Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA TELEMÁTICA

Trabajo Fin de Grado

Programación de controlador de dispositivos IoT para casa inteligente en una Raspberry Pi 4 a partir de un array de consumo establecido en rangos de tiempo.

Autor: Alberto Corral Ortiz

Tutor: Esther Palomar González

2022

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Programación de controlador de dispositivos IoT para casa inteligente en una Raspberry Pi 4 a partir de un array de consumo establecido en rangos de tiempo.

Autor: Alberto Corral Ortiz

Tutora: Esther Palomar González

TRIBUNAL:

Presidente: Álvaro Hernández Alonso

Vocal 1º: Ernesto Martín Gorostiza

Vocal 2º: Esther Palomar González

FECHA: 21 de septiembre de 2022

‘‘Quiero agradecer a mis padres, pareja, familia, profesores de la universidad y a mi tutora, por haberme apoyado de una u otra forma para que este camino que empecé hace tiempo llegase a buen término. También quisiera dar las gracias a mi tutora por haber confiado en mí y por el tiempo dedicado. Recordar y agradecer a las comunidades de software libre que colaboran y ayudan a mantener y dar soporte a los productos sin esperar nada a cambio. Sin duda alguna, no habría llegado este día sin vuestro apoyo. Por último, este TFG quiero dedicárselo a mis hijos, por el tiempo y apoyo prestado, y por la motivación que me suponen; para que esta etapa compartida os anime a luchar por vuestros sueños. ¡Gracias a todos, y a Google! ;-)”

Índice de figuras e ilustraciones

Resumen

Palabras clave: Internet-of-Things, ZigBee, Z-Wave, RaspberryPi, Programación Horaria

En este Trabajo Fin de Grado (TFG) se implementa una solución de software que controla el funcionamiento de enchufes y dispositivos conectados en una casa inteligente. La solución se ejecuta en una RaspberryPi 4 que recibe como entrada un vector por cada elemento de control mediante una API REST; este vector contiene la información relativa al funcionamiento de cada enchufe o dispositivo en las 24 horas de un día, con los que se gestionará, de forma autónoma, el estado encendido (activación) o apagado (desactivación) de los correspondientes dispositivos. Por tanto, los objetivos de este proyecto buscan:

- Facilitar la gestión energética en viviendas mediante la gestión de la programación horaria acorde a las necesidades de los usuarios.
- Facilitar a un usuario el encendido y apagado, o activación y desactivación de elementos controlables mediante los protocolos de comunicación Z-Wave y Zigbee.

Este TFG se enmarca en el proyecto de investigación ENEFF-PILOT del Dpto. de Electrónica de la Universidad de Alcalá.

Summary

Keywords: Internet-of-Things, ZigBee, Z-Wave, RaspberryPi, Time Scheduling

This Final Year Project implements a software solution to control the functioning of a smart home's connected plugs and devices. The solution executes on a Raspberry Pi 4 and, through an API REST, receives a vector as input with the time schedule of every controllable device; this vector contains the information to activate/deactivate every device in hourly slots within the day. Hence, this project seeks:

- Facilitating the energy management in smart homes using a time scheduling software solution that considers the consumer's preferences.
- Enabling the remote control on appliance activation through Z-Wave and ZigBee communication protocols.

This work is conducted within the ENEFF-PILOT research project at the Electronics Department in University of Alcala.

Glosario de acrónimos y abreviaturas

CLI	Interfaz de Línea de Comandos (*Command Line Interface*)
GUI	Interfaz Gráfica de Usuario (*Graphical User Interface*)
HA	Home Assistant
HEMS	Sistema de Gestión de Energía del Hogar (*Home Energy Management System)*
IoT	Internet de las Cosas (*Internet of Things*)
JNI	Interfaz Nativa de Java (Java Native Interface)
M2M	Máquina-máquina
OpenHAB	Bus Domótico Abierto (open Home Automation Bus)
OH	OpenHAB
OSGi	Arquitectura abierta de servicio de pasarelas (*Open Services Gateway Initative*)
RPI	Raspberry Pi
SDK	Kits de desarrollo software (*Software Development Kit*)
SO	Sistema Operativo
TFG	Trabajo Fin de Grado
VNF	Virtual Network Function (*funcionalidad de las redes GSM 5G*)
ZB	ZigBee
ZW	Z-Wave

1 Introducción

En la sociedad actual "de la información", en la que tienen lugar fuertes dinámicas que promueven lo que se ha denominado como la "digitalización", la Domótica1 (también conocida como *Home Automation*) está emergiendo de nuevo con gran fuerza debido a dos factores fundamentales. El primero es que la tecnología necesaria es asequible para una gran cantidad de consumidores, la cual se ha democratizado y está sujeta a una mercantilización en gran parte debido a la omnipresente conexión a Internet que nos brindan tecnologías como 3G/4G y la recién llegada 5G, la cual ha incluido en su diseño el soporte para la Internet-de-las-cosas (IoT, del inglés, Internet-of-Things) (Almomani et Rahman, 2022).

En segundo lugar, se está aceptando cada vez más la adopción de medidas urgentes para que la presencia del hombre en la tierra sea sostenible, y es por ello que la domótica se presenta como un mecanismo indispensable que nos permite:

- Medir el consumo energético para su optimización y control (Chavali et al., 2014).
- Actuar sobre los elementos de consumo energético, de tal forma que podamos operar sobre ellos cuando es más conveniente (Zhou et al., 2016).
- Actuar sobre la demanda con antelación, y así facilitar servicios de balanceo de oferta y demanda eléctrica a través de su planificación (Sowah et al., 2018).
- Avanzar en generación distribuida como las que nos brindan la energía solar o la eólica (Han et al., 2014).

Existe un tercer factor que ha provocado la aceleración de este proceso en lo que se refiere a la adopción de tecnología digital, y que ha afectado también a la Domótica. Dicha adopción se ha visto acelerada en los hogares a causa del COVID-19, ya que hemos pasado más tiempo en casa, y puesto mayor foco en el confort de nuestros hogares. De esta forma, los consumidores comienzan a adoptar con mayor facilidad los llamados sistemas de gestión de energía en casa (Chen et al., 2021), como el mostrado en la Figura 1.

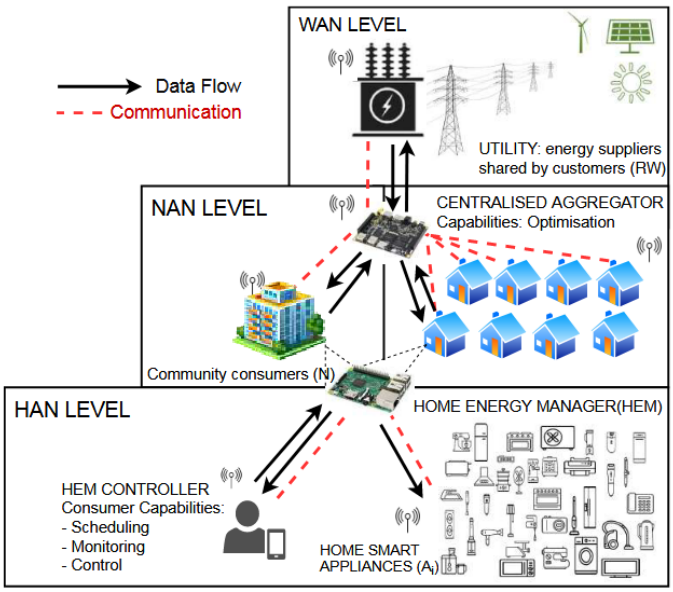

Figura 1 Ejemplo HEMS obtenido del proyecto ENEFF-PILOT (Eneff-pilot, 2022).

[1]«Domótica, puede definirse como el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema.» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Dom%C3%B3tica.

En el campo de la Domótica, considerado un caso específico o de aplicación de la tecnologia IoT en hogares, existen numerosos fabricantes de hardware (HW) y de software (SW), que deben poder interoperar entre sí. Como es común en otros dominios, también aquí se han creado estándares y plataformas de las telecomunicaciones, tanto basados en fuentes abiertas como privativas, como específicas o generalistas, que soportan dicha necesidad. Algunos ejemplos de ello son: Bluetooth (Bluetooth, 2022), Z-Wave (ZWave, 2022), ZigBee (Zigbee, 2022), Wi-Fi, KNX (KNX, 2022), X10 (X10, 2022), Home Assistant (Home Assistant, 2022), OpenHAB (Openhab, 2022), ioBroker (ioBroker, 2022), entre otros.

Es importante reseñar también que, en el campo de la Domótica, el medio de transmisión es de suma importancia, ya que los dispositivos a controlar no siempre tienen alimentación externa, y se ven sujetos a la necesidad de emplear medios y protocolos de muy bajo consumo, pero sin limitar por ello las funcionalidades que se espera de un "dispositivo inteligente".

Tal y cómo se ha comentado anteriormente, la necesidad de crear sistemas que puedan integrarse entre ellos de forma sencilla, cumpliendo siempre ciertos requerimientos, sigue vigente, pero no siempre es algo sencillo de lograr. Por esta razón, los requerimientos pueden estar más relacionados con todos o parte de los siguientes factores:

- que la tecnología empleada sea de código abierto,
- que su coste sea mínimo,
- que pueda ser embebida en HW específico,
- la facilidad de integración para un usuario sin conocimientos específicos, o
- que disponga de una serie de APIs (*Application Program Interface*) que faciliten y garanticen la interoperabilidad con sistemas de terceros y entre distintos fabricantes.

Teniendo todo lo anterior en cuenta, cabe destacar uno de los proyectos de código abierto existentes que más auge está teniendo desde un punto de vista de integración con SW y HW de terceros, soporte a diferentes protocolos, y desde un punto de vista de creación de ecosistema con una comunidad técnica apoyada por empresas y particulares. Aunque inicialmente, en el año 2019 se trataba del proyecto Eclipse SmartHome, finalmente fue OpenHAB (su implementación de referencia) el que acabó prevaleciendo. El proyecto inicial fue amparado por la Eclipse Foundation pensado para su ejecución en dispositivos como Raspberry Pi, BeagleBone Black, o Intel Edison, aunque al estar desarrollado sobre Java 8 con soporte para OSGi (4.2+) está preparado o es fácilmente adaptable para su potencial ejecución en otros dispositivos y/o arquitecturas. El proyecto mantiene aún a día de hoy, año 2022, en su rama de la versión 3 una licencia Eclipse Public License 2.0, por lo que es posible la creación de productos comerciales basados en OpenHAB como base para otros desarrollos. Al comienzo del proyecto, la versión de OpenHAB más actual era 2.5, y debido a la duración de este, la actual versión es la 3.3, aunque la implementación del presente TFG está basada en la 3.1. La implementación base de OpenHAB aporta herramientas en las que apoyarse para extender la funcionalidad, de tal forma que se pueden crear nuevos módulos con las funcionalidades deseadas o utilizar las funcionalidades existentes (lenguajes de scripting) para adaptarlas a las necesidades del proyecto.

El presente proyecto estudia, implementa y valida la instalación y configuración de un sistema de gestión de energía (del hogar, en el escenario principal) en una plataforma RaspberryPi usando el software OpenHAB. El sistema implementa un controlador de los dispositivos conectados en casa (p. ej. lavadora, secadora, aire acondicionado, horno, riego automático, vehículo eléctrico) de tal forma que permita controlar el apagado y encendido de los mismos. Para ello, el controlador recibirá un vector con la programación horaria (23-25 periodos) que almacenará y utilizará para la gestión del funcionamiento de los dispositivos. La validación del sistema se efectúa en una maqueta de una casa (véase Figura 2) con enchufes inteligentes conectados por Z-Wave y ZigBee. Estos enchufes podrían convertir dispositivos "no inteligentes" a un estado de conexión con el controlador domótico. El sistema se explica en mayor detalle en los siguientes apartados.

Figura 2 Fotografía y esquema de la maqueta montada para validar el sistema controlador.

1.1 Motivación y objetivos

En primer lugar, he de indicar que tuve una motivación personal para solicitar la realización de esta propuesta de TFG. Dicha motivación surge de mi interés por la domótica, por las tecnologías de código abierto, y por la integración de sistemas. Un campo al que pude dedicarme en mis primeros años de experiencia laboral como desarrollador.

Este TFG se enmarca en el proyecto ENEFF-PILOT[2] del Dpto. de Electrónica en la UAH; de tal forma que es una parte de un sistema mayor. Concretamente consiste en la implementación de un módulo de software que se ejecute en una RPI 4, y que recibe como entrada un vector por cada elemento de control con información relativa a los 24 periodos horarios de un día, con los que gestionará de forma autónoma el encendido (activación) o apagado (desactivación) de los correspondientes elementos o dispositivos IoT.

El principal campo de aplicación de esta herramienta se plantea como un módulo extensor de una plataforma más amplia (Eclipse SmartHome / OpenHAB), siendo el módulo desarrollado en este TFG el que permita operar (activar/desactivar) sobre dispositivos que soportan los protocolos Z-Wave y ZigBee. Con ello, se pretende lograr la funcionalidad siguiente:

1 Facilitar la gestión energética de un edificio, o casa mediante la gestión de programaciones horarias.
2 Facilitar a un usuario el encendido y apagado, o activación y desactivación de elementos controlables mediante Z-Wave y ZigBee.

[2] Proyecto ENEFF-PILOT con fondos de la Atracción de Talento de la Comunidad de Madrid 2017. Coordinadora: Esther Palomar. https://eneffpilot.web.uah.es/

Adicionalmente, en el diseño de la solución, se ha tratado de facilitar que el resultado final pudiera ser lo más reutilizable, extensible, y escalable posible.

De modo que los objetivos de este TFG son:

1. Estudio y análisis de la integración de diferentes tecnologías de comunicación en dispositivos IoT, mediante el uso de tecnologías abiertas, y dispositivos que emplean diferentes canales de comunicación y protocolos.
2. Diseño, implementación y configuración de un sistema de gestión de energía en casa que permita el control (encendido/apagado) de dispositivos conectados.
3. Validación y prueba de escenarios del sistema sobre una maqueta demostrador.

En el siguiente apartado se detalla la metodología seguida para la consecución de los 3 objetivos de este TFG.

1.2 Metodología

La metodología empleada para el desarrollo del proyecto consta de una parte teórica y otra práctica, ambas desarrolladas simultáneamente, en un ciclo ágil de iteraciones que han permitido ir avanzando en las diferentes fases. Esto ayudó a resolver la principal incógnita que se planteaba al comienzo, consistente en demostrar si sería posible configurar un sistema capaz de hacer funcionar conjuntamente todos los elementos de partida, junto con los requerimientos HW y SW inicialmente establecidos en el anteproyecto.

El plan inicial se describe a continuación, aunque durante el desarrollo del proyecto, se detectó que no era necesario programar módulos específicos para completar los objetivos, como se comentará más adelante. Principalmente, las tareas previstas de programación se han podido resolver mediante una mezcla de investigación práctica, búsqueda de información en foros de Internet, conocimientos en administración de sistemas, configuración y una pequeña parte de programación lógica basada en DSL para la creación de reglas en OpenHAB. Ello ha supuesto que finalmente se ha podido montar el sistema con módulos de código abierto existentes; sin embargo, las tareas de administración de sistemas han sido mayores de las inicialmente estimadas. El desglose general de las tareas finalmente realizadas es como sigue:

A. Estudio de Requerimientos: Se realiza una investigación de las tecnologías de comunicación actuales de los dispositivos conectados en casa con más relevancia en el mercado y en publicaciones científicas. La consecución del Objetivo 1 establece los siguientes resultados:
   - A.1. Definición de requisitos funcionales.
   - A.2. Definición de requisitos no funcionales.

B. Análisis y diseño del sistema controlador del funcionamiento de aparatos conectados en casa. El Objetivo 2 requiere de las siguientes fases iniciales:
   - B.1. Análisis de las tecnologías de comunicación a utilizar: Z-Wave y ZigBee.
   - B.2. Análisis y elección del sistema operativo (SO) que se utilizará en la Raspberry PI 3 (Ubuntu).
   - B.3. Diseño a alto nivel de clases y de las estructuras de datos necesarias.
   - B.4. Montaje de la maqueta e identificación de los dispositivos y adaptadores HW/enchufes conectados.

C. Implementación y validación de prototipos del sistema controlador. Se realizan varias iteraciones de los procesos de implementación y validación. Los Objetivos 2 y 3 se llegan a alcanzar conjuntamente dado que alguna prueba de validación ha necesitado la re-configuración del sistema controlador.
   - C.1. Configuración del SO/Framework para permitir operar con los adaptadores HW para el acceso al medio.
   - C.2. Módulo de ejecución de tareas de activación y desactivación.
   - C.3. Módulo de recepción de vector de programación con MQTT.

C.4. Creación de pruebas unitarias para los módulos (librerías JUnnit o unnittest).

D. Documentación y presentación.

D.1. Documentación del proceso de instalación y configuración.

D.2. Documentación de usuario (para desarrolladores).

D.3. Preparación de la presentación del TFG.

En cuanto a los recursos utilizados para completar este TFG, el proyecto ENEFF-PILOT ha provisto del material y componentes necesarios para fabricar la maqueta. El entorno de desarrollo es código abierto y gratuito.

1.3 Trabajos relacionados

Los paradigmas de casa inteligente y eficiente energéticamente junto con la IoT han desplegado diversas aplicaciones de fácil acceso para el consumidor de electricidad. En la última década, los conocidos como sistemas de gestión de energía han atraído la atención de la comunidad investigadora y el mercado y existen numerosos prototipos y productos que ayudan al consumidor a monitorizar su consumo y/o planificar su demanda. Por ejemplo, los sistemas presentados en (Baig et al., 2013; Ahmed et al., 2015) validan la información proveniente de enchufes inteligentes con tecnología ZigBee y encuentran resultados de alta precisión en la señal monitorizada. Concluyen que, además de su diseño sencillo, bajo coste y consumo, estos enchufes proporcionan un control preciso del apagado y encendido de los dispositivos enlazados a un sistema de gestión de la energía en casa.

La planificación de consumo a partir de una demanda estimada está siendo considerada una de las medidas más efectivas para combatir los picos de consumo y permitir mayor aprovechamiento de fuentes renovables (Zhou et al., 2016). A estos esquemas se les conoce como Respuesta de Demanda y se basan principalmente en algoritmos de optimización de la demanda de energía (Chavali et al., 2014; Ozturk et al., 2013, Zhao et al., 2013). En este TFG, el algoritmo de planificación es un módulo independiente (trabajo desarrollado dentro del proyecto ENEFF-PILOT) cuyo resultado se recibe como input para el funcionamiento de nuestro sistema controlador de los dispositivos conectados en casa.

Productos de mercado que controlen el funcionamiento de estos dispositivos en casa hay varios. Muchos de fabricantes conocidos como Google Home distribuyen sistemas propios y otros usan código abierto como el entorno OpenHAB (Open Home Automation Bus) que facilita la interoperabilidad de dispositivos (Parocha et al., 2019, Sowah et al., 2020). Este entorno es un código abierto escrito en Java para automatización de hogares, independiente de tecnologías domóticas y basada en las especificaciones descritas según el modelo OSGI (Open Source Gateway Initiative). OpenHAB se comunica electrónicamente con dispositivos inteligentes y dispositivos no tan inteligentes, realizando actividades o acciones definidas por el usuario y proporcionando páginas web con información definida por el usuario, así como varias herramientas escogidas o definidas por el usuario para poder interactuar con todos los dispositivos (OpenHAB, 2022).

1.4 Estructura del documento

La estructura de este documento y sus diferentes apartados se describen a continuación:

- Capítulo 2. Diseño del controlador.

  Tras describir los requerimientos, el capítulo B explica todas las decisiones tomadas, junto con sus limitaciones, ventajas e inconvenientes. El apartado Decisiones de diseño incluye un breve análisis de las plataformas existentes actuales, para justificar las decisiones tomadas en cuanto a la implementación de la funcionalidad y el uso de software preexistente.

Por otro lado, el análisis y estudio de las tecnologías de comunicación ZW y ZB empleados por los dispositivos a controlar, no se ha considerado parte del trabajo, debido a la gran literatura existente. Sin embargo, se incluye una breve descripción y una tabla comparativa con el fin de facilitar la comprensión de estos. Esto nos llevará a conocer mejor las limitaciones y posibles mejoras que pueden implementarse.

