro.

El sistema experto que se plantea en este ejemplo es muy simple, y trata de emular las decisiones que tomaría, basándose en su experiencia, un experto humano. El sistema propuesto se compone de cinco componentes fundamentales:

- *Proxy* WoT: maneja todos los dispositivos desplegados y al que se consulta para acceder a ellos.
- Base de datos: almacena, entre otros, todos los valores recopilados de las propiedades publicadas por cada dispositivo del sistema.
- Administrador de dispositivos: lee los datos del *proxy* WoT y los almacena en la base de datos.
- Monitor de dispositivos: permite visualizar en tiempo real el funcionamiento de los dispositivos, así como del sistema BEMS.
- Motor BEMS: implementa la lógica del sistema experto y se encarga de la toma de decisiones en función de los datos recabados por el administrador de dispositivos.

Como se ha mencionado en secciones anteriores (ver Sección 4.1.2 y Sección 4.2.2) y se detalla en la Sección A.4 del Apéndice A, el BGC y su correspondiente catálogo de dispositivos son el enlace entre los componentes de monitorización y actuación del edificio y cualquier otro sistema que necesite acceder a ellos, en este caso, el sistema experto. El BGC gestiona las peticiones de lectura de propiedades, así como los mensajes de control remitidos, que emita el sistema experto, permitiendo que la lógica acceda directamente a sensores y actuadores, utilizando mensajes HTTP simples que se implementan en el motor BEMS.

La lógica del sistema experto es una combinación de datos recabados en tiempo real y un conjunto de normas. Esta combinación conforma los datos de entrada del conjunto de reglas propuesto. La consecuencia al resultado obtenido de esta lógica es la activación de ciertos dispositivos del edificio, por ejemplo, modificar un termostato, abrir una válvula, bombear

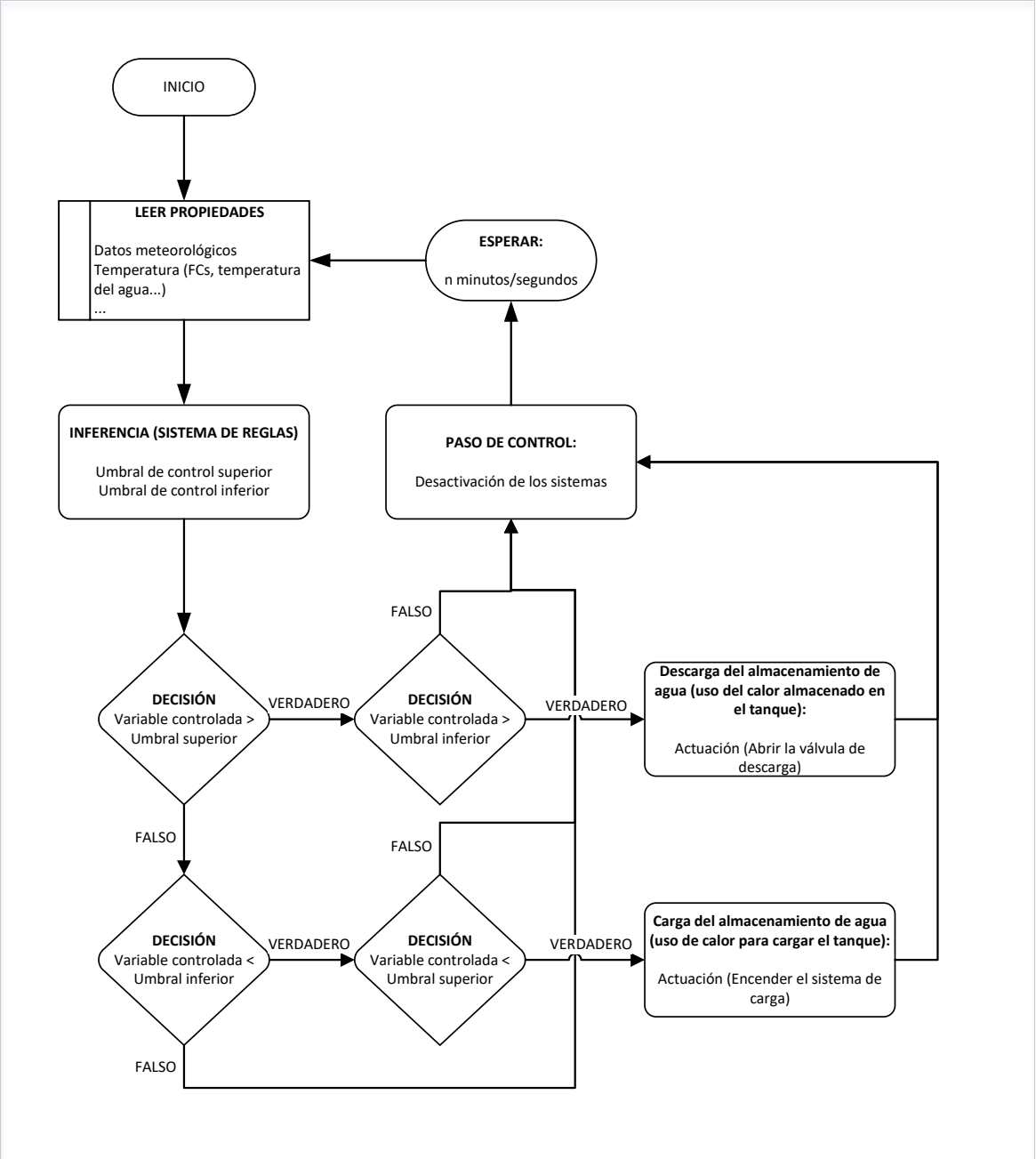


Figura 4.23 Ejemplo de control de temperatura de una estancia mediante un conjunto de reglas del sistema experto.

agua a un depósito, etc. La Figura 4.23 muestra un diagrama de flujo que permite controlar la temperatura de una estancia mediante el sistema experto.

La lógica de control en esta propuesta de ejemplo se implementa en Python 3. Como muestra, el siguiente fragmento de código representa la implementación de una de las reglas

presentes en el sistema, que en este caso se debe activar cuando se requiere la descarga de un tanque de agua:

```
@Rule(
    AS.storageDeviceStatus << StorageStatusFact(
      id=MATCH.idDevice,
      value=MATCH.currentValue,
      status=NE('discharging')),
    StorageConfigurationFact(
      id=MATCH.idDevice,
      upper_threshold=MATCH.upperThreshold,
      lower_threshold=MATCH.lowerThreshold,
      upper_operation_limit=MATCH.upperOperationLimit,
      lower_operation_limit=MATCH.lowerOperationLimit),
    TEST(lambda upperThreshold,
    currentValue : currentValue > upperThreshold),
    TEST(lambda lowerOperationLimit,
    currentValue : currentValue > lowerOperationLimit),
    salience=0)

def storage_discharge(self, idDevice, storageDeviceStatus,
currentValue, upperThreshold, lowerThreshold):
    """Descarga el tanque si el valor actual es mayor que
    el l mite superior as  como el inferior"""
    print('STORAGE '{0}':
      LOWER [{1}]
      UPPER [{2}]
      CURRENT [{3}]
      ACTION -> [{4}]'.format(
        idDevice,
        lowerThreshold, upperThreshold,
        currentValue,
        'DISCHARGE'))
    self.dataManager.saveStatus(StorageStatus(
        id=idDevice,
        value=currentValue,
        status='discharging'))
    self.modify(storageDeviceStatus, status='discharging')
```

La base de datos que da soporte a este sistema se construye mediante InfluxDB, una base de datos de series temporales de código abierto, diseñada para el manejo de grandes cantidades de datos y altas tasas de lectura y escritura. Esta base de datos proporciona consultas y escrituras en tiempo real, así como una API HTTP para la gestión de dichas operaciones.

En la arquitectura propuesta para la lógica de control, todos los componentes se implementan en Docker, y son orquestados por Docker Compose. Así, el módulo de software BEMS implementa el motor de reglas para administrar el flujo de energía del sistema. Este componente se basa en Experta[28], una biblioteca de Python para construir sistemas expertos, que permite el desarrollo de sistemas basados en una solución heurística.

El software se despliega en el BGC del caso de estudio B. Este sistema recopila continuamente información sobre el estado de los dispositivos en el edificio, actúa en consecuencia de los datos recibidos y los expone de forma visual a través de paneles intuitivos.

El motor de reglas en el BEMS se ejecuta periódicamente (el intervalo de tiempo se puede fijar de segundos a minutos, o en base a algún evento determinado), reuniendo los valores más recientes de la base de datos de series temporales y actualizando el estado y los parámetros del sistema en la misma base de datos en consecuencia.

La interfaz gráfica de usuario de BEMS se compone de varios paneles. El comportamiento de todo el sistema se visualiza utilizando series de tiempo y representaciones intuitivas. Un ejemplo de tales visualizaciones se puede observar en la Figura 4.24, que representa cuándo un dispositivo de almacenamiento PCM o un tanque de agua se está cargando, descargando o inactivo (representado en verde, rojo o gris, respectivamente, en la parte superior del panel, identificada como *Thermal Storage Status*). Además, también se muestra información sobre el nivel de almacenamiento de agua en cada dispositivo (*PCM Storage* y *DHW Tank*). Los datos que se muestran en la figura se han simulado mediante un script que hace fluctuar la temperatura del almacenamiento para ir por debajo y por encima de los umbrales definidos periódicamente, de modo que se puede observar el ciclo de carga/descarga. Este ciclo es arbitrado por el sistema experto y en la figura se observa que la temperatura se mantiene en los umbrales definidos para el sistema.

Otro panel permite al usuario monitorizar el estado de los FCs, que depende de la energía térmica involucrada en el intercambio de energía. La Figura 4.25 muestra un ejemplo de este panel con el estado de los FCs así como algunas de sus propiedades. En este *dashboard*, el ID de piso y FCs se puede filtrar para mostrar FCs individuales.

Normalmente, el número de depósitos de agua caliente en un edificio suele ser entre uno y diez, mientras que el número de FCs puede llegar a varios centenares. Por lo tanto, se ha sometido a este prototipo de lógica de control a pruebas de carga para garantizar que pueda manejar correctamente cientos de dispositivos de tipo FC. Para ello se han añadido FCs

[28]https://pypi.org/project/experta/

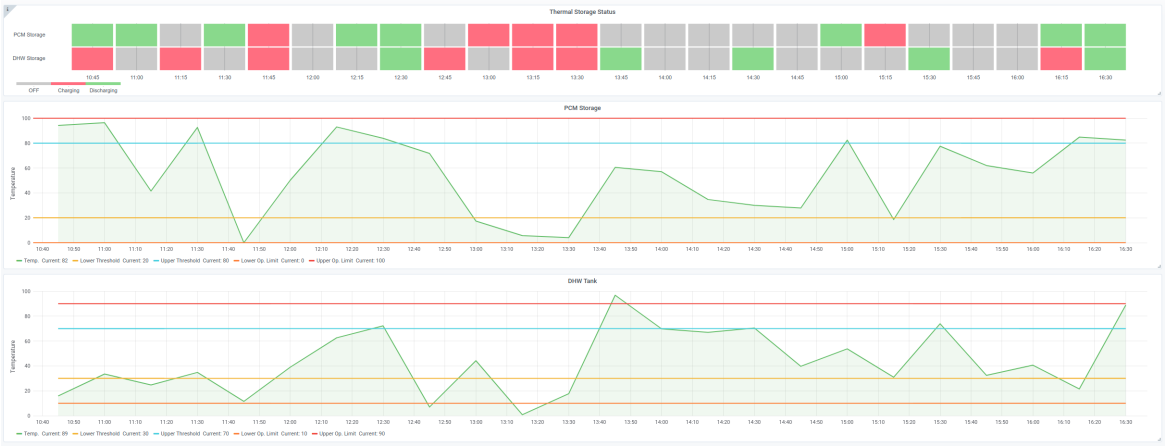


Figura 4.24 Visualización del estado de un almacén PCM y de un tanque de agua del edificio.

