Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Estudio y test de las nuevas tecnologías de red para IoT orientadas a los hogares conectados: Matter

Autor: Juan Antonio Cano Vílchez

Tutor: Antonio Jesús Torralba Silgado

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Máster

Máster en Ingeniería Electrónica, Robótica
y Automática

Estudio y test de las nuevas tecnologías de red para IoT orientadas a los hogares conectados: Matter

Autor:

Juan Antonio Cano Vílchez

Tutor:

Antonio Jesús Torralba Silgado

Catedrático de Universidad

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Máster: Estudio y test de las nuevas tecnologías de red para IoT orientadas a los hogares conectados: Matter

Autor: Juan Antonio Cano Vílchez
Tutor: Antonio Jesús Torralba Silgado

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia, a mis padres, hermano y abuelos, gracias por apoyarme en todo momento, por confiar en mí y darme la oportunidad de estudiar. Gracias especialmente por animarme a continuar siempre y no abandonar mis estudios.

A cada uno de los compañeros y amigos que esta etapa académica me ha regalado, en estos últimos años hemos compartido alegrías y decepciones pero siempre juntos, gracias por formar parte de esta etapa.

A todo el profesorado que ha formado parte de mi vida académica, desde la escuela infantil hasta la etapa universitaria, cada uno de vosotros ha aportado su granito de arena en la persona que soy hoy. Gracias por vuestra dedicación, paciencia y ayuda en todo momento.

Por último, quiero agradecer en especial a la empresa Woodswallow por darme la oportunidad de formar parte de la Cátedra del IoT. Agradezco profundamente a Manuel Álvarez Ortega, cuya visión y dedicación han sido fundamentales para el desarrollo de esta empresa, y a Sebastián del Moral Gallardo, mi coordinador en la Cátedra, quien con paciencia y guía constante me ha acompañado semanalmente en este proyecto. De corazón, mil gracias por su apoyo y confianza.

Resumen

El presente Trabajo Fin de Máster realizado como investigador de la Cátedra del IoT de la Universidad de Sevilla y en colaboración con la empresa Woodswallow, tiene como objetivo el análisis del estándar Matter en el entorno de hogares conectados, partiendo de la última versión 1.3 del estándar, publicada en mayo de 2024.

El proyecto se centra en la implementación de una aplicación de simulación de consumo energético desarrollada utilizando el microcontrolador ESP32-C6-DevKitM-1 de Espressif, acompañado de una interfaz web que permite tanto la visualización como el control de datos. Todo el proyecto se ha implementado sobre contenedores Docker, que facilitan la integración del sistema y su despliegue. el desarrollo se ha abordado empleando las herramientas nativas del ecosistema Matter y se ha tomado como punto de partida el repositorio de código original del estándar para adaptarlo al desarrollo de la solución.

Abstract

The present Master's Thesis, conducted as part of the research work at the IoT Chair of the University of Seville and in collaboration with the company Woodswallow, aims to analyze the Matter standard within the connected home environment, using the latest version 1.3 of the standard, published in May 2024.

The project focuses on the implementation of an energy consumption simulation application developed using the ESP32-C6-DevKitM-1 microcontroller from Espressif, along with a web interface that enables both data visualization and control. The entire project has been deployed within Docker containers, facilitating system integration and deployment. The development approach leverages Matter's native ecosystem tools, beginning with the original source repository of the standard and adapting it to suit the solution's development needs.

Índice Abreviado

Índice

Acrónimos

6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
ACL	Access Control List
API	Application Programming Interface
AES	Advanced Encryption Standard
BLE	Bluetooth Low Energy
BTP	Bluetooth Transport Protocol
CA	Certificate Authority
CASE	Certificate Authenticated Session Establishment
CD	Certification Declaration
CSA	Connectivity Standards Alliance
CHIP	Connected Home over IP
DAC	Device Attestation Certificate
DNS-SD	Domain Name System Service Discovery
ESP-IDF	Espressif IoT Development Framework
GAP	Generic Access Profile
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronics Engineers
ID	Identificador
IP	Internet Protocol
IPv6	Internet Protocol version 6
IoT	Internet de las Cosas (Internet of Things)
LoRa	Long Range
LTS	Long Term Support
MU-MIMO	Multi-user, Multiple Input, Multiple Output
NB-IoT	Narrowband Internet of Things
NOC	Node Operational Certificate
NOCSR	Node Operational Certificate Signing Request
NOKP	Node Operational Key Pair
NVS	Non-Volatile Storage
OFDMA	Orthogonal Frequency Division Multiplexing Access
OSI	Open Systems Interconnection
PAA	Product Attestation Authority
PAI	Product Attestation Identity
PASE	Passcode Authenticated Session Establishment
PKI	Public Key Infrastructure
RAM	Random Access Memory
REPL	Read Eval Print Loop
SDK	Software Development Kit
TCP	Transmission Control Protocol

UDP	User Datagram Protocol
TFM	Trabajo de Fin de Máster
Wi-Fi	Wireless Fidelity
WSLW	Woodswallow
ZAP	ZCL Advanced Platform
ZCL	ZigBee Cluster Library
SSID	Service Set Identifier

1 Introducción

En la actualidad, el Internet de las Cosas (IoT) es probablemente una de las tecnologías disruptivas que mas ha transformando diversos aspectos de la vida cotidiana y apuesta por optimizar la gestión de recursos en entornos domésticos. Este trabajo fin de máster (TFM) tiene como objetivo investigar y experimentar con las nuevas tecnologías de red necesarias para el IoT, con un enfoque específico en los hogares conectados y el nuevo estándar Matter. En este documento, se desarrollan los fundamentos teóricos, la implementación práctica y el análisis de los resultados obtenidos a partir de una solución basada en el microcontrolador ESP32-C6-DevKitM-1, en colaboración con la empresa Woodswallow.

1.1 Motivación

Cátedra de IoT
En 2021, se creó la Cátedra de IoT [6] a raíz de un convenio entre la Universidad de Sevilla y la empresa Woodswallow. Su función principal consiste en promover actividades docentes e investigaciones en el ámbito del IoT y la gestión inteligente de la energía. Esta cátedra tiene como misión acercar a los estudiantes universitarios a la organización, cultura y realidad empresarial, además de facilitar su integración al mercado laboral.

Además de todo lo anterior, la cátedra también promueve una serie de trabajos de investigación en campos relacionados con el IoT y la gestión inteligente de la energía tales como los sistemas embebidos, la medición inteligente, los sistemas operativos en tiempo real o las redes de sensores. Esta cátedra va dirigida a todos los estudiantes de todas las ramas de la ingeniería que se ofertan en la Universidad de Sevilla, así como a docentes, profesionales del sector, grupos de investigación, empresas públicas o privadas, y asociaciones del sector. Todas estas iniciativas son la base para la creación y fortalecimiento de un entorno educativo de tipo práctico y colaborativo, permitiendo la realización de proyectos innovadores como este TFM.

Figura 1.1 Logo Cátedra de IoT de la Universidad de Sevilla.

Woodswallow
Woodswallow [18] es una empresa que cuenta con un equipo competente y experimentado de ingenieros con conocimientos en hardware, desarrollo firmware, sistemas embebidos y productos electrónicos avanzados para IoT. Su objetivo es permitir que sus clientes desarrollen sus sistemas embebidos o productos IoT de la

manera más rápida, eficiente y sencilla posible, desde las fases iniciales hasta la certificación y producción del dispositivo. Woodswallow ya ha asumido proyectos a gran escala a nivel internacional para multinacionales en sectores como Energía (Smart Metering), Movilidad Eléctrica y Telecomunicaciones.

Figura 1.2 Logo Woodswallow.

Durante la conferencia impartida por Woodswallow y la presentación de la Cátedra de IoT dentro de una de las asignaturas del máster que he cursado: Redes Inalámbricas de Sensores. Se subrayó la posibilidad de participar en la cátedra para estudiantes de esta asignatura y realizar un TFM sobre varios temas, uno de los cuales es la exploración de Matter como estándar de conectividad. Impulsado por esta oportunidad y considerándolo la mejor manera de contribuir tanto a los profesionales de Woodswallow como a la Cátedra de IoT de la universidad, opté por centrar mi proyecto final en el estándar Matter.

1.2 Objetivos

El objetivo fundamental de este TFM es el estudio y la validación experimental de nuevas tecnologías de red para IoT, específicamente en los hogares conectados, con especial atención en el estándar Matter. En cuanto a los objetivos específicos, se han propuesto los siguientes:

- Examinar el estándar Matter y su arquitectura, asimilando las bases y el estado del arte en el que se encuentra el estándar dentro del ecosistema de IoT.
- Desarrollar una aplicación en C++ para el microcontrolador ESP32-C6-DevKitM-1 que simule un nodo de consumo energético compatible con Matter.
- Comprobar y validar la funcionalidad de la aplicación mediante herramientas de software nativas del estándar Matter.
- Crear una interfaz web que permita visualizar y editar los datos de consumo energético, integrando los datos simulados en una base de datos.
- Automatizar el funcionamiento del sistema a través de herramientas como Docker que garantizan la portabilidad y escalabilidad del proyecto.
- Establecer una base para futuros desarrollos enfocados en aplicaciones de gestión inteligente de la energía en hogares.

1.3 Metodología

La metodología aplicada se ha mantenido en la línea experimental, orientado a la implementación práctica de este TFM y basada en una colaboración con la empresa Woodswallow. A continuación, se describen los pasos y las herramientas empleadas durante el desarrollo y las pruebas de Matter en el marco de los entornos IoT para hogares inteligentemente conectados.

Reuniones de Seguimiento y Orientación: Se han realizado reuniones semanales con Sebastián del Moral Gallardo, responsable del proyecto en Woodswallow, siendo estas reuniones esenciales para definir las distintas fases del proyecto, aclarar aspectos técnicos y obtener orientación durante el desarrollo.

Análisis de Especificaciones de Matter: Se ha llevado a cabo un análisis preliminar de las especificaciones del estándar Matter, revisando la documentación oficial (versión 1.3) y los *clusters* clave del proyecto.

Desarrollo e Implementación: Se ha desplegado una aplicación simulada utilizando el microcontrolador ESP32-C6-DevKitM-1 como nodo de consumo energético, utilizando las herramientas de software de Matter para la comunicación entre el dispositivo y la base de datos donde quedan almacenados los datos de consumo.

Desarrollo de la Interfaz Web: Se ha llevado a cabo el desarrollo de una interfaz web para visualizar los datos de consumo del nodo y controlar los distintos modos de consumo.

Automatización y Dockerización: Se ha automatizado el despliegue mediante la herramienta Docker, asegurando así la portabilidad y escalabilidad del sistema en entornos diferentes.

Verificación y Pruebas: Se han realizado diversas pruebas para la verificación de la funcionalidad y eficiencia del sistema utilizando registros y logs generados tanto por la aplicación como por por la interfaz web.

1.4 Estructura del documento

Este documento se se distribuye en cinco capítulos principales del siguiente modo::

- **Introducción**: Motivación, descripción de las metas a alcanzar, descripción de la metodología, colaboración con Woodswallow y estructura general del proyecto.
- **Internet de las Cosas y el Proyecto Matter**: Conceptos de IoT, retos y oportunidades en hogares conectados, tecnologías y protocolos existentes e introducción a Matter.
- **Elementos del Proyecto**: Especificaciones de Matter para el trabajo y hardware y software utilizados en el proyecto.
- **Implementación, Resultados y Análisis**: Desarrollo de la aplicación "Electrical Sensor App" para simular el nodo de consumo, desarrollo de la interfaz web, desarrollo del proceso de dockerización, y resumen de pruebas realizadas con los correspondientes resultados obtenidos.
- **Discusión, conclusiones y trabajos futuros**: Evaluación de los objetivos alcanzados, las limitaciones encontradas, las implicaciones prácticas, conclusiones finales y futuras líneas de trabajo.

2 Internet de las Cosas y el Proyecto Matter

En este capítulo se presenta una primera visión general del IoT y de cómo este concepto ha ido evolucionado en el desarrollo de los hogares conectados. También se presenta el Proyecto Matter, destacando su importancia en el ecosistema IoT a la vez que se muestra su potencial para unificar y simplificar las comunicaciones entre dispositivos inteligentes.

2.1 Conceptos generales del IoT y Hogares Conectados

El internet de las Cosas (IoT) es un sistema que enlaza una red de dispositivos físicos (dispositivos u "objetos inteligentes") a través de Internet con el propósito de intercambiar datos. Los dispositivos pueden variar, desde un simple interruptor inteligente o dispositivos portátiles como pueden ser algunos relojes inteligentes hasta sistemas industriales más complejos. El IoT se caracteriza por el hecho de que los dispositivos no solo pueden interactuar y comunicarse con las personas, sino que además lo hacen entre ellos y con otros dispositivos, como teléfonos móviles, formando parte de un ecosistema interconectado, autónomo y no dependiente de la intervención de las personas.

En sectores clave como la agricultura, la industria de fabricación, el transporte o la salud, ya se empiezan a observar los efectos del IoT. Por ejemplo, el IoT hace posible la monitorización de las condiciones meteorológicas en explotaciones agrícolas, la gestión del tráfico utilizando vehículos inteligentes, el control de los procesos de fabricación y de la gestión de inventarios en almacenes, entre otros. En el futuro, aunque los escenarios serán mucho más diversos, los dispositivos IoT se buscarán en todos los contextos empresariales con el fin de recopilar información valiosa. Esto implica un aumento paulatino del número de dispositivos globalmente conectados cada año, según se muestra en la Figura 2.1, se prevé que el IoT seguirá transformando una multitud de sectores de manera significativa y esto provocará cambios sustanciales en los hábitos de vida y laborales. [11]

En el ámbito de los hogares conectados, el IoT ha permitido desarrollar una nueva y avanzada manera de gestionar los hogares de forma más eficiente y cómoda. De esta manera, los dispositivos conectados, como pueden ser termostatos, o cámaras de vigilancia o luces o aparatos electrodomésticos pueden ser controlados a través de aplicaciones móviles o mediante la voz, lo que permite conseguir una automatización avanzada en las tareas cotidianas. Esto no solo lleva a la mejora en la comodidad de los usuarios, sino que también mejora el uso eficiente de los recursos, como la energía. Integrar estos dispositivos en una red doméstica hace que los hogares conectados no solo ofrezcan esta automatización avanzada y el control centralizado de los sistemas, sino también la posibilidad de adaptar el hogar a las preferencias del usuario, mejorando su calidad de vida.

2.2 Retos y oportunidades en los Hogares Conectados

El desarrollo del IoT en el contexto de los hogares conectados ha tenido una incidencia notable en en la manera en que los individuos interactúan con sus hogares. Sin embargo, este avance trae consigo también una serie de retos que deben ser superados a fin de obtener mayor provecho de estas tecnologías.

Retos

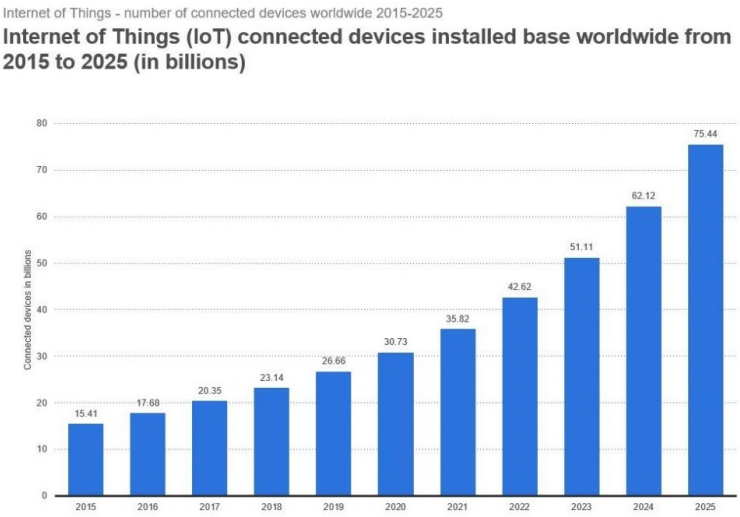


Figura 2.1 Número de dispositivos IoT anualmente [15].

- **Interoperabilidad:** Si bien el estado de avance de las tecnologías IoT ha llegado a ser razonablemente buena la falta de interoperabilidad entre los distintos dispositivos fabricados por diferentes empresas resulta ser un o de los principales retos que presentan los hogares conectados. Muchos de los dispositivos inteligentes utilizan protocolos y estándares propios, lo que dificulta su acoplamiento en una red doméstica unificada. Esto representa un obstáculo para los usuarios, que se ven obligados a utilizar también múltiples aplicaciones o plataformas de control impidiendo la creación de una experiencia única y homogénea.
- **Seguridad y privacidad:** La presencia del IoT dentro de los hogares conectados conlleva también la producción de importantes volúmenes de datos personales, incluyendo patrones de comportamiento, uso de dispositivos y preferencias de los usuarios. La protección de esta información representa un reto muy importante dado que los sistemas conectados son vulnerables a ciberataques. Cualquier brecha de seguridad podría comprometer seriamente la privacidad de los usuarios. Garantizar la confidencialidad de los datos y establecer normativas de ciberseguridad eficaces representan todavía un reto importante en este contexto.
- **Accesibilidad del usuario:** A pesar de que el IoT puede ofrecer muchas ventajas, la instalación y configuración de los dispositivos conectados puede ser compleja para algunos usuarios. La falta de conocimientos o bien la dificultad que puede representar comprender individualmente el uso de cada una de las opciones disponibles limita una adopción generalizada entre los usuarios, especialmente entre aquellos que no están familiarizados con las tecnologías digitales.
- **Dependencia de la conectividad a Internet:** Los hogares conectados dependen en gran medida de una conexión a Internet estable y de alta velocidad. No obstante, en muchas partes del mundo, la conectividad sigue siendo escasa o inestable, lo que afecta la experiencia del usuario e incluso puede frenar el uso eficiente de los dispositivos inteligentes, especialmente en áreas donde la infraestructura de red es deficiente.

Oportunidades

- **Eficiencia energética:** Los hogares conectados son una gran oportunidad para la optimización de recursos energéticos. Dispositivos como termostatos, luces y electrodomésticos inteligentes promueven una gestión eficiente del consumo, el cual se adapta a las demandas del usuario y evita el desperdicio energético. Esto no solo supone un ahorro económico para los hogares, sino que además permite disminuir el impacto ambiental y promueve modos de vida más sostenibles.
- **Bienestar y calidad de vida:** Los hogares conectados brindan oportunidades para mejorar la calidad de vida, especialmente en el ámbito de la salud y el bienestar. Dispositivos que supervisan el estado del individuo o que automatizan procesos y tareas, especialmente aquellos orientados a personas con reducidas capacidades de movilidad tiene un impacto positivo en el día a día, además de permitir la autonomía y el bienestar personal.

- **Automatización personalizada:** La automatización de tareas cotidianas es sin duda una de las oportunidades más destacadas que ofrece el IoT en los hogares conectados. Los usuarios pueden programar y controlar sus dispositivos de forma remota, facilitando la gestión del hogar y mejorando el confort del mismo. Además, los sistemas personalizados se adaptan a los hábitos y preferencias individuales, proporcionando una experiencia más intuitiva y satisfactoria.
- **Crecimiento del ecosistema IoT:** A medida que las tecnologías de los hogares conectados evolucionan, surgen nuevos modelos de negocio tanto para empresas tecnológicas como para proveedores de servicios. La demanda de dispositivos compatibles y plataformas que ofrezcan una mayor integración promueven la búsqueda de soluciones innovadoras, generando oportunidades económicas en diferentes sectores.

Las oportunidades y dificultades comentadas describen actualmente el estado de los hogares conectados, donde el IoT se establece como elemento clave en la mejora de la vida cotidiana. Pese a las propias dificultades relacionadas con la interoperabilidad, seguridad y accesibilidad, las oportunidades en términos de eficiencia energética, sensación de bienestar o la aparición de nuevos modelos de negocio son evidentes. Con el avance de la tecnología y la mejora progresiva de soluciones más integradas y seguras, se espera que estos retos se vayan resolviendo, lo que permitirá una mayor adopción y un impacto positivo en la forma en que las personas gestionan y disfrutan de sus hogares.

2.3 Protocolos y Tecnologías IoT en los Hogares Conectados

El ecosistema de los hogares conectados tiene su base en un conjunto de protocolos de comunicación y de tecnologías. Con el paso del tiempo, este campo ha estado dominado por un conjunto de protocolos, cada uno con unas características propias que los hacen útiles para sus distintas aplicaciones. En este sentido, a pesar de ser efectivos, la pluralidad de protocolos ha llevado a conformar un ecosistema en sí fragmentado, con problemas para cumplir sus objetivos de integración e interoperabilidad.

2.3.1 Principales Protocolos

Los protocolos de comunicación constituyen una parte fundamental del ecosistema de los hogares conectados, ya que permiten a los dispositivos IoT comunicarse entre sí de manera eficiente. Como se puede apreciar en la Figura 2.2, cada protocolo presenta ventajas en relación con aspectos como el consumo eléctrico, el alcance, la capacidad de transmitir datos, etc. A continuación, se presentan algunos de los protocolos más utilizados en el ecosistema del IoT, con referencias a los estándares correspondientes, si es el caso.

- **ZigBee**: Es uno de los protocolos más utilizados ya que está pensado para trabajar en una red en malla en la que los diferentes dispositivos se pueden comunicar indirectamente para ampliar la cobertura que les ofrece la red. ZigBee está basado en el estándar *IEEE 802.15.4*, descrito más a fondo en la sección de estándares. Este protocolo es comúnmente utilizado en productos de iluminación inteligente y sistemas de control climático. Se caracteriza por establecer la comunicación con un bajo consumo energético y una alta fiabilidad en entornos de múltiples nodos.
- **Z-Wave**: Junto con ZigBee, este protocolo también funciona bajo el modelo de red en malla. En este caso a diferencia de ZigBee, Z-Wave opera la red en una banda de frecuencia distinta, intentando evitar de esta forma las interferencias con redes Wi-Fi y Bluetooth. Es comúnmente empleado en aplicaciones de seguridad, como cerraduras y sensores de movimiento. Aunque no está basado en un estándar como el *IEEE 802.15.4*, Z-Wave presenta un buen grado de interoperabilidad en entornos domésticos.
- **Thread**: Es un protocolo emergente enfocado en la eficiencia energética y la escalabilidad, resultando ser ideal para dispositivos IoT que requieren una conexión permanente. Thread hace uso del estándar *6LoWPAN* que permite comunicarse mediante IPv6, facilitando una conexión directa a Internet.
- **Wi-Fi 6 (802.11ax)**: La evolución de Wi-Fi con Wi-Fi 6 ha conseguido mejorar significativamente el rendimiento en dispositivos IoT que requieren de un elevado ancho de banda, como por ejemplo las cámaras de seguridad y los sistemas multimedia. A pesar de que Wi-Fi sigue siendo el protocolo que más elevado consumo energético precisa, las mejoras en la gestión simultánea de múltiples dispositivos han optimizado su uso en hogares conectados.
- **Bluetooth Mesh**: Tradicionalmente utilizado para la conectividad entre dispositivos cercanos, se ha adaptado para operar en redes de malla, lo que permite la interconexión de múltiples dispositivos

en hogares inteligentes. Bluetooth Mesh está basado en el estándar *Bluetooth Low Energy (BLE)*, su aplicación está orientada a terminales de baja demanda de datos a través de la red, como la iluminación inteligente o pequeños sensores.

- **LoRa y NB-IoT**: Este tipo de tecnologías están orientadas a aplicaciones de largo alcance y bajo consumo de energía, y están comenzando a posicionarse en el mercado doméstico. Aunque inicialmente han sido diseñadas para entornos industriales, LoRa y NB-IoT se utilizan cada vez más en aplicaciones como sensores de agua, detectores de humo y sistemas de monitoreo exteriores. Aunque no están ligadas a los estándares mencionados en la sección de estándares, su foco en la comunicación de baja energía las hace relevantes en el contexto de entornos IoT más amplios.

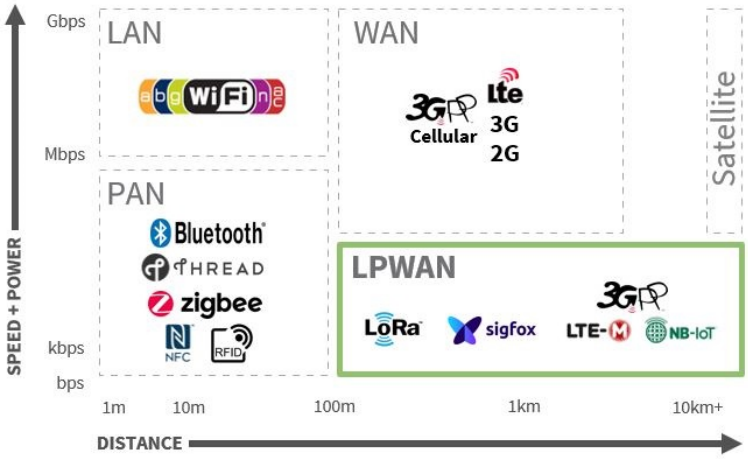


Figura 2.2 Comparativa protocolos de IoT [12].

2.3.2 Principales Estándares

Los estándares en el campo del IoT son de gran importancia para garantizar la interoperabilidad entre dispositivos y tecnologías, facilitando de este modo que el ecosistema de IoT tenga un funcionamiento fluido. En la Figura 2.3 se indica una representación de la pila de protocolos en IoT, donde se muestran las diferentes capas que intervienen en la comunicación, desde la capa física hasta la capa de aplicación, y cómo estos estándares colaboran en la conectividad entre dispositivos IoT. A continuación, se presenta un esquema de los principales estándares que sirven de base para los protocolos de la sección de protocolos.