- Capítulo 3. Implementación.

  En este capítulo C, se detallan los pasos a seguir para la implementación, así como consideraciones relevantes a tener en cuenta durante la misma, que han servido para facilitar el trabajo, y que pueden ser de utilidad también en el caso de ampliación.

  Existen diferentes apartados para las diferentes partes de la implementación, según se trate de la parte de más alto nivel o de las capas de más bajo nivel, cómo el sistema operativo y su puesta en marcha.

- Capítulo 4. Experimentos y validación.

  El capítulo D describe las diferentes pruebas realizadas, los problemas encontrados y resueltos, y aquellos que finalmente se han dejado para futuros trabajos. Se explican también las consideraciones que otorgan validez a la solución final.

- Conclusiones y Anexos.

  Se incluye además, a modo de anexos, documentación más técnica y precisa que permita abordar futuros trabajos de mantenimiento, actualización de versiones, o ampliación de los trabajos, con mayor facilidad.

  El código fuente empleado, usuarios, claves, y demás material del proyecto se entregarán en la tarjeta SD del proyecto.

2 Diseño de un controlador de casa inteligente

En una primera fase de desarrollo del controlador, se analizaron las soluciones de código abierto, las plataformas y librerías existentes con más apoyo desde la comunidad desarrolladora y que podrían ser reutilizadas para lograr el objetivo del proyecto. Gracias al análisis de las diferentes plataformas y de las tecnologías de diseño pudimos conocer a-priori las limitaciones y posibles mejoras de posible implementación.

2.1 Casos de Uso

Se identifica un caso de uso principal (Figura 3) del sistema controlador para el control programado del encendido y apagado de elementos domóticos independientemente del fabricante. El controlador recibe la programación horaria utilizando API HTTP (Figura 4) para facilitar la comunicación M2M y con ello la integración con sistemas existentes y preparado para operar a través de internet.

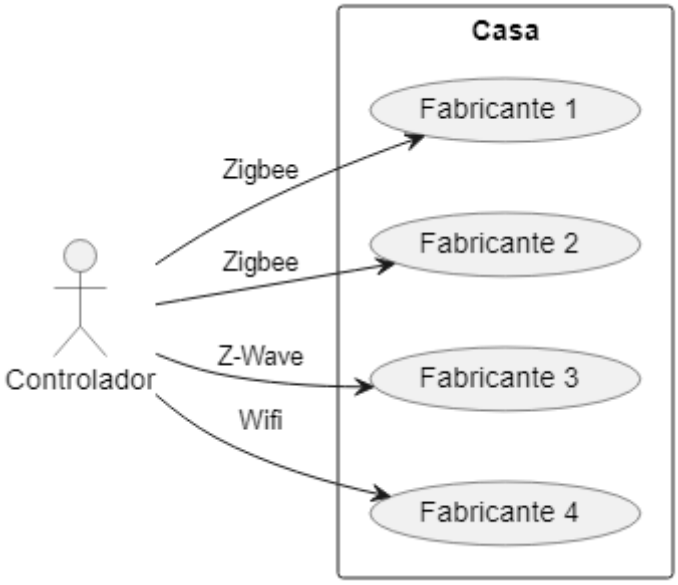

Figura 3 Caso de uso: Operaciones del controlador

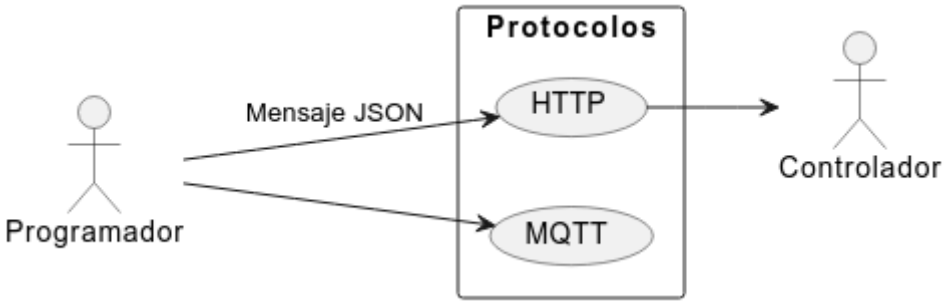

Figura 4 Caso de uso 2 protocolos de comunicación

El programador, módulo fuera del alcance de este TFG, hace un uso óptimo de la energía proveniente de fuentes renovables. Otros objetivos de optimización pueden buscar minimizar el coste de consumo.

2.2 Requisitos funcionales y no funcionales

A continuación, se detallan los requerimientos funcionales establecidos para la implementación SW del controlador:

RQ 01- Uso de software basado en fuentes abiertas.
RQ 02- Solución basada en sistema operativo Ubuntu.
RQ 03- Recepción de vector 24 periodos horarios usando HTTP / (MQTT).
RQ 04- Funcionamiento en hardware Raspberry Pi 4.
RQ 05- Capaz de operar con dispositivos basados en protocolo ZigBee.
RQ 06- Capaz de operar con dispositivos basados en protocolo Z-Wave.
RQ 07- (opcional) Capaz de operar con dispositivos basados en protocolo WiFi.

Los requerimientos no funcionales son los siguientes:

NF 01- Solución fácilmente ampliable a dispositivos de otros fabricantes.
NF 02- Solución fácilmente ampliable a nuevos protocolos y medios físicos (p. ej. el uso de Switches/Bombillas WiFi, o recepción de mensajes MQTT).
NF 03- Solución escalable y reproducible.
NF 04- Solución sencilla de mantener.
NF 05- Facilidad de instalación.
NF 06- Facilidad de realizar pruebas y virtualizar el entorno.

Se incluye una explicación sobre algunos de estos requerimientos por su relevancia. En concreto, RQ 03 trata de dotar de la máxima facilidad de integración con otros dispositivos. Inicialmente se quiere hacer la solución capaz de recibir programaciones horarias vía HTTP, pero con el foco puesto en la posibilidad de hacer también uso de MQTT, y así, seleccionar la plataforma/módulos que aportan mayor flexibilidad para cumplir con este requerimiento.

RQ 07 es un requisito exigente, y muy amplio, debido fundamentalmente a que la tecnología WiFi únicamente refiere a un medio de transmisión, pero no establece el protocolo de la capa de aplicación, al contrario de lo que hacen ZigBee y Z-Wave. Por lo tanto, para garantizar la máxima compatibilidad con dispositivos WiFi, los cuales podrían estar basados en protocolos cerrados, o no estándar, se postula como una buena opción el hacer uso de una plataforma que pueda aportar compatibilidad con los equipos actuales y futuros. Para ello, soluciones sencillas refieren a la actualización de la versión, o instalación de complementos desarrollados por los fabricantes, la propia comunidad, e incluso el propio usuario final, evitando así la necesidad de modificar las actuales partes del proyecto.

NF 02 consiste en tener en cuenta, en el diseño de la solución, la necesidad de flexibilidad para incorporar nuevos dispositivos, incluso soportando diferentes protocolos. En el mejor de los casos, se tratará en el proyecto de configurar un dispositivo con diferente protocolo a los inicialmente propuestos para el proyecto (ZigBee y Z-Wave). Para evaluar la dificultad y complejidad, se utilizará una bombilla WiFi.

NF 04 se refiere a la facilidad o dificultad que supondrá actualizar la solución propuesta por razón de:

- Aplicación de actualizaciones de seguridad: Debe ser una tarea asumible, sencilla y reproducible, sin poner en riesgo los datos de configuración, incluso pudiendo realizarse en remoto, o con una intervención del usuario mínima. **Nota: Gracias a este requisito, se pudo actualizar la solución pasando de la versión 3.1.0 a la 3.1.1 de OpenHAB que solucionaba un grave fallo de seguridad (log4j).**
- Habilitación de nuevas funcionalidades o capas necesarias: El uso de tecnologías modulares, definidas por software, y fácilmente automatizables, permitirá probar nuevas versiones de software o añadir nuevos módulos de forma mucho más sencilla. **Nota: el uso de contenedores,**

por ejemplo, facilitará la documentación técnica, la modularidad, y flexibilidad para añadir nuevas capas que permitan, por ejemplo, dar mayor robustez y seguridad a la solución.

2.3 Decisiones de diseño

La dirección del proyecto fija una capa HW que debe implementarse en una plataforma Raspberry por razones de sencillez y bajo coste. Para la capa SW, se establece el uso de Ubuntu, su sistema de Snap y de contenedores Docker (OpenHub, 2022).

De esta forma, desde un punto de vista de alto nivel, la implementación de este proyecto adopta los componentes de desarrollo que se indica en la Figura 5

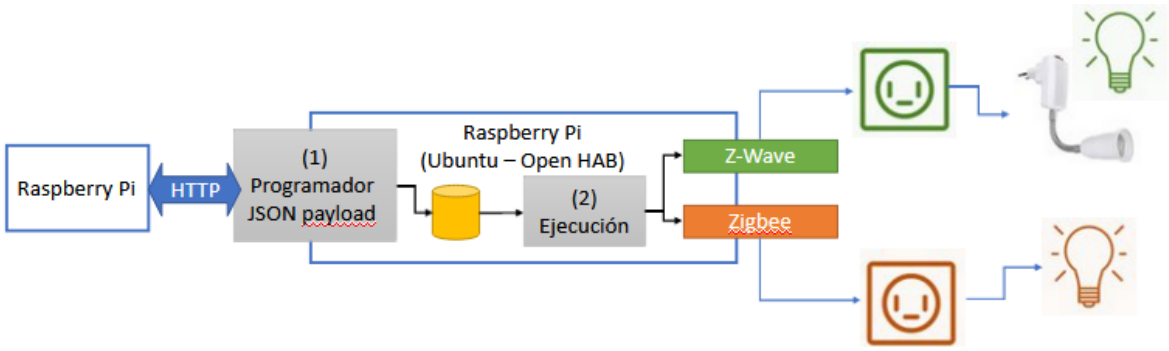

Figura 5 Diagrama de bloques de la implementación del proyecto

Los bloques (1) y (2) hacen referencia a los dos módulos de OpenHAB que, en conjunto, implementan la funcionalidad deseada. Dichos módulos utilizarían los servicios disponibles en la plataforma OH para interactuar con los dispositivos externos. La ventaja principal en delegar en OH la conectividad con los dispositivos ZW y ZB es la independencia de la solución de los fabricantes de dispositivos, ya que OH es un sistema en continuo desarrollo que continúa ampliando y mejorando la compatibilidad con diferentes productos, adaptadores al medio y versiones de los protocolos.

El módulo (1) escuchará peticiones HTTP con un formato JSON compatible con RFC 8259, los cuales procesará y almacenará para su posterior envío y ejecución. El formato de los mensajes es el indicado a continuación, el cual incluye un campo ID que permita identificar los dispositivos de forma unívoca.

```
[{
    "id": "device_1",
    "schedule": [ 10, 0, 50, 100, 100, 0, 0, 0, 0, 0, 0, 0, 20,40, 0, 40, 100, 90, 0, 0, 0, 0, 0, 0, 200]},
{
    "id": "device_2",
    "schedule": [ 10, 0, 50, 100, 100, 0, 0, 0, 0, 0, 0, 0, 20,40, 0, 40, 100, 90, 0, 0, 0, 0, 0, 0, 200]}]
```

El módulo (2) es el encargado de conocer la programación y enviar los comandos correspondientes, de encendido y apagado, a los diferentes dispositivos virtuales. Será OH, quien, en caso de disponer de un dispositivo real mapeado, enviará dicha orden por el medio correspondiente, hasta el dispositivo físico.

Para el acceso al medio, se deberán utilizar los módems ZB y ZW provistos, los cuales se indican en el apartado de requerimientos. Dichos dispositivos disponen de una conexión USB y el sistema operativo los reconocerá cómo dispositivos de comunicación serie. La solución implementada, tiene en cuenta el identificador a nivel SO para que el orden en que estos sean conectados o inicializados no afecte al

mapeo final. Por ello, no deberán emplearse identificadores genéricos, tipo /dev/ttyS0, sino identificadores reconocibles del tipo /dev/serial-by-id/usb-fabricante-id, de tal forma que podrán utilizarse desde cualquier módulo soportado por OH.

La aplicación por desarrollar dispondrá de un interfaz máquina-máquina (Saxena, et al., 2019) para interoperar, es decir, que no está previsto que se disponga de ninguna interfaz gráfica (GUI) o interfaz hombre-máquina, ya que el objetivo es que sea otro programa quien gestione las programaciones utilizando un vector de programaciones para cada dispositivo.

2.3.1 Cambios del análisis y diseño durante el proyecto

La propuesta inicial del anteproyecto, ya se mencionaba el uso de unas tecnologías determinadas, cómo es el caso Ubuntu y OH. Los motivos principales son los expuestos en la introducción y los requisitos, y OH cumplía todos ellos. También se consideró la experiencia previa de tecnologías basadas en Java por parte del estudiante, y el hecho de que OH estuviese en cierto modo amparado por el ecosistema de la Fundación Eclipse, OSGi [3]y basado en Java. Todo ello le brindaba mucha potencialidad y flexibilidad al sistema para mejoras futuras, o en caso de tener que resolver imprevistos durante el desarrollo de la solución, tal y cómo posteriormente sucedió. Véase Figura 6 para comprender las ventajas de la arquitectura OSGi en la que está basado OH.

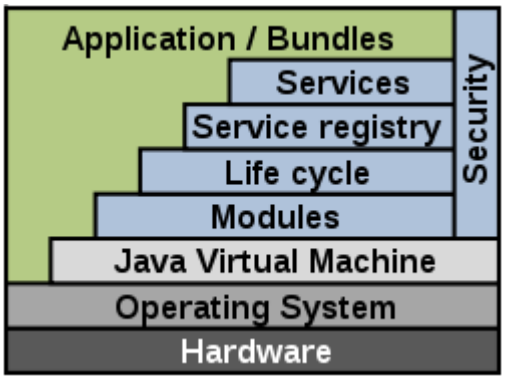

Figura 6 Arquitectura de servicio de pasarelas OSGi. Fuente Wikimedia Commons

No obstante, existen cada vez un mayor número de plataformas de software de código abierto similares para la misma finalidad que OH. En los dos últimos años dichas plataformas, incluso tienen módulos que permiten integrarse entre ellas, y no sería de extrañar que terminasen convergiendo y reutilizando parte de los desarrollos, y sobre todo, de las tecnologías subyacentes y los estándares utilizados. Cómo es el caso de los *payload* de MQTT, la modelización de las diferentes capas compartiendo los mismos modelos, o que cada uno se especialice en una parte de la cadena de valor, por ejemplo, Interfaz de Usuario vs Integración de dispositivos.

En el apartado 2.3.2.2 se de detallan brevemente las diferentes posibilidades existentes en la actualidad, y cómo gracias a las tecnologías seleccionadas, se pudieron sortear las dificultades encontradas sin variar mucho el rumbo.

2.3.2 Breve análisis de software alternativos y tecnologías

2.3.2.1 Versiones de Ubuntu y alternativas

En el momento de comienzo del proyecto, basado inicialmente en RPI 3 Model B v1.2, las posibilidades eran más limitadas en cuanto al uso de Ubuntu en la RPI, sin embargo, actualmente la versión más

[3] https://es.wikipedia.org/wiki/OSGi

reciente del SO ya ofrece soporte oficial de Ubuntu 22.04 para prácticamente todos los modelos de RPI, desde la v2 hasta la v4. Se puede consultar la compatibilidad de las diferentes versiones en el siguiente enlace: https://ubuntu.com/certified/devices?q=&limit=20&vendor=Raspberry+Pi+Foundation,

Ubuntu dispone de soporte oficial para Rapsberry Pi con tres distribuciones diferentes:

- Escritorio. Pensada para usuarios finales, incluye el escritorio de Ubuntu.
- Servidor. La más potente en cuanto funcionalidades, puede usarse en RPI
- Core. Diseñada especialmente para dispositivos IoT

Ante la elección del tipo de distribución Ubuntu a utilizar, se escogió la opción Ubuntu Core principalmente por tratarse de una distribución enfocada a IoT con capacidades de seguridad ya preconfiguradas. Si bien es cierto, que la solución actual podría utilizarse con la distribución para servidores, el trabajo de configuración y preparación del SO requería de un mayor trabajo y conocimientos. Por lo tanto, en lo que se refiere al uso de una distribución específica para sistemas en producción, se mantiene la elección de Ubuntu Core como la más conveniente.

Todas las Ubuntu Appliance existentes en la actualidad, entre ellas OH y LXD, están basadas en Ubuntu Core y despliegan las aplicaciones basadas en Snap. Era la intención de este proyecto utilizar dicho *Appliance*, aunque finalmente se prescindió del Snap para utilizar Docker, dotando al sistema de mayor flexibilidad para la conexión con otros módulos como deconZ[4] o Mosquitto (véase 3.2).

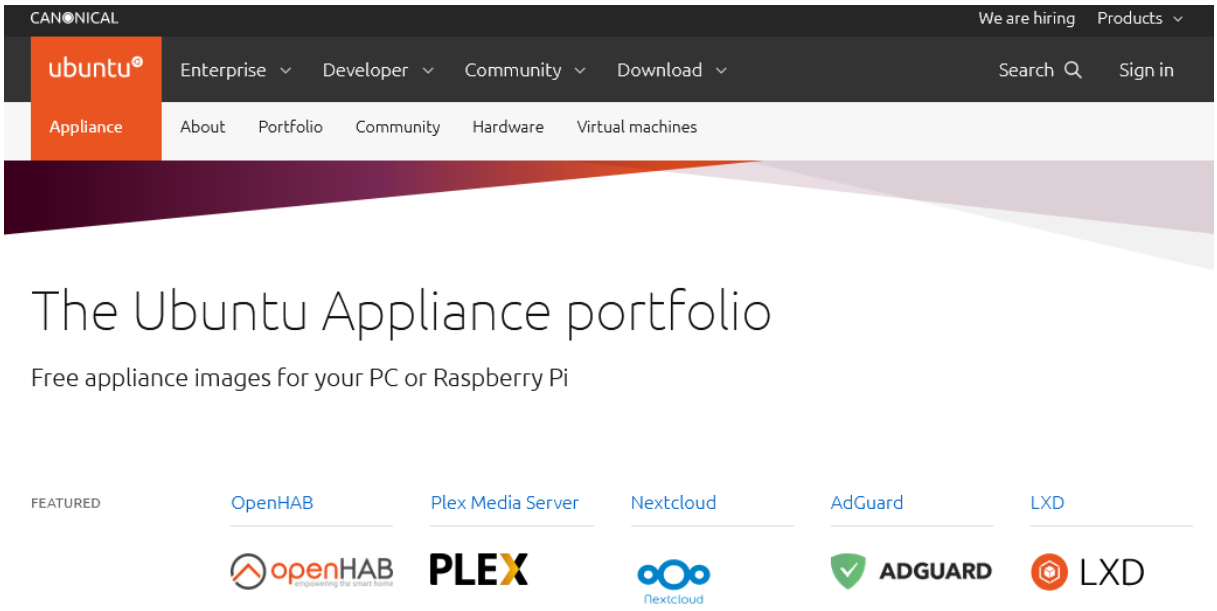

Figura 7 Muestra de las Ubuntu Appliance actuales, entre ellas OH. Septiembre 2022

2.3.2.2 OpenHAB y alternativas

Actualmente existen diferentes alternativas, similares a OpenHAB en funcionalidad, en cuanto a que permite la integración de dispositivos para monitorización y control. Dentro del gran abanico, se pueden categorizar dos grandes grupos: basados en fuentes abiertas y aquellos basados en software privativo. Ambas categorías comparten que son capaces de integrar dispositivos que manejan protocolos estándar, cómo ZB y ZW con cierta facilidad y sencillez por parte del usuario. Sin embargo, ambas

[4] deconZ es un software del fabricante Dresden Elektronic que se necesita para utilizar el adaptador ConBee II. [En línea]. Última visita septiembre 2022. https://www.dresden-elektronik.com/wireless/software/deconz.html

categorías difieren en el momento de integrar tecnologías más propietarias, siendo mucho más sencillo en el caso de los privativos a costa de una menor privacidad.

Tabla 1 Comparativa entre Gateways abiertos y privativos. Elaboración propia.