Figura 4.25 Visualización del estado y algunas propiedades de los FCs.

simulados al BEMS. La adición de estos FCs simulados afecta de manera lineal al rendimiento del sistema de reglas, requiriendo en promedio 2,41 segundos para procesar las reglas de 150 FCs. Este resultado se obtiene como el promedio de 5 minutos de iteraciones constantes del motor de reglas sobre diferentes conjuntos de datos simulados.

Capítulo 5

Conclusiones

En este capítulo se recogen las principales aportaciones a las que ha dado lugar el desarrollo de esta tesis doctoral, se detallan las publicaciones científicas relacionadas y se describen los trabajos de investigación que se podrían derivar y que podrían ampliar los objetivos origianales de esta investigación.

5.1. Aportaciones

Esta tesis doctoral es el resultado del trabajo de investigación del autor desarrollado durante varios años en el campo de la interoperabilidad de dispositivos heterogéneos conectados a través de redes inalámbricas, utilizando como paradigmas tanto *Internet of Things* como *Web of Things*, en este último caso siguiendo las recomendaciones del W3C, en el marco de los proyectos *HEART* y *Web of Things para entornos inteligentes y eneergéticamente eficientes* en CTIC Centro Tecnológico (ver Sección 1.2).

La primera de las aportaciones de este trabajo de investigación se logra al tratar de responder a la primera pregunta de investigación planteada en la tesis, RQ1: ¿cuáles son los estándares que permiten la interconexión, de forma transparente, de dispositivos físicos heterogéneos en redes inalámbricas? La respuesta a esta pregunta se consigue tras una **revisión exhaustiva del estado del arte de estándares, recomendaciones, arquitecturas y tecnologías relacionadas con *Internet of Things* y con *Web of Things***, que se detalla en el Capítulo 2. De este análisis se desprende que, si bien existen varios estándares que persiguen la interoperabilidad de dispositivos en IoT, estos llevan a crear soluciones IoT verticales que, en muchas ocasiones, derivan en silos IoT, tal y como se recoge en la Sección 2.1. Por otro lado, el paradigma *Web of Things* persigue utilizar estándares y tecnologías Web ampliamente extendidas que permitan crear aplicaciones y servicios que de otra forma no podrían ser accesibles mediante IoT. En la Sección 2.2 se describen los principales estándares y recomendaciones relacionados con este paradigma. Dado que el W3C es el principal organismo de estandarización internacional para la Web, se realiza un análisis

exhaustivo de sus recomendaciones para WoT, que se recoge en la Sección 2.3. Además, también **se revisan las arquitecturas propuestas en la literatura científica que dan soporte a los sistemas de gestión de energía en edificios**, en la Sección 3.1, con lo que se responde a la segunda pregunta de investigación planteada, RQ2: ¿en qué estándares se basan las arquitecturas de software para el soporte de sistemas de gestión de energía en edificios? Los trabajos iniciales de investigación con prototipos WoT se centraron en la monitorización de parámetros ambientales, y dieron lugar a las publicaciones *Indoor air quality monitoring sensor for the Web of Things* y *An IoT platform for indoor air quality monitoring using the Web of Things*, que se detallan en la Sección 5.2.

La segunda aportación de este trabajo es la propuesta de una **arquitectura para el soporte de sistemas de gestión de energía en edificios mediante el paradigma *Web of Things***. Esta arquitectura se detalla en la Sección 3.2 y su objetivo es proporcionar interoperabilidad a todos los dispositivos del sistema a través de la capa de aplicación mediante el uso de estándares y tecnologías Web siguiendo las recomendaciones del W3C. La arquitectura propuesta proporciona una solución basada en WoT a un problema de sensorización de edificios para dotarlos de capacidades de monitorización y control de parámetros ambientales, con el objetivo de realizar una gestión eficiente de sus sistemas de calefacción y refrigeración. Esta arquitectura, cuya implementación se detalla en el Apéndice A, da respuesta a la tercera pregunta de investigación planteada en esta tesis, RQ3: ¿cómo se puede dar soporte a sistemas de gestión de energía en edificios mediante el paradigma *Web of Things*? Si bien esta solución añade más latencia que una basada en el uso directo de protocolos de capas inferiores, aporta interoperabilidad con cualquier dispositivo de cualquier fabricante y proporciona metadatos que enriquecen las aplicaciones finales de usuario. La arquitectura propuesta se ha desplegado en dos escenarios concretos: un edificio de oficinas, a modo de caso de estudio de laboratorio (ver Sección 4.1), y un edificio de viviendas, a modo de caso de estudio real (ver Sección 4.2), y ha dado lugar a las publicaciones *Monitoring and control of energy consumption in buildings using WoT: A novel approach for smart retrofit* y *An Expert System for Building Energy Management through the Web of Things*. Estos dos despliegues han permitido validar la arquitectura propuesta, y con ello las contribuciones de esta tesis.

5.2. Publicaciones relacionadas

El trabajo de investigación desarrollado en estos años ha sido difundido a través de un artículo en una revista indexada en el *Journal Citations Report* (JCR), en el primer decil de su categoría (posición 2 de 68), y de tres artículos presentados en congresos internacionales. A continuación, se muestra el listado de publicaciones relacionadas con esta tesis doctoral, en orden cronológico inverso:

- *Monitoring and control of energy consumption in buildings using WoT: A novel approach for smart retrofit.* D. Ibaseta, A. García, M. Álvarez, B. Garzón, F. Díez, P. Coca, C. Del Pero, J. Molleda. *Sustainable Cities and Society*, Vol. 65, February 2021, 102637. Elsevier. (JCR: 10.696 - Q1 [2021]). `http://dx.doi.org/10.1016/j.scs.2020.102637`

- *An Expert System for Building Energy Management through the Web of Things.* D. Ibaseta, J. Molleda, F. Díez, M. Álvarez. *15th International Conference on Hybrid Artificial Intelligence Systems, 2020* (HAIS 2020). `http://dx.doi.org/10.1007/978-3-030-61705-9_39`.

- *An IoT platform for indoor air quality monitoring using the Web of Things.* D. Ibaseta, J. Molleda, F. Díez, J. C. Granda. *27th International Conference on Modelling, Monitoring and Management of Air Pollution, 2019.* `http://dx.doi.org/10.2495/AIR190051`.

- *Indoor air quality monitoring sensor for the Web of Things.* D. Ibaseta, J. Molleda, F. Díez, R. Usamentiaga. *2nd International Research Conference on Sustainable Energy, Engineering, Materials and Environment, 2018* (IRCSEEME 2018). `http://dx.doi.org/10.3390/proceedings2231466`.

Esta tesis doctoral se ha desarrollado en el marco de un proyecto de investigación industrial definido a partir de un subconjunto de tareas relacionadas con la interoperabilidad de dispositivos heterogéneos dentro de un proyecto de investigación europeo centrado en la rehabilitación de edificios bajo criterios de eficiencia energética (ver Sección 1.2). Por ello, el trabajo realizado ha contribuido también a la publicación del siguiente documento relacionado con el proyecto:

- *HEART D4.8 - Final BEMS Hardware Components.* M. Álvarez-Espinar *et al.* January 2020. `https://heartproject.eu/wp-content/uploads/2021/08/D4.8-Final-BEMS-Hardware-Components.pdf`

La metodología de investigación basada en un enfoque constructivo que guió el desarrollo de esta tesis, descrita en la Sección 1.3, determina en su fase tercera que se debe consolidar la contribución del trabajo desarrollado al estado del arte. Todas las publicaciones del listado anterior contribuyen al del estado del arte en la temática relacionada con esta tesis aunque, sin duda, la contribución al desarrollo del estado del arte se logra con la publicación *Monitoring and control of energy consumption in buildings using WoT: A novel approach for smart retrofit* dado que ha sido referenciada en el artículo *A survey on the Web of Things* en *IEEE Access* [52]. El hecho de que una publicación científica derivada del trabajo desarrollado en esta tesis doctoral haya sido referenciada en un *survey* reciente sobre *Web of Things* valida la contribución del trabajo realizado al estado del arte en este ámbito.

5.3. Trabajo futuro

Las contribuciones de este trabajo de investigación propician varias líneas de investigación que podrían ser abordadas en trabajos posteriores para ampliar los objetivos iniciales de esta tesis:

- **Seguridad**. En la arquitectura propuesta no se han tenido en cuenta aspectos relacionados con la seguridad, lo que supone una carencia importante en la solución proporcionada. El diseño e implementación de mecanismos de seguridad *end-to-end* es un reto de investigación en sí mismo en el campo de IoT. Para la arquitectura propuesta se podría evaluar la adopción de las recomendaciones de los grupos de trabajo del *W3C Web of Things*. Una línea de investigación futura se podría constituir en base al diseño de sistemas de control inteligentes utilizando la arquitectura propuesta en los que los agentes y el resto de componentes software adoptaran el enfoque *secure by design*, en el que las estrategias de seguridad se vean reforzadas por la propia arquitectura.

- **Redes**. La arquitectura propuesta en este trabajo se podría extender a otras redes, por ejemplo redes de tipo LPWAN como 5G, LoRa o SigFox, mediante la implementación de adaptadores o *protocol bindings* para ellas. El acceso a este tipo de redes incrementaría la tipología de escenarios en los que se podría utilizar la solución planteada.

- **Casos de estudio**: La arquitectura propuesta en este trabajo se ha desplegado en dos casos de estudio: un edificio de oficinas y un edificio de viviendas. Para evaluar el uso de la arquitectura en entornos diferentes, se podrían plantear casos de estudio en edificios singulares tales como centros comerciales o terminales de aeropuertos, entre otros, o en otros ámbitos, como por ejemplo en el control de la climatización en invernaderos en el campo de la agricultura de precisión.

- **Analítica de datos**. La inclusión de técnicas avanzadas de analítica de datos constituye otra posible ampliación de este trabajo. Por ejemplo, proporcionando acceso a *frameworks* de *machine learning* en la propia arquitectura, de tal forma que se pudieran obtener análisis de datos en tiempo real que guiaran el proceso de toma de decisiones de las aplicaciones finales de usuario.

Apéndice A

Implementación base de la arquitectura propuesta

La arquitectura planteada en el Capítulo 3 constituye una referencia genérica para la implementación de sistemas BEMS. Para construir sistemas de este tipo en base a esta referencia, se deben tomar varias decisiones de implementación, en algunos casos ligadas a los dispositivos hardware que finalmente utilizará el sistema. En la implementación de la arquitectura, varios módulos tanto software como hardware serán reutilizables entre diferentes casos de uso, y se describen en este apéndice. La implementación de la arquitectura propuesta hará uso de dos librerías que cumplen con las recomendaciones del W3C para arquitecturas WoT: uWoT, que permite integrar en una arquitectura WoT dispositivos embebidos o empotrados con recursos computacionales muy limitados (ver Apéndice B), y WoTPy, una implementación experimental plenamente funcional de un *framework* para el desarrollo de aplicaciones WoT [28] que se puede utilizar en sistemas con recursos computacionales más elevados.

A.1. Gateways MIMO/HP

A modo de ejemplo de implementación, se utilizarán dispositivos MIMO y HP con interfaz Modbus[1]. Por tanto, en el MIMO y en la HP se integrará un servidor Modbus TCP. Esta implementación se compone de dos partes. Una parte utiliza la librería WoTPy para exponer las distintas variables, las cuales vendrán definidas en un fichero de configuración. La otra parte, requiere la definición de un fichero Docker Compose, que permitirá desplegar el software en la máquina objetivo de manera sencilla y universal, independientemente de dónde se quiera ejecutar.