- **IEEE 802.15.4**: Este estándar es el más común en redes de baja potencia y baja velocidad de datos, constituyendo la base de protocolos como ZigBee y Thread. Permite la creación de redes en malla en las que los dispositivos pueden transmitir datos mediante nodos intermedios, lo que mejora la cobertura y fiabilidad en entornos con numerosos dispositivos desplegados. Su eficiencia hace que sea el estándar de trabajo más idóneo para dispositivos de bajo consumo, como sensores y luces inteligentes.
- **6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks)**: Este estándar facilita la asociación de dispositivos de bajo consumo a redes IP mediante el uso de IPv6. 6LoWPAN es fundamental para que dispositivos como los basados en el protocolo Thread se conecten a redes globales sin necesidad de gateways intermedios facilitando una integración sencilla con Internet. La escalabilidad de 6LoWPAN lo convierte en el protocolo ideal para hogares conectados que requieren la gestión de un gran número de dispositivos.
- **Bluetooth Low Energy (BLE)**: BLE es un estándar fundamental para redes de corto alcance que se caracteriza por su bajo consumo energético. Esta eficiencia lo ha convertido en una tecnología esencial para dispositivos portátiles y sistemas de automatización del hogar. Aunque BLE presenta ciertas limitaciones en cuanto a su alcance y capacidad de transmisión de datos, se utiliza con frecuencia para la configuración inicial de dispositivos IoT antes de que se conecten a redes permanentes. Además, BLE es la base del protocolo Bluetooth Mesh explicado en la sección de protocolos.
- **IPv6**: Este protocolo es esencial en el ámbito del IoT, ya que ofrece un espacio de direcciones IP prácticamente ilimitado. Esto permite que cada dispositivo IoT cuente con su propia dirección única,

facilitando su identificación en redes de gran escala. IPv6 también permite la comunicación directa entre dispositivos IoT y servicios en la nube o redes externas, eliminando la necesidad de intermediarios. Esta característica mejora la eficiencia de las conexiones y reduce la latencia, lo que resulta especialmente importante para aplicaciones que requieren un control en tiempo real. Protocolos como Thread se benefician de IPv6 para simplificar la gestión y el control remoto de dispositivos.

- **Wi-Fi (IEEE 802.11)**: Aunque Wi-Fi ha sido tradicionalmente utilizado en redes de alta velocidad, sigue siendo relevante en el ámbito del IoT, especialmente para dispositivos que necesitan manejar grandes volúmenes de datos, como cámaras de seguridad o sistemas multimedia. La llegada de *Wi-Fi 6 (802.11ax)* ha mejorado su capacidad para manejar múltiples dispositivos conectados al mismo tiempo, algo crucial en hogares con un gran número de dispositivos IoT. A pesar de que su consumo energético es mayor en comparación con otros estándares, Wi-Fi continúa siendo una tecnología clave para aplicaciones que exigen altas velocidades de transmisión.

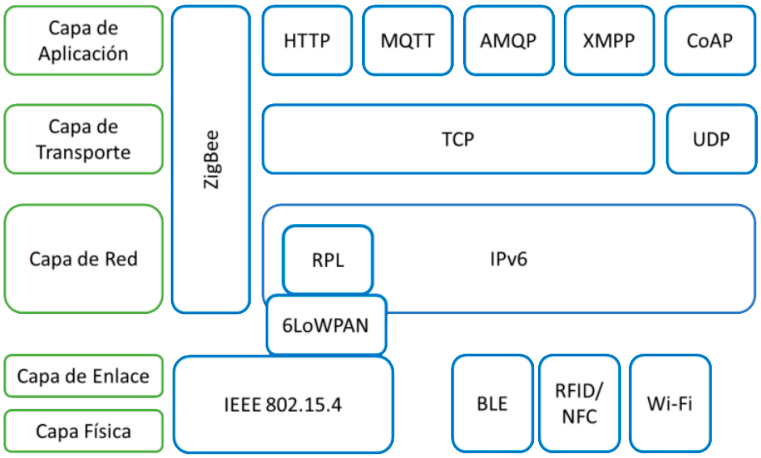


Figura 2.3 Pila de protocolos para IoT [10].

2.4 Proyecto Matter

2.4.1 Origen y fundamentos

Matter es un protocolo de comunicación y conectividad que tiene como base IP, fue creado por la Connectivity Standards Alliance (CSA) para poder impulsar la adopción de un estándar interoperable para los hogares inteligentes. La idea de la iniciativa viene de 2019 cuando antes se conocía como Project CHIP (Connected Home over IP), siendo una colaboración entre grandes compañías tecnológicas (Amazon, Google, Apple, etc.)[5], las cuales vieron la necesidad de unificar la forma en que los dispositivos IoT del hogar podían comunicarse, superando las barreras de compatibilidad entre diferentes plataformas y ecosistemas. En 2021 pasó a llamarse Matter (ver Figura 2.4), comenzando una nueva etapa de desarrollo en la que se quiere desarrollar un ecosistema IoT unificado, seguro y accesible.

Figura 2.4 Logo Matter.

Matter utiliza un kit de desarrollo software (SDK) de código abierto, que no solo implementa la especificación, sino que también proporciona ejemplos y herramientas de integración, facilitando la interoperabilidad. El protocolo se ajusta a las tres capas superiores del modelo OSI, lo que significa que puede ejecutarse sobre

cualquier red IPv6 y utilizar tecnologías de transporte (Wi-Fi, Ethernet o Thread). Respecto a la relación entre las capas de Matter, dentro del modelo OSI y TCP/IP, Matter se encuentra en las capas superiores de estas arquitecturas de red, manteniendo así la flexibilidad y compatibilidad con múltiples tecnologías. Para el emparejamiento y la asignación de dispositivos recurre a Bluetooth de bajo consumo (BLE), simplificando el proceso de incorporar nuevos dispositivos.

Una de las características más destacadas de Matter es su flexibilidad y su alta capacidad de integrar otros protocolos, incluso los ya establecidos en el mercado como Zigbee, Z-Wave y Bluetooth Mesh, mediante puentes que permiten que antiguos dispositivos convivan, en la misma red, con dispositivos que soporten Matter. De esta modo, Matter no solo promueve la interoperabilidad entre dispositivos, sino que permite un progreso gradual de evolución en los ecosistemas inteligentes, aprovechando tecnologías maduras y con grandes bases de usuarios.

En su desarrollo, Matter hace especial énfasis en la seguridad mediante el uso de criptografía avanzada, de manera que la comunicación entre dispositivos sea segura y de confianza. Matter también tiene la propiedad de multiadministrador, es decir, la capacidad de poder utilizar las mismas funciones para gestionar los dispositivos por distintos administradores, creando así un escenario donde llegan a coexistir diferentes plataformas sin conflictos.

2.4.2 Arquitectura

Tal y como se describe en la especificación [1], Matter pretende establecer este protocolo de comunicación universal basado en IPv6 para dispositivos de hogares inteligentes. El protocolo define la capa de aplicación que se implementará en los dispositivos, así como las diferentes capas de enlace, garantizando la ionteroperabilidad entre dispositivos y redes. El diagrama mostrado en la 2.5 ilustra el modo de operación normal de la pila de protocolos de Matter.

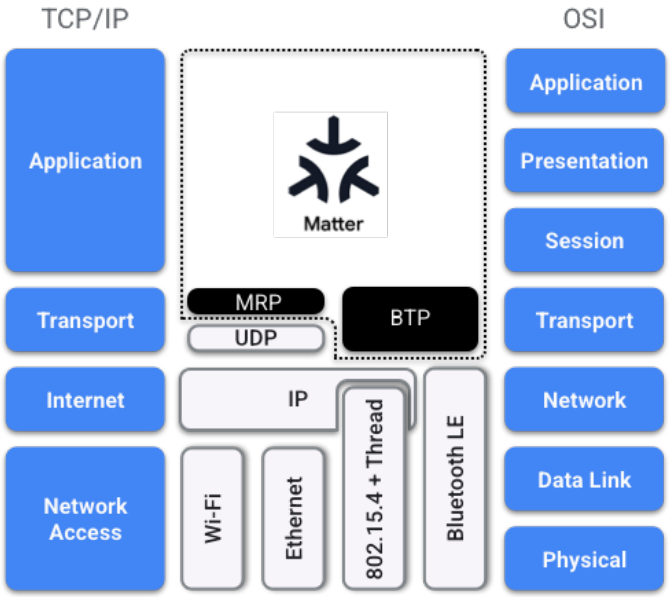


Figura 2.5 Diagrama arquitectura en capas de Matter [8].

La arquitectura Matter se divide en varias capas, permitiendo distribuir de manera clara las diferentes responsabilidades y lograr un buen nivel de encapsulado entre las distintas parte de la pila de protocolos. La interacciones que tienen lugar dentro del sistema se procesan siguiendo el orden establecido por estas capas:

- **Capa de Aplicación**: Maneja la lógica de alto nivel de los dispositivos. Por ejemplo, en una aplicación de iluminación, esta capa se encarga de gestionar la lógica del encendido/apagado de una bombilla, así como de controlar los niveles de color.

- **Capa de Modelo de Datos**: Gestiona los elementos de datos que sustentan la funcionalidad de la aplicación. La aplicación actúa sobre estas estructuras de datos cuando necesita realizar acciones sobre el dispositivo.
- **Capa de Modelo de Interacción**: Establece cuáles son las acciones que se pueden llevar a cabo entre un dispositivo cliente y un dispositivo servidor. Por ejemplo, un dispositivo cliente puede leer o escribir atributos en un dispositivo servidor. Estas acciones se llevan a cabo sobre los elementos de datos que están definidos en la capa de modelo de datos.
- **Capa de Formato de Acción**: Una vez que se ha generado una acción a partir del modelo de interacción, esta es serializada en un paquete compacto en formato binario para ser enviado a través de la red.
- **Capa de Seguridad**: En esta capa, el mensaje codificado se cifra y se le agrega un código de autenticación, garantizando la confidencialidad e integridad de los datos entre emisor y receptor.
- **Capa de Mensajes**: Una vez que la interacción ha sido serializada, cifrada y autenticada, la capa de mensajes construye la carga útil (payload) agregando los campos de encabezado necesarios y opcionales, especificando las propiedades del mensaje y la ruta lógica que debe seguir en la red.

Finalmente, una vez que la capa de mensajes ha construido la carga útil final, esta es enviada utilizando el protocolo de transporte pertinente (MRP,BTP,UDP, etc.). Cuando los datos son recibidos en el dispositivo destino, pasan nuevamente a través de la pila de protocolos. Cada capa revierte las operaciones realizadas previamente hasta que el mensaje llega a la capa de aplicación para su procesamiento final.

2.4.3 Topología de Red

En principio, cualquier red que soporte IPv6 es adecuada para la implementación de Matter, siempre y cuando cumpla con algunos estándares fundamentales de IPv6. La última versión de la especificación [1], se enfoca en tres tecnologías de capa de enlace: Ethernet, Wi-Fi y Thread. La especificación se limita a estas tecnologías para que la cobertura de estas capas sea adecuada y para mantener dentro de un límite razonable la cantidad de pruebas necesarias para la certificación.

Matter trata las redes como recursos compartidos, no establece requisitos de propiedad exclusiva o acceso exclusivo a la red. Como resultado, es posible superponer varias redes Matter sobre el mismo conjunto de redes IP constituyentes.

Este protocolo permite operar en redes que no están conectadas a Internet o que están aisladas de la infraestructura global de IPv6. Además, facilita la implementación en casos donde el proveedor de servicios de Internet no ofrece soporte para IPv6 en el hogar del usuario o donde dicho soporte es limitado. Por ejemplo, esto puede suceder si el prefijo IPv6 asignado no es suficiente para cubrir todas las redes y dispositivos en el hogar.

Matter admite principalmente dos tipos de topologías: red única y red en estrella. Estas topologías definen cómo los nodos (detallados en la sección 2.4.4) se conectan y comunican en la red, ofreciendo diferentes soluciones según la configuración y necesidades del entorno.

Topología de red única

En la topología de red única, todos los dispositivos Matter están conectados a una sola red lógica, que puede ser una red Thread/802.15.4, Wi-Fi o Ethernet. En el caso de Wi-Fi/Ethernet, la red puede abarcar varios segmentos de estas tecnologías, siempre que todos los segmentos estén interconectados a nivel de la capa de enlace. En esta topología, cada nodo se comunica con todos los demás a través de una única interfaz de red.

Topología de red en estrella

La topología de red en estrella consiste en varias redes periféricas unidas por una red central, que normalmente suele ser la red doméstica del usuario (Wi-Fi/Ethernet). Las redes periféricas pueden utilizar cualquier tipo de red compatible, pero siempre debe estar conectada directamente a la red central a través de uno o más "Border Routers" (enrutadores que conectan diferentes tipos de redes, como Wi-Fi y Thread, permitiendo la comunicación entre ellas).

Los nodos pueden tener interfaces tanto en la red central como en las periféricas, y pueden comunicarse directamente con otros nodos en la misma red. Sin embargo, si una comunicación debe cruzar de una red a otra, siempre deberá hacerlo a través de un "Border Router", tal y como se aprecia en la Figura 2.6.

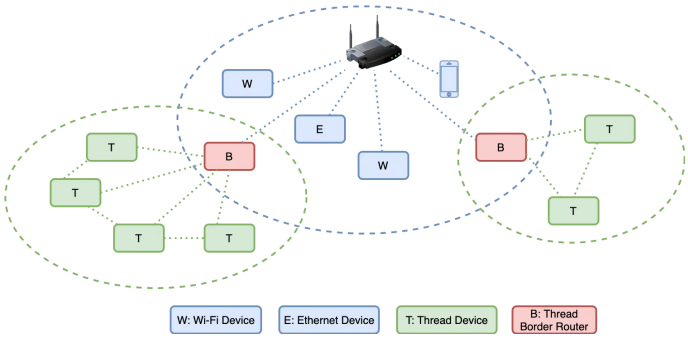


Figura 2.6 Topología red en estrella [9].

2.4.4 Dispositivos Matter

Los dispositivos en Matter tienen un modelo de datos bien definido, el cual es una representación jerárquica de las características de un dispositivo. En el nivel superior de esta jerarquía se encuentra el dispositivo.

Nodos y Endpoints

Todos los dispositivos, incluidos los teléfonos inteligentes y los asistentes domésticos, están compuestos por nodos. Un nodo es un recurso único, identificable y direccionable en una red, que el usuario percibe como una entidad funcional completa. La comunicación en la red de Matter se origina y finaliza en un nodo.

Los nodos son una colección de endpoints. Cada endpoint engloba un conjunto de características. Por ejemplo, un endpoint puede estar relacionado con la funcionalidad de iluminación, mientras que otro puede gestionar la detección de movimiento o encargarse de las actualizaciones OTA (over-the-air) del dispositivo.

En la figura 2.7, se presenta un diagrama que ilustra la estructura jerárquica de un dispositivo Matter. Cada dispositivo está compuesto por uno o varios nodos, y a su vez, cada nodo contiene uno o más endpoints, los cuales representan funcionalidades específicas del dispositivo.

Tipos de nodos

Cada nodo tiene asociado un rol que define el conjunto de comportamientos asociados a dicho nodo. Cada nodo puede desempeñar uno o varios roles, entre los cuales se incluyen:

- ***Commissioner***: Nodo responsable de realizar el proceso de *commissioning* de un dispositivo Matter a la red.
- **Controller**: Nodo que tiene la capacidad de controlar uno o más nodos. Ejemplos de esto incluyen aplicaciones de control del hogar como Apple HomeKit, Amazon Alexa o Google Assistant. Algunos dispositivos, como los interruptores de luz de encendido/apagado, también pueden cumplir con este rol.
- **Controlee**: Nodo que puede ser controlado por uno o varios nodos. La mayoría de los dispositivos pueden actuar como Controlee, salvo ciertos tipos, como los interruptores de luz de encendido/apagado, que solo pueden tener el rol de Controller.
- **OTA provider**: Un nodo que puede proporcionar actualizaciones de software OTA.
- **OTA requestor**: Un nodo que puede solicitar actualización de software OTA.

2.4.5 *clusters*

Dentro de un Endpoint, cada nodo tiene uno o más *clusters*. Estos representan otro nivel adicional en la jerarquía del dispositivo, agrupando funcionalidades específicas. Por ejemplo, un *cluster* puede encargarse de la función de encendido/apagado en un enchufe inteligente, o bien controlar el nivel de brillo en un Endpoint

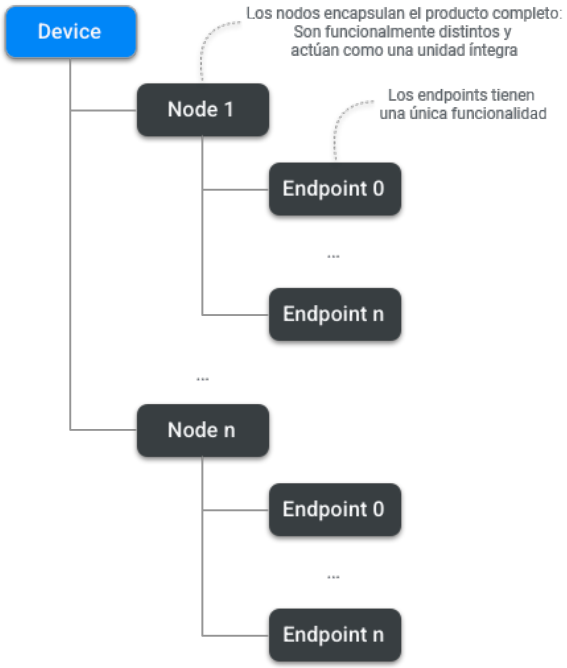


Figura 2.7 Dispositivos, nodos y endpoints [7].

de luz regulable.

Un nodo también puede tener varios Endpoints, cada uno representando una instancia de la misma funcionalidad. Por ejemplo, una lámpara puede permitir el control independiente de varias bombillas, o una regleta de enchufes puede permitir el manejo individual de cada toma.

Atributos
En el nivel más básico, se encuentran los Atributos. Estos son estados que mantiene el nodo, como el nivel actual de un *cluster* de control de nivel. Los Atributos pueden definirse empleando diferentes tipos de datos como uint8, cadenas de texto o matrices. Un ejemplo típico de un Atributo sería el estado ON/OFF de un dispositivo.

Comandos
Además de los Atributos, los *clusters* también tienen Comandos, que son acciones que pueden ser ejecutadas. Los Comandos tiene una naturaleza "verbal", como por ejemplo, "cerrar la puerta" en un *cluster* de cerradura de puerta. Estos Comandos pueden generar respuestas o resultados. En Matter, las respuestas a los Comandos también se definen como Comandos, pero son enviados en la dirección inversa.

Eventos
FInalmente, los *clusters* también pueden registrar Eventos, los cuales pueden verse como un historial de transiciones de estado. Mientras que los Atributos representan el estado actual del dispositivo, los Eventos actúan como un registro del pasado, incluyendo un contador que aumenta de forma continua, una marca de tiempo, y una prioridad. Los Eventos permiten capturar transiciones de estado y modelar datos que no pueden ser fácilmente manejados solo con Atributos.

En la figura 2.8, se muestra un ejemplo completo de la jerarquía del modelo de interacción de los dispositivos Matter.

El Endpoint 0 está reservado para los *clusters* de Utilidad. Estos son *clusters* específicos que incluyen funcionalidades necesarias en el sistema Matter, tales como descubrimiento de dispositivos, direccionamiento, diagnóstico y actualización de software. En cambio, los *clusters* de Aplicación manejan las acciones principales, como el encendido/apagado o la medición de temperatura.

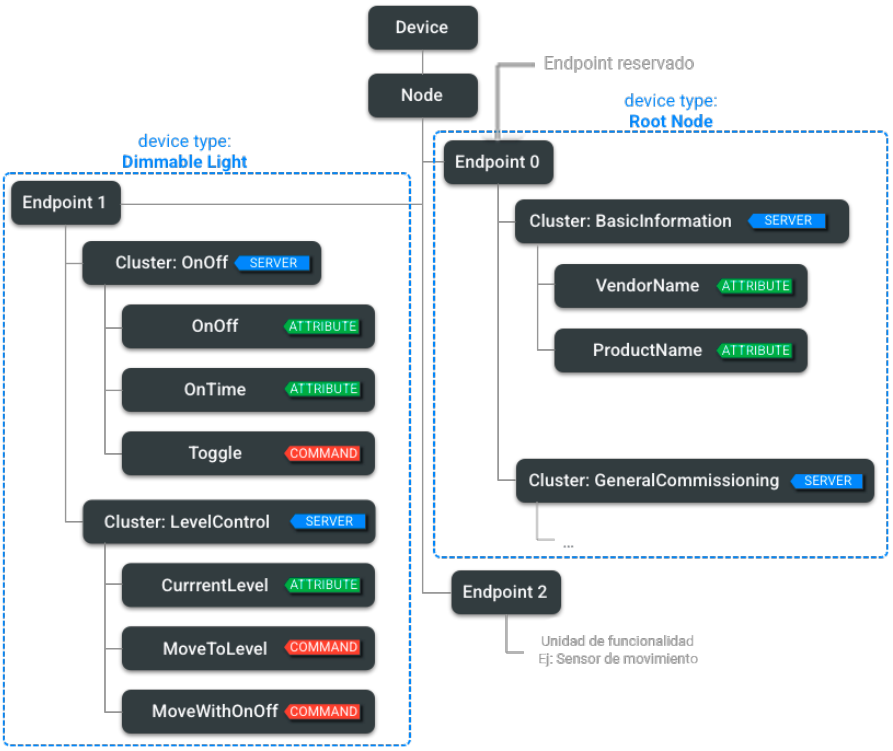


Figura 2.8 Jerarquía modelo interacción dispositivos Matter [7].

Tipos de Dispositivos

Durante el desarrollo de un nuevo dispositivo, las especificaciones de Matter exigen que el dispositivo implemente como mínimo un tipo de dispositivo. Un tipo de dispositivo se define como una colección de *clusters* obligatorios y opcionales que definen los atributos principales de un dispositivo físico, como una lámpara regulable (*Dimmable Light*), una cerradura de puerta (*Door Lock*) o un reproductor de video (*Video Player*).

Los tipos de dispositivos no se especifican en el documento principal de Matter, sino en un documento complementario llamado Device Library [3]. De manera similar, todos los *clusters* de aplicación están definidos en la Application *cluster* Library [2].

Cada Endpoint que implemente un tipo de dispositivo debe incluir los *clusters* obligatorios que lo definen. Además de estos, el Endpoint puede incluir *clusters* adicionales, que pueden ser opcionales del tipo de dispositivo o incluso *clusters* que no forman parte del tipo de dispositivo en cuestión.

2.4.6 Clientes y servidores

Los *clusters* pueden ser de tipo Cliente o Servidor. Un Server *cluster* (Servidor) almacena información y mantiene un estado, almacenando Atributos, Eventos y Comandos. Por otro lado, un *Client cluster* (Cliente) no guarda estado y su función es iniciar interacciones con un Server *cluster* remoto. Esto incluye:

- Lectura y escritura de los atributos almacenados en el servidor.
- Lectura de los eventos registrados en el servidor.
- Ejecución de comandos en el servidor remoto.

Aunque el modelo de datos en un nodo es jerárquico, la relación entre nodos no lo es. En Matter, los nodos no tienen relaciones verticales como controlador/periférico o líder/seguidor. Por el contrario, la relación es horizontal: cualquier *cluster* puede actuar como Cliente o Servidor. De este modo, un nodo puede ser tanto Cliente como Servidor en función de los diferentes *clusters* y funcionalidades.

Por ejemplo, podemos tener dos lámparas de mesa, el Nodo A y el Nodo B. Ambos nodos implementan un Tipo de Dispositivo de Encendido/Apagado de Luz (On/Off Light), que incluye un Server *cluster* de encendido/apagado, el cual controla la salida física de luz de cada lámpara, como se muestra en la Figura 2.9.

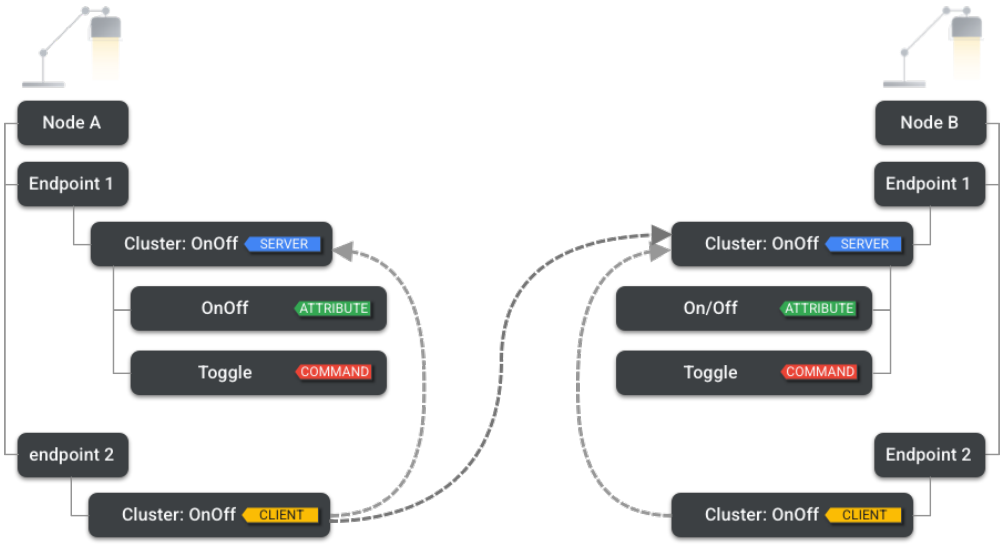


Figura 2.9 *clusters* cliente y servidor [7].

Además, como es típico en las lámparas de mesa, estos dispositivos también incluyen un Tipo de Dispositivo de Interruptor de Luz (On/Off Light Switch) para sus interruptores locales de encendido/apagado. Este Tipo de Dispositivo debe implementar un *Client cluster* de encendido/apagado para poder controlar los Server *clusters*.

En este caso, el *Client cluster* del Nodo A está cambiando los atributos del Server *cluster* de encendido/apagado del Nodo A y del Nodo B, mientras que el *Client cluster* del Nodo B solo está cambiando el Server *cluster* del Nodo B.

Conceptos del modelo de interacción

Cuando un Nodo establece una comunicación cifrada con otro Nodo, crean una relación de interacción entre ambos. Cada interacción puede estar formada por una o más transacciones, a su vez, cada transacción se compone de acciones, que son los mensajes que se envían entre los nodos. Las acciones que se llevan a cabo en las transacciones son variadas, desde una solicitud de lectura que requiere un atributo o evento de otro nodo, hasta la respuesta del propio nodo que lleva la información de vuelta desde el servidor al cliente.

El nodo que inicia la transacción se le conoce como "*initiator*", mientras que el nodo que responde se le conoce como "*target*". Normalmente el "*initiator*" se corresponde con un *cluster* tipo cliente y el "*target*" con un *cluster* tipo servidor, aunque hay excepciones, como en las interacciones de suscripción.

Grupos

En Matter, los nodos pueden formar parte de grupos. Un grupo es una forma de direccionar y enviar mensajes a varios dispositivos al mismo tiempo. Todos los Nodos de un grupo comparten el mismo identificador de grupo (Group ID) y una dirección de multicast IPv6 común. Esto permite optimizar la comunicación, especialmente cuando se necesita realizar la misma acción en varios dispositivos simultáneamente, reduciendo los tiempos de respuesta y evitando una reacción retardada visible por parte de los dispositivos.

Rutas

Para interactuar con un atributo, evento o comando de un nodo, se debe especificar una ruta. Esta ruta define la ubicación del atributo, evento o comando dentro de la jerarquía de datos del nodo. Las rutas pueden incluir operadores comodín y direcciones de grupos para dirigirse a varios nodos o *clusters* de forma simultánea,

optimizando las comunicaciones.

Transacciones Temporizadas y No Temporizadas
Existen dos tipos de transacciones: temporizadas y no temporizadas. Las transacciones temporizadas establecen un límite de tiempo para que se complete una acción, lo que previene posibles ataques de interceptación, en los cuales un atacante podría interceptar y alterar los mensajes antes de que lleguen a su destino. Estas transacciones son especialmente importantes en dispositivos que gestionan la seguridad, como cerraduras o sistemas de apertura de garaje.

Transacciones de Lectura

En una transacción de lectura, representada en la Figura 2.10, un nodo solicita uno o más atributos o eventos de otro nodo. El proceso consiste en lo siguiente:

1. **Solicitud de lectura**: El nodo iniciador envía una lista de rutas de los atributos o eventos que desea consultar del nodo objetivo.
2. **Informe de datos**: El nodo objetivo responde con los atributos o eventos solicitados.
3. **Respuesta de confirmación**: El nodo iniciador confirma la recepción de los datos o un informe sobre posibles errores mediante una acción de respuesta.

Las transacciones de lectura son exclusivamente unicast (comunicación uno a uno) y no se pueden realizar mediante multicast grupal.

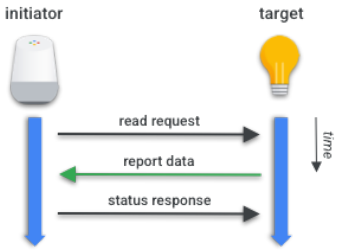


Figura 2.10 Transacción de lectura [8].

Transacción de Suscripción
Un nodo puede también suscribirse para recibir actualizaciones periódicas de un atributo o evento, tal y como se observa en la Figura 2.11. En este caso, el nodo objetivo enviará informes periódicos con la información actualizada en intervalos establecidos, mientras dure la suscripción.

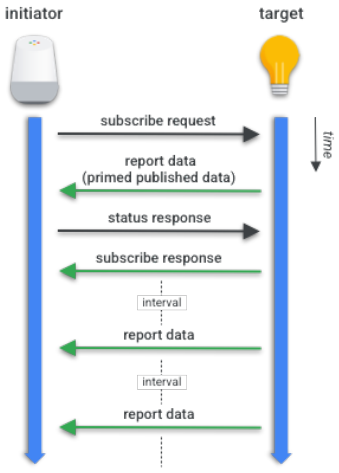
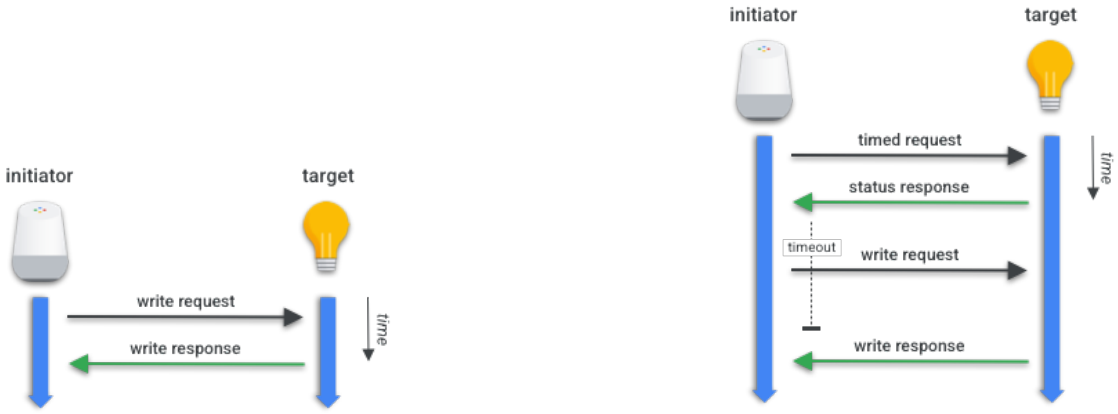


Figura 2.11 Transacción de suscripción [8].

Transacciones de Escritura

En una transacción de escritura, el iniciador solicita al nodo objetivo que modifique uno o más de sus atributos. Al igual que en las transacciones de lectura, las escrituras pueden ser temporizadas o no temporizadas.

- **Escritura no temporizada**: El Iniciador envía una solicitud de escritura con los nuevos valores, según se observa en la Figura 2.12a, y el nodo objetivo responde confirmando la operación.
- **Escritura temporizada**: En este caso, se establece un límite de tiempo para completar la operación, representado en la Figura 2.12b, lo cual añade una capa de seguridad adicional en situaciones críticas.



(a) Escritura no temporizada [8].

(b) Escritura temporizada [8].

Figura 2.12 Comparación entre transacciones de escritura no temporizada y temporizada.

Transacciones de Invocación

Las transacciones de invocación permiten que un nodo invoque uno o más comandos en un *cluster* de otro nodo de forma similar a la que se realizan las invocaciones de procedimientos remotos. El funcionamiento es el siguiente:

1. **Solicitud de invocación**: El nodo que se ha encargado de ejecutar el comando de la transacción envía la ruta de los comandos que pretende invocar y unos posibles argumentos adicionales que desee enviar.
2. **Respuesta de invocación**: El nodo sobre el que se invoca (el nodo objetivo de la invocación) responde con el resultado o estado de los comandos solicitados.

Así como las transacciones de escritura, el funcionamiento de la invocación de transacción puede ser temporizada o no temporizada y con las restricciones de seguridad adecuadas para evitar ataques.

2.4.7 Estructura de una *fabric*

En Matter, una *fabric* es el conjunto de dispositivos que comparten el mismo dominio de seguridad, lo que permite la comunicación entre dispositivos de forma segura. Este concepto incluye mecanismos avanzados de encriptación y desencriptación, junto con métodos seguros para la identificación de nodos y compartir credenciales criptográficas.

Cuando un grupo de dispositivos en una red comparten el mismo dominio de seguridad se dice que pertenecen a una misma *fabric*, todos los nodos de la *fabric* comparten el mismo certificado de autoridad (CA), conocido como la raíz de confianza (Root of Trust) y un identificador único de 64 bits denominado *fabric* ID.

El proceso de *commissioning* (enlace y configuración inicial de un nuevo dispositivo) en Matter consiste en asignar las credenciales de *fabric* a un nuevo nodo, que a partir de este momento será capaz de comunicarse con los demás nodos en la misma *fabric*.

Credenciales operacionales

El Root of Trust se establece en un nodo durante el proceso de enlace y configuración de una nuevo dispositivo, a través de un dispositivo llamado *commissioner*. Este dispositivo configurador, que generalmente es un dispositivo con una interfaz gráfica como un smartphone o hub, recibe las credenciales de una autoridad administrativa, que generalmente es un ecosistema que actúa como autoridad certificadora confiable.

El *commissioner* tiene acceso a la CA y, por lo tanto, puede solicitar las credenciales operacionales del nodo en nombre del nodo que está siendo configurado. Estas credenciales, representadas gráficamente en la Figura 2.13, se componen de dos partes principales:

- **Node Operational Identifier (ID operacional del nodo):** Un número de 64 bits que identifica de manera única a cada nodo dentro de la *fabric*.
- **Node Operational Certificate (NOC):** Un conjunto de credenciales que los nodos usan para comunicarse e identificarse dentro de una *fabric*. Se genera a través del proceso de Node Operational Certificate Signing Request (NOCSR), en el cual el nodo en *commissioning* genera ciertos elementos criptográficos y los envía al *commissioner*, quien a su vez solicita el NOC a la CA.

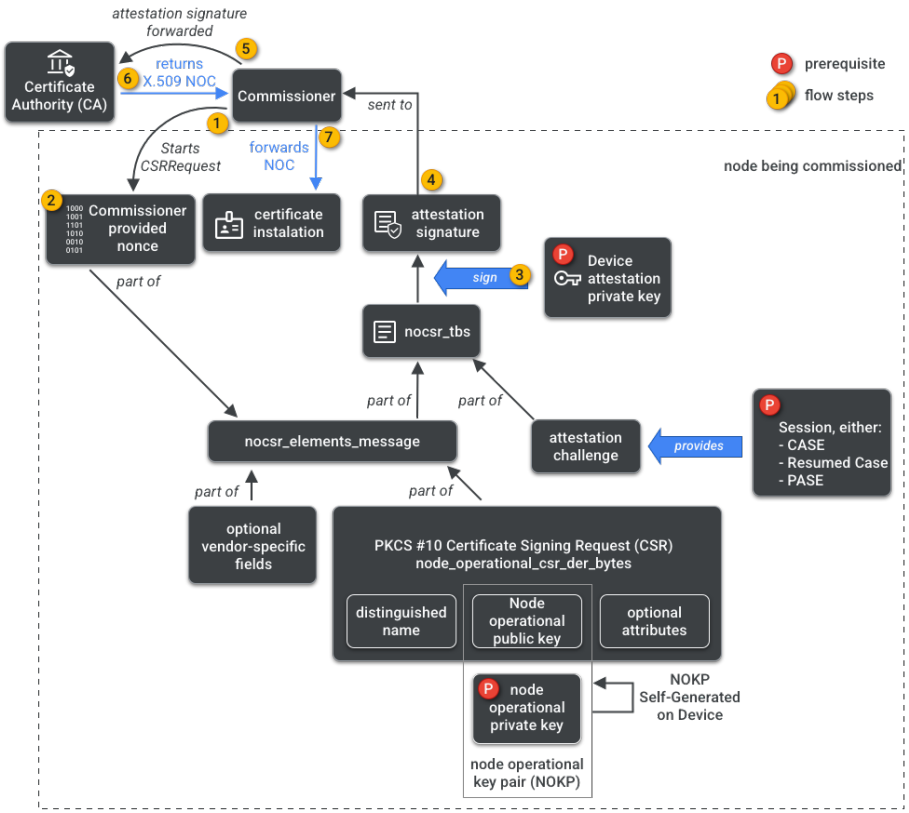


Figura 2.13 Proceso de generación del NOC [8].

El proceso de NOCSR incluye la generación de un par de claves operacionales del nodo (Node Operational Key Pair - NOKP), donde la clave pública se envía a la CA mientras que la clave privada no es compartida ni con el *commissioner* ni con la CA, garantizando así la seguridad del nodo. El proceso también se apoya en un proceso de verificación, que certifica que el dispositivo ha pasado por la certificación Matter y que es realmente lo que afirma ser, validando criptográficamente su fabricante, ID del producto y otros datos de fabricación.

Multi-Admin y múltiples *fabrics*

En Matter, los nodos pueden ser configurados de inicio para pertenecer a múltiples *fabrics*, lo que se conoce como multi-admin. Por ejemplo, un dispositivo puede estar configurado tanto en la *fabric* del fabricante como

en la de un ecosistema en la nube, gestionando de forma independiente diferentes conjuntos de comunicaciones encriptadas.

Aunque un dispositivo pueda disponer de múltiples credenciales operacionales para distintas *fabrics*, el Data Model del nodo se comparte entre todas ellas. Los atributos, eventos y acciones de los *clusters* son comunes a todas las *fabrics* a las que el nodo esté conectado. Las credenciales de red, como las de Wi-Fi o Thread, que se configuran durante el proceso de *commissioning*, forman parte del Networking Operational *cluster*, el cual se comparte entre todas las *fabrics*, y no están directamente vinculadas a las credenciales específicas de cada *fabrics*.

Este enfoque proporciona una mayor flexibilidad en la gestión de redes heterogéneas, sin comprometer la seguridad ni la interoperabilidad entre distintos dominios de control.

2.4.8 Proceso de *commissioning*

Descubrimiento de dispositivos para *commissioning*

El descubrimiento de dispositivos para *commissioning* (Commissionable Discovery) en el protocolo Matter se refiere al proceso de identificación de un nodo que aún no ha sido comisionado, es decir, que aún no ha sido integrado a una red Matter. Antes de que un dispositivo pueda ser comisionado, debe anunciarse o publicitarse de alguna manera para que el *commissioner* (dispositivo encargado de realizar la comisión) lo detecte y pueda establecer una sesión de comunicación segura. Existen tres métodos principales a través de los cuales un nodo comisionable puede anunciarse: Bluetooth low energy (BLE), Wi-Fi Soft AP y DNS-SD en una red IP.

Cada nodo comisionable, al anunciarse, envía información crucial a través de identificadores, como el *discriminator* (12 bits), el *vendor ID* (opcional), el *product ID* (opcional) y datos extendidos opcionales. El *discriminator* es esencial durante el proceso de comisión para asegurar que el dispositivo correcto sea configurado, especialmente cuando múltiples dispositivos similares están presentes. Los campos opcionales como el *vendor ID* y el *product ID* pueden proporcionar detalles adicionales sobre el fabricante y el tipo de producto.

Métodos de anuncio para dispositivos comisionables

- **BLE**: El nodo anuncia su estado no comisionado periódicamente a través de un perfil de acceso genérico (GAP). Durante los primeros 30 segundos después de encenderse, el dispositivo debe emitir anuncios con una alta frecuencia (intervalos de 20 a 60 ms), luego de los cuales la frecuencia disminuye (150 a 1500 ms). BLE no es utilizado para el descubrimiento de dispositivos ya asociados a una red.
- **Wi-Fi Soft AP**: En este modo, el nodo se descubre a través de una red de punto de acceso ad-hoc. El nombre de la red (SSID) toma la forma de "MATTER-ddd-vvvv-pppp", donde ddd es el *discriminator*, vvvv es el *vendor ID* y pppp es el *product ID*. Sin embargo, este método no es utilizado para el descubrimiento de dispositivos ya asociados a la red Matter.
- **DNS-SD**: Este método es común cuando el nodo está conectado a una red IP (como Ethernet o Wi-Fi). El nombre del servicio para el descubrimiento es *"_matterc._udp"*, y los nombres de host son creados a partir de la dirección MAC del nodo. Esta técnica permite descubrir el nodo en la red local y establecer comunicación segura con otros nodos ya comisionados.

Identificación de dspositivos activos en la red

La identificación de dispositivos activos en la red ocurre una vez que un nodo ha realizado el proceso de *commissioning* exitosamente. A través de este proceso, los nodos comisionados se descubren entre sí en la red usando el protocolo DNS-SD basado en IP. La instancia de nombre del nodo contiene el *fabric ID* comprimido y el *node ID*, lo que permite la identificación única dentro de la red. Este proceso asegura que los nodos dentro de una misma red (*fabric*) sepan qué dirección IP y puerto está usando el nodo comisionado.

Commissioning en Matter

El *commissioning* en Matter se refiere al proceso de asignar las credenciales de una *fabric* a un nuevo dispositivo para integrarlo a la red. Este proceso es realizado por el *commissioner*, mientras que el dispositivo que está siendo integrado a la red se denomina *commissionee*. En la Figura 3.1, se detallan las etapas principales del flujo de *commissioning*.

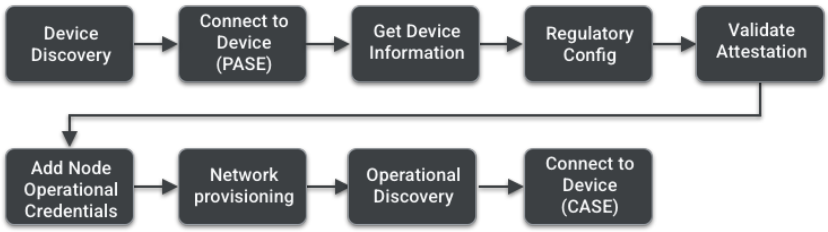


Figura 2.14 Proceso de *commissioning* [8].

1. **Descubrimiento del dispositivo:** El *commissionee* comienza a anunciarse mediante uno de los métodos de anuncio mencionados en la sección 2.4.8. Además, debe proporcionar la información de *onboarding* (*Onboarding payload*), que incluye el código de acceso (*passcode*) y el *discriminator*. Esta información puede ser introducida manualmente o, de manera más sencilla, mediante el escaneo de un código QR que contiene todos estos datos.
2. **Conexión al dispositivo (PASE):** El *commissioner* utiliza el *passcode* del *onboarding* para establecer una sesión segura llamada *Passcode Authenticated Session Establishment* (PASE), que permite que ambos dispositivos establezcan claves compartidas para la comunicación. En esta etapa también se habilita un sistema de seguridad llamado *fail-safe*, que permite restaurar el dispositivo a su estado original si el *commissioning* no se completa correctamente.
3. **Obtención de información del *Commissionee*:** El *commissioner* lee los parámetros de configuración del *commissionee*, como el *DescriptorCluster* y el *Basic Information Cluster*, que contienen información del dispositivo, incluyendo el *vendor ID*, *product ID* y número de serie.
4. **Configuración regulatoria:** El *commissioner* configura la información regulatoria en el *commissionee*, como la ubicación (interior o exterior) y el código del país.
5. **Atestación del *commissionee*:** El objetivo de este proceso es certificar que el dispositivo ha sido aprobado por Matter. El *commissioner* extrae certificados de atestación del *commissionee* (DAC) y del producto (PAI). Una vez se han recibido los certificados, el *commissioner* realiza una verificación para comprobar la autenticidad del dispositivo.
6. **Solicitud de certificado (CSR):** El *commissioner* envía una solicitud de firma de certificado al *commissionee*, que genera un par de claves operacionales. El resultado de esta solicitud se utiliza para creaar el *Node Operational Certificate* (NOC).
7. **Instalación del *Node Operational Certificate* (NOC):** El *commissioner* solicita a la autoridad certificadora que genere el NOC, lo instala en el *commissionee* y establece el certificado raíz, que sirce como punto inicial de seguridad del sistema.
8. **Configuración de la red:** El *commissioner* utiliza el método DNS-SD para encontrar el nodo recién comisionado en la red Matter.
9. Descubrimiento del nodo: El *commissioner* utiliza el DNS-SD para encontrar el nodo recién comisionado en la red operativa.
10. **Establecimiento de la sesión CASE:** Una vez que el nodo ha sido descubierto, se establece una sesión con un certificcado de autenticidad (CASE) entre el *commissioner* y el dispositivo, donde se intercambian credenciales.
11. **Finalización del *commissioning*:** El *commissioner* envía el comando *CommissioningComplete*, desactivando el *fail-safe* y permitiendo que el nodo comisionado funcione como cualquier otro nodo en la red operativa.

2.4.9 Seguridad y comunicaciones

Proceso de *attestation*

Los dispositivos certificados en Matter son aquellos que han pasado por el proceso de certificación de la Connectivity Standards Alliance (CSA), el cual garantiza su cumplimiento con el estándar Matter. Durante el proceso de *commissioning*, un dispositivo certificado debe llevar a cabo el proceso de *attestation*. Esto

implica que el dispositivo debe demostrar que es lo que afirma ser y que es un producto legítimo.

Cada dispositivo Matter tiene credenciales, las cuales incluyen un par de claves de *attestation* y una cadena de certificados asociada, que se denomina cadena de certificados de Device Attestation (DAC). El DAC es un certificado en formato X.509 v3, emitido por una autoridad certificadora (CA) asociada con el fabricante del dispositivo. Este certificado es esencial para poder demostrar que se trata de una dispositivo fabricado por un fabricante certificado y que ha pasado las diferentes pruebas de certificación Matter.

Durante el proceso de *commissioning*, el *commissioner* obtiene el DAC del dispositivo que se está comisionando (*commissionee*) y valida la autenticidad del dispositivo mediante el uso de una clave pública (PKI) basada en certificados raíz y certificados intermedios. La firma del DAC se comprueba frente al certificado intermedio de *Product Attestation* (PAI), que es emitido por una *Product Attestation Authority* (PAA). Esta jerarquía de certificados asegura que el dispositivo es auténtico y que cumple con los requisitos de certificación impuestos por la CSA, conforme a los estándares Matter.

El proceso de attestation incluye la validación de otros documentos como la *Certification Declaration* (CD), el cual permite al dispositivo mostrar su disposición frente al protocolo Matter, y la *Attestation Information*, la cual incluye los elementos de *attestation* firmados por la clave privada de *attestation* del dispositivo. Estos elementos incluyen un sello de tiempo, un número aleatorio (*nonce*) generado en el proceso y la información del firmware del dispositivo.

Comunicaciones IPv6

Matter utiliza IPv6 como protocolo de comunicación para la operación de dispositivos dentro de la *fabric*. Se aprovechan tanto las direcciones unicast de IPv6 para la comunicación entre nodos, como las direcciones multicast para el control simultáneo de grupos de dispositivos.

Los dispositivos Matter pueden operar en redes de alta capacidad, como Wi-Fi o Ethernet, pero también en redes de bajo consumo de energía, como Thread. Thread está diseñado para soportar redes de baja latencia y bajo ancho de banda, utilizando una topología de malla para dispositivos de baja potencia, como sensores. Sin embargo, Thread tiene un ancho de banda limitado de 250 kbps a nivel de la capa física IEEE 802.15.4, con una tasa efectiva de transferencia de datos de alrededor de 125 kbps. Por lo tanto, no es adecuado para la transmisión de datos de gran volumen.

Los *Border Routers* son elementos clave para permitir la coexistencia de nodos en ambos medios de transporte (Wi-Fi y Thread) dentro de la misma *fabric*. Estos enrutadores actúan como conectores entre la red Thread y la infraestructura local de Wi-Fi o Ethernet. Además, los *Border Routers* asignan prefijos de IPv6 diferentes a cada red, gestionan el descubrimiento de servicios en la red (DNS-SD) en nombre de los nodos Thread, y reenvian el tráfico relevante de un medio a otro sin sobrecargar la red Thread con paquetes de Wi-Fi.

En cuanto a las comunicaciones multicast, Matter implementa un esquema de direccionamiento basado en prefijos unicast de IPv6, como se define en el RFC 3306. Este esquema permite dirigir los mensajes multicast a nodos específicos en función de su prefijo unicast compartido dentro de una *fabric*.Matter utiliza el puerto 5540 para las comunicaciones multicast, lo que permite a los dispositivos enviar comandos simultáneamente a varios nodos dentro de la misma red.

2.5 Comparativa Matter con otros protocolos

Tras analizar por encima las ultimas tecnologías del IoT, en especial y en detalle heoms analizado la de Matter, estudiantdo como funciona el protocolo, podemos pasar a realizar una comparativa de estas tecnologias, comparando características calve de Matter frente al resto de protocolos mencionados en el apartado 2.3.1, se ha tomado como base el análisis realizado por Christopher Loreck en su artículo sobre el impacto de Matter en los ecosistemas de hogares conectados [14].

- **Interoperabilidad:** Uno de los mayores retos en los hogares inteligentes es la fragmentación del ecosistema, donde distintos dispositivos de diversos fabricantes no pueden comunicarse fácilmente

entre sí. Los protocolos como ZigBee y Z-Wave han intentado resolver este problema a través de redes en malla, pero siguen estando limitados por sus propios acuerdos y ecosistemas cerrados. Matter, en cambio, está diseñado para ser compatible con una amplia gama de dispositivos de diferentes fabricantes, incluidos gigantes tecnológicos como Apple, Google y Amazon. Matter permite la comunicación nativa entre dispositivos que utilicen tecnologías como Wi-Fi, Ethernet y Thread, superando las limitaciones de compatibilidad observadas en otros protocolos.

- **Seguridad:** La seguridad en los dispositivos IoT es uno de los elementos más impmortantes, sobre todo debido a los riesgos de privacidad. ZigBee y Z-Wave disponen de altos niveles de seguridad en sus redes, ya que el proceso de comunicación utiliza cifrado AES para protegerlas. Por su parte, Matter va un paso más allá, ya que permite que todos los dispositivos conectados dispongan de cifrado extremo a extremo y autenticación, lo cual refuerza considerablemente la protección frente a ciberataques y asegura que los dispositivos puedan operar de forma independiente a los servidores remotos lo que significa que los dispositivos Matter funcionarán también aunque el fabricante original deja de prestar soporte.
- **Consumo:** El consumo energético es un factor decisivo en la adopción de dispositivos IoT en entornos domésticos. Los protocolos ZigBee y Thread ya han sido optimizados para dispositivos de bajo consumo, como sensores y luces inteligentes, al usar redes de baja energía basadas en el estándar IEEE 802.15.4. Matter, que también puede funcionar sobre Thread, mantiene esta eficiencia energética al mismo nivel, lo que lo hace ideal para dispositivos alimentados por baterías. Por otro lado, Mater es capaz de operar en su mayoría en las redes locales y con ello evitar la dependencia del procesamiento en la nube. Esto se traduce en que el gasto energético en las operaciones se reduce significativamente.
- **Escalabilidad:** La escalabilidadjuega un papel muy importante dentro de lo que son los hogares inteligentes a medida. Z-Wave o ZigBee han demostrado que se pueden escalar rápidamente gracias a sus arquitecturas en malla, pero Matter se diferencia por su arquitectura propia basada en IP que permite que se escale en redes de mayor tamaño sin perder rendimiento. Matter no solo permite la integración de nuevos dispositivos mediante un el escaneo de códigos QR, sino que también se asegura de que pueda integrarlos de forma simple l fucnionamiento de las redes existentes, elminando la necesiada de configuraciones complejas.
- **Costes:** En lo que respecta a costes, Matter tiene una consedirable ventaja frente a protocolos como Z-Wave que necesitan hardware especializado. Al funcionar sobre tecnologías estándar como Wi-Fi y Ethernet, Matter permite una reducción de costes al aprovechar las infraestructuras existentes en los hogares. Además, tiene un enfoque de código abierto para promover la innovación y permite que los desarrolladores creen soluciones más asequibles, algo que puede hacer más probable la adopción en el mercado

La entrada de Matter en el mercado IoT puede cambiar radicalmente el paradigma de los hogares conectados, ofreciendo una solución universal a los problemas de interoperabilidad y de seguridad que han limitado la expansión de otros protocolos más tradicionales como ZigBee y Z-Wave. Si bien estos últimos han tenido éxito en nichos de mercado específicos, la apertura, flexibilidad y efectividad de Matter lo convierten en el estándar de futuro en el ámbito de IoT para los hogares inteligentes. Aunque quedan algunos obstáculos todavía por superar, como la ya mencionada adopción masiva por parte de los fabricantes y consumidores.

3 Elementos del Proyecto

El presente capítulo profundiza en todos los elementos que son parte del proyecto en el marco del estándar Matter que se han utilizado en el proceso de desarrollo. La motivación principal del mismo ha sido el diseño y desarrollo de una placa de evaluación (*evalboard*) para simular un nodo de consumo energético en hogares conectados, utilizando el microcontrolador ESP32-C6-DevKitM-1 y diversas herramientas de software adaptadas al entorno del estándar Matter.

En primer lugar, se expondrán las características generales del proyecto, con la intención de establecer el contexto necesario para hacer comprensible el alcance del mismo. Posteriormente, se describirán los elementos de hardware utilizados, haciendo énfasis en las características de la placa ESP32-C6-DevKitM-1 y otros dispositivos periféricos que han permitido la simulación y las pruebas del nodo. A continuación, se discutirá el entorno de desarrollo software utilizado, prestando atención a las herramientas y las plataformas que han permitido hacer una adecuada integración con el ecosistema Matter. Finalmente, realizará un análisis a fondo de las principales especificaciones del estándar Matter para la implementación del nodo de consumo energético.

3.1 Estrategia y fundamentos del proyecto

Desde las primeras reuniones que se celebraron con los responsables de la Cátedra de IoT en Woodswallow, se concretaron las directrices generales del proyecto, acordando que el objetivo final sería una solución de conectividad para el IoT del hogar basada en el estándar Matter. En un primer momento se consideró incluir un estudio del entorno Thread, dada la importancia que tiene el mismo para las redes de malla del IoT, aunque finalmente se decidió que no se integraría en este proyecto concreto, dado que requería una complejidad técnica considerable que implicaba muchos más recursos adicionales para su integración en la fase actual. En su momento, desde la empresa se propuso realizar un nodo de consumo energético orientado hacia el entorno doméstico, lo que para Woodswallow suponía un gran interés dado su enfoque en soluciones que implementan la medición inteligente y gestión energética en el entorno de hogares conectados.

Con este enfoque se realizó un análisis exhaustivo de las opciones disponibles que existían en el mercado, con el fin de determinar cuál sería el microcontrolador más adecuado. Dado que el estándar Matter es promovido por la CSA, se tuvieron en cuenta primero los fabricantes que mantienen una colaboración activa con dicha alianza, ya que sus soluciones suelen estar mejor adaptadas a los últimos avances en la implementados en el estándar. Durante esta fase de investigación se analizaron fundamentalmente las soluciones de las empresas Nordic Semiconductor y Espressif Systems, que implementan plataformas de desarrollo ampliamente reconocidas en el ámbito del IoT y que cuentan con soporte activo para Matter.

La elección del microcontrolador se llevo a cabo teniendo en cuenta una seria de criterios técnicos, entre ellos la compatibilidad con la versión 1.3 de Matter, la cual incorporaba cambios importantes en cuanto a la interoperabilidad y al soporte de *clusters* relevantes en el proyecto. También se estudió la facilidad de integración de los repositorios de software y de las herramientas de desarrollo asociadas a Matter. Finalmente y tras llevar a cabo un estudio comparativo de las alternativas existentes se optó por la utilización de la placa de Espressif Systems, la ESP32-C6-DevKitM-1. Otro de los aspectos importantes a la hora de decidirse por

esta plataforma era el poder servir de entorno de desarrollo basado en Ubuntu. Espressif posee repositorios especializados en Matter y herramientas de desarrollo adaptadas a las versiones más recientes del estándar, lo que permitía poder implementar de forma efectiva las funcionalidades necesarias para la simulación del nodo de consumo energético.

Este enfoque permitió sentar las bases necesarias para poder implementar de forma eficiente y acorde a las necesidades de Woodswallow, el uso de un entorno de desarrollo y de pruebas adecuado para poder evaluar y validar el nodo de consumo en contexto de hogares conectados.

3.2 Hardware Utilizado

El desarrollo hardware de este proyecto se ha basado principalmente en el uso del microcontrolador ESP32-C6-DevKitM-1 de Espressif Systems, el cual ha desempeñado la labor principal en la implementación del nodo de consumo energético compatible con el estándar Matter. Este tipo de microcontrolador fue escogido por los siguientes motivos: su compatibilidad con las versiones más actuales del protocolo Matter, su gran aceptación en proyectos de investigación sobre el mismo estándar, y su capacidad de operar bajo el estándar de conectividad Wi-Fi 6, una tecnología que actualmente esta siendo introducida en la mayoría de entornos domésticos. Esta compatibilidad ha permitido desarrollar los ensayos en un entorno controlado con un router compatible con Wi-Fi 6, garantizando la interoperabilidad y el rendimiento del sistema en escenarios reales.

Figura 3.1 Logo Espressif Systems.

En lo relativo a la implementación y la comprobación del sistema, se utilizó un ordenador portátil como entorno de desarrollo, implementando el uso de una máquina virtual con el sistema operativo Ubuntu 24.02, cuyas características proporcionan soporte de forma nativa a las herramientas de desarrollo de Espressif y a los repositorios de Matter, facilitando el despliegue y las pruebas del nodo de consumo. Se puede conocer la información correspondiente a las herramientas y configuraciones utilizadas en este entorno en la sección 3.3.

No obstante, debido a problemas de conectividad derivados del módulo bluetooth integrado en el portátil y las limitaciones que implica el hecho de ejecutar el entorno en una máquina virtual (como se puede ver en la sección **??**), terminó siendo necesaria la utilización de un dongle USB externo de la marca Ugreen compatible con BLE con la finalidad de garantizar una comunicación estable con el microcontrolador durante el proceso de *commissioning*.

3.2.1 Microcontrolador ESP32-C6-DevKitM-1

La placa de desarrollo de nivel básico ESP32-C6-DevKitM-1 (ver Figura 3.2) basada en el módulo ESP32-C6-MINI-1(U), es capaz de ofrecer soporte completo para las tecnologías Wi-Fi 6, Bluetooth LE, Zigbee, y Thread, lo que la convierte en una solución ideal para la implementación de nodos de consumo energético en entornos de hogares inteligentes que requieren alta compatibilidad. A continuación, se indican los aspectos técnicos más importantes acerca de esta placa, tomando como referencia el *datasheet* para los módulos ESP32-C6-MINI-1(U) [16].

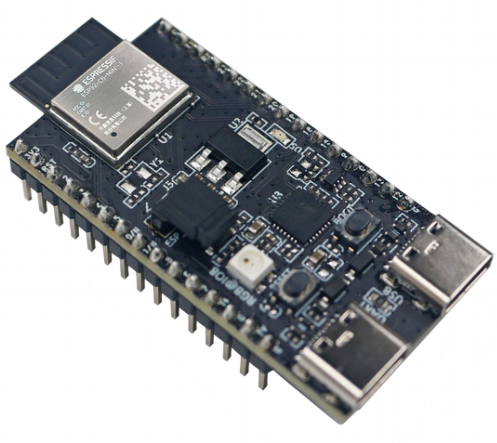


Figura 3.2 ESP32-C6-DevKitM-1.

Arquitectura y memoria
El módulo que compone esta placa ESP32-C6-MINI-1(U), se basa en un procesador RISC-V de 32 bits que puede operar a una frecuencia de hasta 160 MHz, ofreciendo el rendimiento necesario para llevar a cabo aplicaciones de IoT. Este módulo presenta también 4 MB de memoria flash SPI, la cual se integra directamente en el dispositivo, junto con 512 KB de SRAM y 320 KB de ROM, los cuales ofrecen el espacio suficiente espacio necesario para ejecutar aplicaciones complejas y asegurar la conectividad simultáneamente a redes Wi-Fi y Bluetooth.

Conectividad
Uno de los aspectos más destacados de esta placa es su soporte para Wi-Fi 6 (802.11ax) en la banda de 2.4 GHz. Esta tecnología ofrece características avanzadas como OFDMA y MU-MIMO, que mejoran la eficiencia de la transmisión y reducen la latencia en redes con múltiples dispositivos conectados. Además, la placa es compatible con Bluetooth 5 (BLE) y el protocolo IEEE 802.15.4, lo que facilita la integración con redes Zigbee y Thread. Estas capacidades hacen que la ESP32-C6-DevKitM-1 sea idónea para proyectos que exigen alta interoperabilidad, como es el caso de los dispositivos compatibles con el estándar Matter.

Periféricos y entradas
La placa cuenta con la mayoría de los pines GPIO accesibles a través de cabeceras en ambos lados, lo que facilita la conexión de periféricos mediante cables de puente o montando la placa en una protoboard. Además, incluye un regulador de voltaje que convierte la alimentación de 5 V a 3.3 V, un puente USB-UART que permite la programación y depuración, y un puerto USB Tipo-C que sirve para la alimentación, la programación, y la comunicación directa con el chip mediante protocolos USB.

El ESP32-C6-DevKitM-1 ofrece una plataforma versátil y potente para el desarrollo de aplicaciones IoT, especialmente aquellas que requieren compatibilidad con Wi-Fi 6, Bluetooth 5, Zigbee y Thread. Su bajo consumo energético, combinado con características de conectividad avanzadas, la hacen ideal para proyectos IoT relacionados con Matter.

3.2.2 Dongle USB Ugreen

El Dongle USB Bluetooth 5.0 de Ugreen, ver Figura 3.3, ha sido utilizado en este proyecto para garantizar una conectividad Bluetooth estable, necesaria para la comunicación con dispositivos periféricos en las pruebas del nodo de consumo energético basado en Matter. Este adaptador proporciona soporte para Bluetooth 5.0, lo que permite una mayor estabilidad en la transmisión de datos y un rango extendido de hasta 20 metros.

Además, el dongle cuenta con un chipset RTL8761BUV, que optimiza el consumo de energía, reduciendo el calor generado y prolongando la vida útil del dispositivo. Esta eficiencia energética, junto con su facilidad de uso *Plug and Play*, ha permitido resolver los problemas de conectividad Bluetooth presentes en el entorno de desarrollo sin necesidad de configuraciones complejas, garantizando una comunicación fluida en las pruebas realizadas.

Figura 3.3 Dongle USB Ugreen.

3.3 Software y herramientas empleadas

A continuación se exponen las herramientas de software y los repositorios que han sido utilizados en el desarrollo del proyecto. Desde la puesta en marcha de una máquina virtual para el entorno de trabajo, hasta el uso de herramientas concretas de Matter.

3.3.1 Entorno Ubuntu 24.02 en máquina virtual

Dado que la mayoría de las herramientas de desarrollo actuales para el desarrollo de aplicaciones de IoT están optimizadas para la plataforma Linux, se decidió optar por este sistema operativo desde el inicio del proyecto. En concreto, se optó por la versión Ubuntu 24.02, la cual era la versión más reciente al inicio del desarrollo.

Ubuntu es una distribución de Linux basada en Debian y ampliamente utilizada en entornos de desarrollo por su estabilidad y facilidad de uso. La versión 24.02, lanzada en 2024, correspondiente a una versión LTS (Long Term Support), implica que Ubuntu 24.02 ofrece soporte a largo plazo, garantizando actualizaciones de seguridad y estabilidad durante cinco años. Con respecto a versiones anteriores, Ubuntu 24.02 presenta mejoras como un kernel Linux más reciente que soporta mejor el hardware moderno, mejoras en ZFS (sistema de archivos) y en el rendimiento de Snap (el sistema de empaquetado), a la vez que se incluyen actualizaciones en la interfaz gráfica GNOME que optimizan la experiencia de usuario cuando se trabaja en entornos de desarrollo que involucran la creación de aplicaciones como en este proyecto.

Con la intención de evitar los inconvenientes que suponen la modificación de las particiones del disco duro del ordenador principal, que utiliza Windows 10, se decidió ejecutar Ubuntu en una máquina virtual, para lo cual se utilizó la herramienta VirtualBox.

VirtualBox es un software de virtualización de código abierto desarrollado por Oracle que permite ejecutar otros sistemas operativos adicionales dentro de un sistema operativo principal sin necesidad de modificar las configuraciones de hardware del equipo. En este caso, VirtualBox ha permitido la instalación de Ubuntu 24.02 como sistema adicional sobre Windows 10, permitiendo que ambos sistemas operativos se ejecuten simultáneamente. Esto es especialmente útil para evitar cualquier tipo de problema relacionado con la configuración de particiones. Además, VirtualBox ofrece la posibilidad de compartir carpetas entre el sistema anfitrión y el sistema adicional, y configurar la cantidad de recursos (CPU, RAM, almacenamiento) asignados a la máquina virtual lo que permite crear un entorno de desarrollo óptimo sin que se vea afectado significativamente el rendimiento del sistema anfitrión.

Este entorno de trabajo me ha permitido aprovechar todas las ventajas que ofrece Linux para la parte del desarrollo de las aplicaciones IoT, manteniendo también la flexibilidad y la compatibilidad con el sistema operativo principal del equipo (Windows 10).

3.3.2 Repositorio Connectedhomeip

El repositorio connectedhomeip, se encuentra disponible en GitHub [4], es el repositorio oficial del proyecto Matter y desde sus inicios ha estado gestionado por la CSA con el nombre de *Project CHIP*. Este repositorio actúa como espacio centralizado donde los desarrolladores pueden tener acceso al código fuente de Matter,

colaborar mejorándolo e implementarlo en sus proyectos personales. De hecho GitHub, la plataforma donde está alojado este repositorio, es ampliamente utilizada para almacenar repositorios de código.

El motivo de la existencia de este repositorio es que el proyecto Matter está pensado para ser de código abierto, transparente y disponible para el uso público general, incluyendo a colaboradores que no pertenezcan de forma directa a la CSA. El motivo de este enfoque según los principales desarrolladores del repositorio, es llevar los beneficios de Matter tanto a los consumidores como a los fabricantes lo más rápido posible.

Estructura del repositorio

El contenido del repositorio es bastante amplio y está organizado en diferentes carpetas y archivos que facilitan la localización de los recursos requeridos en función de las necesidades concretas del proyecto. Los componentes principales se estructuran de la siguiente manera:

- **/build:** Archivos y configuraciones relacionados con el sistema de construcción y los directorios de salida.
- **/config:** Archivos de configuración para diferentes plataformas y entornos de aplicación.
- **/docs:** Documentación referente al proyecto Matter, incluyendo guías y especificaciones teóricas.
- **/examples:** Proporciona ejemplos prácticos de aplicaciones basadas en Matter, como controladores de iluminación, sensores, y otros dispositivos IoT.
- **/integrations:** Código y configuraciones para la integración de Matter con plataformas de terceros.
- **/src:** Implementación del protocolo Matter, incluyendo el modelo de datos, el modelo de interacción, el enrutamiento de mensajes y las capas de seguridad.
- **/third_party:** Código de terceros utilizado por Matter, como por ejemplo bibliotecas auxiliares o dependencias externas.

Al margen de todo este conjunto de archivos que ofrece el repositorio, este también proporciona herramientas y utilidades que facilitan el proceso de desarrollo, configuración y depuración de aplicaciones Matter. En relación al contexto de este proyecto, las herramientas más destacadas y que han sido utilizadas son *Chip Tool*, *ZAP Tool* y *Chip-REPL*.

Chip Tool

Chip Tool es herramienta de control de Matter que permite realizar el proceso de *commissionig* de un dispostivo Matter dentro de una red y comunicarse con él a través de mensajes Matter. Esta herramienta se ejecuta desde una terminal de comandos y permite interactuar directamente con dispositivos Matter, configurar de parámetros, realizar pruebas de conectividad o ejecutar comandos para evaluar el comportamiento del dispositivo.

Estructura y funcionalidades

- ***Commissioning***: *Chip Tool* permite la incorporación de dispositivos Matter en redes Thread o Wi-Fi, empleando BLE como canal de comisionamiento inicial para dispositivos no enlazados con la red. Durante este proceso, además de especificar las credenciales de red necesarias, se establece una conexión segura con el dispositivo para integrarlo en la red Matter.
- **Interacción con *clusters*:** La herramienta trabaja utilizando el modelo de datos de Matter, donde organiza las funcionalidades del dispositivo en *clusters* (como encender/apagar o ajustar brillo). Los comandos de *Chip Tool* permiten ejecutar acciones y leer atributos en los *clusters*, para comprobar y validar el comportamiento de los dispositivos.
- ***Binding*:** *Chip Tool* soporta la funcionalidad de *binding*, permitiendo enlazar dispositivos entre sí directamente en la red, por ejemplo, enlazar un interruptor a una bombilla sin necesidad del controlador central.

- **Control de acceso:** La herramienta permite gestionar las *Access Control List* (ACL) que regulan las interacciones permitidas entre dispositivos. Así se garantiza que solamente los dispositivos autorizados sean capaces de leer o modificar atributos concretos.
- **Descubrimiento de dispositivos:** *Chip Tool* permite realizar el descubrimiento de dispositivos Matter que están disponibles para ser comisionados o que ya funcionan en la red, facilitando la identificación y la configuración inicial.
- **Pruebas y validación:** Permite ejecutar pruebas automáticas y específicas para asegurar que los dispositivos comisionados cumplan con los estándares de Matter, incluyendo la validación de interacciones entre dispositivos y la captura de logs de depuración.

Ejecucción y configuración
Para utilizar *Chip Tool*, primero es necesario compilarlo desde el código fuente. El repositorio *connectedhomeip* proporciona los paquetes y scripts para realizar este proceso. La compilación se ejecuta a través de un comando desde el directorio local donde se ha clonado el repositorio:

Código 3.1 Compilación de *Chip Tool*.

```
./scripts/examples/gn_build_example.sh examples/chip-tool BUILD_PATH
```

En este caso, `BUILD_PATH` hace referencia al directorio donde se generarán los binarios, que tras compilarse permitirán ejecutar el binario resultante para iniciar la herramienta y obtener los comandos disponibles:

Código 3.2 Ejecucción de *Chip Tool*.

```
$ ./chip-tool
```

La herramienta ofrece una gran variedad de comandos generados para llevar a cabo las funciones descritas previamente, cada funcionalidad tiene asociado un conjunto de comandos específicos que requieren una serie de parámetros para poder ejecutarse correctamente.

ZAP Tool

El *ZAP Tool* (*ZCL Advanced Platform*) es una herramienta gráfica de usuario (GUI), ver figura 3.4, utilizada para la generación de un archivo `.zap`, que describe la composición de los endpoints de un dispositivo en el ecosistema Matter. Este archivo `.zap` contiene información sobre los endpoints, *clusters* y tipos de dispositivos, así como las características específicas de los *clusters*, como atributos, comandos y eventos que estarán disponibles en el dispositivo. Además del archivo `.zap`, el ZAP Tool también puede generar archivos `.matter`, que son una versión legible por humanos del archivo `.zap`, permitiendo una revisión más sencilla de la configuración del dispositivo antes de ser compilado.

Proceso de generación de la Capa Ember
El archivo `.zap` se utiliza junto con los archivos de definición de *clusters* en el proceso de compilación para generar una capa Ember, que luego se compila en el firmware del dispositivo. La capa Ember es esencial para la implementación de las funcionalidades definidas por Matter en el dispositivo. El proceso de creación de esta capa se realiza de manera automática como parte de la compilación del proyecto.

Configuración y personalización
El archivo zap se encarga de albergar todos los atributos disponibles en un *cluster*, permitiendo configurar varios parámetros:

- **Enabled:** Activa o desactiva el atributo en el dispositivo.
- **Attribute ID:** Identificador único del atributo según las especificaciones Matter.
- **Required:** Indica si el atributo es obligatorio. Algunos atributos se vuelven obligatorios cuando se habilitan ciertas características o atributos adicionales.
- **Client/Server:** Para Matter, todos los atributos están en el lado del servidor.

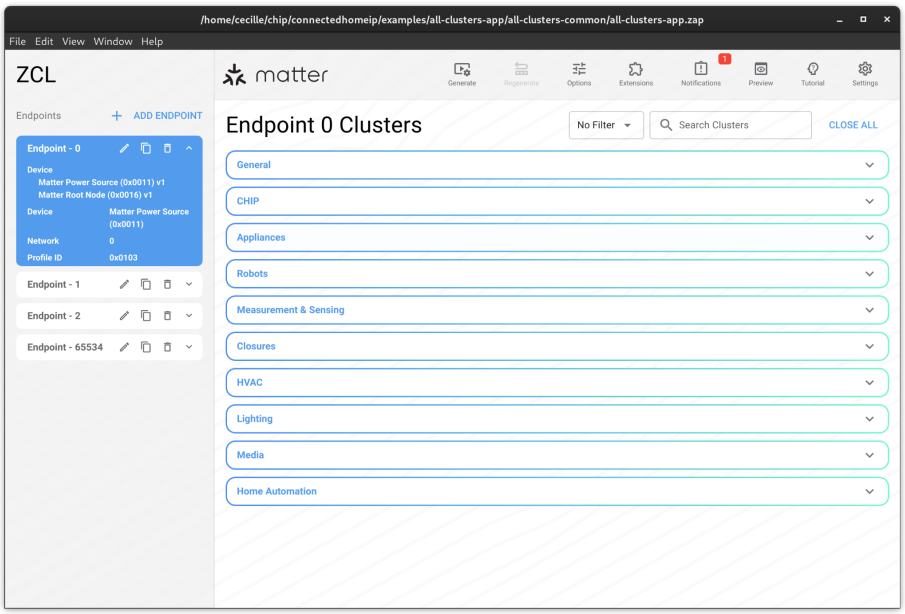


Figura 3.4 GUI *ZAP Tool*.

- **Storage option:** Define si el almacenamiento para el atributo se gestiona en RAM o externamente. Los clusters que usan la capa Ember almacenan en RAM, mientras que otros pueden implementar almacenamiento externo.
- **Type:** Tipo de dato del atributo, según la especificación.
- **Default:** Valor inicial deseado para el atributo, distinto del valor predeterminado en la especificación, si se implementa en la capa Ember.

Ejecución y configuración de *ZAP Tool*

Para abrir un archivo en ZAP y comenzar a configurarlo, se puede ejecutar el siguiente comando en la terminal:

Código 3.3 Ejecución de *ZAP Tool*.

```
./scripts/tools/zap/run_zaptool.sh <nombre_de_archivo.zap>
```

Esto abre la interfaz gráfica de ZAP, donde se pueden configurar los endpoints y clusters del dispositivo. La herramienta permite editar los clusters y sus atributos, como habilitar o deshabilitar ciertas funcionalidades, configurar valores predeterminados y definir el tipo de almacenamiento (RAM o externo).

Chip-REPL

Chip-repl es una herramienta interactiva basada en Python que actúa como un puente de comunicación entre el controlador Matter y los dispositivos Matter. A diferencia de *Chip Tool*, *chip-repl* proporciona un entorno de desarrollo en Python donde los desarrolladores pueden interactuar de manera flexible con dispositivos Matter en tiempo real, ejecutando comandos y probando funcionalidades a través de un REPL (Read-Eval-Print-Loop). En la figura 3.5 se puede observar la interfaz de *chip-repl* en la terminal.

Su principal función consiste en permitir a los usuarios realizar operaciones tales como *commissioning*, control de dispositivos y pruebas de atributos y comandos de los clusters de un dispositivo de forma directa dentro de un entorno Python. Esto se puede realizar a trevés de una interfaz interactiva que proporciona acceso directo a la API del controlador de Matter, lo que facilita las pruebas y el desarrollo de scripts personalizados para gestionar dispositivos, enviar comandos o ejecutar interacciones complejas con los dispositivos Matter.

La principal ventaja de chip-repl es la flexibilidad que ofrece, siendo muy sencillo su uso en combinación con otras herramientas y entornos Python, lo que lo hace ideal para realizar pruebas avanzadas, depuración o automatización de procesos. A través de este entorno, se pueden ejecutar el mismo tipo de acciones que se

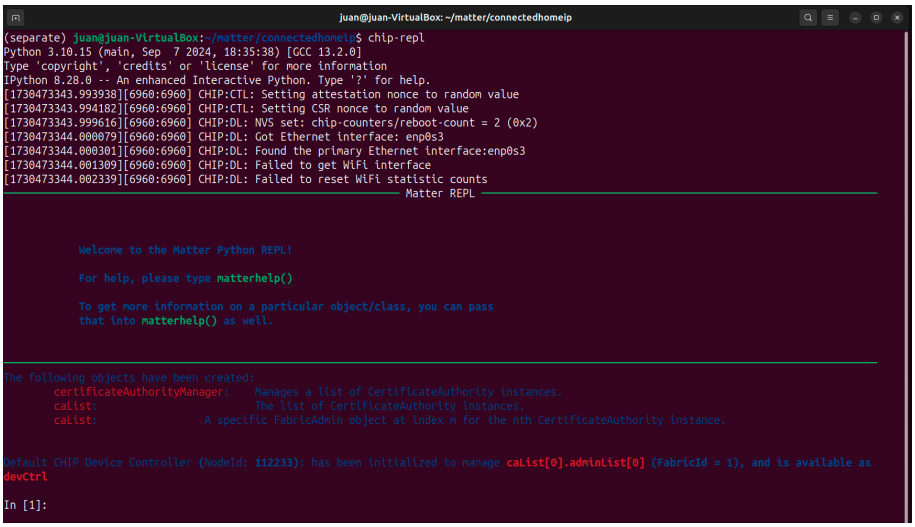


Figura 3.5 Interfaz *chip-repl*.

ejecutan con la herramienta *Chip Tool*. Además, el entorno permite trabajar tanto localmente como desde un navegador usando un Jupyter Lab alojado en la nube, proporcionando aún más accesibilidad y comodidad para los desarrolladores.

Configuración de *Chip-REPL*

Para trabajar con *chip-repl*, el usuario ha de generar en primer lugar un entorno controlado de Python en el que se incorporen todas las dependencias y bibliotecas necesarias. Para esto, se ha de generar un entorno virtual, de esta forma se pueden aislar los paquetes específicos requeridos por Matter y evitar conflictos con otras instalaciones globales de Python.

Para compilar y configurar el entorno virtual de Python, se ejecuta el siguiente script que genera el entorno y lo coloca en la carpeta `out/python_env`

Código 3.4 Creación entorno virtual de *chip-repl*.

```
scripts/build_python.sh -m platform -i out/python\_env
```

Una vez que el entorno ha sido configurado correctamente, se debe activar para poder utilizar chip-repl mediante el comando:

Código 3.5 Activación entorno virtual de *chip-repl*.

```
source out/python_env/bin/activate
```

Al activar este entorno, el sistema está listo para ejecutar el controlador de Matter dentro del entorno aislado.

Ejecucción de *chip-repl*

Después de activar el entorno virtual, se puede iniciar chip-repl. Este entorno interactivo se abre mediante el siguiente comando:

Código 3.6 Ejecución de *chip-repl*.

```
sudo out/python_env/bin/chip-repl
```

Este comando ejecuta el Read-Eval-Print-Loop (REPL) en un shell de Python, permitiendo que el usuario interactúe directamente con el controlador Matter desde la terminal. El entorno de REPL proporciona un termianl en bucle desde el que se puede introducir comandos, recibir resultados de inmediato y continuar interactuando con los dispositivos Matter.

3.3.3 Repositorio *ESP-IDF*

El *ESP-IDF* (Espressif IoT Development Framework) es el entorno de desarrollo oficial de Espressif para sus SoCs (System-on-Chip) basados en el microcontrolador ESP32, incluyendo el modelo ESP32-C6. Este framework, alojado en GitHub [17] al igual que el repositorio Matter, está diseñado para facilitar el desarrollo de aplicaciones IoT con soporte en múltiples plataformas incluyendo Linux.

Estructura y funcionalidades

El repositorio de *ESP-IDF* se encuentra organizado en las siguientes carpetas y archivos:

- **/components:** Contiene los componentes del framework, cada uno de los cuales proporciona funcionalidades específicas, como soporte para protocolos de red, controladores de hardware y bibliotecas.
- **/docs:** Documentación asociada con la utilización de *ESP-IDF*, que incluye guías de instalación, configuración y uso.
- **/examples:** Proyectos de ejemplo que explican cómo utilizar el framework para desarrollar aplicaciones sobre ESP32 y otros SoCs de Espressif.
- **/tools:** Scripts y herramientas para ayudar al desarrollo, instalación y uso del framework.
- **Archivos de configuración:** Contiene archivos tipo `CmakeList.txt` necesarios para gestionar la compilación y las dependencias del proyecto.

El entorno *ESP-IDF* ofrece una variedad de funcionalidades que permiten modificar las configuraciones del proyecto. Gracias al menu interactivo que incorpora, es posible realizar ajustess en parámetros específicos del proyecto según las necesidades del usuario. Además, el entorno facilita la compilación de la aplicación, generando tanto la tabla de particiones como el bootloader necesarios para el arranque del dispositivo. Una vez que el proyecto ha sido compilado, se puede flashear directamente el dispositivo *ESP*. También se proporciona la capacidad de visualizar la salida del puerto serial del dispositivo para poder monitorear su comportamiento en tiempo real.

Configuración del entorno *ESP-IDF*

Además de descargar los archivos necesarios desde el repositorio indicado para poder trabajar con el microcontrolador ESP32-C6-DevKitM-1, es necesario instalar herramientas y dependencias adicionales específicas para este modelo. Una vez clonado el repositorio, es importante ejecutar un comando que permita obtener tanto el compilador como el depurador. Para ello, desde el directorio `ESP-IDF`, se debe ejecutar lo siguiente:

Código 3.7 Instalación herramientas adicionales *ESP-IDF*.

```
cd ~/esp/esp-idf
./install.sh esp32c6
```

Tras instalar las herramientas del compilador y depurador, se han de configurar las variables de entorno para que *ESP-IDF* pueda utilizarlas.

Código 3.8 Configuración variables de entorno *ESP-IDF*.

```
. $HOME/esp/esp-idf/export.sh
```

Para evitar tener que ejecutar este comando cada vez que se desee utilizar *ESP-IDF*, es recomendable crear un alias que simplifique el proceso.

Código 3.9 Creación alias.

```
alias get_idf='. $HOME/esp/esp-idf/export.sh'
```

De esta forma cada vez que se requiera el uso del *ESP-IDF*, se puede ejecutar el comando `get_idf` directamente en el shell.

3.3.4 Docker

Docker [13] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización. Los contenedores son entornos aislados que contienen todo lo necesario para ejecutar una aplicación, garantizando que el entorno de ejecución sea siempre consistente sin importar el sistema que se esté utilizando.

Figura 3.6 Logo Docker.

Esto es especialmente interesante para proyectos como este donde se requiere consistencia entre las diferentes etapas de desarrollo, pruebas y producción que se realicen. Las ventajas principales que proporciona Docker son las siguientes:

- **Aislamiento y seguridad:** Docker permite mantener la ejecución de múltiples contenedores dentro del mismo host de forma aislada, otorgando una mayor seguridad y evitando conflictos entre dependencias.
- **Portabilidad:** Un contenedor Docker puede ser ejecutado en cualquier sistema que soporte Docker, desde ordenadores hasta servidores en la nube, sin necesidad de modificar el entorno de la aplicación.
- **Escalabilidad:** Los contenedores son ligeros y pueden ser escalados de forma rápida y eficiente, por lo que se facilita el ajuste de carga de trabajo según sea necesario.
- **Rapidez:** Como los contenedores son muy ligeros, se pueden ejecutar más servicios en el mismo hardware, optimizando la capacidad del servidor.

Arquitectura

Docker mantiene una arquitectura de modelo cliente-servidor. El cliente Docker interactúa con el *Docker Daemon*, el cual es el encargado de gestionar las imágenes, los contenedores, las redes y los volúmenes. La comunicación entre ambos se produce a través de una API REST, una interfaz de comunicación entre sistemas de información que utiliza el protocolo de transferencia HTTP, siendo posible ejecutar ambos componentes, cliente y *Daemon* en la misma máquina o bien en máquinas remotas.

- **Docker Daemon:** Es el proceso que gestiona las solicitudes de la API y se encarga de manejar los objetos Docker como contenedores e imágenes.
- **Docker Client:** Es la interfaz principal desde la cual los usuarios interactúan con Docker.
- **Docker Registries:** Representan los directorios donde se encuentran las imágenes Docker, siendo Docker Hub el registro por defecto aunque es posible configurar registros privados.

Docker Compose

Docker Compose es una herramienta derivada de Docker que simplifica el uso de aplicaciones que requieren múltiples contenedores para funcionar en conjunto. Mientras que Docker permite ejecutar contenedores de forma individualizada, Docker Compose es adecuado para casos en los que una aplicación necesita ejecutar múltiples servicios, tales como bases de datos, servicios web o microservicios que se comunican e interactúan entre ellos. Esta característica es fundamental para los entornos de trabajo como los utilizados en este proyecto.

Compose proporciona una mejor gestión de estos servicios utilizando un archivo de configuración YAML que explica cómo tiene que configurarse cada contenedor y las dependencias entre ellos. Este enfoque permite de una forma sencilla iniciar, detener o reiniciar todos los servicios que intervienen en la aplicación con un solo comando, además de verificar el estado de ejecución del sistema o la realización de pruebas específicas, permitiendo de esta manera facilitar la gestión y el despliegue de aplicaciones multicontenedores.

3.4 Especificaciones Matter empleadas

Entre las especificaciones de Matter empleadas, se han seleccionado una serie de *clusters* en base a los objetivos e intereses específicos del proyecto. Debido a que gran parte del proyecto se centra en la gestión de la monitorización y gestión del comsumo energético ligado al nodo sensor, se han seleccionado los *clusters Electrical Power Measurement* y *Energy Preference*. El motivo de la selección de estos *clusters* se ha decidido en base a la documentación oficial de Matter [1].

Destacar que ambas especificaciones forman parte de las nuevas funcionanlidades introducidas en la version 1.3 del estándar Matter, por lo que se encuentran clasificadas como *initial release* en su historial de revisiones. Esto indica que son características nuevas en el ecosistema Matter, aún en fases tempranas de adopción. Además, el soporte para el *cluster Energy Preference* está marcado como provisional en la documentación oficial, lo que sugiere que su implementación todavía está en evolución y probablemente sufran modificaciones en el futuro. Debido a su reciente lanzamiento, las implementaciones reales de dispositivos que utilizan estas funcionalidades no son muy numerosas por el momento.

3.4.1 *Electrical Power Measurement Cluster*

El *Electrical Power Measurement Cluster* resulta ser fundamental para la monitorización de los parámetros eléctricos en dispositivos Matter. Este *cluster* ofrece una interfaz para la lectura de datos de consumo que son medidos en tiempo real: voltaje, potencia activa y corriente activa, siendo claves para el desarrollo de cualquier aplicación que requiera un análisis del consumo energético. En el contexto de este proyecto, se ha trabajado directamente con tres atributos fundamentales: *Voltage* (Voltaje), *Active power*(Potencia activa) y *Active current*(Corriente activa).

Cada uno de estos atributos mide parámetros clave del consumo eléctrico. *Voltage* mide la tensión aplicada al dispositivo en milivoltios, *Active Power* mide la cantidad de energía consumida por el dispositivo en milivatios y *Active Current* mide la corriente que circula a través del dispositivo en miliamperios.

Estos tres atributos se representan como enteros sin signo*(unsigned integers)* de 16 bits. En cuanto a su acceso, todos estos atributos son *read-only* (de sólo lectura), siendo posible leerlos y obtener sus valores pero no modificar estos atributos por el sistema. Para cubrir la necesidad de simular variaciones en los datos eléctricos se ha ha implementado un delegado *(delegate)* desde el que se puede modificar los valores de estos atributos de forma interna. De esta modo se logra simular las variaciones reales del sistema en un entorno a través de una actualización continua de los atributos. Este delegado ha sido desarrollado a partir de la documentación técnica de Matter y de los códigos base del *cluster Electrical Power Measurement*, verificando que los valores que se almacenen puedan ser escritos y leídos tanto de forma interna como de forma externa a través de las herramientas de Matter. De esta manera, se permite efectuar finalmente una simulación precisa y completa del comportamiento energético del dispositivo dentro de una red Matter.

3.4.2 *Energy Preference Cluster*

El *Energy Preference Cluster* ofrece una interfaz para especificar y modificar las preferencias energéticas de un dispositivo, permitiendo así optimizar el consumo energético sin perjudicar el rendimiento. Este *cluster* resulta especialmente adecuado para dispositivos que han de gestionar dinámicamente los recursos energéticos de los que dispone en función de las prioridades del sistema. De esta manera, se puede establecer una estrategia como la de modos de ahorro de energía o de balance energético, siendo así más adaptable el dispositivo a diferentes entornos.

En este proyecto, el atributo del *cluster Energy Preference* utilizado es el *CurrentEnergyBalance*, que especifica la preferencia de energía actual del dispositivo. Este atributo es un entero de 8 bits sin signo que actúa como un índice que señala la posición actual en una lista de balances energéticos predefinidos. En su configuración estándar de Matter, los valores de este índice están relacionados con diferentes prioridades energéticas, como por ejemplo confort, velocidad o eficiencia. Sin embargo, para los fines de este proyecto, se ha redefinido la interpretación de estos índices para adaptarlos a tres modos: *Modo Normal* (0), *Modo Ahorro* (1) y *Modo Rendimiento* (2). Aunque la lista de balances previa no se ha modificado estructuralmente, la lectura de los índices 0, 1 y 2 se realiza de manera interna y externa para reflejar los tres modos operativos

de este proyecto.

A diferencia del *cluster Electrical Power Measurement*, el atributo *CurrentEnergyBalance* es de lectura y escritura, lo cual permite al sistema ajustar la preferencia energética del dispositivo en un momento determinado. Aun así, debido a las modificaciones previamente realizadas en la forma de interpretar los balances energéticos, ha sido necesario realizar modificaciones en el propio sistema de escritura de ese atributo y asegurarse de que existe coherencia con los nuevos índices definidos, lo cual ha requerido implementar un delegado de escritura para que se manejen correctamente los nuevos modos de energía, asegurando el cambio entre balances energéticos de un modo a otro.

4 Implementación, Resultados y Análisis

En el presente capítulo se detalla el proceso de implementación del proyecto, centrándose en el desarrollo del código necesario para el funcionamiento del microcontrolador ESP32C6-DevKitM-1 y los procesos de matter que permiten la interacción con el dispositivo. Se explica también cómo se ha desarrollado una interfaz web interactiva con la que el usuario puede ir monitorizando y cambiando, en tiempo real, los modos de funcionamiento del dispositivo. Todo lo descrito en este capítulo se fundamenta en los aspectos explicados previamente en el capítulo 3.

Todo el código desarrollado en este apartado se puede consultar en el siguiente repositorio de GitHub: *https://github.com/juanezete/electrical-sensor-matter*.

4.1 Desarrollo de la aplicación *Electrical Sensor App*

Esta sección se analiza el desarrollo de la aplicación denominada *Electrical Sensor App* diseñada para el microcontrolador ESP32C6-DevKitM-1. Esta aplicación tiene como objetivo producir valores simulados con el fin de generar de forma aleatoria los datos de consumo eléctrico, los cuales se ajustan de acuerdo al modo de operación en el que se encuentra el dispositivo.

Dentro de esta sección, se tendrán en cuenta las distintas cuestiones que se han abordado en su desarrollo, las cuales pasan por la modificación de la configuración del dispositivo utilizando la herramienta *Zap Tool*, con la que se configuraron los parámetros del dispositivo de acuerdo con las especificaciones Matter (como se describe en la seccion 3.4). Además, se analizará el código principal de la aplicación (*main.cpp*), entrando en detalle en su funcionamiento y en la descripción de los delegados que se han implementado a partir de las necesidades del propio proyecto.

Como punto de partida para el proyecto se han tomado de referencia los ejemplos del repositorio *connectedhomeip* de la carpeta *examples* [4]. Todo el desarrollo de la aplicación se ha organizado en una carpeta llamada *electrical-sensor-app*, la cual alberga todos los archivos generados específicamente para el microcontrolador en cuestión.

4.1.1 Configuración del archivo *Zap*

Para la configuración inicial del dispositivo, se generó un archivo *.zap* (sección 3.3.2) definiendo de esta forma la estructura de los *endpoints* y los *clusters*, estableciendo la configuración de este dispositivo en términos de conectividad y funcionalidad acorde con la especificación Matter.

En este proyecto se han creado los *endpoints* 0 y 1, cada uno con una función específica. El *endpoint* 0 está configurado como el dispositivo *Root Node*, que es el nodo principal del dispositivo, mientras que el *endpoint* 1 se configura como un dispositivo tipo *Electrical Sensor*.

En el *endpoint* 0 se han incluido todos los *clusters* requeridos en la sección general, incluyendo todos los obligatorios para la configuración base del dispositivo. Por otro lado, se han incluido los *clusters* específicos

de la sección CHIP, fundamentales para asegurar la conectividad de acuerdo al protocolo Matter. La configuración del *endpoint* 0 se muestra en la Figura 4.1.

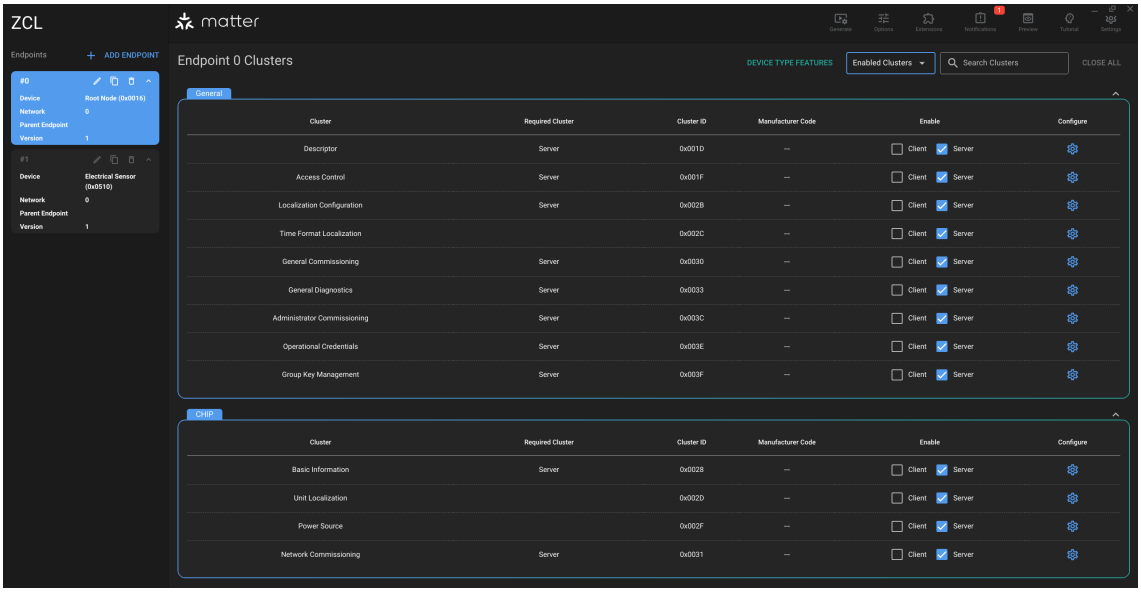


Figura 4.1 Configuración *Zap Tool Endpoint* 0.

En el *endpoint* 1, se configuró el *cluster Descriptor*, que permite al dispositivo describir a otros dispositivos de la red los servicios que ofrece , facilitando de esta manera su descubrimiento y operabilidad. En la sección *Measurement & Sensing*, se configuraron los *clusters Electrical Power Measurement*, *Electrical Energy Measurement* y *Power Topology* que son necesarios para configurar el dispositivo *Electrical Sensor*. Sin emabrgo, el proyecto se centra únicamente en tres atributos específicos del cluster Electrical Power Measurement (voltaje, potencia activa y corriente activa), como se detalla en la sección 3.4, por lo que el resto de *clusters* de esta sección permanecerán inactivos a pesar de haberse habilitado. La configuración del endpoint 1 se muestra en la Figura 4.2.

Por último, en la sección de *Energy Management* se activó el cluster *Energy Preference*, el cual es indispensable para configurar y gestionar el modo energético del dispositivo a través de su atributo *CurrentEnergyBalance*.

De esta forma, mediante el archivo zap generado, almacenado dentro de la carpeta *electrical-sensor-app/electrical-sensor-common*, el dispositivo queda completamente configurado con los parámetros definidos previamente. Esta configuración zap sirve como base para el desarrollo del código asociado al dispositivo, permitiendo establecer el código específico mientras se consideran las limitaciones de configuración definidas en esta etapa inicial.

4.1.2 Desarrollo de los delegados asociados

Para el desarrollo de código de la aplicación, se han utilizado diversos archivos asociados a los *clusters* definidos y empleados en el dispositivo. Estos archivos se encuentran organizados en las carpetas *include* y *src* de dentro de *electrical-sensor-common*, donde además se han implementado los delegados específicos para cada *cluster*. A continuación, se explicarán los principales componentes y se mostrarán las partes de código esenciales para cada delegado, específicamente para los *clusters Electrical Power Measurement* y *Energy Preference*. Estos delegados proporcionan la implementación necesaria para la lectura y configuración de los atributos, asegurando el funcionamiento adecuado del dispositivo. Para un análisis exhaustivo de los archivos de código, se puede acceder a su contenido completo en las rutas mencionadas (*include* y *src*).

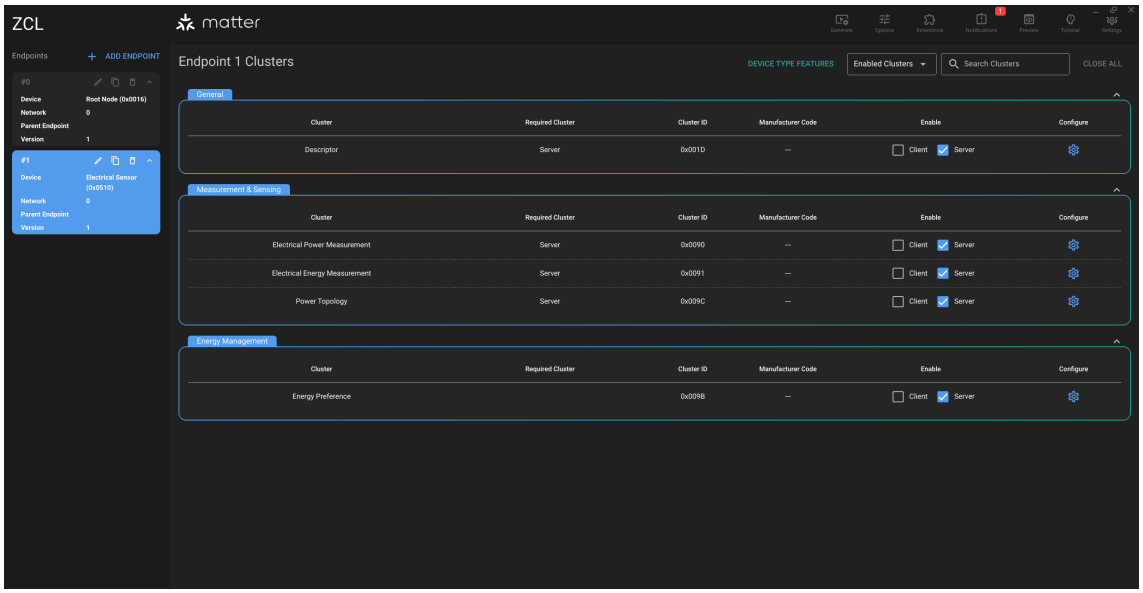


Figura 4.2 Configuración *Zap Tool Endpoint* 1.

ElectricalPowerMeasurementDelegate

El delegado *ElectricalPowerMeasurementDelegate* se encarga de gestionar las operaciones de lectura y actualización de los atributos del *cluster Electrical Power Measurement*. Su implementación se lleva a cabo a través de los archivos `electricalpowermeasurementdelegate.h` y `electricalpowermeasurementdelegate.cpp`, ubicados en las carpetas *include* y *src*, respectivamente. Este delegado permite el acceso controlado a los atributos de este *cluster*, aunque en el contexto de este proyecto se centra únicamente en la parte de atributos de medición de potencia eléctrica, en este caso voltaje, potencia activa y corriente activa. La clase define un conjunto de métodos que implementan las interfaces `Delegate` y `AttributeAccessInterface`, de tal forma que estos valores se encuentren accesibles de forma segura por parte de dispositivos Matter externos.

Para leer los atributos principales se utilizan métodos como `GetVoltage`, `GetActiveCurrent` y `GetActivePower` entre otros, los cuales devuelven los valores de los atributos encapsulados en variables de tipo `Nullable<int64_t>`, permitiendo que estos valores puedan ser devueltos en tiempo real desde el propio dispositivo, tal y como se puede ver en el código 4.1.

Código 4.1 Configuración obtención de atributos *ElectricalPowerMeasurementDelegate*.

```
DataModel::Nullable<int64_t> GetVoltage() override { return mVoltage; }
DataModel::Nullable<int64_t> GetActiveCurrent() override { return
    mActiveCurrent; }
DataModel::Nullable<int64_t> GetReactiveCurrent() override { return
    mReactiveCurrent; }
```

Para devolver el valor a un dispositivo externo, se emplea el método de lectura `Read`. Este método emplea un `ConcreteReadAttributePath` y un `AttributeValueEncoder` para determinar y codificar el valor del atributo solicitado. Si el atributo solicitado es *Voltage*, *ActiveCurrent* o *ActivePower*, el método devuelve el valor actual del atributo correspondiente. Si el atributo no es relevante para el proyecto, devuelve un error `CHIP_ERROR_UNSUPPORTED_CERT_FORMAT`. El comportamiento de este método se muestra en el código 4.2.

Código 4.2 Configuración lectura de atributos *ElectricalPowerMeasurementDelegate*.

```
CHIP_ERROR ElectricalPowerMeasurementDelegate::Read(const
    ConcreteReadAttributePath & aPath, AttributeValueEncoder & aEncoder)
{
    if (aPath.mAttributeId == Voltage::Id)
```

```
    {
        // Aquí se obtiene el valor actual del voltaje
        return aEncoder.Encode(mVoltage);
    }
    else if (aPath.mAttributeId == ActiveCurrent::Id)
    {
        // Aquí se obtiene el valor actual de la corriente activa
        return aEncoder.Encode(mActiveCurrent);
    }
    else if (aPath.mAttributeId == ActivePower::Id)
    {
        // Aquí se obtiene el valor actual de la potencia activa
        return aEncoder.Encode(mActivePower);
    }
    // Si el atributo no es manejado, retorna un error apropiado
    return CHIP_ERROR_UNSUPPORTED_CERT_FORMAT;
}
```

Para actualizar los valores internos de estos atributos, se definen métodos como `SetVoltage`, `SetActiveCurrent` y `SetActivePower`, que permiten modificar los valores de cada atributo. Estos métodos implementan callbacks de cambio (*MatterReportingAttributeChangeCallback*) para notificar al sistema de cualquier actualización, asegurando que los cambios sean reflejados en tiempo real. El código 4.3 muestra la implementación de `SetVoltage`.

Código 4.3 Ejemplo de modificación de atributo *Voltage*.

```
CHIP_ERROR ElectricalPowerMeasurementDelegate::SetVoltage(DataModel::Nullable<
    int64_t> newValue)
{
    DataModel::Nullable<int64_t> oldValue = mVoltage;

    mVoltage = newValue;
    if (oldValue != newValue)
    {
        // We won't log raw values since these could change frequently
        MatterReportingAttributeChangeCallback(this->Delegate::mEndpointId,
            ElectricalPowerMeasurement::Id, Voltage::Id);
    }

    return CHIP_NO_ERROR;
}
```

Estas secciones son las mas relevantes del delegado, permitiendo que *ElectricalPowerMeasurementDelegate* funcione como un intermediario, proporcionando una interfaz segura de acceso a los datos eléctricos del dispositivo almacenados en el *cluster*.

EnergyPreferenceDelegate

El delegado *EnergyPreferenceDelegate* implementa la lógica de control del *cluster Energy Preference*, que permite la gestión de las preferencias energéticas del dispositivo. Su implementación se lleva a cabo a través de los archivos `energypreferencedelegate.h` y `energypreferencedelegate.cpp`, ubicados en las carpetas *include* y *src*, respectivamente. Su función principal es proporcionar acceso y control sobre los modos energéticos, ajustando el comportamiento del dispositivo según las necesidades energéticas especificadas.

La función principal del delegado es proporcionar control de escritura sobre el atributo *CurrentEnergyBalance*, para ello el delegado incluye el método `WriteCurrentEnergyBalance`, el cual permite sobreescribir el valor del atributo. Este método comprueba que el valor introducido se sítue dentro del rango establecido para los modos de consumo energético (entre 0 y 2 en este caso), en caso de ser válido, actualiza el valor de

la variable. De esta manera, se asegura que el balance de energía esté sincronizado con el modo energético seleccionado. El código 4.4 muestra la implementación de `WriteCurrentEnergyBalance`.

Código 4.4 Modificación de *CurrentEnergyBalance* mediante `WriteCurrentEnergyBalance`.

```
CHIP_ERROR chip::app::Clusters::EnergyPreferenceDelegate::
    WriteCurrentEnergyBalance(chip::EndpointId aEndpoint, uint8_t newBalance)
{
    if (newBalance > 2)
    {
        ESP_LOGE(TAG, "Valor inválido para CurrentEnergyBalance: %u",
            newBalance);
        return CHIP_ERROR_INVALID_ARGUMENT;
    }

    // Actualizar el valor almacenado en RAM
    gCurrentEnergyBalance = newBalance;
    ESP_LOGI(TAG, "CurrentEnergyBalance actualizado a: %u", newBalance);

    return CHIP_NO_ERROR;
}
```

En el desarollo de este delegado no se han incorporado métodos de lectura adicionales, ya que el valor de *CurrentEnergyBalance* se puede acceder de forma externa gracias a la implementación proporcionada por el propio *cluster Energy Preference*. Los archivos de configuración de este *cluster* incluyen el método `Read`, utilizando los métodos del delegado como soporte. La implementación del método `Read` se presenta en el código 4.6.

Código 4.5 Lectura del atributo *CurrentEnergyBalance*.

```
Status MatterEnergyPreferenceClusterServerPreAttributeChangedCallback(const
    ConcreteAttributePath & attributePath,
EmberAfAttributeType attributeType, uint16_t size,
uint8_t * value)
{
    EndpointId endpoint = attributePath.mEndpointId;
    Delegate * delegate = GetDelegate();
    uint32_t ourFeatureMap;
    const bool featureMapIsGood = FeatureMap::Get(attributePath.mEndpointId, &
        ourFeatureMap) == Status::Success;
    const bool balanceSupported = featureMapIsGood && ((ourFeatureMap &
        to_underlying(Feature::kEnergyBalance)) != 0);
    const bool lowPowerSupported = featureMapIsGood && ((ourFeatureMap &
        to_underlying(Feature::kLowPowerModeSensitivity)) != 0);

    if (delegate == nullptr)
    {
        return Status::UnsupportedWrite;
    }

    switch (attributePath.mAttributeId)
    {
        case CurrentEnergyBalance::Id: {
            if (balanceSupported == false)
            {
                return Status::UnsupportedAttribute;
            }
```

```
            uint8_t index  = Encoding::Get8(value);
            size_t arraySize = delegate->GetNumEnergyBalances(endpoint);
            if (index >= arraySize)
            {
                return Status::ConstraintError;
            }

            return Status::Success;
        }
    ...
    }
}
```

Esta integración entre el código del cluster y el delegado desarrollado permite crear una interfaz para el acceso y la configuración de los modos de consumo eléctrico asociados al dispositivo, garantizando la correcta interacción con dispositivos Matter externos.

4.1.3 Desarrollo del código principal

Para el desarrollo del código principal que gestiona el funcionamiento del microcontrolador, se han utilizado dos archivos fundamentales: *main.cpp*, encargado del control general del dispositivo, y *DeviceCallbacks.cpp*, que contiene el *callback* específico del dispositivo encagado de gestionar las respuestas ante cambios en los atributos internos. Estos archivos se encuentran ubicados en la carpeta *esp32* dentro de *electrical-sensor-app*.

Antes de abordar en detalle el funcionamiento del código principal y del *callback*, se presenta la lógica general del programa, para facilitar así un contexto de su funcionamiento mucho más claro. A continuación se muestran las secciones que se consideran clave dentro del código implementado, con base en desarrollo de los delegados mencionados en el apartado anterior y en el *callback* que garantiza respuesta correcta del dispositivo. El código completo está disponible para su consulta en el directorio mencionado en la introducción de esta sección.

Lógica general del dispositivo

Para el funcionamiento del dispositivo es importante entender la lógica que se ha planteado para poder simular valores aleatorios de voltaje, corriente activa y potencia. Es importante recordar que el dispositivo opera en tres modos de trabajo: modo ahorro, modo normal y modo de rendimiento, que influyen directamente en las mediciones y el consecuente ajustes de los valores.

El voltaje se inicia con un valor fijo de 230 voltios. Cada medición simula una variación del 1 % en el valor del voltaje respecto a la medida inicial provocando un pequeño offset, este porcentaje de variación es un valor modificable desde el código. Para evitar que el voltaje se desvíe excesivamente, se ha establecido un límite adicional de variación en un 3 % superior y un 3 % inferior respecto al valor base, lo que permite mantener el voltaje dentro de márgenes razonables.

Para la corriente activa, se han establecido tres configuraciones diferentes, cada una asociada a los tres modos de funcionamiento: modo ahorro, modo normal y modo de rendimiento. En este caso y al igual que con el voltaje, la simulación se puede mover dentro de unos límites, en este caso entre 0 y 10 amperios. El modo de funcionamiento actual se determina a través del atributo *CurrentEnergyBalance* del *cluster Energy Preference*, que codifica el modo normal con el valor 0, el modo ahorro con el valor 1, y el modo rendimiento con el valor 2. El dispositivo verifica el modo de funcionamiento en cada lectura para ajustar el rango de variación en consecuencia. En modo ahorro, la corriente activa presenta un pequeño offset de ±5 mA; en modo normal, el offset es de ±50 mA; y en modo rendimiento, se permite una mayor variación de ±500 mA.

Tanto el voltaje como la corriente activa se ajustan mediante valores generados aleatoriamente que pueden ser tanto positivos como negativos, reflejando el comportamiento de un dispositivo en condiciones variables sin superar los límites establecidos.

Finalmente, la potencia activa se calcula en cada lectura multiplicando el valor de voltaje por el de corriente activa en ese momento, obteniendo así el consumo de energía en tiempo real.

Las mediciones y ajustes de los valores se realizan en intervalos de tiempo controlado, fijados en 30 segundos, aunque este periodo es también modificable si se requiere una frecuencia de actualización distinta.

En la Figura 4.3 se presenta un diagrama de flujo representativo de la lógica del dispositivo.

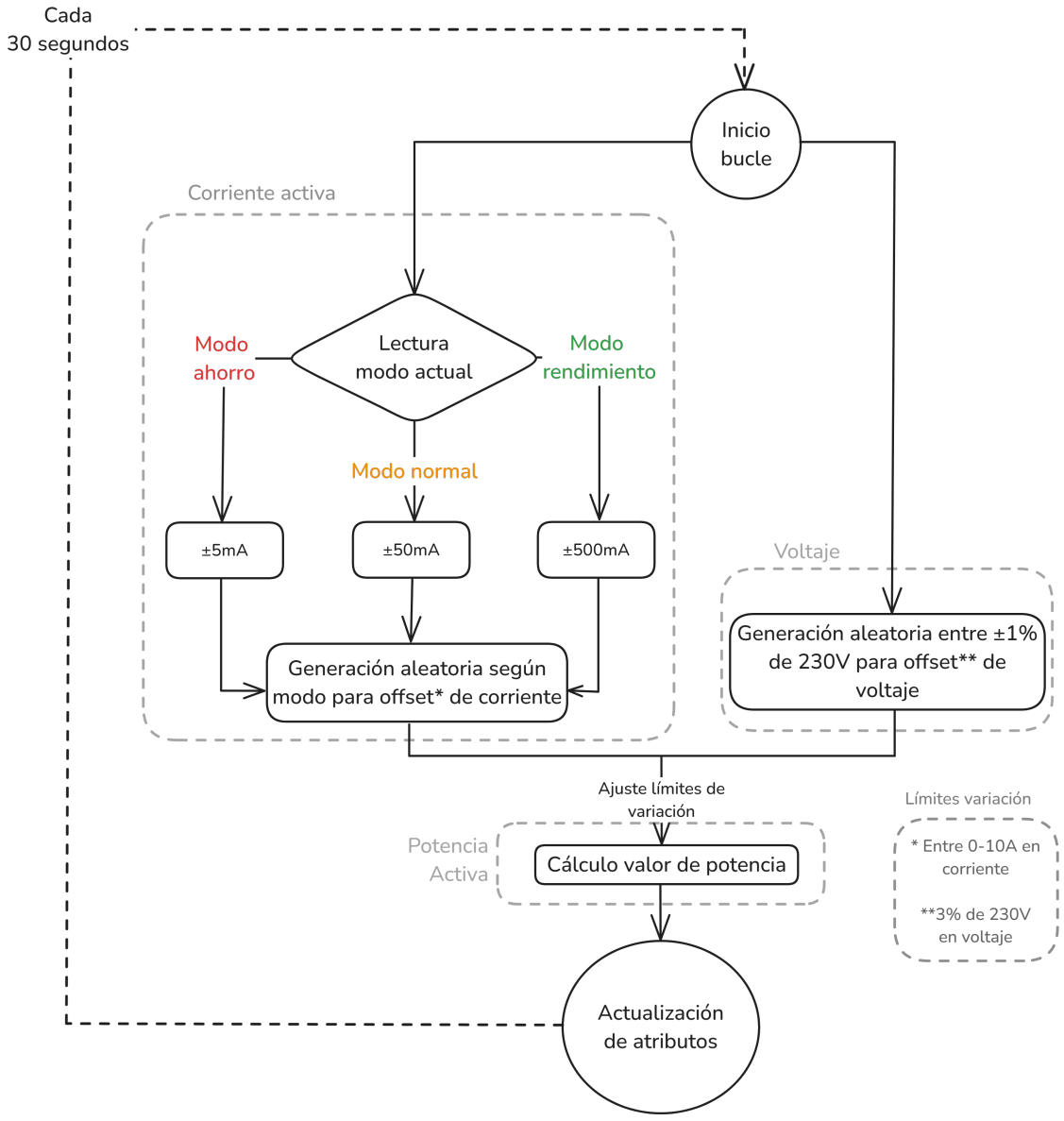


Figura 4.3 Diagrama de flujo lógica del dispositivo.

Callback del dispositivo

El dispositivo implementa un callback específico en el archivo *DeviceCallbacks.cpp*, diseñado para gestionar la respuesta ante modificaciones en los atributos configurados. Este único callback, denominado `PostAttributeChangeCallback`, es fundamental para la interacción del dispositivo con el entorno, ya que se invoca cada vez que un atributo del dispositivo cambia su valor.

`PostAtributeChangeCallback`, recibe varios parámetros: `EndpointId` (identificador del endpoint), `clusterId` (identificador del cluster), `attributeId` (identificador del atributo), además de infomación

adicional acerca del tipo y tamaño del atributo. En este caso, se resistra en el log todos estos parámetros junto con la cantidad de memoria libre en el dispositivo, permitendo monitorear en tiempo real el rendimiento del dispositivo.

Código 4.6 *Callback* del dispositivo.

```
void AppDeviceCallbacks::PostAttributeChangeCallback(EndpointId endpointId,
    ClusterId clusterId, AttributeId attributeId, uint8_t type, uint16_t size,
    uint8_t * value)
{
    ESP_LOGI(TAG, "PostAttributeChangeCallback - Cluster ID: '0x%" PRIx32 "',
        EndPoint ID: '0x%x', Attribute ID: '0x%" PRIx32 "'",
    clusterId, endpointId, attributeId);
    ESP_LOGI(TAG, "Current free heap: %u\n", static_cast<unsigned int>(
        heap_caps_get_free_size(MALLOC_CAP_8BIT)));
}
```

Este *callback* permite al dispositivo no solo gestionar y registrar los cambios de atributos, sino también notificar estos eventos de manera eficiente dentro del ecosistema Matter.

Código principal del dispositivo

El código principal del dispositivo está implementado en el archivo *main.cpp*. Este archivo integra todos los delegados explciados previamente junto con el *callback* y otros elementos esenciales para aplicar la lógica descrita en el apartado anterior.

Inclusión y definición de archivos clave

El código incluye todos los archivos de cabecera necesarios, tanto los generados en el apartado 4.1.2 como los archivos específicos de la plataforma ESP32C6-DevKitM-1 y del ecosistema Matter. Estos archivos proporcionan las funciones y estructuras necesarias para la interacción del dispositivo. Entre los archivos clave se encuentran las instancias de los delegados como `gElectricalPowerMeasurementDelegate` y `gEnergyPreferenceDelegarte`. Además, se configuran los proveedorees de credenciales para realizar la atestación del dispositivo mediante la función `get_dac_provider`.

Definición de constantes y parámetros

Se defunen una serie de constantes que regulan el comportamiento del dispositivo.

- `UPDATE_INTERVAL_MS`: Intervalo de tiempo entre actualizaciones (30 segundos).
- `INITIAL_VOLTAGE`,`INITIAL_ACTIVE_CURRENT` y `INITIAL_ACTIVE_POWER`: Valores iniciales de los atributos eléctricos del dispositivo.
- `MODE_NORMAL`, `MODE_SAVING`, y `MODE_INTENSIVE`: Definición de los modos de operación asociados a los diferentes modos de consumo energético.
- `OFFSET_VOLTAGE_PERCENTAGE` y `OFFSET_VOLTGE_MAX_PORCENTAGE`: Controlan el porcentaje de variación respecto al valor anterior y el porcentaje límite máximo de variación respectivamente.
- `OFFSET_CURRENT_MODE_*`:Valores de offset definidos para la corriente activa en miliamperios, dependiendo del modo de operación.

Funciones principales

1. `GetOperationMode()`: Esta función obtiene el modo de de operación actual del dispositivo consultando el atributo *CurrentEnergyBalance* del *cluster Eenrgy Preference*. Utiliza `StackLock`, bloqueando durante el proceso de lectura, para garantizar una lectura segura del atributo, y en función de su valor, devuelve el modo correspondiente: (`MODE_NORMAL`, `MODE_SAVING` y `MODE_INTENSIVE`).
2. `SetOperationMode(uint8_t newEnergyBalance)`: Actualiza el modo de operación estableciendo un nuevo valor en *CurrentEnergyBalance*. Al igual que en la función anterior, utiliza *StackLock* para segurar la operación y genera un log con el resultado.

3. `UpdateAttributes(void *pvParameters)`: Esta es la tarea principal que se ejecuta periódicamente, simulando las variaciones en voltaje, corriente y potencia en el dispositivo siguiendo la lógica descrita en la Figura 4.3. Utiliza las funciones de obtención y configuración del modo de operación, junto con las constantes definidas al inicio, para simular el comportamiento del dispositivo.
4. `ApplicationInit()`: Esta función inicializa la aplicación configurando componentes y lanzando tareas. Su secuencia es la siguiente: primero llama a `ElectricalControllerInit()` para configurar el controlador. Luego asigna un `EndpointId` al delegado de *Electrical Power Measurement*, establece `gEnergyPreferenceDelegate` como delegado principal para el cluster de gestión de modos de consumo y, finalmente, lanza la tarea de *FreeRTOS* `UpdateAttributes` para actualizar periódicamente los atributos (*FreeRTOS* es un sistema operativo de tiempo real que facilita la gestión de tareas concurrentes especialmente en microcontroladores).

Funciones de inicialización Matter

- `InitServer(intptr_t context)`: Esta función inicializa el servidor Matter y configura servicios adicionales. Muestra los códigos de *onboarding* para la configuración inicial, configura el servidor de aplicación apra la ESP32 (`Esp32AppServer`) y define los parámetros para el proceso de atestación del dispositivo. Si esta habilitado, también activa `ESP Insights` para monitorear el microcontrolador. Al finalizar, incia la aplciación principal llamando a `ApplicationInit`.
- `app_main`: Este es el punto de entrada de la aplicación, donde se realizan las configuraciones generales del sistema, se define el nivel de *log* para depuración, se inicializa la capa de almacenamiento no volátil (NVS) para guardar datos persistentes y se crea un bucle de eventos predeterminado. Posteriormente, se configura el `CHIPDeviceManager` para manejar las interacciones con Matter y se programa una tarea para ejecutar `InitServer`, iniciando de esta forma el servidor Matter.

Estos componentes forman parte de las secciones principales del archivo *main.cpp* que implementa la lógica funcional del dispositivo. Aunque existen otras secciones de código adicionales, se han destacado las partes mas importnates del código para facilitar la comprensión del funcionamiento.

A continuación, en la Figura 4.4, se presenta un diagrama de flujo que muestra la secuencia de ejecución y organización de las funciones principales del código, proporcionando una visión a grandes rasgos del flujo de control en el microcontrolador.

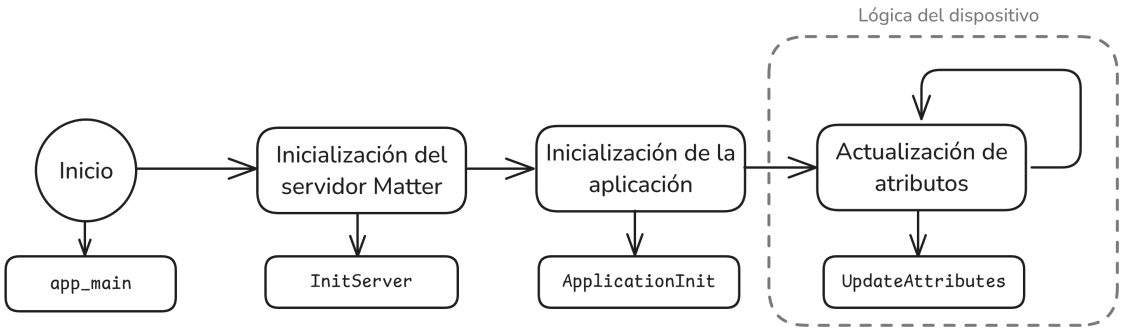


Figura 4.4 Diagrama de flujo código del dispositivo.

Compilación del código

Una vez finalizado el desarrollo completo de la aplicación *Electrical Sensor App* para el microcontrolador ESP32C6-DevKitM-1, se procede a compilar el archivo *main.cpp* junto con los demás archivos asociados, generando un binario *(build)* que permite cargar la aplicación en el microcontrolador.

Activación del entorno ESP-IDF

El primer paso es activar el entorno ESP-IDF, para ello se utiliza el alias creado previamente (ver código 3.8). Ejecutando en una terminal el siguiente comando:

Código 4.7 Activación entorno ESP-IDF.

```
get_idf
```

Configuración modelo microcontrolador
A continuación, se debe navegar hasta la carpeta *electrical-sensor-app/esp32*,y configurar el entorno para el modelo del microcontrolador. Esto se realiza con el comando:

Código 4.8 Configuración para modelos ESP32C6.

```
idf.py set-target esp32c6
```

Compilación del proyecto
Una vez realizada la configuración, se puede proceder a compilar el código con el siguiente comando:

Código 4.9 Compilación de la aplicación del microcontrolador.

```
idf.py build
```

Si el proceso de compilación se completa correctamente, se generará el binario que se podrá cargar en el microcontrolador.

Flasheo del binario
Para cargar el programa compilado en el dispositivo, se utiliza el comando de *flash*:

Código 4.10 Flasheo del binario.

```
idf.py flash
```

Monitorización en tiempo real
Finalmente, para monitorear los *logs* del dispositivo en tiempo real y verificar el funcionamiento, se utiliza el siguiente comando en una terminal:

Código 4.11 Monitorización del dispositivo.

```
idf.py monitor
```

De esta forma es posible visualizar los registros de actividad del dispositivo y diagnosticar posibles errores durante su funcionamiento.

4.2 Desarrollo interfaz de interacción

En esta sección se detalla el proceso de prueba y verificación de la conectividad del dispositivo tras la implementación de la aplicación *Electrical Sensor App* en el microcontrolador ESP32C6-DevKitM-1. El propósito de estas pruebas es garantizar una integración correcta del dispositivo dentro del ecosistema Matter, comprobando tanto la lectura de atributos simulados como la moficación del modo de funcionamiento en tiempo real.

En un primer momento, se realiza una verificación del correcto funcionamiento de la aplicación mediante una interacción directa con el dispositivo utilizando la herramienta *chip repl*. Una vez realizada la prueba, se desarrolla un script en Python que implementa la ejecución del test de forma automatizada, proporcionando una comunicación estable para la lectura de los atributos y la modificación dinámica y en tiempo real del modo de funcionamiento del dispositivo.

4.2.1 Comprobación de interacción con el dispositivo mediante chip-repl

Para la validación del correcto funcionamiento de la aplicación *Electrical Sensor App* en el microcontrolador se emplea la herramienta *chip-repl*, descrita en el apartado 3.3.2. Esta herramienta permite realizar las pruebas de *commissioning*, así como poder realizar pruebas de lecturas y modificaciones de atributos en tiempo real mediante la interfaz de control del dispositivo Matter (`devCtrl`) que se gestiona de forma interna mediante

chip-repl, lo que facilita la verificación del correcto funcionamiento del dispositivo dentro del ecosistema Matter.

El primer paso en el proceso es la configuración de la red Wi-Fi del dispositivo. Para ello, se pasan las credenciales de red (SSID y contraseña) al controlador de Matter (`devCtrl`), habiendo definido previamente estas credenciales como variables:

Código 4.12 Configuración de red wifi.

```
devCtrl.SetWiFiCredentials(WIFI_SSID, WIFI_PASSWORD)
```

Una vez configurada la red, se procede a realizar el *commissioning* a través de BLE. En este paso, también se definen previamente las variables `DISCRIMINATOR`, `SETUP_PIN_CODE` y `NODE_ID`, necesarias para el proceso de autenticación y emparejamiento del dispositivo:

Código 4.13 *Commissioning* del dispositivo.

```
await devCtrl.ConnectBLE(discriminator=DISCRIMINATOR, setupPinCode=
    SETUP_PIN_CODE, nodeid=NODE_ID)
```

Con el dispositivo correctamente enlazado, se confirma que el proceso de *commissioning* ha sido exitoso, lo que establece una conexión Matter entre el controlador y el dispositivo. Tras esta verificación, se continúa con la comprobación de los métodos de lectura y escritura de atributos.

Para leer un atributo específico, como el voltaje en este caso, se utiliza el siguiente comando, donde se identifica el `NODE_ID` y el atributo de *ElectricalPowerMeasurement* correspondiente:

Código 4.14 Lectura atributos del dispositivo.

```
response = await devCtrl.ReadAttribute(NODE_ID, [(1, clusters.
    ElectricalPowerMeasurement.Attributes.Voltage)])
```

Para modificar un atributo, como el nivel de consumo energético representado por `CurrentEnergyBalance` en el *cluster Energy Preference*, se ejecuta el siguiente comando. Aquí, `change_level` representa el valor que se desea asignar al atributo:

Código 4.15 Escritura atributos del dispositivo.

```
attribute_instance = clusters.EnergyPreference.Attributes.CurrentEnergyBalance(
    value=change_level)

await devCtrl.WriteAttribute(nodeid=NODE_ID,attributes=[(1, attribute_instance)
    ],interactionTimeoutMs=10000)
```

La correcta ejecución de estos comandos en *chip-repl* y la respuesta adecuada en el entorno, acompañada de los registros (logs) esperados en el dispositivo, demusetran que la aplicación *Electrical Sensor App* funciona como se espera, permitiendo la interacción completa con el ecosistema Matter.

4.2.2 Lógica de interacción con el dispositivo

En esta sección se desarrolla la lógica de interacción y comunicación planteada para automatizar el proceso de conexión con el microcontrolador y optimizar el flujo de datos. El sistema que se plantea integra varios elementos: un control del dispositivo basado en *chip-repl*, una base de datos para almacenamiento de las lecturas y el estado del dispositivo, y una interfaz web para la interacción con el usuario.

La interacción con el dispositivo y la ejecución de comandos Matter, como la lectura y escritura de atributos, se gestionan mediante *chip-repl*. Esta herramienta se utiliza dentro de un script en Python que también se comunica con una base de datos, la cual actúa como un puente entre el código de control del dispositivo y la interfaz web, permitiendo compartir información y estados del dispositivo de manera eficiente y continua.

El flujo general de interacción del usuario con el dispositivo se desarrolla de siguiente manera:

1. **Interfaz web**: El usuario interactua con el sistema a traves de una interfaz web, desde donde se pueden consultar las lecturas de los atributos del dispositivo y cambiar el modo de funcionamiento del dispositivo.
2. **Base de datos**: La interfaz web y el código de control del dispositivo están conectados a una base de datos local, diseñada para almacenar el historial de lecturas y los cambios en el modo de funcionamiento del dispositivo. Esta base de datos se compone de los siguientes elementos:
   - `Timestamp`: Marca temporal que indica el momento exacto de cada lectura de los atributos, permitiendo realizar un seguimiento histórico de los datos.
   - *Atributos de lectura*: Se registran los valores de los atributos principales del dispositivo (voltaje, corriente activa y potencia activa) en cada instante.
   - `current_level`:Este campo almacena el modo de operación en el que se encontraba el dispositivo en el momento de la lectura (0 para el modo normal, 1 para el modo ahorro, y 2 para el modo rendimiento).
   - `updated`: Este campo actúa como una bandera (*flag*) que indica si el usuario ha solicitado un cambio en el modo de operación desde la última lectura. Un valor de "1" en este campo señala que hay una solicitud pendiente de cambio que aún no se ha aplicado, mientras que un valor de "0" indica que el dispositivo está operando en el modo configurado adecuado, sin modificaciones pendientes.
   - `change_level`: Este campo almacena el modo de operación nuevo que se desea aplicar por el usuario a través de la interfaz web. De esta forma es posible registrar el cambio sin interferir en el modo en que se encuentra el dispositivo en el momento de la lectura. Cuando `updated` está activo ("1"), el sistema de control inicia el proceso de actualización para aplicar el nuevo modo en el dispositivo.
3. **Control del dispositivo**: El sistema de control del dispositivo, implementado mediante *chip-repl* y gestionado a través de un script en Python, se encarga de llevar a cabo el *commissioning* y de realizar lecturas y modificaciones de los atributos del dispositivo según los cambios solicitados.
4. **Microcontrolador ESP32C6-DevKitM-1**: El dispositivo ejecuta de forma continua la aplicación *Electrical Sensor App*, explicada en el apartado 4.1.

La figura 4.5 presenta un diagrama de flujo de la lógica de interacción entre los distintos elementos del sistema, desde el usuario hasta el dispositivo final.

Este sistema de interacción planteado permite automatizar el proceso de comunicación con el dispositivo, optimizando el flujo de datos y permitiendo un acceso completo al historial de lecturas.

4.2.3 Código de control del dispositivo

En este apartado se describe el código Python desarrollado para el control del dispositivo a través de la herramienta *chip-repl*. El script, denominado *control.py* emplea las funciones proporcionadas por *chip-repl* para gestionar el dispositivo de forma automatizada además de gestionar la comunicación con la base de datos que almacena las lecturas y el estado del dispositivo siguiendo la lógica comentada en apartados previos. El código al completo se encuentra disponible junto con el resto de archivos del proyecto y sigue la estructura que se detalla a continuación:

Configuración Inicial

El código comienza importando librerías esenciales para la funcionalidad, como `argparse` (manejo de argumentos de línea de comandos), `asyncio` (para operaciones asíncronas), `sqlite3` (interacción con la base de datos) y `logging` (para la creación de logs), además del módulo `clusters` de `chip` (módulo heredado de la herramienta *chip-repl*) para acceder a los atributos Matter del dispositivo. Asimismo, se configuran las credenciales de red Wi-Fi y los parámetros de conexión del dispositivo (`SETUP_PIN_CODE`, `DISCRIMINATOR`, `NODE_ID`) para realizar el proceso de *commissioning* o reconexión.

Inicialización de la base de datos

La función `initialize_database` verifica la existencia del archivo donde se almacena la base de datos

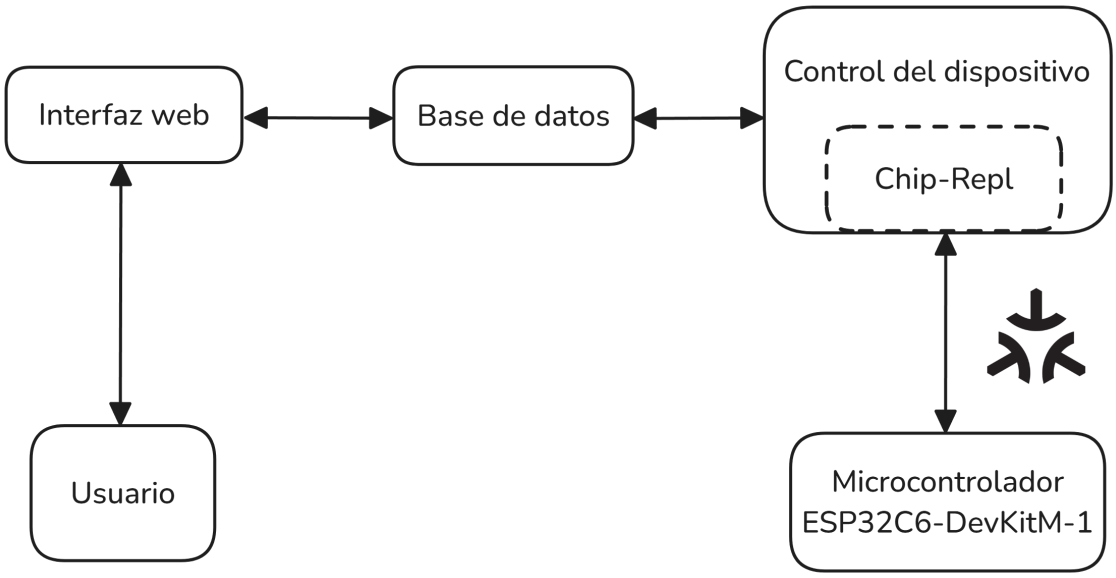


Figura 4.5 Diagrama de flujo lógica general.

`sensor_data.db` y en caso de no existir, crea la base de datos y su tabla asociada `measurements`. Esta tabla se ha configurado para almacenar los siguientes valores de `timestamp`, `voltage`, `active_current`, `active_power`, `current_level`, `updated` y `change_level`, cuyas funciones se detallaron en la sección anterior 4.2.2.

Lectura de atributos

La función `read_attributes` se ejecuta periódicamente cada 30 segundos para leer los atributos del dispositivo. Utilizando el método de lectura de *chip-repl*, se obtienen los valores de voltaje, corriente activa, potencia activa y modo de operación actual, los cuales se muestran en la terminal y se almacenan en la base de datos con un `timestamp`. La función inscribe estos valores como un nuevo registro en la tabla `measurements` de la base de datos y establece el valor de `updated = 0`, indicando que no hay cambios pendientes en el modo. Además cuenta con manejo de errores para capturar y registrar cualquier posible fallo durante el proceso.

Control modo de funcionamiento

La función `control_mode` revisa contínuamente la base de datos para detectar solicitudes de cambio de modo. Si encuentra que el valor `updated = 1` en el registro más reciente, toma el valor de `change_level` y crea una instancia del atributo `CurrentEnergyBalance` con este nuevo modo de operación. A continuación, usa el método `WriteAttribute` para actualizar el dispositivo. Una vez aplicado el cambio, actualiza el registro en la base de datos devolviendo `updated = 0` para marcar el cambio como procesado.

Función principal

La función principal usa `argparse` para manejar los argumentos `-c` o `-commissioning`. Si alguno de estos argumentos está presente al ejecutar *control.py*, el código configura las credenciales de la red Wi-Fi y realiza el *commissioning* mediante BLE. En caso de no incluir estos argumentos, el código asume que el dispositivo ya ha sido comisionado y procede a intentar la reconexión a través de `GetConnectedDevice`. Si la reconexión falla, intenta restablecer la sesión PASE o resolver el nodo para obtener una nueva dirección, según sea necesario. Finalizados los procesos de conexión, el código lanza las funciones `read_attributes` y `control_mode` en paralelo utilizando `asyncio.gather`, lo que permite que ambas funciones se ejecuten de manera asíncrona.

Ejecución del script

Al iniciar el script, se utiliza el siguiente comando para ejecutar `main` directamente, lo que permite iniciar el sistema al completo:

Código 4.16 Comando de ejecución del *main* en *control.py*.

```
if __name__ == "__main__": asyncio.run(main())
```

De esta forma, con la estructura planteada se obtiene un script que permite una interacción automática con el dispositivo directamente desde Python, eliminando la necesidad de utilizar manualmente la herramienta *chip-repl* e ingresar comandos en la terminal. En la Figura 4.6 se puede observar una representación en forma de diagrama de la lógica que sigue el código de control.

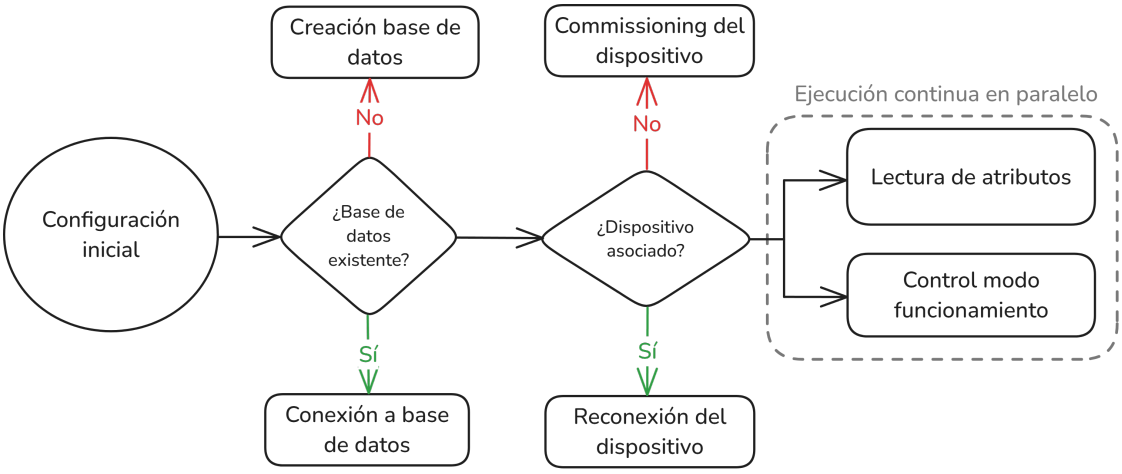


Figura 4.6 Diagrama de flujo código de control.

4.3 Desarrollo de la interfaz web

En esta sección se describe el proceso de generación de la web creada para permitir al usuario interactuar con el dipositivo y la interfaz de interacción. El objetivo principal es desarrollar una plataforma visual que permita seguir los atributos del dispositivo en tiempo real y cambiar los modos de operación de manera sencilla.

Inicialmente, se describe el código de control de la interfaz web, el cual se ha desarrollado en Python utilizando la herramienta *Flask*, un microframework que se centra en el desarrollo de aplicaciones web ligeras y eficientes. Este código, contenido en el archivo `web.py`, pone en marcha la comunicación con la base de datos, manejando la visualización y la actualización de los datos en tiempo real. El archivo mencionado se encuentra junto al resto de los elementos que componen el proyecto. En la subsección dedicada a la implementación de la interfaz web se detalla la organización de la plataforma web en términos de diseño HTML, con una descripción de su estructura y una explicación de la funcionalidad de cada parte de la página. El archivo `index.html` que implementa dicha estructura, se encuentra en la carpeta *templates* junto al resto de recursos en el repositorio del proyecto.

4.3.1 Código de control de la interfaz web

El código desarrollado para la aplicación web se ha elaborado en Python empleando el framework de *Flask*, que proporciona la infraestructura necesaria para la gestión de la interfaz HTML mediante la cual el usuario puede interactuar con el sistema. El archivo fuente de la aplicación es `web.py`, el cual se conecta con la base de datos a traves de *SQlite*, una solución ligera que permite el almacenar y manejar bases de datos en Python, la estructura que sigue la base de datos es la que se ha definido anteriormente.

El código se organiza en secciones específicas que gestionan funcionalidades determinadas. En primer lugar, se inicializa la aplicación *Flask* con la instancia `app = Flask(__name__)`, la cual actúa como base de la aplicación, encargándose de definir rutas y controlar el flujo de solicitudes HTTP.

La función `get_data()` es la encargada de conectarse a la base de datos `sensor_data.db` y recuperar los 10 últimos registros de la tabla *measurements*. Esta función incluye un mecanismo de manejo de errores con `try-except` para detectar posibles fallos de conexión y devolver una lista vacía en caso de error. Los registros obtenidos siguen el formato de la base de datos descrito anteriormente, conteniendo información como el `timestamp`, `voltage`, `active_current`, `active_power`, `current_level`, un indicador de actualización (`updated`) y el `change_level`. Estos datos se formatean y se devuelven en una lista que se envía a la interfaz de usuario para su visualización.

La ruta `@app.route('/update_level', methods=['POST'])`, permite al usuario modifcar el nivel de operación del dispositivo. La función asociada a esta ruta recibe el nuevo nivel de operación desde el formulario HTML, lo convierte a un valor entero y actualiza la base de datos estableciendo el campo `change_level` y marcando el indicador `updated` a 1 en el último registro advirtiendo al sistema de un cambio de modo de opración pendiente.

La ruta principal, definida por `@app.route('/')`, se vincula a la función `index()`, que se encarga de cargar la página de inicio de la interfaz web. Esta función utiliza `get_data()` para obtener los datos más recientes de la base de datos y los envía al archivo `index.html` mediante `render_template()`. Si no hay datos disponibles, la función proporciona valores por defecto para prevenir errores de renderizado en la página web.

La ejecución de la aplicación se inicia cuando el script se ejecuta directamente. En ese caso, la función `app.run()` lanza el servidor Flask en la dirección `0.0.0.0` en el puerto 5000, con el modo `debug` desactivado para asegurar un entorno de producción adecuado, permitiendo que los usuarios accedan a la interfaz y realicen actualizaciones en tiempo real.

La arquitectura del código permite una interacción continua con la base de datos, asegurando un flujo de datos en tiempo real y un soporte adecuado para la gestión de la interfaz web y sus funcionalidades.

4.3.2 Desarrollo de la interfaz web

El diseño de la interfaz web se ha implementado mediante el archivo `index.html` que define la estructura y la funcionalidad de una pagina web diseñada para interactuar con el dispositivo. A continuación, se explica el contenido y las secciones clave del archivo:

Estructura general
El documento HTML comeinza con la definicion de `<head>`, que incluye referencias a bibliotecas externas esenciales para la funcionalidad y el estilo de página empleado. Entre estas bibliotecas se encuentran *Bootstrap* para proporcionar un diseño en detallado, *Chart.js* para la representación de gráficos interactivos y *Font Awesome* para la utilización de iconos visuales. También se definen estilos personalizados mediante la etiqueta `<style>`, que mejoran la presentación de los elementos de la página, como las tablas y los indicadores (*gauges*).

Encabezado
Dentro de `<head>`, se especifica el título de la página (`<title>Mediciones del Sensor</title>`). Se cargan las hojas de estilo de *Bootstrap*, los iconos de *Font Awesome* y la biblioteca *Google Fonts*, que permite emplear la fuente *Roboto* para un diseño moderno y legible. Se define un bloque de estilos personalizados que mejoran la visualización de elementos como los (*gauges*) y tablas mediante animaciones, agregando un toque dinámico a la experiencia del usuario.

Cuerpo de la página (`<body>`)
El cuerpo comienza con un contenedor principal que incluye el título "Nodo Sensor ESP32C6". A continuación, la estructura se divide en diferentes secciones:

- **Indicadores del estado actual (*Gauges*)**: Esta sección muestra tres indicadores, cada uno representa un atributo del dispositivo (voltaje, corriente activa y potencia activa). Los valores mostrados corresponden a las mediciones más recientes registradas en la base de datos. Los indicadores muestran íconos representativos de cada atributo para una mejor identificación visual.

- **Visualización y control del modo de operación**: En esta sección, se muestra el modo actual del dispositivo junto a un icono indicativo. Justo a la derecha, un formulario permite al usuario seleccionar entre los modos de ahorro, normal y rendimiento mediante botones de selección. Al elegir un nuevo modo y pulsar el botón "Actualizar", se envía una solicitud `POST` al servidor, gestionado por el script *web.py*, para actualizar la base de datos con el nuevo nivel de operación.
- **Gráficas de evolución de los atributos**: Esta sección contiene un sistema de pestañas que permiten alternar entre gráficos de voltaje, corriente activa y potencia activa. Cada pestaña está vinculada a un elemento donde se renderiza la gráfica correspondiente usando *Chart.js*. Las gráficas son interactivas y muestran la evolución temporal de los datos, con el tiempo en el eje de abcisas y los valores medidos en el eje de ordenadas.
- **Tabla de últimos valores recibidos**: La sección final muestra una tabla que presenta los últimos 10 registros de medición recibidos, las columnas que incluyen la fecha y hora de la lectura, el voltaje, la corriente activa, la potencia activa y el modo de operación durante la lectura.

Funcionalidad dinámica de la interfaz
Al final del archivo *index.html* se incluye un bloque de código JavaScript que se ejecuta cuando la página termina de cargar (`DOMContentLoaded`). Este código define la lógica para la creación de gráficos interactivos mediante *Chart.js* y la actualización dinámica de los indicadores en la página. Las funciones JavaScript también manejan la actualización del icono que representa el modo de operación actual del dispositivo.

En la figura 4.7 se presenta un ejemplo de la disposición final de la interfaz web, mostrando la forma en que los datos se representan y la disposición de los elementos en la página.

4.4 Dockerización del proyecto

En este apartado se explica la automatización del proceso de ejecución del código *control.py*, que gestiona la comunicación entre el dispositivo y la base de datos, y del código *web.py*, responsable de la interacción entre la interfaz web y la base de datos. Para ello, se ha decidido utilizar la herramienta *Docker* para crear dos contenedores independientes que gestionen cada una de estas partes. De este modo, se pueden ejecutar ambos componentes de forma conjunta utilizando *docker-compose*. En esta sección se explican tanto los detalles de cada contenedor como el código que se ha empleado para su creación.

4.4.1 Contenedor chip-repl

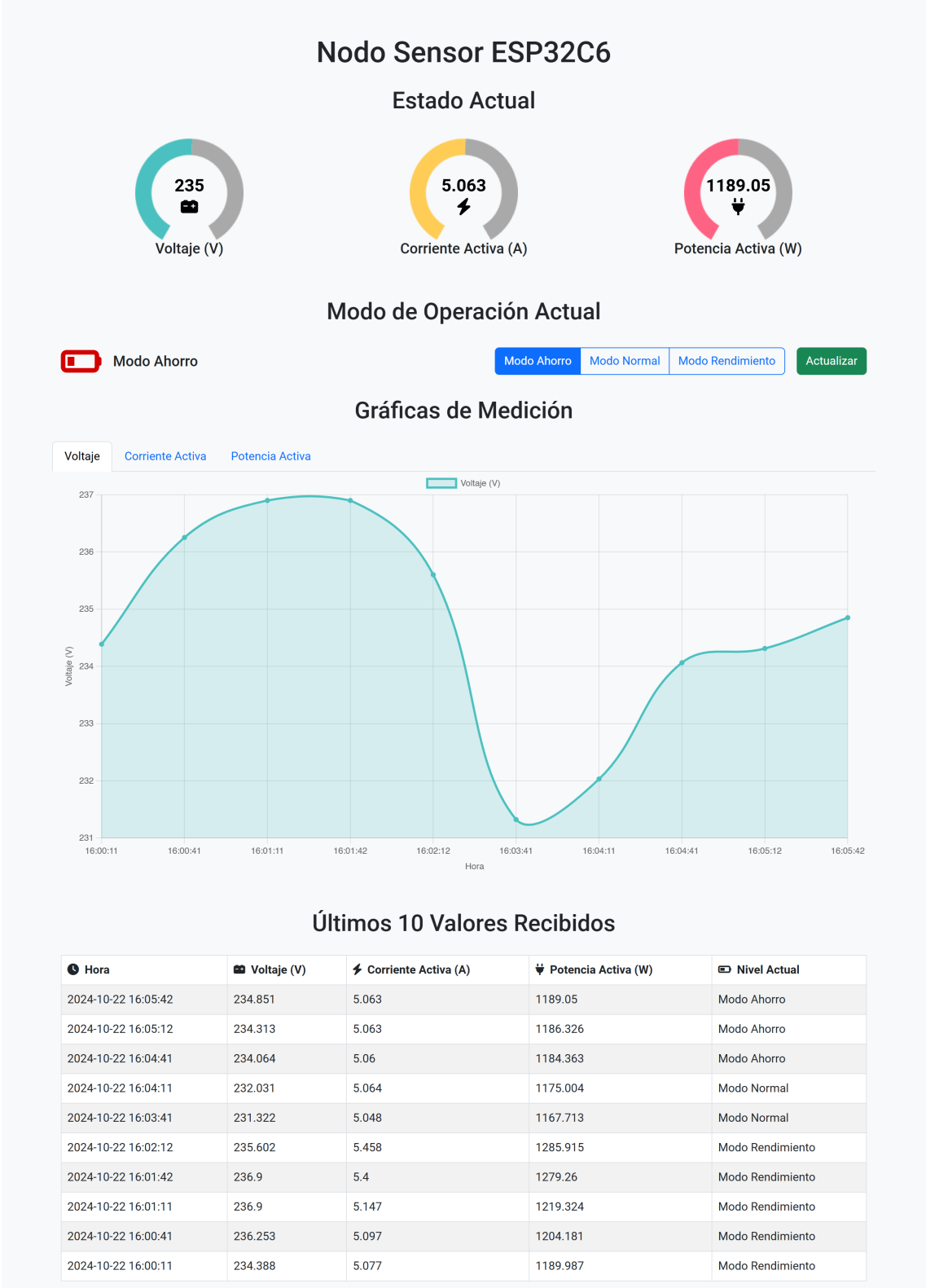
El contenedor tiene como objetivo ejecutar *control.py*. Para lograr esto, el contenedor sebe tener acceso a los archivos del respositorio *connectedhomeip* que se ha descargado localmente y también disponer de acceso al entorno virtual donde se ha instalado la herramienta *chip-repl*. Dadas estas dependencias, además del *Dockerfile* particular llamado *Dockerfile_chip-repl*, es necesario crear un *shell* llamado *control_chip-repl.sh* que se ejecute dentro del contenedor para gestionar estos procesos y asegurar que la herramienta funcione correctamente. El código del *Dockerfile* así como el *shell* asociado están disponibles junto con el resto de archivos del proyecto.

Dockerfile chip-repl

El archivo *Dockerfile_chip-repl* se ha diseñado para replicar un entorno similar al del sistema local, permitiendo la ejecución del script *control.py* mediante la herramienta *chip-repl*. A continuación, se detallan las secciones principales del *Dockerfile*:

Imagen base
El contenedor se construye sobre la imagen de Ubuntu 24.04. Esto garantiza que sea compatible con el entorno local de desarrollo, así como con las herramientas y dependencias requeridas para ejecutar *chip-repl* y el script de control.

Configuración de directorio y herramientas
El *Dockerfile* establece un directorio de trabajo en el mismo directorio donde se encuentra el repositorio *connectedhomeip*. De esta manera, los comandos y los scripts asociados se asegura que se ejecuten dentro de un contexto adecuado. Se lleva a cabo una actualización del sistema y se instalan herramientas básicas



Hora	Voltaje (V)	Corriente Activa (A)	Potencia Activa (W)	Nivel Actual
2024-10-22 16:05:42	234.851	5.063	1189.05	Modo Ahorro
2024-10-22 16:05:12	234.313	5.063	1186.326	Modo Ahorro
2024-10-22 16:04:41	234.064	5.06	1184.363	Modo Ahorro
2024-10-22 16:04:11	232.031	5.064	1175.004	Modo Normal
2024-10-22 16:03:41	231.322	5.048	1167.713	Modo Normal
2024-10-22 16:02:12	235.602	5.458	1285.915	Modo Rendimiento
2024-10-22 16:01:42	236.9	5.4	1279.26	Modo Rendimiento
2024-10-22 16:01:11	236.9	5.147	1219.324	Modo Rendimiento
2024-10-22 16:00:41	236.253	5.097	1204.181	Modo Rendimiento
2024-10-22 16:00:11	234.388	5.077	1189.987	Modo Rendimiento

Figura 4.7 Disposición interfaz web.

como `software-properties-common` y `curl` para gestionar repositorios y descargas. Se realiza también la instalación de Python 3.10, garantizando la compatibilidad con la versión de Python utilizada en el proyecto.

A continuación se llevan a cabo las instalaciones de todas las dependencias necesarias, incluyendo bibliotecas y herramientas de compilación, para asegurar el correcto funcionamiento de *chip-repl* y sus componentes.

Configuración de servicios y almacenamiento
Para poder detectar los servicios de red y llevar a cabo las comunicaciones entre procesos, se habilitan los servicios de `Avahi` y `D-Bus`. Esta parte es esencial para posibles conflictos con los drivers de Bluetooth o los servicios de conectividad del entorno de la máquina virtual.

Se establece también un directorio persistente para el almacenamiento de datos, guardando en un archivo temporal las credenciales y detalles necesarios para la reconexión del dispositivo. Esto permite que el contenedor guarde información relevante entre reinicios y se prepare para futuras reconexiones.

Comando shell de inicio
El archivo de *shell* denominado *control_chip-repl.sh* se copia al contenedor y a su vez se le coneceden los permisos de ejecución necesarios. Este script constituye el punto de entrada del contenedor y es responsable de gestionar la ejecución de *chip-repl* así como de otros procesos relacionados, habiendo asegurando previamente que el entorno esta correctamente configurado y listo para operar.

De esta forma, se define el archivo de configuración *Dockerfile* para el contenedor para permitir la instalación de todos los archivos y herramientas necesarias. A continuación, se procede a la ejecución del *shell* asociado para continuar con la configuración e interacción con el contenedor.

Shell para el contenedor

El archivo de script *control_chip-repl.sh* se encarga de iniciar los servicios necesarios, activar el entorno virtual de Python y ejecutar la herramienta *chip-repl* para que se garantice que el código *control.py* puede funcionar sin problemas. A continuación se detallan las secciones más relevantes del código:

Preparación de directorios y entorno de ejecución
El script es el encargado de establecer previamente la variable de entorno `TERM` como `xterm` para asegurar la compatibilidad de la terminal. Posteriormente, se crean los directorios requeridos dentro de `/run` para el funcionameinto de los servicios de `dbus` y `avahi-daemon`, eliminando los posibles archivos de procesos anteriores para evitar conflictos durante la ejecución.

Inicio de servicios esenciales
La siguiente sección del script se encarga de iniciar los servicios esenciales para la operación del contenedor:

- `dbus-daemon -system -fork`: Inicia el servicio `D-Bus` a nivel de sistema, permitiendo la comunicación entre procesos del propio sistema operativo en el contenedor.
- `avahi-daemon -daemonize -no-chroot`: Activa el servicio `Avahi` para la detección de servicios de red, operando en segundo plano y sin estar en un entorno aislado (*chroot*).
- `bluetoothd &`: Inicia el servicio, permitiendo la gestión de conexiones *Bluetooth* dentro del contenedor.

Posteriormente, a través del comando `hciconfig hci0 up` se utiliza para comprobar que el adaptador *Bluetooth* está activo y listo para el funcionamiento. Este paso es esencial para la comunicación con dispositivos que emplean *Bluetooth Low Energy* (BLE).

Activación del entorno virtual y dependencias
La siguiente parte del script activa el entorno virtual de Python mediante `source /home/juan/matter/connectedhomeip/`
Esta activación es crucial, ya que *chip-repl* y el script *control.py* dependen de un entorno controlado con las bibliotecas específicas instaladas.

Ejecución de *chip-repl* y *control.py*
La lógica del script esta diseñada para la ejecución condicional de *chip-repl*, , permitiendo al contenedor operar en dos modos distintos según cómo se lance el contenedor:

- Si la variable `MODE` se establece en "1" al iniciar el contenedor, se procede con el *commissioning* del dispositivo. Esto se logra mediante un bloque `EOF` que ejecuta la instrucción `%run control.py -c`, configurando el dispositivo para su primera conexión.
- Si la variable `MODE` no está configurada o tiene otro valor, el contenedor ejecuta *control.py* sin argumentos adicionales, iniciando el proceso de reconexión al dispositivo ya comisionado.

Esta estructura permite al contenedor realizar tanto la configuración inicial (*commissioning*) como la reconexión a un dispositivo previamente configurado, siguiendo la misma lógica planteada anteriormente, pero adaptada al entorno del contenedor.

Mantenimiento del contenedor en ejecución
Finalmente, el script emplea `tail -f /dev/null` para mantener el contenedor en ejecución de forma indefinida, evitando que se cierre una vez completados los procesos iniciales. Esto asegura que el código *control.py* se ejecute de manera continua mientras el contenedor esté activo.

De este modo, el contenedor queda configurado para ejecutar *control.py* de forma continua, garantizando una interacción automática, estable y activa entre el microcontrolador y la base de datos.

4.4.2 Contenedor web

El contenedor destinado a la aplicación web está definido en el archivo *Dockerfile_web*, diseñado para ejecutar *web.py* de manera independiente en un entorno controlado.

Imagen base
El contenedor se construye a partir de la imagen `python:3.9-slim`, una versión ligera de Python que proporciona un entorno ideal para aplicaciones basadas en Python, reduciendo el tamaño de la imagen y mejorando el rendimiento.

Instalación y configuraciones del contenedor
El contenedor comienza con la actualización de las dependencias del sistema y la instalación de `sqlite3`, imprescindible para la interacción con la base de datos local utilizada por la aplicación web.

Se define el directorio de trabajo `/home/juan/matter/connectedhomeip`. Este es el espacio que es el mismo que utiliza el ordenador local, sirve como punto central para la ejecución de los procesos y la ubicación de los archivos necesarios. A continuación, se copian al contenedor tanto *web.py*, que es el script principal que gestiona la aplicación web, como la carpeta *templates* que contiene *index.html*.

La instalación de *Flask* se realiza mediante la herramienta `pip` instalada en el contenedor, garantizando que el framework esté disponible para que *web.py* pueda responder a solicitudes y ejecutar el servidor web.

Exposición de puertos y ejecución
Se expone el puerto 5000 para permitir la comunicación externa con la aplicación web. Finalmente, el contenedor se configura para que al arrancar el contenedor, *web.py* se ejecute automáticamente y el servidor Flask se inicie, dejando la aplicación lista para procesar solicitudes.

Así, el contenedor queda preparado para alojar y gestionar la interfaz web de forma automatizada, facilitando la interacción continua con la base de datos.

4.4.3 Docker-compose

Para automatizar y gestionar la ejecución de los dos contenedores desarrollados, se ha implementado un archivo llamado *docker-compose.yml* que permite realizar un despliegue simultáneo de ambos contenedores *chip-repl* y *web* haciendo uso de la herramienta *docker-compose*.

Estructura de *docker-compose.yml*
El archivo *docker-compose.yml* se ha desarrollado siguiendo la versión 3.8 de *Docker Compose* y define dos servicios principales: *chip-repl* y *web*. A continuación, se explican sus configuraciones clave.

Servicio *chip-repl*
El servicio se construye utilizando el archivo *Dockerfile_chip-repl*, previamente explicado, que incluye la instalación de dependencias y la configuración necesaria para ejecutar *control.py* con *chip-repl*.

- **Construcción del contenedor**: El servicio se construye utilizando el archivo *Dockerfile_chip-repl*.
- **Privilegios y modo de red**: Se establece `privileged: true` y `network_mode: "host"` para garantizar acceso completo a la red del host y a los dispositivos de hardware, necesarios para la gestión de conexiones *Bluetooth* y otros servicios.
- **Volúmenes**: Se especifican volúmenes para compartir directorios y archivos clave entre el host y el contenedor:
  - `/var/run/dbus` para la comunicación con D-Bus.
  - `/home/juan/matter/connectedhomeip` para acceso al repositorio base y sus scripts.
  - `/home/juan/matter/connectedhomeip/persistent_tmp` para diposner almacenamiento temporal de las credenciales.
  - `/home/juan/bin` para acceder a binarios y scripts adicionales.
- **Dispositivos y capacidades**: Se incluyen dispositivos como `/dev/bus/usb` y `/dev/hci0` para permitir la interacción con el dongle usb bluetooth, junto con capacidades como `SYS_ADMIN` y `NET_ADMIN` para el manejo de funciones avanzadas del sistema.
- **Variables de entorno**: La variable `MODE` se utiliza para definir el modo de ejecución (por ejemplo, *commissioning* o reconexión), lo cual se gestiona mediante la configuración en el script.

Servicio *web* El servicio *web* está diseñado para ejecutar la interfaz web desarrollada en *Flask*. Las configuraciones principales de este servicio incluyen:

- **Construcción**: Utiliza el *Dockerfile_web* para construir la imagen del contenedor, como se detalló anteriormente.
- **Modo de red**: Configurado con `network_mode: "host"` para que la aplicación web pueda comunicarse sin restricciones con el sistema del host, facilitando el acceso a la base de datos y a otros servicios necesarios.
- **Comando de inicio**: Al iniciar, se ejecuta el script `web.py` para lanzar el servidor de *Flask*, permitiendo que la interfaz esté disponible para recibir solicitudes.
- **Volúmenes**: Se comparte el directorio `/home/juan/matter/connectedhomeip` para garantizar que la aplicación tenga acceso a los archivos y recursos del proyecto, incluida la base de datos *SQLite* y el archivo HTML de la interfaz.

Funcionamiento integrado
La configuración conjunta de los servicios en el archivo `docker-compose.yml` permite que , desde el directorio principal del repositorio que contiene todos los archivos del proyecto, se pueda compilar y ejecutar el sistema de manera coordinada. Para compilar los contenedores, se utiliza el comando mostrado en el código 4.17.

Código 4.17 Comando de compilación para *Docker-compose*.

```
docker-compose build
```

Una vez finalizada la compilación, el sistema puede ejecutarse utilizando el comando indicado en el código 5.1.

Código 4.18 Comando para ejecutar el sistema mediante *Docker-compose*.

```
(MODE=1) docker-compose up
```

De este modo, ambos contenedores se inician de forma coordinada. La variable `MODE=1` en la ejecución avisa al sistema que debe realizar el *commissioning* del dispositivo. Si no se indica, el sistema procede directamente a la reconexión del dispositivo. El contenedor `chip-repl` es responsable de gestionar el control y monitoreo del microcontrolador, mientras que el contenedor `web` ejecuta la interfaz web, proporcionando un entorno de gestión que permite al usuario interactuar con el sistema y controlar el dispositivo.

De esta forma finaliza el desarrollo de la solución propuesta, logrando automatizar todo el proceso de comunicación y gestión. Permitiendo una interacción directa entre el microcontrolador, la base de datos y la interfaz web, gracias un sistema integral que facilita el control, monitoreo y operación del dispositivo que se ejecuta de forma continua y optimizada.

4.5 Verificación y pruebas

En esta sección se procede a la evaluación y prueba del funcionamiento de la solución completa desarrollada. Es importante destacar que para garantizar el funcionamiento del sistema, son imprescindibles como mínimo tres elementos fundamentales: un ordenador que cuente con todas las herramientas y configuraciones descritas en apartados anteriores, que tenga capacidad de conexión vía Bluetooth y que soporte la ejecución de contenedores Docker; el microcontrolador ESP32-C6-DevKitM-1 provisto del firmware implementado; y un router que permita la comunicación del microcontrolador con el ordenador a través de la red Matter.

Para la obtención de los mensajes de log que provenientes del microcontrolador, se ha conectado el microcontrolador al ordenador durante las pruebas lo que ha permitido ver los registros de los mensajes generados. Adicionalmente, para garantizar la comunicación por Bluetooth durante el proceso de *commissioning*, se ha empleado un dongle USB Bluetooth. La configuración inicial del entorno de pruebas se ha planteado tal y como se ilustra en la figura 4.8.

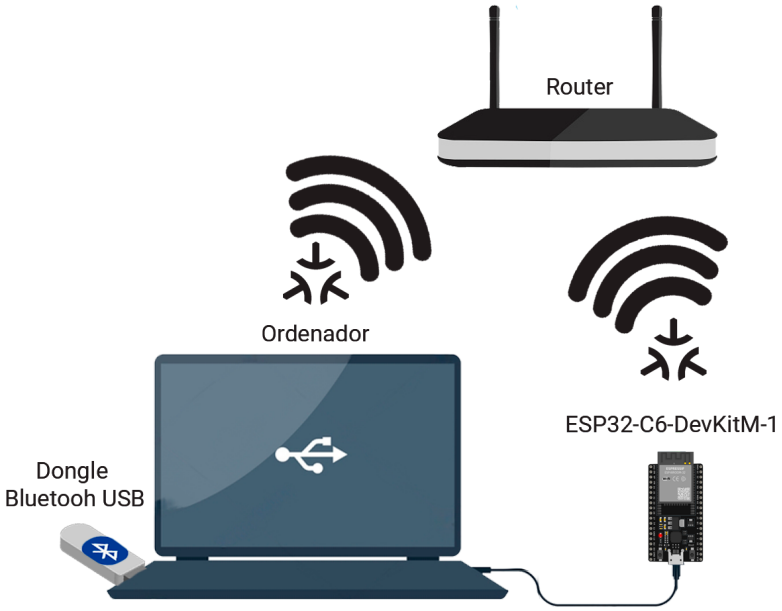


Figura 4.8 Esquema de dispositivos durante pruebas.

Funcionamiento sistema completo

Después de realizar varias pruebas iniciales y comprobar que los datos se reciben y los cambios de modo de operación se aplican de manera correcta, se decide llevar a cabo una demostración general. Esta demostración

integra todos los procedimientos y pruebas parciales de cada aspecto del sistema recogidos en un único ensayo que se ha documentando mediante un vídeo con el fin de mostrar una visión completa de la solución implementada y su comportamiento en tiempo real.

El video comienza mostrando la disposición descrita en la figura 4.8, se realiza una grabación de pantalla en la que se visualizan simultáneamente la interfaz web, la ejecución de los contenedores *Docker* mediante *Docker-compose* y los registros (*logs*) del microcontrolador, permitiendo un seguimiento detallado del funcionamiento de cada parte del proyecto.

En la primera fase de la demostración, se inicia el proceso de *commissioning* del dispositivo que tras finalizar con éxito, el sistema queda establecido por defecto en el modo normal tal y como se describe en la configuración inicial. Durante este tiempo, se deja que la base de datos, inicialmente vacía, comience a recibir y registrar los primeros atributos, aprovechando para mostrar algunas de las funcionalidades interactivas de la interfaz web.

A continuación, se procede a cambiar al modo operación de ahorro desde la interfaz web, observando los registros generados en el microcontrolador y en el sistema *Docker-compose*. Este modo se mantiene durante el tiempo suficiente para permitir la recopilación de datos y poder apreciar las variaciones en los gráficos de la interfaz web. Destaca como al cambiar de modo normal a ahorro, la corriente activa muestra una variación de valores mucho menos pronunciada, un cambio que también se refleja, aunque de manera más moderada, en la potencia activa. Esta últimar recordar que depende tanto del voltaje, que varía de forma aleatoria independientemente independientemente del modo de operación, como de la propia corriente activa que sí que depende de forma directa.

Después de un periodo de funcionamiento en modo de ahorro, se cambia al modo de rendimiento, documentando de nuevo todo el proceso y observando un aumento significativo en la variación de los valores de corriente activa. Este cambio es mucho más pronunciado que el de modo normal a modo ahorro, y se observa claramente en la tabla de los últimos 10 valores registrados. Se deja que el dispositivo opere en este modo durante un tiempo sufuciente como para registrar un conjunto de datos representativo.

En la segunda fase de la demostración, se simula una desconexión del microcontrolador al desconectarlo de la conexión serial USB al ordenador y por tanto de la fuente de alimentación, además se realiza un `docker-compose down` para detener los contenedores. En esta etapa, se sustituye la conexión USB, que proporcionaba los registros de *logs*, por una batería externa, demostrando que el microcontrolador puede funcionar de forma independiente y que no es necesaria su conexión directa al ordenador para la comunicación.

Desde el ordenador, se reinician los contenedores Docker (sin activar `MODE=1`, es decir, sin realizar un nuevo *commissioning*), y se comprueba cómo el dispositivo se reconecta de forma automática. Cabe mencionar que, al desconectar el dispositivo de la alimentación, aunque las credenciales necesarias para la reconexión se mantienen, los atributos y el modo de operación se reinician a los valores predeterminados debido a la naturaleza volátil de la memoria configurada en los *clusters* del dispositivo. Por tanto, el dispositivo se inicializa nuevamente en modo normal.

Finalmente, tras la reconexión, se vuelve a cambiar al modo de ahorro, mostrando las medidas en tiempo real para verificar el correcto funcionamiento del dispositivo. Con esto, se concluye la demostración y el video asociado.

El video completo se ha subido a YouTube y se puede visualizar haciendo clic la figura 4.9, que proporciona un acceso directo al contenido o bien accediendo desde el siguiente enlace: *https://www.youtube.com/watch?v=UagbwvF-_o0*.

La prueba completa, mostrada en el video, confirma que el planteamiento desarrollado en el proyecto funciona de manera correcta. Se observa la interacción entre el microcontrolador, la base de datos y la interfaz web, desde el *commissioning* inicial hasta la gestión de modos y la reconexión automática del dispositivo. Con estos resultados, se puede validar la operatividad del sistema y se cumplen las metas propuestas, concluyendo satisfactoriamente el desarrollo y la verificación de la solución.

Figura 4.9 Enlace al video demostrativo de la solución implementada.

5 Discusión, conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones obtenidas después del trabajo de investigación, desarrollo y demostraciones realizadas en el proyecto, que tenía como finalidad el estudio y la aplicación del estándar de conectividad del hogar Matter. Como eje central del proyecto, se ha implementado de forma práctica un nodo simulador del consumo energético utilizando un microcontrolador ESP32C6-DevKitM-1 junto con los elementos de conectividad necesarios para su funcionamiento en el ecosistema Matter.

A continuación, se evalúan los objetivos alcanzados, las dificultades técnicas encontradas en el desarrollo del proyecto, las limitaciones del proyecto y posibles trabajos futuros. Finalmente, se expondrá una conclusión global donde se comentan los aprendizajes obtenidos y el impacto del proyecto.

5.1 Evaluación de los objetivos alcanzados

A continuación, se presentan los logros alcanzados en relación con los objetivos específicos establecidos al inicio del proyecto:

Análisis del estándar Matter

El objetivo basado en examinar el estándar Matter, así como la arquitectura de este, se ha cumplido en la medida de los objetivos propuestos. Se ha realizado un análisis de la última versión, la 1.3, de dicho estándar, observando sus características generales y cómo interaccionan dentro del ecosistema IoT. A lo largo del análisis del estándar Matter, se han ido identificando y explicado las partes más importantes de su arquitectura y, en especial, de aquellos elementos que han sido necesarios para integrar en el proyecto de nodo sensor. A pesar de que el estándar tenga continuos avances y actualizaciones, se ha documentado el estado actual, teniendo en cuenta que algunos aspectos empleados en el proyecto podrían recibir modificaciones en un futuro próximo, conforme a las advertencias de la Connectivity Standards Alliance (CSA). ahora bien, la base expuesta es robusta y permite comprender y aplicar correctamente el estándar.

Desarrollo de la aplicación para el microcontrolador ESP32-C6-DevKitM-1

El propósito de desarrollar una aplicación en C++ que simule un nodo de consumo energético compatible con Matter también se ha cumplido, ya que se ha desarrollado la aplicación *Electrical Sensor App*, diseñada y construida específicamente para el microcontrolador ESP32-C6-DevKitM-1. Esta aplicación incluye, además, la posibilidad de modificar el modo de operación que tiene configurado el dispositivo en tiempo real.

Validación con herramientas nativas

Se ha realizado un uso permanente de herramientas nativas durante el desarrollo del proyecto; *chip-repl* se ha utilizado como la herramienta software principal para interactuar con el microcontrolador, comprobando funcionalidades implementadas, en combinación de otras herramientas y compiladores nativos del proyecto Matter en conjunto con las herramientas del proyecto Espressif. Las pruebas ejecutadas con el sistema indican que este opera sin errores significativos cumpliendo con los criterios de comunicación del estándar para realizar lectura, escritura y modificación de atributos.

Desarrollo de la interfaz web

Otro objetivo alcanzado ha sido la creación de una interfaz web que visualice y permita la modificación de los

datos de consumo energético. La interfaz desarrollada ofrece un registro de los últimos 10 valores recibidos y presenta de forma clara y presenta de forma clara tanto los valores actuales como históricos de los atributos. Además, permite al usuario cambiar de modo de operación de forma instantánea y recibir una respuesta inmediata del dispositivo. Estos aspectos se han demostrado de manera efectiva en el video demostrativo.

Automatización del sistema mediante Docker
La automatización del sistema mediante *Docker* funciona satisfactoriamente. Se han configurado dos contenedores: uno para la ejecución del código de control de *chip-repl* y otro para la interfaz web, ambos están conectados a través de la base de datos central. Se ha demostrado que ambos contenedores interactúan entre ellos de forma eficaz y funcionan de manera coordinada, facilitando la implementación de la solución y permitiendo activar todo el sistema con un solo comando de lanzamiento.

Base para futuros proyectos
El proyecto ha sentado una base sólida para futuros desarrollos. La integración de tecnologías y herramientas como Matter, *Docker* y el microcontrolador ESP32-C6-DevKitM-1, junto con la estructura modular de la solución, abre la posibilidad de implementar nuevas funcionalidades o desarrollar proyectos en un ámbito similar. Además, el enfoque adoptado durante el proyecto ha sido guiado por la empresa Woodswallow, lo que ha permitido centrarlo en áreas de interés industrial y potenciar su desarrollo a largo plazo.

En resumen, la mayoría de los objetivos específicos se han cumplido de manera satisfactoria. El proyecto ha demostrado ser una solución adecuada y conforme al estándar Matter, con un alto potencial para ser optimizado y adaptado a otros dispositivos en el futuro.

5.2 Dificultades técnicas encontradas

Durante el desarrollo del proyecto, se han presentado varias dificultades técnicas que se describen a continuación:

5.2.1 Fallo en la comunicación bluetooth

En las primeras etapas del proyecto, al comprobar el *commissioning* del dispositivo mediante BLE (Bluetooth Low Energy), surgieron problemas relacionados con el uso de la máquina virtual. Aunque mi ordenador personal anfitrión cuenta con un adaptador Bluetooth integrado, este se bloqueaba al ser utilizado simultáneamente por el sistema operativo original con windows y la máquina virtual con Ubuntu. Para resolver esto, se optó por emplear un dongle USB Bluetooth dedicado exclusivamente a la máquina virtual, lo que solucionó el problema durante las pruebas preliminares.

Sin embargo, al dockerizar el proyecto, la configuración del Bluetooth en Docker se complicó debido a la coexistencia de múltiples adaptadores Bluetooth (el nativo del sistema y el dongle USB) y la naturaleza de la máquina virtual. La solución a este problema se llevo a cabo mediante la ejecución de los siguientes comandos antes de iniciar los contenedores de Docker:

Código 5.1 Comandos para desactivar el Bluetooth en la máquina virtual.

```
sudo systemctl stop bluetooth
sudo systemctl disable bluetooth
```

Estos comandos desactivan el servicio de Bluetooth en la máquina virtual, permitiendo que Docker maneje exclusivamente el adaptador USB. Este ajuste no se ha incluido en la solución final documentada, ya que es un problema específico de las condiciones de mi entorno de trabajo y probablemente no afectará a otros usuarios con configuraciones diferentes. Sin embargo, se menciona aquí para advertir sobre el problema.

5.2.2 Uso del *cluster* de *Energy Preference*

Otra dificultad surgió al implementar la funcionalidad de establecer y modificar el modo de operación del dispositivo. Inicialmente, no existía un *cluster* o atributo específico en el estándar para gestionar modos de consumo energético. Esto llevó a utilizar el *cluster* `Level Control` como una solución temporal, aunque

era una implementación incorrecta.

Durante el desarrollo del proyecto, se introdujo una nueva actualización del repositorio con nuevos *clusters* disponibles que permitiendo el uso del *cluster* `Energy Preference`, con el atributo `Current Energy Balance`, el cual es adecuado y está especificado en el estándar para gestionar modos de operación energética. Esta actualización permitió que la implementación fuera más precisa y alineada con las especificaciones de Matter.

5.3 Perspectivas y trabajos Futuros

Este proyecto cumple con las expectativas planteadas, pero también ofrece oportunidades para realizar mejoras que podrían elevar su funcionalidad a un nivel superior. A continuación se exponen algunos de los posibles desarrollos futuros y mejoras que podrían implementarse para aumentar las capacidades del proyecto.

5.3.1 Inclusión de tecnología Thread

La incorporación de la tecnología Thread al proyecto es una posibilidad que se había contemplado inicialmente; sin embargo, desde la empresa se recomendó priorizar otros aspectos del proyecto debido a que, en este momento, la implementación de Thread presenta ciertos desafíos complejos.

Matter está diseñado para funcionar de forma óptima sobre redes basadas en Thread, un protocolo de red inalámbrico de baja potencia y topología de malla que garantiza una conectividad sólida, escalable y segura para dispositivos IoT. Integrar Thread permitiría que el nodo sensor desarrollado se conectase de manera más eficiente y formase parte de una red de malla autorreparable, lo que aumentaría la seguridad y la cobertura de la red.

Sin embargo, la inclusión de Thread significaría modificar el firmware del microcontrolador con el objetivo de adaptar la arquitectura de comunicación y poder hacerla compatible con esta tecnología. También habría que modificar la aplicación *Electrical Sensor App* para incorporar el nuevo protocolo y probablemente habría que revisar la infraestructura de red del proyecto adaptando la comunicación al nuevo sistema de comunicación basado en Thread. Aunque la implementación de Thread presenta ciertos elementos con una complejidad técnica alta, se podrían extraer importantes ventajas energéticas, de fiabilidad y escalabilidad de la red.

5.3.2 Contenedor docker específico

Otra de las mejoras que se podrían implementar en el proyecto, sería recurrir una imagen Docker específica y ligera que solo incluya los elementos necesarios de *connectedhomeip* para la comunicación con el microcontrolador. Sin embargo, como el proyecto de *connectedhomeip* se encuentra en continua fase de desarrollo, actualmente no existe una imagen de Docker que contenga todas las herramientas en la versión más estable. Algunas funcionalidades de *chip-repl* y otras herramientas no están completamente integradas requieren de ciertas configuraciones que no están desarrolladas.

Por esta razón, en este proyecto se ha utilizado una solución intermedia en la que se recurre al repositorio *connectedhomeip* por completo y de ese modo se garantiza el acceso a todas las herramientas y funcionalidades necesarias. Aun así, una mejora interesante de cara al futuro del propio proyecto, sería el uso de una imagen Docker especializada con la que evitar la necesidad de tener que descargar el repositorio completo y optimizar el proceso de comunicación desde el propio contenedor. Así, se ganaría en eficiencia a la hora de realizar el despliegue del sistema, optimizando el peso de la imagen y facilitando la integración de la solución.

5.4 Conclusiones generales

El desarrollo del presente proyecto ha permitido realizar un análisis detallado y el desarrollo de una aplicación práctica del estándar Matter en un entorno experimental, poniendo de manifiesto su potencial en la creación de redes IoT para hogares conectados. A lo largo de este proceso, se ha cumplido con los objetivos específicos

planteados, llegando a desarrollar un nodo sensor de consumo energético que junto con una infraestructura de soporte desarrollada, permite al usuario realizar una monitorización continua del mismo e interactuar con el dispositivo de manera interactiva y cómoda.

La colaboración con la empresa Woodswallow, promovida por la cátedra de IoT, ha sido fundamental en el desarrollo de este proyecto ya que su apoyo tanto a nivel técnico como organizativo ha permitido organizar el proyecto de manera eficiente, siguiendo un enfoque adecuado. Además, la experiencia que acumulan dentro del sector de sistemas embebidos junto con un la retroalimentación semanal recibida, ha sido fundamentales para realizar ajustes y mejoras en el desarrollo.

En conclusión, los resultados obtenidos en este proyecto destacan el papel prometedor de Matter en el horizonte a corto plazo de los hogares conectados, demostrando su viabilidad y su potencial en redes inteligentes de gestión energética.

Índice de Figuras

Índice de Códigos

Bibliografía

[1] Connectivity Standards Alliance, *Matter 1.3 core specification*, *https://csa-iot.org/developer-resource/specifications-download-request/*, 2024, Accedido el 09 de octubre de 2024.

[2] _____, *Matter 1.3 matter application cluster specification*, *https://csa-iot.org/developer-resource/specifications-download-request/*, 2024, Accedido el 09 de octubre de 2024.

[3] _____, *Matter 1.3 matter device library specification*, *https://csa-iot.org/developer-resource/specifications-download-request/*, 2024, Accedido el 09 de octubre de 2024.

[4] Connectivity Standarts Alliance, *Connectedhomeip*, *https://github.com/project-chip/connectedhomeip*, 2024, Accedido el 09 de octubre de 2024.

[5] _____, *Members*, *https://csa-iot.org/members/*, 2024, Accedido el 09 de octubre de 2024.

[6] Universidad de Sevilla, *Web cátedra de iot*, *https://catedra.us.es/catedraiot/*, 2024, Accedido el 03 de octubre de 2024.

[7] Google Home Developers, *Device data model*, *https://developers.home.google.com/matter/primer/device-data-model#device-types*, 2024, Accedido el 09 de octubre de 2024.

[8] _____, *Matter primer - what is matter?*, *https://developers.home.google.com/matter/primer*, 2024, Accedido el 07 de octubre de 2024.

[9] Espressif, *Thread border router in matter*, *https://developer.espressif.com/blog/matter-thread-border-router-in-matter/*, 2024, Accedido el 09 de octubre de 2024.

[10] Leonardo González, Osiris Sofía, Daniel Laguía, Esteban Gesto, and Karim Hallar, *Internet del futuro-estudio de tecnologías iot*, *https://publicaciones.unpa.edu.ar/index.php/ICTUNPA/article/view/744*, 2024, Accedido el 07 de octubre de 2024.

[11] IBM, *What is the internet of things (iot)?*, *https://www.ibm.com/topics/internet-of-things*, 2024, Accedido el 05 de octubre de 2024.

[12] Goto IoT, *Protocolos iot comparativa*, *https://www.gotoiot.com/pages/articles/iot_protocols_intro/index.html*, 2024, Accedido el 07 de octubre de 2024.

[13] Aula Software Libre, *¿qué es docker?*, *https://aulasoftwarelibre.github.io/taller-de-pas/Sesion-1/Introducci%C3%B3n/*, 2024, Accedido el 09 de octubre de 2024.

[14] Christopher Loreck, *Innovation through standardization in smart home ecosystems*, *https://www.edit.fis.uni-hamburg.de/ws/files/54825461/EURAS_2024_Loreck_Matter.pdf*, 2024, Accedido el 09 de octubre de 2024.

[15] Statista, *Iot connected devices installed base worldwide from 2015 to 2025*, *https://www.statista.com/statistics/512673/worldwide-internet-of-things-market/*, 2024, Accedido el 07 de octubre de 2024.

[16] Espressif Systems, *Esp32-c6-mini-1(u) datasheet version 1.2*, *https://www.espressif.com/sites/default/files/documentation/esp32-c6-mini-1_mini-1u_datasheet_en.pdf*, 2024, Accedido el 09 de octubre de 2024.

[17] _____, *Espressif iot development framework*, *https://github.com/espressif/esp-idf*, 2024, Accedido el 09 de octubre de 2024.

[18] Woodswallow, *Web woodswallow*, *https://woodswallow.tech/es/*, 2024, Accedido el 03 de octubre de 2024.