	Fuentes abiertas OpenHAB Home Assistant DomoticZ	**Privativos** Google Home Apple Home Kit
Ventajas	- Flexibilidad para modificarse - Facilitan la operación offline - Respetuosos con la privacidad - Promueven uso de tecnologías abiertas - Económicamente más asequibles - Facilitan economías de emprendimiento	• HW y SW forman un único producto, listo para su uso • Mejor integración con dispositivos cerrados o protocolos propietarios
Incovenientes	• Peor integración dispositivos menos respetuosos con la privacidad • Aún suponen un mayor conocimiento técnico.	• Menos respetuosos con la privacidad. • Mayor coste económico

En cuanto a las plataformas basadas en fuentes abiertas, cada una presenta características en las que destacan frente al resto. La elección de OpenHAB en un principio se hizo porque ésta cumplía con todas las características que se buscaban, y no porque el resto no pudiesen haberse utilizado indistintamente. No se puede afirmar que el uso de Home Assistant [5]o DomoticZ [6]hubiera sido mejor o peor en cuanto a los problemas y soluciones que se han encontrado durante el proyecto, ya que no se disponen de datos para hacer tal afirmación. Lo que sí se puede afirmar es que OH ha sido una elección acertada.

Comentar, que una de las características que inclinó balanza en la elección, era la constatación de que los proyectos de la Fundación Eclipse a menudo cumplen una serie de características en cuanto a soporte, documentación, calidad, y apoyo de la comunidad, que ya brindaba de partida unas garantías mínimas, como así se ha demostrado en los casos en los que se ha necesitado resolver algún problema de implementación.

Por último, cabe mencionar que probablemente, si nos basamos en la información que nos facilita una comparativa a Google Trends[7], la plataforma más popular y con mayor crecimiento de usuarios ha sido Home Assistant. En los siguientes párrafos se pone cierto contexto a este crecimiento para justificar por qué el uso de OH sigue siendo una buena elección.

[5] Plataforma Home Assistant [En línea]. Última vista septiembre 2022 https://www.home-assistant.io/
[6] Plataforma DomoticZ [En línea]. Última visita septiembre 2022 https://www.domoticz.com/Domoticz_es.html
[7] https://trends.google.es/trends/explore?q=%2Fg%2F11fzxlb_q4,%2Fg%2F1yw9kr31z,%2Fg%2F11gcgtrb33

Figura 8 Comparativa de búsquedas plataformas Home Automation. Google Trends Sept 2022

No es de extrañar, que, si nos fijamos en el desglose por región, en Alemania las búsquedas, y probablemente el uso de OpenHAB en comparación con HA sea mucho mayor que en el resto de las regiones. Se puede afirmar, sin miedo a equivocarse, que esto es así porque la fundación Eclipse, y la mayoría de sus proyectos tienen un apoyo mayoritario en Alemania, donde existe un ecosistema no sólo de usuarios finales de adopción temprana, sino de empresas que utilizan tecnologías Eclipse para fabricar sus productos y que participan en los mismos. Fabricantes como BOSCH, por ejemplo, han creado y apoyan proyectos IoT que se engloban dentro de la fundación Eclipse[8] en su apuesta por competir utilizando código *open source* (Bosch-Eclipse).

[8] Bosch-Eclipse, Bosch purses an open strategy to Transoform IoT [en línea]. Último acceso en septiembre de 2022.
https://iot.eclipse.org/community/resources/case-studies/pdf/Eclipse%20IoT%20Success%20Story%20-%20Bosch.pdf

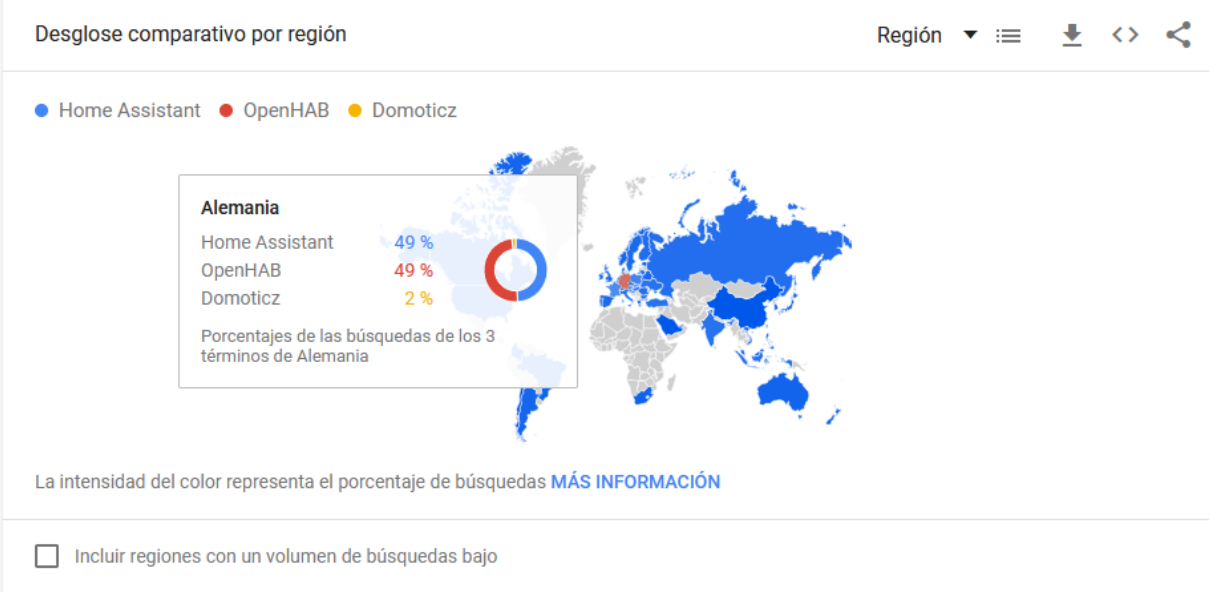

Figura 9 Alemania, país con mayor predominancia en el uso de OpenHAB

Figura 10 Porcentaje de búsqueda OH frente a otros en España. Google Trends. sep 2022

Es importante resaltar que aquellos países que tienen un mayor uso de OH tienen también una industria que se apoya en tecnologías abiertas, lo que en parte se ha considerado también como un buen criterio para la elección de OH como plataforma. Dicha industria completa el hueco existente en lo que se refiere a la adopción de tecnologías IoT que aún requieren de cierto conocimiento IT para su instalación y manejo (Almomani Rahman 2022 p26).

En segundo lugar, resaltar la importancia que tiene la confianza en los proveedores tecnológicos para la adopción de ciertas tecnologías (Almomani Rahman 2022), algo que puede cambiar muy rápidamente con algún escándalo relativo a la privacidad. Por lo tanto, apostar por sistemas que facilitan la privacidad de la solución final puede ser determinante para la adopción y éxito futuro de la misma, y ese es el caso de OpenHAB.

Tabla 2 Estadística nº búsquedas OpenHAB vs otros. Google Trends. Septiembre 2022.

1	País	Home Assistant: (18/9/17 - 18/9/22)	OpenHAB: (18/9/17 - 18/9/22)	Domoticz: (18/9/17 - 18/9/22)
2	Alemania	50%	48%	2%
3	Austria	51%	47%	2%
4	Suiza	63%	32%	5%
5	Lituania	80%	20%	
6	Bélgica	67%	18%	15%
7	Eslovaquia	71%	18%	11%
8	India	78%	18%	4%
9	Bulgaria	84%	16%	
10	Eslovenia	85%	15%	
11	Croacia	85%	15%	
12	Bielorrusia	85%	15%	
13	Grecia	86%	14%	
14	Ucrania	77%	14%	9%
15	Turquía	78%	14%	8%
16	Dinamarca	85%	13%	2%
17	Hungría	76%	13%	11%
18	Chequia	74%	13%	13%
19	Rumanía	81%	13%	6%
20	Emiratos Árabes Unidos	87%	13%	
21	Indonesia	79%	13%	8%
22	Italia	84%	12%	4%
23	Vietnam	83%	12%	5%
24	Argentina	88%	12%	
25	Finlandia	82%	11%	7%
26	Irlanda	89%	11%	
27	Rusia	80%	11%	9%
28	Noruega	87%	10%	3%
29	Nueva Zelanda	90%	10%	
30	Polonia	62%	10%	28%
31	España	84%	10%	6%
32	Canadá	88%	10%	2%
33	Sudáfrica	90%	10%	
34	Reino Unido	86%	10%	4%
35	Japón	90%	10%	

Obsérvese con detalle en la Tabla 3 y la Tabla 4 qué plataformas reciben un mayor porcentaje de búsquedas Internet en función del país. Los primeros puestos en el uso previsible de OpenHAB frente a HA y DomoticZ los ocupan Alemania, Austria, Suiza y Lituania. En los dos primeros su uso sería, según estos datos, similar al de Home Assistant.

Tabla 3 Estadística nº búsquedas Home Assitant vs otros. Google Trends. Septiembre 2022.

País	Home Assistant: (18/9/17 - 18/9/22)	OpenHAB: (18/9/17 - 18/9/22)	Domoticz: (18/9/17 - 18/9/22)
China	100%		
Estonia	100%		
Hong Kong	100%		
Chile	100%		
Arabia Saudí	100%		
Filipinas	100%		
Colombia	100%		
Taiwán	97%	3%	
Corea del Sur	96%	4%	
Singapur	93%	7%	
Israel	92%	8%	
Estados Unidos	92%	8%	
Malasia	92%	8%	
Nueva Zelanda	90%	10%	
Sudáfrica	90%	10%	
Japón	90%	10%	
Australia	90%	9%	1%
Brasil	90%	8%	2%
Portugal	90%	7%	3%
Irlanda	89%	11%	
Tailandia	89%	9%	2%
Argentina	88%	12%	
Canadá	88%	10%	2%
Emiratos Árabes Unidos	87%	13%	
Noruega	87%	10%	3%
Grecia	86%	14%	
Reino Unido	86%	10%	4%
Eslovenia	85%	15%	
Croacia	85%	15%	
Bielorrusia	85%	15%	
Dinamarca	85%	13%	2%
Suecia	85%	8%	7%
Bulgaria	84%	16%	
Italia	84%	12%	4%
España	84%	10%	6%

Téngase en cuenta que Google es el principal buscador con la mayor cuota de mercado 92%, salvo en China y Rusia, donde tienen un peso importante Baidu 65% y Yandex 48% respectivamente (Statcounter, 2022). Por otro lado, el uso de Google Trends se puede considerar una fuente fiable de información para detectar tendencias (Liu et Al. 2020, Quillay Huertas J 2020).

2.3.3 Z-wave vs Zigbee (y sus diferentes versiones)

Aunque existen varios protocolos que podrían competir con ZW y ZB a nivel técnico, como puede ser Thread (Morales et al, 2016) con su implementación OpenThread, parece que hasta la fecha estos (ZW y ZB) siguen presentes en el mercado de consumo y continúan evolucionando, y junto con los dispositivos Wifi (que no suponen necesariamente un estándar más allá del medio físico, y la capa de red), son los que principalmente se utilizan en la actualidad. Por esta razón y debido a la amplia literatura sobre ellos, se han escogido para este proyecto. Y aunque no es el objetivo de este apartado, el profundizar técnicamente en los mismos, sí que parece interesante el incluir unas breves pinceladas para que el lector pueda comparar.
Hay que destacar que mientras ZW siempre ha mantenido compatibilidad entre versiones, no es el caso de ZB, pues hasta la llegada de la versión 3 podía darse el caso de que dispositivos con el mismo nombre de protocolo, no pudiesen comunicarse, al utilizar diferentes perfiles de comunicaciones.

Se incluye en la Tabla 4 una breve comparativa entre ambos protocolos (Driss Ounes et al. 2019, Manuel Asdruval et al. 2021)

Tabla 4 Comparativa entre Zigbee y Z-Wave. Últimas versiones.

	Z-wave Plus	ZigBee 3.0
Caudal	9.6-100kbps	40-250 kbps
Banda de frecuencia	868 MHz	868 Mhz, 2.4 GHz
Alcance	1-40 m. (En línea recta)	10 m (En línea recta)
Requiere puente	Necesita puente Z-Wave	Necesita puente ZigBee
Tipo de red	Malla	Malla
Número máximo de dispositivos	232	65.000+
Número máximo de saltos	4	ilimitados
Retrocompatibilidad entre versiones	Si	No
Compatibles entre dispositivos garantizada[9]	Si	Sólo en v3.0
Seguridad	AES-128 / S2 (evita ataques DDoS)	AES-128
Consumo energético		
Precio	Asequible	Asequible
Código fuente de red disponible	No disponible	Disponible
Identificador único de dispositivo	Si	No

2.3.4 Raspberry-pi y los sistemas operativos con soporte oficial

Los sistemas operativos con soporte oficial para Raspberry Pi han ido en aumento con el paso de los años (se contabilizan hasta 23[10]), siendo una de las razones principales que su principal sistema operativo Raspbian, creado por el fabricante, está basado en Linux. De esta forma, el resto de las distribuciones basadas en el *Kernel* Linux lo han tenido mucho más sencillo para empezar. Lógicamente, cada distribución puede ofrecer características diferentes, así como diferentes niveles de soporte.

De los tres sistemas que ofrecían ciertas garantías históricas Raspbian, Ubuntu, Windows 10 IoT Core, se ha seleccionado Ubuntu, principalmente por su buen soporte, por ser de fuentes abiertas y porque el mismo sistema ofrece soporte no sólo para RPI, como es el caso de Raspbian, sino que soporta otros HW IoT disponibles en el mercado.

2.3.5 Ubuntu Server vs Core , uso de Docker y alternativas descartadas.

Como se ha comentado en el apartado 2.3.2.1 en un inicio se tenían disponibles dos tipologías de Ubuntu: la versión Server y la versión Core, siendo esta última la específicamente creada por Canonical para dispositivos IoT.

Según lo anterior, la elección parecía obvia, ya que la versión Core, incorporaba de serie y por diseño, todo el conocimiento y experiencia acumulados en el área de IoT, desde el procedimiento de instalación, hasta la comunicación entre procesos dentro de un mismo sistema, junto con su característica estrella: las actualizaciones transaccionales.

[9] Artículo de Spiceworks Zigbee vs Z-Wave [En línea] Septiembre 2022.
https://www.spiceworks.com/tech/iot/articles/zigbee-vs-z-wave/
[10] Sistemas operativos para RPI [en línea] Septiembre 2022
https://es.wikipedia.org/wiki/Raspberry_Pi#Sistemas_operativos

Las actualizaciones transaccionales no son lo único destacable, pero si una de las características más importantes que se incorporan con Ubuntu Core, ya que garantizan la estabilidad del sistema frente errores de actualización. Esto es de vital importancia en dispositivos que pueden contarse por miles o millones y se encuentran repartidos por extensas áreas geográficas.

Sin embargo, una vez seleccionado Ubuntu Core, durante unas pruebas se detectó que el Snap de OpenHAB no tenía acceso a los dispositivos adaptadores ZW y ZB que se conectan por USB , al menos esto era así en las versiones que se utilizaron al comienzo del proyecto. Tras una breve investigación se concluyó que, aunque se podían realizar modificaciones, estas eran de un calado y curva de aprendizaje importantes para el alcance del proyecto, pues se trataba principalmente de generar una imagen de Ubuntu Core propia[11], con paquetes Snap específicamente creados a modo "conectores", un concepto manejado por los Snap para mantener las aplicaciones dentro un *sandbox* o contenedor, pero con la posibilidad de interactuar con diferentes dispositivos usando dichos conectores, y que está especialmente limitado cuando se trata de acceder a la capa HW.

La solución más inmediata, tras unas pruebas de verificación, fue utilizar el Snap de Docker (dentro de Ubuntu Core para no perder todas las ventajas ya mencionadas), y desplegar los diferentes módulos mediante contenedores. Se observó que los contendores que se ejecutaban en dicho Snap de Docker, si tenían un mecanismo para acceder a los puertos serie sin ningún tipo de problema.

2.3.6 UDP vs COAP vs HTTP vs MQTT

En el momento de valorar las diferentes opciones disponibles para comunicar la programación al controlador, se tiene en consideración los protocolos UDP, COAP, HTTP, y MQTT, todos basados en IP y aunque no pertenecen necesariamente a la misma capa OSI, si que pueden ser todos ellos empleados para la comunicación que debe llevarse a cabo, con mayor o menor esfuerzo.

Tabla 5 Tabla comparativa protocolos UDP, COAP, HTTP, MQTT

	UDP	COAP	HTTP	MQTT
Soporte para UDP	Nativo	Si	No	Si (v0.5+)
Soporte para TCP	No	Si	Imprescindible	Si
Tamaño relativo de las cabeceras	Mínimo	Mínimo+4	Typical 700-800 bytes	Mínimo+2
Cifrado	No	Si (DTLS)	Si (SSL)	Si (SSL)
Autenticación	No	En borrador	Si	Si
Basado en RESTFul	No	Si	Si	No
Ofrece persistencia	No	No	No	Si
Modelo de comunicación	1-a-1	1-a-1	1-a-1	Muchos-a-muchos

En la Tabla 5 se resume brevemente la comparativa realizada, incluyendo un estudio preliminar de la facilidad para utilizarlos de forma nativa en OpenHAB, es decir, sin tener que realizar módulos o trabajo extra para su integración.

Cómo se puede observar, sería posible utilizar todos ellos, pero en el caso de UDP este queda descartado, ya que no aporta apenas beneficios, al no disponer de mecanismos de seguridad nativos, que otros protocolos como COAP y MQTT si que disponen. MQTT ofrece la ventaja adicional de que también soporta UDP. Y cómo principal desventaja de MQTT está el tipo de comunicación, muy potente, pero requiere de un servidor central.

[11] Imágenes personalizadas de Ubuntu Core. [en línea] septiembre 2022 https://ubuntu.com/core/docs/custom-images

Por lo tanto, si la comunicación va a realizarse 1-a-1 exclusivamente, puede utilizarse COAP y HTTP en función de la eficiencia respectiva, siendo COAP el más eficiente.

Por otro lado, si ya se requiere el uso de un servidor MQTT para otros fines, como comunicar con dispositivos Wifi que lo soporten (Tasmota, Tuya), parece que tiene sentido el reutilizar el componente, y habilitar el soporte UDP para aquellos casos en los que sea posible utilizarlo.

Para el presente proyecto, se utilizará HTTP para la recepción de programación, pero teniendo en cuenta que futuras implementaciones pueden utilizar MQTT, de cara al resto de las decisiones de diseño tomadas.

3 Implementación en OpenHAB

La implementación del presente proyecto se ha realizado con un hardware concreto, facilitado por el Dept. de Electrónica, que se describe a continuación:

- Raspberry Pi 4. Modelo B 4 GB
- Tarjeta SD de 16 GB
- Adaptador USB -Zigbee. Marca Dresden Elecktronik y Modelo ConBee II
- Adaptador USB- Z-Wave. Marca Aeotec y Modelo Z-Stick Gen5 (actualizado 2021)
- Enchufe Z-Wave. Marca Aeotec y Modelo Smart Switch 6 (ZW096-C16)
- Enchufe Z-Wave. Marca NEO Coolcam y Modelo Power plug (NAS-WR01ZE)
- Enchufe Zigbee. Marca Desconocida y Modelo Smart Socket EU Mini (C338-ZZigbee)
- Bombilla Wifi. Marca Alecto y Modelo Smart Bulb-10 (Compatible Plataforma Tuya)

Sin embargo, cómo ya se ha comentado, más allá de las especificidades de cada fabricante, la actual implementación debería ser compatible con otros dispositivos que soporten ZB y ZW, o incluso con otros diferentes, pues el diseño actual en capas, y basado en una plataforma abierta, así lo contempla.

3.1 Entorno de desarrollo

Cómo entorno de desarrollo se ha utilizado Visual Studio Code, el cual dispone de un plugin para trabajar con OpenHAB, es una herramienta muy versátil, y dispone también de soporte para conexiones remotas SSH, trabajar con la CLI integrada, o soporte para edición de documentos con Markdown[12] y PlantUML[13], los cuales se han utilizado para ir documentado las notas que servirían para la posterior realización de esta memoria, así como para las figuras empleadas.