La librería WoTPy permite definir un protocolo para el cual se ejecutará el *servient* WoT. En este caso se utilizará el protocolo HTTP, que será el utilizado para las comunicaciones

[1]https://www.modbus.org/

en todo el sistema. Para este cometido, el módulo software que se ejecutará en los *gateways* de estos dispositivos dispone de una clase *Main*, en la cual se implementarán el resto de características. En esta clase se definen los métodos de comunicación con el servidor Modbus TCP objetivo. Las principales funciones que se implementan en esta clase se muestran en la Tabla A.1.

Método	**Descripción**
main()	Constructor que obtiene las propiedades definidas en el fichero de configuración y las añade de manera pertinente a la TD, creando el *servient* y desplegando los servicios en los puertos indicados.
read_value(register)	Obtiene el último valor almacenado de un registro.
write_register(register, value)	Escribe un valor en un registro dentro del servidor Modbus TCP.
read_register(register)	Lee un valor de un registro dentro del servidor Modbus TCP.
create_modbus_reader_task()	Genera un proceso que lee cada *s* segundos todos los registros listados en el fichero de configuración y los almacena en memoria. Ciertas variables se pueden almacenar de manera temporal (por ejemplo en una base de datos de series temporales) para su posterior utilización; estas variables se identifican con el flag *transform* con el valor *cumulative* en el fichero de configuración.

Tabla A.1 Métodos en el *firmware* de los *gateway* del MIMO y de la HP.

Por otro lado, se debe proporcionar un fichero de configuración en el que se almacenan los variables a leer del servidor Modbus TCP, así como los diferentes parámetros que las definen. El fichero se compone de una serie de elementos que marcan las propiedades a leer en la HP y/o en el MIMO. Estos elementos contienen la información necesaria para obtener los diversos datos a través de la conexión Modbus TCP. Es necesario tanto un identificador del esclavo Modbus como una dirección de registro, que corresponde con una de las direcciones que se pueden usar en un servidor Modbus para alojar los datos de los sensores. Los parámetros que componen cada propiedad se muestran en el siguiente ejemplo:

```
{
  "Outdoor_temperature":{
    "value":0,
    "description":"Unit in 0,1 C ",
    "slave_addr":"1",
    "starting_address":10,
    "register_quantity":1,
```

```
    "signed":true,
    "type":"ANALOG_INPUT",
    "wot_type":"property",
    "wot_unit":"DeciDEG_C",
    "transform": "decimal"
  },
}
```

Cada uno de los parámetros representa:

- *value*: el último valor leído del servidor Modbus TCP.
- *description*: la descripción legible de la variable, qué representa y en qué unidades.
- *slave_addr*: el identificador del esclavo Modbus dentro del servidor Modbus TCP.
- *starting_address*: la dirección de inicio de la variable en la memoria del servidor Modbus TCP.
- *register_quantity*: número de direcciones de memoria a leer a partir de la dirección inicial.
- *signed*: si la variable tiene valores negativos o no.
- *type*: tipo de registro Modbus (*ANALOG_INPUT, HOLDING_REGISTER,* etc.).
- *WOT_type*: unidad estándar para representar el dato dentro de WoT, por ejemplo: grados Celsius en formato decimal(DeciDEG_C).
- *transform*:
  - *"decimal"*: indica que el valor ha de transformarse a formato decimal directamente.
  - *"cumulative"*: indica que es un valor acumulado y que debe ir guardándose y sumándose al de la base de datos.

Finalmente, el software de estos *gateways* se despliega a través de un fichero Docker Compose, que integra múltiples piezas de software de otros componentes del BEMS. En este fichero se define la dirección IP del servidor Modbus TCP, los puertos que utilizará el *servient* WoT, así como la dirección y el puerto de la base de datos a utilizar. El servicio que integra este componente se implementa, como ejemplo, de la siguiente manera dentro del fichero *compose*:

```
...
mimo_gateway:
  container_name: mimo_gateway
  image: heart-bgc-mimo-gateway
  restart: always
  environment:
    - MODBUS_SLAVE_IP=192.168.2.1
    - MODBUS_SLAVE_PORT=502
    - PORT_CATALOGUE=9090
    - PORT_WS=9091
    - PORT_HTTP=9092
    - INFLUXDB_HOST=localhost
    - INFLUXDB_PORT=55556
    - INFLUXDB_USERNAME=root
    - INFLUXDB_PASSWORD=root
    - INFLUXDB_DATABASE=heart_mimo
  ports:
    - "9090:9090"
    - "9091:9091"
    - "9092:9092"
  network_mode: host
...
```

A.2. Gateway PLC

El software que se diseña para la integración del PLC con el resto de componentes del BEMS permite la consulta y modificación de registros de un PLC utilizando el protocolo *Snap7*. El uso de este protocolo no limita la generalidad de la arquitectura, ya que podría utilizarse otro protocolo en función del hardware finalmente seleccionado. Esta implementación se compone de dos partes. Una parte que utiliza la librería WoTPy para exponer las distintas variables, que vendrán definidas en un fichero de configuración. La otra parte se corresponde con la definición de un fichero Docker Compose, que permitirá desplegar el software en la máquina objetivo de manera sencilla y universal, independientemente de la máquina donde se quiera ejecutar.

Como se ha indicado anteriormente, la librería WoTPy permite definir un protocolo para el cual se ejecutará el *servient* WoT, en este caso se utilizará el protocolo HTTP. En este módulo software, la clase *Main*, en la cual se implementarán el resto de características, define

los métodos de comunicación con el PLC. Las principales funciones que se implementan en esta clase se muestran en la Tabla A.2.

Método	**Descripción**
main()	Constructor que obtiene las propiedades definidas en el fichero de configuración y las añade de manera pertinente a la TD, creando el *servient* y desplegando los servicios en los puertos indicados.
read_value(register)	Obtiene el último valor de un registro almacenado en memoria.
read_register(db, offset, is_dint)	Lee un registro de una base de datos concreta dentro del PLC, indicando el registro con el offset y obteniendo el dato en función de si es un número entero o un real.
def create_plc_reader_task()	Tarea que se ejecuta continuamente leyendo todas las posiciones de memoria del PLC y almacenándolas en memoria.

Tabla A.2 Métodos en el *firmware* del *gateway* del PLC.

Como en el caso anterior, este módulo software se despliega a través de un fichero Docker Compose. En este fichero se define la dirección IP del PLC dentro de la red del edificio y los puertos que utilizará el *servient* WoT. El servicio que integra este componente se implementa, como ejemplo, de la siguiente manera dentro del fichero *compose*:

```
...
plc_gateway:
  container_name: plc_gateway
  image: plc-gateway
  restart: always
  environment:
    - PLC_IP=192.168.1.10
    - PLC_PORT=102
    - PLC_RACK=0
    - PLC_SLOT=1
    - PORT_CATALOGUE=30000
    - PORT_WS=30001
    - PORT_HTTP=30002
  ports:
    - "30000:30000"
    - "30001:30001"
    - "30002:30002"
...
```

A.3. Gateways FCs

El el Apéndice B se plantea una implementación de las recomendaciones W3C para arquitecturas WoT denominada uWoT. Estas recomendaciones se implementan en MicroPython[2], un lenguaje de programación para microcontroladores basado en Python y que permite un control total del dispositivo. En este marco, el software de los FCs se plantea como una implementación en este lenguaje que permite la integración de la librería uWoT para la lectura, escritura y procesamiento de los sensores, actuadores y demás elementos inherentes a los FCs. La librería uWoT permite definir una serie de propiedades y acciones que se corresponderán con las propiedades y acciones del FC. Además, permite definir los metadatos de estas propiedades, cómo se capturan los valores de los sensores, cómo se modifican los actuadores y genera automáticamente la TD con la información pertinente.

La implementación del software de los FCs se desarrolla en base a una serie de clases que integran distintas funciones que hacen uso de la librería uWoT. Estas funciones utilizan una librería asíncrona, que permite, de una manera virtual, ejecutar múltiples procesos de forma simultánea. Por otro lado, se implementa también una funcionalidad para actualizar el *firmware* de los controladores del FC sin necesidad de hacerlo manualmente, teniendo que desmontar el sistema cada vez, o ejecutando procesos de actualización manual en remoto. Esto se denomina en sus siglas en inglés OTA (*Over The Air*), y es una tecnología que permite programar, en este caso actualizar, el software de un dispositivo IoT a través de su conexión inalámbrica, en este caso wifi. Para ello se define un servidor en el BGC y un cliente que se ejecuta continuamente en el dispositivo controlador del FC. El cliente es el encargado de, periódicamente, solicitar al servidor las últimas actualizaciones de software.

La clase principal del software de estos dispositivos es la clase *Main*. Esta clase contiene la inicialización del sistema, definiendo las funciones que leerán los sensores, exponiendo el *servient* WoT y declarando las funciones auxiliares inherentes al dispositivo controlador, y necesarias para su correcto funcionamiento. Las funciones principales que se implementan en esta clase se muestran en la Tabla A.3. Estas funciones permiten definir e inicializar el *servient* WoT, así como mantener el controlador en condiciones óptimas para su ejecución. Además de estas funciones, esta clase define una serie de métodos que se asocian con la lectura y/o escritura de los sensores y actuadores. Como ejemplo, se muestra a continuación el método necesario para, en primer lugar, leer un sensor de temperatura, y en segundo lugar, modificar un parámetro del FC:

```
async def DS18b20_Read(self, sensor):
  sleep_time = ASYNC_IO_SLEEP_TIME
  temp = []
  try:
      for i in range(NUM_READ_ATTEMPTS):
```

[2]https://micropython.org/

```
            sensor.start_conversion()
            await asyncio.sleep_ms(sleep_time)
            temp.append(sensor.read_temp_async())

        res = self.median(temp)
        return res

    except Exception as ex:
        self.logger.error(ex)
```

```
async def set_Season(self, value):
    self.logger.info(value)
    if value == 1 or value == 0:
        self.Season_PIN.value(value)
        self.storage.add_Update_Key("season", value)
        return True
    else:
        return False
```

A.4. Building Gateway and Controller

El BGC es el núcleo del sistema que da soporte al BEMS. Su implementación requiere la coordinación de múltiples *gateways*, que convivirán en este dispositivo hardware y que se encargarán de obtener y almacenar los datos de los componentes del BEMS. El BGC, de base, se compone de los siguientes servicios:

- *Gateway* PLC.
- Servicio OTA.
- Panel de control.

De todas formas, el BGC también puede implementar los *gateways* del MIMO y la HP, siempre y cuando los dispositivos se encuentren en la misma red local. A continuación, se explican los servicios que se encuentran en el BGC y que no se habían expuesto previamente. Todos los servicios que se engloban dentro del BGC se ejecutan en contenedores Docker, que permiten la ejecución y gestión paralela de múltiples procesos.

Método	Descripción
__init__(logger, broker, ssid, ssid_password, id, name, context, type, description, server="HTTP")	Constructor que obtiene las propiedades definidas en el fichero de configuración y las añade de manera pertinente a la TD, creando el *servient* y desplegando los servicios en los puertos indicados. También define los sensores y actuadores y los enlaza con las distintas propiedades y acciones.
async action_Restart()	Reinicia el controlador. Se usa como herramienta para ciertos eventos que puedan bloquear el proceso principal.
async action_Update()	Inicia el proceso de actualización del *firmware* del controlador.
async get_WiFi_RSSI()	Obtiene el valor de la potencia del wifi que recibe el controlador.
async restart_Procedure_Daemon()	Proceso asíncrono que espera a la señal de reinicio para reiniciar el controlador de manera segura.
async OTA_Update_Daemon()	Proceso que se encarga de recibir la señal de actualización del controlador, iniciando los procesos necesarios para sustituir el *firmware* del mismo por el más reciente que se encuentre en el BGC.
network_Status_Daemon()	Proceso que se encarga de mantener la conexión wifi activa. Además, almacena la intensidad de la señal wifi.
get_WiFi_RSSI()	Retorna la última medida de intensidad de la señal wifi.
memory_Garbage_Collector_Daemon()	Recolector de memoria RAM del controlador.