3.2 Arquitectura software

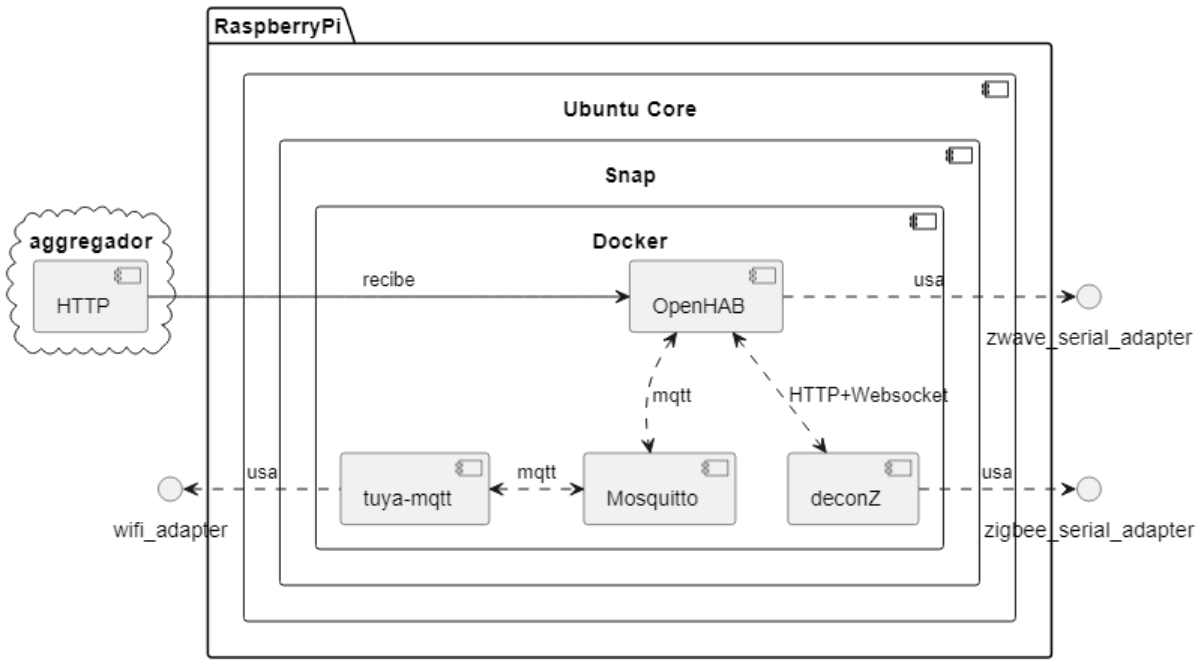

Figura 11 Arquitectura software del sistema

[12] https://es.wikipedia.org/wiki/Markdown
[13] https://plantuml.com/es/guide

La Figura 11 muestra las diferentes capas de software utilizadas y como se comunican entre sí en la implementación final. En un primer nivel se encuentra el SO Ubuntu Core que está estrechamente ligado con Snap. Docker, montado a su vez sobre Snap es la capa que contiene los módulos de aplicación OH y DeconZ. Como se puede observar, es necesario un módulo externo denominado DeconZ, que contiene un controlador propietario y una API REST del fabricante del adaptador ZB (Apartado 3.6). DeconZ, provee también de un mecanismo basado en Websockets[14], para que el cliente, en este caso OH, pueda recibir la información sin tener que pedirla, simplemente estableciendo una conexión permanente.

Los contenedores Mosquitto y tuya-mqtt, forman parte de la infraestructura necesaria para operar con dispositivos Wifi Tuya basados en Firmware original. Sin embargo, el servidor Mosquitto configurado, podría utilizarse en un futuro para recibir la configuración de los arrays de consumo.

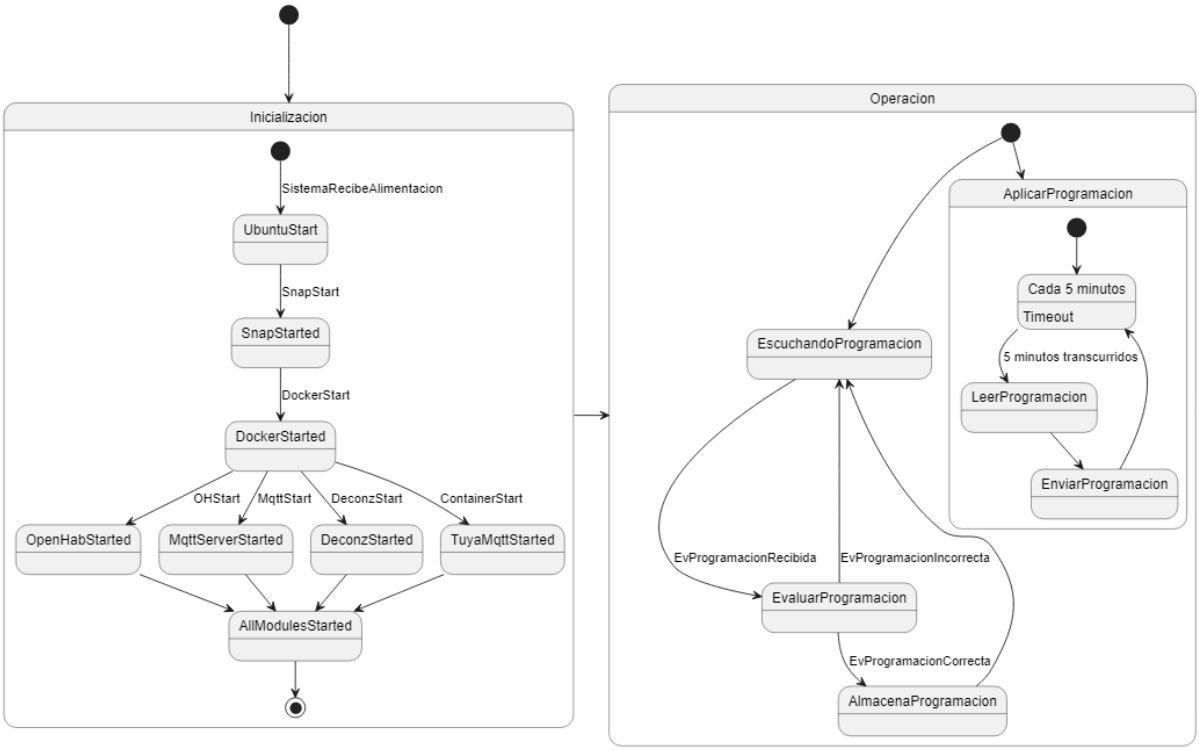

Figura 12 Diagrama de estados general del sistema

En el diagrama de estados (Figura 12) se detalla el funcionamiento del controlador desde el arranque, y durante la operación para recibir y enviar programaciones. Como se puede observar, una vez todos los sistemas se inician, el sistema permanece a la escucha de los arrays de consumos que le llegan del agregador. Cuando los recibe y éstos son correctos, los almacena. Por otro lado, existe una rutina que evalúa la necesidad de enviar la correcta programación a los dispositivos, forzando el envío de programaciones. La lógica del estado "Operación" está implementada mediante un sistema de reglas en OpenHAB (véase 3.4).

3.3 Sistema de reglas OpenHAB

OH dispone de un potente motor de reglas, el cual puede utilizarse incluso sin necesidad de conocimientos específicos de programación. Aunque también, permite crear reglas que utilicen bloques programados con lenguajes de alto nivel, utilizando algunos de los mecanismos de Scripting disponibles (véase 3.4).

[14] Documentación Websockets DeconZ API [en línea]. Septiembre 2022 https://dresden-elektronik.github.io/deconz-rest-doc/endpoints/websocket/

Una regla simple podría ser: “Cuando ocurra X, si Y se cumple, hacer Z”, donde:

- X es un disparador, el cual puede ser activado por algún evento, en un instante concreto, o periódicamente. En el caso que nos ocupa, se utiliza como desencadenante cuando se recibe un mensaje del programador, o bien para, de forma periódica enviar la programación correspondiente a los dispositivos.
- Y es una condición que será evaluada, y puede utilizarse, por ejemplo, para realizar comprobaciones de consistencia, el estado del sistema o de algún dispositivo antes de realizar una acción o ejecución de código innecesaria. Por ejemplo, si la RPI está correctamente sincronizada en hora.
- Z es la acción para realizar, y puede ser tan simple como encender o apagar un dispositivo, o algo más complejo, como es el caso del actual proyecto, donde el script de ejecución deberá leer la programación horaria de los dispositivos, y a partir de la misma, enviar un el correspondiente comando al dispositivo, en función de si debe apagarlo o encenderlo.

En 3.4 se explican las reglas que se han creado, las cuales mantienen el diagrama de estados que se muestra Figura 12 en el estado de operación.

3.4 Implementación lógica OpenHAB usando DSL

La lógica de funcionamiento implementada para la recepción y envío de la programación se ha realizado mediante scripts de OpenHAB.

Tal y cómo ya se comentaba en la parte de diseño de la solución, OpenHAB es un sistema lo suficientemente flexible y abierto como para realizar una misma función utilizando diferentes mecanismos. En la versión utilizada de OpenHAB estos mecanismos son:

- Programa externo realizando uso de la **API HTTP de OpenHab**, pudiendo utilizarse cualquier lenguaje de programación generalista cómo Java, C, C++, C#, Python.
- **Add-On en Java para OpenHAB utilizando SDK interno**, ejecutándose en el mismo contexto y JVM que OpenHAB. Usuarios muy avanzados, mayor complejidad de integración y depuración.
- **JSR223 scripts**[15]. Esta opción Brinda acceso a todos los paquetes disponibles en OH, permite utilizar diferentes lenguajes soportados por el motor de Scripts de la JVM, y acceder a Código interno no soportado en las API oficiales. Es una opción potente, aunque el uso de APIS no soportadas es siempre bajo la responsabilidad del usuario, ya que pueden dejar de funcionar, o suponer la inestabilidad del sistema.
- **OH Rules DSL**[16]OpenHAB DSL. Seleccionado para la implementación en este proyecto.
    - JSScript, basado en JavaScript EcmaScript 6, un estándar, y que requiere de la instalación del Add-On oficial [17]. Este Add-On contine una librería JavaScript, que también es utilizada por software de terceros para integrar sistemas con OH.

Para el caso del proyecto actual, una vez evaluadas las necesidades del mismo, se comprobó que no era necesario realizar programas externos basados en otros lenguajes, o fuera del control de OpenHAB, ya que se ha seleccionado dicho sistema como plataforma central, y las ventajas de utilizar sus propios mecanismos, en este caso la funcionalidad de Scripts mediante el uso del DSL nativo, provee de los mecanismos suficientes y ayuda a la portabilidad futura entre versiones.

[15] https://v31.openhab.org/docs/configuration/jsr223.html
[16] https://v31.openhab.org/docs/configuration/rules-dsl.html
[17] https://www.openhab.org/addons/automation/jsscripting/

En cualquier caso, no sería estrictamente necesario ceñirse a una opción concreta para la implementación de las lógicas, pudiendo utilizarse diferentes interfaces para diferentes lógicas, también en función de los conocimientos y necesidades de cada módulo y lógica a implementar.

La lógica se ha implementado mediante 3 scripts DSL encargados de procesar los mensajes recibidos, almacenarlos, y transformarlos en comandos para los dispositivos virtuales. Dichos scripts, se identifican por switches-1, switches-2 y switches-3 y trabajan en conjunto.

El código resultante, tras varias iteraciones, se ha conseguido simplificar en los 3 scripts mencionados más 1 de inicialización que se ejecuta una vez, con el arranque del sistema.

```
// Triggers:
// - Every 5 minutes

// context: switches-2

    //get item from group
    logInfo(logName, "Periodic item schedule rule started")

    lock.lock()
    try {

        gSwitchesSchedule.members.forEach[ switchitemSchedule |

            logInfo(logName, "proccessing schedule = " + switchitemSchedule )

            // Get the associated Schedule Item
            val dtStr = switchitemSchedule.name.replace("_Schedule","")

            logInfo(logName, "dtStr = " + dtStr)

            val assocDT = gSwitches.members.filter[dt|dt.name == dtStr].head as SwitchItem

            logInfo(logName, "assocDT = " + assocDT )

            //validate if object exists
            if(assocDT===null) {
                logWarn(logName, "Item "+dtStr+" not exists, schedule ignored")
            }else if( switchitemSchedule.state == NULL){
                logWarn(logName, "Item "+switchitemSchedule.name+" doesn't have schedule saved")
            }else{

                // get COMMAND depending on the hour
                // Assumes device is working in GMT time
                // If there is no value, it sends error
                // 0 = OFF
                // > 0 = ON
                logInfo(logName, "hour = "+ now.getHour)
                val filter = "$.["+now.getHour+"]"

        logInfo(logName, "transofrm = " + transform("JSONPATH", filter, switchitemSchedule.state.toString))

                val state = Integer.parseInt(transform("JSONPATH",filter, switchitemSchedule.state.toString),10) as Number

                //TODO poner aqui algo para controlar errores

                if(state==0) assocDT.sendCommand(OFF)
                else         assocDT.sendCommand(ON)
            }

        ]

    } finally{
        lock.unlock()
    }
```

Figura 13 Script switches-2 encargado de enviar la programación periódicamente

El script encargado de enviar la programación (Figura 13) se puede observar cómo el código de los DSL es de muy alto nivel, y su sintaxis es muy simple, aunque potente y con poca verbosidad. Se considera que el propio código es lo suficientemente descriptivo como para incluirlo directamente en la memoria, junto con algunos apuntes que se detallan a continuación.

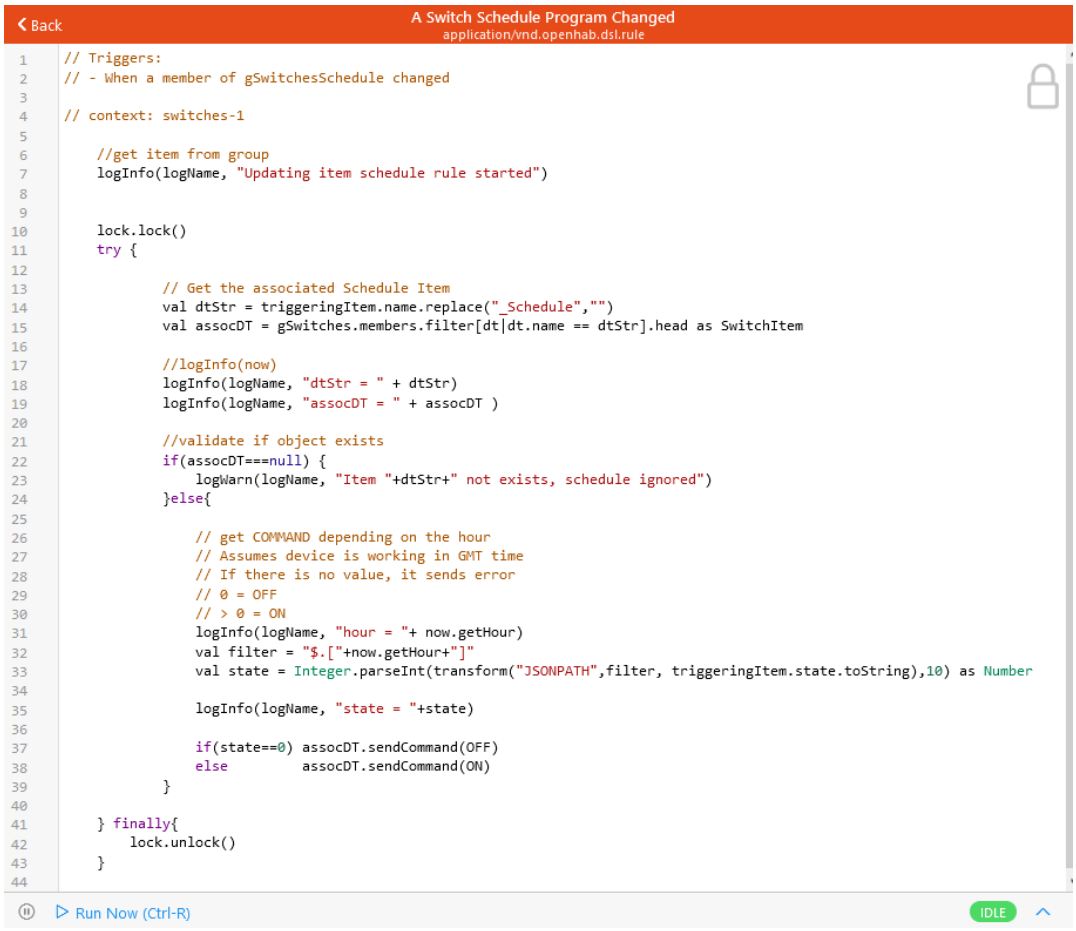

Figura 14 Script switches-1 encargado de detectar el cambio en una programación y enviarlo

Un ejemplo de su potencia y sencillez se puede observar en la línea 15 de la Figura 14, en el script encargado de detectar un cambio en la programación de un "item" (dispositivo virtual), para enviar el nuevo estado. En dicha línea se condensa la obtención de un "item" concreto filtrando por su identificador (nombre). En el caso de no existir dicho elemento, devolverá "null", lo cual se comprueba en la línea 22. Así, en el caso de que se reciban o existan programaciones para dispositivos que ya no existen o no han existido nunca, no se generará ninguna acción ni envío, aunque si se emitirá un mensaje de log con criticidad de *warn* (aviso), el cual quedará registrado en el log de OH. Dicho log ha sido de utilidad para depurar el desarrollo.

Hacer hincapié una vez más, en la potencia del DSL que permite enviar mensajes a un log, o que los errores no afecten al funcionamiento del sistema, y facilitando la labor del usuario que quiere crear reglas sencillas o complejas sin arriesgar la estabilidad del sistema y en muy pocas líneas. La única desventaja que se ha observado es que durante la realización del TFG la documentación del DSL[18] no estaba aún muy completa en la web de OpenHAB, seguramente por tratarse de una funcionalidad reciente. No obstante, el uso de repositorios de código públicos, y saber que la sintaxis está basada en Xtend [19]fue de gran ayuda para ir resolviendo las dudas que surgieron, dando como resultado unos scripts muy sencillos.

[18] OH DSL v3.3 [en línea] septiembre 2022 https://www.openhab.org/docs/configuration/rules-dsl.html#scripts
[19] Eclipse Xtend [en línea] septiembre 2022 https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html

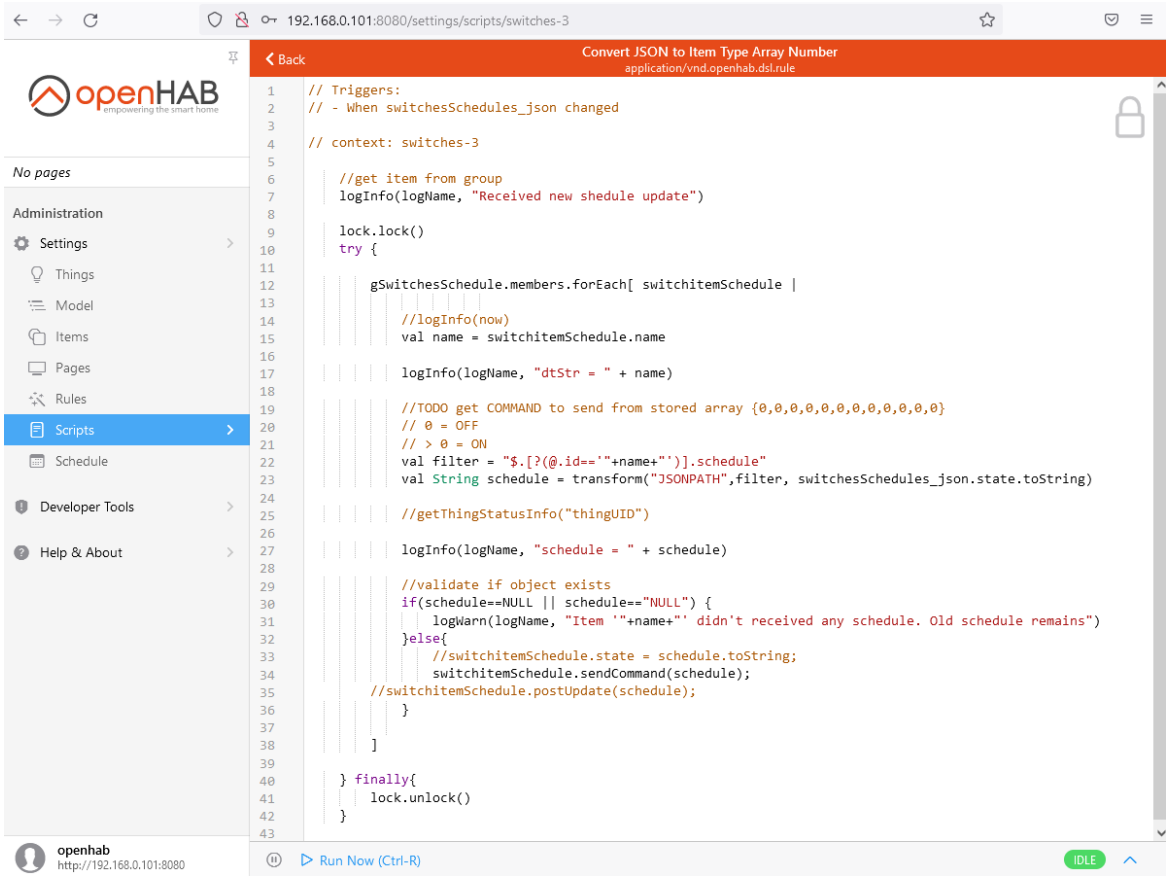

Figura 15 Script switches-3 que procesa el mensaje JSON recibido por la API

El script "switches-3" (Figura 15), encargado de procesar los mensajes que se reciben en la API, lo que hace en realidad es iterar sobre una lista de "ítems" tipo texto (línea 12). Dicha lista agrupa los elementos creados para tal fin Switch-01 a Switch-06. Después, para cada uno de ellos busca en el mensaje JSON recibido del programador mediante una regla JSONPATH (línea 23), y si existe un mensaje para ese "item", lo almacena en la en el "item" específico como la programación válida para dicho dispositivo (línea 34). Para la búsqueda, los identificadores dependerán del nombre de los elementos que se agrupen en un objeto de mayor nivel denominado gWitchesSchedule, al que podrían añadirse nuevos "ítems" virtuales a conveniencia.

Como se ha tratado de transmitir en esta sección, una vez se tienen estos scripts que cumplen la función encomendada de recepción, almacenamiento, y envío, el código es sencillo de modificar a futuras adaptaciones y mejoras simplemente editando el fichero ~/openhab/conf/rules/switches.rules.

3.5 Mapeado de dispositivos virtuales y físicos.

OH en su versión 3.0+ incorpora conceptos que facilitan el trabajo con dispositivos físicos, y separa muy bien las capas lógicas, así se pretenden:

- dotar de semántica a lo que se denomina "items" (dispositivos virtuales con atributos)
- evitar que un cambio en la tecnología, fabricante o modelo afecte al modelo

Tabla 6 Mapeado de dispositivos físicos y virtuales en OH

Dispositivo Físico	Dispositivo Virtual
NAS-WR01ZE	Swtich_01
ZW096-C16	Switch_02
Bombilla Alecto Bulb 10 Wifi	Switch_03
C338-ZZigbee	Switch_04

En la Tabla 6 se detalla el mapeo realizado para los dispositivos virtuales y físicos en OH.

Aprovechando esta característica, para este proyecto se han definido una serie de "ítems" que denominaremos "dispositivos virtuales" en adelante, y que tienen un nombre fijo. Dicho nombre se utilizará como identificador para los mensajes recibidos por la API.

Al usar éste mapeado, el controlador y programador siempre manejen los mismos identificadores, genéricos y privados para sus comunicaciones, manteniendo así la privacidad y dotando de flexibilidad al usuario para modificar los dispositivos conectados entre diferentes fabricantes sin afectar a las programaciones del proveedor de programación horaria.

Aquí es importante destacar, que esos mismos identificadores también podrán utilizarse si en el futuro los mensajes se recibiesen por MQTT. Se han definido hasta 6 "ítems", aunque sólo se han utilizado 4.

3.6 Configuración dispositivos Zigbee

Para el uso de dispositivos Zigbee desde OH se ha utilizado el adaptador con conexión USB Modelo Conbee II. Dicho dispositivo requiere del uso de una aplicación denominada DeconZ específica del fabricante Dresden Elektronik, siendo esta multiplataforma y multi-arquitectura.

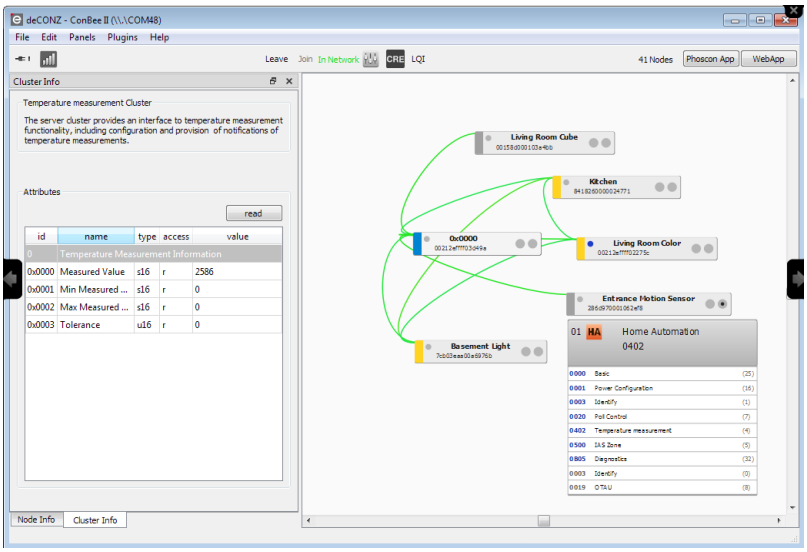

Figura 16 Ejemplo de aplicación deconZ. Fuente: dresden-elektronik.com

DeconZ, aunque no es de fuentes abiertas, dispone de amplia documentación y se en sus versiones mas recientes también como una imagen para contenedores disponible en el repositorio público de Docker. También dispone de un módulo de fuentes abiertas para proveer de una API REST a DeconZ[20].

[20] Repositorio público API REST de DeconZ [en línea]. Último acceso septiembre 2022 https://github.com/dresden-elektronik/deconz-rest-plugin

DeconZ se distribuye como una aplicación (escritorio o web) que permite configurar y ver información de la red Zigbee (Figura 16), como por ejemplo la topología. También permite gestiones más avanzadas. Esta herramienta puede resultar útil para diagnosticar problemas en la red, aunque no es estrictamente necesaria para dar de alta y configurar dispositivos en OH.

Por lo tanto, para que OH pueda comunicar con los dispositivos ZB se debe contar con dos elementos HW y SW del fabricante, así como el *add-on* específico de OH para comunicarse con la API REST de DeconZ (véase Figura 21 Arquitectura de comunicaciones)

Se incluye aquí una observación sobre la forma de integrar OpenHAB y Conbee II, que puede ser relevante en algunos sistemas futuros:

Nótese que el a*dd-on* para la API REST de DeconZ sería prescindible si OpenHAB utilizase directamente las librerías / drivers que provee el fabricante, optimizando de esta forma mucho más la integración de las comunicaciones con ZB. Lo cual también reduciría el número de módulos necesarios. Sin embargo, para llevar a cabo dicha integración optimizada, OpenHAB al estar realizado en Java, debería crear una librería basadas en JNI para interoperar con las librerías DeconZ de más bajo nivel, basadas en c++. Se entiende ahora, porqué hasta la fecha la forma de integrar DeconZ con la mayoría de las plataformas para la automatización del hogar se emplea la API REST, pues de esta forma, también se aprovecha mejor el esfuerzo de integración en todas ellas. Por lo tanto, se recomienda el uso de las librerías c/c++ para DeconZ en sistemas que requieran la máxima optimización posible, o al menos más de la que puede lograrse utilizando la API REST, y el uso de la API en el resto de las situaciones.

Continuando con la configuración, una vez se dispone de todas las partes previamente indicadas, es posible comenzar la configuración de los diferentes switches ZB, siguiendo para ello las instrucciones de cada fabricante, que normalmente consisten en los siguientes pasos:

- Conectar y desconectar el dispositivo N veces (3 típicamente) en un periodo corto de tiempo para habilitar el modo configuración.
- Una vez el dispositivo está en modo configuración el concentrador o Gateway podrá detectarlos para ser incluidos en su red.
- El concentrador, en este caso Conbee II, hace de Gateway por lo que OpenHAB, que está escuchando en la mencionada API REST, recibirá una notificación para dar de alta el nuevo dispositivo.
- El usuario debe seguir los pasos indicados por OpenHAB para configurar el dispositivo detectado.
- Una vez dado de alta el dispositivo, este deberá asociarse o mapearse a un Item virtual de los preconfigurados en OH para el actual proyecto (Switch_01 a Switch_06). Este paso es necesario para que los dispositivos físicos reciban los correspondientes comandos.

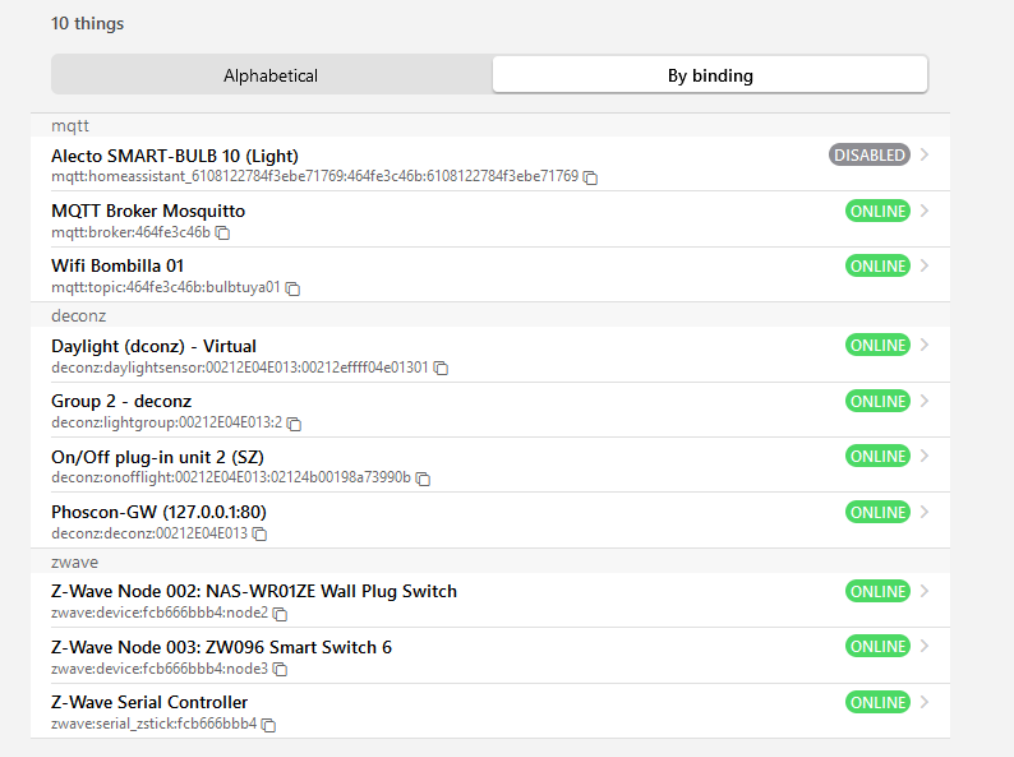

Figura 17 Listado de dispositivos configurados en OH

Puede verse en la Figura 17 como se muestran en OH los diferentes dispositivos configurados, en este caso, se muestran ordenados por el tipo de conexión. Obsérvese que dentro de los dispositivos listados no se encuentran únicamente los enchufes o bombillas, sino también los diferentes adaptadores / conectores que se están empleando para acceder a los mismos, y que para OH son un dispositivo más, aunque con la característica especial de que, si el conector no está operativo, los dispositivos que cuelgan de él automáticamente se consideran en modo offline.

3.7 Configuración dispositivos Z-Wave

La configuración de los dispositivos Z-Wave dentro de OH es más simple que para el caso Zigbee. Esto es así, porque el correspondiente *Add-On* Z-Wave ya incluye lo necesario para comprender el protocolo y gestionar el canal de comunicación serie con el adaptador correspondiente (Z-Stick). Por lo tanto, una vez se añade el dispositivo OpenHAB ya puede trabajar con él.

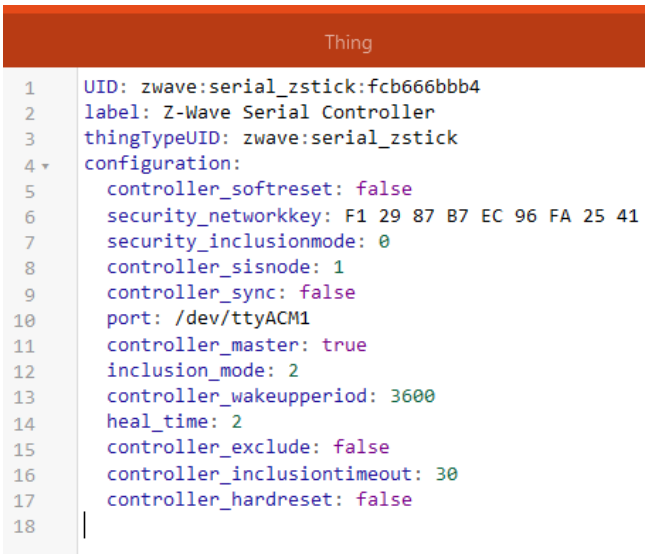

Figura 18 Parámetros de configuración del Addon Z-Wave

En la Figura 18 se puede observar los parámetros que necesita conocer OH para gestionar el adaptador ZW, obsérvese la línea 10, donde se muestra el puerto serie donde buscar el adaptador. Recordar aquí, que dicho puerto serie no es el real del host, sino el que se ha especificado en la configuración del contenedor Docker con el fin de que la configuración de OH sea siempre la misma, independientemente del puerto físico donde éste se conecte a la RPI.

Sin embargo, una característica importante en este caso es que, aunque OH conoce la configuración de los dispositivos y de su existencia, debido a que el adaptador ZW utiliza encriptación por defecto cuando se configura la red, los dispositivos sólo podrán comunicar con el adaptador Z-Stick en el que se hayan configurado, que almacena claves y dispositivos en su memoria interna.

Por lo tanto, un cambio del adaptador implica volver a configurar todos los dispositivos de nuevo. Si se quiere evitar que esto ocurra, existe una herramienta del fabricante que únicamente funciona en Windows y puede utilizarse para realizar una copia de seguridad[21].

3.8 Configuración dispositivos WiFi (opcional)

A continuación, se describen brevemente los pasos seguidos para configurar el dispositivo Wifi. Sin embargo, aquí se refiere expresamente al tipo de bombilla que se ha tratado de configurar para el proyecto, consistente en el modelo Smart Bulb Multicolour Led Bulb (SMART-BULB 10) del Fabricante ALECTO.

Paso 1. Creación de una cuenta *trial* en iot.tuya.com (cuenta gratuita de 30 días como desarrollador). Es necesario seleccionar el *datacenter* adecuado, en mi caso era uno de los disponibles para Europa

Paso 2. Realizar todos los pasos necesarios para dar de alta una nueva aplicación, y obtener así credenciales y claves para utilizar la plataforma como desarrolladores

Paso 3. Preparar el dispositivo en el modo enlace (depende del fabricante y el dispositivo)

[21] Herramienta Backup Z-Stick [en línea]. Septiembre 2022 https://aeotec.freshdesk.com/support/solutions/articles/6000108806-z-stick-gen5-backup-software

Paso 4. Utilizar credenciales y datos obtenidos con el cliente Tuya-Cli desde un ordenador con acceso a la red wifi que se quiere configurar y a Internet, para ejecutar un comando, similar al siguiente:

```
> tuya-cli link --api-key <your api key> --api-secret <your api secret>
--schema <your schema/Channel ID> --ssid <your WiFi name> --password
<your WiFi password> --region us
```

Paso 5. El software Tuya-Cli que actúa como una aplicación de cualquier proveedor, y haciendo uso de las librerías del fabricante, configura la bombilla y obtiene la clave privada necesaria.

Paso 6. Utilizar la clave (*local_key*) obtenida para configurar tuya-mqtt para hablar con el dispositivo, indicando el resto de datos como identificador, nombre, o IP.

Por otro lado, tal y cómo se describe en los pasos realizados, una vez obtenida la información necesaria para comunicar con el dispositivo y al no existir un conector específico para dispositivos Tuya en OH, fue necesario montar dos contenedores Docker adicionales únicamente para este fin (véase Figura 11).

Uno de ellos es el relativo al servidor MQTT, para el que se ha utilizado la implementación oficial de Eclipse Mosquitto (Eclipse Mosquito, 2022) mediante el uso de una imagen oficial de Docker. Por otro lado, para crear el contenedor Tuya-MQTT, se ha utilizado un código fuente libre disponible en internet y denominado Tuya-mqtt (véase Figura 11 Arquitectura software del sistema).

Tuya-mqtt es el software encargado de conocer la implementación MQTT de Tuya para, utilizando la clave privada única obtenida para cada dispositivo, traducir los mensajes entre el dispositivo y OpenHAB, utilizando para ello el servidor MQTT.

Comentar, que el proyecto Tuya-MQTT no está mantenido por su creador desde junio de 2021. Es por ello, que no se recomienda su uso para futuras ampliaciones, aunque si se desea mantener, la implementación del código es muy sencilla.

Dado que Tuya-MQTT no disponía de una imagen Docker, la misma se ha creado expresamente para este proyecto (disponible en https://github.com/gavioto/Tuya-mqtt/tree/tfg).

La arquitectura final adoptada, para ofrecer soporte a bombillas basadas en Tuya sería la que abarca todos los módulos, incluidos los opcionales, mostrados en la Figura 11 Arquitectura software del sistema.

El Anexo C, contiene información adicional sobre la tecnología empleada por los fabricantes de dispositivos Wifi basados en Tuya y sus dificultades de integración en plataformas y redes que no disponen de acceso directo a Internet para el dispositivo a integrar. También se explica el estado del arte actual para dotar a dichos dispositivos de un firmware de código libre.

4 Experimentos y validación

4.1 API Interfaz de usuario

Como se ha explicado, la RPI recibe la programación horaria a través de llamadas a una API HTTP en la implementación actual. Aunque se ha comentado que se puede adaptar fácilmente el script que procesa las peticiones para que dicha programación se reciba vía MQTT, esta parte se deja para futuros proyectos, con la conectividad MQTT ya habilitada.

Por lo tanto, para la ejecución de los siguientes ejemplos, se requiere un cliente HTTP. En este caso, se utilizará la herramienta curl, una herramienta CLI la cual puede utilizarse de forma sencilla desde cualquier sistema basado en Linux o en sus equivalentes para otros sistemas operativos.

El uso de la API HTTP de OpenHAB requiere el uso de autenticación, por lo que, para poder enviar comandos, el primer paso es disponer de la clave, que puede generarse desde la propia interfaz de OpenHAB, tal y cómo se explica en el manual de usuario (Anexo B).

Para enviar la programación a uno o varios dispositivos el comando utilizado es el siguiente:

```
> curl 'http://${IP_PORT_RPI}/rest/items/switchesSchedules_json/state'
-X PUT \
-H "Content-Type: text/plain" -H "Accept: application/json" \
-d ^
'\[{"id":"Switch04_Schedule","schedule":\[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0\]}\]'
```

Donde la variable IP_PORT_RPI debe tener el valor correcto para la dirección IP y el puerto donde se encuentra la API. Para el caso del ejemplo, el

El contenido enviado por la petición debe ir en formato JSON, tratándose de un array que cumpla el criterio de contener 24 valores.

A continuación, se describe en mayor detalle, cómo está funcionando este mecanismo para el envío de la programación.