Tabla A.3 Métodos en el *firmware* de los *gateway* de los FC.

A.4.1. Servicio OTA

Este servicio se encarga de procesar las peticiones OTA. En él se ejecuta un servidor HTTP que proporciona los ficheros de configuración necesarios en formato JSON para las actualizaciones, además de las actualizaciones en sí mismas. Dentro del contenedor Docker que aloja este servicio se almacenan todas las versiones del *firmware* de los controladores de los FC. Una vez iniciado el servicio, este expone un fichero denominado `manifest.json` que se genera de manera automática en función de los ficheros que se han de actualizar, crear o borrar en el dispositivo objetivo realizando una comparativa entre los archivos actuales del mismo y los alojados en el servidor OTA. Este es un ejemplo de cómo se organiza un fichero *manifest.json*:

```
{
  "delete": [
      "flash/old_file.py",
```

```
        "flash/other_old_file.py"
    ],
    "firmware": {
        "URL": "http://192.168.1.144:8000/firmware_1.0.1b.bin",
        "hash": "ccc6914a457eb4af8855ec02f6909316526bdd08"
    },
    "new": [
        {
            "URL": "http://192.168.1.144:8000/1.0.1b/
              flash/lib/new_lib.py",
            "dst_path": "flash/lib/new_lib.py",
            "hash": "1095df8213aac2983efd68dba9420c8efc9c7c4a"
        }
    ],
    "update": [
        {
            "URL": "http://192.168.1.144:8000/1.0.1b/
              flash/changed_file.py",
            "dst_path": "flash/changed_file.py",
            "hash": "1095df8213aac2983efd68dba9420c8efc9c7c4a"
        }
    ],
    "version": "1.0.1b"
}
```

Este fichero contiene los siguientes campos:

- *"delete"*: lista de archivos que ya no son necesarios.
- *"firmware"*: URL y firma SHA1 del *firmware* a instalar en el controlador.
- *"new"*: Lista de nuevos ficheros y su dirección objetivo en el controlador destino.
- *"update"*: Lista de ficheros que ya existían previamente, pero es necesario modificar en el controlador destino.
- *"version"*: Número de versión a la que se va a actualizar el *firmware* del controlador.
- *"previous_ version"*: Número de versión anterior.

A.4.2. Panel de control

Este servicio es una herramienta que permite visualizar los datos proporcionados por los componentes del BEMS en tiempo real. Este servicio dispone de una lista de direcciones URL donde encontrar los distintos servicios a los que conectarse para solicitar los datos a los componentes del BEMS. A continuación, se muestra un ejemplo de este recurso:

```
WOT_ENTRYPOINTS = {
  "things":[
      "http://192.168.1.200:80/",
      "http://192.168.1.100:10107/heart-heat-pump-monitor-thing-
        78210a89-01b9-86bd-3263-95a00399a41f"
  ],
  "hidden_things":[
      "http://192.168.1.100:10107/heart-heat-pump-monitor-thing-
        78210a89-01b9-86bd-3263-95a00399a41f"
```

Este fichero se divide en *"things"*, que define la lista de dispositivos que deberán aparecer en la representación gráfica, y *"hidden_ things"*, que define cuáles de los dispositivos deberán guardarse en la base de datos, pero no mostrarse en la representación gráfica. La Figura A.1 muestra un ejemplo de representación en el *dashboard* de datos medidos por varios dispositivos del sistema.

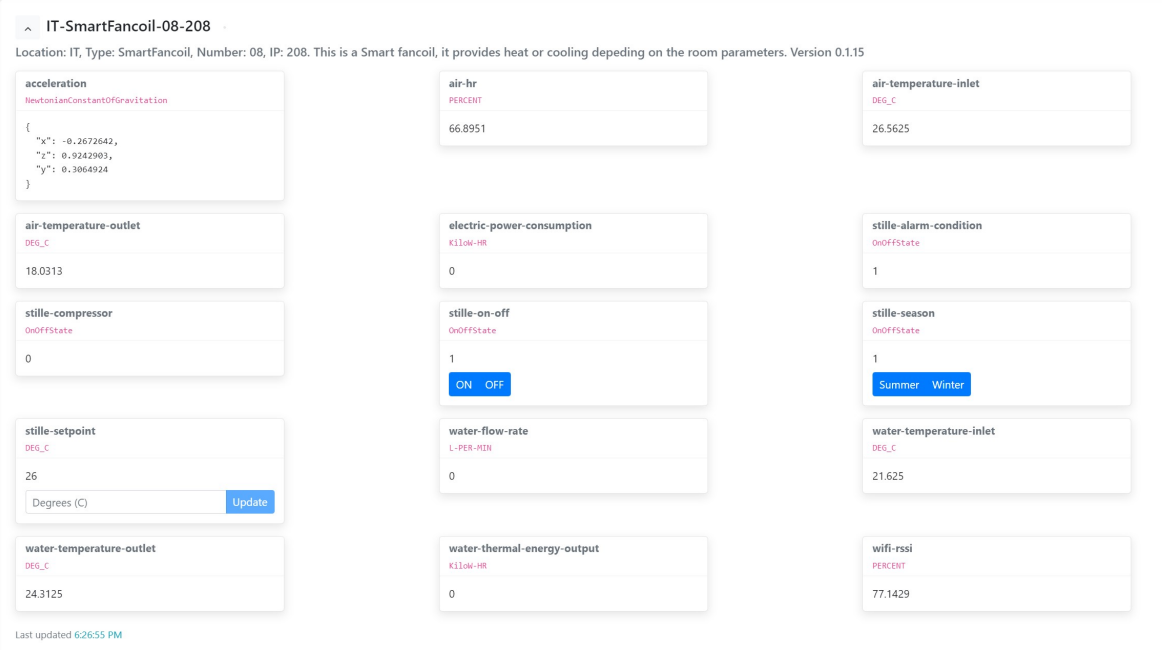


Figura A.1 Detalle del *dashboard* donde se muestra un FC de ejemplo.

Apéndice B

Librería uWoT

B.1. Introducción

uWoT es una implementación experimental parcial de las especificaciones para arquitecturas *Web of Things* generadas por el W3C. En concreto presenta las siguientes características:

- Proporciona un *W3C WoT Runtime* que contiene la funcionalidad base que permite consumir y exponer *Things* en la red.
- Ofrece una API según las especificaciones de la *W3C WoT Scripting API* que estandariza la interfaz proporcionada por los desarrolladores para utilizar el *Runtime*.
- Proporciona un conjunto de *W3C WoT Protocol Bindings* que permite traducir entre las interacciones de alto nivel en una *Thing* y los mensajes de bajo nivel que se transmiten en la red.
- Soporta Micropython[1].
- Implementa un modelo asíncrono basado en co-rutinas.

En este apéndice se detalla la implementación de las recomendaciones del W3C sobre WoT para la interoperabilidad de sistemas IoT. Esta implementación se dirige a sistemas embebidos de capacidades limitadas que no pueden ejecutar *frameworks* WoT más complejos (como por ejemplo WoTPy), que puedan ejecutar MicroPython y que posean capacidades de red. La librería se basa en las recomendaciones del W3C y se limita a las capacidades software de los microcontroladores que ejecutan MicroPython.

Como se ha mencionado previamente, el sistema objetivo para esta implementación es un microcontrolador que ejecute MicroPython y que tenga capacidades de red, ya sean inalámbricas o cableadas. Si bien, en otras implementaciones se podría seleccionar otro tipo de microcontroladores y migrar a su *firmware* esta librería. MicroPython es un sistema operativo

[1]https://micropython.org/

embebido a la par que una implementación reducida de Python 3 que proporciona una terminal, interfaces con los distintos pines hardware del microcontrolador, árbol de directorios, gestión de excepciones y control de las interfaces de red. MicroPython es una implementación de código abierto y existen múltiples librerías diseñadas por la comunidad para ampliar sus funcionalidades. En el mercado hay disponibles varios microcontroladores capaces de ejecutar MicroPython. Entre ellos destacan:

- ESP8266 (Múltiples versiones).
- ESP32 (Múltiples versiones).
- Pycom (GPy, FiPy, WiPy, etc.).
- Pyboard.

B.2. APIs asíncronas

La base de código de uWoT está construida utilizando las librerías disponibles para MicroPython. MicroPython implementa un modelo de programación mono-hilo, esto quiere decir que el desarrollador debe utilizar un modelo de programación asíncrona basado en co-rutinas que permitan exponer recursos Web, leer y escribir sensores y generar acciones y eventos de manera simultánea. Para ello, se diepone de la librería `uasyncio`[2] (implementación minimalista derivada de la librería asyncio de Python 3), que permite la generación de co-rutinas, su invocación y su gestión.

Los beneficios de las APIs asíncronas residen en el aprovechamiento de los recursos de la máquina en sistemas monohilo, como es el caso de las placas que ejecutan MicroPython. Su principal característica es el uso compartido del tiempo de ejecución, en el que una tarea reserva el hilo de ejecución durante un intervalo de tiempo y después lo libera para que otra tarea, que podría estar en espera y lista para ejecutar, lo use. Esto permite una ejecución similar a una ejecución paralela, no tanto en rendimiento como en funcionalidad, ya que permite tener varias tareas ejecutándose al unísono, lo cual es muy útil en la recepción de peticiones Web, por ejemplo.

B.3. Requisitos

Previo a la implementación se han propuesto una serie de requisitos que la librería ha de cumplir, que se recogen el la Tabla B.1. Algunos de estos requisitos pueden ser alternativos y adaptarse así a las capacidades del hardware o el software objetivo. Estos requisitos se verán reflejados más adelante en la implementación software de la herramienta uWoT.

[2]https://docs.micropython.org/en/latest/library/uasyncio.html

Descripción	Prioridad
Se pueden crear y destruir *Things*.	Alta
Se pueden crear *Things* en base a una *Thing Description* (TD) existente.	Media
Se pueden añadir y eliminar interacciones (*Property*, *Action*, *Event*) a las *Things*.	Alta
Las interacciones tipo *Property* son de lectura y pueden ser de escritura.	Alta
Se pueden añadir controladores de lectura y/o escritura a las interacciones de tipo *Property*.	Alta
Se pueden definir controladores de invocación para interacciones de tipo *Action*.	Alta
Se pueden definir controladores de eventos para las interacciones de tipo *Event*.	Alta
Se puede generar el documento JSON de la TD de una *Thing*.	Alta
Se pueden añadir anotaciones semánticas a la TD de una *Thing*.	Baja
La TD se expone en la dirección raíz de la *Thing*.	Alta
Se pueden exponer todas las interacciones a través del protocolo HTTP.	Alta
Las conexiones HTTP han de poder securizarse.	Media
Se pueden exponer todas las interacciones a través del protocolo MQTT.	Alta

Tabla B.1 Requisitos de la librería uWoT.

B.4. Implementación

Teniendo en cuenta las restricciones anteriormente mencionadas, se ha realizado la implementación de una librería en MicroPython que permite la creación de un servidor WoT en uno de los dos protocolos implementados (HTTP y MQTT). Este servidor expondrá de manera asíncrona una serie de recursos definidos por una TD autogenerada en base a los mismos. La implementación se divide en varios paquetes o *Building Blocks* (BBs) como se definen en las recomendaciones del W3C (ver Figura B.1). Para esta implementación, se han desarrollado tres BBs: *Thing Description* (TD), *Scripting API* y *Protocol Bindings*. Además, de la denominada *System API* que permite interactuar con el hardware local, sensores y actuadores.

B.4.1. MicroPython

MicroPython es una implementación del lenguaje de programación Python 3 que incluye una pequeña parte de sus funcionalidades, permitiendo su ejecución en entornos de reducidas capacidades, como los microcontroladores. MicroPython pretende ser lo más compatible

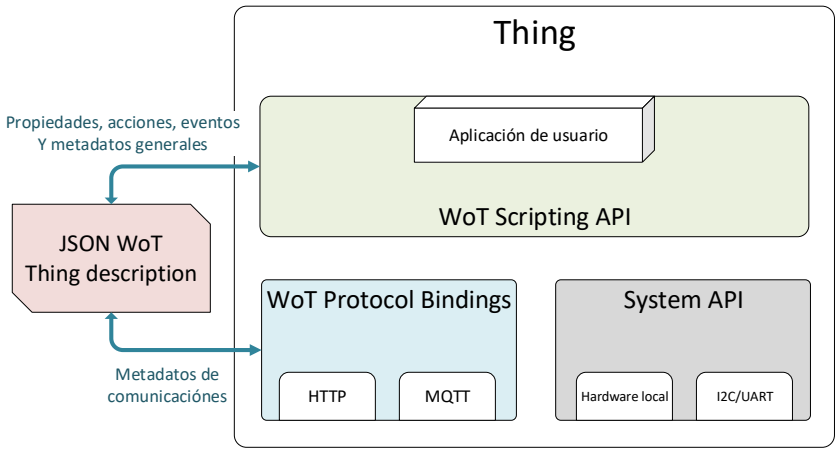


Figura B.1 Organización de la librería uWoT según el esquema de *Building Blocks* del W3C [63].

posible con Python, para simplificar la traducción de código. La librería de WoT creada soporta la versión de MicroPython 1.9 o superior. Esta librería está implementada utilizando un modelo asíncrono, basado en una implementación de la librearía `asyncio` para MicroPython denominada `uasyncio`. Esta implementación se basa en la utilización de un bucle infinito en el que se ejecutan las corrutinas. Estas corrutinas se encargan de tareas diversas, como la gestión de mensajes entrantes, llamada a los controladores de las propiedades y acceso al hardware. Como elemento a destacar, todas las funciones que sirven como controladores de los recursos expuestos han de ser asíncronas, y por tanto se definirán así:

```
async def read_accelerometer_data(self):
  await asyncio.sleep_ms(10)
  return accelerometer.acceleration()
```

Todas las funciones asíncronas deben incluir una instrucción `await` o `yield` que permita al bucle seguir ejecutándose. En caso de no necesitarla, como es el caso en el ejemplo anterior, se utilizará un `sleep` asíncrono proporcionado por la propia librería `uasyncio`.

B.4.2. Thing Description

La descripción asociada a una *Thing* es un modelo formal que representa una *Thing Web*, generalmente en formato JSON. Una TD describe los metadatos, las interfaces y los recursos que la *Thing* expone de manera inteligible tanto para una máquina como para un usuario. Las TD proveen una serie de interacciones basadas en un pequeño vocabulario que permite la integración con otras *Things* o aplicaciones de manera simple. Estas interacciones se dividen en propiedades, acciones y eventos, tal y como se describe en la Sección 2.3.2. Cada una de las interacciones definidas en estos tres tipos posee metadatos referentes a las

comunicaciones denominados *forms*, que contienen información del tipo de protocolo que se utiliza para exponer la *Thing*, el tipo de respuesta que se espera, seguridad, etc.; además de una URI que indica la dirección a consultar para obtener los datos deseados.

B.4.3. Scripting API

WoT provee una capa de interoperabilidad basada en la forma en la que las *Things* se definen, pudiendo estas ser expuestas y/o consumidas. Para exponer o consumir una *Thing*, es necesaria una TD y el software necesario para exponer sus recursos o acceder a ellos. La implementación realizada de este *Biulding Block* permite servir tanto la TD como los recursos. Para ello ofrece una serie de métodos que permiten añadir, eliminar y crear funcionalidades para cada uno de los recursos. En primer lugar, la clase WoT permite crear un objeto de la clase `ExposedThing` a partir de una TD previamente construida mediante el método `produce`. Por otro lado, la clase `ExposedThing` ofrece métodos para la creación de propiedades, acciones y eventos, así como para asignar *handlers* o controladores a cada uno de ellos. Cada recurso solo podrá tener un controlador, exceptuando las propiedades que pueden tener dos, uno para lectura y otro para escritura en caso de permitirse esta última, por lo que si se añaden varios controladores para un recurso sólo se mantendrá el último. Las restricciones de este tipo de microcontroladores sólo permiten tener un servicio expuesto, por lo que una vez se comienza a exponer los recursos no es posible iniciar otro servidor o hacer cambios en el actual de manera interna, solo se podría interaccionar mediante las URIs expuestas y el protocolo seleccionado. Respecto a la definición de los eventos, esta cambia ostensiblemente de un protocolo a otro. Por un lado, a través de MQTT es posible crear un método asíncrono que publique un evento cuando sea necesario o se cumpla cierta condición. En cambio, a través de HTTP, no es posible publicar a placer, ya que es un protocolo reactivo, pensado para recibir peticiones y contestarlas. En ese caso se utiliza el método `long polling` para esperar por una respuesta y así poder recibir el evento en caso de producirse. Todo esto implica que el `event handler` tendrá una implementación distinta para cada protocolo.

Clase Property Esta clase se utiliza para definir propiedades de la *Thing*. De manera genérica se le asocian unos valores por defecto, además de un método vacío de lectura. Los métodos que incluye son los definidos en la Tabla B.2.

Método	**Descripción**
Property.__init__(name, type, unit, description)	Crea una propiedad con los parámetros aportados, además de añadirle un método de lectura y escritura genérico.
Property.default_handler()	Método genérico de lectura de la propiedad.

Tabla B.2 Listado de métodos de la clase `Property` de la librería uWoT.

Clase Action Esta clase se utiliza para definir las acciones de una *Thing*. De manera genérica se le asocia un método que define la interacción. Los métodos que incluye se definen en la Tabla B.3.

Método	**Descripción**
Action.__init__(name, safe, idempotent, description)	Crea una propiedad con los parámetros aportados, además de añadirle un método de activación genérico.
Action.default_handler()	Método genérico de lectura de la propiedad.

Tabla B.3 Listado de métodos de la clase `Action` de la librería uWoT.

Clase Event Esta clase se utiliza para definir los eventos de una *Thing*. De manera genérica se le asocia un método que define la interacción. Los métodos que incluye se definen en la Tabla B.4.

Método	**Descripción**
Event.__init__(name, description)	Crea un evento con los parámetros aportados, además de añadirle un método de control genérico.
Event.default_handler()	Método genérico de lectura de la propiedad.

Tabla B.4 Listado de métodos de la clase `Event` de la librería uWoT.

Clase WoT Clase de entrada que permite crear la *Exposed Thing*, ya sea vacía o en base a una TD dada. En su implementación se desarrollan los métodos indocados en la Tabla B.5.

Método	**Descripción**
WoT.create_empty_thing(id, name, description, security)	Crea una *Thing* con los parámetro aportados y retorna un objeto de la clase `ExposedThing`.
WoT.produce(thingDescription)	Genera una *Thing* en base a una TD que se le pase como parámetro.

Tabla B.5 Listado de métodos de la clase `WoT` de la librería uWoT.

Clase ExposedThing Esta clase se encarga de recopilar todos los parámetros de la *Thing* a exponer, aportando funcionalidades para definir propiedades, acciones y eventos, y enlazarlos con otras funciones. También se encarga de generar la TD, recopilando todos los datos almacenados de la *Thing* y exponiéndolos en un archivo JSON. Esta clase se compone de los métodos desglosados en la Tabla B.6.

Método	**Descripción**
ExposedThing.expose(server)	Expone, mediante el protocolo definido en el parámetro de entrada, todos los recursos previamente añadidos a la *Thing*. Este método invoca a los *Protocol Bindings* para crear los distintos puntos de acceso. En caso de `MQTT`, habrá que pasar también la IP y puerto del *broker*, así como los datos de la red wifi como `**kwargs`.
ExposedThing.add_property(name, description)	Añade un recurso del tipo propiedad a la *Thing*.
ExposedThing.remove_property(name)	Elimina una propiedad de la *Thing*.
ExposedThing.add_action(name, input_type, safe, idempotent, description)	Añade un recurso del tipo acción a la *Thing*.
ExposedThing.remove_action(name)	Elimina una acción de la *Thing*.
ExposedThing.add_event(name, description)	Añade un recurso del tipo evento a la *Thing*.
ExposedThing.remove_event(name)	Elimina un evento de la *Thing*.
ExposedThing.set_property_read_handler(property_name, read_handler)	Añade un método controlador (`read_handler`) de lectura a una de las propiedades.
ExposedThing.set_property_write_handler(property_name, write_handler)	Añade un método controlador (`write_handler`) de escritura a una de las propiedades. Si se añade este controlador, la propiedad pasa a ser editable, y así se reflejará en la TD.
ExposedThing.set_action_handler(action_name, action_handler)	Añade un método controlador (`action_handler`) a una de las acciones.
ExposedThing.set_event_handler(event_name, event_handler)	Añade un método controlador (`event_handler`) a uno de los eventos.
ExposedThing.generate_thing_description()	Genera la TD de la *Thing* en su estado actual y la almacena en una variable interna de la misma.

Tabla B.6 Listado de métodos de la clase `ExposedThing` de la librería uWoT.

Clase ProtocolBinding En esta implementación, los *Protocol Bindings* permitirán enlazar los distintos recursos con sus respectivos métodos de acceso a través de un protocolo determinado. Una *Thing* solo puede tener un protocolo asociado, y todos sus recursos se expondrán a través de él. Las implementaciones son completamente asíncronas, lo que permite responder a múltiples peticiones al mismo tiempo, así como mantener procesos ejecutándose en paralelo, lo cual es muy conveniente para la monitorización de eventos. Las particularidades de cada uno se explican a continuación. En la Tabla B.7 se desglosan los métodos que componen esta clase.

Método	Descripción
ProtocolBinding.http_ bind(exposed_ thing)	Recibe una `ExposedThing` y la enlaza con un servidor HTTP
ProtocolBinding.produce(exposed_ thing, broker, ssid, ssid_ password, port, user, user_ password)	Recibe una `ExposedThing` y la enlaza con un cliente MQTT.

Tabla B.7 Listado de métodos de la clase `ProtocolBinding` de la librería uWoT.

Clase HTTP_Bind La implementación del protocolo HTTP REST se ha realizado mediante la librería `picoweb`[3], que permite la creación de un servidor Web asíncrono en MicroPython. Este se crea tras la invocación del método `expose` de la clase `ExposedThing` prporcionánmdole el parámetro `HTTP`. Internamente, se utiliza `ExposedThing` para recabar los distintos recursos a exponer, asignarles una URL, asociar los distintos *handlers* y añadirla al servidor, así como almacenarlas en la TD. Finalmente, se genera la TD por última vez con todas las URIs incluidas y se expone en el directorio raíz (`http://mything/`). Los métodos que implementa esta clase se detallan en la Tabla B.8.

Método	Descripción
HTTPBind.read_ req_ body(req)	Recibe una petición HTTP y retorna la información contenida en la misma.
HTTPBind.bind(exposed_ thing)	Recibe una `ExposedThing` y enlaza todas sus propiedades, acciones y eventos con diferentes recursos HTTP. También crea y expone la TD en el directorio raiz, por defecto, en el puerto 80 de la *Thing*.

Tabla B.8 Listado de métodos de la clase `HTTP_Bind` de la librería uWoT.

Clase MQTT_Bind La implementación del protocolo MQTT se ha realizado mediante la librería `mqtt_as`[4] que implementa el protocolo MQTT de manera asíncrona. Este se crea tras la invocación del método `expose` de la clase `ExposedThing` pasando el parámetro `MQTT`. Los tópicos o URIs MQTT asociados a cada propiedad serán el doble que en HTTP, ya que es necesario uno para recibir las peticiones (tópicos de entrada) y otro para publicar las respuestas (tópicos de salida). La `ExposedThing` se utiliza para recabar los distintos recursos a exponer, asignarles sus respectivos tópicos, asociar los distintos *handlers* y añadirlos al servidor que escuchará los tópicos de entrada, así como almacenarlas en la TD. Finalmente, se genera la TD por última vez con todas las URIs incluidas y se expone en el directorio raíz (`mqtt://mything`). En cuanto al *broker* MQTT que se utilizará, queda a elección del

[3]https://github.com/pfalcon/picoweb
[4]https://github.com/peterhinch/micropython-mqtt

usuario indicar cuál es su IP y puerto. Los métodos que implementa esta clase se detallan en la Tabla B.9.

Método	**Descripción**
MQTTBind.event_ emitter(event, topic)	Función continua que se encarga de poner en cola los eventos en los tópicos correspondientes.
MQTTBind.callback(event, topic)	Crea tareas para gestionar los mensajes entrantes, invocando a la función `message_handler`.
MQTTBind.message_ handler(topic, message)	Gestiona los mensjaes entrantes, dilucidando si corresponden a una propiedad, una acción o un evento, y actuando en consecuencia.
MQTTBind.subscribe_ all(event, mqtt_ client)	Se suscribe a todos los tópicos de entrada, por ejemplo, a los de las aacciones, esperando por los mensajes entrantes.
MQTTBind.mqtt_ loop(client)	Bucle de ejecución que se encarga de publicar los mensajes salientes en los tópicos correspondientes.
MQTTBind.connect_ and_ expose()	Inicializa las conexiones con el *broker* externo, a través del cual se realizan las comunicaciones.
MQTTBind.bind(exposed_ thing)	Recibe una `ExposedThing` y enlaza todas sus propiedades, acciones y eventos con diferentes recursos MQTT, asociando tópicos a cada uno de ellos. También crea y expone la TD en el directorio raiz, por defecto el tópico vacío.

Tabla B.9 Listado de métodos de la clase `MQTT_Bind` de la libtería uWoT.

B.5. Instalación

Dado que uWoT está desarrollado en MicroPython, el resultado es una serie de librerías necesarias para su funcionamiento junto a la propia librería uWoT. Estas librerías adicionales se encargan de los servicios de *Protocol Binding* de manera asíncrona, así como de los procesos de *logging*.

El primer paso es instalar MicroPython en una placa compatible (SoC ESP32 y derivados, como por ejemplo Pycom). Esto se hará de diferente manera dependiendo del sistema operativo del *host* desde el que se vaya a realizar el despliegue. Una vez instalado, se procederá a la subida de código. Las librerías que componen este paquete han de alojarse en la carpeta `/lib/` dentro del sistema de archivos de la placa. El software recomendado para realizar esta acción es Pymakr[5], que se integra con el editor de texto Atom o con Visual Studio Code y proporciona herramientas para subir, ejecutar y probar el código directamente sobre las placas.

[5]https://github.com/pycom/Pymakr

Apéndice C

Propiedades expuestas por los dispositivos del caso de estudio B

En este apéndice se recogen las propiedades a las que se puede acceder en diferentes dispositivos desplegados en el caso de estudio B, descrito en la Sección 4.2.

La Tablas C.1 y C.2 muestran las propiedades a las que se puede acceder en el MIMO y en la HP. En ellas, la primera columna muestra el modo de interación con la propiedad, ya sea leyéndola o escribiéndola; la segunda columna representa el nombre de la propiedad; la tercera columna indica en qué dirección Modbus se encuentra dicha propiedad; y, la cuarta columna, muestra el tipo de dato almacenado y cuántos registros ocupa. Por ejemplo, una variable tipo *UINT16* ocupa 16 bytes, lo que corresponde a un registro, mientras que una variable del tipo *UINT32* ocupa el doble, 32 bytes, lo que implica el uso de dos registros para leer la variable completa.

La Tabla C.3 muestra las propiedades a las que se puede acceder en el PLC. La primera columna representa el nombre de la propiedad; la segunda, el índice de la base de datos interna en el PLC en el que se encuentra la propiedad; y, la tercera, el *offset* dentro del índice indicado.

Tabla C.1 Propiedades expuestas por el MIMO.

Modo	**Propiedad**	**Registro/s Modbus**	**Formato**
Write	Connect_Grid	40001	UINT16
Write	Disconnect_Grid	40002	UINT16
Write	Connect_Central_Heat_Pump	40003	UINT16
Write	Connect_Local_Heat_Pump	40004	UINT16
Write	Connect_PV	40005	UINT16
Write	Connect_Battery	40006	UINT16
Continúa en la siguiente página			

Tabla C.1 – Continuación de la página anterior

Modo	Propiedad	Registro/s Modbus	Formato
Read	Battery_Power_Demand	40007	UINT16
Read	Grid_Contactor_Status	40008	UINT16
Read	PV_Contactor_Status	40009	UINT16
Read	Battery_Contactor_Status	40010	UINT16
Read	Grid_Protection_Relay_Status	40011	UINT16
Read	DC_Link_Status	40012	UINT16
Read	Grid_Voltage_L1	40013	UINT16
Read	Grid_Voltage_L2	40014	UINT16
Read	Grid_Voltage_L3	40015	UINT16
Read	Grid_Current_L1	40016	UINT16
Read	Grid_Current_L2	40017	UINT16
Read	Grid_Current_L3	40018	UINT16
Read	Grid_Power	40019	UINT16
Read	PV_Voltage	40020	UINT16
Read	PV_Current	40021	UINT16
Read	PV_Input_Power	40022	UINT16
Read	CHP_Voltage	40023	UINT16
Read	CHP_Current	40024	UINT16
Read	CHP_Output_Power	40025	UINT16
Read	Battery_Port_Voltage	40026	UINT16
Read	Battery_Port_Current	40027	UINT16
Read	Battery_Port_Power	40028	UINT16
Read	Local_Heat_Pump_Volts	40029	UINT16
Read	Local_Heat_Pump_Output_Power	40030	UINT16
Read	DC_Link_Voltage	40031	UINT16
Read	Temperature_1	40032	UINT16
Read	Temperature_2	40033	UINT16
Read	Temperature_3	40034	UINT16
Read	Temperature_4	40035	UINT16
Read	Temperature_5	40036	UINT16
Read	Temperature_6	40037	UINT16
Read	Temperature_7	40038	UINT16
Read	Temperature_8	40039	UINT16
Read	Battery_Chargue_Demand	40040	UINT16
Read	Battery_Master_Status	40041	UINT16
Read	Battery_Master_Fault_Bits	40042	UINT16

Continúa en la siguiente página

Tabla C.1 – Continuación de la página anterior

Modo	Propiedad	Registro/s Modbus	Formato
Read	Battery_Highest_Cell_Voltage	40043	UINT16
Read	Battery_Lowest_Cell_Voltage	40044	UINT16
Read	Battery_Array_Voltage	40045	UINT16
Read	Battery_State_Charge	40046	UINT16
Read	Battery_Charger_Connected	40047	UINT16
Read	Battery_Array_Current	40048	UINT16
Read	Battery_Minutes_To_Full	40049	UINT16
Read	Battery_System_Temp_High	40050	UINT16
Read	Battery_System_Temp_Low	40051	UINT16
Read	Battery_Minutes_To_Empty	40052	UINT16
Read	Battery_Nodes_Missing	40053	UINT16
Read	Grid_Connection_Status	40054	UINT16
Read	PV_Connection_Status	40055	UINT16
Read	Battery_Connection_Status	40056	UINT16
Read	CHP_Connection_Status	40057	UINT16
Read	LHP_Connection_Status	40058	UINT16
Write	Disconnect_Central_Heat_Pump	40059	UINT16
Write	Disconnect_Local_Heat_Pump	40060	UINT16
Write	Disconnect_PV	40061	UINT16
Write	Disconnect_Battery	40062	UINT16
Read	Battery_Highest_Cell_V_Node_Id	40063	UINT16
Read	Battery_Lowest_Cell_V_Node_Id	40064	UINT16
Read	Battery_Highest_Cell_T_Node_Id	40065	UINT16
Read	Battery_Lowest_Cell_T_Node_Id	40066	UINT16
Read	Battery_Discharge_Current_Limit	40067	UINT16
Read	Battery_Ok_To_Charge	40068	UINT16
Read	Battery_Ok_To_Discharge	40069	UINT16
Read	Battery_FaultStatus_MIMO	40070	UINT16
Read	Battery_Derating_Multiplier	40071	UINT16
Read	LhpEnabled	40073	UINT16
Read	ChpEnabled	40074	UINT16
Read	PvEnabled	40075	UINT16
Read	BatEnabled	40076	UINT16
Read	AcEnabled	40077	UINT16
Read	PV_Derating_Multiplier	40078	UINT16
Read	CHP_Derating_Multiplier	40079	UINT16

Continúa en la siguiente página

Tabla C.1 – Continuación de la página anterior

Modo	Propiedad	Registro/s Modbus	Formato
Read	LHP_Derating_Multiplier	40080	UINT16
Read	Lhp_Current	40081	UINT16
Read	Manual_Command_Status	40083	UINT16
Write	ExcessPvPowertoBattery	40085	UINT16
Read	ExcessPvPowertoBattery_Status	40086	UINT16
Read	StatisticsAcImportPower	40087	UINT16
Read	StatisticsAcExportPower	40088	UINT16
Read	StatisticsLhpPower	40089	UINT16
Read	StatisticsChpPower	40090	UINT16
Read	StatisticsBatteryImportPower	40091	UINT16
Read	StatisticsBatteryExportPower	40092	UINT16
Read	StatisticsPvPower	40093	UINT16
Read	StatisticsTime	40094	UINT16
Read	FaultAcL1	40095	UINT16
Read	FaultAcL2	40096	UINT16
Read	FaultAcL3	40097	UINT16
Read	FaultLhp	40098	UINT16
Read	FaultChp	40099	UINT16
Read	FaultBattery	40100	UINT16
Read	FaultPv	40101	UINT16
Read	FaultSystem	40102	UINT16
Write	FaultClearLockout	40103	UINT16
Write	FaultClearBits	40104	UINT16
Read	FaultLfAcL1	40105	UINT16
Read	FaultLfAcL2	40106	UINT16
Read	FaultLfAcL3	40107	UINT16
Read	FaultLfLhp	40108	UINT16
Read	FaultLfChp	40109	UINT16
Read	FaultLfBattery	40110	UINT16
Read	FaultLfPv	40111	UINT16
Read	FaultLfSystem	40112	UINT16
Read	FaultL1DateMonth	40113	UINT16
Read	FaultL1YearHour	40114	UINT16
Read	FaultL1MinSec	40115	UINT16
Read	FaultL2DateMonth	40116	UINT16
Read	FaultL2YearHour	40117	UINT16

Continúa en la siguiente página

Tabla C.1 – Continuación de la página anterior

Modo	Propiedad	Registro/s Modbus	Formato
Read	FaultL2MinSec	40118	UINT16
Read	FaultL3DateMonth	40119	UINT16
Read	FaultL3YearHour	40120	UINT16
Read	FaultL3MinSec	40121	UINT16
Read	FaultLhpDateMonth	40122	UINT16
Read	FaultLhpYearHour	40123	UINT16
Read	FaultLhpMinSec	40124	UINT16
Read	FaultChpDateMonth	40125	UINT16
Read	FaultChpYearHour	40126	UINT16
Read	FaultChpMinSec	40127	UINT16
Read	FaultBatteryDateMonth	40128	UINT16
Read	FaultBatteryYearHour	40129	UINT16
Read	FaultBatteryMinSec	40130	UINT16
Read	FaultPvDateMonth	40131	UINT16
Read	FaultPvYearHour	40132	UINT16
Read	FaultPvMinSec	40133	UINT16
Read	FaultSystemDateMonth	40134	UINT16
Read	FaultSystemYearHour	40135	UINT16
Read	FaultSystemMinSec	40136	UINT16

Tabla C.2 Propiedades expuestas por las HP.

Modo	Propiedad	Registro/s Modbus	Formato
read	Outdoor_temperature	10	INT16
read	DHW_temperature	11	INT16
read	Heat_outlet_temp	12	INT16
read	Heat_inlet_temp	13	INT16
read	Buffer_storage_temp	14	INT16
read	ES_inlet_temp	15	INT16
read	ES_outlet_temp	16	INT16
read	Heating_circuit_pump	23	INT16
read	Buffer_charging_pump	24	INT16
read	Compressor	25	INT16
read	Error	26	INT16
read	four_way_valve_Air	27	INT16

Continúa en la siguiente página

Tabla C.2 – Continuación de la página anterior

Modo	Propiedad	Registro/s Modbus	Formato
read	COP	30	INT16
read	Operating_hours_in_DHW_Modo	42-43	UINT32
read	Operating_hours_in_Heat_Modo	44-45	UINT32
read	Calorimeter_Heating	60-61	UINT32
read	Electric_meter_Heating	62-63	UINT32
read	Calorimeter_DHW	64-65	UINT32
read	Electric_meter_DHW	66-67	UINT32
read	Electric_meter_total	68-69	UINT32
read	Electric_meter_capacity	70-71	UINT32
read	Calorimeter_total	72-73	UINT32
read	Calorimeter_capacity	74-75	UINT32
write	Operating_Modo	100	UINT16
write	HC_Setpoint_Room_setpoint_temp	102	INT16
write	HC_Setpoint_Heat_inlet_Setpoint_temp	103	UINT16
write	HI_T_min_Cool	104	INT16
write	DHW_Normal_temp	105	INT16
write	DHW_Minimum_temp	106	INT16
write	PV_request	117	UINT16
write	Power_input_specification	125	UINT16
write	Clear_error_reset	128	UINT16
write	Outdoor_temperature_value	129	INT16
write	Outdoor_temperature_Active	130	UINT16
write	Buffer_temperature_value	131	INT16
write	Buffer_temperature_Active	132	UINT16
write	DHW_temp_value	133	INT16
write	DHW_temp_Active	134	UINT16

Tabla C.3 Propiedades expuestas por el PLC.

Propiedad	Identificador BD	Offset
energyMeterPumps_pmpST	1	0
energyMeterPumps_pmpGEN	1	4
tempProbes_t1Bot	10	0
tempProbes_t1Top	10	4
tempProbes_t2Bot	10	8

Continúa en la siguiente página

Tabla C.3 – Continuación de la página anterior

Propiedad	Identificador BD	Offset
tempProbes_t2Top	10	12
tempProbes_t3Bot	10	16
tempProbes_t3Top	10	20
tempProbes_tMixOut	10	24
clHp1Data_energyTotalizerHeating	14	12
clHp1Data_energyTotalizerTariff1	14	16
clHp1Data_energyTotalizerTariff2	14	20
clHp1Data_energyStoredSt1	14	24
clHp1Data_energyTariff1StoredSt1	14	28
clHp1Data_energyTariff2StoredSt1	14	32
clHp1Data_energyStoredSt2	14	36
clHp1Data_energyTariff1StoredSt2	14	40
clHp1Data_energyTariff2StoredSt2	14	44
clHp1Data_mfHp1	14	48
clHp1Data_tHp1F	14	52
clHp1Data_tHp1R	14	56
clHp1Data_volume	14	60
clHp1Data_volumeTariff1	14	64
clHp1Data_volumeTariff2	14	68
clHp1Data_volumeStoredSt1	14	72
clHp1Data_volumeTariff1StoredSt1	14	88
clHp1Data_volumeTariff2StoredSt1	14	92
clHp1Data_volumeStoredSt2	14	84
clHp1Data_powerConsumption	14	96
clHp2Data_energyTotalizerHeating	23	12
clHp2Data_energyTotalizerTariff1	23	16
clHp2Data_energyTotalizerTariff2	23	20
clHp2Data_energyStoredSt1	23	24
clHp2Data_energyTariff1StoredSt1	23	28
clHp2Data_energyTariff2StoredSt1	23	32
clHp2Data_energyStoredSt2	23	36
clHp2Data_energyTariff1StoredSt2	23	40
clHp2Data_energyTariff2StoredSt2	23	44
clHp2Data_mfHp2	23	48
clHp2Data_tHp2F	23	52
clHp2Data_tHp2R	23	56

Continúa en la siguiente página

Tabla C.3 – Continuación de la página anterior

Propiedad	Identificador BD	Offset
clHp2Data_volume	23	60
clHp2Data_volumeTariff1	23	64
clHp2Data_volumeTariff2	23	68
clHp2Data_volumeStoredSt1	23	72
clHp2Data_volumeTariff1StoredSt1	23	88
clHp2Data_volumeTariff2StoredSt1	23	92
clHp2Data_volumeStoredSt2	23	84
clHp2Data_powerConsumption	23	96
clStData_energyTotalizerHeating	26	12
clStData_energyTotalizerTariff1	26	16
clStData_energyTotalizerTariff2	26	20
clStData_energyStoredSt1	26	24
clStData_energyTariff1StoredSt1	26	28
clStData_energyTariff2StoredSt1	26	32
clStData_energyStoredSt2	26	36
clStData_energyTariff1StoredSt2	26	40
clStData_energyTariff2StoredSt2	26	44
clStData_mfSt	26	48
clStData_tStF	26	52
clStData_tStR	26	56
clStData_volume	26	60
clStData_volumeTariff1	26	64
clStData_volumeTariff2	26	68
clStData_volumeStoredSt1	26	72
clStData_volumeTariff1StoredSt1	26	88
clStData_volumeTariff2StoredSt1	26	92
clStData_volumeStoredSt2	26	84
clStData_powerConsumption	26	96
clGenData_energyTotalizerHeating	28	12
clGenData_energyTotalizerTariff1	28	16
clGenData_energyTotalizerTariff2	28	20
clGenData_energyStoredSt1	28	24
clGenData_energyTariff1StoredSt1	28	28
clGenData_energyTariff2StoredSt1	28	32
clGenData_energyStoredSt2	28	36
clGenData_energyTariff1StoredSt2	28	40

Continúa en la siguiente página

Tabla C.3 – Continuación de la página anterior

Propiedad	Identificador BD	Offset
clGenData_energyTariff2StoredSt2	28	44
clGenData_mfGen	28	48
clGenData_tGenF	28	52
clGenData_tGenR	28	56
clGenData_volume	28	60
clGenData_volumeTariff1	28	64
clGenData_volumeTariff2	28	68
clGenData_volumeStoredSt1	28	72
clGenData_volumeTariff1StoredSt1	28	88
clGenData_volumeTariff2StoredSt1	28	92
clGenData_volumeStoredSt2	28	84
clGenData_powerConsumption	28	96

Referencias

[1] A. Souri, A. Hussien, M. Hoseyninezhad, and M. Norouzi. A systematic review of IoT communication strategies for an efficient smart environment. *Transactions on Emerging Telecommunications Technologies*, 33(3), 2019. doi:10.1002/ett.3736.

[2] Y. Hajjaji, W. Boulila, I.R. Farah, I. Romdhani, and A. Hussain. Big data and IoT-based applications in smart environments: A systematic review. *Computer Science Review*, 39, 2021. doi:10.1016/j.cosrev.2020.100318.

[3] M.W. Ahmad, M. Mourshed, D. Mundow, M. Sisinni, and Y. Rezgui. Building energy metering and environmental monitoring - A state-of-the-art review and directions for future research. *Energy and Buildings*, 120:85–102, 2016. doi:10.1016/j.enbuild.2016.03.059.

[4] J. Liang, Y. Qiu, T. James, B.L. Ruddell, M. Dalrymple, S. Earl, and A. Castelazo. Do energy retrofits work? Evidence from commercial and residential buildings in Phoenix. *Journal of Environmental Economics and Management*, 92:726–743, November 2018. doi:10.1016/j.jeem.2017.09.001.

[5] T. Labeodan, C. De Bakker, A. Rosemann, and W. Zeiler. On the application of wireless sensors and actuators network in existing buildings for occupancy detection and occupancy-driven lighting control. *Energy and Buildings*, 127:75–83, September 2016. doi:10.1016/j.enbuild.2016.05.077.

[6] F. Pardo-Bosch, C. Cervera, and T. Ysa. Key aspects of building retrofitting: Strategizing sustainable cities. 248:109247, October 2019. doi:10.1016/j.jenvman.2019.07.018.

[7] A. Farahani, H. Wallbaum, and J. O. Dalenbäck. The importance of life-cycle based planning in maintenance and energy renovation of multifamily buildings. *Sustainable Cities and Society*, 44:715 – 725, 2019. doi:https://doi.org/10.1016/j.scs.2018.10.033.

[8] S.F. Tadeu, R.F. Alexandre, A.J.B. Tadeu, C. Henggeler, N.A.V. Simões, and P.P. Silva. A comparison between cost optimality and return on investment for energy retrofit in buildings-A real options perspective. *Sustainable Cities and Society*, 21:12 – 25, 2016. doi:https://doi.org/10.1016/j.scs.2015.11.002.

[9] G. Luddeni, M. Krarti, G. Pernigotto, and A. Gasparella. An analysis methodology for large-scale deep energy retrofits of existing building stocks: Case study of the italian office building. *Sustainable Cities and Society*, 41:296 – 311, 2018. doi:https://doi.org/10.1016/j.scs.2018.05.038.

[10] V.A. Dakwale, R.V. Ralegaonkar, and S. Mandavgane. Improving environmental performance of building through increased energy efficiency: A review. *Sustainable Cities and Society*, 1(4):211 – 218, 2011. doi:https://doi.org/10.1016/j.scs.2011.07.007.

[11] A. Kylili and P.A. Fokaides. European smart cities: The role of zero energy buildings. *Sustainable Cities and Society*, 15:86 – 95, 2015. doi:https://doi.org/10.1016/j.scs.2014.12.003.

[12] F. Pacheco-Torgal, C.G. Granqvist, B.P. Jelle, G.P. Vanoli, N. Bianco, and J. Kurnitski. *Cost-Effective Energy Efficient Building Retrofitting: Materials, Technologies, Optimization and Case Studies*. Woodhead Publishing, 2017. doi:10.1016/b978-0-08-101128-7.00001-0.

[13] P.H. Shaikh, N.B.M. Nor, P. Nallagownden, I. Elamvazuthi, and T. Ibrahim. A review on optimized control systems for building energy and comfort management of smart sustainable buildings. *Renewable and Sustainable Energy Reviews*, 34:409–429, 2014. doi:10.1016/j.rser.2014.03.027.

[14] O.H. Uribe, J.P.S. Martin, M.C. Garcia-Alegre, M. Santos, and D. Guinea. Smart building: Decision making architecture for thermal energy management. *Sensors (Switzerland)*, 15(11):27543–27568, 2015. doi:10.3390/s151127543.

[15] B. Qolomany, A. Al-Fuqaha, A. Gupta, D. Benhaddou, S. Alwajidi, J. Qadir, and A.C. Fong. Leveraging machine learning and big data for smart buildings: A comprehensive survey. *IEEE Access*, 7:90316–90356, 2019. doi:10.1109/ACCESS.2019.2926642.

[16] UN Climate Technology Centre & Network. Building Energy Management Systems (BEMS). https://www.ctc-n.org/technologies/building-energy-management-systems-bems. Accedido el 10/01/2020.

[17] Z. Yu, F. Haghighat, and B. Fung. Advances and challenges in building engineering and data mining applications for energy-efficient communities. *Sustainable Cities and Society*, 25:33 – 38, 2016. doi:https://doi.org/10.1016/j.scs.2015.12.001.

[18] B Gunay and W. Shen. Connected and distributed sensing in buildings: Improving operation and maintenance. *IEEE Systems, Man, and Cybernetics Magazine*, 3(4):27–34, October 2017. doi:10.1109/MSMC.2017.2702386.