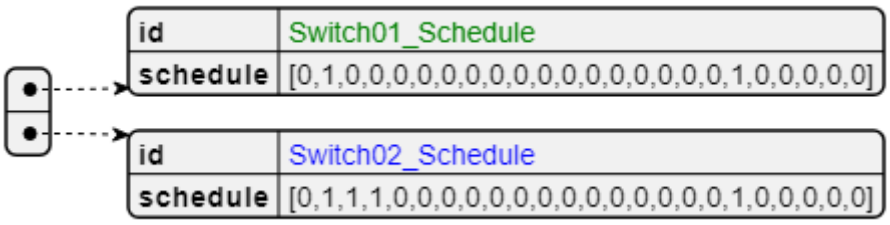

Figura 19 Ejemplo de mensaje JSON con programacion horaria

Tal y cómo se muestra en la Figura 19, el mensaje que se envía la RPI hace referencia al elemento controlable "virtual", por ejemplo, Switch01, que a su vez estará conectado a un dispositivo concreto con función de apagado y encendido (ON/OFF), siendo indiferente que se trate de una bombilla o de un enchufe. En el apartado 3.5 se explica sobre el mapeado de dispositivos virtuales y físicos. Por lo tanto, los nombres de los dispositivos para el que envía la programación horaria serán siempre los mismos: Switch01..06.

4.2 Escenarios de prueba

Como escenario de prueba se ha realizado un montaje de demostración portátil (Figura 2). Dicho montaje consta de una maqueta de una casa con habitaciones, cada una de ellas con una toma de corriente donde alojar y conectar los diferentes dispositivos a controlar. También se alojan en la casa, la RPI, y un enrutador con conectividad Wireless que permita la conexión de los dispositivos a una red conocida con el fin de evitar la reconfiguración de redes y claves de acceso, así como evitar depender de infraestructuras externas para realizar las pruebas y demostraciones.

De esta forma, se pueden realizar pruebas directamente desde cualquier dispositivo móvil con Wifi, o bien conectarlo a una red de más alto nivel mediante cable ethernet. Se trata de una maqueta portátil que se puede emplear para realizar demostraciones con mucha facilidad.

El *router* Wifi, también se ha utilizado para conectar la bombilla Wifi a OpenHAB, y aunque se consiguió hacerla funcionar, tal como se explica en el Apartado 3.8, la misma dejó de funcionar al desconectarla y volver a conectarla un tiempo después.

Una vez realizado el montaje, se han realizado una serie de pruebas, que se explican a continuación.

4.2.1 Escenario 1. Prueba de estabilidad

Se ha probado a dejar durante varios días el sistema funcionando, y comprobar que el mismo ejecuta las programaciones enviadas.

4.2.2 Escenario 2. Envío de mensajes incorrectos

Se han enviado mensajes incorrectos, con mensajes JSON que no cumplen las características, bien por no ser JSON válido, o bien por no contener el mensaje correcto. El resultado es que dichos mensajes han sido descartados, permaneciendo la programación anterior en el sistema.

4.3 Resolución de problemas

Durante la realización de este proyecto, se han encontrado diferentes limitaciones y problemas, que han supuesto horas de investigación y pruebas. Se incluye a continuación una explicación de las mismas por si fueran de utilidad para futuros proyectos, junto con las soluciones realizadas y/o propuestas de solución en aquellos casos donde no se ha implementado la misma.

4.3.1 Reloj RPI

Los dispositivos RPI no incorporan RTC, por lo que la única fuente de sincronización horaria posible es vía conexión de red a un sistema externo. Dicho sistema no tiene por qué ser un servidor NTP ubicado en internet, aunque esta es la opción más común. En el caso de que el dispositivo no tuviese acceso a un servidor externo, podría utilizarse uno local, ubicado en la misma LAN. En última instancia, la opción más recomendable para un equipo que quisiese utilizarse para un sistema en producción o de venta al público, basado en RPI4, el mismo debería incluir un RTC externo montado en la placa.

Un ejemplo de integración de dispositivo RTC [22]en RPI podemos encontrarlo en la web del fabricante AdaFruit (AdaFruit, 2022). En la mayoría de los casos, la integración de estos dispositivos se realiza mediante I2C.

[22] https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi/wiring-the-rtc

4.3.2 Configuración bombillas Wifi Alecto Smart Bulb. Requieren conexión a internet

Es muy común que los dispositivos de fabricantes que dicen no necesitar ningún Gateway intermedio, como es el caso de las bombillas Wifi Alecto Smart Bulb con las que se ha trabajado en el proyecto, puedan requerir una red Wifi con conexión a Internet para su configuración. Y en algunos casos, también para su operación.

Dichas bombillas están basadas comúnmente a en el chip ESP8266[23] y derivados, lo que debido al amplio uso de este chip por parte de los fabricantes de dispositivos Wifi hace también que hayan florecido herramientas no soportadas por los fabricantes para reprogramar dichos dispositivos con firmware que respete la privacidad, y que dote a los mismos de una mayor facilidad de integración. Uno de estos *firmware* se denomina Tasmota[24], y es de código abierto, y el proyecto dispone de un repositorio para consultar los dispositivos soportados[25].

Durante la realización del proyecto, tras una ardua investigación, se añadieron los componentes necesarios para hacer funcionar la bombilla con OH sin hacer uso de Tasmota, ya que en ese momento se desconocía su existencia (ver Apartado 3.8 donde se detallan los pasos realizados), sin embargo, tras un tiempo, la bombilla dejó de responder al sistema. Casualmente ese tiempo la bombilla no tuvo conexión de ningún tipo a internet, y posteriormente, cuando se dio acceso a internet a su red (sin modificar nada en la misma), continuó sin funcionar.

En el aire queda la cuestión de porqué los fabricantes obligan a los usuarios a conectarse a sus plataformas en la nube antes de poder utilizar los dispositivos "inteligentes" que éstos adquieren.

4.3.3 Nuevas versiones de software debido a la duración del proyecto

El proyecto comenzó con OpenHAB 2.5, sin embargo, cuando se publicó la versión 3.0, el sistema de demostración se actualizó automáticamente y sin previo aviso. Esto ocurrió tal y como se explica más adelante, debido al uso de Ubuntu Core, basado en el uso de Snap.

Tabla 7 Software. Versiones utilizadas y últimas disponibles

	Utilizada	Última disponible
DeconZ	2.07.01	2.18.2
OpenHab	3.1.0	3.3.0
Mosquitto	1.6.5	2.0.15
Docker	20.10.14	20.10.14
Snapd	2.56.2	2.57.1
Ubuntu Core	18	22

Finalmente, el proyecto llegó a una madurez y estabilidad cuando OpenHAB 3.1 se publicó, y desde entonces la versión utilizada para el proyecto no se ha actualizado con el objetivo de evitar contratiempos en la implementación.

Es por ello, que en el momento de publicación del proyecto las versiones más actuales del software disponible y las utilizadas se incluyen en la Tabla 7.

[23] https://es.wikipedia.org/wiki/ESP8266

[24] « Tasmota es un firmware de código abierto para tarjetas basadas en el chip Esp8266, este software proporciona interfaz web, actualizaciones inalámbricas y compatibilidad con diversidad de sensores, permitiendo el control a través de diferentes protocolos como HTTP, MQTT, Serial y KNX.>> (Simbaña et Al 2021)

[25] Respositorio Tasmota de dispositivos soportados [en línea]. Último acceso septiembre 2022. https://templates.blakadder.com/index.html

En principio, la actualización del sistema es sencilla de probar, una vez se tiene copia de seguridad de la imagen de la tarjeta SD, y se puede hacer tanto offline cómo online, especificando la versión de la imagen para usar con los contenedores. Véase Anexo B. el apartado de configuración.

4.3.4 Limitaciones HW encontradas para cumplir con los requisitos/diseño inicialmente propuestos

Migración a RPI 4 e incompatibilidad con adaptador Z-WAVE

Debido al uso de Ubuntu, Snaps, y finalmente Docker, junto con todos los módulos de los que se quería hacer uso, la carga del sistema es algo mas elevada, y la RPI 3 que trató de utilizarse a veces tenía problemas de capacidad. Aunque teóricamente, podría llegar a utilizarse si el sistema se optimiza al máximo, pero no en la primera iteración del controlador, que es el caso que nos ocupa, donde el objetivo principal era conectar todas las partes del mismo y tener un sistema funcional.

Por esta razón se pasó a utilizar la RPI 4 con mayor capacidad de memoria, sin embargo, con el cambio de HW apareció un problema de incompatibilidad con el adaptador Z-wave. Tras un tiempo de investigación se localizó el problema en el propio adaptador[26], que al no implementar correctamente el protocolo USB y sus voltajes, era incompatible con RPI 4. El fabricante ofrecía la posibilidad de actualizar el FW o adquirir uno nuevo, ya corregido, siendo esta segunda la opción escogida. Una tercera solución, consistía en utilizar un HUB USB intermedio, lo cual suponía complicar más el montaje, razón por la que se descartó.

La información relativa al problema está disponible en varios foros, sin embargo, el fabricante publicita el modelo como "Actualizado en 2020 y compatible con RPI 4." pero no se puede encontrar una declaración oficial de la incompatibilidad detectada.

4.3.5 Limitaciones SW para cumplir con los requisitos/diseño inicialmente propuestos

Uso de Snap

En un inicio se trató de utilizar lo que la compañía Canonical denomina "Ubuntu Appliance" y que consiste en una serie de aplicaciones basadas en Ubuntu que se distribuyen ya listas para su ejecución y configuración por un usuario final en dispositivos tipo RPI. Una de ellas es OpenHAB, y por esta razón, una vez tomadas las decisiones de diseño, y observando que Canonical ofrecía una imagen de su sistema operativo ya listo para su uso en RPI como plataforma IoT, no hubo duda de que sería la mejor opción.

Sin embargo, una vez configurado el sistema, y cuando ya se habían configurado algunas de sus partes, se encontró que algunas dejaron de funcionar, y esto llevó a investigar las causas, con lo que finalmente, y tras la constatación de los hechos que se detallan en la siguiente lista, hubo que pensar en una alternativa que hiciese el sistema más reproducible y estable.

- Ubuntu Appliance está basado en Ubuntu Core. Ubuntu Core es un sistema pensado para IoT y por lo tanto, por razones de seguridad, su sistema funciona con muchas limitaciones. Principalmente Ubuntu Core sólo soporta el uso de contenedores, y para ello, Canonical desarrolló la tecnología de Snap.
- La idea de Ubuntu Core es que pueda actualizarse cuando sea necesario, tanto el propio sistema como las aplicaciones que tenga ejecutándose como Snap. Por defecto los Snaps se actualizan cuando hay una nueva versión disponible, lo que equivale a actualizar la versión de la aplicación sin control del usuario.

[26] Enlace al hilo principal del problema. [en línea] sep 2022 https://community.home-assistant.io/t/sticky-aeotec-z-stick-gen5-raspberry-pi4/218405

- OpenHAB Ubuntu Appliance es básicamente un Ubuntu Core que tiene instalado el Snap de OpenHAB, por lo tanto, cuando se publica una nueva versión del Snap de OpenHAB, este se actualiza.
- Las versiones del Snap y de OH no tienen porqué ir a la par, aunque normalmente si lo hacen, no obstante, si fuera necesario la intervención manual para dicha actualización, el sistema podría dejar de funcionar sin previo aviso.
- Y así es como sucedió durante el desarrollo del proyecto, con lo que hubo que solucionarse el problema causado con las actualizaciones, ya que determinado HW dejó de funcionar, debido principalmente a que los Snaps están muy limitados para el acceso al HW y más a un puerto serie o a un dispositivo USB como es el caso de los módems ZB y ZW.
- Es cierto, que para que el sistema se actualizase sólo, se requería que el mismo tuviese conexión a internet, pero también se identificó como un problema el no poder conectarlo en caso necesario, o el volver a instalar una versión antigua, cuando dicho software está pensado para instalar siempre en la última versión.

Por las razones anteriores, se decidió que debía buscarse otra forma de configurar el sistema, donde se tuviese el control total sobre las versiones a utilizar, y así, garantizar que el paso del tiempo no haría que el proyecto dejase de funcionar. Y así es cómo se llegó al uso de Docker, que ya se estaba utilizando para otros módulos del sistema (DeconZ), instalado en este caso como un Snap, pero que garantiza, si se toman las debidas precauciones, que todas las aplicaciones que se ejecuten en dicho entrono (Docker) si estarán realmente aisladas y controladas en lo que a su versión se refiere.

Para más información sobre Docker, sus ventajas, y los casos de uso, se remite a la web del fabricante, ya que la documentación es muy extensa. Entre ellas destacan, la posibilidad de limitar de forma sencilla los recursos del sistema que empleará cada módulo, así como la ejecución de programas de terceros en un entorno controlado (ej. Tuya-mqtt).

4.3.6 Actuadores mecánicos para electrodomésticos con botones o sensores de huellas.

Cómo ya se ha comentado, el objetivo final del controlador es el encendido y apagado de ciertos electrodomésticos o, más bien, la puesta en marcha.

En la maqueta de demostración se han utilizado actuadores tipo Switch a modo de enchufes controlables mediante los protocolos ZB y ZW, los cuales, podrían llegar a emplearse para el control de ciertos dispositivos cómo una tostadora, la iluminación, persianas o toldos. La idea consistía en emular el control sobre electrodomésticos capaces de comunicarse por los mencionados protocolos. Pero ¿qué ocurre con todo el parque de electrodomésticos ya instalado, y otros elementos que no disponen de la tecnología necesaria?

Para lidiar con la dificultad de integrar dichos electrodomésticos se propone la utilización de unos dispositivos capaces de comunicar por dichos protocolos, y que disponen de actuadores mecánicos a modo de “dedos robóticos”. Mediante el uso de dicha tecnología, la cual es económicamente asequible, se podrían integrar los electrodomésticos más antiguos. Una búsqueda en internet ha servido para localizar los siguientes modelos:

- Fingerbot Plus (Wifi). Fabricante Adaprox
- SwitchBot (Bluethoot). Requiere Gateway

La principal desventaja que supondría el uso de este tipo de actuadores es que no en todos los casos se conoce el estado del electrodoméstico, ni existe un *feedbak* real sobre el estado del mismo. No obstante, esta desventaja no se considera relevante para el tipo de aplicación al que se destina el proyecto, donde principalmente se quiere trabajar con la activación.

5 Conclusión y Trabajo Futuro

Tras la realización del proyecto, se puede concluir que ya no hay vuelta atrás, y las tecnologías IoT / Hogar Inteligente tienen un futuro prometedor para este tipo de aplicaciones, debido fundamentalmente al bajo coste, la mejora en la facilidad de instalación para determinados casos, y el beneficio que puede suponer en su uso para múltiples aplicaciones, en concreto, aquellas que promueven un ahorro económico y de recursos (p.ej. de proyectos como ENEFF-PILOT). El inconveniente se presenta, cuando la facilidad de uso aumenta a costa de la privacidad de los usuarios, y es ahí donde se presenta el verdadero reto: equilibrar la balanza.

Tras lidiar con varias de las dificultades técnicas encontradas, es fácil imaginar la dificultad para los usuarios finales de comprender el nivel de exposición de su privacidad con los productos que se adquieren, ya que no existe ninguna normativa clara al respecto que permita identificar claramente el nivel de privacidad que se tiene con el uso o integración de un dispositivo concreto, incluso en muchos casos, es extremadamente complejo identificar al fabricante de los dispositivos y sus chips. Y esta tendencia sólo va en aumento, con los nuevos estándares que tratan de eliminar el uso de *HUBs* y permitir la comunicación directa entre los dispositivos IoT y las diferentes plataformas.

Sin embargo, existe una vibrante comunidad *opensource* que, aunque no puede competir con el "gran mercado" de Apple, Google, y demás fabricantes de *Gateways* comerciales, sí que ofrece una alternativa real para aquellos más preocupados por su privacidad, así como al florecimiento de un ecosistema de empresas que construyan proyectos, soluciones y plataformas apoyándose en plataformas de fuentes abiertas. Así ha quedado demostrado con el uso de OpenHAB, y la flexibilidad que este ha proporcionado para integrarse con otros módulos (MQTT, Zigbee, Z-Wave).

Finalmente, es importante aclarar que algunos fabricantes (cómo Dresden Elektronik) también fabrican productos específicos pensados para operar sin conexión a internet, que promueven la privacidad, incluso cuando éstos proveen de software privativo. Y, por otro lado, existen fabricantes que emplean estándares como MQTT o Zigbee, y no necesariamente promueven la privacidad (cómo Tuya), obligando al usuario a que sus dispositivos se conecten a la plataforma centralizada del fabricante.

A continuación, se incluyen posibles líneas futuras para este proyecto, algunas de ellas identificadas como interesantes y otras como necesarias para un sistema que debiera entrar en producción.

Controlador consulta la programación

En un entorno real, el controlador se encontrará en una red local a la que un sistema externo no podrá ni debería tener acceso directo. Ello sin contar con la complejidad de configuración que puede suponer a un usuario sin conocimientos IT. Por ello, se propone invertir los papeles, y que sea el controlador el que, mediante el protocolo establecido HTTP o MQTT consulte o reciba las programaciones cuando éstas se publiquen, en un servidor MQTT externo en el segundo caso. OH se suscribiría a un servidor MQTT de la misma forma que se suscribe al servidor MQTT desplegado en la propia RPI para comunicar con los dispositivos.

Procedimiento para instalación mejorado

Tal y cómo se ha comentado a lo largo de la memoria, entre los objetivos buscados se encuentran la simplicidad de instalación. Sería muy interesante empaquetar el sistema completo en un Snap, de tal forma que pudiese instalarse en dos pasos:

- Instalación del SO Ubuntu Core correspondiente
- Instalación del Snap ENEFF-PILOT

El Snap ENEFF-PILOT vendría configurado con todo lo necesario para utilizar el HW ya validado para este proyecto, y tendría la ventaja de que podría distribuirse directamente desde la Snap Store. Además, incorporaría todo el software necesario para cada versión publicada, se actualizaría automáticamente, y sería mucho más sencillo para el usuario final.

Con dicho Snap ya disponible, además se podría dar un paso más, y crear una imagen propia de Ubuntu Core para su distribución[27].

Software complementario a OpenHAB

En el capítulo 2 se realizaba una breve exposición de algunas plataformas software que compiten entre sí. No obstante, también existen algunos módulos más especializados y menos genéricos, que permitirán ampliar las capacidades de dichas plataformas.

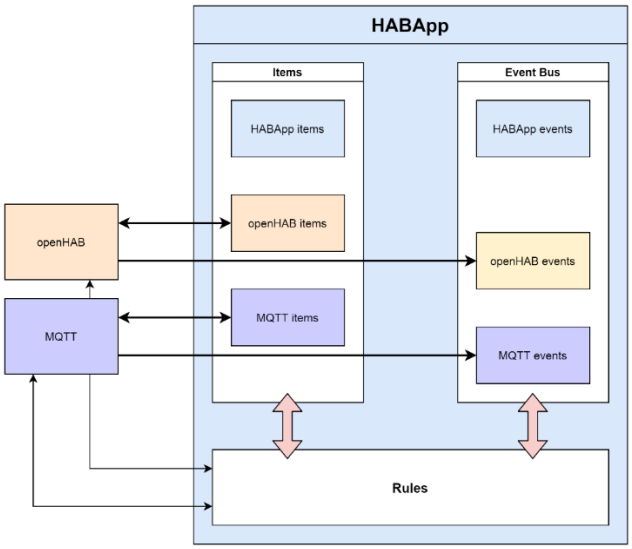

Figura 20 Alternativa HABApp para definir reglas y lógicas. Fuente:Web HABApp

Para fomentarlo, las plataformas ponen a disposición *APIs* públicas y *SDKs*. Es así como un software denominado HABApp[28] ha logrado destacar, y permite ampliar las capacidades de automatización y scripting de OH, entre otros, al especializarse en la tarea de la creación de reglas.

Se propone por lo tanto la posibilidad de estudiar el motor de reglas basado en Python que ofrece HABApp, y utilizar OH como sistema para modelar dispositivos y cómo plataforma central de comunicaciones, delegando en HABApp aquellas lógicas que deban implementarse (Figura 20).

Uso de las capas de seguridad en protocolos Zigbee, Z-Wave, MQTT, HTTP

Todos los protocolos utilizados para el proyecto han sido incluidos pensando en la seguridad y privacidad, al igual que los diferentes módulos software, sin embargo, no era el objetivo principal del proyecto el estudio y configuración de las capas de seguridad, lo que sin duda supone una ardua tarea y requiere de un estudio específico para ver las mejores combinaciones y estrategias, las cuales se deberán emplear sin duda alguna en proyectos reales.

[27] https://ubuntu.com/core/docs/quick-start
[28] << HABApp es un motor de reglas de Python para la automatización del hogar. Tiene elementos locales, un bus de eventos y puede integrar sistemas externos, por ejemplo. openHAB y MQTT….. hay un programador disponible que hace que la activación basada en el tiempo sea muy fácil.>> [En línea] (inglés) https://habapp.readthedocs.io/en/latest/about_habapp.html

La principal contrapartida en el aumento de capas de seguridad es que añaden una complejidad adicional para la configuración de los dispositivos de cara a un usuario sin conocimientos específicos IT. Especialmente si se quiere prescindir de plataformas en la nube de terceros.

Uso de protocolo binario en *payload*

El proyecto actual utiliza una API REST, basada en HTTP, razón por la cual no tenía sentido optimizar el *payload*, máxime teniendo en cuenta que el sistema está previsto para comunicar una única vez al día, y tan sólo el establecimiento de la conexión TCP es órdenes de magnitud mayor que el volumen de datos a transmitir.

No obstante, de cara a futuras implementaciones, donde el intercambio de datos sea continuo, por protocolo eficiente como puede ser MQTT (suponiendo una conexión permanente) para comunicar programador y controlador, se recomienda que el mensaje y la información de los vectores, esté codificado en binario (Google IoT, 2018 [en línea] septiembre 2022 https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols) .

Compatibilidad con otros protocolos

Teóricamente, sería posible dotar a la RPI de otros protocolos adicionales, que también sean compatibles con OpenHab.

A modo de ejemplo, la compañía Google liberó y promociona desde 2017 una implementación libre de Thread denominada OpenThread, la cual utiliza en sus dispositivos Nest. Thread es un protocolo basado en IEEE 802.15.4 que comparte espectro que Zigbee. Existen adaptadores[29] en el mercado, que son compatibles tanto con Zigbee como con Thread. La idea de Thread consiste en facilitar la puesta en funcionamiento de nuevos dispositivos IoT (Morales et al, 2016). Por otro lado, Thread está detrás de Matter[30], que varios fabricantes de electrodomésticos y otros dispositivos ya lo han incluido en su estrategia.

Alternativamente, existen otros protocolos como Bluethoot LE, además de los mencionados Wifi, siendo estos últimos los más complejos de integrar tal y cómo se ha expuesto en el Anexo C.

[29] JN5189 USB Dongle [en línea] https://www.nxp.com/products/wireless/thread/jn5189-usb-dongle-for-zigbee-and-thread-network:OM15080-JN5189

[30] Artículo sobre Thread y Matter [en línea] https://www.theverge.com/23165855/thread-smart-home-protocol-matter-apple-google-interview

Bibliografía

AdaFruit Industries. Adding a Real Time Clock to Raspberry Pi [en línea]. Último acceso en septiembre de 2022. https://cdn-learn.adafruit.com/downloads/pdf/adding-a-real-time-clock-to-raspberry-pi.pdf

Adai Mohammed Almomani 1, Mohd Nordin Abdul Rahman (2022, January). Adoption of IoT Smart Homes in Jordan: The Role of Trust and IT Knowledge. *In 2022 International Journal of Contemporary Management and Information Technology (IJCMIT)* (Volume 2, No 2, January 2022, pp. 24- 31, e-ISSN: : 2773-5036)

Ahmed, M. S., Mohamed, A., Homod, R. Z., Shareef, H., Sabry, A. H., & Khalid, K. B. (2015, December). Smart plug prototype for monitoring electrical appliances in Home Energy Management System. In *2015 IEEE Student Conference on Research and Development (SCOReD)* (pp. 32-36). IEEE.

Baig, F., Mahmood, A., Javaid, N., Razzaq, S., Khan, N., & Saleem, Z. (2013). Smart home energy management system for monitoring and scheduling of home appliances using zigbee. *J. Basic. Appl. Sci. Res*, *3*(5), 880-891.

Bluetooth® wireless technology [en línea]. Último acceso en mayo de 2022. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Chavali, P., Yang, P., & Nehorai, A. (2014). A distributed algorithm of appliance scheduling for home energy management system. *IEEE Transactions on Smart Grid*, *5*(1), 282-290.

Chen, C. F., Nelson, H., Xu, X., Bonilla, G., & Jones, N. (2021). Beyond technology adoption: Examining home energy management systems, energy burdens and climate change perceptions during COVID-19 pandemic. Renewable and Sustainable Energy Reviews, 145, 111066.

Driss Iounes, Juan Manuel López-Torralba, Pablo Pico-Valencia2, Juan Antonio Holgado-Terriza Construyendo un Dispositivo de Internet de las Cosas para el Hogar Conectado. *Jornadas SARTECO 2019, Septiembre, Cáceres*, 655-662.

Eclipse Mosquitto (2022, septiembre). MQTT server with Eclipse License [en línea]. Último acceso en septiembre 2022. https://mosquitto.org/

ENEFF-PILOT. Proyecto de investigación del Departamento de Electrónica de la Universidad de Alcalá [en línea]. Último acceso en septiembre de 2022. https://eneffpilot.web.uah.es

Statcounter (2022, septiembre). Statcounter Search Engine Market Share. [en línea]. Último acceso en septiembre de 2022. https://gs.statcounter.com/search-engine-market-share

HABApp. Software de automatización para crear reglas programables interoperable con OpenHAB y MQTT [en línea]. Último acceso en septiembre de 2022. https://habapp.readthedocs.io

Han, J., Choi, C. S., Park, W. K., Lee, I., & Kim, S. H. (2014). Smart home energy management system including renewable energy based on ZigBee and PLC. IEEE Transactions on Consumer Electronics, 60(2), 198-202.

Home Assistant. Open Source Home Automation [en línea]. Último acceso en septiembre de 2022. https://www.home-assistant.io

IoBroker Smarthome. Open Source Automation Platform [en línea]. Último acceso en septiembre de 2022. https://www.iobroker.net

Jussery Estefhany del Carmen Quillay Huertas (2020), “Análisis de los datos de Google Trends para predecir la demanda turística en Barcelona”, Memoria del Trabajo de Fin de Grado. p 6. http://hdl.handle.net/11201/153133

KNX Vivienda Inteligente [en línea]. Último acceso en septiembre de 2022. https://www.knx.org/knx-es/para-su-vivienda/

Liu, Y., Peng, G., Hu, L., Dong, J. and Zhang, Q. (2020), "Using Google Trends and Baidu Index to analyze the impacts of disaster events on company stock prices", Industrial Management & Data Systems, Vol. 120 No. 2, pp. 350-365. https://doi.org/10.1108/IMDS-03-2019-0190

Morales, Joshua & Lopez, Nicole Angelyn & Parado, Justin & Pasaoa, John. (2016). A Comparative Study of Thread Against ZigBee, Z-Wave, Bluetooth, and Wi-Fi as a Home-Automation Networking Protocol.. 10.13140/RG.2.2.36693.22249.

OpenHAB Open Source Automation Software for your home [en línea]. Último acceso en septiembre de 2022. https://www.openhab.org

Ozturk, Y., Senthilkumar, D., Kumar, S., & Lee, G. (2013). An intelligent home energy management system to improve demand response. *IEEE Transactions on smart Grid*, *4*(2), 694-701.

Parocha, R. C., & Macabebe, E. Q. B. (2019, November). Implementation of home automation system using OpenHAB framework for heterogeneous IoT devices. In *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)* (pp. 67-73). IEEE.

Saxena, S., Jain, S., Arora, D., & Sharma, P. (2019, March). Implications of MQTT connectivity protocol for iot based device automation using home assistant and OpenHAB. In 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 475-480). IEEE.

Simbaña, Santiago & Parra, Fidel & Chimarro, Juan & Urdaneta, Maryory. (2021). Sistema de seguimiento de requerimientos, eventos e incidentes para los clientes de la empresa TELCONET S.A en la ciudad de Quito.

Sowah, R. A., Ofoli, A. R., Tetteh, M. K., Opoku, R. A., & Armoo, S. K. (2018, August). Demand side management of smart homes using OpenHAB framework for interoperability of devices. In *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)* (pp. 1-8). IEEE.

Sowah, R. A., Boahene, D. E., Owoh, D. C., Addo, R., Mills, G. A., Owusu-Banahene, W., ... & Sarkodie-Mensah, B. (2020). Design of a secure wireless home automation system with an open home automation bus (OpenHAB 2) framework. *Journal of Sensors*, *2020*.

X10 Home Automation [en línea]. Último acceso en septiembre de 2022. https://www.x10.com

Zhao, Z., Lee, W. C., Shin, Y., & Song, K. B. (2013). An optimal power scheduling method for demand response in home energy management system. *IEEE transactions on smart grid*, *4*(3), 1391-1400.

Zhou, B., Li, W., Chan, K. W., Cao, Y., Kuang, Y., Liu, X., & Wang, X. (2016). Smart home energy management systems: Concept, configurations, and scheduling strategies. *Renewable and Sustainable Energy Reviews*, *61*, 30-40.

Gislason, D. (2008). Zigbee wireless networking.

Z-Wave Wireless Solutions for the Smart Home, MDUs and Hospitality. Silicon Labs [en línea] Último acceso en mayo de 2022. https://www.silabs.com/wireless/z-wave

Anexo A. Planos y diagramas

Desde un punto de vista general, se incluye aquí información útil de la configuración de los dispositivos que están montados en la maqueta. La idea de es que simplemente con la información aquí incluida sea posible enviar programaciones, realizar modificaciones en el sistema.

En primer lugar, se muestra la arquitectura de red del proyecto (Figura 21), incluyendo la parte opcional LAN_CASA y el acceso a Internet. Nótese, que la IP del router_wifi en LAN_CASA podría variar en función de la red externa, pues la misma será asignada por DCHP. Durante la realización del proyecto, se asignó una IP fija por DHCP, tal y como se muestra en la Figura 21.

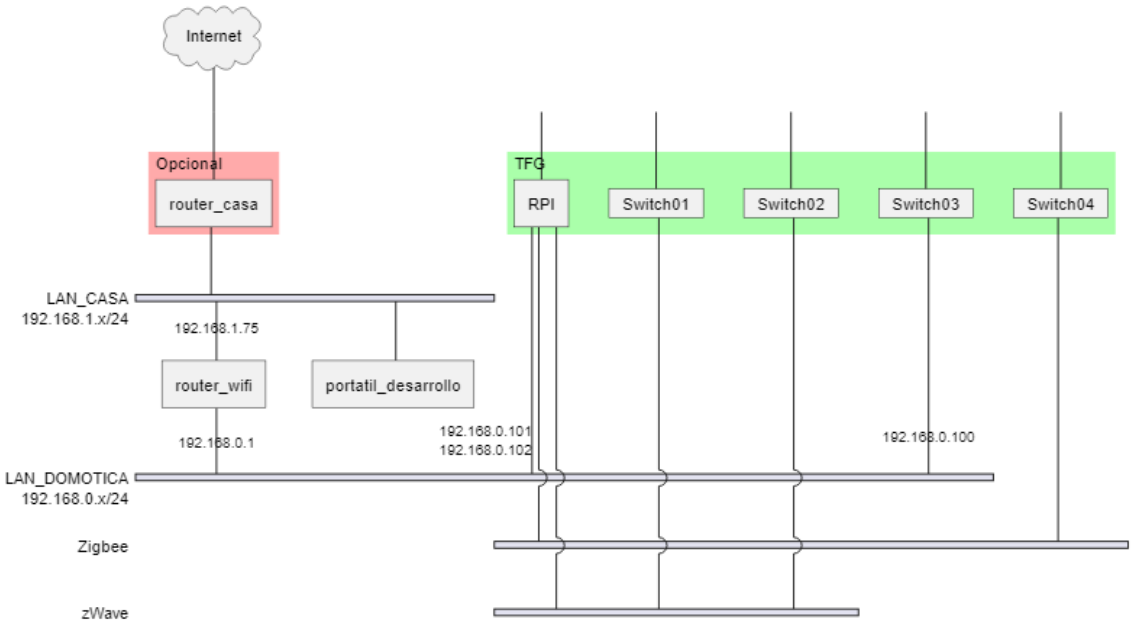

Figura 21 Arquitectura de comunicaciones

La conexión a nivel físico entre el router_wifi y la RPI puede realizase tanto Wifi como por cable. Dependiendo del caso, la IP de la RPI dentro en la red LAN_DOMOTICA tendrá una IP diferente, las cuales están fijadas para asignarse por DHCP. Ello implica que un cambio de RPI para la maqueta requiere de una modificación de esta configuración en el router_wifi.

En cuanto a la configuración del propio router_wifi que se incluye en la maqueta, el mismo tiene habilitada la gestión desde la WAN protegida por contraseña

Tabla 8 Conexiones abiertas en router_wifi para acceso desde LAN_CASA

ID	Puerto abierto	Puerto interno	Dirección IP	Protocolo	Tipo conexión RPI	Aplicación
1	18080	8080	192.168.0.102	TCP	RPI Ethernet	OpenHAB
2	10022	22	192.168.0.102	TCP	RPI Ethernet	ssh
3	18081	8080	192.168.0.101	TCP	RPI Wifi	OpenHAB
4	10023	22	192.168.0.101	TCP	RPI Wifi	ssh

Por último, con el fin de poder acceder desde redes externas al sistema para facilitar el desarrollo, el router_wifi puede conectarse a dicha red externa a través del conector WAN, y por otro lado, tiene configurado NAT para el acceso a la RPI. En total, son 2 los puertos abiertos, aunque se requiere de 4 reglas para contemplar las dos posibles IP de la RPI. La Tabla 8 contiene el detalle de las conexiones

Anexo B. Manual de usuario

La documentación de la versión OpenHAB 3.1, es amplia y muy completa, y puede encontrarse tanto en su página Web cómo en su repositorio de GitHub. Por este motivo, se incluyen en este apartado las instrucciones, y los pasos que no son generales de OH. Para el resto, se hará referencia a la documentación oficial, incluyendo el link en el momento de la elaboración de este documento.

Repositorio Documentación: https://github.com/openhab/openhab-docs
Docuemntación Online v3.1: https://v31.openhab.org/docs/

Acceso a la web de OpenHAB para consultar estado de los dispositivos y su programación

Al conectar la maqueta, puede ser útil acceder a la página Web de OH, donde se muestran el estado de los dispositivos ZW y ZB, no se requiere autenticación.

Si se necesita información adicional, como ver la configuración actual del sistema, entonces se deberá acceder con las credenciales de administrador.

El acceso es diferente en función la red en la que se encuentre el cliente, LAN_DOMOTICA o LAN_CASA donde debe utilizarse la IP y el puerto indicadas en el Anexo A. A modo de ejemplo, para acceder desde LAN_DOMOTICA cuando la RPI está conectada por Wifi, el acceso se realizaría con la URL: http://192.168.0.101:8080. Una vez se accede, se mostrará pantalla como la de la Figura 22.

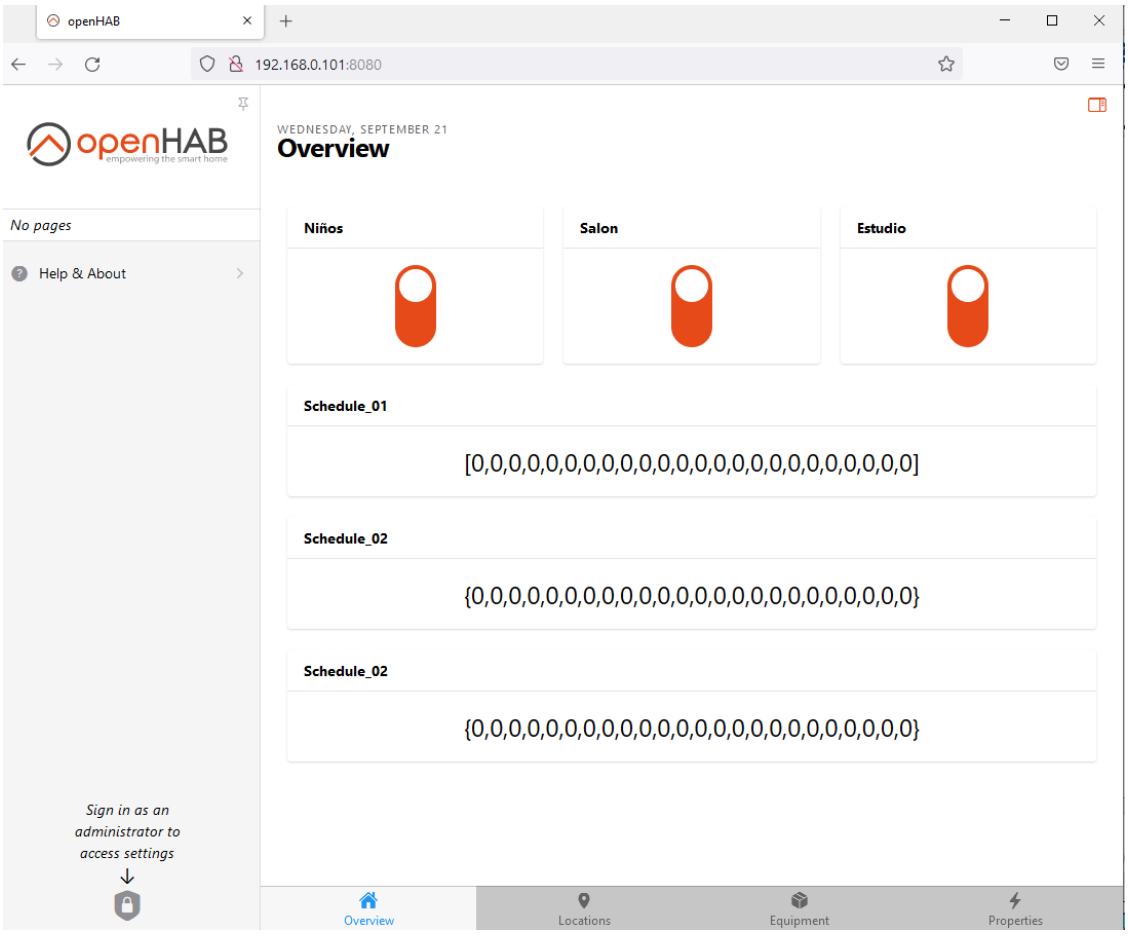

Figura 22 Ejemplo de página web principal

Obtener token para realizar llamadas a la API

El uso del sistema para enviar programaciones requiere de un paso previo, consistente en obtener un *token* válido con el usuario de OH. Dicho *token* es necesario para poder comunicarse con la API. El *token* estará asociado al usuario de OH y es recomendable utilizar uno para cada cliente que deba conectarse.

Instrucciones:
https://v31.openhab.org/docs/configuration/apitokens.html#generate-an-api-token

Una vez creado el *token* se mostrará en pantalla como la de la Figura 23 de donde debemos copiarlo ya que se requiere para realizar llamadas a la API de la forma que se indica en la memoria.

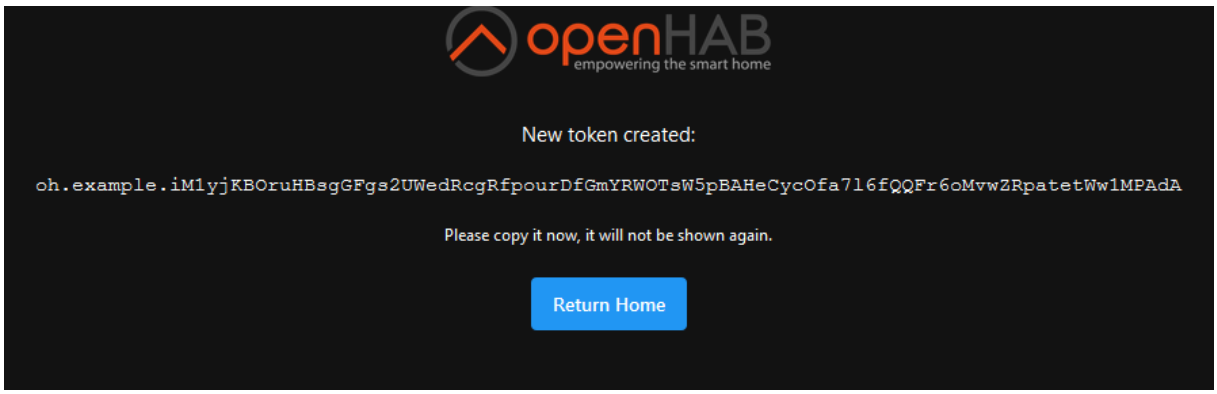

Figura 23 Página donde se muestra el token de la API de OH

Manual de instalación y configuración

Se asumen conocimientos previos de administración de sistemas y manejo de sistemas Linux como puede ser el uso de ssh, Docker o Visual Studio Code remote, así como conocimientos sobre el montaje de una RaspberryPi

Nota: esta guía no incluye aquellos pasos que ya se detallan en la guía de los diferentes productos empleados, o que no son específicos para este proyecto. Se incluyen, por lo tanto, todos los pasos que sirven para unir las diferentes partes, o que suponen una configuración específica. En el resto de casos se hace referencia a los manuales de los fabricantes y proveedores.