[19] W. Tushar, N. Wijerathne, W.-T. Li, C. Yuen, H.V. Poor, T.K. Saha, and K.L. Wood. Internet of Things for green building management: Disruptive innovations through low-cost sensor technology and artificial intelligence. *IEEE Signal Processing Magazine*, 35(5):100–110, September 2018. doi:10.1109/MSP.2018.2842096.

[20] M.D. Ruiz, J. Gomez-Romero, C. Fernandez-Basso, and M.J. Martin-Bautista. Big data architecture for building energy management systems. *IEEE Transactions on Industrial Informatics*, 18(9):5738–5747, 2022. doi:10.1109/TII.2021.3130052.

[21] A. Ferrante. Energy retrofit to nearly zero and socio-oriented urban environments in the mediterranean climate. *Sustainable Cities and Society*, 13:237 – 253, 2014. doi:https://doi.org/10.1016/j.scs.2014.02.001.

[22] A. Martín-Garín, J.A. Millán-García, A. Baïri, J. Millán-Medel, and J.M. Sala-Lizarraga. Environmental monitoring system based on an Open Source Platform and the Internet of Things for a building energy retrofit. *Automation in Construction*, page 14, 2018. doi:10.1016/j.autcon.2017.12.017.

[23] Y. Cascone, M. Ferrara, L. Giovannini, and G. Serale. Ethical issues of monitoring sensor networks for energy efficiency in smart buildings: A case study. 134:337–345, October 2017. doi:10.1016/j.egypro.2017.09.540.

[24] D. Minoli, K. Sohraby, and B. Occhiogrosso. IoT considerations, requirements, and architectures for smart buildings—Energy optimization and next-generation building management systems. *IEEE Internet of Things Journal*, 4:269–283, 2017. doi:10.1109/JIOT.2017.2647881.

[25] M.A. Hannan, M. Faisal, P.J. Ker, L.H. Mun, K. Parvin, T.M.I. Mahlia, and F. Blaabjerg. A review of internet of energy based building energy management systems: Issues and recommendations. *IEEE Access*, 6:38997–39014, 2018. doi:10.1109/ACCESS.2018.2852811.

[26] M. Arnesano, G.M. Revel, and F. Seri. A tool for the optimal sensor placement to optimize temperature monitoring in large sports spaces. 68:223–234, August 2016. doi:10.1016/j.autcon.2016.05.012.

[27] D. Yoganathan, S. Kondepudi, B. Kalluri, and S. Manthapuri. Optimal sensor placement strategy for office buildings using clustering algorithms. 158:1206–1225, January 2018. doi:10.1016/j.enbuild.2017.10.074.

[28] A. García Mangas and F.J. Suárez Alonso. WoTPy: A framework for web of things applications. *Computer Communications*, 147:235–251, 2019. doi:10.1016/j.comcom.2019.09.004.

[29] HEART The Sum of All Things. https://cordis.europa.eu/project/id/768921/es. Accedido el 10/09/2019. doi:10.3030/768921.

[30] HEART The Sum of All Things. https://heartproject.eu. Accedido el 10/09/2019.

[31] C. Robson and K. McCartan. *Real World Research*. Willey, 2016.

[32] A. Oyegoke. The constructive research approach in project management research. *International Journal of Managing Projects in Business*, 4(4):573–595, 2011. doi:10.1108/17538371111164029.

[33] M. Alvesson and J. Sandberg. Generating research questions through problematization. *Academy of Management Review*, 36(2):247–271, 2011. doi:10.5465/amr.2009.0188.

[34] J. Sandberg and M. Alvesson. Ways of constructing research questions: Gap-spotting or problematization? *Organization*, 18(1):23–44, 2011. doi:10.1177/1350508410372151.

[35] V. Tsiatsis, J. Höller, C. Mulligan, S. Karnouskos, and D. Boyle. *Internet of Things: Technologies and applications for a new age of intelligence*. Elsevier, 2018. doi:10.1016/C2017-0-00369-5.

[36] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, 2017.

[37] E. Borgia. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014. doi:10.1016/j.comcom.2014.09.008.

[38] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013. doi:10.1016/j.future.2013.01.010.

[39] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of Things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012. doi:10.1016/j.adhoc.2012.02.016.

[40] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. doi:10.1016/j.comnet.2010.05.010.

[41] B. Negash, T. Westerlund, and H. Tenhunen. Towards an interoperable Internet of Things through a web of virtual things at the Fog layer. *Future Generation Computer Systems*, 91:96–107, 2019. doi:doi:10.1016/j.future.2018.07.053.

[42] Overview of the Internet of Things. Recommendation ITU-T Y.4000/Y.2060, International Telecommunication Union (ITU), June 2012. URL: https://www.itu.int/rec/T-REC-Y.2060-201206-I.

[43] D. Rotondi R. Minerva, A. Biru. Towards a definition of the Internet of Things (IoT). Technical report, Institute of Electrical and Electronic Engineers (IEEE), May 2015. URL: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

[44] V. Gazis. A survey of standards for Machine-to-Machine and the Internet of Things. *IEEE Communications Surveys and Tutorials*, 19(1):482–511, 2017. doi:10.1109/COMST.2016.2592948.

[45] K. Ponnusamy and N. Rajagopalan. Internet of Things: A survey on IoT protocol standards. *Advances in Intelligent Systems and Computing*, 564:651–663, 2018. doi:10.1007/978-981-10-6875-1_64.

[46] I. Ishaq, D. Carels, G.K. Teklemariam, J. Hoebeke, F. Van Den Abeele, E. De Poorter, I. Moerman, and P. Demeester. IETF Standardization in the field of the Internet of Things (IoT): A survey. *Journal of Sensor and Actuator Networks*, 2(2):235–287, 2013. doi:10.3390/jsan2020235.

[47] E. Lee, Y. Seo, S. Oh, and Y. Kim. A survey on standards for interoperability and security in the Internet of Things. *IEEE Communications Surveys and Tutorials*, 23(2):1020–1047, 2021. doi:10.1109/COMST.2021.3067354.

[48] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the Internet of Things. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, volume 2015, 2015. doi:10.1109/ETFA.2015.7301661.

[49] M. Garriga, C. Mateos, A. Flores, A. Cechich, and A. Zunino. RESTful service composition at a glance: A survey. *Journal of Network and Computer Applications*, 60:32–53, 2016. doi:10.1016/j.jnca.2015.11.020.

[50] OMA SpecWorks Lightweight M2M. https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/. Accedido el 08/11/2021.

[51] Develop with oneM2M. https://onem2m.org/using-onem2m/developers/basics. Accedido el 08/11/2021.

[52] L. Sciullo, L. Gigli, F. Montori, A. Trotta, and M.D. Felice. A survey on the Web of Things. *IEEE Access*, 10:47570–47596, 2022. doi:10.1109/ACCESS.2022.3171575.

[53] D. Raggett. The Web of Things: Challenges and opportunities. *Computer*, 48(5):26–32, 2015. doi:10.1109/MC.2015.149.

[54] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002. doi:10.1023/A:1016591616731.

[55] E. Wilde. Putting things to rest. *Transport*, 15:567–583, 2007.

[56] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the Web of Things. In *2010 Internet of Things, IoT 2010*, 2010. doi:10.1109/IOT.2010.5678452.

[57] D. Guinard and Trifa. *Building the Web of Things*. Manning, 2016.

[58] D. Guinard. *A Web of Things Application Architecture Integrating the Real-World into the Web*. PhD thesis, ETH Zurich, Switzerland, 2011.

[59] OGC SensorThings API Part 1 – Sensing. OGC Standard, Open Geospatial Consortium (OGC), July 2016. URL: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html.

[60] OGC SensorThings API Part 2 – Tasking Core. OGC Standard, Open Geospatial Consortium (OGC), January 2019. URL: http://docs.opengeospatial.org/is/17-079r1/17-079r1.html.

[61] Framework of the Web of Things. Recommendation ITU-T Y.4400/Y.2063, International Telecommunication Union (ITU), July 2012. URL: https://www.itu.int/rec/T-REC-Y.2060-201206-I.

[62] Submission Request to W3C: Web Thing Model. https://www.w3.org/Submission/2015/01/. Accedido el 24/06/2021.

[63] W3C Web of Things (WoT) Architecture. https://www.w3.org/TR/wot-architecture/. Accedido el 10/05/2020.

[64] W3C Web of Things (WoT) Architecture GitHub repository. https://github.com/w3c/wot-architecture. Accedido el 10/11/2019.

[65] S. Käbisch, T. Kamiya, M. McCool, V. Charpenay, and M. Kovatsch. Web of Things (WoT) Thing Description. W3C Recommendation, W3C, April 2020. https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/.

[66] M. Koster and E. Korkan. Web of Things (WoT) Binding Templates. W3C Note, W3C, January 2020. https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/.

[67] Z. Kis, D. Peintner, J. Hund, and K. Nimura. Web of Things (WoT) Scripting API. W3C Working Draft, W3C, October 2019. https://www.w3.org/TR/2019/WD-wot-scripting-api-20191028/.

[68] E. Reshetova and M. McCool. Web of Things (WoT) Security and Privacy Guidelines. W3C Note, W3C, November 2019. https://www.w3.org/TR/2019/NOTE-wot-security-20191106/.

[69] S. Kraijak and P. Tuwanut. A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends. In *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*, pages 1–6, September 2015. doi:10.1049/cp.2015.0714.

[70] W. Kassab and K.A. Darabkh. A-Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations. Elsevier Enhanced Reader. doi:10.1016/j.jnca.2020.102663.

[71] A.A. Alwan, A. Baravalle, M.A. Ciupala, and P. Falcarin. An open source software architecture for smart buildings. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Systems and Applications*, volume 869. Springer International Publishing. URL: http://link.springer.com/10.1007/978-3-030-01057-7_14, doi:10.1007/978-3-030-01057-7_14.

[72] Z. Chevallier, B. Finance, and B.C. Boulakia. A reference architecture for smart building digital twin. In *Proceedings of the International Workshop on Semantic Digital Twins Co-Located with the 17th Extended Semantic Web Conference (ESWC 2020)*, page 12, Heraklion, Greece,, June 2020. CEUR.

[73] L. Lagsaiar, I. Shahrour, A. Aljer, and A. Soulhi. Modular software architecture for local smart building servers. *Sensors*, 21(17):5810. doi:10.3390/s21175810.

[74] M. Beaudin and H. Zareipour. Home energy management systems: A review of modelling and complexity. *Renewable and Sustainable Energy Reviews*, 45:318–335, 2015. doi:10.1016/j.rser.2015.01.046.

[75] A. R. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, and M. Alikarar. A smart home energy management system using iot and big data analytics approach. *IEEE Transactions on Consumer Electronics*, 63(4):426–434, 2017. doi:10.1109/TCE.2017.015014.

[76] T. Balikhina, A. A. Maqousi, A. Albanna, and F. Shhadeh. System architecture for smart home meter. In *Proceedings of 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes and Systems, IT-DREPS 2017*, volume 2018-January, pages 1–5, 2018. doi:10.1109/IT-DREPS.2017.8277811.

[77] Y. Lin. Novel smart home system architecture facilitated with distributed and embedded flexible edge analytics in demand-side management. *International Transactions on Electrical Energy Systems*, 2019. doi:10.1002/2050-7038.12014.

[78] C. Ren and S.-J. Cao. Development and application of linear ventilation and temperature models for indoor environmental prediction and HVAC systems control. *Sustainable Cities and Society*, 51:17, 2019. doi:10.1016/j.scs.2019.101673.

[79] T. Zhang, X. Li, Q. Zhao, and Y. Rao. Control of a novel synthetical index for the local indoor air quality by the artificial neural network and genetic algorithm. *Sustainable Cities and Society*, 51:9, 2019. doi:10.1016/j.scs.2019.101714.

[80] REST - Semantic Web Standards. https://www.w3.org/2001/sw/wiki/REST. Accedido el 21/12/2021.