Conexión de dispositivos

Figura 24 Ejemplo de montaje RPI y adaptadores ZB y ZW

La única conexión cableada que se precisa es la alimentación para RPI (Figura 24). Mientras que los dispositivos adaptadores al medio para ZW y ZB no requieren de ninguna conexión especial, pudiendo ser conectados a cualquier puerto USB de los disponibles. La configuración se ha realizado, para que siempre se identifiquen por marca de fabricante en las configuraciones y Docker ha ayudado en esta labor (ver fichero "docker-compose.yml" para más información).

Los dispositivos para controlar hay que conectarlos y darles alimentación. Una vez el sistema se inicia, comenzará a recibir y enviar las programaciones.

Instalación imagen del proyecto en la tarjeta SD

Puede utilizarse la imagen ISO de la maqueta de demostración denominada sd_20220921.iso, para grabarse en una tarjeta SD y con ello, montar un sistema idéntico. Se trata de la forma más rápida de disponer de un sistema funcional, sin embargo, aún en este caso, es posible que sea necesario realizar algunos ajustes menores, para adaptar la configuración de la red. Si se quiere partir de la imagen en la SD será necesario cumplir alguno de los siguientes requisitos:

- Disponer de una red wifi con el mismo nombre y contraseña que la utilizadas para la demostración
- Una conexión mediante cable ethernet y DHCP a la RPI para acceder a la misma.

Una vez se tiene acceso a RPI se podrá modificar la red wifi preestablecida, si se desea. Para ello, pueden seguirse los pasos indicados en la documentación de Ubuntu Core, mediante el comando:

> 'sudo console-conf'

el cual realizará las preguntas necesarias para configurar el sistema. Para ello, primero deberá accederse por ssh al sistema.

El nombre de la red Wifi por defecto es: ONO2159

Acceso remoto al sistema utilizando SSH

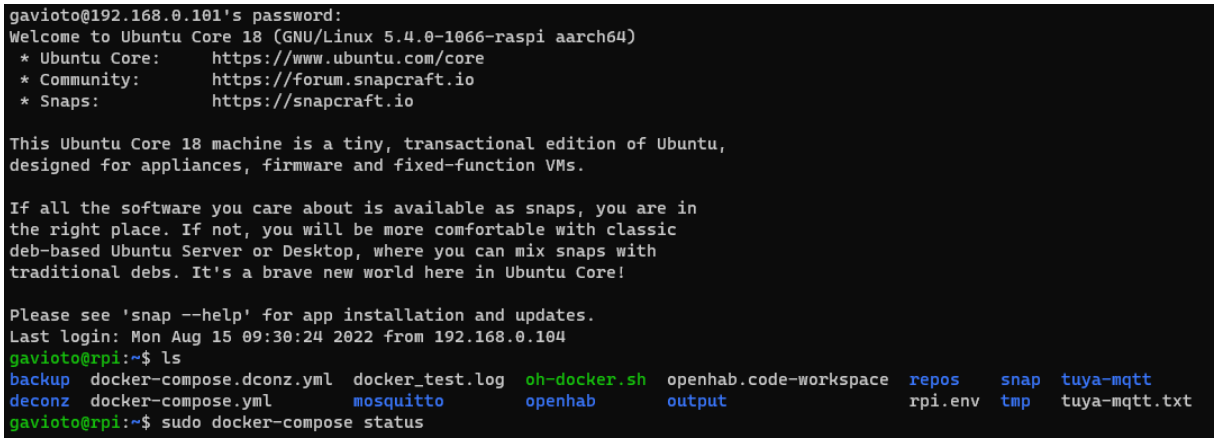

Figura 25 Pantalla ejemplo y carpetas de usuario

Nota: un sistema basado en Ubuntu Core que no sea de demostración normalmente no dispondrá de acceso por SSH por seguridad. En caso de requerirlo, deberá accederse físicamente a la tarjeta SD y habilitarlo. En la instalación actual está habilitado por conveniencia.

El acceso ssh es la única forma de acceder, ya que Ubuntu Core no dispone de interfaz gráfica. Por lo tanto, es la vía para realizar las siguientes tareas:

- Copia de seguridad
- Administrar Ubuntu Core
- Modificar contraseñas
- Arrancar y detener los servicios los contenedores
- Modificar los scripts de reglas de OH
- Realizar otras operaciones avanzadas, como actualizar versiones (no se incluyen en este manual)

Al acceder por SSH se mostrará una pantalla como la de la Figura 25. A partir de este momento, ya podemos realizar las diferentes tareas.

Estructura de carpetas

Las carpetas que contienen la información más importante son:

- ~/deconz todos los datos relativos a la aplicación DeconZ
- ~/mosquitto ficheros de configuración de Mosquitto (opcional)
- ~/openhab ficheros de configuración de OpenHAB
- ~/tuya-mqtt ficheros de configuración de tuya-mqtt (opcional)

El fichero “docker-compose.yml” describe el sistema de contenedores, las imágenes utilizadas, y otras configuraciones como los dispositivos, el acceso a la red y los volúmenes. En él, se definen las 4 carpetas anteriores como volúmenes mapeando de esta forma cada una de ellas con su respectivo contenedor para asegurar que la configuración se almacena en el *host*.

El resto de los ficheros que aparecen en la Figura 26, no son estrictamente necesarios para el funcionamiento, se han utilizado durante el proyecto para realizar tareas de desarrollo y se dejan para su consulta si se desea.

Un ejemplo de lo anterior sería la carpeta ~/repos, que contiene una copia descargada del repositorio tuya-mqtt de GitHub, que se utilizó crear la imagen del contenedor tuya-mqtt para poner en funcionamiento la bombilla Wifi.

```
gavioto@rpi:~$ ls -l
total 64
drwxrwxr-x 2 gavioto gavioto 4096 Jul 12  2021 backup
drwxrwxr-x 2 gavioto gavioto 4096 Sep 21 02:07 deconz
-rw-rw-r-- 1 gavioto gavioto  856 Feb 13  2022 docker-compose.dconz.yml
-rw-rw-r-- 1 gavioto gavioto 2812 Aug 15 15:59 docker-compose.yml
-rw-rw-r-- 1 gavioto gavioto   76 Mar  3  2022 docker_test.log
drwxr-xr-x 2 gavioto gavioto 4096 Jun  1 03:25 mosquitto
-rwxrwxr-x 1 gavioto gavioto  668 Aug  2  2021 oh-docker.sh
drwxrwxr-x 6 gavioto gavioto 4096 Jul 13  2021 openhab
-rw-rw-r-- 1 gavioto gavioto   66 Mar 13  2022 openhab.code-workspace
drwxrwxr-x 2 gavioto gavioto 4096 Mar 13  2022 output
drwxrwxr-x 3 gavioto gavioto 4096 Aug  8 01:02 repos
-rw-rw-r-- 1 gavioto gavioto   92 Mar 13  2022 rpi.env
drwxr-xr-x 5 gavioto gavioto 4096 Mar 13  2022 snap
drwxrwxr-x 2 gavioto gavioto 4096 Mar 15  2022 tmp
drwxrwxr-x 2 gavioto gavioto 4096 Aug 15 16:08 tuya-mqtt
-rw-rw-r-- 1 gavioto gavioto  450 Aug  8 03:25 tuya-mqtt.txt
```

Figura 26 Listado de carpetas

Arranque y parada de los servicios utilizando Docker Compose

```
gavioto@rpi:~$ sudo docker-compose ps
  Name                 Command                State       Ports
-------------------------------------------------------------------
deconz      /start.sh                          Up
mosquitto   /docker-entrypoint.sh /usr ...     Up
openhab     /entrypoint gosu openhab t ...     Up (healthy)
tuya-mqtt   docker-entrypoint.sh node  ...     Up
```

Figura 27 Comando docker-compose para visualizar estado de contenedores

Para facilitar la gestión de contenedores y las dependencias entre ellos, se utiliza la herramienta Docker Compose, la cual se instala de automáticamente con Docker. Se trata de una herramienta muy potente para la gestión de contenedores, aunque en este apartado simplemente se detallan los dos comandos necesarios para arrancar y detener contenedores. Ver Figura 27 con el estado arrancado de todos los contenedores.

Si se desean arrancar o detener contenedores, puede hacerse mediante el comando start o stop, seguido del nombre del contendor (Figura 28). En el caso de que no se especifique ningún contenedor, éste aplicará a todos los 4.

```
gavioto@rpi:~$ sudo docker-compose start <nombre_contenedor>
```

Figura 28 Arrancar un contenedor

Copia de seguridad de OpenHAB

Aunque el sistema requiere de todos sus módulos para funcionar, sin duda el que tiene la configuración más compleja es OpenHAB. Por ello, se recomienda realizar una copia de seguridad si se realizan cambios.

Las copias de seguridad de OpenHAB facilita restaurar dicha configuración en otro entorno con una configuración de versión igual o superior, al menos en lo que respecta a las versiones 3.x. No obstante, se recomienda revisar las notas de cada versión por si hubiese alguna incompatibilidad.

```
> sudo docker exec -it  openhab /bin/bash
> /openhab/runtime/bin/backup

Using '/openhab/conf' as conf folder...
Using '/openhab/userdata' as userdata folder...
Using '/openhab/runtime' as runtime folder...
Using '/openhab/userdata/backup' as backup folder...
Writing to '/openhab/userdata/backup/openhab-backup-21_08_02-07_05_41.zip'...
Making Temporary Directory if it is not already there
Using /tmp/openhab/backup as TempDir

########################################
        openHAB backup script
########################################

Copying configuration to temporary folder...
Removing unnecessary files...
Zipping folder...
Removing temporary files...
Success! Backup made in /openhab/userdata/backup/openhab-backup-21_08_02-07_05_41.zip
```

Figura 29 Realización copia de seguridad de OpenHAB

La forma de proceder es mediante la ejecución de un comando dentro del contenedor de OpenHAB, tal y cómo se muestra en la Figura 29, haciendo uso del comando "exec" de Docker. De este modo, el comando se ejecutará dentro del contendor, y el fichero se copiará a la carpeta indicada del contenedor. Dado que la configuración de OH se almacena en un volumen de Docker que está mapeado en el host, se puede acceder a la misma copia de seguridad desde la ruta local ~/openhab/userdata/backup.

Configuración de *gateways* en OpenHAB

La conexión de los adaptadores/*gateways* ZB y ZW no requieren de ninguna conexión específica, pudiendo conectarse a cualquier de los puertos USB disponibles en RPI4. El sistema los reconocerá por su identificador del modelo y los mapeará correctamente como puertos serie en el contenedor de OpenHAB y DeconZ. El mapeo realizado, que se describe en el fichero Docker-compose.yml en la propiedad "devices" se puede ver en la y en

```
  deconz:
    image: marthoc/deconz:aarch64-2.07.01
#    image: marthoc/deconz:stable
    container_name: deconz
    network_mode: host
    restart: always
    volumes:
      - /home/gavioto/deconz:/root/.local/share/dresden-elektronik/deCONZ
    devices:
      - "/dev/serial/by-id/usb-dresden_elektronik_ingenieurtechnik_GmbH_ConBee_II_DE2121494-if00:/dev/ttyAMA0"
```

Figura 30 Mapeo de adaptador Zigbee en contenedor DconZ

```
  openhab:
    image: openhab/openhab:3.1.0-debian
    network_mode: host
    container_name: openhab
    restart: always
    devices:
#       - $PORT_HOST:${PORT}
      - /dev/serial/by-id/usb-0658_0200-if00:/dev/ttyACM1
```

Figura 31 Mapeo de adaptador Z-Wave en contenedor OH

Configuración de nuevos dispositivos ZB y ZW

Se deben seguir las instrucciones del fabricante, que normalmente implican poner el dispositivo en modo "broadcast" para que OpenHAB que estará escuchando a través de los Gateways pueda reconocerlos y configurarlos.

Anexo C. Dispositivos Wifi de la plataforma Tuya y privacidad

Explicación Plataforma Tuya [en línea]. Septiembre 2022 https://pypi.org/project/tinytuya/

La plataforma Tuya es proveedora de hardware y software a diferentes fabricantes de dispositivos inteligentes. La misma es fácilmente identificable porque todos los dispositivos compatibles incluyen alguno de los distintivos "Smart Life App" y "Tuya App".

Figura 32 Aspecto bombilla Alecto Smart Bulb 10

Los dispositivos WIFI basados en tecnología del fabricante Tuya están preparados para comunicarse con su plataforma, un valor añadido para el integrador y vendedor del producto final, como es el caso de la marca Alecto [31] (Fabricante HESDO BV). Una empresa de 12 empleados según LinkedIn[32] que fabrica un producto con marca propia, pero basado en una tecnología de terceros (Tuya) y que por lo tanto emplea dicha tecnología.

Para el caso de las bombillas, y según datos recopilados en Internet, muchos de estos dispositivos basados en Tuya (de diferentes integradores), están basados en los mismos chips ESP8266 o compatibles, por lo tanto, utilizan en su mayoría el mismo Firmware de Tuya, con un identificador para cada integrador, lo que permite diferenciarlos cuando estos, irremediablemente se conectan a la plataforma en la nube de Tuya antes de que el usuario final pueda operar con ellos offline.

[31] Web de Alecto [en línea]. Último acceso septiembre 2022 https://alectohome.com/pages/impressum
[32] Web de Alecto en LinkedIn [en línea]. Último acceso septiembre 2022 https://www.linkedin.com/company/hesdo-bv/?originalSubdomain=es

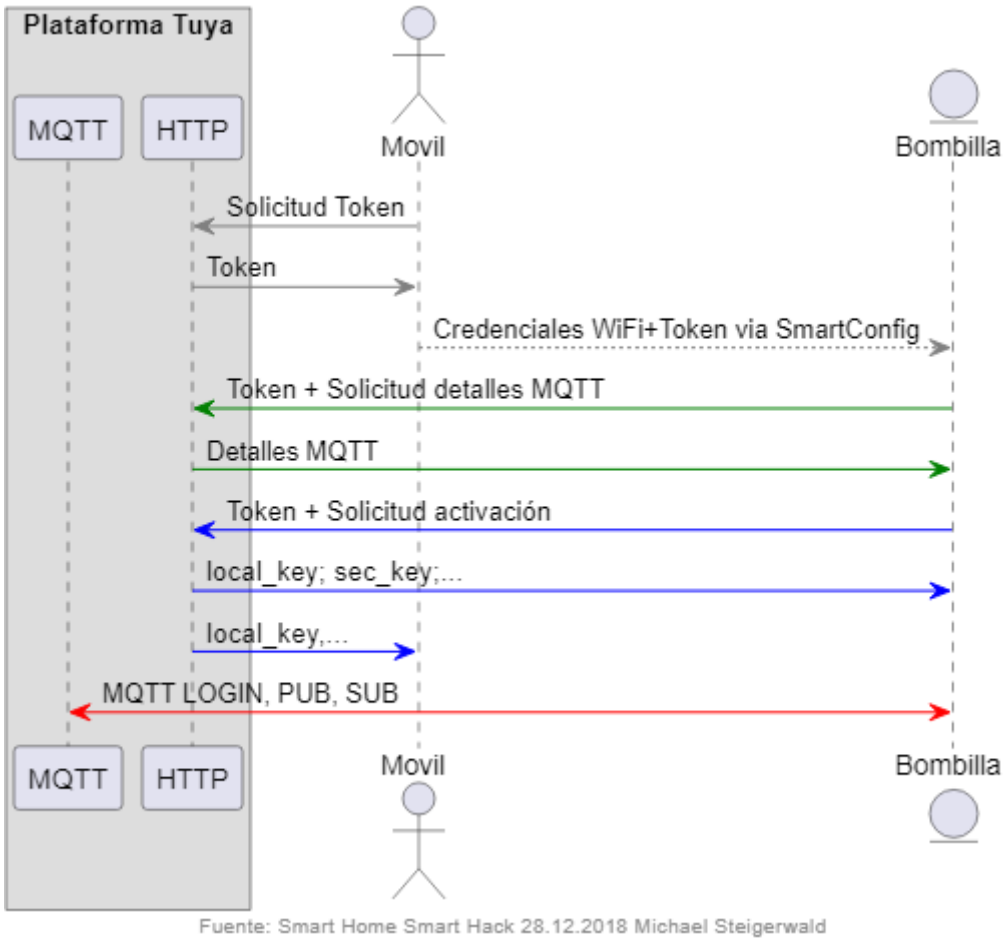

Figura 33 Secuencia de conexión dispositivos Tuya. Fuente: Smart Home Smart Hack

La comunicación con un dispositivo Tuya requiere conocer una clave local (*local key*) que es generada por la plataforma. Incluso aunque el control se quiera realizar en local, el proceso de registro es obligatorio, y requiere una conexión a Internet.

En la Figura 33 se describe la secuencia de registro de un dispositivo. Una vez el dispositivo se conecta a la plataforma, genera una clave única que finalmente debe ser utilizada por cualquier equipo que quiera comunicarse con el dispositivo. Dicha clave cambiará, cada vez que el dispositivo cambia de red. Según lo anterior, las dos únicas formas de obtenerlas son:

- Registrar el dispositivo para posteriormente descargarla de la plataforma Tuya, la cual conoce dicha clave en todo momento y está asociada al fabricante final del dispositivo.
- obtener la clave interviniendo las comunicaciones mediante mecanismos MITM[33].

La obligatoriedad de tener que conectarse a la plataforma del fabricante para poder funcionar supone un grave riesgo para la privacidad, pues estos dispositivos pueden acceder a la plataforma cuando lo desean, y de hecho, mantienen un canal de comunicación directo y encriptado con la misma para poder recibir comandos e informar del estado. La única ventaja para el usuario es que puede ver el estado y enviar comandos desde su teléfono inteligente incluso cuando no se encuentra conectado a la red Wifi local donde están configurados los dispositivos, ya que un simple acceso a internet dará acceso a la API de Tuya y todos sus dispositivos configurados.

[33] Ataque de intermediario. Man In The Middle [en línea] septiembre 2022 https://es.wikipedia.org/wiki/Ataque_de_intermediario

Por otro lado, se deduce, que los dispositivos Wifi, los cuales están instalados en una red sin limitaciones para el acceso a Internet, se conectarán a la plataforma en la nube de forma cuando así lo decidan, en función de su programación, sin que además sea posible conocer que tipos de datos transmiten, cuestión de vital importancia, ya que, si no se configura de otro modo, dichos dispositivos tendrán acceso además a todos los dispositivos de la red local.

Todos estos dispositivos funcionan con Google Home y Apple Kit, y sus respectivos *Gateways*/*HUBs* y se configuran de forma relativamente sencilla. Aunque no se ha investigado como opera esta conexión, una posible hipótesis sería que Google y Apple se integran con la plataforma Tuya y se comunican con la misma para obtener la clave de acceso a los dispositivos.

Durante las pruebas para integrar las mencionadas bombillas en el presente proyecto, se encontró una presentación muy detallada[34] sobre la forma de funcionar de Tuya, y que fue el inicio de lo que hoy se conoce como Tuya-Convert, una iniciativa que, gracias a la ingeniería inversa realizada por los investigadores, permite reprogramar el chip ESP8266 con otro firmware. Dicho mecanismo y aplicación (Tuya-Convert) se utilizan por los usuarios finales para cargar el firmware de fuentes abiertas denominado Tasmota.

Tras la experiencia tratando de integrar bombillas de este fabricante, no se recomienda su uso, ni los chips preconfigurados que este pueda proveer a integradores cuando se quiera dotar a la solución de unos altos niveles de privacidad.

[34] Vtrust [en línea] septiembre 2022. https://media.ccc.de/v/35c3-9723-smart_home_-_smart_hack

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR