Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

EXTRACCIÓN DE CONOCIMIENTO DESCRIPTIVO EN UN PROTOTIPO DE CASA INTELIGENTE BASADA EN LA PLATAFORMA HOME ASSISTANT

Alumno: Sergio Bonachera Romero.

Tutores: Prof. D. Ángel Miguel García Vico.
Prof. D. Cristóbal José Carmona del Jesus.

Dpto: Departamento de Informática.

Julio, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

D. Ángel Miguel García Vico y D. Cristóbal José Carmona del Jesus, tutores del Trabajo Fin de Grado titulado:
Extracción de conocimiento descriptivo en un prototipo de casa inteligente basada en la plataforma HomeAssistant, que presenta Sergio Bonachera Romero, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, 20 de julio de 2021

El alumno: Los tutores:

CARMONA DEL JESUS CRISTOBAL JOSE - 75102603J

Firmado digitalmente por CARMONA DEL JESUS CRISTOBAL JOSE - 75102603J
Fecha: 2021.07.20 09:25:07 +02'00'

Firmado digitalmente por GARCIA VICO ANGEL MIGUEL - 26256679V el día 20/07/2021 con un certificado emitido por AC FNMT Usuarios

Sergio Bonachera Romero

Ángel Miguel García Vico

Cristóbal José Carmona del Jesus

Agradecimientos

A mis profesores de la Universidad de Jaén por enseñarme y guiarme en mi formación de esta etapa universitaria. En especial, a mis dos tutores Ángel y Cristóbal por sus sabios consejos y la constante atención que han dedicado sobre mí.

A Irene Padilla, Irene Moreno, Javier Martínez y Paul Andrade, porque sin ellos este largo camino hubiese sido mucho más difícil y hubiese aprendido mucho menos. Porque han resultado ser un pilar fundamental de mi vida y me he podido apoyar en ellos para crecer tanto académicamente como persona.

Por último, pero no menos importante, a mi familia por estar siempre ahí y acompañarme en los momentos más difíciles. Por facilitarme todo lo que han podido la vida y estar continuamente apoyándome en las decisiones que tomo.

Índice

Índice de Figuras

Índice de Tablas

Índice de Códigos

Capítulo 1.- Introducción y planificación.

En este primer capítulo se presentará en la sección 1.1 una motivación y una introducción sobre este trabajo fin de grado. Además, se presentará una declaración de objetivos del proyecto en la sección 1.2. Para terminar el capítulo se estimarán en la sección 1.3 tanto los costes temporales como materiales, junto con la planificación del proyecto.

1.1. Motivación.

Los entornos de interacción humana basados en sistemas de telecomunicaciones y control automático son cada vez más comunes, debido al avance que han experimentado estas áreas tecnológicas en los últimos años.

"Domótica" es el término científico que se utiliza para denominar esta parte de la ingeniería, que integra el control y supervisión de los elementos existentes en una vivienda. Estos elementos se clasifican principalmente en dos categorías: los sensores y los actuadores.

Los sensores se encargan de recoger datos del medio en el que se encuentran instalados. De esta manera, se encuentran distintos tipos de sensores para casi cualquier magnitud que se quiera medir, desde temperatura y humedad, hasta campos electromagnéticos.

Figura 1.- Esquema general del funcionamiento de los sistemas domóticos. Fuente: Elaboración propia.

Los actuadores son elementos que permiten al sistema interactuar con su entorno, modificándolo de alguna manera. Habitualmente, esta actuación sobre el entorno es provocada por la información previamente proporcionada por lo sensores. Es decir, si se cumplen los requisitos para que cierta acción se lleve a cabo, serán los actuadores los encargados de realizarla.

En estos últimos años ha habido un creciente interés en el mundo de la domótica al igual que las posibilidades que hay y lo prometedor que se espera que sea este campo de estudio en un futuro no muy lejano. De hecho, se estima que en este mismo año habrá 46 mil millones de dispositivos conectados (Sam Barker, 2020) y que para 2030 esa cifra aumente hasta los 125 mil millones de dispositivos (Heslop, 2019). Con esta cantidad de dispositivos, el campo de trabajo de la inteligencia artificial (IA) se vuelve muy interesante y puede jugar un papel clave.

La evolución de estos elementos se ve afectada por el auge de las telecomunicaciones, electrónica e informática. Además, el abaratamiento de los dispositivos y la explosión de nuevas plataformas y asistentes virtuales hacen que en estos últimos tiempos la domótica haya tenido una evolución muy rápida y prácticamente sin freno.

En este ámbito, la cantidad de información que se genera acerca del estado de una casa es muy grande. La IA se nutre de estos datos para extraer información y conocimiento útil para el usuario/experto. Y en concreto, una de las posibles aplicaciones basadas en IA que se pueden demandar en un futuro es la extracción de patrones simples de comportamiento, de modo que el usuario sea capaz de entender sus actividades cotidianas y ajustar el sistema domótico de acuerdo a éstas.

En definitiva, en este trabajo se incorporan dos temáticas interesantes: el empleo de algoritmos metaheurísticos para la extracción de conocimiento

y la construcción de un prototipo de sistema domótico mediante una herramienta orientada expresamente al desarrollo de las casas inteligentes o "*smart homes*[1]".

1.2. Objetivos.

El objetivo del proyecto es realizar la construcción e instalación de un prototipo de sistema domótico, del cual se extraerán reglas de conocimiento aplicando un algoritmo de minería de datos para poder crear una serie de automatizaciones en función del comportamiento de los habitantes de la vivienda.

Para alcanzar este objetivo principal, se definen los siguientes subobjetivos:

- Realizar una revisión de las distintas plataformas de domotización disponibles.
- Realizar una revisión del estado del arte con respecto a la extracción supervisada de conocimiento y otras técnicas de minería de datos.
- Creación de un prototipo de casa inteligente donde se instalen una serie de sensores y actuadores.
- Extraer conocimiento descriptivo de los datos generados por los sensores instalados.

Todo esto bajo unas condiciones concretas:

- **Bajo coste**: que la instalación del sistema no suponga un coste económico elevado. De esta manera, se buscarán las distintas alternativas y se estudiarán los diversos supuestos para determinar la viabilidad del proyecto.

[1] Son aquellas viviendas que incorporan un sistema que permite automatizar muchas tareas, así como tener un control total y en vivo sobre lo que ocurre en el hogar. (gruporp, 2020)

- **Procesamiento local**: dado que todos los datos con los que se va a trabajar pueden considerarse sensibles y transmiten mucha información personal en las distintas comunicaciones, se tenderá hacia un procesamiento que no requiera de envío de datos a sistemas que no se encuentran en la vivienda.
- **Tolerancia a fallos**: en algún momento se pueden producir fallos, ya sean por cortes en el suministro eléctrico, como por fallos en los propios dispositivos. De esta manera, se instalarán medidas para controlar estos riesgos y asegurar que los dispositivos estén siempre disponibles y que, ante un corte de suministro eléctrico, el sistema sea capaz de iniciarse automáticamente.

1.3. Planificación y estimación de costes.

Una vez determinados los objetivos de este proyecto, en este apartado se va a realizar un análisis y estudio referente a la viabilidad del proyecto, tanto en costes de tiempo como en coste de material necesario para la implantación del prototipo.

1.3.1. **Temporal.**

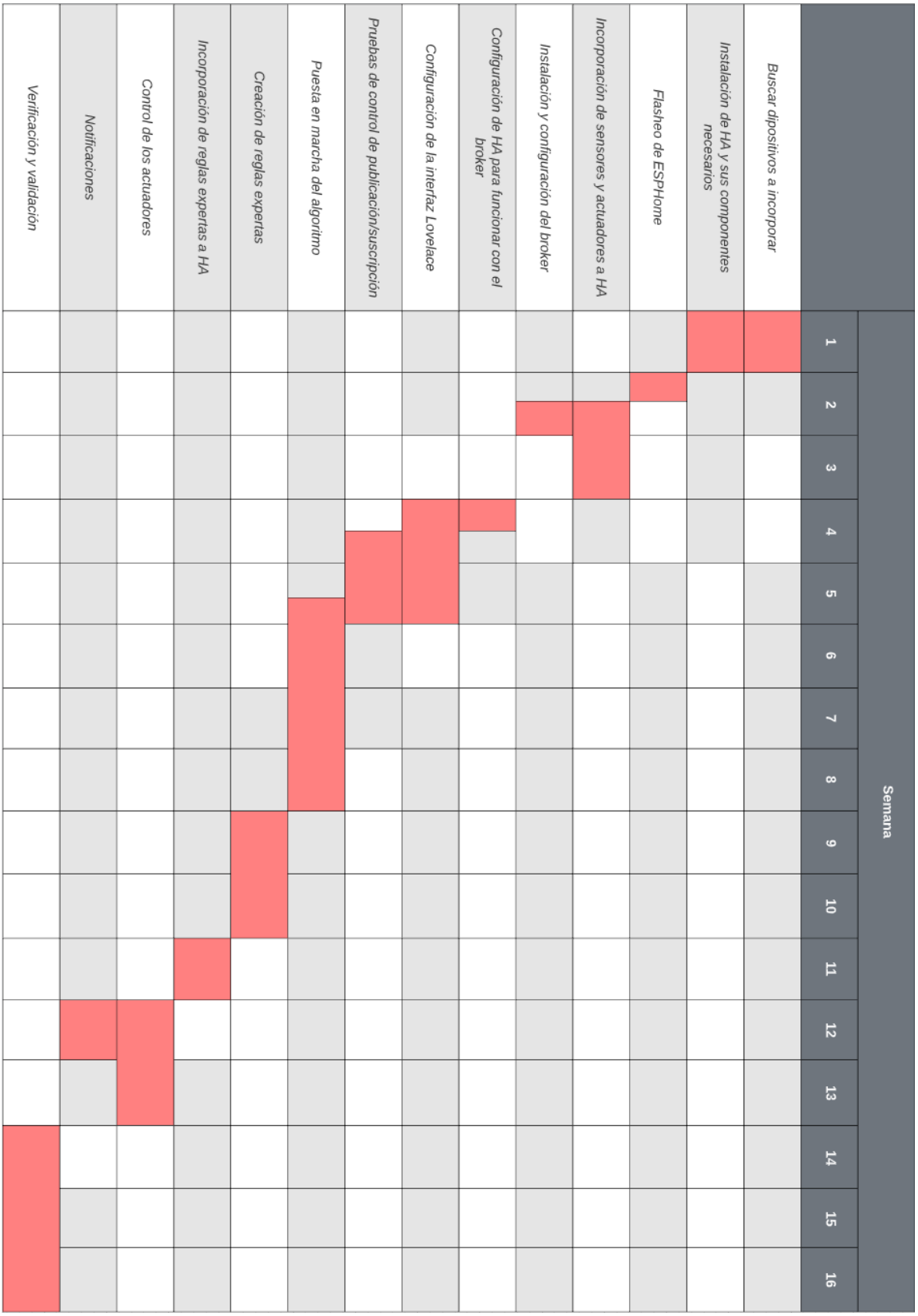

Figura 2.- Diagrama de Gantt. Fuente: Elaboración propia.

En la Figura 2 se refleja la asignación temporal de las tareas en las que ha sido dividido el proyecto con un nivel de abstracción moderado. Se han tomado 35 horas semanales de trabajo a 25 euros la hora.

En cuanto a los costes asociados a cada tarea se pueden observar en la Tabla 1.

Tarea	Duración	Coste Asociado
Buscar dispositivos a incorporar	1 semana	875€
Instalación de Home Assistant y sus componentes necesarios	1 semana	875€
Flasheo[2] de ESPHome	Media semana	437’50€
Incorporación de sensores y actuadores a Home Assistant	1 semana y media	1312’50€
Instalación y configuración del bróker	Media semana	437’50€
Configuración de Home Assistant para funcionar con el bróker	Media semana	437’50€
Configuración de la interfaz Lovelace	2 semanas	1750€
Pruebas de control de publicación/suscripción	1 semana y media	1312’50€
Puesta en marcha del algoritmo	3 semanas y media	3062’50€
Creación de reglas expertas	2 semanas	1750€
Incorporación de reglas expertas a Home Assistant	1 semana	875€
Control de los actuadores	2 semanas	1750€
Notificaciones	1 semana	875€
Verificación y validación	3 semanas	2625€
TOTAL	**16 semanas**	**14000€**

Tabla 1.- Relación de costes temporales.

[2] Flashear significa escribir en la ROM un código personalizado distinto al que incorpora de fábrica.

1.3.2. **Material.**

Para la construcción del prototipo es necesario material físico además del software utilizado. En la Tabla 2 se muestran recogidos los sensores y dispositivos que ha sido necesario adquirir junto con sus precios y unidades.

Artículo	Descripción	Coste Unitario	Unidades	Coste Total
BME280	Sensor de temperatura, humedad y presión	3’68€	4	14’72€
BH1750	Sensor de luz más potente. Para luz del sol.	1’95€	1	1’95€
FR-04	Sensor de lluvia	1’08€	1	1’08€
TEMT6000	Sensor de luminosidad interior	0’7€	4	2’81€
Senseair S8	Sensor de CO_2	25’24€	4	100’96€
Sonoff Basic R2	Placa 1 relé wifi	7’43€	1	7’43€
Emisor de infrarrojos	Emisor de frecuencia infrarroja para el control de dispositivos	1’03€	1	1’03€
USB TTL	USB TTL a 3.3 Voltios necesario para flashear Sonoff	1’92€	1	1’92€
Wemos D1 mini	Módulo ESP-8266 mini	1’996€	5	9’98€
Fuente de alimentación	Cables para alimentar los Wemos	3’29€	5	16’45€
Placa Protoboard	Placa para conectar los distintos dispositivos	0’96€	4	3’83€
Led KY-016	Dispositivo led RGB	0’171€	10	1’71€
Raspberry Pi 4 Modelo B 4 GB	Miniordenador	59’90€	1	59’90€
Tarjeta micro-sd 32 GB	Almacenamiento interno del miniordenador	4’32€	1	4’32€
			TOTAL	**228,09€**

Tabla 2.- Relación de costes materiales.

Capítulo 2.- Situación actual.

En este capítulo se presentará una revisión del estado del arte de la domótica en general, así como las distintas técnicas que están usándose actualmente tanto en ámbitos de domótica como en extracción de conocimiento y de análisis de Big Data. Además, se presentará también un estudio de viabilidad del proyecto en cuestión. Dicho estudio se explicará en la sección 2.1. En la sección 2.2 se muestra el estudio del estado del arte revisando los distintos frameworks actuales junto con sus principales características.

2.1. Estudio de viabilidad.

Como se ha visto en el capítulo anterior, el coste para disponer de un sistema domótico orientado a la recogida de datos ambientales y actuación sobre los dispositivos pertinentes no es excesivamente caro. A este presupuesto habría que añadir los actuadores, los cuales sustituirán a los leds RGB que se han escogido y que, una vez instalados realizarán las tareas físicamente que manden las reglas de conocimiento, extraídas a partir de los datos de los sensores.

Cabe destacar que, la implantación de estos actuadores durante la construcción de la vivienda es muchísimo más barata y fácil que una vez la casa ya está construida. Esto se debe a que los elementos que conforman el sistema, en su mayoría, se deben ocultar tras las paredes para mejorar la estética. Por tanto, cualquier cambio a posteriori en el sistema requerirá de obra. De aquí viene ese aumento de costo y de posibles problemas que generen más gasto que si se hiciese directamente junto con la vivienda.

Aun así, estos actuadores tampoco tienen un precio elevado, se mueven al igual que los sensores en unos rangos de precios bastante anchos dependiendo de lo que se busque y se quiera instalar. Por otra parte, introducir sensores nuevos es sencillo y tiene un coste bajo. En resumen,

el sistema es fácilmente escalable respecto a la implantación de nuevos sensores, mientras que para los actuadores la escalabilidad se reduce ligeramente.

Otro punto a detallar sería el beneficio que se obtendría, en relación al ahorro energético, de tener ciertos elementos que se puedan automatizar incorporados a la plataforma, como por ejemplo aires acondicionados, que, aunque no sean controlables mediante Internet se podría configurar un dispositivo emisor de infrarrojos para controlar el mando a distancia simulando el mismo efecto; u optimizar el uso de dispositivos apagándose cuando se dejan encendidos por descuido. Esto último también tiene un beneficio para el medio ambiente ayudando a contaminar menos.

El único problema o dificultad sería la actualización de viviendas. No obstante, la situación actual de expansión en las ciudades soluciona parcialmente este problema, ya que, la construcción de nuevas viviendas posibilita la incorporación de la instalación del prototipo en una fase temprana ahorrando todo el coste económico que supondría la actualización de la implantación de la domótica, como ya se ha visto anteriormente. Aunque por esta parte, también se soluciona de una manera más o menos eficaz al existir ayudas económicas por parte del gobierno en lo que adaptación o incluso adquisición de viviendas adaptadas se refiere. Además, existen incluso ayudas para la adquisición de viviendas por parte de parejas, gente joven y gente que vaya a adquirir su primera vivienda, entre otras.

2.2. Estado del arte.

Lo primero que se debe definir es un centro de automatización o "*Automation Hub*". Esto es un dispositivo inteligente que puede comunicarse con otros y conectarse a ellos para mandar órdenes a los mismos (blogdedomotica, 2019). Además, pueden hacer acopio y detección de los dispositivos de manera automática sin tener que introducirlos

manualmente. De esta manera, se presentará en el siguiente apartado los frameworks actuales y más relevantes.

2.2.1. Frameworks de domótica actuales.

2.2.1.1. OpenHAB.

OpenHAB (Open House Automation Bus) (OpenHAB, 2021) es una de las herramientas más conocidas y que goza de una amplia comunidad de usuarios activos y una gran cantidad de dispositivos compatibles con esta plataforma de Internet de las Cosas.

Este framework está escrito en Java, por lo que es compatible con la mayoría de los principales sistemas operativos. Además, trae consigo aplicaciones tanto para Android como para iOS para el control de los dispositivos que se integren. Con respecto a los elementos que se pueden integrar, está pensado para ser independiente de los mismos e incluso permite agregar módulos propios. Cuenta también con herramientas de diseño para poder crear tus propias interfaces de usuario. Con respecto a la seguridad cabe destacar que todo el procesamiento de datos se realiza de manera local.

Figura 3.- Vista previa OpenHab. Fuente: (domoticaencasa.es, 2019).

2.2.1.2. OpenMotics.

Este sistema domótico está bajo licencias de hardware y software de código abierto. La diferencia de éste con respecto a otros, es que se basa en una solución cableada. De esta manera, en vez de controlar muchos dispositivos de diferentes fabricantes se centra en la centralización de ellos mediante módulos hardware (OpenMotics, 2021).

OpenMotics también se combina con soluciones más modernas en la nube. Sin embargo, este tipo de sistema no es interesante en este proyecto por sus posibles implicaciones en la privacidad y seguridad de los datos.

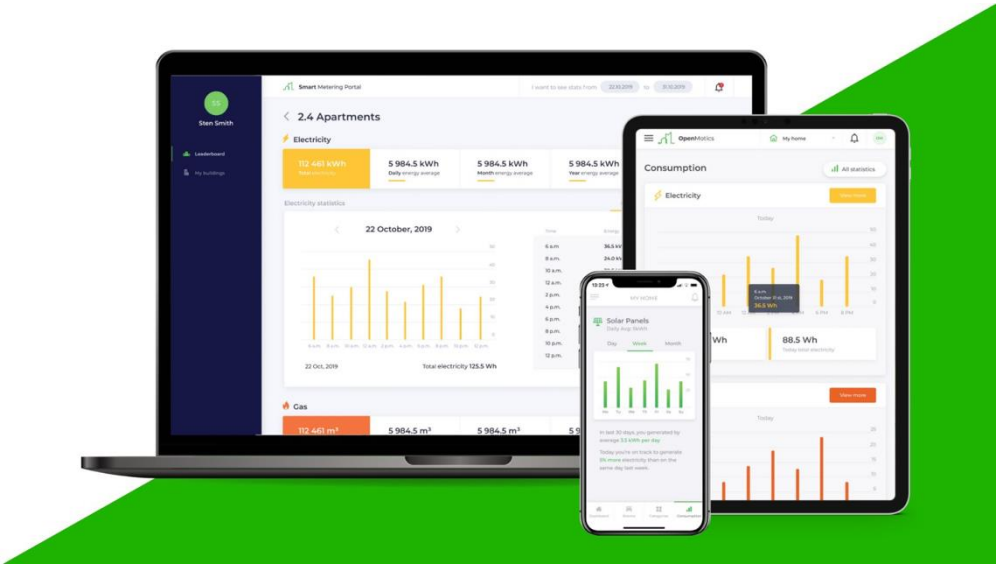

Figura 4.- Vista previa OpenMotics. Fuente: (OpenMotics, 2021)

2.2.1.3. Home Assistant.

Home Assistant es una plataforma de domótica de código abierto que se ejecuta sobre Python (blogdedomotica, 2019), por lo que se puede ejecutar en casi cualquier máquina que pueda ejecutar Python 3. Se encuentra entre los dispositivos más usuales que pueden usarse como servidor la Raspberry Pi, ya que es barata, potente y fácil de usar e instalar.

Este centro de automatización es capaz de detectar nuevos dispositivos de manera automática, elemento que le proporciona un gran valor adicional. Resulta muy cómodo tener que configurar únicamente ciertos dispositivos que, al venir de distintos proveedores, no sean detectables automáticamente, pero, la inmensa mayoría son ya autoconfigurables. Esto es consecuencia también de la gran expansión y las continuas actualizaciones que tiene la plataforma por parte de su comunidad. De hecho, es uno de los 10 repositorios más activos en GitHub (Home Assistant, 2021).

Otro punto a favor es que no está pensado para trabajar con la nube. Es decir, todos los datos se manejan dentro de la red local de la vivienda. Con esto, se aumenta notoriamente la seguridad y privacidad. Home

Assistant permite trabajar con VLAN y reglas del firewall para aumentar aún más la autenticación e integridad de los datos.

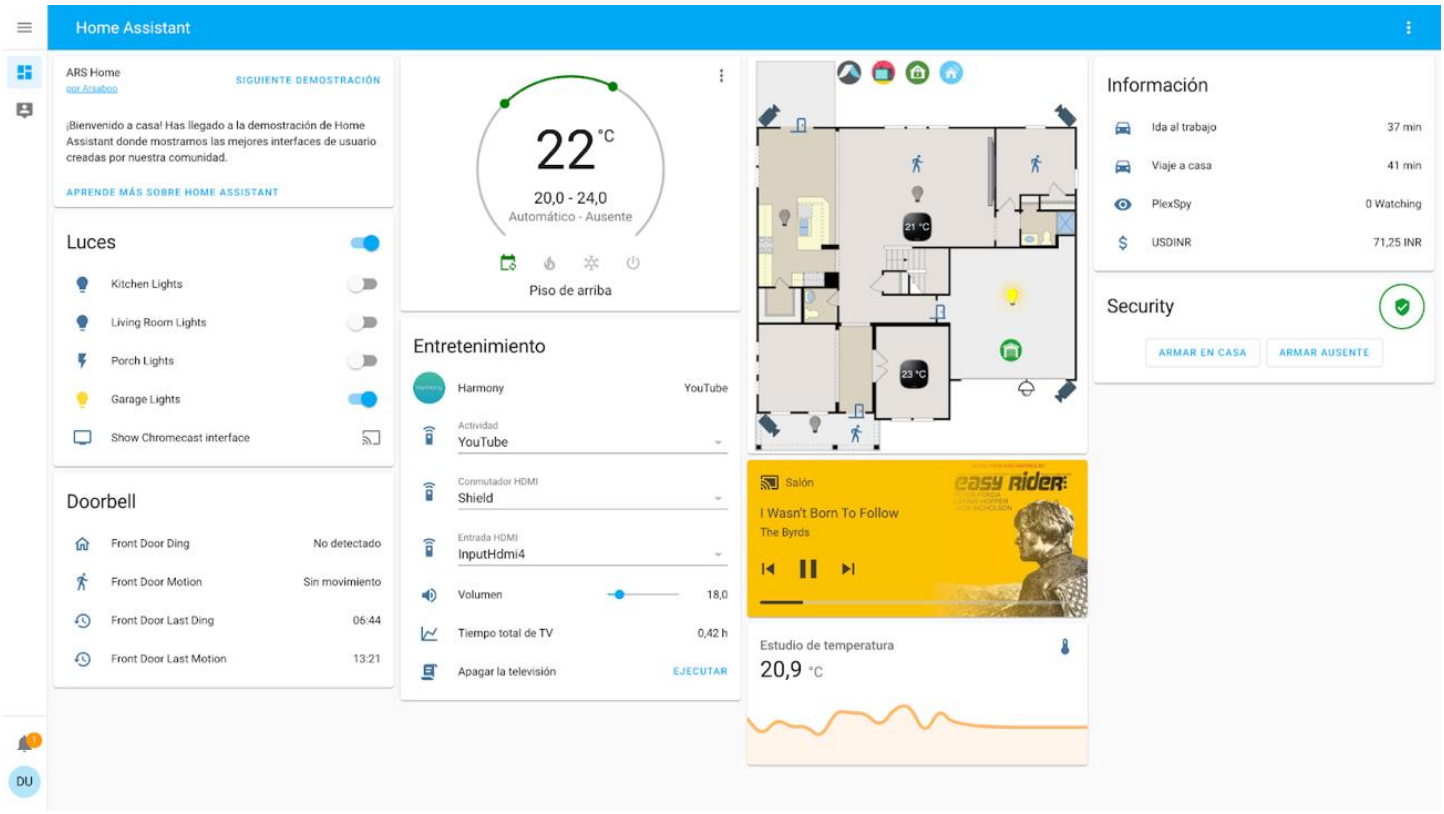

Figura 5.- Vista previa Home Assistant. Fuente: (demo.home-assistant.io, 2021).

Es por ello que este framework ha sido el escogido para realizar el trabajo ya que tiene mejor integración con el hardware, es más fácil de usar y gestionar, la extracción de los datos no resulta complicada y además trabaja con la red local sin necesidad de que los datos viajen por Internet.

2.2.2. Características principales y funciones de Home Assistant.

Además de la seguridad y la capacidad de conectar componentes de sistemas de muy diversas marcas capaces de integrar protocolos de comunicación como pueda ser MQTT o Zigbee, se encuentra la capa de personalización y la interfaz gráfica que incorpora. Esta interfaz es bastante moderna, basada en Material Design y cuenta con varias características interesantes como el soporte de WebSockets. Esto permite ver los valores de los sensores en tiempo real sin necesidad de recargar la página. Además,

también es compatible con dispositivos móviles. De esta manera, se puede controlar todo aquello que sea controlable directamente desde el teléfono móvil.

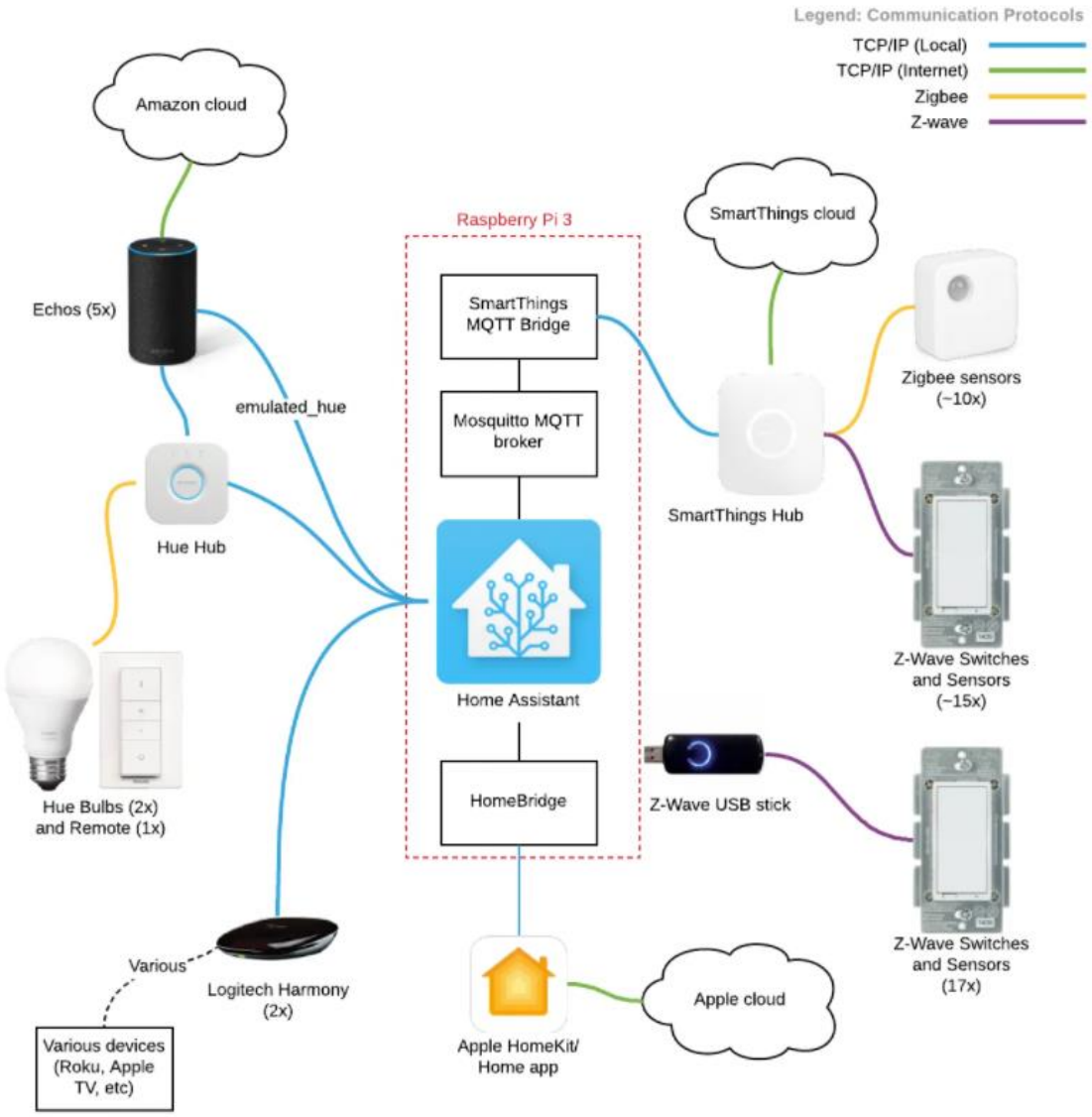

Figura 6.- Topología de una SmartHome. Fuente: (SmartHome, 2017).

Además de la capacidad de detectar dispositivos nuevos y mostrar su estado, Home Assistant permite crear escenas entre casi cualquier dispositivo. Estas escenas se basan en la ejecución automática de una serie de acciones cuando se produzca una cierta condición preestablecida. Un posible par de ejemplos serían:

- Al tener conectado un asistente de voz y decirle que reproduzca cualquier película en Netflix en la televisión del salón, Home Assistant se percataría de ello y mandaría una serie de órdenes para ambientar la habitación y ver la película correctamente. Para ello, activaría el sensor de infrarrojos que se tiene instalado en el salón y que encenderá la televisión, atenuará las bombillas inteligentes, cambiándolas de color blanco a un tono amarillo cálido para no dejar la estancia a oscuras, pero poder ver la película bien. Esta automatización se muestra gráficamente en la Figura 7.

Figura 7.- Ejemplo 1 de automatización. Fuente: Elaboración propia.

- Al tener varios perfiles configurados, el sistema sabe quién entra y sale de la casa. Por tanto, cuando se vuelva a casa se podrá configurar el sistema para que active la calefacción o el aire acondicionado a la temperatura deseada. Otra acción que se puede incorporar es activar el hilo musical mediante altavoces inteligentes para aumentar el confort y el relax tras llegar a casa una vez finalizada la jornada laboral. Además, mediante sensores

de presencia podrán ir encendiéndose las lámparas a nuestro paso si es de noche. Tal y como puede apreciarse en la Figura 8.

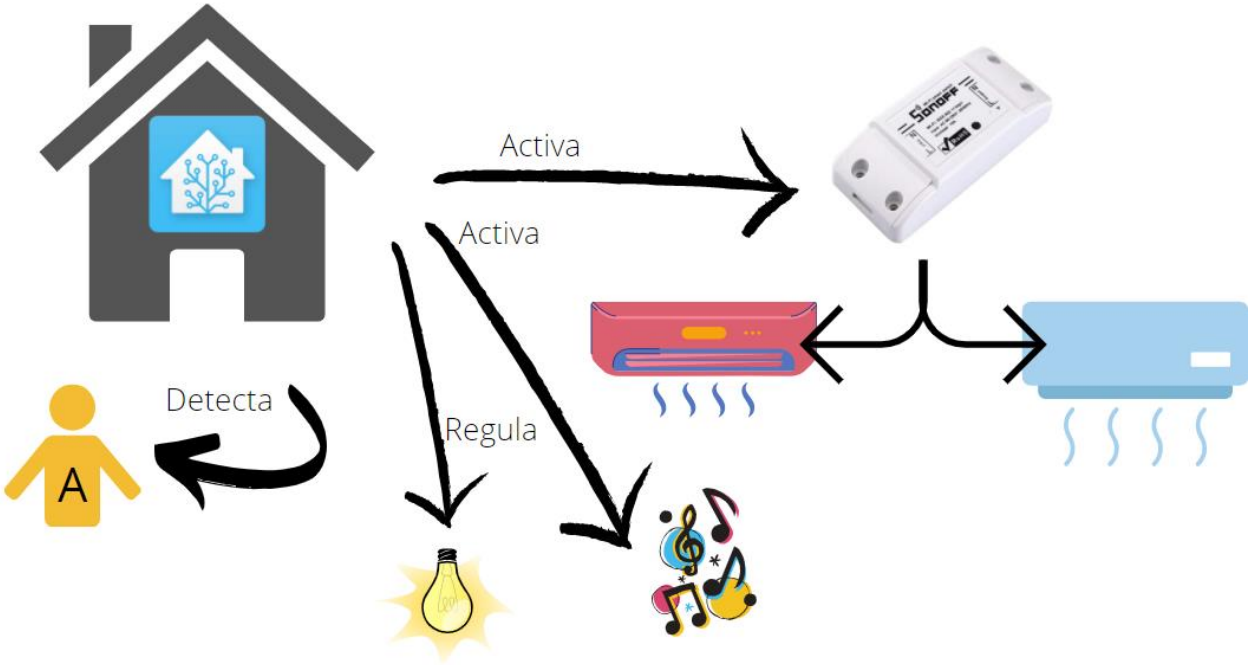

Figura 8.- Ejemplo 2 de automatización. Fuente: Elaboración propia.

Otra característica que cabe destacar es la posibilidad de comunicación directa con el usuario, de manera que si se tienen instaladas cámaras de seguridad o detectores de movimiento se podrán recibir alertas y notificaciones, por ejemplo, a la plataforma Telegram, si se detecta que hay movimiento donde no debería y se está fuera de casa. No obstante, todas estas automatizaciones las debe crear el usuario de antemano.

2.2.3. **Expansiones a la plataforma.**

Como se ha visto anteriormente, la creación de reglas y automatizaciones puede llegar a ser muy potente. No obstante, como es el usuario el encargado de definirlas, es posible que estas se queden en meras secuencias de acciones y que no se cubran todas las necesidades del mismo adecuadamente.

Es por ello que en este proyecto se propone extraer conocimiento y elaborar esas reglas expertas de actuación mediante los datos de los

sensores ambientales que se tienen dispuestos por toda la casa. De esta manera, se crearán reglas que se presentarán al usuario como sugerencia, pudiendo implementarlas si así lo cree conveniente. Esto permite enriquecer el ecosistema de Home Assistant con la posibilidad de incorporar automatizaciones que sean respuesta de un análisis más complejo del comportamiento del usuario dentro del ambiente inteligente. Esto, combinado a los actuadores que se dispongan realizará una configuración completa con lo que respecta a tener una casa inteligente y domotizada.

A continuación, se van a describir algunas herramientas para la creación de este tipo de automatizaciones.

2.2.3.1. TensorFlow.

TensorFlow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores impulsar un aprendizaje automático innovador y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de aprendizaje automático. (TensorFlow, 2021).

Esta es la definición que proporciona TensorFlow en su sitio web. Además de proporcionar un ecosistema completo para ayudar a resolver problemas complejos del mundo real, permite establecer distintos niveles de abstracción según el problema que se quiera resolver. En principio, permitiría desarrollar el modelo en cualquier lugar, ya sea en servidores o incluso en la web, sin importar el lenguaje que se utilice (TensowFlow, 2021). Para ello utiliza una estructura, generalmente, de redes neuronales, cuyo esquema principal se parece bastante a lo que se va a realizar.

Habitualmente, esta herramienta se utiliza para clasificación de imágenes o cualquier tipo de procesamiento de las mismas. Un ejemplo de cómo se puede usar TensorFlow con la plataforma Home Assistant viene de

la mano de otro proyecto donde se propone la detección de objetos para las automatizaciones ya mencionadas anteriormente (Cole, Hackster, 2018). De esta manera y a modo de resumen, con la entrada en el sistema de una imagen con sus elementos clasificados se podrían desencadenar respuestas en base a los elementos que aparezcan en dicha imagen. Para que eso funcione, es necesario tener el modelo que se va a ejecutar y que va a clasificar la imagen. No obstante, los modelos basados en redes neuronales más precisos son muy grandes, requieren de muchos recursos computacionales y no pueden ser empleados en una Raspberry Pi, cuya CPU y RAM están limitadas, por lo que hay que llegar a una solución de compromiso.

Para poder implementar este proceso de IA es necesario contar con cámaras de buena calidad y que sean compatibles con Home Assistant, algo que quizás conlleve un aumento en el costo de la implantación del sistema al ser, generalmente, más caras que otros sensores. Además, se necesitaría volver a establecer manualmente qué reglas se desea que se ejecuten en unas determinadas condiciones, no obstante, resulta un enfoque muy interesante que puede dar lugar a una cantidad mucho más extensa de reglas al poder incorporar detección de objetos a las automatizaciones.

2.2.3.2. Rhasspy Voice Assistant.

Rhasspy es un conjunto de servicios de asistentes de voz de código abierto que funciona completamente offline y que es compatible con una serie de frameworks, como pueda ser Node-Red, Jeedom o Home Assistant y Hass.io entre otros. Con él se pueden definir plantillas de acciones en formato JSON y que desencadenan acciones en la plataforma. En concreto, Home Assistant tiene soporte directo.

En la versión actual, soporta una multitud de idiomas entre los que se encuentra el español y está orientado a aquellos usuarios que quieren tener

soporte de comandos de voz sin tener que usar asistentes que estén conectados a Internet. De esta manera, se tiene un asistente que funciona totalmente desconectado de Internet y que es de código abierto, además de ser gratuito y funcionar con un software de automatización del hogar.

2.2.3.3. Home Assistant Data Science.

Recientemente, también salió un portal dedicado a la ciencia de datos a través de Home Assistant. Son los propios desarrolladores los que han desarrollado este portal y se centra en explotar los datos de los eventos que se van desencadenando y almacenando en la base de datos interna de Home Assistant.

Este estudio de los datos se realiza a través del add-on JupyterLab lite, permitiendo así el análisis local sin necesidad de exportar los datos a través de Internet con el IDE JupyterLab. Los desarrolladores han creado también una librería escrita en Python para hacer más fácil el trabajar con los datos y analizarlos. Esta librería recibió el nombre de HASS-Data-Detective. No obstante, este complemento está aún en una versión muy preliminar y es por ello que no ha sido usado en este proyecto.

2.2.3.4. Grafana y Hastic.io.

Grafana es una plataforma, que también cuenta con su propio add-on en Home Assistant, que permite realizar consultas y visualizar los datos mediante tableros. Es de código abierto y compatible con la gran mayoría de sistemas operativos, estando también la opción de usar Docker. A la hora de visualizar los datos se pueden crear mapas de calor e histogramas, entre otras opciones. También se pueden definir umbrales para los datos y si son sobrepasados que genere algún tipo de alarma.

Hastic.io es un plugin para Grafana que se puede usar para reconocimiento de patrones y detección de anomalías. Mediante esta herramienta, y gracias a los datos almacenados de medidas anteriores, se

puede establecer un patrón y una forma de cómo deberían de ser las medidas, permitiendo detectar valores inesperados que se salgan de dicho patrón (NilSk1lz, 2020).

Este sería uno de los pasos iniciales en cualquier algoritmo de IA, el detectar patrones basados en un comportamiento previo y predecir acciones que se puedan llevar a cabo. Por lo que reconocer dichos patrones es esencial, y esta herramienta lo permite.

2.2.4. **Protocolos de comunicación.**

Una vez visto qué es lo que se está haciendo actualmente y qué se puede hacer hasta la fecha con la creación de sistemas domóticos y el procesamiento de los datos que generan, solamente faltaría el medio por el cual poder comunicar y poner en contacto a las diferentes piezas del sistema.

Se encuentra de esta manera, al igual que con los diversos frameworks de domotización, múltiples propuestas. De todas ellas se destacarán dos, particularmente: Apache Kafka y MQTT, ambos utilizados en este proyecto.

- **MQTT**: Es un protocolo ligero basado en publicación/suscripción de tópicos donde se envían y desde donde se reciben los mensajes.

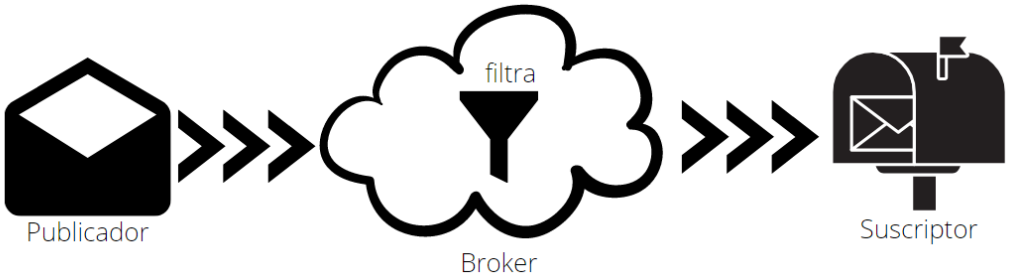

Figura 9.- Esquema de publicación/suscripción. Fuente: Elaboración propia.

Estos tópicos se organizan de manera jerárquica, con la posibilidad de suscribirse a un tópico padre y recibir todo aquello que se publique en él y sus hijos:

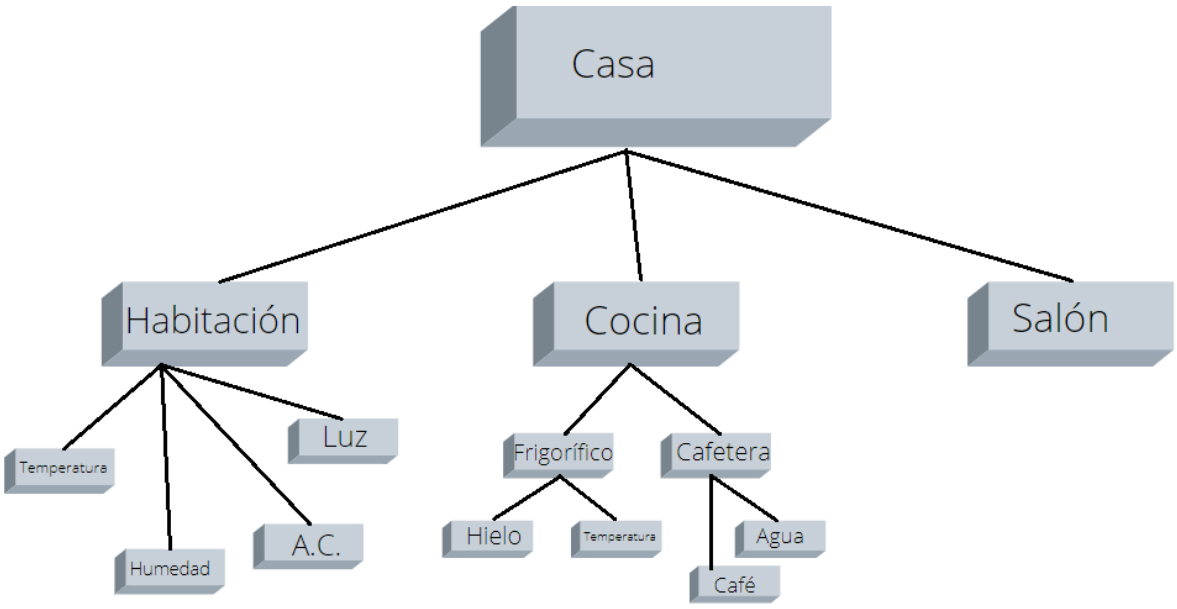

Figura 10.- Esquema de jerarquía de tópicos. Fuente: Elaboración propia.

- **Apache Kafka**: sigue un esquema parecido al de MQTT. También dispone de un bróker que organiza los distintos tópicos que se van creando, y al igual que con el otro protocolo, los clientes se suscriben y publican en ellos. La diferencia con MQTT es que esta está pensada y diseñada para manejar flujos de diversas fuentes. De esta manera, se obtiene una herramienta capaz de manejar una cantidad inmensa de información y distribuirla a múltiples entidades de manera tanto asíncrona como síncrona, según se requiera o establezca. Está escrito en Java y en Scala y sus características son (enmilocalfunciona.io, 2018):
    - **Alto rendimiento**: para conseguirlo trabaja con trozos denominados "*chunks*" de datos. No realiza acceso aleatorio a los datos y evita copiar los buffers en memoria.

Además, realiza una escritura secuencial[3] en disco obteniendo un rendimiento de O(1).

- **Distribuido**: en varios nodos que se organizan dentro de clústeres. Cada clúster es visto por un usuario como una única entidad. Facilita el escalado horizontal y la tolerancia a fallos.
- **Persistencia de datos**: los datos no se borran y estos son replicados entre los clústeres.
- **Tolerancia a fallos**: al estar los datos replicados se tiene una mayor tolerancia a costa de rendimiento.

2.2.5. **Docker.**

Docker es una tecnología de código abierto que permite virtualizar e instalar herramientas. Es muy potente y se usa ampliamente en el ámbito empresarial.

El objetivo de Docker es incorporar una tecnología que permita aislar los programas y aplicaciones instalados en un servidor de forma que sea fácil moverlos entre distintos servidores. De esta forma se evitan muchos fallos y problemas con el control de servidores.

Una solución sería el disponer de varias máquinas virtuales instaladas y ejecutándose a la vez, pero esto puede llegar a ser un proceso muy pesado y que haga que el rendimiento de la máquina anfitrión disminuya por debajo del umbral mínimo. Por lo que una solución intermedia para tener los beneficios de las máquinas virtuales sin las pérdidas de rendimiento son los contenedores.

Los contenedores comparten el núcleo del anfitrión, por lo que no hay que traducir las llamadas. El gestor del contenedor va a propagar la

[3] Hace uso de registros inmutables y escritura siempre por el final.

llamada, solamente va a controlar que se tenga permiso a aquello a lo que se va a acceder. Por ejemplo, no hay que traducir de llamadas de Windows a Linux, solamente se propaga y esto hace que sea casi instantáneo, aumentando considerablemente el rendimiento.

No obstante, y en general, si prima la seguridad y aislamiento se prefiere el uso de máquinas virtuales. Es por ello que, en el caso de trabajo que nos ocupa, al tratarse de una red privada doméstica a la que solamente deberían de poder acceder aquellos dispositivos que se tienen conectados a la red y donde no se provee ningún servicio hacia el exterior y, además, donde la potencia del servidor, que es una Raspberry Pi, es limitada. Se ha decidido por instalar la plataforma de Home Assistant en un contenedor Docker.

Para poder usar de manera correcta los contenedores y la tecnología de Docker, hay que saber cuáles son sus dos componentes principales:

1. **Imágenes**: una imagen Docker es un fichero que contiene código, librerías, herramientas y dependencias para poder hacer que una aplicación se ejecute. Son inmutables, es decir, una vez se crean no se pueden modificar, para poder modificarlas se tendría que crear una copia con los cambios que se estimen oportunos.
2. **Contenedores**: es la instancia de una imagen con una serie de configuraciones adicionales que se pueden realizar, tanto antes como después de la creación del mismo.

Docker va montando un sistema de archivos donde va introduciendo capas y lo que se quiere que se ejecute. Para evitar tener que reinstalar múltiples veces lo mismo, o si luego se quiere volver a un punto anterior, se usan dichas capas. Estas capas se comparten para ahorrar espacio.

Otro punto importante en el que hay que tener cuidado es con las imágenes que se descarguen y que no sean oficiales ya que podrían incluir algún tipo de código malicioso.

2.2.6. **Algoritmos evolutivos.**

Este tipo de algoritmos ha sufrido un aumento de interés entre la comunidad debido a los buenos resultados que suelen ofrecer consumiendo una cantidad de recursos razonables y teniendo tiempos de cómputo aceptables.

La computación evolutiva se engloba en una rama de la IA y son usados principalmente ante problemas con espacios de búsqueda muy extensos y no lineales, donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable (Wikipedia, 2021). Los algoritmos tratan de optimizar y mejorar una solución ya existente basándose en los postulados de Charles Darwin y la evolución biológica o incluso otras características o peculiaridades que se encuentran en la naturaleza, como las colonias de hormigas, entre otras.

Su esquema general de funcionamiento es el siguiente:

1. Se crea una población inicial de individuos, donde cada uno de ellos representa una solución al problema.
2. Se crea una población descendiente mediante la selección y cruce de algunos individuos. Estos, generalmente, después de cruzarse y crear un descendiente pueden o no mutar en mayor o menor medida para obtener soluciones más diferenciadas.
3. La población descendiente reemplazará de forma parcial a la población generadora mediante algún mecanismo de reemplazo. Este proceso se repetirá hasta una cierta condición de parada, como puedan ser un número determinado de generaciones o el

llegar a una solución que sea suficientemente buena para el problema.

Para llevar a cabo este proceso, es necesario definir una serie de elementos y operaciones:

- **Representación de los individuos**: debe de ser adecuada al problema y para un mismo problema puede haber varias representaciones, cada una con sus ventajas y desventajas. Es un elemento muy importante que condicionará futuros aspectos, como el operador de cruce o de mutación. Suele usarse una representación binaria, conjuntos o permutaciones de enteros, reales, etc.
- **Operador de selección**: representa el criterio por el que se elegirán a los individuos candidatos a reproducirse y generar la nueva población. El uso de uno u otro puede influir en la presión a la que se someta a la población, como, por ejemplo, escoger siempre a los mejores ejercería una fuerte presión evolutiva.
- **Operadores genéticos**: estos operadores se encargan de generar la nueva población tomando como base la población intermedia generada mediante el operador de selección. Los más destacados son los operadores de cruce y mutación.
    - **Cruce**: especifica cómo se va a generar un nuevo individuo partiendo de n padres previamente seleccionados.
    - **Mutación**: establece cómo va a variar un individuo nuevo creado para generar soluciones más dispersas.
- **Reemplazo de la población**: se tienen dos modelos, el generacional y el estacionario. Lo que se especifica aquí es cómo va a evolucionar la población durante todo el proceso evolutivo.

- **Generacional**: se cambian gran parte de los individuos que forman parte de los padres por los hijos, pero se trata siempre de mantener alguno de la población antigua.
- **Estacionario**: se cambia un número pequeño de padres por sus hijos, es decir, cambia un número pequeño de individuos de la generación de anterior con individuos de la generación actual o nueva.

2.2.7. **MapReduce.**

Se trata de uno de los paradigmas más populares y comunes en computación distribuida debido a que desplegarlo en un clúster de ordenadores es fácil, automático y transparente (Dean, 2004). Este está basado en el concepto divide y vencerás, donde se encuentran dos acciones u operaciones principales:

- **Map**: transforma una pareja de clave-valor a otra pareja con un resultado intermedio. En esta fase, cada nodo lee y transforma los datos de diferentes particiones a un modo distribuido.

$$map: \mathbb{Z} \times \mathbb{R}^n \rightarrow \mathbb{Z} \times \mathbb{R}^n$$

- **Reduce**: agrega los resultados intermedios de la fase de map con la misma clave para obtener el resultado final.

$$reduce: (\mathbb{Z} \times \mathbb{R}^n) \times (\mathbb{Z} \times \mathbb{R}^n) \rightarrow \mathbb{Z} \times \mathbb{R}^n$$

Capítulo 3.- Análisis y diseño.

El contenido del capítulo se distribuirá en dos secciones diferenciadas. En primer lugar, se presentará el análisis y la especificación de los requisitos con sus respectivos diagramas en la sección 3.1. A continuación, se realiza un completo análisis y diseño del algoritmo a utilizar para extraer conocimiento en la sección 3.2.

3.1. Análisis y especificación de requisitos.

En el siguiente apartado, se presentarán los diferentes requisitos tanto funcionales como no funcionales que requiere el proyecto. Lo primero que se muestra es el análisis textual del sistema mediante la Tabla 3.

3.1.1. Análisis textual.

	Actor	Caso de Uso	Valor de Negocio
ID	Como...	, quiero	para
1	Usuario	ver las reglas de las que dispongo	activar o desactivarlas.
2	Usuario	modificar las reglas de las que dispongo	añadir o borrar reglas.
3	Usuario	modificar la interfaz de Home Assistant	ponerla mi gusto.
4	Sistema	recoger los datos de los sensores	analizar y crear reglas.
5	Sistema	Crear reglas expertas	presentárselas al usuario.
6	Sistema	presentar las reglas al usuario	que las acepte o las rechace.

Tabla 3.- Análisis textual.

En la Tabla 3 se pueden ver los diferentes casos de uso que se encuentran en el sistema y que se detallarán más adelante mediante los correspondientes diagramas.

3.1.2. **Casos de uso.**

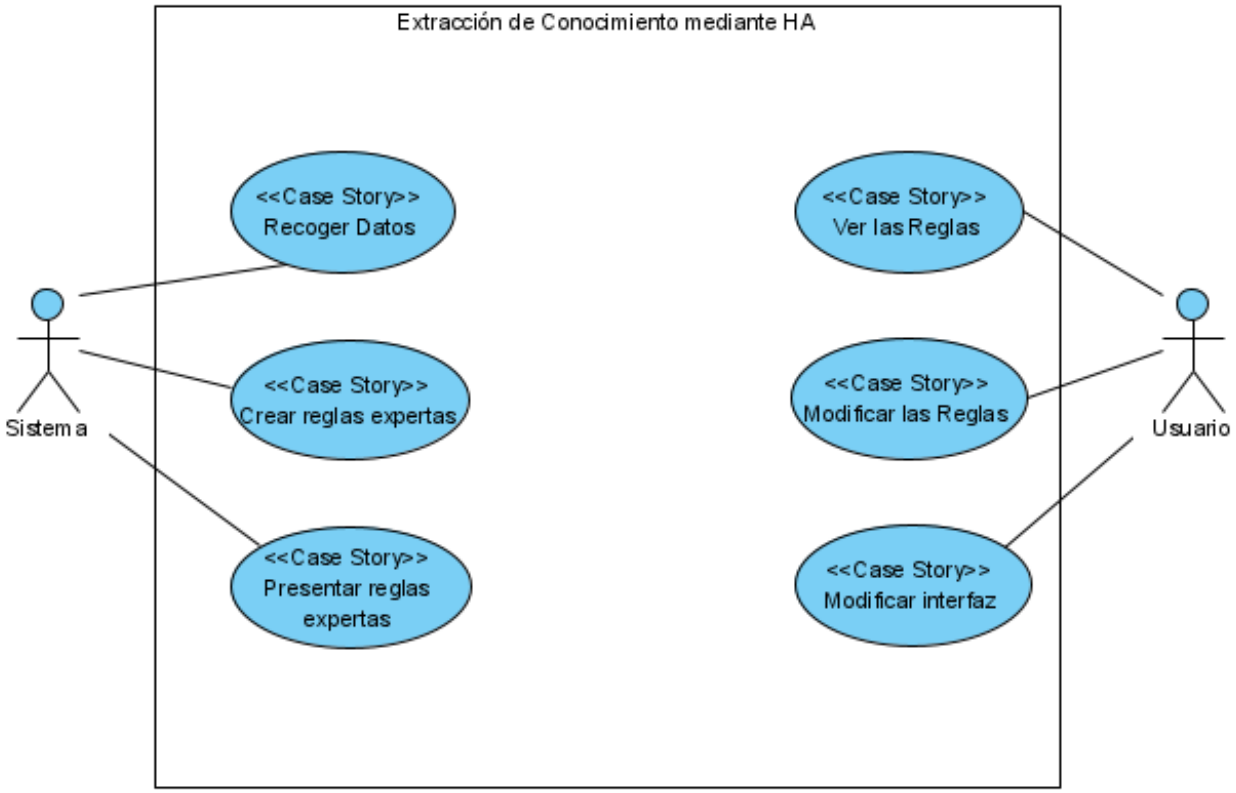

Figura 11.- Casos de uso. Fuente: Elaboración propia.

3.1.2.1. Ver las reglas.

Caso de uso	Ver las reglas.
Actor Primario	Usuario.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Usuario y sistema.
Nivel	Objetivo de usuario.
Precondiciones	-
Flujo Básico	
1	Abrir supervisor.
2	Seleccionar apartado de reglas.
3	Ver listado disponible.
4	Seleccionar regla.
5	Activar o desactivar regla.
6	Terminar.
Flujo Alternativo	
4.A	Si no hay reglas se salta a 6.
4.B	Si no se selecciona ninguna se salta a 6.

Tabla 4.- Análisis textual. Ver las reglas.

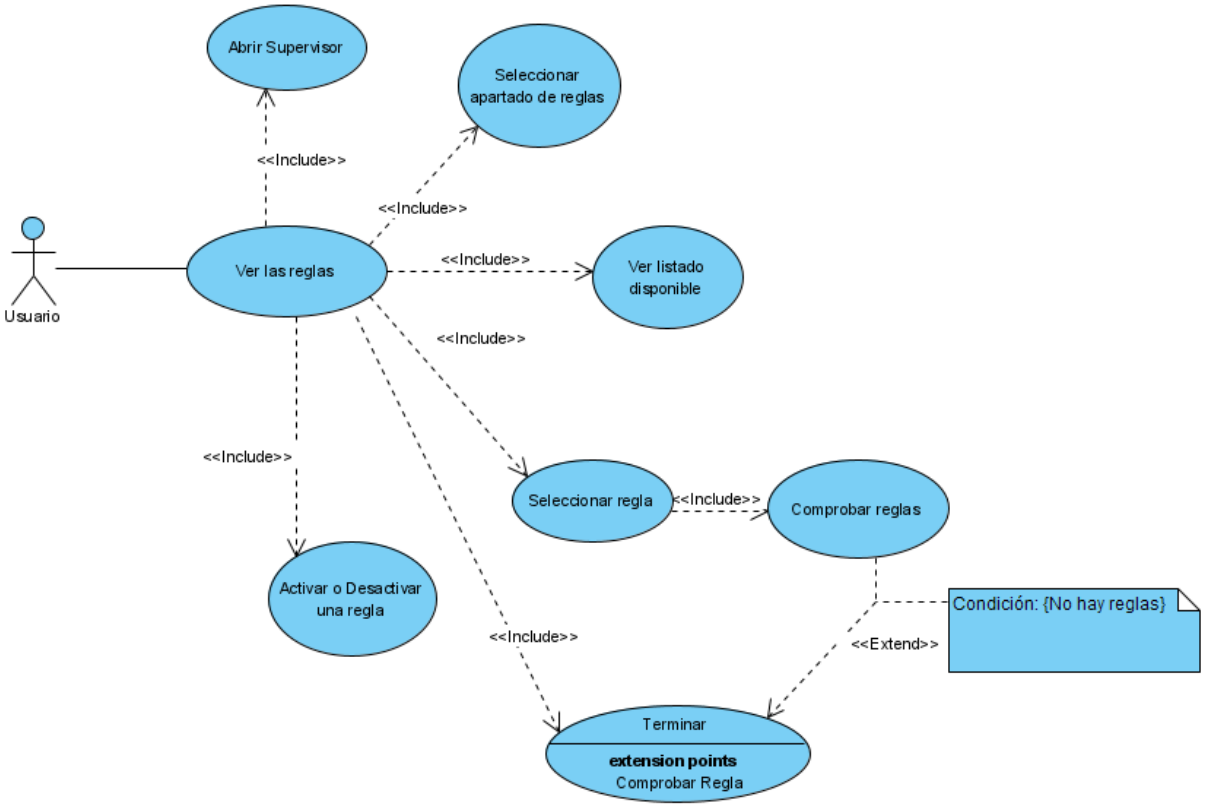

Figura 12.- Diagrama de caso de uso. Ver las reglas. Fuente: Elaboración propia.

3.1.2.2. Modificar las reglas.

Caso de uso	Modificar las reglas.
Actor Primario	Usuario.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Usuario y sistema.
Nivel	Objetivo de usuario.
Precondiciones	Que haya reglas.
Flujo Básico	
1	Abrir supervisor.
2	Seleccionar apartado de reglas.
3	Ver listado disponible.
4	Seleccionar regla.
5	Modificar la regla.
6	Terminar.
Flujo Alternativo	
4.A	Si no se selecciona ninguna se salta a 6.

Tabla 5.- Análisis textual. Modificar las reglas.

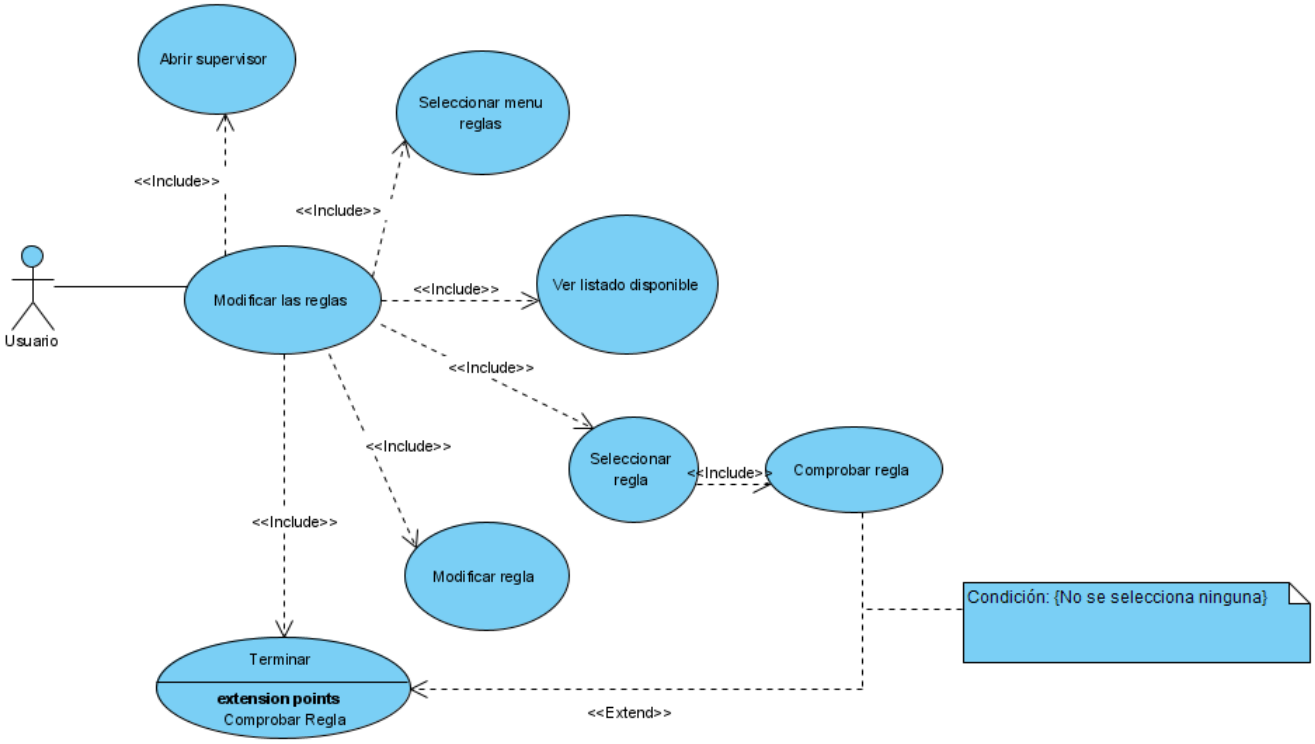

Figura 13.- Diagrama de caso uso. Modificar las reglas. Fuente: Elaboración propia.

3.1.2.3. Modificar interfaz.

Caso de uso	Modificar interfaz.
Actor Primario	Usuario.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Usuario y sistema.
Nivel	Objetivo de usuario.
Precondiciones	-
Flujo Básico	
1	Abrir supervisor.
2	Seleccionar apartado de resumen.
3	Seleccionar modo edición.
4	Modificar tarjetas.
5	Guardar.
6	Terminar.
Flujo Alternativo	
5.A	Si no se quiere guardar se salta 6.

Tabla 6.- Análisis textual. Modificar interfaz.

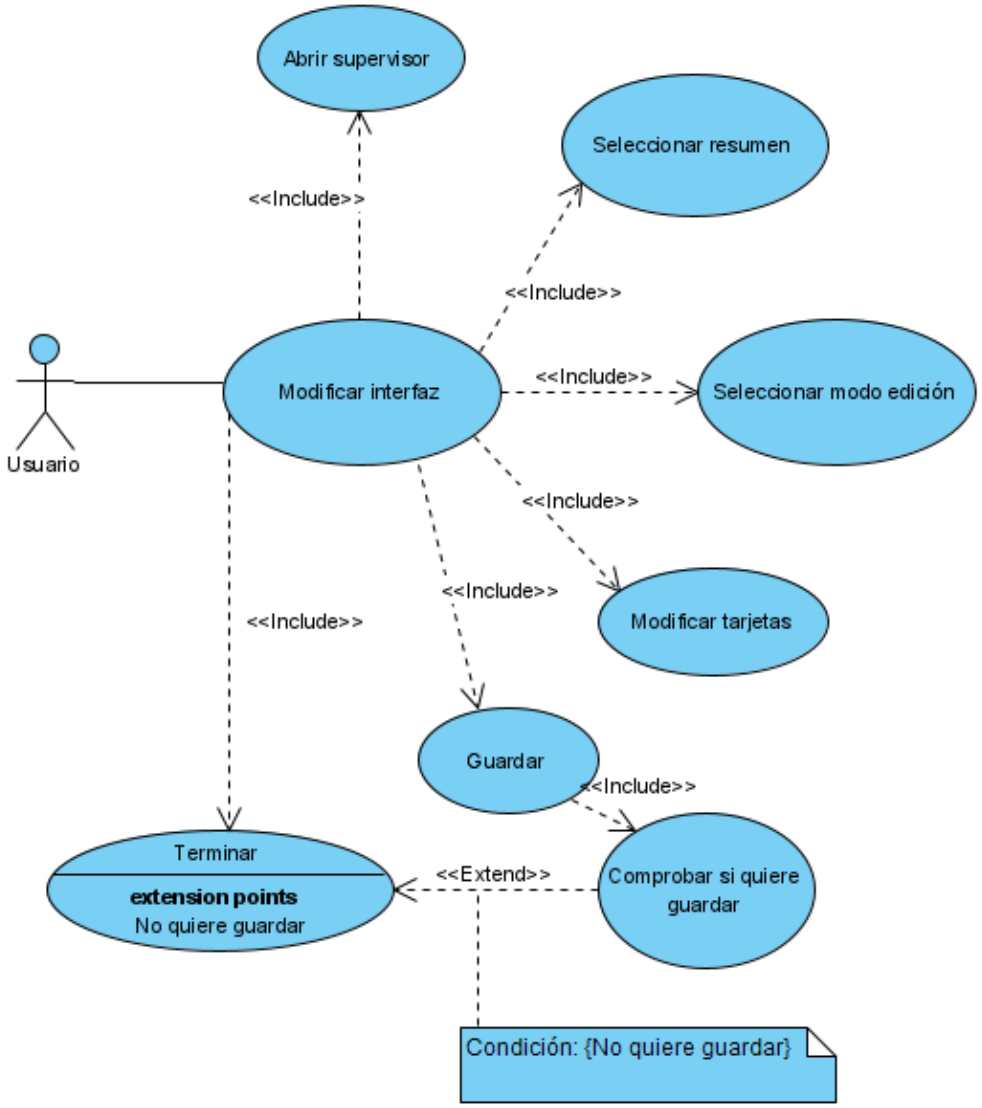

Figura 14.- Diagrama de caso de uso. Modificar Interfaz. Fuente: Elaboración propia.

3.1.2.4. Recoger los datos de los sensores.

Caso de uso	Recoger los datos de los sensores.
Actor Primario	Sistema.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Sistema.
Nivel	Objetivo del sistema.
Precondiciones	Que haya sensores recogiendo datos.
Flujo Básico	
1	Establecer comunicación con los sensores.
2	Establecer comunicación con el bróker.
3	Recoger datos del sensor.
4	Publicar datos en el tópico.
5	Terminar.
Flujo Alternativo	
1.A	Si falla, se salta a 5.
2.A	Si falla, se salta a 5.

Tabla 7.- Análisis textual. Recoger los datos de los sensores.

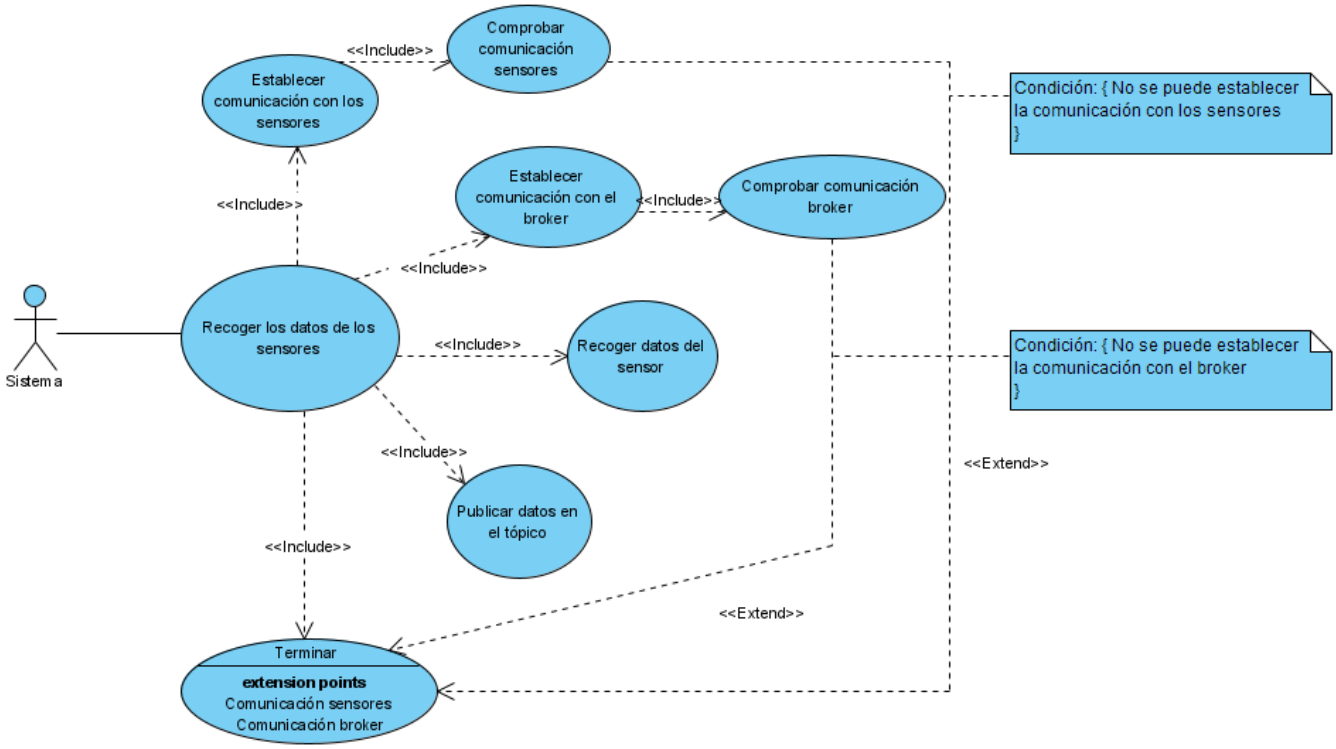

Figura 15.- Diagrama de caso de uso. Recoger los datos de los sensores. Fuente: Elaboración propia.

3.1.2.5. Crear reglas expertas.

Caso de uso	Crear reglas expertas.
Actor Primario	Sistema.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Sistema.
Nivel	Objetivo del sistema.
Precondiciones	-
Flujo Básico	
1	Establecer comunicación con el bróker.
2	Recoger los datos del tópico.
3	Formatear y adecuar los datos.
4	Analizar y crear reglas.
5	Terminar.
Flujo Alternativo	
1.A	Si falla se salta a 5.
3.A	Si no hay ninguno se salta a 5.

Tabla 8.- Análisis textual. Crear reglas expertas.

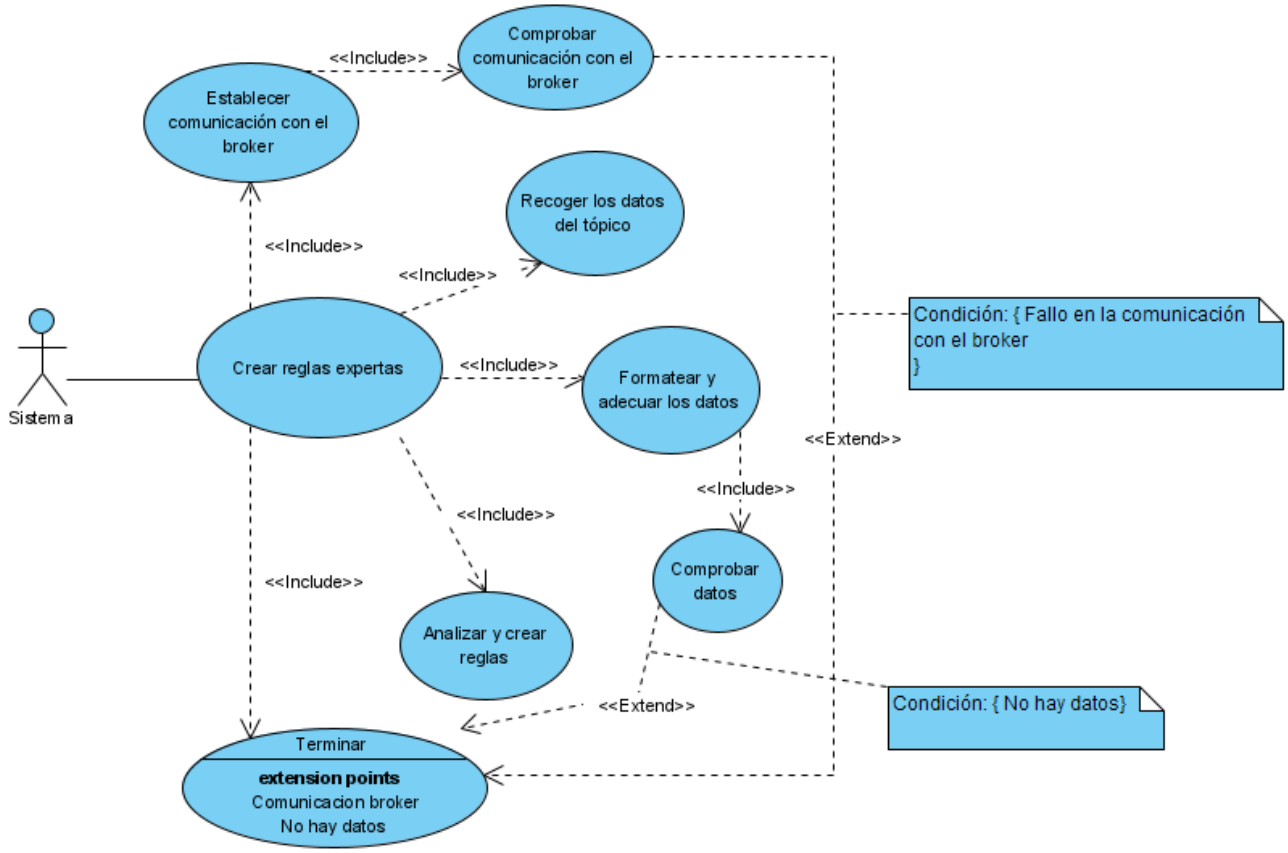

Figura 16.- Diagrama de caso de uso. Crear reglas expertas. Fuente: Elaboración propia.

3.1.2.6. Presentar reglas expertas.

Caso de uso	Presentar reglas expertas.
Actor Primario	Sistema.
Sistema	Extracción de conocimiento mediante Home Assistant.
Participantes	Sistema y usuario.
Nivel	Objetivo del sistema.
Precondiciones	Que haya reglas que presentar.
Flujo Básico	
1	Recopilar reglas creadas.
2	Presentar reglas al usuario.
3	Esperar confirmación por parte del usuario.
4	Actuar en consecuencia de las respuestas del usuario.
5	Terminar.
Flujo Alternativo	
3.A	Usuario acepta y quiere la regla.
3.B	Usuario rechaza y no quiere la regla.

Tabla 9.- Análisis textual. Presentar reglas expertas.

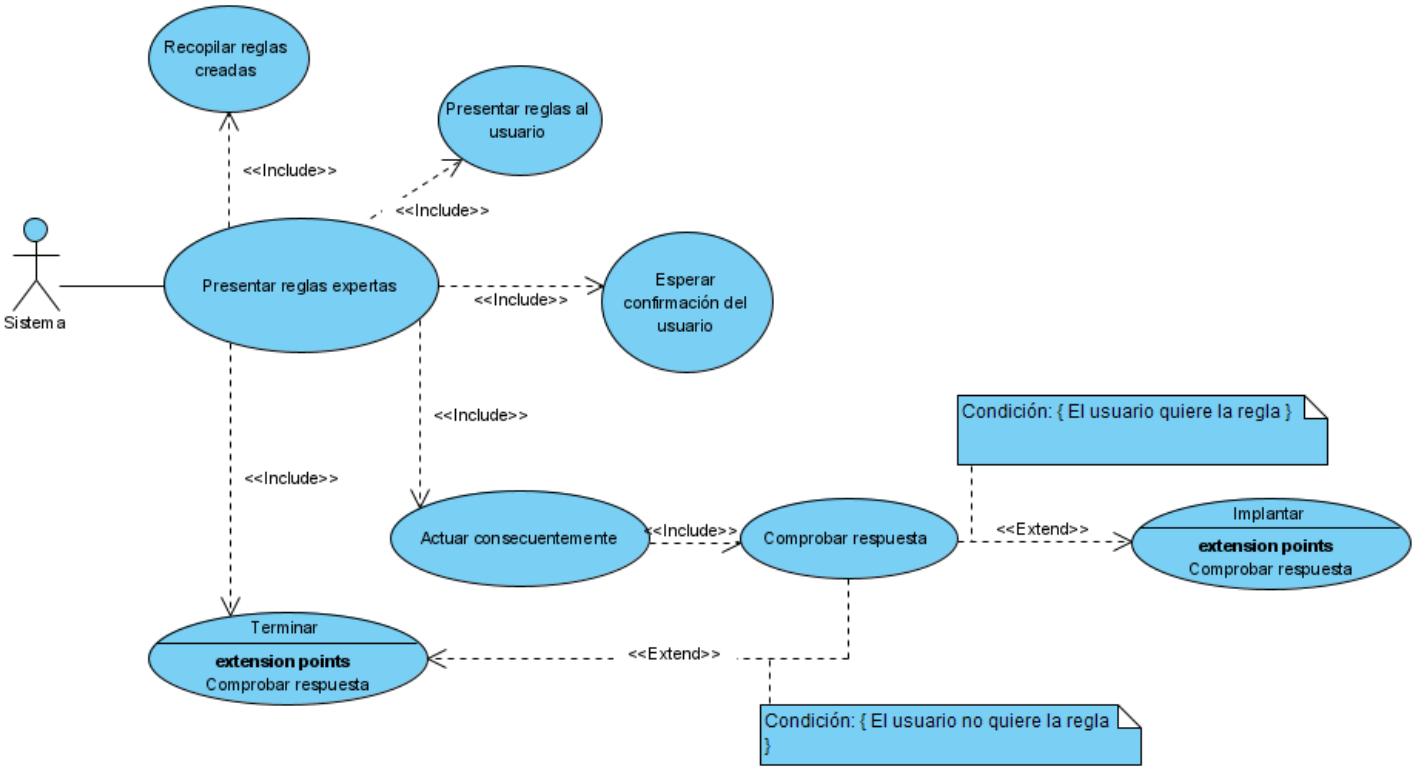

Figura 17.- Diagrama de caso de uso. Presentar reglas expertas. Fuente: Elaboración propia.

3.1.3. Requisitos funcionales y no funcionales.

Con respecto a los requisitos funcionales, ya se ha mostrado una descripción de los mismos ya que dichos requisitos se traducen en los casos de uso anteriores.

De esta manera, el sistema debe de poder hacer lo presentado en el análisis textual de la Tabla 3, y resumido en la siguiente tabla:

Requisito 1: Ver las reglas instaladas.
Requisito 2: Modificar las reglas.
Requisito 3: Modificar la interfaz de Home Assistant.
Requisito 4: Recoger los datos de los sensores.
Requisito 5: Crear reglas expertas.
Requisito 6: Presentar las reglas al usuario.

Tabla 10.- Requisitos funcionales.

Con respecto a los requisitos no funcionales se pueden extraer los siguientes requisitos:

Requisito 1: Escalable
Requisito 2: Rendimiento aceptable
Requisito 3: Usable
Requisito 4: Tolerante a fallos

Tabla 11.- Requisitos no funcionales.

1. **Escalabilidad**: se busca que el sistema sea escalable tanto vertical como horizontalmente.
    a. **Verticalmente**: la escalabilidad vertical consiste en poder mejorar el hardware con un esfuerzo mínimo. En este caso, se podría incluso migrar por completo el hardware requiriendo de poco esfuerzo, ya que las plataformas usan Docker y el algoritmo de extracción de conocimiento está escrito en Scala ejecutándose por debajo en Java. Por lo que la compatibilidad de estas herramientas es casi completa.
    b. **Horizontalmente**: la escalabilidad horizontal consiste en mejorar el modelo añadiendo más nodos con la finalidad de repartir el trabajo necesario. En este caso, el empleo de Apache Kafka y Apache Spark permiten incorporar fácilmente nuevos nodos de cómputo al sistema.
2. **Rendimiento**: se busca también que el sistema tenga un rendimiento lo suficientemente bueno. Esto quiere decir que tenga una eficiencia sobre la memoria que se requiere y el tiempo que se demora en realizar sus tareas. En el contexto en el que se trabaja, el tiempo necesario es vital ya que se engloba en el minado de flujo de datos continuos.
3. **Usabilidad**: también es deseable que el sistema no sea difícil de usar y que presente una curva de aprendizaje mínima. El usuario

que vaya a usar el sistema final no tiene por qué tener conocimientos previos de informática y el enfrentamiento con el sistema debe de ser intuitivo y fácil de aprender.

4. **Tolerancia a fallos**: por último, se busca que el sistema sea capaz de recuperarse ante problemas que ocurran. Como puedan ser: sensores que quedan desconectados temporalmente o cortes de suministro eléctrico.

3.1.4. Arquitectura global del sistema.

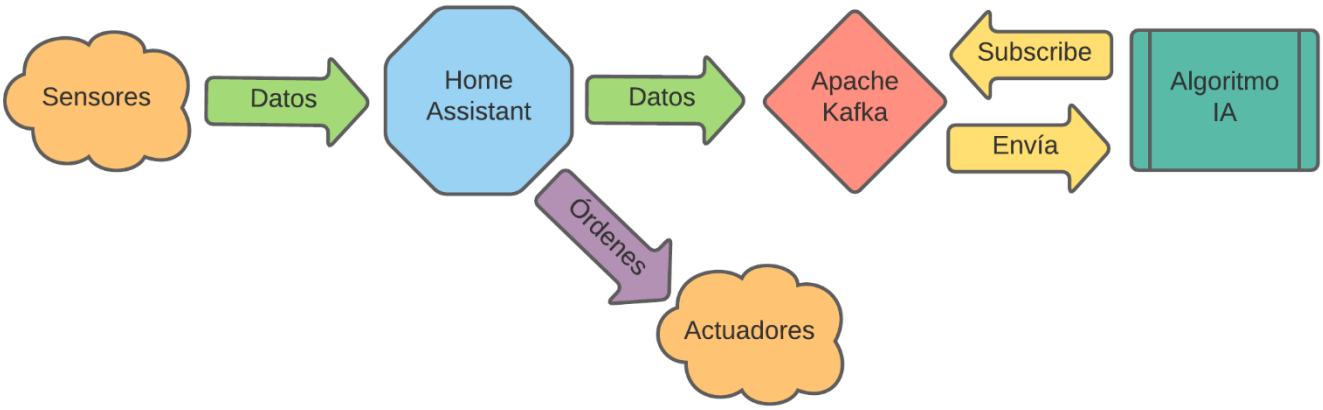

Figura 18.- Arquitectura global del sistema. Fuente: Elaboración propia.

Para comenzar, se ha decidido elaborar este pequeño diagrama donde se puede apreciar de manera sencilla el funcionamiento global del sistema completo. Este consta de varias partes:

- **Sensores**: conjunto de Wemos D1 Mini flasheados con sus respectivos sensores conectados y transmitiendo datos por la red privada de casa.
- **Home Assistant**: se encargará de recoger los datos de los sensores, publicarlos en el bróker de Apache Kafka y además mandar órdenes a los actuadores que correspondan. También será el encargado de mostrar la interfaz amigable al usuario.
- **Actuadores**: conjunto de dispositivos que se encargarán de realizar las órdenes y acciones pertinentes según una serie de condiciones previas.

- **Apache Kafka**: servidor que actuará de bróker, recogiendo y publicando los datos de los sensores a aquellos dispositivos que se subscriban al tópico que gestiona.
- **Algoritmo IA**: será el encargado de extraer conocimiento y reglas expertas a través de los datos recogidos de los sensores por medio de la subscripción al tópico del bróker.

Los distintos componentes del servidor se encuentran distribuidos según se puede observar en el diagrama de la Figura 19.

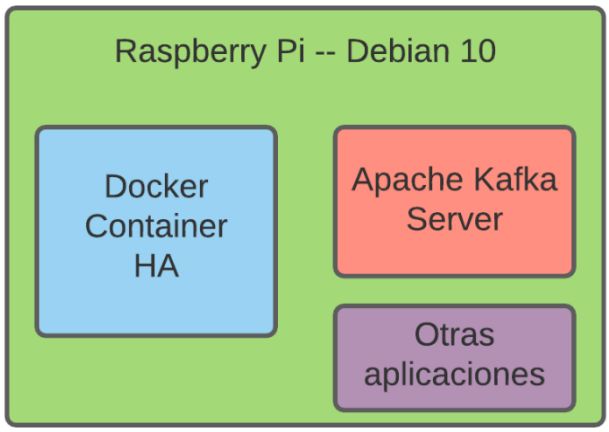

Figura 19.- Diagrama de componentes internos Raspberry Pi.
Fuente: Elaboración propia.

De esta manera, el sistema operativo (Debian 10) está instalado en la Raspberry Pi y en ella se ejecuta un contenedor de Docker con la imagen de Home Assistant.

A su vez, y al ser un SO normal y no dedicado exclusivamente a Home Assistant, se puede ejecutar también todo lo que la capacidad de cómputo del miniordenador permita, como pueda ser el bróker y otras aplicaciones que se quiera (como los contenedores de Heimdall y Portainer, los cuales se explicarán más adelante en la sección 8.1.2 del primer anexo).

3.1.5. Diagramas de secuencia.

3.1.5.1. Diagrama de secuencia del lado del cliente.

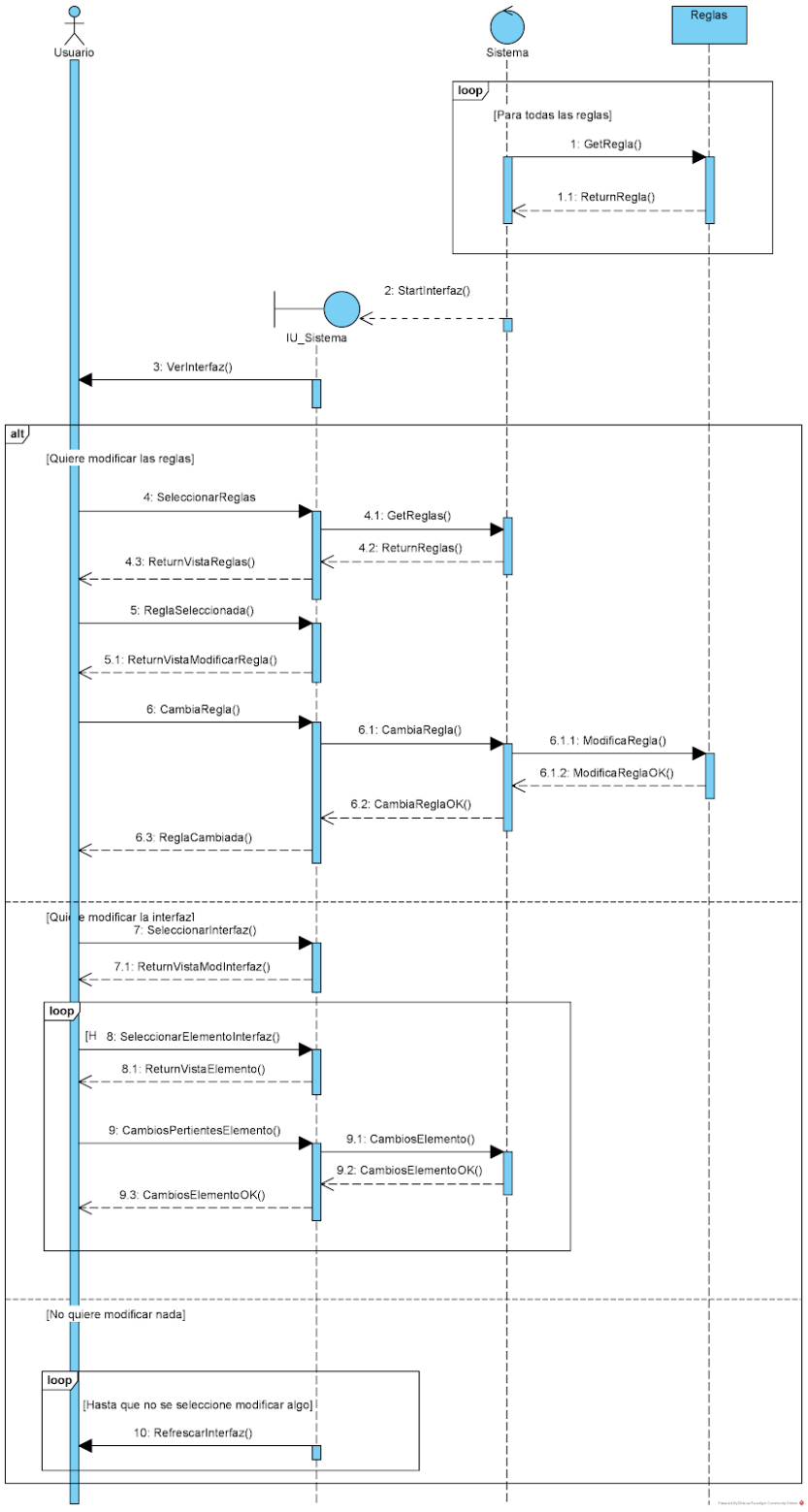

Figura 20.- Diagrama de secuencia del lado del cliente. Fuente: Elaboración propia.

3.1.5.2. Diagrama de comunicación del lado del cliente.

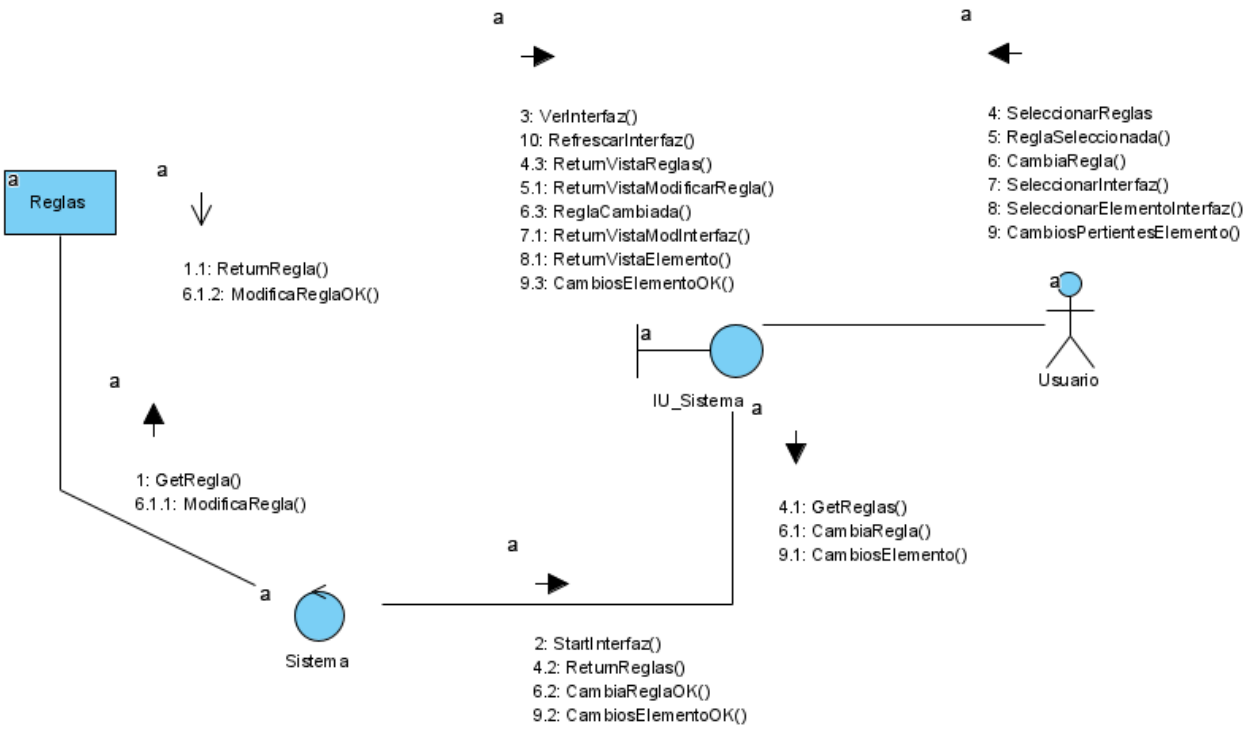

Figura 21.- Diagrama de comunicación del lado del cliente. Fuente: Elaboración propia.

3.1.5.3. Diagrama de secuencia del lado de los sensores.

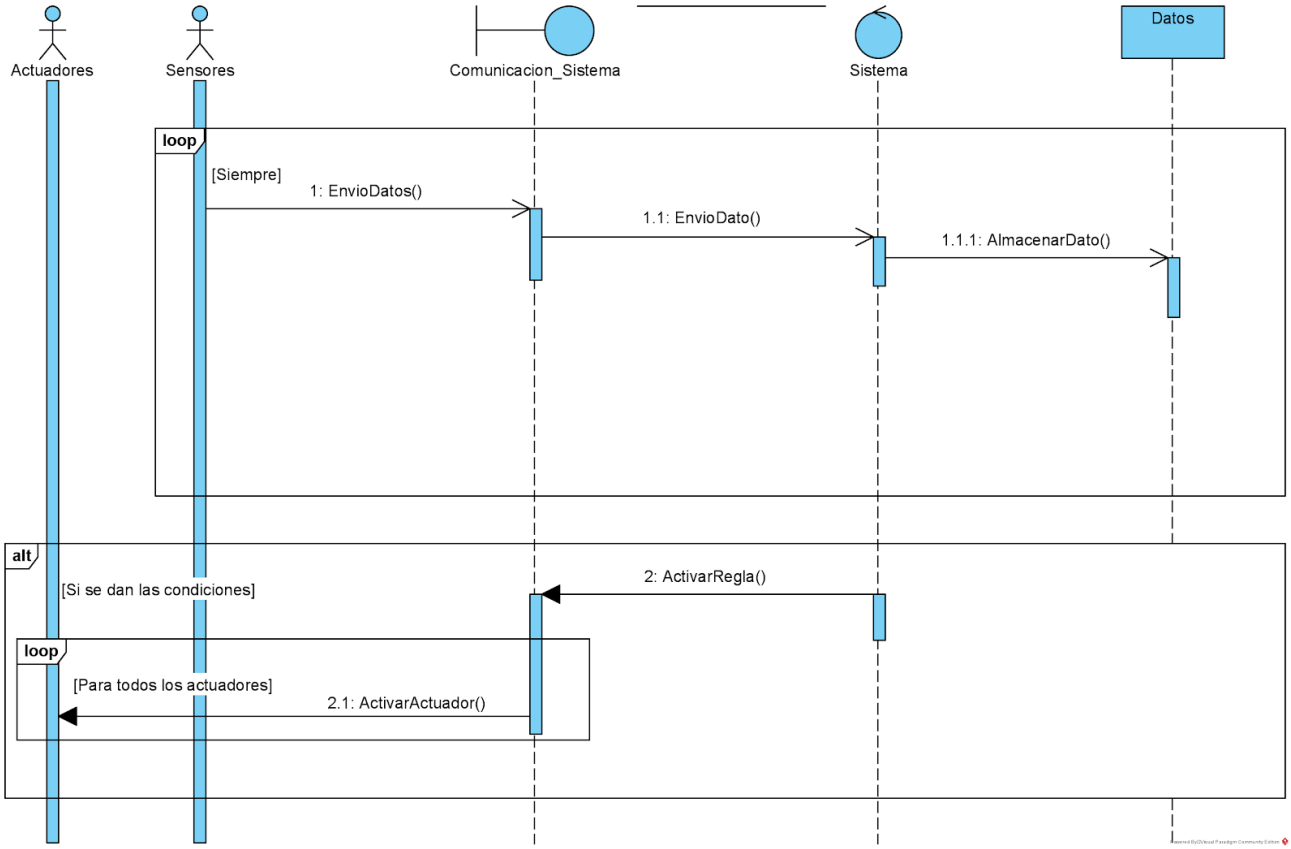

Figura 22.- Diagrama de secuencia del lado de los sensores. Fuente: Elaboración propia.

3.1.5.4. Diagrama de comunicación del lado de los sensores.

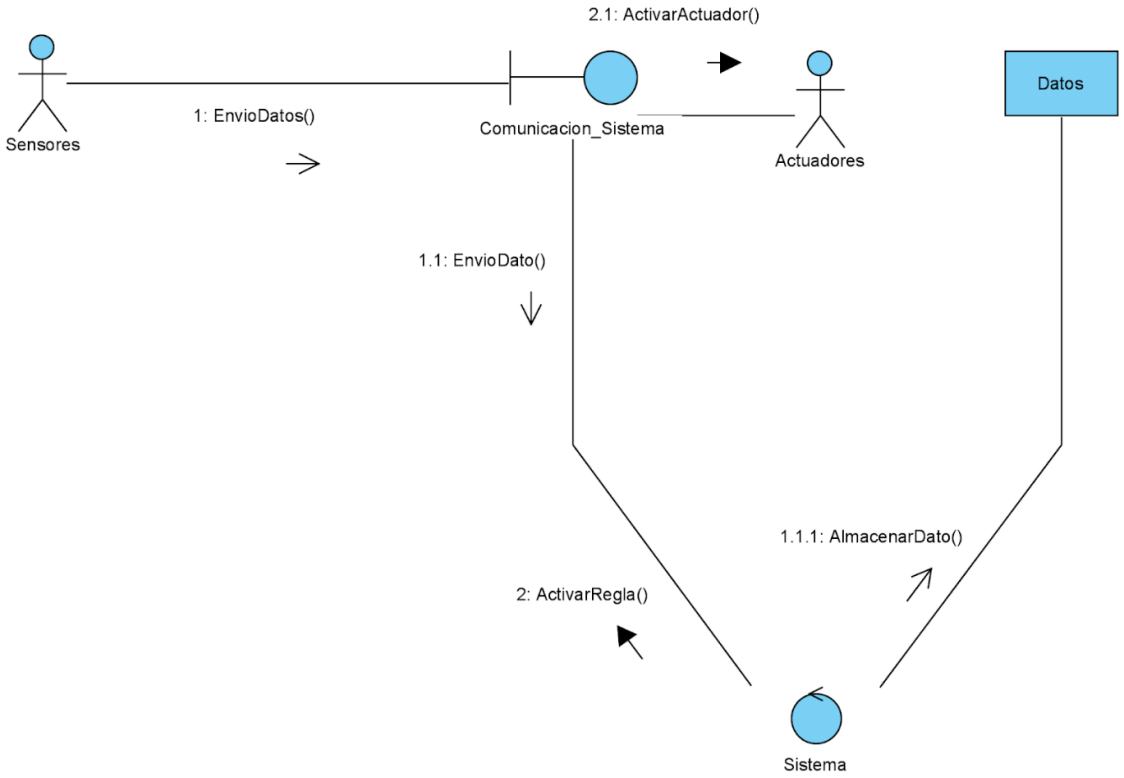

Figura 23.- Diagrama de comunicación del lado de los sensores. Fuente: Elaboración propia.

3.1.5.5. Diagrama de secuencia del lado del algoritmo.

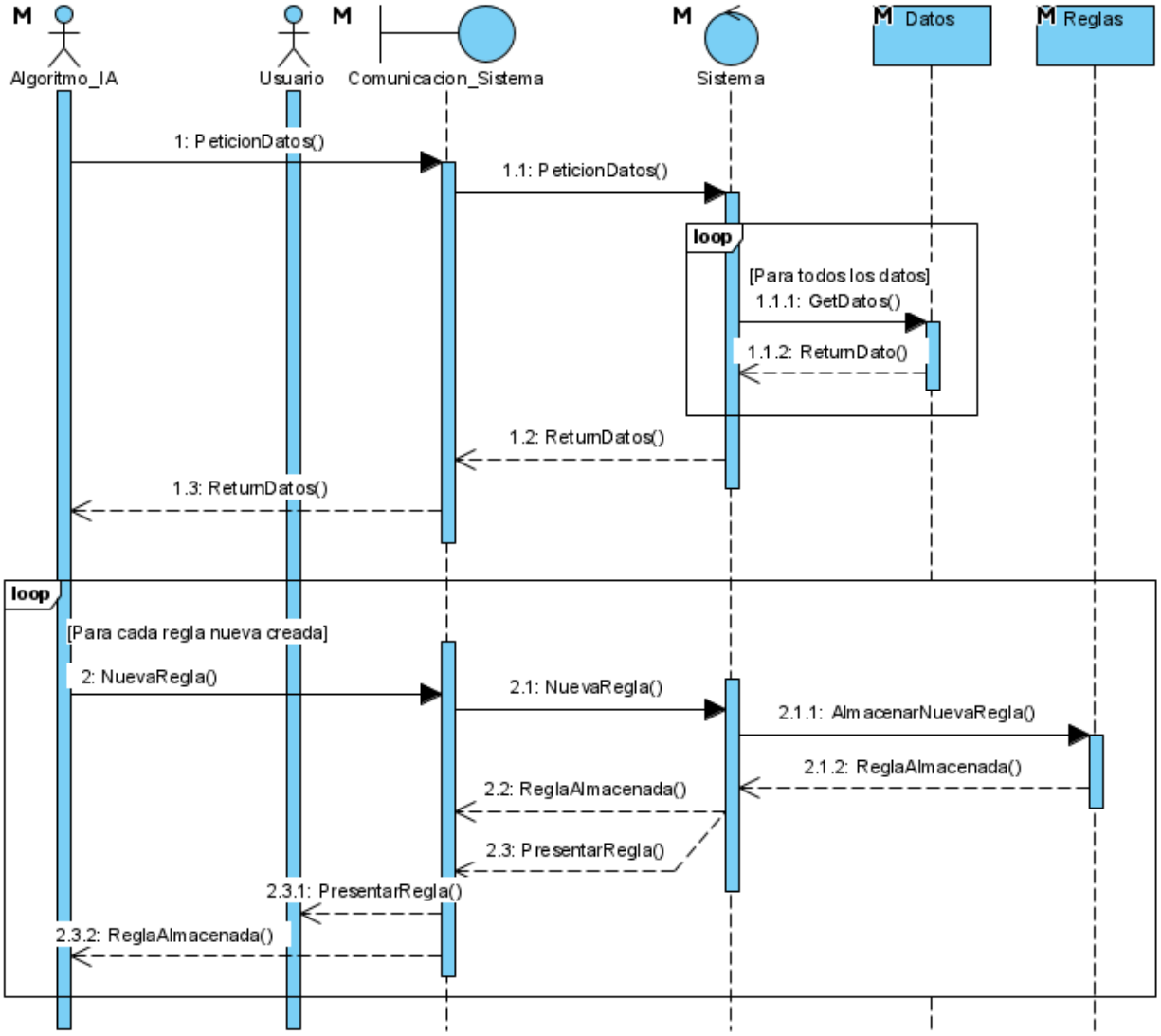

Figura 24.- Diagrama de secuencia del lado del algoritmo. Fuente: Elaboración propia.

3.1.5.6. Diagrama de comunicación del lado del algoritmo.

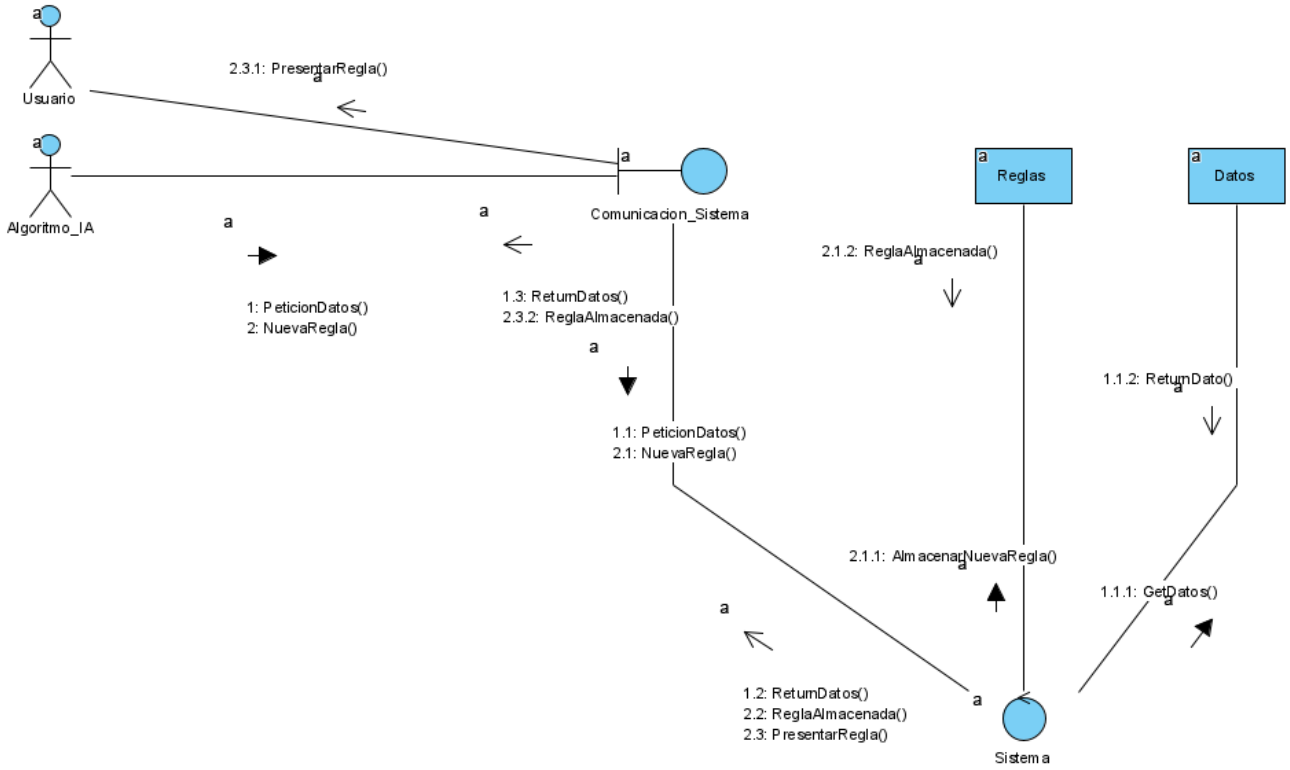

Figura 25.- Diagrama de comunicación del lado del algoritmo. Fuente: Elaboración propia.

3.1.6. Diagramas de conexión los sensores.

Para los diagramas de conexión de los distintos sensores de 4 de las 5 placas que se disponen, se mostrará una única Figura 26 debido a que el proceso de montaje, el cual se detalla en el anexo de la sección 8.2, ha sido el mismo para todos ellos.

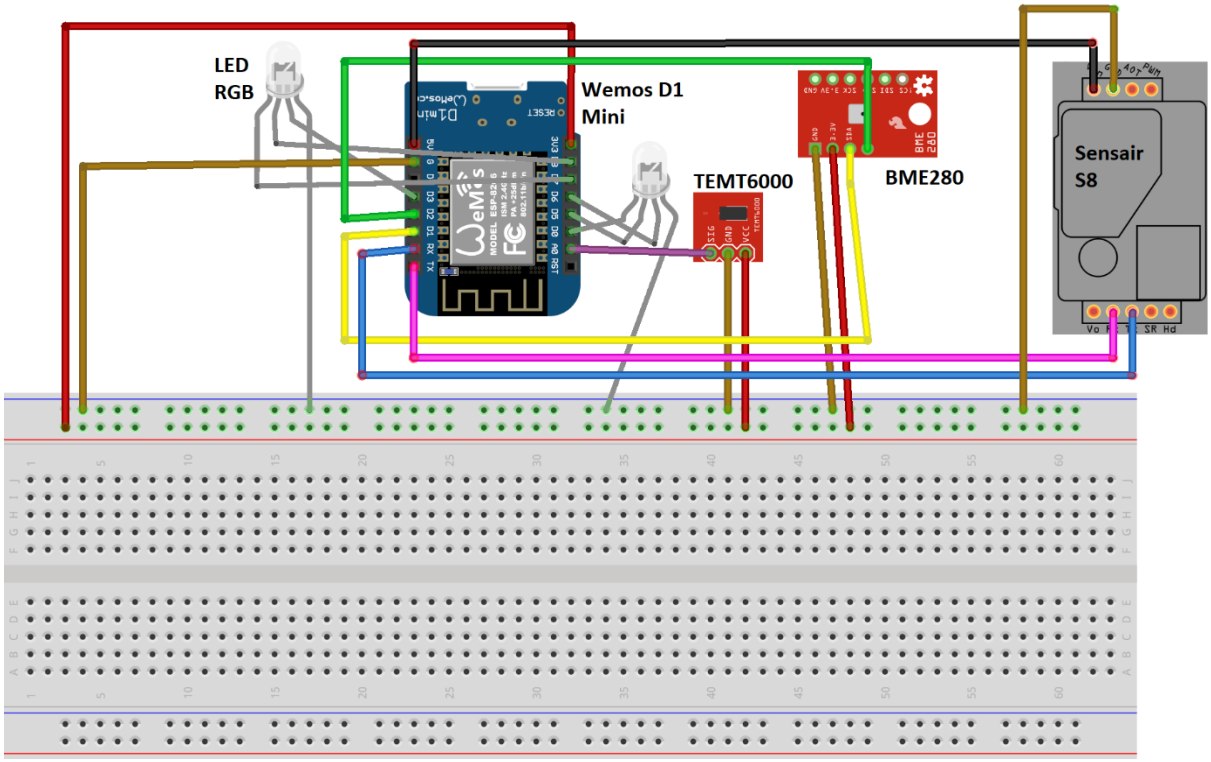

Figura 26.- Diagrama de conexión de los sensores de las placas principales. Fuente: Elaboración propia.

De esta manera, se puede apreciar cómo se dispone de una serie de sensores ya presentados en la Tabla 2 y resumidos en la Tabla 12. La conexión de los mismos se describe a continuación:

Sensor	Medición
BME280	Sensor de temperatura, humedad y presión.
BH1750	Sensor de luz más potente. Para el sol.
FR-04	Sensor de lluvia.
TEMT6000	Sensor de luminosidad interior.
Senseair S8	Sensor de CO_2
Sonoff Basic R2	Placa 1 relé wifi.
Emisor de infrarrojos	Emisor de frecuencia infrarroja para el control de dispositivos.
Wemos D1 mini	Módulo ESP-8266 mini.
Led KY-016	Dispositivo led RGB.

Tabla 12.- Resumen sensores del sistema.

- 3.3V: todos los sensores y luces funcionan a 3.3 voltios, salvo el sensor Senseair S8 (CO_2) que funciona a 5 voltios.
- BME280:
  - SDA → D1
  - SCL → D2
- TEMT6000:
  - SIG → A0
- LED KY-016:
  - Pin Rojo → D0 y D7
  - Pin Verde → D5 y D8
  - Pin Azul → D6 y D3
- Senseair S8:
  - RX y TX de manera cruzada → TX y RX

Para la última placa, se muestra el siguiente diagrama de conexión:

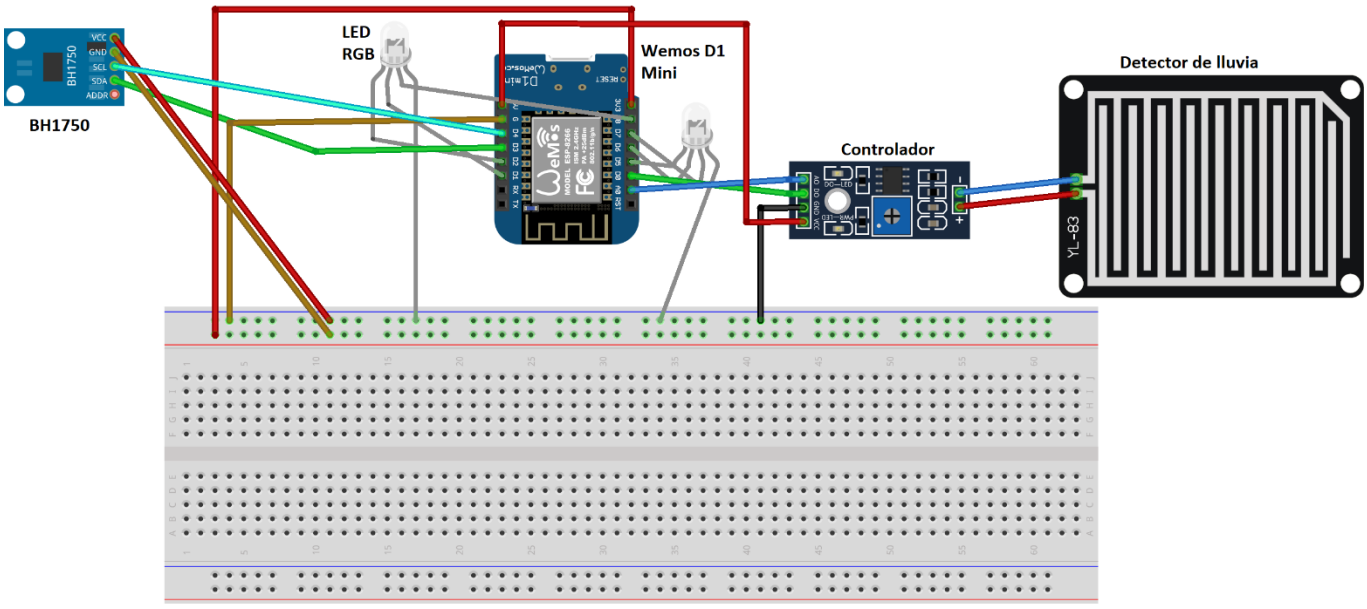

Figura 27.- Diagrama de conexión de los sensores en la placa con el sensor de lluvia. Fuente: Elaboración propia.

- 3.3V: tanto las luces como el sensor de lluvia y el sensor de luz funcionan a 3.3 voltios
- BH1750:

    - SDA → D1
    - SCL → D2
- Sensor Lluvia:
    - A0 → A0
    - D0 → D0
- LED KY-016:
    - Pin Rojo → D4 y D5
    - Pin Verde → D3 y D6
    - Pin Azul → D8 y D7

Para terminar esta sección, se presentará el diagrama de conexión del Sonoff Basic R2 con el respectivo transmisor infrarrojos (IR):

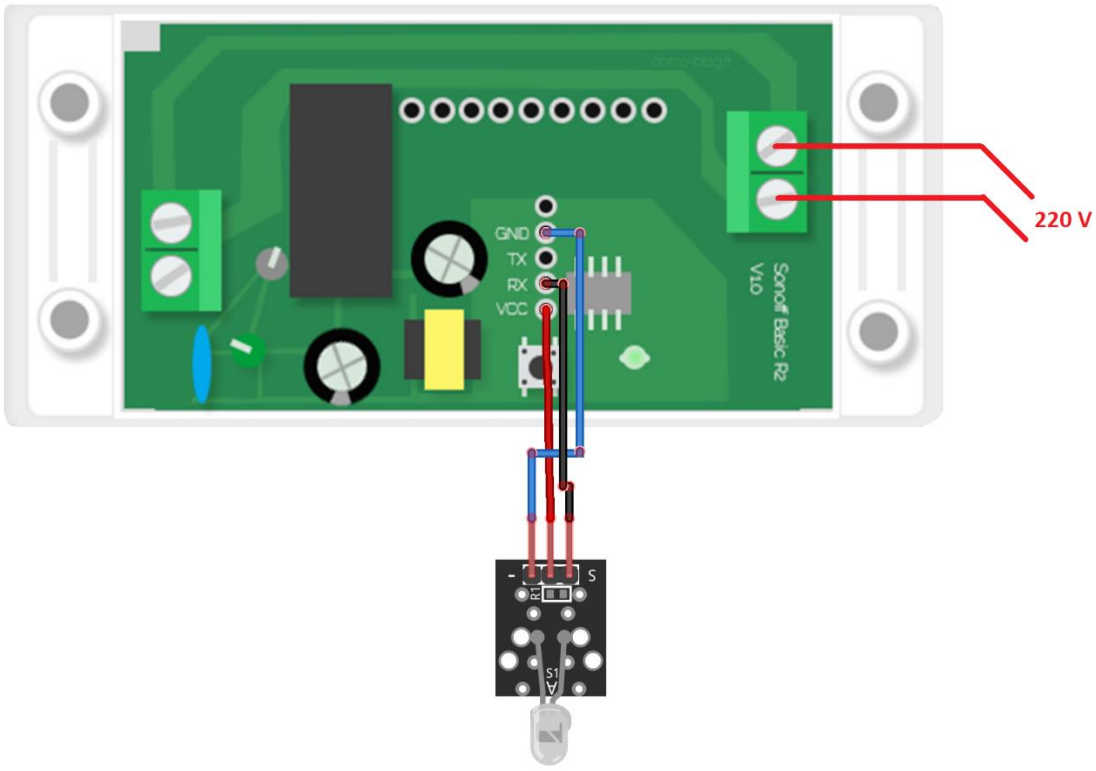

Figura 28.- Diagrama de conexión del transmisor IR con Sonoff Basic R2. Fuente: Elaboración propia.

3.2. Estudio del problema.

Para comenzar el análisis del algoritmo se definirán los tipos de datos abstractos o TDA que se necesitan, así como la forma en la que se obtienen los datos de la publicación/suscripción mediante Apache Kafka:

3.2.1. Tipos de datos.

- TDA **Medida**: los datos de los sensores que se recogen de la plataforma Home Assistant siguen un formato homogéneo y estructurado, que es aquel que será publicado en el bróker de Apache Kafka. El formato sigue la siguiente estructura:

```
{
      "entity_id": "sensor.co2_habitacion",
      "state": "786",
      "attributes": {
            "unit_of_measurement": "ppm",
            "friendly_name": "CO2 Habitacion",
            "icon": "mdi:molecule-co2"
      },
      "last_changed": "2021-05-01T16:55:16.034571+00:00",
      "last_updated": "2021-05-01T16:55:16.034571+00:00",
      "context": {"
            id": "5de307f30a2a139fa0a928cc177338cb",
            "parent_id": null,
            "user_id": null
      }
}
```

Código 1.- JSON de una medida de sensor cualquiera.

De las cuales se usarán:

- **`Entity ID (String)`**: será el identificador unívoco que se establezca a la hora de crear/registrar un dispositivo. Es la manera de identificar los diferentes sensores.
- **`State (Flotante)`**: es el valor numérico de cualquiera de los sensores que se establezcan y uno de los parámetros clave.
- **`Last Updated (Fecha)`**: indica la fecha de la última actualización que tuvo el sensor, y, por lo tanto, la fecha de la medida que se está procesando.

A estos, le se añadirá además:

- **`Fecha (String)`**: será el valor de clase y tendrá los valores "actual" o "anterior".

- TDA **RDD[String]**: representará una colección de medidas en formato CSV. Se podrá implementar como una cola o un vector. En este caso, tendrá 24 elementos[4] y en cada una de esas posiciones albergará el Resilient Distributed Dataset (RDD) con las medidas de los sensores. El RDD es la estructura de datos para almacenar y procesar de manera distribuida datos en Apache Spark.
- TDA **Vector<T>**: colección de elementos de tipo T.
- TDA **Cola<T>**: colección ordenada mediante FIFO de elementos de tipo T.

3.2.2. Atributos necesarios.

Durante la ejecución del algoritmo se necesitará almacenar diversos valores en distintas estructuras de datos. Estas son:

- **`ArrayBuffer<RDD[String]>` arrayOfRDD**: será el uso del segundo TDA definido anteriormente. Dispondrá de 24 posiciones

[4] Una por cada hora del día, desde las 00 hasta las 23 horas.

con un RDD en cada una de ellas con las medidas del histórico o base de datos.

- **`ArrayBuffer<RDD[String]>` instancesByHourActual**: es parecido al anterior pero solamente tendrá las medidas que sean posteriores al número de días siguientes al día establecido en los parámetros.[5]

- **`Queue<RDD[String]>` queueRDD**: resultará necesario en el contexto de Spark y albergará la unión de cada uno de los RDD de los vectores anteriores por sus respectivas posiciones[6]. Permitirá realizar la simulación de publicación/suscripción de Apache Kafka.

3.2.3. Procedimientos de apoyo.

A continuación, se presentan los procedimientos de apoyo más destacados:

3.2.3.1. Para las medidas.

- **`toWeka(clase, sensores)`**: devuelve la cadena de caracteres de una medida clasificada como "actual" o "anterior", dependiendo del parámetro de entrada clase, en el formato arff que usará Spark.

- **`esSensor(json)`**: indica si una medida en formato JSON es de un sensor o de otro elemento.

- **`parseJsonToWeka(json)`**: convertirá la medida en formato JSON a una cadena de caracteres que siga el formato arff.

3.2.3.2. Para el algoritmo.

- **`prechargeRDD(SparkContext)`**: precarga la base de datos con los valores anteriores a un día especificado como parámetro clasificados como "anterior" en un vector de 24 posiciones, una por hora del día. Además, genera otro vector con las medidas de ese día establecido como parámetro. Para dicho día especificado,

[5] La intersección de ambos vectores será el conjunto vacío. Es decir, el primer vector se rellena hasta el día y hora establecido en los parámetros y el segundo desde dicho día y hora.

[6] queueRDD[i] = arrayOfRDD[i] + instancesByHourActual[i]

habrá que indicar un día, mes, año y una hora. Se contempla un día como las 24 horas siguientes a dicho momento.

- **`loadConfig()`**: carga los parámetros de configuración especificados en el fichero `params.txt`.
- **`loadSensors()`**: carga los sensores que haya especificados en el fichero `sensors.txt`.
- **`addRDD(rdd, hora)`**: une al RDD que vaya acumulando los datos de una cierta hora al RDD que se le pase como parámetro de entrada.
- **`sameDayOfWeek(día)`**: indica si el día que se pasa como parámetro de entrada es el mismo día de la semana[7] que el establecido en los parámetros anteriores.
- **`readDatasetFromRDD_String(rdd, header)`**: devuelve una cadena de caracteres con todas las medidas que haya en el RDD formateadas en arff separadas por saltos de línea y con la cabecera acoplada antes de ellas.

3.2.4. Parámetros de ejecución.

A la hora de ejecutar el algoritmo, se quiere poder modificar su comportamiento en mayor o menor medida. Para ello, será necesario establecer una serie de parámetros de control, los cuales son:

- **`kafka (boolean)`**[8]: establece si se usará la base de datos incorporada simulando el funcionamiento de Apache Kafka o se usarán los datos que se vaya recibiendo por Apache Kafka para las instancias clasificadas como actuales.
- **`completeHistory (boolean)`**: indica si se quiere buscar en toda la historia completa o en una cantidad de días anteriores únicamente.

[7] Lunes, martes, miércoles, jueves, viernes, sábado o domingo.
[8] Si el parámetro kafka está establecido a verdadero, los siguientes parámetros solamente especificarán hasta donde se quiere precargar la base de datos con instancias clasificadas como anteriores.

- **`sameDay (boolean)`**: al igual que los anteriores, configura si se quiere limitar la búsqueda a únicamente los mismos días de la semana (ya sea en toda la historia o los "*backwardDays*" anteriores).
- **`startYear (entero)`**: indica el año del día que se establece como límite o día actual.
- **`startMonth (entero)`**: IDEM al anterior, pero con el mes.
- **`startDay (entero)`**: IDEM al anterior, pero con el día.
- **`startHour (entero)`**: instaura a partir de qué hora del día concreto señalado por los 3 anteriores parámetros se va a considerar medidas actuales y anteriores[9].
- **`forwardDays (entero)`**: configura el número de días que se buscará hacia delante y serán consideradas medidas actuales si kafka=false.
- **`backwardDays (entero)`**: establece el número de días que se buscará hacia atrás si kafka=false y serán consideradas medidas anteriores.
- **`timeWindow (entero)`**: en segundos, indica el tamaño de la ventana deslizante.
- **`sensorRefreshTime (entero)`**: en segundos, fija el tiempo de refresco que tienen los sensores, es decir, cada cuanto toman medidas.
- **`ficheroBBDD (string)`**: será el nombre del fichero donde se tienen los datos de las medidas guardados en formato CSV.

[9] Con estos 4 parámetros, se obtiene un día concreto YYYY-MM-DD HH:00. Hasta dicho día y hora será considerado hacia atrás una cantidad de días (backwardDays) o en toda la historia de la base de datos. Por lo que para dicho día y a esa hora debe tener al menos un valor la base de datos.

3.2.5. **Descripción del algoritmo.**

En la minería de datos hay dos corrientes claramente diferenciadas, el aprendizaje supervisado y el no supervisado. No obstante, hay técnicas como el descubrimiento de reglas descriptivas basadas en el aprendizaje supervisado que tiene como objetivo describir conocimiento de datos etiquetados que se encuentra a medio camino de ambos grupos. Entre las técnicas existentes se encuentra la minería de cambio que es una rama dentro de la minería de patrones emergentes cuyo objetivo es la obtención de reglas o patrones que sean frecuentes en un valor de la clase objetivo y poco frecuentes en el resto (Dong G. & Li, 1999) (García-Vico et al., 2018).

El principal objetivo de la minería de cambio (Wai-Ho Au, 2005) es la detección de tendencias emergentes en conjuntos de datos marcados temporalmente. Este tipo de aprendizaje es adecuado para este tipo de proyecto, ya que se trabaja con medidas de sensores claramente diferenciadas en el tiempo y múltiples variables como son esas medidas y sensores.

El algoritmo de aprendizaje que se usará en este proyecto está basado en un sistema difuso evolutivo multiobjetivo que provee de un buen equilibrio entre la exploración y la explotación en el proceso de búsqueda. Tiene un mecanismo de reinicio y además un filtrado para eliminar patrones redundantes con baja fiabilidad para mejorar la interpretabilidad y que originalmente estaba pensado para minería de patrones emergentes (Á. M. García-Vico et al., 2020).

3.2.5.1. Sistema de reglas difusas.

Estos sistemas de conocimiento están compuestos por un conjunto de reglas del tipo SI-ENTONCES donde el antecedente y el consecuente son ambos conjuntos difusos.

Hay dos componentes en un FRBS(Fuzzy Rule-Based System):

1. La base de conocimiento (KB), que contiene reglas difusas.
2. Las definiciones de los conjuntos difusos en la base de datos (DB).

La construcción de una base de datos es un proceso complejo ya que muchos factores deben ser optimizados para cada variable en un espacio de búsqueda muy grande y continuo. Además, debe de darse una KB ya provista y esto hace que sea poco flexible, sobre todo en contextos de flujos continuos de datos. Por otra parte, se podría aprender la KB, algo interesante y que aporta flexibilidad para monitorizar el estado del flujo junto con un espacio de búsqueda discreto al difuminar los valores (aunque una DB debe de ser provista por el experto).

Con esto anterior, el algoritmo es un Sistema Difuso Evolutivo (EFS) que aprende/actualiza su KB cuando sea necesario por medio de la extracción de patrones emergentes difusos usando los datos que llegan de distintas fuentes. Lo hace gracias a:

1. Un procedimiento para lanzar de manera inteligente el método de aprendizaje para ahorrar recursos.
2. El método de aprendizaje basado en un algoritmo evolutivo celular multi-objetivo que contiene operadores específicos para extraer patrones emergentes descriptivos.
3. Un enfoque basado en Apache Spark y Apache Kafka que provee una entrada de datos de distintas fuentes eficiente, escalable y tolerante a fallos.

Las etiquetas asociadas a una variable son inicializadas por medio de una partición uniforme difusa a lo largo de todo el dominio actual y con el mismo número de etiquetas para todas las variables. En el método son

definidas por medio de una función de pertenencia triangular ya que su definición y cómputo son rápidos, algo que es clave en este contexto.

3.2.5.2. Adaptación al cambio.

La adaptación del modelo al estado actual del flujo es llevada a cabo usando un sistema de monitorización de cambios, es decir, usando una estrategia informada. Este componente monitorizará ambos drifts o cambios de concepto, el real y el virtual. De esta manera, sólo se actualiza cuando hay un descenso significativo en el rendimiento.

El sistema de monitorización de cambios está basado en la idea de que un patrón emergente (EP) es una descripción que resume el estado actual del flujo. Por lo que es un objeto representativo de los datos, y si estos cambian, el objeto también y, por consiguiente, la calidad de la descripción hecha por el EP también cambia. Por lo tanto, los indicadores supervisados pueden ser empleados para detectar drifts.

Una vez el nuevo bloque de datos llega, el actual modelo de patrones es evaluado usando esos datos para la determinación de su calidad. Luego se monitoriza si la confianza o la memoria de los EP anteriormente extraídos baja de un cierto umbral. Como se usa el nuevo bloque de datos, si alguna de las condiciones se cumple se considera un cambio de concepto (real o virtual). Y entonces se lanza el proceso de aprendizaje sobre esos datos actuales.

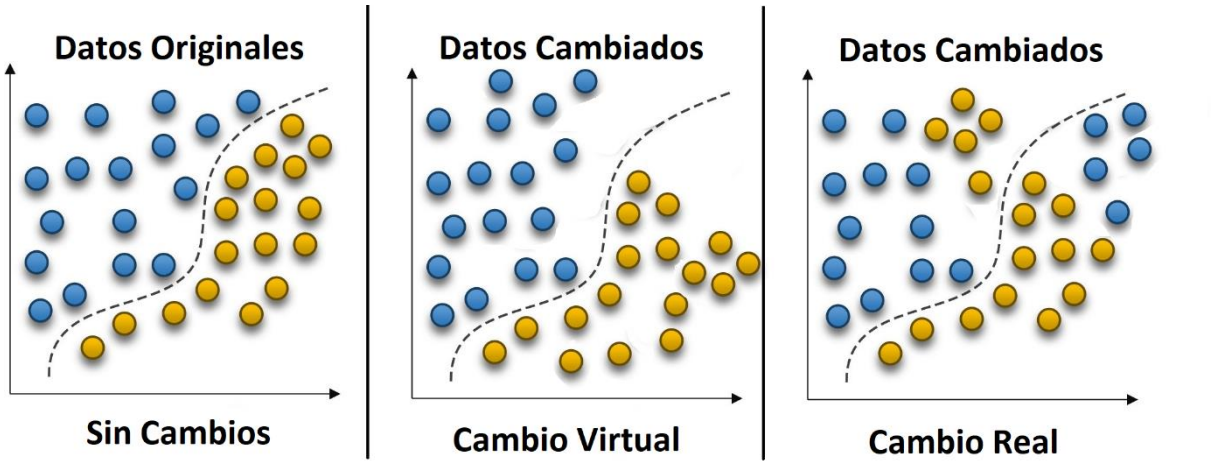

Figura 29.- Cambios reales y virtuales. Modificado a partir de: (Ali Pesaranghader, 2017).

También se almacena el mínimo y el máximo vistos hasta la fecha para cada variable para determinar su dominio. Entonces, en el momento en que el dominio de una variable se expande, sus etiquetas lingüísticas asociadas se recalculan para adaptar las etiquetas lingüísticas al nuevo dominio.

3.2.5.3. Proceso de aprendizaje.

Los patrones emergentes descriptivos deberían de maximizar 3 características principales para ser relevantes al experto:

1. Debe ser tan preciso como se pueda.
2. Debería de cubrir el máximo número de instancias de la clase.
3. La ganancia de información que genere debería ser relevante al experto.

El método propuesto se basa en el algoritmo MOCell (Nebro et al., 2009), en el cual la población se organiza en una cuadricula de nxn individuos y se establecen vecindarios solapados entre sí donde cada individuo únicamente interactúa con los vecindarios en los que esté incluido. Gracias a estos vecindarios solapados, el método es capaz de converger de manera controlada.

3.2.5.4. Codificación de los cromosomas.

El proceso evolutivo se encarga de aprender el KB, que está formado por los patrones emergentes que describen el comportamiento del flujo de datos.

Cada cromosoma representa un potencial EP a ser extraído. Este enfoque es adecuado para el problema, siguiendo una representación en forma normal disyuntiva (DNF). Están compuestos por un antecedente que une elementos que pertenecen a la misma variable por disyunciones, mientras que diferentes variables se unen por medio de conjunciones. La parte consecuente representa la clase que el EP está describiendo.

Teniendo en cuenta lo anterior, un cromosoma p es codificado como $p=(\lambda, c)$, donde λ representa el antecedente y es un vector binario de longitud L (el nº de conjuntos de elementos del problema) y $c \in \mathbb{N}$ es la clase del patrón (anterior o actual). L se determina por la cantidad de categorías para cada variable nominal más el número de conjuntos difusos definidos por cada variable numérica.

3.2.5.5. Operadores genéticos.

La población inicial se crea introduciendo un modelo de patrones M para ir actualizándolo. Luego, el resto de cromosomas son inicializados aleatoriamente hasta que se llega al tamaño de la población. Después de esto, se comienza el proceso evolutivo. En esta fase, cada cromosoma solamente interactúa con su vecindario a través de los siguientes operadores genéticos.

- **Selección**: torneo binario clásico. Se eligen dos individuos y el que más calidad tenga es el que se selecciona para cruzarse.
- **Cruce**: cruce en dos puntos. Permite intensificar la búsqueda en el área entre los dos padres y por tanto dentro del mismo vecindario.
- **Mutación**: se puede o eliminar una variable entera aleatoriamente o cambiar un bit del cromosoma. Ambos con la misma probabilidad de ocurrir.

3.2.5.6. Método de reinicio.

Una de las características de un modelo de patrones emergentes es que debería de describir la máxima cantidad de instancias evitando redundancias, ya que incrementan la complejidad del modelo. Así que, es deseable que los patrones extraídos se extiendan a través del espacio para mejorar su descripción. Para conseguir esto se aplica un mecanismo de reinicio el cual se lanza cuando la población actual no es capaz de cubrir nuevos ejemplos no cubiertos para al menos el 25% del total de evaluaciones (la población se estanca). Después de eso, las redundancias se eliminan del conjunto de soluciones candidatas por medio de un procedimiento basado en la ratio de cobertura (Li et al., 2014).

Una vez terminado, los cromosomas que hayan sobrevivido actualizarán el archivo A, mediante el criterio de no dominancia. Los individuos restantes que queden hasta llegar al tamaño de la población serán generados siguiendo un esquema cuyo objetivo es explorar zonas aún no cubiertas, permitiendo así ampliar la cobertura del espacio de búsqueda reduciendo a su vez la redundancia de los patrones extraídos.

3.2.5.7. Función de evaluación.

Como la extracción de patrones emergentes es un problema multi-objetivo, algunos objetivos deben ser definidos para poder determinar la

calidad de los individuos. Algunas de estas métricas son el índice de crecimiento, generalidad… etc. Esto se computa mediante el uso de la tabla de contingencias del patrón P. El problema de este enfoque está en que, para calcular dicha tabla, se requiere recorrer todo el bloque de datos para cada potencial patrón P, algo totalmente inviable en un escenario de flujos masivos de datos.

Para solucionar el problema y poder realizarlo de manera distribuida, la tabla de contingencias se calcula mediante el método Bit-LUT (Garcia-Vico et al., 2020). Básicamente, el procedimiento está basado en la premisa de que las definiciones de las etiquetas lingüísticas no cambian en cada trozo de datos. Es por ello que se aprovecha esta premisa y se determina si las instancias del bloque están cubiertas o no por las diferentes etiquetas lingüísticas definidas mediante operaciones de bits, aumentando significativamente el rendimiento en la evaluación.

La principal ventaja es que este procedimiento se puede calcular de manera distribuida mediante el cálculo de tablas de contingencias parciales con respecto a los datos que disponga cada nodo y la unión o suma posterior de todas esas tablas parciales en el clúster.

3.2.6. **Esquema operacional.**

El proceso que sigue el algoritmo se puede ver, en forma de pseudocódigo, en el Código 2. Además, también se proporciona la Figura 30 como apoyo a la explicación del funcionamiento del algoritmo.

```
1  Entrada:
2     D: Un lote de datos recogidos del flujo
3     Mt-1: Modelo de patrones anterior
4     uC,uS: Umbral de confianza y soporte, para el sistema de
      monitorización de cambios
5     maxGen: Máximo número de generaciones en el proceso evolutivo
6
7  Salida:
8     Mt: El nuevo modelo de patrones
9  ------------------------------------------------------------
10 aprender <-- falso
11 Actualizar DB si se requiere
12 SI Mt-1 != 0 ENTONCES
13    Testea Mt-1 usando D
14    aprender <-- ChangeMonitoring(Mt-1, uC, uS)
15 SI NO
16    aprender <-- verdadero
17 FIN SI;
18
19 SI aprender es verdadero ENTONCES
20    g <-- 0
21    A <-- 0
22    Pg <-- Inicialización(Mt-1)
23    Evaluar(Pg)
24    MIENTRAS g < maxGen HAZ
25          Pg+1 <-- paso del algoritmo evolutivo MOCell
26          F <-- OrdenacionPorDominancia(Pg+1)
27          SI F0 no progresa ENTONCES
28                Pg+1 <-- Reinicialización(F0)
29          FIN SI;
30          g <-- g + 1
31    FIN MIENTRAS;
32    M <-- FiltroDeRatiosCobertura(A)
33
34    DEVUELVE M;
35 SI NO
36    DEVUELVE Mt-1;
37 FIN SI;
```

Código 2.- Pseudocódigo del esquema operacional del algoritmo.

Una vez se tiene el bloque de datos sacado de la memoria, se actualiza la DB o por los métodos por defecto o por la inclusión de conocimiento experto. Si no se trata de la primera ejecución y se dispone de un modelo anterior, se testea por el procedimiento de test-then-train.

El `ChangeMonitoring` (línea 14) es el proceso en el que se mira si la media de la confianza y el soporte están por debajo de un umbral para devolver que se debe lanzar el proceso de aprendizaje.

Si no hay que lanzar el proceso de aprendizaje, se devuelve el modelo que sigue siendo válido (línea 36). Pero si no, lo primero que se hace es poner el contador de generaciones a 0 y el archivo (élite) a vacío (líneas 20 y 21). En la inicialización de la población se debe incluir al modelo anterior y luego rellenar hasta el tamaño de la población deseada.

El algoritmo propuesto empieza con un frente de pareto vacío A mientras la población inicial es creada aleatoriamente y organizada en la cuadrícula nxn. Acto seguido, se aplican los operadores genéticos sucesivamente hasta una condición de parada. En el bucle de reproducción del proceso evolutivo se aplica el mismo proceso que el algoritmo MOCell. Para cada cromosoma que pertenezca a la población: primero se determina su vecindario y luego se escogen un número de padres del mismo vecindario por el operador de selección. Estos padres se cruzan de acuerdo al operador de cruce establecido para obtener el hijo y este muta si es necesario con el operador de mutación escogido. Este hijo reemplaza p si es dominado por el susodicho o, si no lo domina, si su distancia de "*crowding*" es menor. La distancia de "*crowding*" es la distancia que hay entre los dos vecinos más próximos para cada una de las funciones objetivo. Este hijo también es introducido en A si pertenece al frente de pareto.

Finalmente, se reintroducen m individuos de A (frente de pareto no dominados) en la población reemplazando de manera aleatoria m individuos de la población. Con esto último se permite la intensificación en áreas prometedoras en el espacio de búsqueda.

Una vez se termina y se dispone de la siguiente población, se aplica el elitismo con el archivo A (uno aleatorio de P por uno aleatorio de A) y si la población no mejora se reinicializa con el proceso de ratios de cobertura.

Para terminar, siempre se ejecuta el filtrado para eliminar alguna redundancia que haya podido surgir (línea 32) y el resultado es devuelto al experto como modelo nuevo y actual.

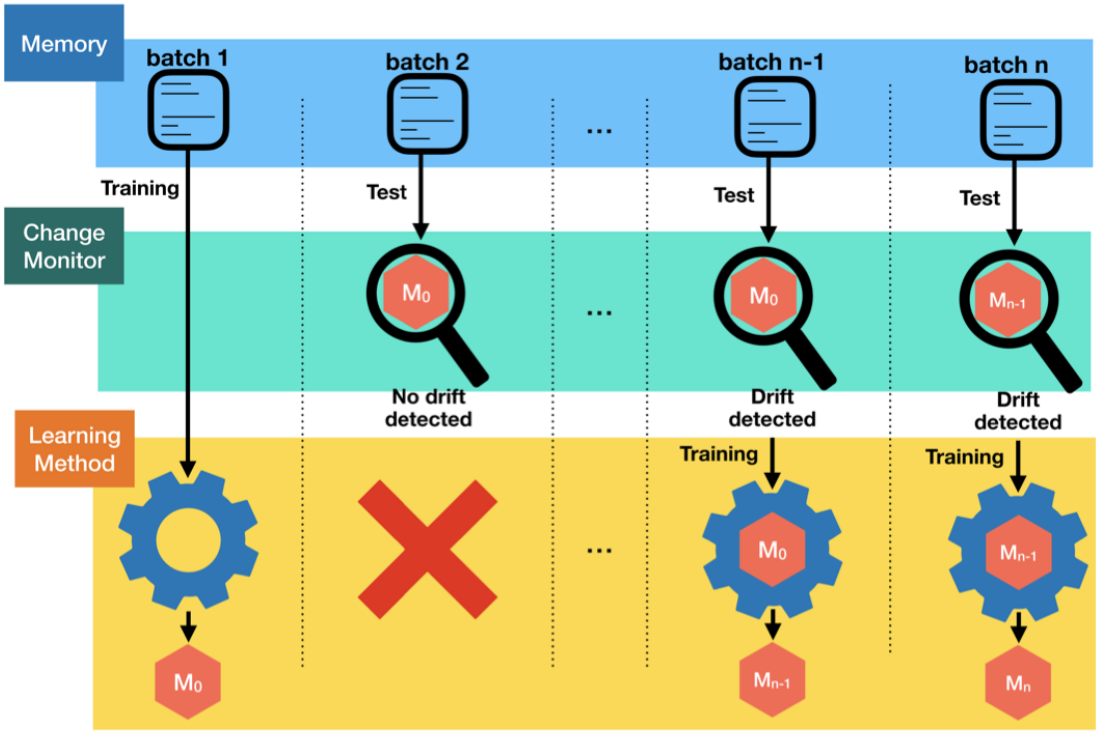

Figura 30.- Esquema operacional del algoritmo. Fuente: (Ángel M. García-Vico et al., 2021).

Capítulo 4.- Implementación.

Este capítulo presentará el proceso de implementación que se ha llevado a cabo donde se encontrarán varias secciones. Para comenzar, en las secciones 4.1, y 4.2 se centran en la plataforma Home Assistant para incorporar: Apache Kafka y la gestión de la base de datos, respectivamente. Continuando con la sección 4.3 que proporcionará la capacidad de afrontar y tener una cierta tolerancia a los fallos que se pueda prever que van a ocurrir. Finalmente, la sección 4.4 muestra el código que ha sido implementado para poder generar el conocimiento que se busca. Cabe destacar que, una de las partes relevantes es la preparación de componentes. Esta parte puede consultarse en la sección 8.2.

4.1. Incorporación de Apache Kafka a Home Assistant.

Una vez se tienen todas las estaciones conectadas con Home Assistant y recibiendo datos, el siguiente paso es conseguir que se publiquen dichos datos en el bróker de Apache Kafka que se instaló anteriormente. Para ello, la integración que se proporciona por medio de los desarrolladores tiene bastantes limitaciones. Esta se encuentra aún en una etapa de desarrollo temprana donde apenas permite la modificación del comportamiento que tiene. Para activarla es necesario abrir y editar el fichero de configuración correspondiente y añadir las líneas que se muestran en el Código 3, usando el add-on `File editor`.

```
apache_kafka:
  ip_address: 192.168.1.29
  port: 9092
  topic: demo02
  filter:
    include_entities:
```

Código 3.- Líneas de configuración de Apache Kafka.

De esta manera, se establece mediante los tres primeros parámetros aquello referente al bróker:

- La dirección IP donde se ubica el bróker. Será la misma que la de la plataforma Home Assistant ya que ambos se están ejecutando en la misma Raspberry Pi.
- El tópico donde se desean publicar los datos de los sensores.
- Y con la opción "*filter*", se limitan aquellas entidades y dominios que vayan a publicarse. En este caso se filtrará únicamente por los identificadores de las entidades.

4.2. Almacenamiento y recogida de datos de los sensores.

La información que se va almacenando en la base de datos de Home Assistant es bastante diversa y compleja. Mediante la activación de una integración nativa se puede configurar y personalizar aquello que se quiere guardar, durante cuánto tiempo y si se desea que se vaya limpiando periódicamente.

Lo primero que se hará será activar la integración de "*recorder*" en el fichero de configuración para poder personalizar la recogida de datos. Se establece que no se vaya auto-limpiando cada día con: "*auto_purge: false*". Con esto, lo que se consigue es que la base de datos, aunque se vaya haciendo cada vez más grande, no vaya borrando aquellos datos que se puedan considerar caducados para incluir los nuevos. Permitiendo almacenar todo el conocimiento en bruto en ella para luego poder exportarlo y explotarlo con el algoritmo. Acto seguido, se incluyen las entidades que se quieren guardar. En este caso, solamente se requieren los datos de los sensores y de las luces, quedando el apartado de la base de datos de la siguiente manera:

```yaml
recorder:
  auto_purge: false
  include:
    domains:
      - sensor
      - light
      - climate
```

Código 4.- Configuración necesaria para gestionar y personalizar el uso de la base de datos.

Con estos sencillos pasos, ya se tiene establecida una persistencia robusta totalmente automática y transparente para el usuario. Además, la base de datos solamente almacenará aquello que se le indique, con lo que se controlará también, en cierta medida, el posible aumento sin límites de la misma gracias a estos filtrados.

4.3. Tolerancia a fallos.

Como ya se comentó anteriormente, es posible que haya fallos y problemas de conectividad al tratarse de dispositivos inalámbricos repartidos por la casa. Para solucionarlo es tan simple como recargar la integración del dispositivo y para ello se creará una automatización que detecte esto y recargue las entidades caídas. No obstante, es posible que, debido a un corte del suministro eléctrico haya que reiniciar el sistema de manera automática.

4.3.1. Recarga de integraciones.

1. Se crea la automatización que detectará la desconexión con la plataforma y que recarga las integraciones mediante la interfaz gráfica de usuario, en este caso será para la integración de la placa del salón:

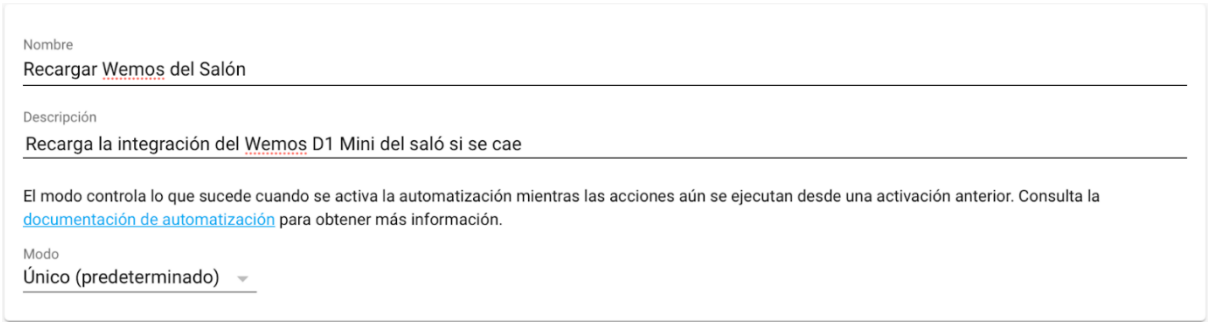

Figura 31.- Paso 1 para el restablecimiento de la placa del salón. Fuente: Elaboración propia.

2. El siguiente paso se realizará mediante la escritura del "*.yaml*" por facilidad de lectura y presentación del mismo:

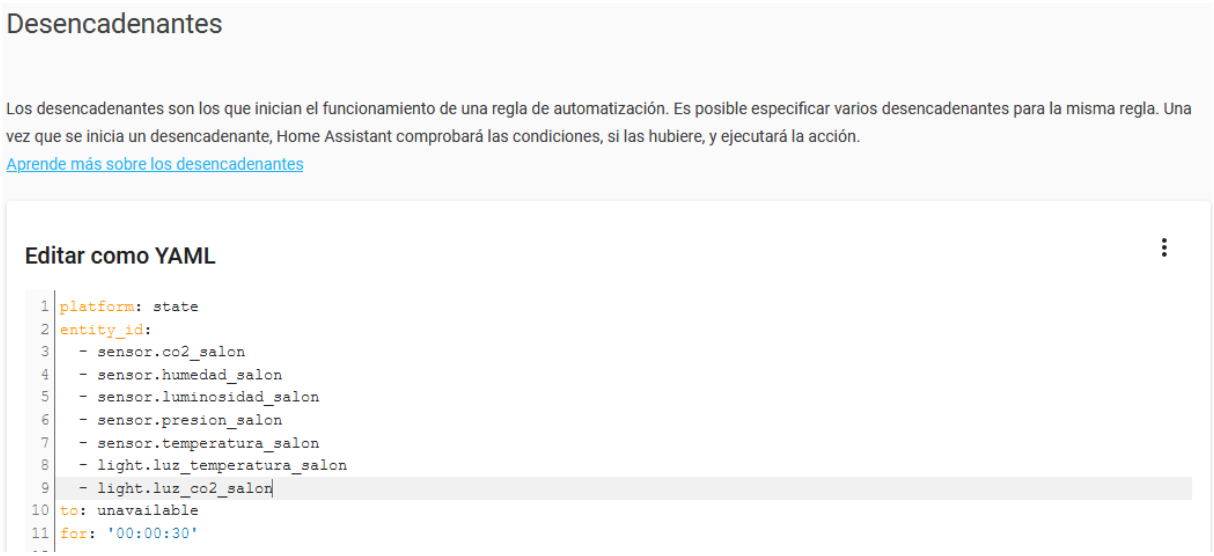

Figura 32.- Paso 2 para el restablecimiento de la placa del salón. Fuente: Elaboración propia.

Así, si alguna de las entidades o sensores pasara a no estar disponible durante 30 segundos se desencadenarían las acciones que se ven reflejadas en la Figura 33.

3. Acto seguido se establecerían las condiciones, pero en este caso no se establecen ya que no se requiere de ninguna.

4. El último paso es establecer las acciones. En este caso, se llamará al servicio que se creará a continuación.

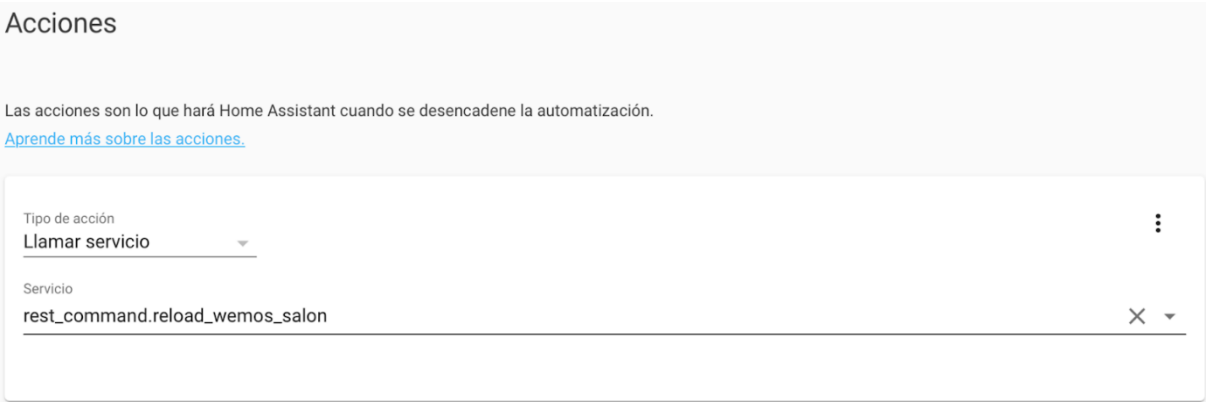

Figura 33.- Paso 3 para el restablecimiento de la placa del salón. Fuente: Elaboración propia.

5. Se termina guardando el proceso.

Ahora, solamente quedaría definir qué hace la acción que se ha especificado en el campo de los servicios a llamar. Para definirla hay que dirigirse al fichero de configuración de la plataforma y escribir aquello que se puede leer en el Código 5.

```
rest_command:
  reload_wemos_salon:
    url: http://192.168.1.29:8123/api/config/config_entries/entry/8d...85/reload
    method: POST
    headers:
      authorization: 'Bearer eyJ0eXA ... A_PWwK2SWeNr8'
      content-type: application/json
```

Código 5.- Acción definida en el fichero de configuración para la recarga de integraciones.

Los elementos que habría que modificar serían:

- En la **url:**
  - después de "*https://*" se establece la dirección **IP** donde esté Home Assistant.
  - en el penúltimo campo, se establece el **identificador** de la entidad que se quiere recargar.
- Un **token de larga duración** generado previamente.

Para poder escribir dicha acción, es necesario encontrar antes aquellos elementos. Por lo tanto, se siguen unos pasos para poder crear la recarga de cada integración:

1. ‘`rest_command`:’ para activar el servicio que se va a usar.
2. Se busca dentro del fichero ubicado en: `/root/config/.storage/core.config_entries` aquel “*`entry_id`*” que se corresponda con la integración de ESPHome del Wemos D1 Mini que se busque, en este caso el del salón.


```
{
    "entry_id": "8d6920994ab14fab6ee311f6a43f6285",
    "version": 1,
    "domain": "esphome",
    "title": "wd1m_salon",
    "data": {
        "host": "192.168.0.22",
        "port": 6053,
        "password": ""
    },
    "options": {},
    "system_options": {
        "disable_new_entities": false
    },
    "source": "zeroconf",
    "connection_class": "local_push",
    "unique_id": "wd1m_salon",
    "disabled_by": null
},
```


Figura 34.- Información referente al Wemos del salón que aparece en el fichero core.config_entries. Se usará la primera línea. Fuente: Elaboración propia.

3. Otro paso necesario es la creación de un token de larga duración para que se permita a los scripts interactuar con la instancia de Home Assistant.

    a. Para ello hay que dirigirse al perfil de usuario abajo a la izquierda, en la última opción llamada “Tokens de acceso de larga duración”. Se le proporciona un nombre cualquiera y se copia el token. Un ejemplo de token sería el siguiente: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJmNzZlOTQ1NGMwZDk0MjVjOGRhMmQ5NWEzYmI1Y2MxZSIsImlhdCI6MTYxOTg2MzIwNiwiZXhwIjoxOTM1MjIzMjA2fQ.-V4GEzZpW1i5ngxok8z1fq-2Q7lt4NA_PWwK2SWeNr8`

4. Se guarda el fichero y se repiten los pasos citados para todas las demás integraciones de las diversas placas Wemos obteniendo el resultado mostrado en el código 6.

```
rest_command:
  reload_wemos_cocina:
    url: http://192.168.1.29:8123/api/config/config_entries/entry/bb...09/reload
    method: POST
    headers:
      authorization: 'Bearer eyJ0eXA ... A_PWwK2SWeNr8'
      content-type: application/json
  reload_wemos_salon:
    url: http://192.168.1.29:8123/api/config/config_entries/entry/8d...85/reload
    method: POST
    headers:
      authorization: 'Bearer eyJ0eXA ... A_PWwK2SWeNr8'
      content-type: application/json
  reload_wemos_habitacion:
    url: http://192.168.1.29:8123/api/config/config_entries/entry/3c...26/reload
    method: POST
    headers:
      authorization: 'Bearer eyJ0eXA ... A_PWwK2SWeNr8'
      content-type: application/json
  reload_wemos_habitacion_2:
    url: http://192.168.1.29:8123/api/config/config_entries/entry/ab...c0/reload
    method: POST
    headers:
      authorization: 'Bearer eyJ0eXA ... A_PWwK2SWeNr8'
      content-type: application/json
```

Código 6.- Resultado final de las acciones de recarga de integraciones.

4.3.2. Reinicio del núcleo de Home Assistant.

A veces es posible que los fallos no se produzcan por la conectividad entre la plataforma y los dispositivos. Hay veces en las que recargar las integraciones no soluciona el problema de que aparezcan como no disponibles, por lo que será necesario realizar un reinicio del núcleo del sistema. Esto es factible, ya que el reinicio de la plataforma al estar en un contenedor Docker se hace en algo menos de 1 minuto, por lo que la pérdida de datos no es significativa. Para realizar esto, se crea otra automatización:

Figura 35.- Paso 1 para la automatización de reinicio del sistema. Fuente: Elaboración propia.

```yaml
trigger:
  - platform: state
    entity_id: >-
      sensor.co2_cocina,sensor.humedad_cocina,sensor.luminosidad_cocina,sen
    to: unavailable
    for: '00:02:00'
  - platform: state
    entity_id: >-
      sensor.co2_habitacion,sensor.humedad_habitacion,sensor.luminosidad_hal
    to: unavailable
    for: '00:02:00'
  - platform: state
    entity_id: >-
      sensor.co2_habitacion_2,sensor.humedad_habitacion_2,sensor.luminosida
    to: unavailable
    for: '00:02:00'
  - platform: state
    entity_id: >-
      sensor.co2_salon,sensor.humedad_salon,sensor.luminosidad_salon,sensor
    to: unavailable
    for: '00:02:00'
condition: []
```

Figura 36.- Paso 2 para la automatización de reinicio del sistema. Fuente: Elaboración propia.

Como se aprecia en la Figura 36, si alguna de las entidades o sensores pasa a no estar disponible durante 2 minutos se desencadenarían las acciones que se ven reflejadas en la Figura 37. Además, también se aprecia que no hay condiciones previas.

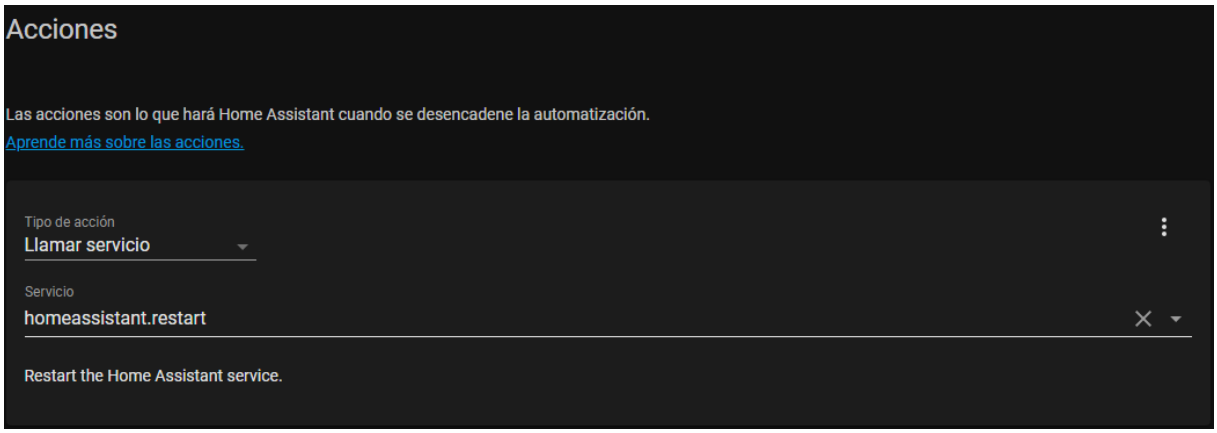

Figura 37.- Paso 3 para la automatización de reinicio del sistema. Fuente: Elaboración propia.

Al reiniciarse el sistema y volver a cargar todo, todos los dispositivos estarán conectados con la plataforma y no se perderán datos de sensores que puedan quedarse huérfanos.

4.3.3. Recuperación ante cortes del suministro eléctrico.

Lo necesario para poder recuperarse de un fallo de este tipo es poder levantar el contenedor de Docker que contiene Home Assistant y volver a poner en marcha el bróker de Apache Kafka.

Para lo primero la solución al problema se encuentra establecida en la configuración por defecto de Docker. Este se inicia automáticamente junto con el sistema y será él mismo el que se encargará de ejecutar aquellos contenedores que tengan en su configuración "`restart=always`". Así, se consigue que tanto Home Assistant, como Heimdall y Portainer se ejecuten y arranquen solos junto con el Sistema Operativo. Cabe destacar que una de las partes relevantes es el proceso de instalación detallado en la sección 8.1.2.

Para lo segundo se procederá a crear un servicio similar al que tiene Docker para ser arrancado junto con el sistema. En el fichero *start_kafka_script.sh* se añaden las siguientes líneas que se muestran en el Código 7.

```
#!/bin/bash

~/proyectos/kafka_2.13-2.7.0/bin/zookeeper-server.start.sh
config/zookeeper.properties
sleep 5
~/proyectos/kafka_2.13-2.7.0/bin/kafka-server-start.sh
config/server.properties
```

Código 7.- Script que inicia el bróker de Apache Kafka.

Dicho fichero se guardará en la carpeta de proyectos que cuelga del directorio home del usuario del servidor. Acto seguido, en la ruta */etc/systemd/system/* se creará el fichero "*.service*" llamado *start_kafka_service.service* con el Código 8.

```
[Unit]
Description=Inicia el servidor de Apache Kafka
Before=docker.service

[Service]
TimeoutStartSec=1
ExecStart=/home/pi/proyectos/start_kafka_script.sh

[Install]
WantedBy=multi-user.target
```

Código 8.- Servicio de arranque de Apache Kafka.

Para finalizar, hay que indicar a la utilidad *systemctl*, que al arrancar el sistema arranque también el servicio recién creado que se encarga de ejecutar el script personalizado. El sistema procederá entonces siguiendo la Figura 38.

```
1. sudo systemctl enable start_kafka_service.service
```

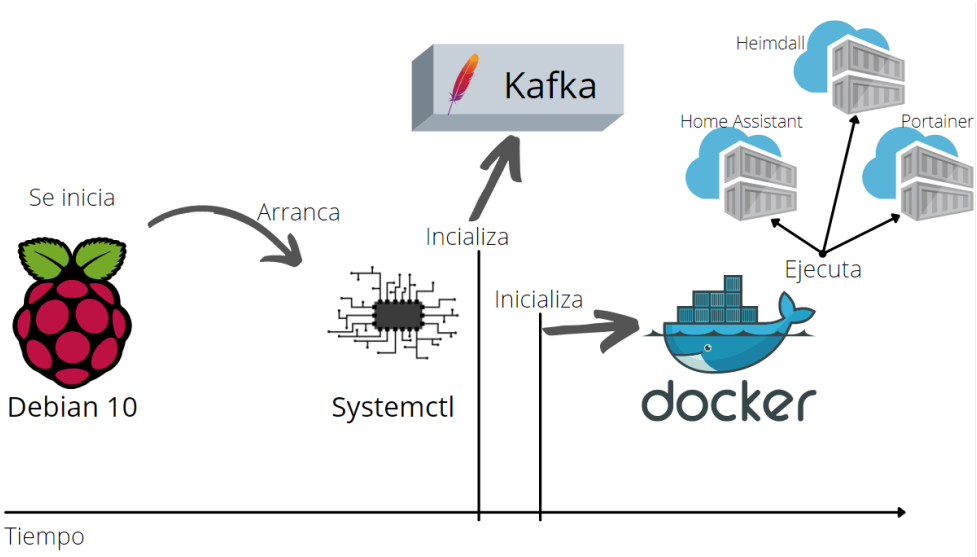

Figura 38.- Proceso de inicio de la Raspberry y los distintos procesos y elementos que se suceden en un orden determinado. Fuente: Elaboración propia.

4.4. Proyecto CMTEP.

Este nuevo algoritmo llamado CMTEP (Change Mining Through Emerging Pattern) se ha diseñado a partir del algoritmo FEPDS (Á. M. García-Vico et al., 2020) para adaptarlo a la minería de cambio. Para ello lo primero que se realizó fue descargar del repositorio de GitHub (SIMIDAT, 2021) el proyecto con el planteamiento inicial. En este trabajo se trata con ventanas de tiempo como clase en vez de tener una clase predefinida, de esta manera, se tienen datos sin clase y se enfoca el trabajo a manejar datos no supervisados. Estos datos no supervisados se etiquetarán en ventanas de tiempo. Por consiguiente, una de las primeras modificaciones que se debe de realizar es el fichero de cabecera donde se especifican las distintas variables a manejar:

```
@RELATION medidas-home-assistant

@ATTRIBUTE sensor.co2_cocina NUMERIC
@ATTRIBUTE sensor.humedad_cocina NUMERIC
@ATTRIBUTE sensor.luminosidad_cocina NUMERIC
@ATTRIBUTE sensor.presion_cocina NUMERIC
@ATTRIBUTE sensor.temperatura_cocina NUMERIC

@ATTRIBUTE sensor.co2_habitacion NUMERIC
@ATTRIBUTE sensor.humedad_habitacion NUMERIC
@ATTRIBUTE sensor.luminosidad_habitacion NUMERIC
@ATTRIBUTE sensor.presion_habitacion NUMERIC
@ATTRIBUTE sensor.temperatura_habitacion NUMERIC

@ATTRIBUTE sensor.co2_habitacion_2 NUMERIC
@ATTRIBUTE sensor.humedad_habitacion_2 NUMERIC
@ATTRIBUTE sensor.luminosidad_habitacion_2 NUMERIC
@ATTRIBUTE sensor.presion_habitacion_2 NUMERIC
@ATTRIBUTE sensor.temperatura_habitacion_2 NUMERIC

@ATTRIBUTE sensor.co2_salon NUMERIC
@ATTRIBUTE sensor.humedad_salon NUMERIC
@ATTRIBUTE sensor.luminosidad_salon NUMERIC
@ATTRIBUTE sensor.presion_salon NUMERIC
@ATTRIBUTE sensor.temperatura_salon NUMERIC


@ATTRIBUTE date {anterior, actual}

@DATA
```

Código 9.- Fichero HEAD.arff

En dicho fichero, se especifican las distintas variables que se manejarán. Por lo que si hubiera que añadir un nuevo sensor solamente se tendría que añadir una nueva etiqueta de atributo y el nombre del sensor que se haya añadido.

La siguiente modificación realizada es la toma del flujo de datos que recibe desde la publicación de Kafka. También está la posibilidad de simular dicho flujo sin usar la propia aplicación mediante el empleo de una base de datos sintética. No obstante, esto resulta transparente a efectos del algoritmo, que únicamente recibe los datos y los procesa sin importar su procedencia. En el Código 10 se muestra el esquema operacional de esta modificación.

```
t <- 0
v <- 0
Reglas para enviar ← ∅
cargarConfiguración()
cargarSensores()
instanciasActuales <- precargarBaseDatos() //precarga también las
instanciasAnteriores
unido <- ∅
REPETIR
      SI kafka ENTONCES
            recogerFlujoDeDatos(TiempoVentanaDeslizante)
            agrupados <- agrupaMedidas(tasaRefrescoSensores)
            unido <- union(agrupados, instanciasAnteriores(t))
      SINO
            unido <- union(instanciasActuales(t),
            instanciasAnteriores(t))
            recogerFlujoDeDatosSimulado(unido)
      FIN SI;

      SI flujoVacio() ENTONCES v <- v + 1

      procesaRDD(unido)
      t <- t + 1
HASTA v == MAX_BATCHES_VACÍOS
```

Código 10.- Pseudocódigo del esquema operacional principal del algoritmo.

Para simplificar la lectura, el pseudocódigo del método `procesa RDD()` se ha separado en otro distinto. Al igual que los métodos de la precarga de la base de datos y la lectura de los sensores que haya disponibles. Por tanto, en el Código 11 se encuentra el pseudocódigo de procesar cada RDD, en el Código 12 el de la precarga de la base de datos y finalmente en el Código 13 la lectura de sensores.

Además, para evitar un exceso de datos que pueda confundir al algoritmo por tener demasiadas instancias clasificadas como anteriores

frente a pocas clasificadas como actuales, se dispone de un método que agrupa las medidas en intervalos de tiempo de la tasa de refresco de los sensores especificada como parámetro. En dicho método, se genera siempre una medida por cada sensor. Esto se lleva a cabo ya que, habitualmente, el envío de datos por parte de los sensores se produce únicamente cuando hay un cambio en la magnitud medida. Si no se recibe un dato por parte de un sensor éste no puede considerarse como un valor perdido, pues esto indica que su magnitud no ha cambiado. Por tanto, el valor que le corresponde es el mismo que en el instante de tiempo anterior. El pseudocódigo de este proceso se puede apreciar en el Código 14, la base de datos a usar ya trae consigo esta característica, por lo que no es necesario preprocesarla.

```
lecturaRDD(rdd, cabecera)
SI primerLote ENTONCES
      generarAtributos
      generarConjuntosDifusos
SINO
      PARA CADA variable HAZ
            SI atributo es numérico ENTONCES
                  minAntiguo <- getMin(conjuntoDifuso(variable))
                  maxAntiguo <- getMax(conjuntoDifuso(variable))
                  dato <- atributo.toDouble
                  minActual <- dato.min
                  maxActual <- dato.max

                  SI minActual < minAntiguo AND maxActual > maxAntiguo
ENTONCES
                        actualizaAmbos
                  SINO minActual < minAntiguo AND maxActual < maxAntiguo
ENTONCES
                        actualizaMínimo
                  SINO minActual > minAntiguo AND maxActual > maxActual
ENTONCES
                        actualizaMáximo
                  FIN SI;
            FIN SI;
      FIN PARA;
FIN SI;
PARA CADA atributo HAZ crearBitset(numeroInstancias)
PARA CADA instancia HAZ actualizaBitset()
actualizaClasesProblema()

ejecutaAlgoritmo(problema)
```

Código 11.- Pseudocódigo procesaRDD.

```
resultado <- ArrayVacío()
instanciasDiasEspecificado <- ArrayVacío()
instanciasAnteriores <- ArrayVacío()

fActual <- FechaDe(añoInicio, mesInicio, diaInicio, horaInicio)
fFutura <- FechaDe(fechaActual + diasHaciaDelante)
fPasada <- FechaDe(fechaActual - diasHaciaAtras)

APERTURA FICHERO de la base de datos
      PARA CADA linea del fichero
            instancia <- parsear(linea)
            medida <- instancia.datos
            fInstancia <- instancia.fecha

            SI fInstancia > fActual AND fInstancia < fFutura ENTONCES
                  instanciasDiasEspecificado += medida.toWeka(actual)
            FIN SI;

            SI historiaCompleta ENTONCES
                  SI filtroMismoDiaSemana ENTONCES
                        SI esMismoDiaSemana(diaInicio, instancia.dia) AND fInstancia <
                        fActual ENTONCES
                              instanciasAnteriores += medida.toWeka(anterior)
                        FIN SI;
                  SINO ENTONCES
                        SI fInstancia < fActual ENTONCES
                              instanciasAnteriores += medida.toWeka(anterior)
                        FIN SI;
                  FIN SI;
            SINO ENTONCES
                  SI fInstancia > fPasada AND fInstancia < fActual ENTONCES
                        SI filtroMismoDiaSemana ENTONCES
                              SI esMismoDiaSemana(diaInicio, instancia.dia) ENTONCES
                                    instanciasAnteriores += medida.toWeka(anterior)
                              FIN SI;
                        SINO ENTONCES
                              instanciasAnteriores += medida.toWeka(anterior)
                        FIN SI;
                  FIN SI;
            FIN SI;
      FIN PARA;
CIERRE FICHERO;
PARA CADA HORA
      rdd <- crearRDD(instanciasAnteriores)
      rdd2 <- crearRDD(instanciasDiasEspecificado)
      instanciasGlobales += rdd
      resultado += rdd2
FIN PARA;

return resultado
```

Código 12.- Pseudocódigo precargaBBDD.

```
sensores <- ArrayVacío()

APERTURA FICHERO sensores
      PARA CADA linea del fichero
            sensor <- parsear(linea)
            sensores += sensor
      FIN PARA;
CIERRE FICHERO;
```

Código 13.- Pseudocódigo lectura de sensores.

```
medidasActuales
últimosValores <- Ø
medidasAgrupadas <- Ø
medidasResultado <- Ø
medidasActuales <- ordenarPorFecha(medidasActuales)
horaUmbral <- medidasActuales.getFirst()
k <- 0

PARA CADA medida en medidasActuales HAZ
      SI medida.hora es anterior a horaUmbral ENTONCES
            medidasAgrupadas(k) <- medida
      SINO ENTONCES
            horaUmbral <- (medida.hora + tasaRefrescoSensores)
            k <- k + 1
            medidasAgrupadas(k) <- medida
      FIN SI;
FIN PARA;

PARA CADA conjuntoMedidas en medidasAgrupadas HAZ
      PARA CADA medida en conjuntoMedidas HAZ
            actualizaÚltimosValores(últimosValores, medida)
      FIN PARA;
      PARA CADA sensor HAZ
            SI últimosValores != valorNulo ENTONCES //Para los primeros
            bloques
                  medidasResultado <- últimoValor(i)
            FIN SI;
      FIN PARA;

DEVUELVE medidasResultado;
```

Código 14.- Pseudocódigo del método de agrupamiento de valores de los sensores.

El mismo esquema que representa el pseudocódigo presentado en el Código 14 fue utilizado para la modificación de la base de datos. La única diferencia es que las instancias se leen del fichero, se agrupan y luego se escriben en un nuevo fichero. Esto se realizó con otro proyecto escrito en Scala diferenciado del algoritmo actual.

Otra modificación necesaria es la inclusión de la clase que representará y albergará la lógica de las medidas. Este tipo de dato abstracto se presentó en la sección 3.2.1 de manera teórica, la implementación de la misma se resume en la inclusión de los atributos necesarios que recibe mediante el constructor. Con respecto a los métodos, se describen dos: `toWeka()` y `toJSON()`, ambos para transformar la medida a los respectivos formatos.

Para hacer escalable el sistema, el método `toWeka()` necesita recibir el vector con los sensores para hallar qué posición ocupa el sensor de la medida en el vector y poder rellenar la cadena con '?' salvo su posición donde se incluirá la propia medida del sensor. Además, recibe la clase como parámetro de entrada, ya sea actual o anterior.

```
indice <- encuentraIndice(sensores, entity_id)
weka <- ""

PARA CADA indice en sensores -> i
      SI i == indice
            weka <- weka + medida + ","
      SINO
            weka <- weka + "?,"
FIN PARA;

weka <- weka + clase + "\n"
return weka
```

Código 15.- Pseudocódigo método toWeka.

Con respecto al método `toJSON()`, se resume en construir y devolver la cadena formateada de la siguiente manera:

```
toJSON() {
      return creaString( entity_id, medida, unidadMedida, friendly_name,
      fecha );
}
```

Código 16.- Método toJSON.

Capítulo 5.- Experimentación.

Este capítulo presentará el proceso de minería de datos realizado tal y como se muestra en la Figura 39. De esta manera, en la sección 5.1 se explicará cómo se gestionan los datos que se reciben y qué datos recibe/envía cada sensor y qué se almacena en la base de datos, además, se detallará el proceso de inyección de datos sintéticos. Acto seguido, en la sección 5.2 se explicarán cómo se han organizado y gestionado los experimentos sintéticos, junto con sus parámetros. Finalmente, en la sección 5.3 se mostrarán los resultados de los experimentos realizados y se analizarán en la sección 5.4.

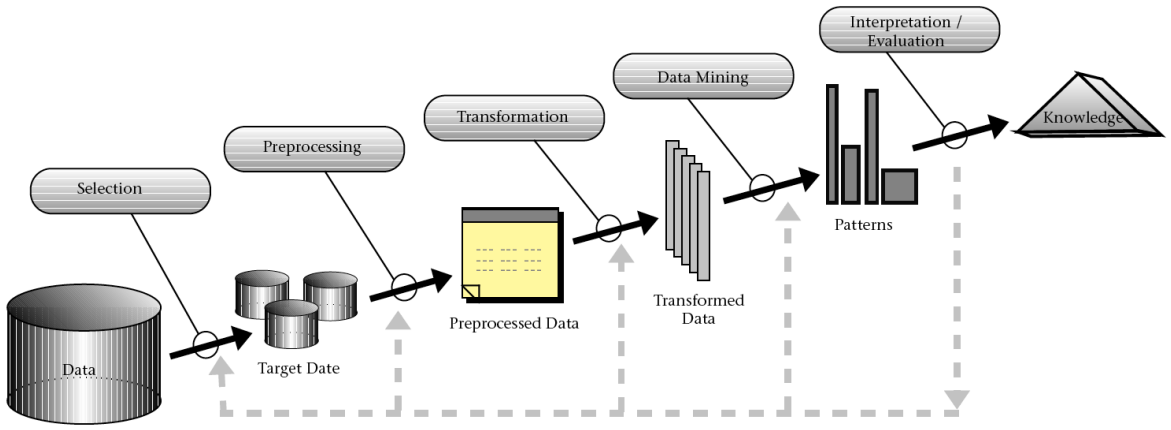

Figura 39.- Proceso de minería de datos. Fuente: (danrodgar.github.io, 2021).

5.1. Preparación de los datos.

Para comenzar, se explicará cómo se gestionan los datos que generan los sensores. Estos siguen un formato unificado y son por tanto los mismos sin importar el sensor con el que esté tratando. Como ya se mostró en el Código 1, se dispone de un JSON de ejemplo. En el Código 17 se puede apreciar el formato general que siguen este tipo de información.

```
{
      "entity_id": "nombre del sensor",
      "state": "medida del sensor",
      "attributes": {
            "unit_of_measurement": "unidad medida del sensor",
            "friendly_name": "nombre amistoso",
            "icon": "icono que se presenta en Home Assistant"
      },
      "last_changed": "fecha de creación",
      "last_updated": "fecha de ultimo refresco",
      "context": {"
            id": "identificador autogenerado",
            "parent_id": autogenerado (null),
            "user_id": autogenerado (null)
      }
}
```

Código 17.- Plantilla formato medidas de los sensores.

La información referente a las medidas numéricas es creada por los sensores y es enviada por ellos mismos a la plataforma, la cual la recibe y genera la información con el formato citado anteriormente. La plataforma es luego la encargada de realizar la publicación en el bróker de Apache Kafka, desde la cual el algoritmo propuesto en este trabajo recibirá los datos. De igual manera, esta información enviada al bróker se guarda en la base de datos interna que tiene el propio Home Assistant. En ambos sitios se gestiona el mismo tipo de información, y, por consiguiente, se puede proceder a crear datos sintéticos y simular que provienen de la plataforma y de Apache Kafka.

5.1.1. **Generación de datos sintéticos.**

Para poder aumentar la cantidad de datos con los que trabajar y no estar limitados al último mes, se decidió crear un generador de datos artificial con el que poder alimentar la base de datos con instancias anteriores al 20 de abril de 2021. En esta fecha se comenzó la recolección de datos verídicos con la plataforma.

Para ello, se han tomado dos enfoques distintos de generadores creados a partir de la base de datos: uno orientado a los datos de abril y mayo donde cada sensor se simula mediante una distribución normal N(μ, σ).

$$N(\mu, \sigma)$$

Donde:

μ = media de cada sensor.

σ = desviación típica entre las medidas de cada sensor.

El otro está orientado a los datos frecuentes de cada mes por horas en Jaén para crear una distribución triangular T(m, mo, M).

$$T(m, mo, M)$$

Donde:

m = valor mínimo encontrado.

$$mo = (m+M)/2$$

M = valor máximo encontrado.

Se decidió utilizar una distribución normal en el caso descrito, ya que se tiene una cierta cantidad de datos verídicos. Por otra parte, se decidió utilizar una distribución triangular para esas fechas debido al hecho de estar ante una ausencia de datos y la relativa facilidad de encontrar valores

máximos y mínimos entre diversos conjuntos de datos de Internet orientados a la meteorología.

5.1.1.1. Distribución normal.

En dicha base de datos, se procede a crear las tablas para cada sensor. Se ejecuta entonces, de manera análoga para cada sensor y zona, la siguiente sentencia SQL reflejada en el Código 18, donde "*nombreHabitación*" será un valor entre los posibles valores, al igual que "*nombreSensor*".

```
CREATE TABLE "nombreHAbitacion_nombreSesnor" (
	"entity_id" VARHCAR(255) NOT NULL,
	"state" FLOAT NOT NULL,
	"last_changed" DATETIME,
	"last_updated" DATETIME
);
```

Código 18.- Sentencia SQL para la creación de cada tabla para cada sensor/zona.

Una vez realizado esto, se rellenan con los valores ejecutando el Código 19. Hay que tener en cuenta que el valor de los estados viene en formato de texto y hay que cambiarlo a un valor numérico decimal y que se ejecutará cambiando el "*entity_id*" por cada uno de los sensores que se tienen y volcando los datos en su respectiva tabla creada anteriormente.

```
INSERT INTO salon_humedad
SELECT entity_id, CAST(state AS float), last_changed, last_updated
FROM states WHERE (entity_id = "sensor.humedad_salon" AND state <>
"unavailable" AND state <> "unknown");
```

Código 19.- Sentencia SQL para el volcado de datos entre tablas.

Además de transformar el texto del estado a un número decimal, se quitan los posibles pocos valores que haya a no disponibles o con el estado desconocido.

```
Resultado: consulta ejecutada con éxito. Tardó 229ms, 34973 filas afectadas
```

Figura 40.- Resultado de la consulta SQL de volcado de datos entre tablas. Fuente: Elaboración propia.

Se aprecia que se tienen alrededor de 35000 valores de medidas que no son nulas y por lo tanto tienen un valor correcto. Se continúa calculando la media y la desviación estándar que tienen los datos con el Código 20.

```
SELECT AVG(state) AS media, AVG((cocina_co2.state - sub.a) *
(cocina_co2.state - sub.a)) as var from cocina_co2,
      (SELECT AVG(state) AS a FROM cocina_co2) AS sub;
```

Código 20.- Sentencia SQL para la extracción de la media y la varianza.

Se consigue la salida de la Figura 41. Hay que tener en cuenta que lo que se obtiene es la varianza, ya que SQLite no tiene la funcionalidad de poder realizar raíces cuadradas.

media	var
705.167480865377	57058.0184567329

Figura 41.- Resultado del cálculo de la media y la varianza. Fuente: Elaboración propia.

Los datos que se obtuvieron se han representado en forma de tabla para facilitar la lectura. Con lo que la Tabla 13 alberga por cada Sensor/Zona la media de las lecturas de dicho sensor en dicha zona y la desviación típica o estándar de dichas lecturas:

Zona	Sensor	Media	Desviación Típica
Cocina	co2	705.17	238.87
	humedad	36.28	9.57
	luminosidad	33.42	32.65
	presión	957.62	3.69
	temperatura	25.78	3.73
Salón	co2	816.47	319.43
	humedad	41.52	9.3
	luminosidad	4.9	9.86
	presión	957.61	3.74
	temperatura	22.08	2.75
Habitación	co2	1583.31	808.07
	humedad	38.84	8.25
	luminosidad	5.33	7.25
	presión	958.42	3.73
	temperatura	23.86	2.87
Habitación 2	co2	1502.54	989.87
	humedad	47.72	10.27
	luminosidad	10.8	18.43
	presión	959.34	3.86
	temperatura	23.74	2.33

Tabla 13.- Resultados de la media y desviación típica de cada sensor/zona.

Con estos datos recabados, ya es posible construir la distribución normal N:

$$N(\mu, \sigma)$$

Donde:

μ = media

σ = desviación típica

Con ella, se podrá crear los objetos generadores artificiales mediante un objeto de Scala.

Para terminar, solamente hace falta un método invocable para obtener una medida que siga la distribución. Dicho método, tendrá dos parámetros de entrada que serán: la zona de la vivienda y el sensor de dicha zona. Por lo que, el pseudocódigo del mismo será el reflejado mediante el Código 21.

```
generaMedida( zona, sensorX ){
      switch( zona ):
            case salón: generadorDeSalónParaSensorX.muestrea()
            case cocina: generadorDeCocinaParaSensorX.muestrea()
            case habitación: generadorDeHabitaciónParaSensorX.muestrea()
            case habitación 2: generadorDeHabitación-
2ParaSensorX.muestrea()

            case default: ERROR en la zona establecida
      }
}
```

Código 21.- Pseudocódigo del método para generar valores artificiales.

5.1.1.2. Distribución triangular.

Para el segundo tipo de generador que se ha creado, se ha tomado la base de datos que se descargó y se exportó la tabla que contiene las medidas registradas por la plataforma a formato CSV para facilitar el trabajo, tanto de cómputo como de gestión. Acto seguido, mediante el uso de distintos datasets orientados a la meteorología (tutiempo.net, 2021) (es.weatherspark.com, 2021) y que además realizan el estudio por horas

de los distintos meses, se buscaron los valores mínimos y máximos de cada magnitud a medir. Se realizó esta división en meses y tramos de una hora, ya que no tiene sentido que se genere, por ejemplo, para la temperatura de la cocina, un valor de 30ºC en enero a las 4 de la mañana.

Cabe destacar que los valores han sido generados para el año 2019/2020 y que deberían haber sido generados para 2020/2021, pero como el tratamiento de los datos se va a resumir en clasificarlos como anteriores da igual la fecha. De esta manera, se pueden obtener los datos artificiales mucho más fácilmente al preguntar por aquellos que sean anteriores a 2021.

Para cada mes, se buscan los máximos y mínimos y se introducen de la manera que se muestra en el Código 22.

```
object Params{
  // ------------------------------------------------------------------------------From 00:00 to 01:00
  private val minCO2_00 = 300; private val maxCO2_00 = 800.0
  private val minHumedad_00 = 30.00; private val maxHumedad_00 = 90.00
  private val minLuminosidad_00 = 1.5; private val maxLuminosidad_00 = 2.5
  private val minPresion_00 = 1015.0; private val  maxPresion_00 = 1032.0
  private val minTemperatura_00 = 9.0; private val  maxTemperatura_00 = 13.0

  // ------------------------------------------------------------------------------From 01:00 to 02:00
  private val minCO2_01 = 300; private val maxCO2_01 = 700.0
  private val minHumedad_01 = 36.28; private val maxHumedad_01 = 90.0
  private val minLuminosidad_01 = 1.5; private val maxLuminosidad_01 = 2.5
  private val minPresion_01 = 1015.0; private val  maxPresion_01 = 1032.0
  private val minTemperatura_01 = 9.0; private val  maxTemperatura_01 = 13.0
```

Código 22.- Inserción de mínimos y máximos para las distintas magnitudes, desde las 00 hasta las 23 horas ambas inclusive.

Y con estos valores se calculan las medias que serán usadas como modas en cada generador, tal y como se muestra en el Código 23. El hecho de que la humedad vaya dividida entre 3 es para que los valores estén más agrupados en torno a valores más reales o verídicos.

```
// ------------------------------------------------------------------------------Means and STDESV for CO2
private val mediaCO2_00 = (maxCO2_00+minCO2_00)/2
private val mediaCO2_01 = (maxCO2_01+minCO2_01)/2
private val mediaCO2_02 = (maxCO2_02+minCO2_02)/2
private val mediaCO2_03 = (maxCO2_03+minCO2_03)/2
private val mediaCO2_04 = (maxCO2_04+minCO2_04)/2
```

Código 23.- Generación de las medias de las distintas magnitudes.

Con estos 3 elementos, ya se es capaz de crear los atributos públicos que se usarán para muestrear la distribución triangular T:

T(m, mo, M)

Donde:

m = valor mínimo encontrado

mo = (m+M)/2

M = valor máximo encontrado

```
//-------------------------------------------------------------------------------Generators
val generadorCO2_00: TriangularDistribution = new TriangularDistribution(minCO2_00, mediaCO2_00, maxCO2_00)
val generadorCO2_01: TriangularDistribution = new TriangularDistribution(minCO2_01, mediaCO2_01, maxCO2_01)
val generadorCO2_02: TriangularDistribution = new TriangularDistribution(minCO2_02, mediaCO2_02, maxCO2_02)
val generadorCO2_03: TriangularDistribution = new TriangularDistribution(minCO2_03, mediaCO2_03, maxCO2_03)
val generadorCO2_04: TriangularDistribution = new TriangularDistribution(minCO2_04, mediaCO2_04, maxCO2_04)
```

Código 24.- Generadores que siguen una distribución triangular T(m, mo, M).

Y, por consiguiente, el código que utilizaría estos generadores es el que se puede leer en el Código 25. Dicho código, genera un vector por cada magnitud que almacena las distintas medidas artificiales, ya en formato CSV para su inclusión en el final del fichero. Solamente se debe tener cuidado de que el parámetro de entrada "*append*" esté configurado a "*true*" para que no sobrescriba el fichero y añada las líneas por el final sin borrar nada.

```
override def run(): Unit = {

  val co2 = getCO2()
  val hum = getHumidities()
  val lum = getLuminosities()
  val press = getPressures()
  val temp = getTemperatures()

  val fw = new FileWriter(FILE, append)
  try {
    for(l <- co2) fw.write(l + "\n")
    for(l <- hum) fw.write(l + "\n")
    for(l <- lum) fw.write(l + "\n")
    for(l <- press) fw.write(l + "\n")
    for(l <- temp) fw.write(l + "\n")
  }finally fw.close()

 }
}
```

Código 25.- Script que genera las medidas y las incorpora al final del fichero de la base de datos previamente exportado a CSV.

El pseudocódigo de cada función que obtiene dichas líneas se puede apreciar en el Código 26. Dicho pseudocódigo es análogo a cada una de las magnitudes a medir.

```
para cada dia
      para cada hora, h
            para cada minuto, m
                        co2 += creaLineaCSV(Cocina, CO2, h, m);
                        co2 += creaLineaCSV(Salón, CO2, h, m);
                        co2 += creaLineaCSV(Habitacion, CO2, h, m);
                        co2 += creaLineaCSV(Habitación2, CO2, h, m);
return co2;
```

Código 26.- Pseudocódigo del método getCO2().[10]

Para terminar este apartado, solamente habría que ejecutar los 12 scripts que se tienen con el mismo fichero de la base de datos y se conseguiría un fichero con todos los valores sintéticos anexionado al final, con el mismo formato y con las fechas dispuestas como si hubiesen sido generados el año anterior. Así la cantidad de datos que se tienen aumenta de un único mes a trece meses completos.

[10] CrearLineaCSV devuelve una línea formateada de la misma manera que las realiza la exportación de la base de datos a formato CSV (separador = ‘;’)

5.2. Experimentos y ejecución del algoritmo.

Con relación a los distintos experimentos que se realizarán se dispone de 2 tipos de posibles ejecuciones: ejecución mediante un flujo de datos continuo con Apache Kafka y la simulación de un flujo de entrada mediante una base de datos precargada. Es posible distinguir dicho comportamiento mediante los parámetros establecidos en la sección 3.2.4, estos parámetros se especificarán para cada experimento.

- Si se escoge la ejecución "real" del sistema mediante Apache Kafka, se puede elegir contrastar con todos los datos almacenados en la base de datos o con los mismos pero filtrados y limitados a que sean del mismo día de la ejecución.
- Si se escoge la simulación, se pueden controlar algunos aspectos más: elegir una cierta cantidad de días anteriores con los que contrastar los datos de un día que se especifique, establecer que solamente se requiere de aquellos que sean del mismo día de la semana o escoger toda la base de datos. El primero y el tercero son mutuamente excluyentes, es decir, solamente se puede escoger uno de ellos.

De esta manera, se ha decidido establecer una serie de ejecuciones de las cuales se extraerán las pertinentes conclusiones previo análisis de los resultados. Dichos experimentos son:

1. Ejecución en tiempo real durante 4 horas con el mes de mayo precargado de la base de datos. Los parámetros para dicho experimento son los mostrados en el Código 27. Cada hora de esta ejecución será un batch para el algoritmo.

```
ficheroBBDD = statesFinal60segundos.csv
kafka = true
completeHistory = false
sameDay = true
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 60
```

Código 27.- Parámetros experimento 1.

Para este experimento, se tienen en la base de datos una cantidad de instancias de 144.000 instancias aproximadamente, las cuales se dividirán en bloques de 6000 instancias, por norma general. De esta manera, se tendrán esas 6000 instancias clasificadas como anterior frente al número de medidas que se vayan recibiendo de la plataforma. Dicho valor debería de estar en torno a 1200 ejemplos.

2. Simulación con los datos del día 31 de mayo con toda la base de datos agrupada en bloques de 60 segundos. Al igual que el experimento anterior, cada hora de las 24 horas que se analizarán será un bloque de datos para el algoritmo.

```
ficheroBBDD = statesFinal60segundos.csv
kafka = false
completeHistory = true
sameDay = false
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 60
```

Código 28.- Parámetros experimento 2.

En este primer experimento sintético, se tienen en total 11.376.000 instancias. De las cuales, y por cada bloque, habrá alrededor de 474.000 instancias clasificadas como anteriores frente a 1200 como actuales.

3. Simulación con los datos del día 31 de mayo con toda la base de datos agrupada en bloques de 600 segundos. También se tendrá 24 bloques de datos, una por cada hora.

```
ficheroBBDD = statesFinal600segundos.csv
kafka = false
completeHistory = true
sameDay = false
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 600
```

Código 29.- Parámetros experimento 3.

En este caso, se tendrá en la base de datos 1.137.600 instancias almacenadas. Por cada trozo de datos se enfrentarán alrededor de 47.000 instancias clasificadas como anteriores con 120 ejemplos clasificados como actuales.

4. Simulación con los datos del día 31 de mayo con la base de datos extraída durante ese mismo mes, agrupada en lotes de 60 segundos. Cada hora será un batch a analizar por el algoritmo.

```
ficheroBBDD = statesFinal60segundos.csv
kafka = false
completeHistory = false
sameDay = false
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 60
```

Código 30.- Parámetros experimento 4.

En este experimento, se dispone en la base de datos 864.000 filas, de las cuales se usarán alrededor de 36.000 medidas clasificadas como anteriores y 1200 como actuales.

5. Simulación con los datos del día 31 de mayo con la base de datos extraída durante ese mismo mes, agrupada en lotes de 600 segundos. Se dispondrá, como hasta ahora, de 24 lotes de datos, uno por cada hora del día.

```
ficheroBBDD = statesFinal600segundos.csv
kafka = false
completeHistory = false
sameDay = false
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 600
```

Código 31.- Parámetros experimento 5.

En este experimento similar al anterior, se tienen 86.400 medidas en total. Las cuales se dividirán en bloques que

albergan 3.600 medidas clasificadas como anteriores y 120 medidas clasificadas como actuales.

6. Simulación con los datos del día 31 de mayo con la base de datos extraída durante ese mismo mes, pero solamente los mismos días de la semana y con los datos agrupados en bloques de 60 segundos. Cada hora se convertirá en un bloque de datos.

```
ficheroBBDD = statesFinal60segundos.csv
kafka = false
completeHistory = false
sameDay = true
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 60
```

Código 32.- Parámetros experimento 6.

Continuando con las bases de datos usadas para los experimentos, se tiene en este caso un total de 144.000 instancias. Para cada bloque se tendrá alrededor de 6.000 clasificadas como anteriores y 1.200 clasificadas como actuales.

7. Simulación con los datos del día 31 de mayo con la base de datos extraída durante ese mismo mes, pero solamente los mismos días de la semana y con los datos agrupados en bloques de 600 segundos. Cada hora será un batch para el algoritmo.

```
ficheroBBDD = statesFinal600segundos.csv
kafka = false
completeHistory = false
sameDay = true
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 30
timeWindow = 3600
sensorRefreshTime = 600
```

Código 33.- Parámetros experimento 7.

En este caso, se tiene por norma general 14.400 instancias totales. Para cada bloque, se tienen 600 clasificadas como anteriores y 120 como actuales

8. Simulación con los datos del día 31 de mayo con toda la base datos, pero rellena con valores sintéticos para completar 5 años de uso ininterrumpido y sin agrupar en lotes de tiempo. Se busca con este experimento el determinar la escalabilidad del sistema. Para este experimento, también se dispondrá de 24 bloques de datos.

```
ficheroBBDD = statesGordo.csv
kafka = false
completeHistory = true
sameDay = false
startYear = 2021
startMonth = 5
startDay = 31
startHour = 0
forwardDays = 1
backwardDays = 2500
timeWindow = 3600
sensorRefreshTime = 15
```

Código 34.- Parámetros experimento 8.

En este último experimento, la base de datos alberga unas 53.424.000 instancias. De ellas, 2.226.000 instancias serán clasificadas como anteriores y se enfrentarán con unas 4.800 actuales por lote.

Experimento	Totales	Anteriores	Actuales
1	144.000	6.000	1.200
2	11.376.000	474.000	1.200
3	1.137.600	47.000	120
4	864.000	36.000	1.200
5	86.400	3.600	120
6	144.000	6.000	1.200
7	14.400	600	120
8	53.424.000	2.226.000	4.800

Tabla 14.- Resumen instancias actuales frente anteriores.

5.3. Resultados de los experimentos.

En esta sección se van a presentar los resultados obtenidos por el algoritmo para cada uno de los experimentos propuestos. Se ha escogido una representación en forma de tabla donde se dispone de 10 columnas. Cada una especifica lo siguiente:

- **Timestamp**: representa el bloque de datos o batch que se procesa.
- **NumRules**: número de reglas o patrones extraídos.
- **NumVars**: número de variables que forman parte del antecedente de un patrón.
- **CONF**: establece la confianza o precisión con respecto al contexto local de los patrones. Establece la relación entre el total de ejemplos cubiertos por la regla y el número de aciertos que ha tenido.

- **FPR**: tasa de falsos positivos. Determina la confianza con respecto al contexto global. Básicamente establece la relación entre cuantos ejemplos mal cubiertos (en total) han sido cubiertos por la regla en cuestión.
- **GR_PCT**: es el porcentaje de patrones extraídos. Calcula la relación entre los ejemplos cubiertos correctamente actuales entre los correctamente cubiertos en un instante anterior.
- **TPR**: tasa de verdaderos positivos. Indica la generalidad de las reglas calculando el porcentaje de instancias que han sido cubiertas correctamente con respecto a la totalidad de instancias pertenecientes a la clase de la regla.
- **WRAcc_Norm**: permitirá aproximar el interés de una regla. Pondera el TPR y la confianza de una manera normalizada para poder compararse. Se busca que una regla o patrón cubra muchos ejemplos y que se equivoque poco o muy poco.
- **Exec_time_ms**: es el tiempo de cómputo que ha tardado en procesar dicho batch, expresado en milisegundos.
- **Memory_mb**: es la cantidad de memoria que ha sido necesaria para procesar dicho batch, expresado en megabytes.

En concreto, se busca que estas medidas se maximicen o minimicen:

A maximizar	A minimizar
WRAcc_Norm	N.º Patrones
TPR	N.º Variables
GR_PCT	FPR
Confianza	Tiempo de ejecución
-	Memoria necesaria

Tabla 15.- Medidas a maximizar o minimizar.

Además, también se buscará el número máximo y mínimo de instancias procesadas y la hora en las que se registraron en las simulaciones ya que en flujo de datos es más complejo de obtener. Esto ayudará en el estudio de escalabilidad para saber cuántas instancias es capaz de procesar el algoritmo.

5.3.1. **Primer experimento.**

Como ya se ha comentado, en este primer experimento se usará la base de datos de mayo limitada al mismo día de la semana con la que se contrastarán las medidas de los sensores durante 4 horas ininterrumpidas. Se obtienen los resultados reflejados en la Tabla 16.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	4	0,226	0,815	1	0,957	0,571	268	745,869
2	1	6	0,246	0,704	1	0,92	0,607	246	588,266
3	1	6	0,243	0,701	1	0,901	0,600	170	471,255
4	1	6	0,246	0,694	1	0,90	0,603	132	768,070
MEDIA	1 ± 0	5,5 ± 1	0,24 ± 0,01	0,73 ± 0,06	1 ± 0	0,92 ± 0,03	0,60 ± 0,02	204 ± 63,77	643,37 ± 139,9

Tabla 16.- Resultados primer experimento. 4 Horas.

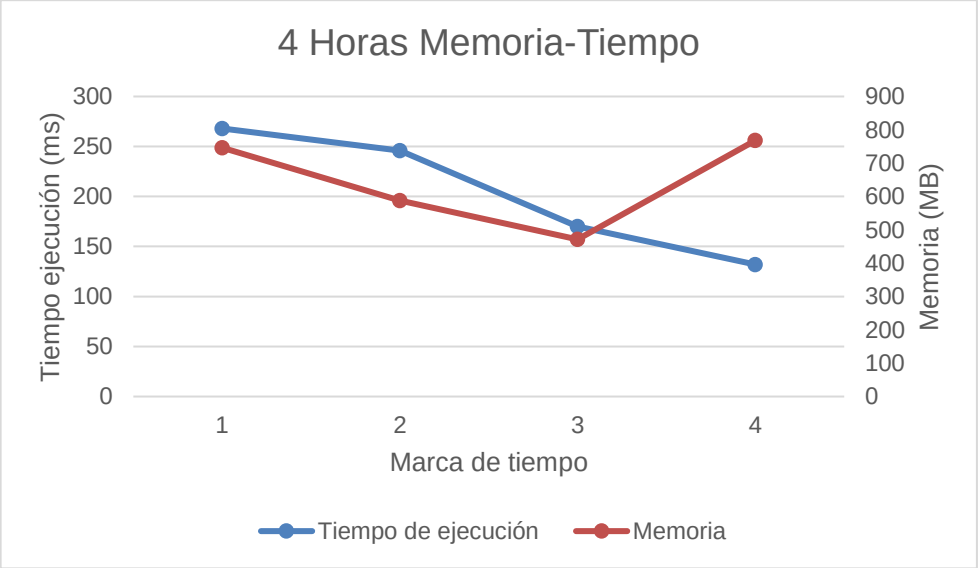

Figura 42.- Relación memoria/tiempo del primer experimento. Fuente: Elaboración propia.

5.3.2. **Segundo experimento.**

Tanto en este experimento como en los siguientes, se simulará una conexión con Apache Kafka mediante la precarga de una base de datos y el "envío" de estos datos como si vinieran de la plataforma de manera real.

Los datos reflejados en la Tabla 17 son los resultados obtenidos por el algoritmo al ejecutarlo con toda la base de datos agrupada en bloques de 60 segundos.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	6	0,003	0,784	1	0,949	0,582	1378	5020,918
2	1	8	0,003	0,748	1	0,915	0,583	1536	5659,249
3	1	8	0,003	0,726	1	0,898	0,586	1249	6299,656
4	1	8	0,003	0,720	1	0,900	0,589	1252	6677,502
5	1	12	0,004	0,532	1	0,851	0,659	1790	6483,745
6	1	12	0,003	0,579	1	0,850	0,635	1800	6243,723
7	1	12	0,004	0,537	1	0,896	0,679	1819	5959,025
8	1	12	0,004	0,541	1	0,892	0,675	1836	5706,885
9	1	12	0,004	0,534	1	0,915	0,690	1793	5003,423
10	1	12	0,004	0,513	1	0,902	0,694	1837	4652,997
11	1	12	0,004	0,555	1	0,900	0,672	1858	4540,298
12	1	13	0,004	0,543	1	0,851	0,654	1974	5614,497
13	1	13	0,004	0,545	1	0,916	0,685	2062	6978,155
14	1	13	0,004	0,550	1	0,925	0,68	1972	3640,026
15	1	13	0,004	0,558	1	0,921	0,681	2001	4851,508
16	1	13	0,004	0,571	1	0,9	0,66	1911	5630,531
17	1	13	0,003	0,584	1	0,898	0,657	1929	6286,708
18	1	13	0,003	0,584	1	0,879	0,647	1936	7201,068
19	1	13	0,003	0,542	1	0,836	0,646	2528	2893,469
20	1	13	0,003	0,546	1	0,844	0,649	1868	3407,595
21	1	13	0,003	0,544	1	0,828	0,641	3004	3913,594
22	1	13	0,003	0,552	1	0,814	0,630	1937	4302,780
23	1	13	0,003	0,549	1	0,801	0,625	1901	4984,247
MEDIA	1 ± 0	11,74 ± 2,07	0,0038 ± 0,0003	0,58 ± 0,08	1 ± 0	0,88 ± 0,04	0,65 ± 0,04	1877 ± 366.27	5302,24 ± 1168.46

Tabla 17.-Resultados segundo experimento. Toda historia (60 segundos).

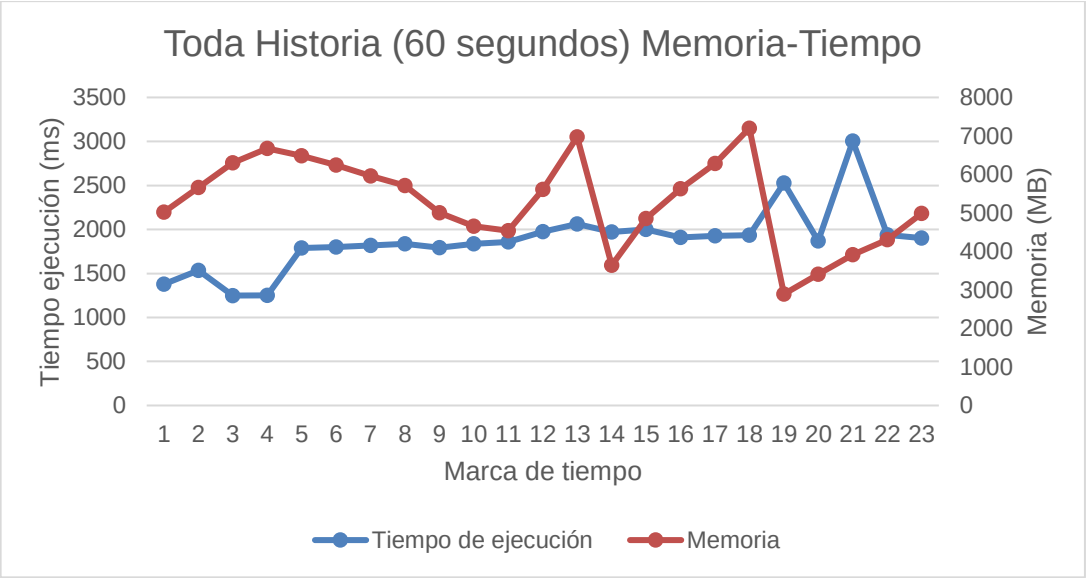

Figura 43.- Relación memoria/tiempo del segundo experimento. Fuente: Elaboración propia.

5.3.3. **Tercer experimento.**

En este experimento se utilizan al igual que en el anterior, los datos completos, pero agrupados en bloques de 10 minutos. Los resultados de esta ejecución se pueden apreciar en la Tabla 18.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	6	0,0003	0,788	1	0,95	0,580	3216	4671,908
2	1	6	0,0002	0,813	1	0,963	0,575	1269	2565,492
3	1	8	0,0003	0,695	1	0,975	0,640	1320	2375,839
4	1	7	0,0004	0,695	1	1	0,652	1162	5719,727
5	1	9	0,0004	0,593	1	0,950	0,678	1381	6375,750
6	1	11	0,0004	0,547	1	0,907	0,680	1633	4249,398
7	1	11	0,0004	0,522	1	0,948	0,713	1615	6669,117
8	1	11	0,0005	0,501	1	0,932	0,715	1646	4368,018
9	1	12	0,0006	0,443	1	0,941	0,748	1712	2480,637
10	1	12	0,0005	0,457	1	0,900	0,721	1726	5497,036
11	1	12	0,0005	0,526	1	0,966	0,720	1761	4027,455
12	1	12	0,0004	0,534	1	0,940	0,703	1861	2746,496
13	1	12	0,0005	0,535	1	0,966	0,715	2268	6075,325
14	1	12	0,0005	0,542	1	0,966	0,712	1724	5282,645
15	1	12	0,0004	0,549	1	0,967	0,709	1710	4229,614
16	1	12	0,0004	0,557	1	0,932	0,687	1704	2898,223
17	1	12	0,0004	0,558	1	0,950	0,695	1708	6261,745
18	1	12	0,0004	0,552	1	0,932	0,689	1723	4799,240
19	1	12	0,0004	0,533	1	0,903	0,684	1704	3018,933
20	1	12	0,0004	0,538	1	0,862	0,661	1712	6147,616
21	1	12	0,0004	0,529	1	0,852	0,661	1699	4005,765
22	1	12	0,0003	0,544	1	0,796	0,625	1956	6593,842
23	1	13	0,0004	0,499	1	0,8	0,650	1830	4998,521
MEDIA	1 ± 0	10,87 ± 2,10	0,0004 ± 0	0,57 ± 0,09	1 ± 0	0,93 ± 0,05	0,68 ± 0,04	1740.87 ± 395.01	4611.23 ± 1428.16

Tabla 18.- Resultados tercer experimento. 31 de mayo con los datos de toda la historia (600 segundos).

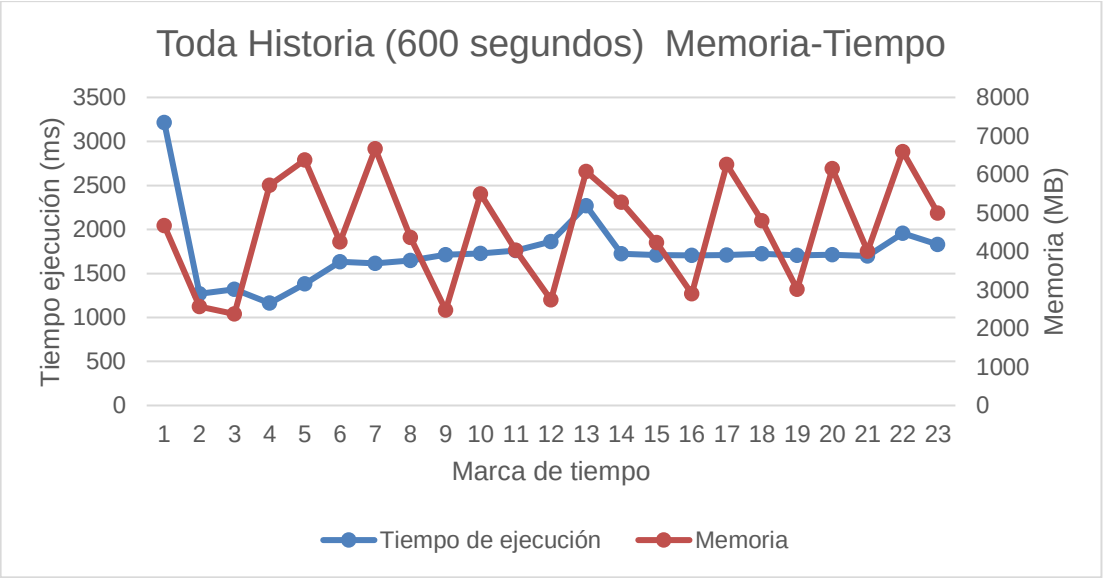

Figura 44.- Relación memoria/tiempo del tercer experimento. Fuente: Elaboración propia.

5.3.4. **Cuarto experimento.**

En este experimento se usarán solamente los datos de mayo agrupados en lotes de 1 minuto. Los resultados se aprecian en la Tabla 19.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	4	0,035	0,894	1	1	0,552	331	1752,828
2	1	5	0,036	0,844	1	0,965	0,560	249	3111,346
3	1	5	0,036	0,835	1	0,949	0,556	224	1378,038
4	1	5	0,036	0,832	1	0,950	0,559	143	2785,508
5	1	4	0,036	0,877	1	1	0,561	182	622,1327
6	1	4	0,036	0,872	1	1	0,563	140	1940,00
7	1	4	0,036	0,862	1	0,994	0,566	144	3300,387
8	1	4	0,036	0,859	1	0,965	0,552	135	1247,626
9	1	7	0,038	0,753	1	0,912	0,579	161	2735,881
10	1	6	0,038	0,782	1	0,944	0,580	161	959,613
11	1	6	0,038	0,783	1	0,941	0,579	161	2425,514
12	1	6	0,035	0,788	1	0,868	0,539	165	772,588
13	1	10	0,039	0,694	1	0,863	0,584	192	2555,462
14	1	8	0,037	0,801	1	0,949	0,574	176	1261,063
15	1	9	0,039	0,777	1	0,950	0,586	194	3021,008
16	1	9	0,039	0,777	1	0,95	0,586	188	1966,132
17	1	9	0,038	0,783	1	0,949	0,583	229	999,740
18	1	10	0,039	0,750	1	0,935	0,592	209	273,135
19	1	11	0,040	0,725	1	0,929	0,601	244	2203,839
20	1	11	0,040	0,708	1	0,905	0,598	196	1643,858
21	1	9	0,041	0,777	1	1	0,611	211	1085,451
22	1	9	0,041	0,776	1	1	0,611	190	559,468
23	1	9	0,040	0,785	1	1	0,607	181	2291,715
MEDIA	1 ± 0	7,13 ± 2,47	0,038 ± 0.002	0,79 ± 0,05	1 ± 0	0,95 ± 0,04	0,58 ± 0,02	191.57 ± 44.26	1777.93 ± 896.52

Tabla 19.- Resultados cuarto experimento. 31 de mayo con todos los datos de dicho mes (60 segundos).

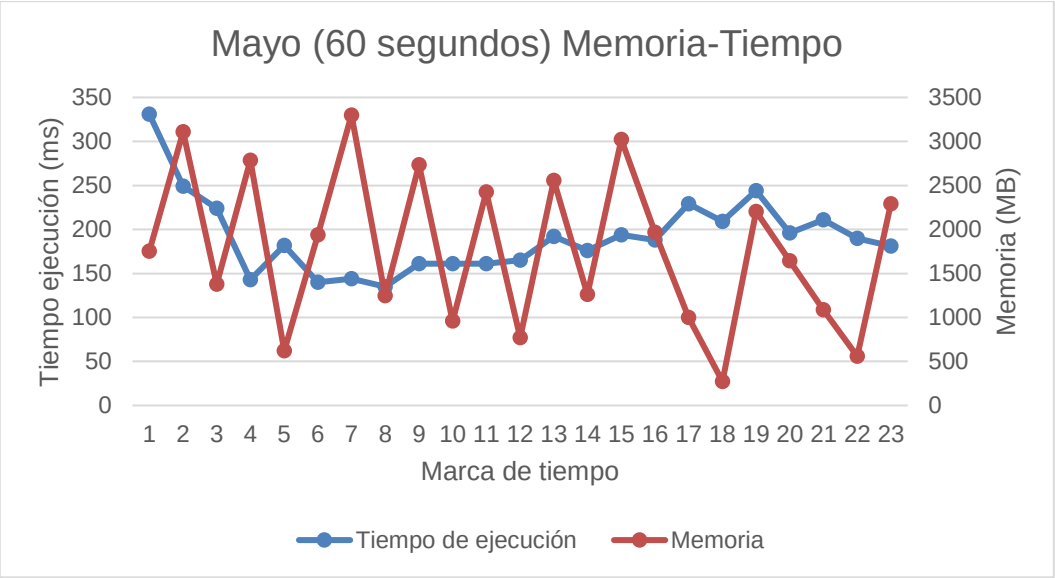

Figura 45.- Relación memoria/tiempo del cuarto experimento. Fuente: Elaboración propia.

5.3.5. **Quinto experimento.**

Es este experimento se incluyen las mismas restricciones que el anterior pero los datos están agrupados en bloques de 10 minutos. Los resultados aparecen expresados en la Tabla 20.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	4	0,035	0,882	1	1	0,558	310	464,812
2	1	6	0,033	0,853	1	0,963	0,555	342	535,095
3	1	6	0,035	0,846	1	0,950	0,552	129	632,732
4	1	7	0,038	0,807	1	0,951	0,571	92	160,925
5	1	7	0,037	0,804	1	0,950	0,573	89	307,896
6	1	7	0,037	0,810	1	0,949	0,569	88	476,084
7	1	7	0,036	0,798	1	0,939	0,570	92	170,835
8	1	7	0,037	0,789	1	0,915	0,563	88	399,346
9	1	6	0,037	0,832	1	0,975	0,571	95	196,967
10	1	6	0,035	0,844	1	0,933	0,544	92	68,012
11	1	7	0,038	0,791	1	0,966	0,587	90	373,141
12	1	6	0,034	0,820	1	0,898	0,538	87	329,603
13	1	6	0,036	0,840	1	0,95	0,554	105	338,892
14	1	6	0,038	0,820	1	1	0,589	98	105,721
15	1	7	0,041	0,772	1	1	0,613	94	237,368
16	1	7	0,040	0,782	1	1	0,608	101	158,806
17	1	9	0,043	0,731	1	1	0,632	110	334,126
18	1	9	0,042	0,733	1	1	0,633	98	199,032
19	1	9	0,044	0,734	1	1	0,632	114	324,987
20	1	9	0,040	0,738	1	0,965	0,613	114	106,232
21	1	9	0,041	0,740	1	0,950	0,605	106	249,489
22	1	8	0,040	0,773	1	1	0,613	110	169,753
23	1	8	0,041	0,766	1	1	0,616	94	324,866
MEDIA	1 ± 0	7,09 ± 1,31	0,038 ± 0.003	0,79 ± 0,04	1 ± 0	0,98 ± 0,03	0,59 ± 0,03	119.04 ± 66.33	289.77 ± 146.34

Tabla 20.- Resultados quinto experimento. 31 de mayo con los datos de dicho mes (600 segundos).

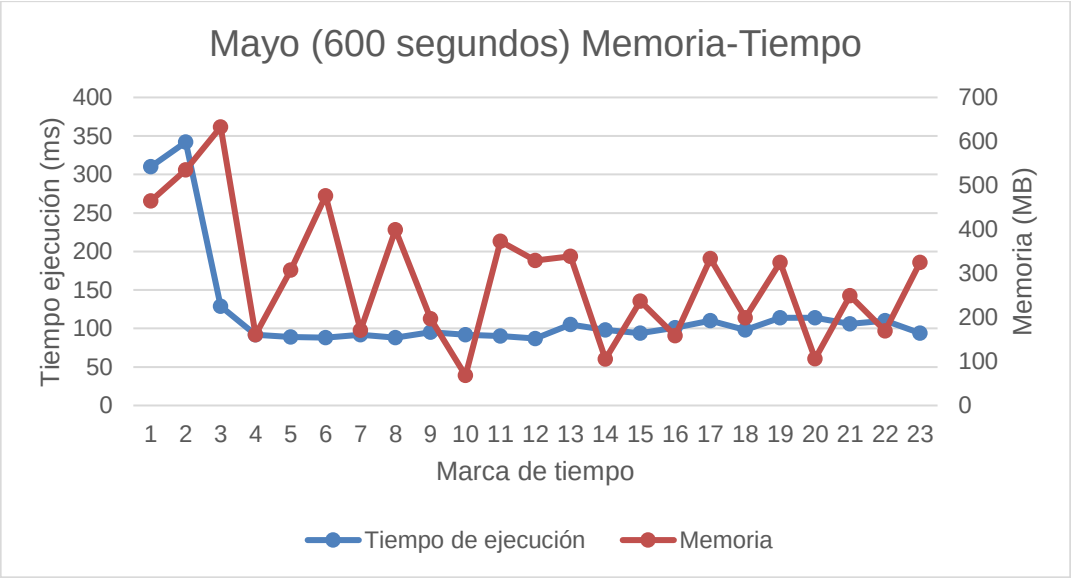

Figura 46.- Relación memoria/tiempo del quinto experimento. Fuente: Elaboración propia.

5.3.6. **Sexto experimento.**

Es este experimento se incluye la limitación de que tenga que ser el mismo día de la semana. En este caso, el 31 de mayo de 2021 fue lunes, por lo que se tomarán los datos de los sensores que tengan fecha de alguno de los 4 lunes de ese mes. Los datos reflejados en la Tabla 21 son los resultados.

Timestamp	Num Rules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	6	0,224	0,815	1	0,949	0,567	367	606,737
2	1	8	0,229	0,804	1	0,965	0,580	259	665,211
3	1	6	0,231	0,825	1	0,998	0,586	212	718,109
4	1	6	0,230	0,834	1	1	0,582	116	99,896
5	1	8	0,247	0,757	1	1	0,621	104	222,780
6	1	8	0,252	0,736	1	1	0,631	99	377,474
7	1	8	0,250	0,735	1	0,994	0,629	96	569,872
8	1	8	0,246	0,762	1	1	0,618	101	220,023
9	1	8	0,246	0,758	1	1	0,620	100	461,141
10	1	8	0,248	0,752	1	0,998	0,623	101	239,819
11	1	8	0,234	0,765	1	0,940	0,587	105	99,604
12	1	8	0,224	0,763	1	0,891	0,563	96	432,949
13	1	7	0,228	0,792	1	0,941	0,574	104	382,484
14	1	6	0,232	0,810	1	0,986	0,588	106	374,453
15	1	7	0,233	0,811	1	0,987	0,587	96	107,680
16	1	7	0,223	0,826	1	0,954	0,563	98	150,167
17	1	8	0,224	0,813	1	0,949	0,568	106	245,535
18	1	8	0,228	0,801	1	0,950	0,574	98	105,203
19	1	8	0,232	0,781	1	0,949	0,583	101	231,083
20	1	8	0,221	0,817	1	0,930	0,556	106	120,833
21	1	8	0,244	0,691	1	0,899	0,603	102	258,643
22	1	7	0,251	0,701	1	0,949	0,623	107	160,071
23	1	8	0,270	0,668	1	1	0,665	93	321,605
MEDIA	1 ± 0	7,48 ± 0,79	0,237 ± 0.013	0,77 ± 0,05	1 ± 0	0,97 ± 0,03	0,60 ± 0,03	124.91 ± 65.86	311.8 ± 189.38

Tabla 21.- Resultados sexto experimento. 31 de mayo con todos los datos, limitado a los lunes (60 segundos).

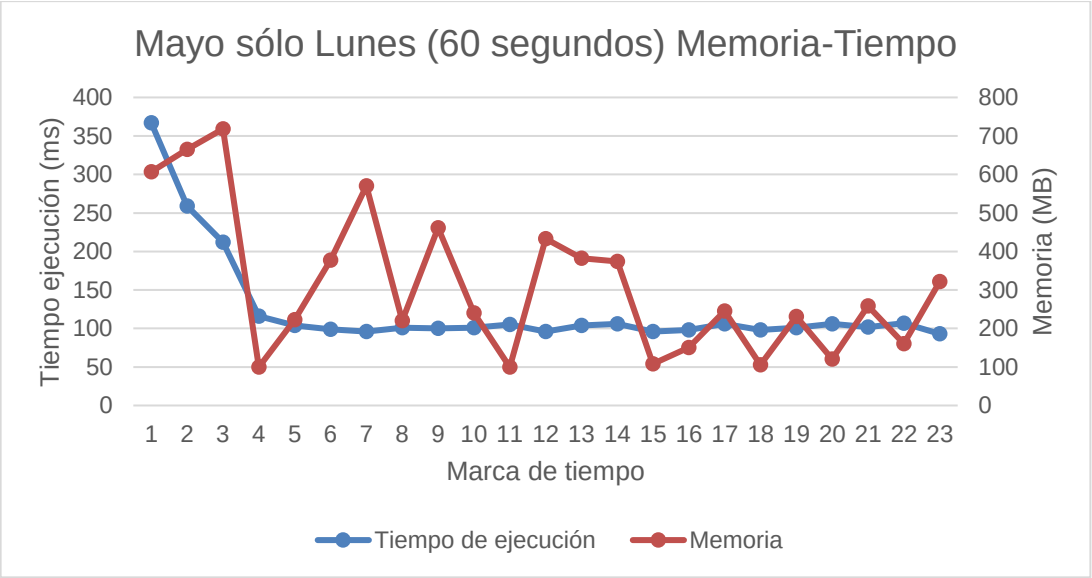

Figura 47.- Relación memoria/tiempo del sexto experimento. Fuente: Elaboración propia.

5.3.7. **Séptimo experimento.**

Al igual que en el antepenúltimo experimento, los datos con los que se contrastarán serán de los lunes únicamente. Con la diferencia de que esta vez se establecerán bloques de 10 minutos. Los resultados de este experimento son los presentados en la Tabla 22.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	1	5	0,222	0,805	1	1	0,597	300	218,953
2	1	7	0,217	0,725	1	0,963	0,619	192	861,355
3	1	6	0,234	0,759	1	1	0,620	130	686,061
4	1	6	0,239	0,756	1	1	0,621	95	545,955
5	1	7	0,241	0,732	1	1	0,633	84	411,507
6	1	7	0,244	0,704	1	1	0,647	91	301,603
7	1	8	0,240	0,700	1	1	0,649	98	238,466
8	1	9	0,242	0,642	1	0,899	0,628	83	195,153
9	1	8	0,246	0,705	1	1	0,647	84	167,078
10	1	8	0,251	0,695	1	1	0,652	88	166,476
11	1	9	0,243	0,703	1	0,983	0,639	90	190,378
12	1	10	0,227	0,689	1	0,898	0,604	85	239,399
13	1	8	0,231	0,690	1	0,9	0,604	84	304,479
14	1	6	0,239	0,734	1	1	0,632	79	399,063
15	2	5,5	0,219	0,781	1	0,926	0,572	91	508,368
16	1	9	0,236	0,664	1	0,899	0,617	85	185,502
17	1	9	0,236	0,678	1	0,900	0,611	90	357,905
18	1	7	0,230	0,724	1	0,957	0,616	82	132,259
19	1	9	0,248	0,643	1	0,887	0,621	83	353,247
20	1	8	0,251	0,625	1	0,948	0,661	78	218,990
21	1	8	0,260	0,635	1	0,950	0,657	85	138,493
22	1	9	0,256	0,622	1	0,949	0,663	86	113,462
23	1	9	0,257	0,632	1	0,95	0,658	88	131,690
MEDIA	1.04 ± 0.20	7,72 ± 1,36	0,24 ± 0.012	0,69 ± 0,05	1 ± 0	0,957 ± 0,043	0,63 ± 0,023	102.21 ± 49.26	307.21 ± 191.27

Tabla 22.- Resultados séptimo experimento. 31 de mayo con todos los datos, limitado a los lunes (600 segundos).

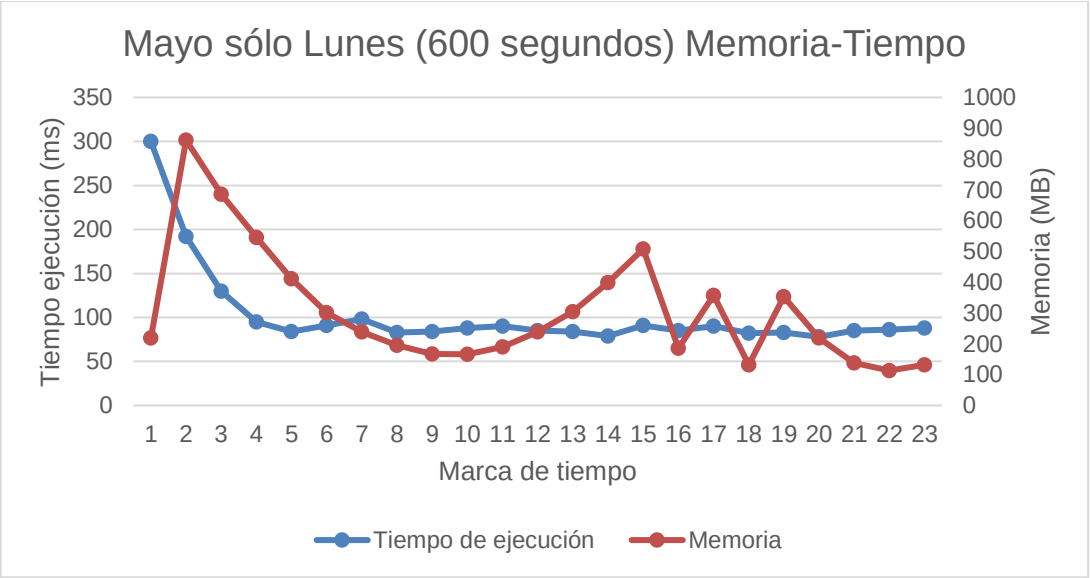

Figura 48.- Relación memoria/tiempo del séptimo experimento. Fuente: Elaboración propia.

5.3.8. **Octavo experimento.**

En este experimento final se buscará llevar el algoritmo y el sistema de extracción de conocimiento al límite. Para ello y como ya se indicó en la sección 5.2 se ha rellenado la base de datos con valores artificiales para los últimos 5 años, eso quiere decir que se dispone de una cantidad de ejemplos muy grande y se buscará descubrir cómo se comporta y desenvuelve ante unos bloques de datos de mayor tamaño.

Los resultados después de la larga ejecución son los presentados en la Tabla 23.

Timestamp	NumRules	NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm	Exec_time_ms	Memory_mb
1	49	5,653	0,0001	0,912	0,020	0,478	0,283	8641	12339,254
2	49	6,265	0,0005	0,884	0,020	0,725	0,420	22555	9305,735
3	49	7,693	0,0005	0,852	0,020	0,636	0,391	26537	11277,610
4	49	7,571	0,0005	0,840	0,020	0,653	0,406	14655	11604,556
5	49	7,510	0,0004	0,856	0,020	0,563	0,353	47349	11913,203
6	49	7,632	0,0003	0,850	0,020	0,532	0,341	34762	10480,563
7	49	6,857	0,0007	0,889	0,020	0,714	0,412	35053	10265,868
8	49	6,795	0,0009	0,902	0,020	0,732	0,415	12096	12429,613
9	49	7,020	0,0006	0,888	0,020	0,682	0,396	34609	10609,311
10	49	7,061	0,0008	0,881	0,020	0,755	0,436	49958	11357,642
11	49	7,163	0,0006	0,862	0,020	0,646	0,391	90288	11653,488
12	49	7,693	0,0008	0,844	0,020	0,652	0,403	60448	10260,673
13	49	7,612	0,0007	0,837	0,020	0,660	0,411	9110	12306,022
14	49	7,020	0,0007	0,853	0,020	0,639	0,392	77763	11633,417
15	49	7,612	0,0006	0,830	0,020	0,619	0,394	9319	10362,078
16	49	7,673	0,0006	0,832	0,020	0,617	0,392	6826	12224,855
17	49	7,795	0,0006	0,831	0,020	0,614	0,391	46197	9978,446
18	49	7,714	0,0006	0,846	0,020	0,637	0,395	11223	12434,050
19	49	7,714	0,0006	0,846	0,020	0,630	0,391	67654	12312,858
20	49	7,530	0,0006	0,864	0,020	0,675	0,405	42728	10836,795
21	49	7,979	0,0007	0,853	0,020	0,718	0,432	46257	10146,764
22	49	7,448	0,0007	0,870	0,020	0,737	0,433	8168	10183,578
23	49	6,816	0,0006	0,888	0,020	0,695	0,403	206469	9953,036
MEDIA	49 ± 0	7.3 ± 0,55	0.0006 ± 0	0,86 ± 0,02	0.02 ± 0	0,65 ± 0,07	0,4 ± 0,03	42115,87 ± 43060,89	11124.76 ± 978,04

Tabla 23.- Resultados octavo experimento. 31 de mayo con todos los datos de 5 años.

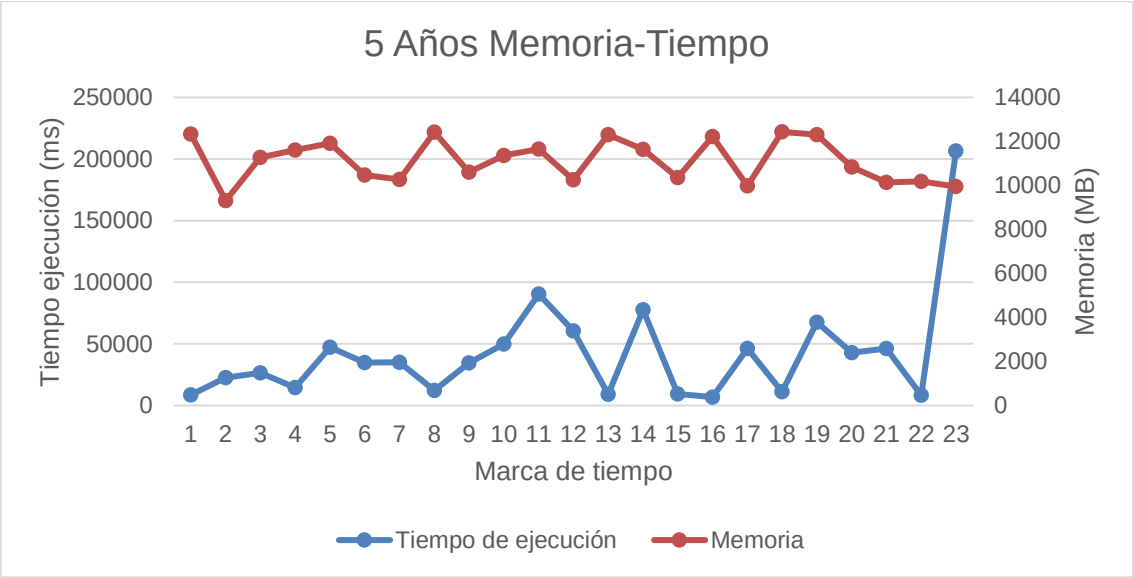

Figura 49.- Relación memoria/tiempo del octavo experimento. Fuente: Elaboración propia.

5.4. Análisis de los resultados.

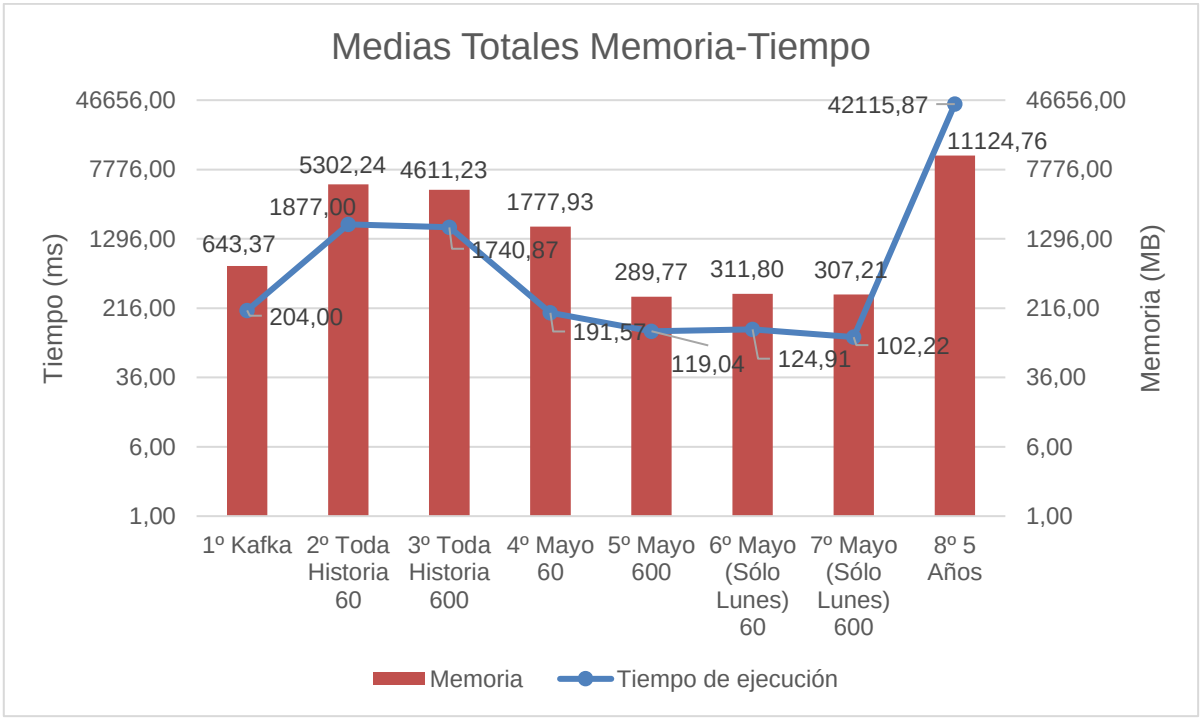

Figura 50.- Medias totales (memoria/tiempo). Fuente: Elaboración propia.

5.4.1. Estudio de calidad.

Para comenzar el análisis de los resultados se ha optado por comenzar con el estudio de la calidad de las reglas obtenidas. Para ello, se analizará: la complejidad, el interés, la generalidad y la fiabilidad, tomando como

referencia los resultados medios de las distintas ejecuciones. Dichos resultados se muestran en la Tabla 24.

Experimento	CONF	GR_PCT	TPR
1	0,24 ± 0,01	1 ± 0	0,92 ± 0,03
2	0,00 ± 0	1 ± 0	0,88 ± 0,04
3	0,00 ± 0	1 ± 0	0,93 ± 0,05
4	0,04 ± 0	1 ± 0	0,95 ± 0,04
5	0,04 ± 0	1 ± 0	0,97 ± 0,03
6	0,24 ± 0,01	1 ± 0	0,97 ± 0,03
7	0,24 ± 0,01	1 ± 0	0,96 ± 0,04
8	0,00 ± 0	0,02 ± 0	0,65 ± 0,07

Tabla 24.- Resultados medios de todos los experimentos.

En el primer caso, para la medida de la complejidad hay que fijarse en el número de patrones extraídos y el número de variables usadas. De esta manera, se puede observar fácilmente que el número de patrones o reglas es casi siempre 1, esto se debe a la peculiaridad del algoritmo y su construcción de manera que no generen demasiados patrones o reglas. Con respecto al número de variables, se aprecia que todas las ejecuciones emplean en torno a 7 variables, bajando en algunos casos a 4 o subiendo a 12 variables. Dicho esto, se puede concluir que el sistema no genera una complejidad muy alta y que el modelo generado debería de ser considerado interpretable por el experto.

Para proseguir, se estudiará el interés. Para ello, hay que centrarse en las medidas de WRAcc_norm, que como ya se explicó en la sección 5.3, pondera el TPR y la confianza. Cabe destacar que, para todos los experimentos, excluyendo aquel que incorpora la base de datos generada durante 5 años, se obtiene una medida mayor al 0.60. Este valor puede ser

más que aceptable en cuanto al interés que demuestra el modelo y que la decaída en dicho octavo experimento se deba a la inclusión masiva de datos sintéticos que no sean tan fieles como los reales. No obstante, ese último experimento fue formulado para determinar más bien la escalabilidad del modelo que la calidad del mismo, como ya se mencionó anteriormente.

En tercer lugar y en cuanto a la generalidad, se puede observar mediante los valores de las columnas TPR que los valores están entorno al 0.90. Esto es muy aceptable ya que se obtienen resultados del 90% e incluso con algunas reglas del 100% obteniendo en esos casos reglas más generales que cubren un mayor número de ejemplos. Es decir, se cubren una mayor cantidad de ejemplos del instante actual con respecto al anterior.

Por último, se estudiarán las columnas de FPR y la confianza para determinar la fiabilidad del sistema. Como se aprecia en la Tabla 15, se debe buscar minimizar el valor FPR frente a la maximización de la confianza. Se destaca que, para todos los experimentos el valor de los falsos positivos y por lo tanto la confianza con respecto al contexto global de los patrones no es muy buena. En todos los casos se obtienen valores superiores a 0.5. Frente a esto, se obtienen unos valores de confianza respecto al contexto local que rondan, en los mejores casos, el 0.25. Esto es debido al desbalanceo en la base de datos, es decir, según se han ido realizando los experimentos las cantidades de instancias anteriores y actuales de cada bloque de datos se ha ido equilibrando. Es por ello que se consiguen mejores valores de confianza en estos últimos experimentos y en el experimento que implica a Apache Kafka. Dicha variación decreciente entre la diferencia de las cantidades de instancias se puede apreciar también en la Tabla 14.

Por otra parte, y viendo los resultados de las desviaciones típicas se aprecia que dichos resultados, salvo el tiempo de ejecución, la memoria y el número de variables empleadas, son inferiores a 1. Esto quiere decir que los valores medios son muy robustos y varían muy poco, ya que se desvían

en apenas una unidad de los mismos. Con respecto a la variabilidad del número de variables empleadas, puede haber una diferencia de entre 1 y 3 variables, por lo que también se obtienen unos resultados robustos y que no se alejan mucho del valor medio. Solamente se encuentra una mayor desviación en los tiempos de ejecución y en la memoria utilizada. En el primer caso, es normal que se desvíen ya que influyen factores externos como la sobrecarga del ordenador y del sistema en el cual se está ejecutando el algoritmo. Con respecto a la memoria, también es normal, dado que los ejemplos que se tienen en cada uno de los batches o lotes de datos varía en el tiempo, esto quiere decir que un trozo de datos puede albergar más instancias que otro dependiendo de la variabilidad que haya en las medidas de los sensores. En cualquier caso, las desviaciones de los tiempos son de apenas medio segundo. Por otro lado, las desviaciones de la memoria van desde los 200 megabytes hasta el gigabyte y medio. Esto si podría suponer un problema al ejecutarse en un sistema de capacidades limitadas. Con respecto al último experimento, se comentarán con más profundidad los resultados obtenidos en la siguiente sección ya que como se comentó al comienzo de la sección 5.8, dicho experimento está más orientado a la escalabilidad.

Juntando lo anterior, se obtiene como resultado del estudio la conclusión de que el algoritmo tiene la capacidad de extraer conocimiento interpretable y de interés. Todo esto bajo unos tiempos de ejecución bastante aceptables, aunque esto se explicará y se profundizará mejor en la siguiente sección. Además, se obtienen mejores resultados en los experimentos que involucran únicamente a datos verídicos, y estos se mejoran aún más si se utilizan solamente los del mismo día de la semana en vez de todo el mes. Con esto último, se proporciona una ligera idea de cuál es la mejor estrategia: buscar en una ventana de tiempo más reciente y/o común, como puedan ser todos los lunes.

5.4.2. **Reglas obtenidas.**

Para continuar el análisis se mostrará una de las reglas obtenidas por el algoritmo para el Primer experimento. Estas reglas, como ya se comentó, son del tipo SI-ENTONCES, por lo que se tendrán una serie de antecedentes que determinan la condición y un antecedente que representará la clase, en este caso será actual. Esta regla se puede apreciar en el Código 35.

```
Rule: 0(ID: 1730779384)
    Variable sensor.luminosidad_cocina =
        Label 1 (Triangular : 2.34 , 24.9 , 47.46)
        Label 2 (Triangular : 24.9 , 47.46 , 70.02)
    Variable sensor.presion_habitacion = Label 0 (Triangular : 953.60 , 956.7 , 959.8)
    Variable sensor.luminosidad_habitacion_2 = Label 1 (Triangular : 1.37 , 8.3 , 15.23)
    Variable sensor.temperatura_habitacion_2 = Label 2 (Triangular : 24.45 , 30.7 , 36.95)
    Variable sensor.presion_salon = Label 0 (Triangular : 952.75 , 955.9 , 959.05)
    Variable sensor.temperatura_salon = Label 2 (Triangular : 25.75 , 29.8 , 33.85)
Consequent: actual
```

Código 35.- Ejemplo de regla obtenida por el algoritmo.

Con esta regla se pueden ver 6 antecedentes relacionados con las medidas de los sensores. Los resultados de Label 0, 1 y 2 se refieren a los 3 rangos que se generan por defecto, siendo: el 0 un nivel bajo, el 1 un nivel medio y el 2 un nivel alto. Por lo tanto, se establece que el nivel medio de la luminosidad de la cocina se da cuando el sensor mide entre 2.34 lx y 47.46 lx, con una media de 24.9 lx. Para el resto de variables el proceso es análogo.

Dicho esto, lo que esta regla quiere decir es: SI el nivel de luminosidad de la cocina está en un valor medio o alto, la presión de la habitación 1 se encuentra en un nivel bajo, la luminosidad de la habitación 2 está en un

nivel medio, la temperatura de la misma habitación se encuentra en un nivel alto, la presión del salón da una medida de valor bajo y la temperatura del salón se encuentra en un nivel alto ENTONCES la situación se corresponde con el instante actual. Esto quiere decir que, la situación en concreto se está dando en el instante actual, entre las 13:00-14:00 concretamente, y que no se daba en el instante anterior. Estas reglas se obtienen de dicha forma debido a que usa una representación en DNF y se puede apreciar de una manera más legible mediante la Tabla 25. De esta manera, entre las diferentes columnas se encontrarán conjunciones y dentro de cada celda se aplicarán disyunciones.

Regla obtenida					
Lum. cocina	Pres. hab. 1	Lum. hab. 2	Temp. hab. 2	Pres. salón	Temp. salón
Medio o alto	Bajo	Medio	Alto	Bajo	Alto

Tabla 25.- Condiciones de la regla obtenida para el instante actual.

Con respecto a los resultados que ofrece está regla, se mostrarán en la Tabla 26. En dicha tabla, se puede apreciar que los valores no distan de los valores medios obtenidos y que debería de considerarse una regla interpretable y de interés, ya que el valor de la columna WRAcc_Norm está por encima del 0.6. Además, también se obtiene una regla bastante general, esto es debido a que se cubren el 90% de ejemplos del instante actual con respecto al anterior. No obstante, se obtienen unos valores algo altos para la columna del FPR y algo bajos para la confianza. Como ya se comentó, esto es reflejo de una base de datos desbalanceada con respecto a las instancias que se dispone para el instante anterior frente a las que se dispone para el instante actual. Por último, se destaca un índice de crecimiento de 1, esto quiere decir que la regla ha cubierto correctamente una mayor cantidad de ejemplos en el instante actual frente a los que cubrió correctamente de un instante anterior.

NumVars	CONF	FPR	GR_PCT	TPR	WRAcc_Norm
6	0,246	0,693	1	0,90	0,604

Tabla 26.- Resultados para la regla obtenida.

5.4.3. Estudio de escalabilidad.

En esta sección se realizará el estudio de la escalabilidad del sistema. Para ello, se presenta en la Tabla 27 las cantidades máximas y mínimas de instancias, que, junto con las medidas de tiempos y memorias se podrá observar cómo se comporta el sistema frente a diferentes cantidades de datos.

Experimento	Min_Num_Inst	Max_Num_Inst	Hora_Min	Hora_Max
2º Toda Historia 60	480.150	498.946	3	18
3º Toda Historia 600	44.228	44.422	2	17
4º Mayo 60	37.151	37.252	3	17
5º Mayo 600	3.691	3.754	2	17
6º Mayo (Sólo Lunes) 60	5.973	6.007	0	21
7º Mayo (Sólo Lunes) 600	588	611	2	23
8º 5 Años	2.242.919	2.262.201	4	16

Tabla 27.- Número mínimo y máximo de instancias y la hora a la que se alcanzaron de cada experimento.

Para comenzar, se aprecia a simple vista que las cantidades de instancias o ejemplos analizados va disminuyendo progresivamente con los distintos experimentos. Si se comparan estos valores con el tiempo y memoria que se debe de utilizar se ve algo que no resulta extraño: a más instancias más tiempo de cómputo y memoria es necesario para poder evaluarlos, esto se puede apreciar en la Figura 50 y en la Tabla 28. Otro aspecto que se comporta como cabría esperar es que a mayor cantidad de memoria requerida mayor es el tiempo que requiere el algoritmo para procesar el bloque de datos, esto se aprecia muy bien con las curvas generadas en la gráfica de la Figura 50.

Si se prosigue el estudio, se puede apreciar que los tiempos de ejecución no llegan a los dos segundos, aunque se esté tratando con 500.000 instancias de datos. Esto proporciona una ligera idea de cuán escalable resulta el sistema, ya que, a pesar de enfrentarse a una cantidad bastante grande de datos el tiempo que tarda no es para nada excesivo y es inferior a 4 segundos, umbral aproximado por el que algo superior se considera "lento" en este entorno de trabajo. Además, aunque la cantidad media de instancias sea unas 800 veces superior entre el primer experimento de la Tabla 28 y el penúltimo, el tiempo medio de ejecución no evoluciona de la misma manera y este solamente aumenta unas 18 veces.

Experimento	Instancias Medias	Tiempo de ejecución (ms)
2º Toda Historia 60	489.548	1.877
3º Toda Historia 600	44.325	1.740,87
4º Mayo 60	37.201,50	191,57
5º Mayo 600	3.722,50	119,04
6º Mayo (Sólo Lunes) 60	5.990	124,91
7º Mayo (Sólo Lunes) 600	599,50	102,22
8º 5 Años	2.252.560	42.115,87

Tabla 28.- Relación Tiempo-Instancias.

Con respecto al comportamiento del sistema, cuando se procesa una cantidad alrededor de 4.5 veces mayor al máximo anteriormente citado sí que se aprecia un aumento significativo en el tiempo de ejecución necesario, pasando de 1.8 segundos a 42 segundos. Esto supone un enfrentamiento entre aumentar 4.5 veces el número de instancias a aumentar 22.5 veces el tiempo de cómputo.

Es decir, se ha aumentado unas 3757 veces aproximadamente el número medio de instancias, pero el tiempo medio necesario ha aumentado solamente alrededor de 412 veces.

No obstante, y como explicación acerca de por qué aparecen los picos en la Figura 49 y por ende el porqué de esa desviación estándar tan grande en los resultados del Octavo Experimento principalmente. Es posible que se deba al uso de la CPU y RAM de la máquina host donde estaba corriendo el algoritmo por otros programas, haciendo necesario un tiempo de cómputo mayor, aunque el número de instancias sea similar. Es por ello, que deberían de considerarse como datos anómalos y no tenerse mucho en cuenta, tomando mejor como promedio de tiempo necesario alrededor de 25-30 segundos.

Con todo esto, se puede concluir que, al igual que el estudio anterior, se obtienen los mejores resultados en los casos más reales, demorándose menos de un segundo en procesar cada bloque de datos. Hay que tener en cuenta que, en los otros experimentos se analiza frente a toda la base de datos y todos los datos hasta la fecha. Con respecto a la memoria necesaria es donde se podría decir que hay más peligro, ya que como se ha comentado antes, el hecho de necesitar 5GB aproximadamente en la ejecución puede limitar la implantación del algoritmo en sistemas limitados. Por último, destacar que esta anterior idea es la que podría limitar algo más la escalabilidad del sistema, ya que, en vista de los resultados, se podría aumentar la cantidad de instancias bastante y el tiempo necesario seguiría en unos valores muy por debajo de lo que se podría considerar problemático. De hecho, en este caso se dispone del tiempo suficiente para procesar el flujo continuo de datos, ya que, incluso añadiendo algo más de 2 millones de instancias, tarda menos de 1 minuto cuando el tiempo de la ventana deslizante es de una hora.

Capítulo 6.- Conclusiones.

6.1. Conclusión.

Este proyecto es un trabajo pionero en el aprendizaje de patrones descriptivos para minería de cambio (change mining), donde se incorpora una instalación de un sistema domótico de extracción de conocimiento mediante una serie de medidas de sensores y la posterior aplicación de un EFS para la extracción de reglas descriptivas.

En cuanto a la primera parte, se ha conseguido instalar el sistema de una manera satisfactoria y con una capacidad de ampliación cómoda, ya que solamente habría que incorporar nuevos sensores a la plataforma y a sendos ficheros destinados a los sensores, el de cabecera y fichero de texto (ambos con los sensores dispuestos en el mismo orden). Con esto, se podrían incluir una mayor cantidad de variables para las reglas. Además, se ha realizado todo en un entorno local como se buscaba, siendo la opción más segura para protegerse.

Con respecto a la segunda parte, el algoritmo, cabe destacar que es capaz de gestionar los datos de 5 años en menos de un minuto. Este tiempo necesario, aunque sea alto en entornos de flujos de datos continuos, es aceptable en este caso ya que se dispone de ventanas de tiempo de una hora, con lo que no supone problema alguno. No obstante, y a la vista de los resultados, resulta mucho más interesante generar reglas y realizar el estudio sobre datos no tan distantes en el tiempo. Es decir, el hecho de buscar únicamente los mismos días de la semana y durante el mismo mes hace que se obtengan reglas con una confianza mayor que analizando el año completo. Esto último beneficia también los requisitos de memoria, que decaerían hasta apenas los 200 megabytes, siendo esta cifra mucho más factible en entornos de cómputo limitados como se dan en este caso.

Por lo tanto, la premisa inicial que se tenía de realizar comparativas del estado actual con respecto a todo lo que se tenía ha resultado ser inviable porque se compara con una cantidad muy grande de datos. La base de datos está muy desbalanceada y se han ido realizando pruebas en las que se van haciendo más concretos los datos anteriores para que se puedan asociar con el dato del momento actual.

6.2. Trabajo futuro.

En un futuro no muy lejano casi todos los dispositivos tendrán conectividad a Internet y pasarán a formar parte del Internet de las Cosas con los roles de productores y consumidores de datos (W. Shi et al., 2016). Es por ello que el trabajo realizado en este proyecto abre una serie de líneas de trabajo muy interesantes y demuestra la complejidad del problema y la necesidad de mejorar los algoritmos de cara a que puedan analizar lo que esté pasando actualmente con respecto a lo que pasó anteriormente. También, se desvía un poco la atención hacia la necesidad de que el instante anterior sea lo más cercano posible al instante actual.

Por ello, la primera línea de trabajo tiene relación con la cercanía de los instantes. Por ejemplo, una posible inclusión a los datos sería las condiciones climatológicas para que la base de datos no se centre únicamente en la recogida de datos de sensores, sino que se puedan incluir elementos referentes al clima. Además, la plataforma Home Assistant trae por defecto la integración de un instituto meteorológico noruego que permite almacenar gran cantidad de información metrológica, tales como: temperatura, velocidad del viento y previsión de precipitaciones y cielos nublados. No obstante, existen multitud de integraciones referentes al clima, de diferentes institutos y empresas, de entre las que elegir.

Otra línea de trabajo va relacionada con la fase final del proceso que se ha llevado a cabo. Es decir, se dispone de una recogida de datos y una

creación de reglas de forma normal disyuntiva, lo que se propone para profundizar es la conexión entre la salida del algoritmo de conocimiento y la plataforma de Home Assistant para presentar al usuario dichas reglas y, que, según su conveniencia, decida implantarlas o no. Esto debería de llevarse a cabo de una manera amigable y únicamente de aquellas reglas que supongan una verdadera mejora o denoten un gran interés para el usuario.

Otra línea de trabajo podría ser la modificación del algoritmo para la extracción de reglas en forma canónica en vez de forma normal disyuntiva. De esta manera, se podría llevar a cabo un estudio sobre qué forma de presentación es mejor o con cuál forma trabaja mejor el algoritmo.

Por último, y siguiendo la línea anterior, se propone una modificación del proceso de aprendizaje para genere más diversidad entre los individuos y no se converja hasta una única regla.

7. Bibliografía.

Á. M. García-Vico, C. J. (2020). FEPDS: A Proposal for the Extraction of Fuzzy Emerging Patterns in Data Streams. *IEEE Transactions on Fuzzy Systems*, 3193-3203.

Ali Pesaranghader, H. L. (2017). McDiarmid Drift Detection Methodsfor Evolving Data Streams. Obtenido de https://www.researchgate.net/figure/Real-Concept-Drift-vs-Virtual-Concept-Drift-Similar-to-Fig-1-in-3_fig1_320241820

Ángel M. García-Vico, C. C. (2021). A cellular-based evolutionary approach for the extraction of emerging patterns in massive data streams. *Expert Systems With Applications.*, 50.

Awesome-ha. (2021). *www.awesome-ha.com*. Obtenido de https://www.awesome-ha.com/

blogdedomotica. (30 de 10 de 2019). *blogdedomotica.com*. Obtenido de https://blogdedomotica.com/home-assistant-que-es-y-para-que-sirve/

casasdigitales. (28 de 12 de 2017). *www.casasdigitales.com*. Obtenido de https://www.casasdigitales.com/domotica-open-source-2/

Claire. (25 de 11 de 2020). *github.com*. Obtenido de https://github.com/daugaard/claire

Cole, R. (15 de 11 de 2018). *Hackster*. Obtenido de https://www.hackster.io/robin-cole/tensorflow-object-detection-with-home-assistant-7cc04b

Cole, R. (2020). *notebooks.gesis.org*. Obtenido de https://notebooks.gesis.org/binder/jupyter/user/robmarkcole-hass-data-detective-1t0dt6gd/notebooks/notebooks/Getting%20started%20with%20detective.ipynb

danrodgar.github.io. (2021). *https://danrodgar.github.io/DASE/figures/Fayyad96kdd-process.png*. Obtenido de https://danrodgar.github.io/DASE/figures/Fayyad96kdd-process.png

Dean, J. &. (2004). Mapreduce: Simplified data processing on large clusters. *Operating Systems Design and Implementation*, 137-150.

demo.home-assistant.io. (2021). *https://demo.home-assistant.io/#/lovelace/0*. Obtenido de https://demo.home-assistant.io/#/lovelace/0

domoticaencasa.es. (23 de 08 de 2018). *https://domoticaencasa.es/home-assistant-controla-aire-acondicionado-con-un-sonoff-basic/*. Obtenido de https://domoticaencasa.es/home-assistant-controla-aire-acondicionado-con-un-sonoff-basic/

domoticaencasa.es. (11 de 8 de 2019). *https://domoticaencasa.es/openhab-se-prepara-para-su-version-2-5-0/*. Obtenido de https://domoticaencasa.es/openhab-se-prepara-para-su-version-2-5-0/

Dong G. & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. *In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, (págs. 43-52).

enmilocalfunciona.io. (27 de 09 de 2018). *https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/*. Obtenido de https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/

es.weatherspark.com. (2021). *https://es.weatherspark.com/m/36637/1/Tiempo-promedio-en-enero-en-Ja%C3%A9n-Espa%C3%B1a*. Obtenido de https://es.weatherspark.com/m/36637/1/Tiempo-promedio-en-enero-en-Ja%C3%A9n-Espa%C3%B1a

Garcia-Vico, A. M. (2020). E2PAMEA: A fast evolutionary algorithm for extracting fuzzy emerging patterns in big data environments. *Neurocomputing., 415*, 60-73.

García-Vico, A. M.-B. (2018). An overview of emerging pattern mining in supervised descriptive rule discovery: taxonomy, empirical study, trends, and prospects. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery.*

Grafana. (2021). *grafana.com*. Obtenido de https://grafana.com/grafana/

gruporp. (17 de 1 de 2020). *https://gruporp.es/blog/que-es-smart-home-como-funciona-n11*. Obtenido de https://gruporp.es/blog/que-es-smart-home-como-funciona-n11

Hastic.io. (31 de 05 de 2018). *hastic.io*. Obtenido de https://hastic.io/blog/2018/05/31/public-release-of-hastic/

Heslop, B. (4 de 5 de 2019). *Martech Advisor*. Obtenido de https://www.martechadvisor.com/articles/iot/by-2030-each-person-will-own-15-connected-devices-heres-what-that-means-for-your-business-and-content/

Home Assistant. (27 de 12 de 2018). *data.home-assistant.io*. Obtenido de https://data.home-assistant.io/blog/2018/12/27/data-science-for-home-assistant-launched

Home Assistant. (27 de 12 de 2018). *www.home-assistant.io*. Obtenido de https://www.home-assistant.io/blog/2018/12/27/data-science-portal/

Home Assistant. (2021). *https://github.com/home-assistant/home-assistant.io*. Obtenido de https://github.com/home-assistant/home-assistant.io

Jupyterlab. (2018). *jupyterlab.readthedocs.io*. Obtenido de https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html

letscontrolit.com. (29 de 12 de 2017). *https://www.letscontrolit.com/forum/viewtopic.php?t=3934*. Obtenido de https://www.letscontrolit.com/forum/viewtopic.php?t=3934

Li, J. L. (2014). Discovering statistically non-redundant subgroups. *Knowledge-Based Systems., 67*, 315-327.

Nebro, A. J. (2009). Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems.*, 726-746.

NilSk1lz. (01 de 10 de 2020). *www.reddit.com/*. Obtenido de https://www.reddit.com/r/homeassistant/comments/j35lyr/using_ai_to_detect_anomalies/

Open Electronics. (27 de 02 de 2017). *www.open-electronics.org*. Obtenido de https://www.open-electronics.org/guest_projects/openmotics-a-completely-open-source-home-automation-platform/

OpenHAB. (2021). *https://www.openhab.org/*. Obtenido de https://www.openhab.org/

OpenMotics. (2021). *https://www.openmotics.com/*. Obtenido de https://www.openmotics.com/

OpenMotics. (2021). *https://www.openmotics.com/smart-metering/*. Obtenido de https://www.openmotics.com/smart-metering/

Rhasspy. (2021). *rhasspy.readthedocs.io*. Obtenido de https://rhasspy.readthedocs.io/en/latest/

Sam Barker, M. R. (2020). The Internet of Things: Consumer, Industrial & Public Services 2020-2024. *Juniper Research*.

SIMIDAT. (2021). *https://github.com/SIMIDAT/FEPDS*. Obtenido de https://github.com/SIMIDAT/FEPDS

SmartHome. (01 de 12 de 2017). *https://smartyhome.io/state-of-my-smart-home-late-2017/*. Obtenido de https://smartyhome.io/state-of-my-smart-home-late-2017/

soloelectronicos. (14 de 11 de 2019). *soloelectronicos.com*. Obtenido de https://soloelectronicos.com/2019/11/14/6-herramientas-de-domotica-de-codigo-abierto/

TECNOYFOTO. (2021). *tecnoyfoto.com*. Obtenido de https://tecnoyfoto.com/home-assistant-que-es

TensorFlow. (2021). *https://www.tensorflow.org/?hl=es-419*. Obtenido de https://www.tensorflow.org/?hl=es-419

TensowFlow. (22 de 4 de 2021). *https://www.tensorflow.org/api_docs*. Obtenido de https://www.tensorflow.org/api_docs

tutiempo.net. (2021). *https://www.tutiempo.net/clima/01-2020/ws-84170.html y https://es.weatherspark.com/m/36637/1/Tiempo-promedio-en-enero-en-Ja%C3%A9n-Espa%C3%B1a* . Obtenido de https://www.tutiempo.net/clima/01-2020/ws-84170.html y https://es.weatherspark.com/m/36637/1/Tiempo-promedio-en-enero-en-Ja%C3%A9n-Espa%C3%B1a

Velasco, J. G. (2009). *Energías renovables.* Editorial Reverte.

W. Shi, J. C. (2016). Edge Computing: Vision and Challenges,. *IEEE Internet of Things Journal.*, 637-646.

Wai-Ho Au, K. C. (2005). Mining changes in association rules: a fuzzy approach. *Fuzzy Sets and Systems., 149*, 87–104.

Wikipedia. (2021). *es.wikipedia.org*. Obtenido de https://es.wikipedia.org/wiki/Random_forest

Wikipedia. (2021). *https://es.wikipedia.org/wiki/Algoritmo_evolutivo*.

8. Apéndices.

8.1. Manual de instalación.

En esté apéndice se encontrará información acerca de cómo se realiza la instalación de todos los componentes que son necesarios para el trabajo. En la sección 8.1.1 se indicarán los pasos para instalar el sistema operativo a usar. Una sección 8.1.2 donde se explica cómo instalar y poner en marcha la plataforma Home Assistant, junto con dos herramientas más.

En la sección 8.1.3 se abordará la instalación de Apache Kafka, herramienta la cual permitirá efectuar la intercomunicación entre la plataforma y el algoritmo.

8.1.1. Instalación del sistema operativo.

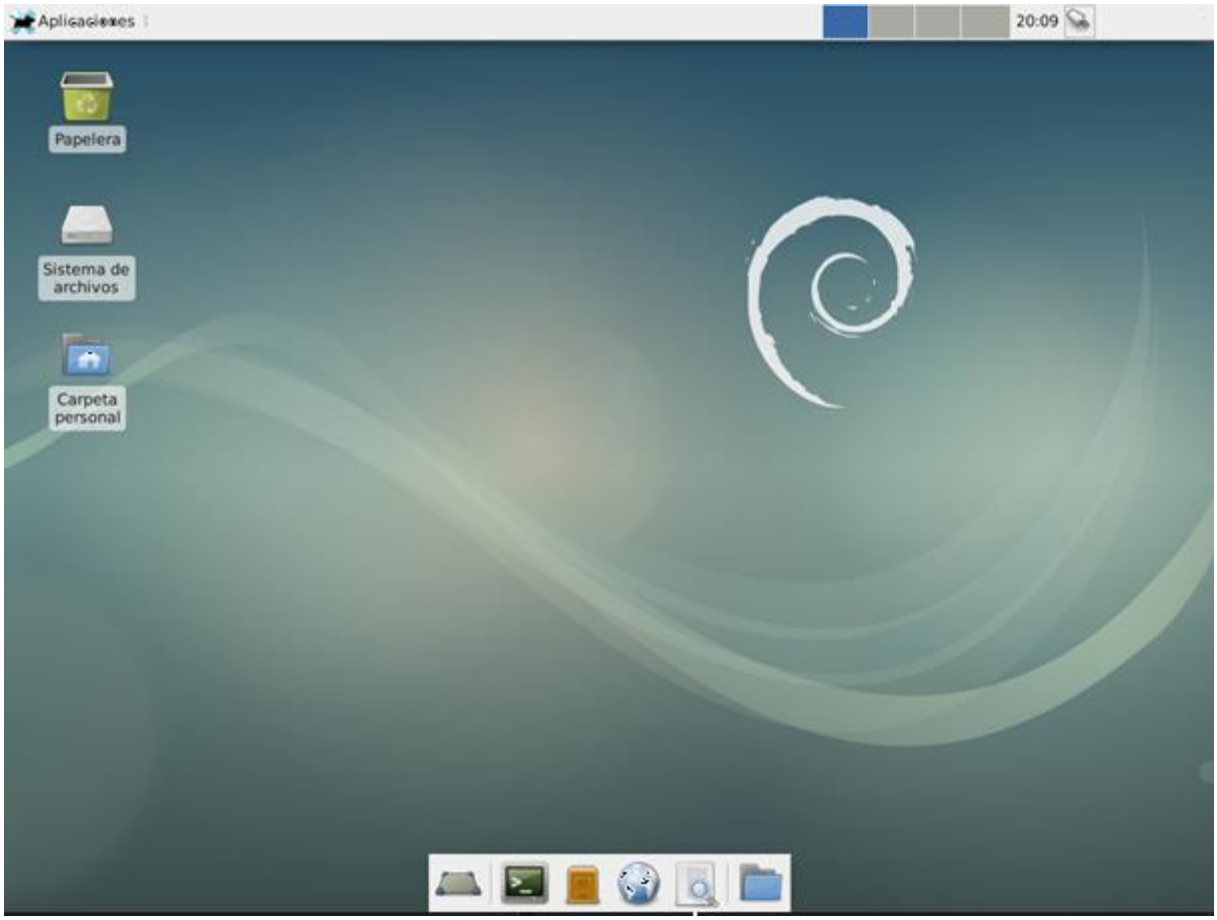

Figura 51.- Escritorio de Debian 10 con Xfce. Fuente: Elaboración propia.

Para comenzar el proceso de implantación es necesario descargar e instalar el sistema operativo que se utilizará. En este caso, se instalará el sistema operativo Debian 10. Para ello:

1. Se descargó del repositorio oficial de Debian: `https://raspi.debian.net/tested-images/`
2. Se introdujo la tarjeta SD en un adaptador y mediante el programa "*Balena Etcher*"[11] se realizó el flasheo de la imagen.

Se introdujo la tarjeta SD en la Raspberry Pi y se conectó todo lo necesario, tanto pantalla por HDMI, como periféricos apuntadores y teclado. Además del cable RJ-45 para tener Internet y la fuente de alimentación.

Figura 52.- Elementos conectados a la Raspberry Pi. Fuente: Elaboración propia.

Una vez conectado y arrancado, se procede a introducir las primeras órdenes:

1. Se realiza el login con el usuario "*root*".
2. `apt update`
3. `apt upgrade -y`
4. `apt install sudo -y`

[11] https://www.balena.io/etcher/

Una vez hecho esto se crea el usuario que se va a manejar:

1. `adduser` **`pi`**
2. Se introduce la contraseña que se quiera dos veces: **`raspipass`**
3. Se introduce la información adicional si se desea.
4. Se añade el usuario al grupo de los "*sudoers*": `usermod -aG sudo pi`

Acto seguido, y como desea tener una interfaz gráfica para poder trabajar de una manera más cómoda, se procede a instalarla:

1. `apt install -y xfce4`
2. Se selecciona "*Other*" → Spanish y luego otra vez Spanish, con las flechas y el intro.
3. Se instalan también otras herramientas que pueden ser interesantes o necesarias luego:
   a. `apt install -y xfce4-goodies`
4. `startx`
5. Se inicia una terminal para poner el sistema en español y se introduce: `sudo apt-get install locales`
6. `dpkg-reconfigure locales`, y se selecciona con la barra espaciadora:
   a. `s_ES@euro ISO-8859-15`
   b. `es_ES ISO-8859-1`
   c. `es_ES.UTF-8 UTF-8`
7. Se pulsa la tecla de intro y, en este caso, se ha seleccionado: `es_ES`
8. Luego se reinicia el sistema con: `reboot`

Por último, se instala un navegador. El elegido es "*Chromium*":

1. `sudo apt-get install chromium chromium-l10`

Una vez realizado esto y reiniciado. Se dispondría del sistema operativo instalado y listo para funcionar. Lo siguiente que se realizará será la instalación de la plataforma Home Assistant.

8.1.2. Instalación de Home Assistant, Portainer y Heimdall.

Una vez de vuelta en el sistema, se crean una serie de directorios necesarios, tanto para Docker como para el lugar donde se introducirá Hass.io.

1. Se inicia una terminal y se introducen las siguientes órdenes:
   a. `mkdir Docker`
   b. `mkdir Docker/Hassio`

Se prosigue con una serie de órdenes necesarias para instalar Home Assistant:

2. `sudo -i` (se necesita ser "*root*")
3. `apt-get install -y software-properties-common apparmor-utils apt-transport-https ca-certificates curl dbus jq network-manager`
4. `systemctl disable ModemManager`
5. `systemctl stopModemManager`
6. `curl -fsSL get.docker.com | sh`
7. `curl -sL "https://raw.githubusercontent.com/Kanga-Who/home-assistant/master/supervised-installer.sh" | bash -s -- -m raspberrypi4 .d /home/pi/Docker/Hassio`
   Tanto en mitad de la orden como al final se debe introducir: el modelo del que se dispone, los cuales aparecen en: `https://github.com/home-assistant/supervised-installer`; como el directorio donde se quiere instalar.

Ahora se procede a instalar Docker-Compose para poder manejar los contenedores, se prefiere a Docker normal ya que es más cómodo de gestionar mediante un fichero .yml como se verá a continuación:

1. `sudo apt-get update`
2. `sudo apt-get install -y docker-compose`

Se añade el usuario "pi" que es el que se está manejando al grupo de Docker con la orden:

1. `sudo usermod -a -G docker pi`

Ahora se instalan las dependencias que tiene Hassio mediante la orden:

1. `sudo apt-get install bash avahi-daemon`

Una vez termine, proporcionará la IP donde se tiene disponible el supervisor de Home Assistant. En este caso es la: **192.168.0.29:8123**.

Habrá que esperar alrededor de 20 minutos para que termine de inicializarse, aunque debería de tardar menos. Una vez dentro se siguen los pasos guiados que aparecen:

1. Se introduce el nombre de usuario y la contraseña,
    a. sergio, sergio
2. Ubicación de dónde se encuentra la vivienda,
    a. Jaén a 573 metros de altitud.
    b. Se usarán grados Celsius.

Ahora se podrían instalar las integraciones que se quiera, pero se realizará más adelante. Antes de seguir más con Hassio lo que se va a realizar es instalar dos utilidades que vendrán bien por si se quiere realizar luego otras gestiones con la Raspberry Pi.

Se crea un fichero llamado `docker-compose.yml` en la ruta `/home/pi/Docker` con la siguiente información:

```
version: '3'

services:
  portainer:
    image: portainer/portainer:latest
    container_name: portainer
    ports:
      - 9000:9000
    volumes:
      - /home/pi/Docker/Portainer:/data
      - /var/run/docker.sock:/var/run/docker.sock
    restart: always
  heimdall:
    image: linuxserver/heimdall:latest
    container_name: heimdall
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Madrid
    volumes:
      - /home/pi/Docker/Heimdall/config:/config
    ports:
      - 85:80
    restart: always
```

Código 36.- Docker-Compose.yml

Hay que dirigirse a dicho directorio y se ejecuta:

1. `cd Docker`
2. `docker-compose up -d`

Con esto se dispone de Portainer y Heimdall instalados:

- **Portainer**: permite ver todos los contenedores Docker que se tengan instalados y manejarlos fácilmente desde una interfaz web. Además de controlarlos se puede acceder a sus archivos de registro de manera muy simple. Está ubicada en el puerto 9000, con lo que se accede mediante la IP: 192.168.0.29:9000.

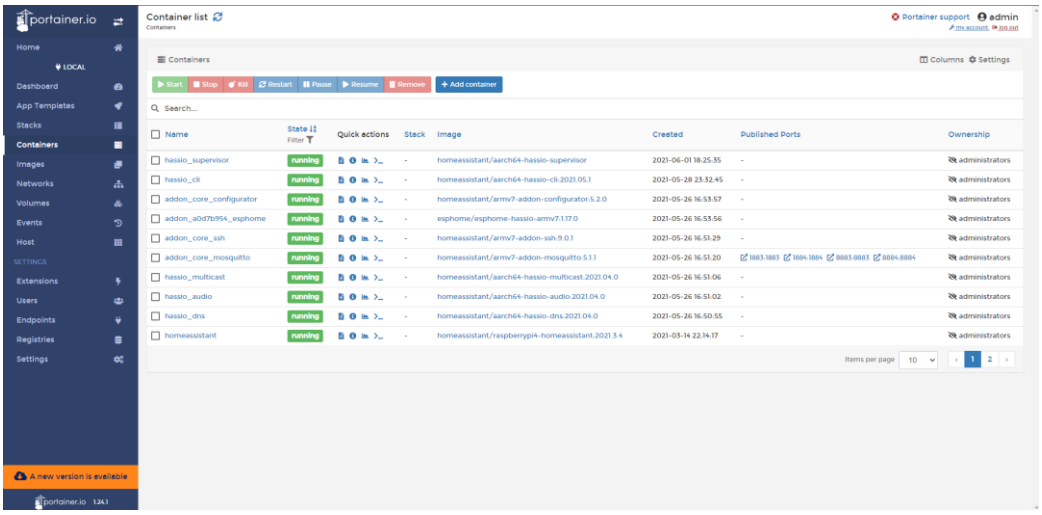

Figura 53.- Vista previa de Portainer. Fuente: Elaboración propia. Fuente: Elaboración propia.

- **Heimdall**: permite tener "tarjetas" personalizadas para acceder directamente y con un único click a lo que se establezca en su configuración. Se puede configurar también el idioma para ponerlo en español y se usará básicamente para entrar a Home Assistant y a Portainer sin necesidad de introducir las IP y los puertos. Solamente hay que guardar esta página y desde ella acceder a todas, se accede mediante el puerto 85 →

192.168.0.29:85.

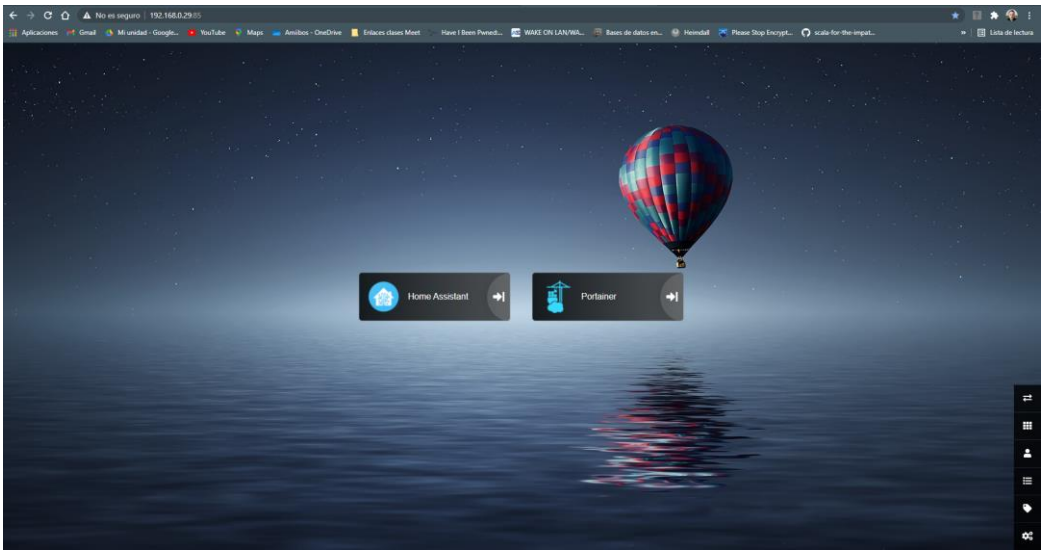

Figura 54.- Vista previa de Heimdall. Fuente: Elaboración propia. Fuente: Elaboración propia.

Por último, se va a activar el modo avanzado, para ello:

1. Se accede, en el supervisor de la web, al perfil de usuario situado abajo a la izquierda.
2. Se activa el botón de radio del Modo avanzado.

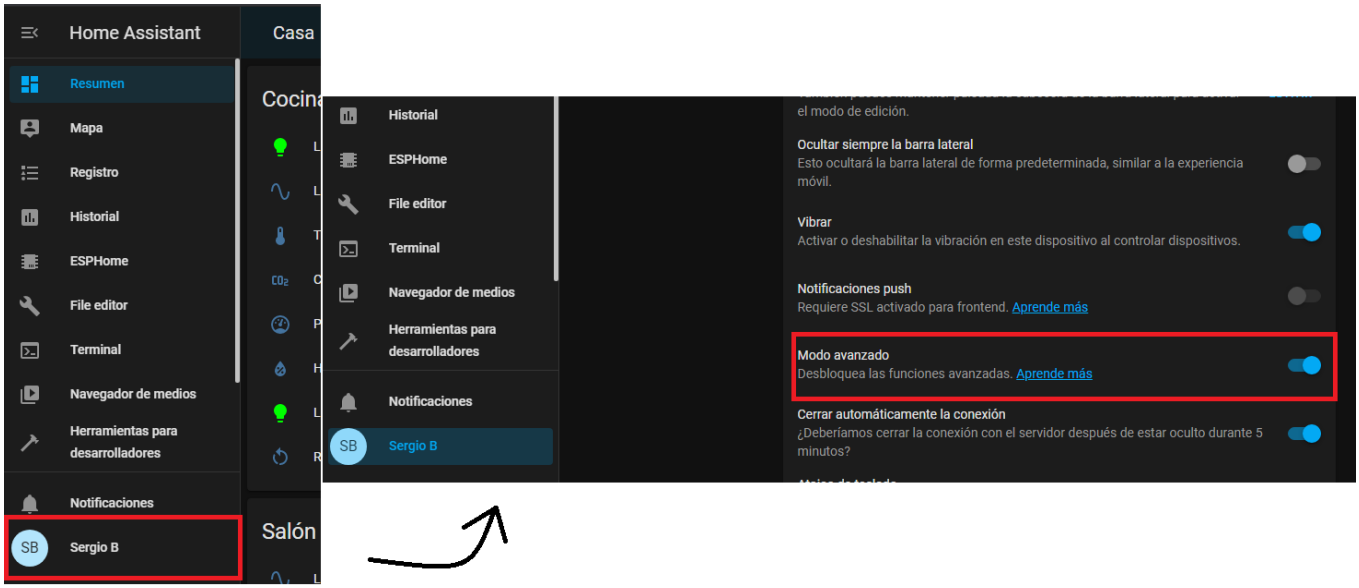

Figura 55.- Activación del modo avanzado. Fuente: Elaboración propia.

8.1.3. **Instalación de Apache Kafka.**

Se instala java: `sudo apt-get install default-jdk`

Se crea el directorio `/home/pi/proyectos` donde se descargará Apache Kafka. Se descargan los binarios de Apache Kafka de la página oficial: `kafka.apache.org`

Se mueve a la carpeta de proyectos y se descomprime:

1. `sudo mv ./Downloads/kafka-2.7.0-src.tgz ./proyectos/`
2. `cd proyectos`
3. `tar -xzf kafka_2.13-2.7.0.tgz`
4. `rm ./proyectos/kafka_2.13-2.7.0.tgz`

Se procede a configurar Apache Kafka mediante los ficheros de configuración dentro de la carpeta `/proyectos/kafka_2.11-2.4.0/config`

1. `./server.properties`
    a. `listeners=PLAINTEXT://192.168.0.29:9092`

Ahora se inician los procesos de Zookeeper y el servidor propiamente: (cada uno en una terminal diferente)

1. `sudo bin/zookeeper-server-start.sh config/zookeeper.properties`
2. `sudo bin/kafka-server-start.sh config/server.properties`

Se crean un par de tópicos para comprobar que funciona bien:

1. `bin/kafka-topics.sh --create --bootstrap-server 192.168.0.29:9092 --replication-factor 1 --partitions 1 --topic demo01`

2. `bin/kafka-topics.sh --create --bootstrap-server 192.168.0.29:9092 --replication-factor 1 --partitions 1 --topic demo02`

En esa misma terminal se procede a crear el que será el "*consumer*" y en otra consola el "*producer*" para ver que todo va bien. Además, en una tercera se arranca el "*consumer*" del tópico 2 para ver que solamente se publican los mensajes dónde deben:

1. `bin/kafka-console-consumer.sh --bootstrap-server 192.168.0.29:9092 --from-beginning --topic demo01`
2. `bin/kafka-console-consumer.sh --bootstrap-server 192.168.0.29:9092 --from-beginning --topic demo02`
3. `bin/kafka-console-producer.sh --broker-list 192.168.0.29:9092 --topic demo01`

Se aprecia así que el funcionamiento es el adecuado y se termina la instalación de Apache Kafka.

8.2. Preparación de componentes.

8.2.1. **Paso 1: Flasheo de Wemos D1 Mini con ESPHome.**

Lo primero será añadir el "*add-on*" de ESPHome mediante la tienda que trae incorporada en el supervisor del Home Assistant para poder manejar los dispositivos de manera más cómoda y mediante OTA.

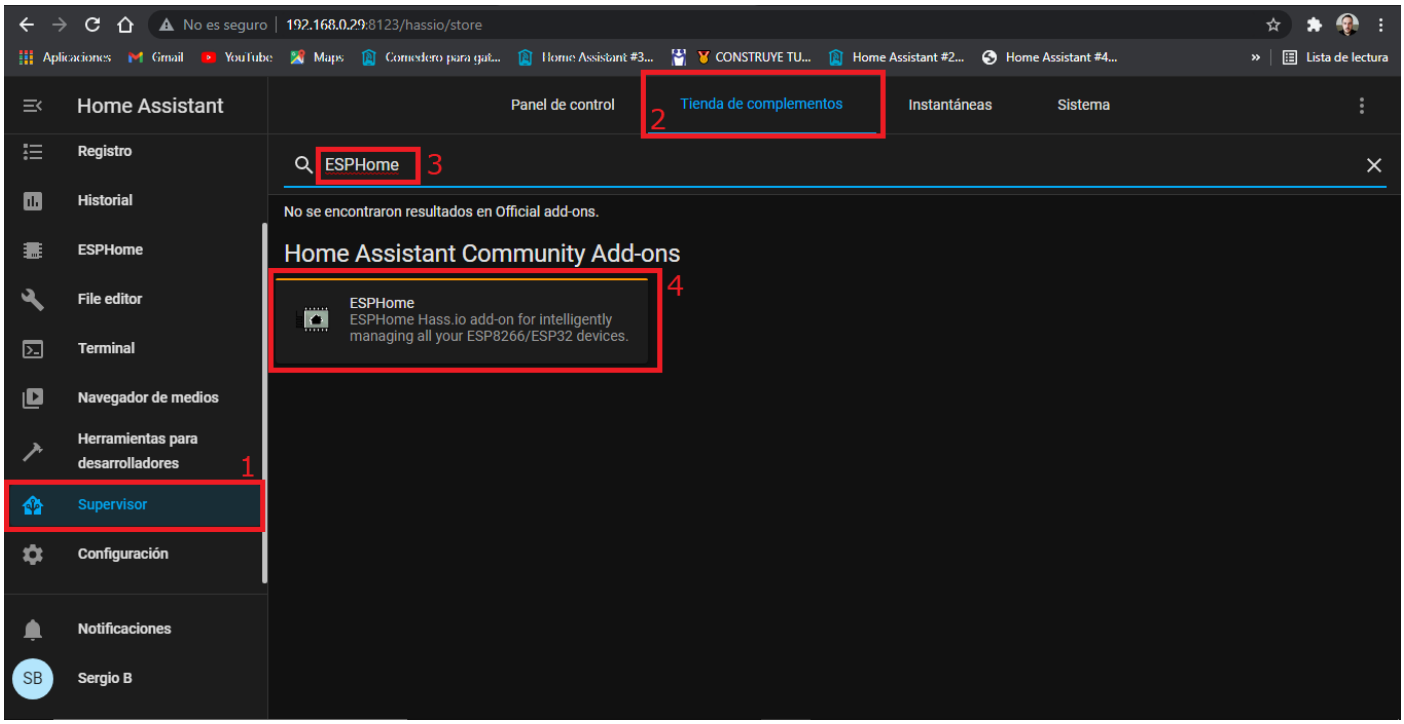

Figura 56.- Pasos para instalar cualquier add-on. Fuente: Elaboración propia.

Una vez instalado e iniciado se conecta la placa Wemos D1 Mini a la Raspberry Pi mediante un cable usb-microUsb y se selecciona el puerto usb como se muestra en la Figura 57.

Figura 57.- Selección del puerto USB correspondiente en el menú desplegable. Fuente: Elaboración propia.

Se pulsa el botón "+" verde que aparece abajo y se rellenan los datos:

1. Nombre: wd1m_habitacion.
2. Tipo de dispositivo: Wemos D1 Mini.
3. Las credenciales del Wi-Fi.
4. Sumbit!

Se activa la opción de editar y se añade que se use la versión de arduino 2.5.1 debajo del tipo de placa. Si no se realiza esto, cuando se trate de subir la plantilla a la placa nos dará el error 127.

```
esphome:
    name: wd1m_habitacion
    platform: ESP8266
    board: d1_mini
    arduino_version: 2.5.1
```

Figura 58.- Configuración inicial de cada placa. Fuente: Elaboración propia.

Una vez hecho esto, hay que cerciorarse de que siga escogido en el desplegable de arriba la opción de USB y se presiona la opción de "*Upload*".

8.2.2. **Paso 2: Soldadura de componentes.**

El siguiente paso será soldar las patillas que trae cada sensor a los correspondientes pines. Para ello, con el soldador y un poco de estaño se van uniendo y soldando cada patilla con cuidado de no provocar ningún cortocircuito entre las patillas.

Figura 61.- Soldador y Wemos D1 Mini sin soldar. Fuente: Elaboración propia.

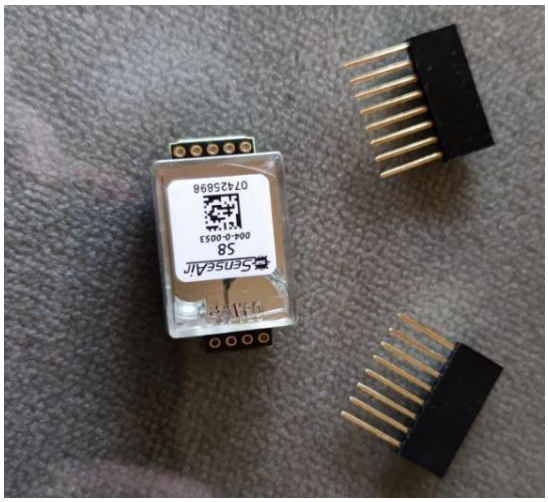

Figura 59.- Senseair S8 (CO_2). Fuente: Elaboración propia.

Figura 60.- TEMT6000 (Luminosidad). Fuente: Elaboración propia.

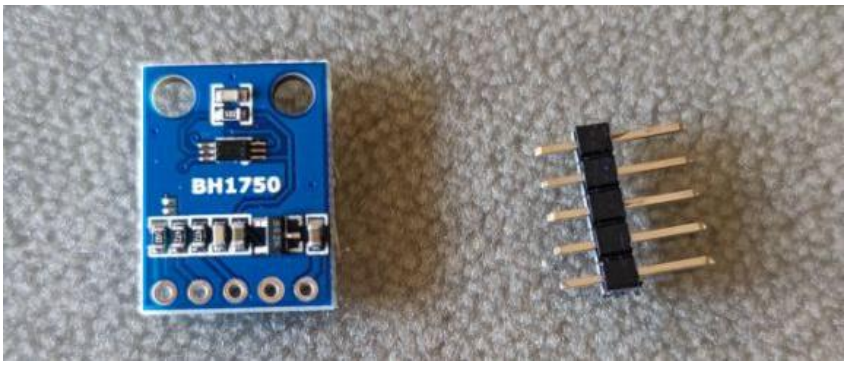

Figura 62.- BH1750 (Luminosidad). Fuente: Elaboración propia.

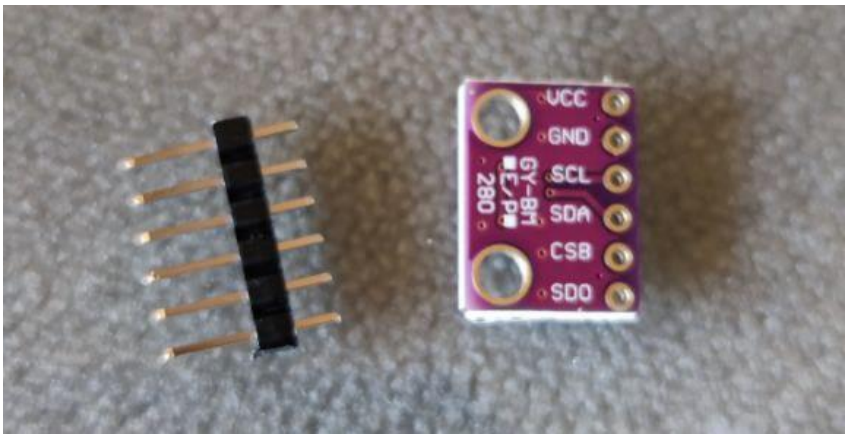

Figura 63.- BME280 (Temperatura, presión y humedad). Fuente: Elaboración propia.

En la Figura 64 se puede apreciar cómo se tienen todos los sensores con sus patillas unidas a las placas a las cuales se podrán unir cables Dupond y/o conectarlas a las placas Protoboard de las que se dispone.

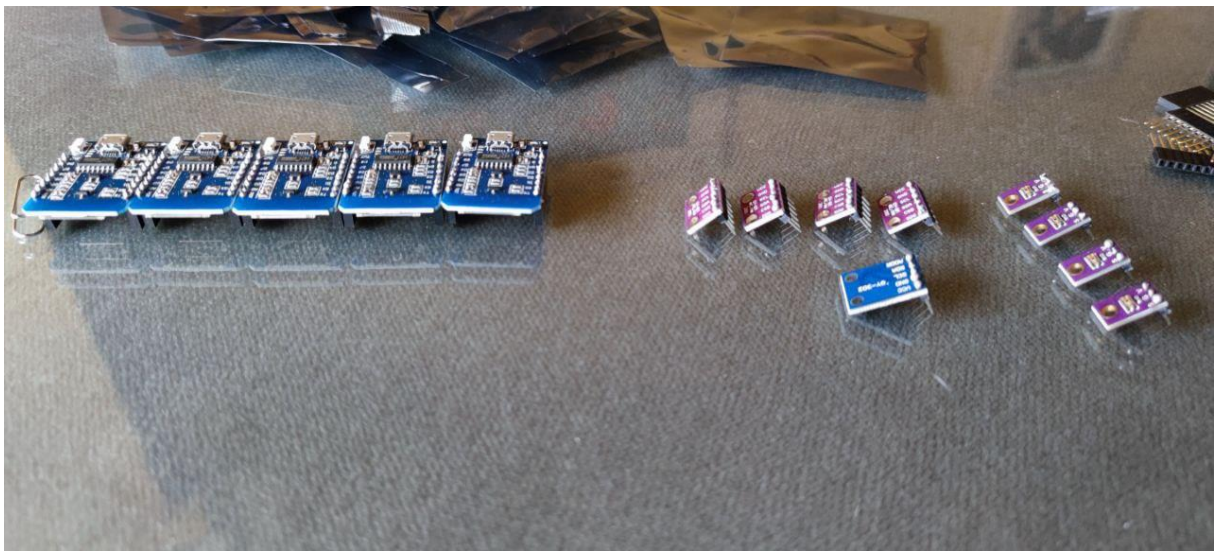

Figura 64.- Componentes soldados. Fuente: Elaboración propia.

En el caso del sensor de CO_2 se ha decido usar un tipo de patillas de tipo de "regleta". Esto es así ya que se podrá pinchar de una manera robusta en las placas Protoboard, pudiendo además conectarle los cables necesarios en los pines que se necesitan usar.

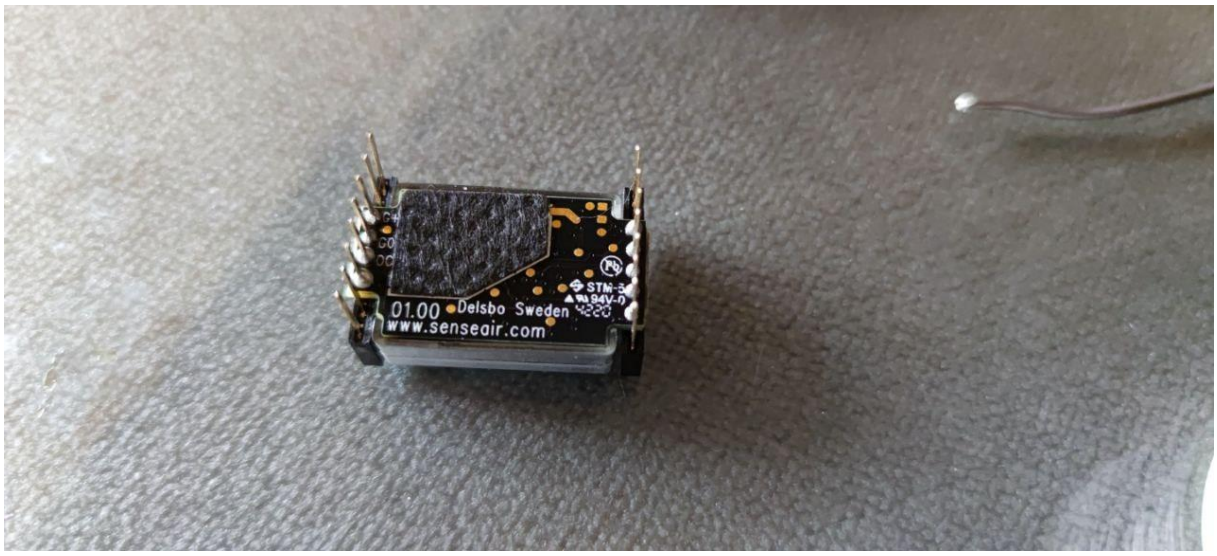

Figura 65.- Senseair S8 con las patillas soldadas. Fuente: Elaboración propia.

Una vez se termina este sensor, se debe proceder con el emisor de infrarrojos que se usará para poder controlar el aire acondicionado. En este, se ha realizado una pequeña abertura con una sierra para poder conectar los cables Dupond hembra-hembra. Además, se ha cortado una alargadera por la mitad para poder alimentarlo con los 220 voltios que requiere el dispositivo.

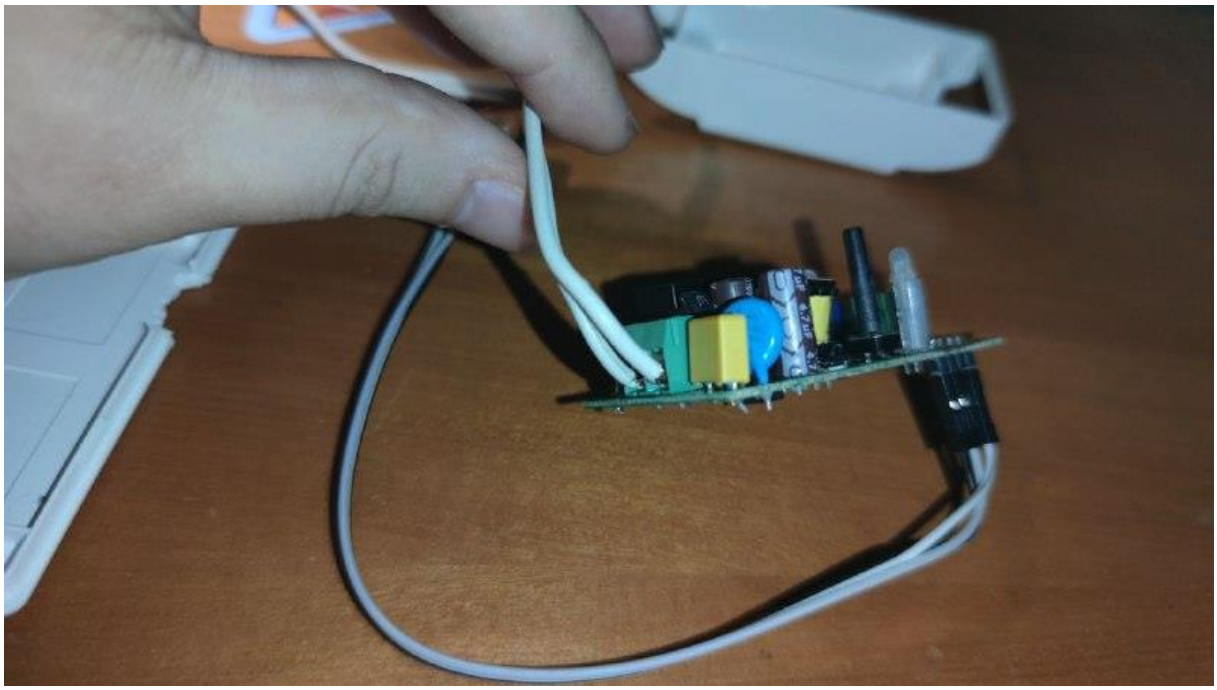

Figura 66.- Sonoff con el cable de alimentación. Fuente: Elaboración propia.

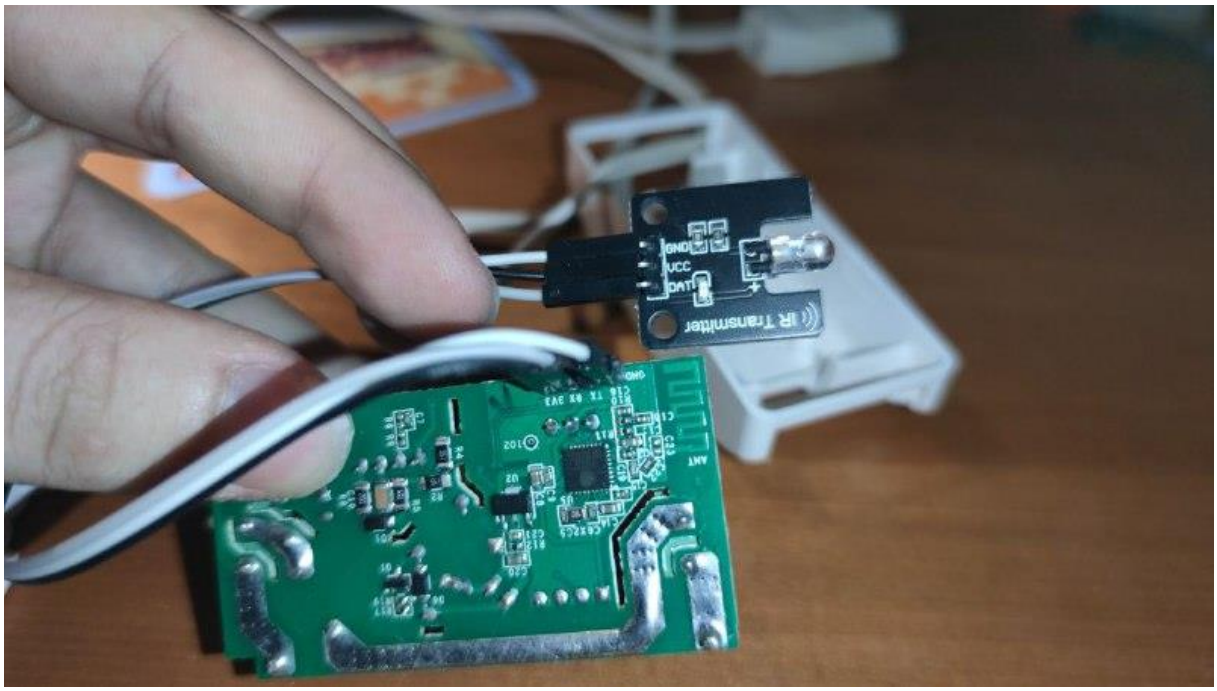

Figura 67.- Conexión de los pines del Sonoff con el emisor de infrarrojos:
VCC → 3.3V
GND → GND
DAT → RX
Fuente: Elaboración propia.

Figura 68.- Modificación de la carcasa del Sonoff. Fuente: Elaboración propia.

Para terminar, el último paso es gestionar el sensor de lluvia. Pero este ya viene soldado y listo para usar, solamente hay que conectar el controlador al detector como se muestra en la Figura 69.

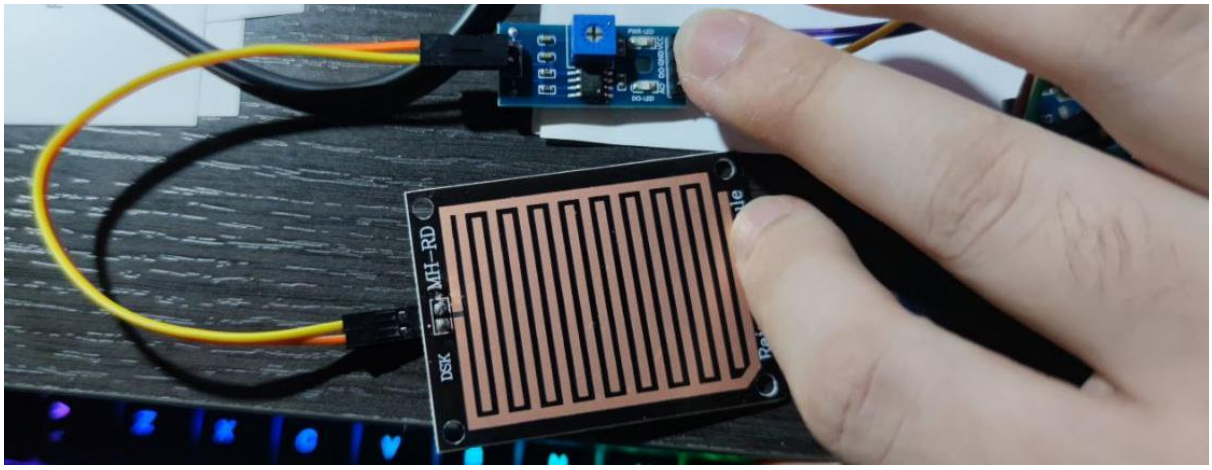

Figura 69.- Sensor de lluvia y controlador. Fuente: Elaboración propia.

8.2.3. **Paso 3: Montaje de los prototipos de estaciones.**

Una vez se tiene todo soldado y preparado, hay que proceder a conectarlo todo entre sí en lo que serán los prototipos que se colocarán en

las distintas partes de la casa para poder empezar a recoger datos y analizarlos posteriormente.

Para poder comunicarlos, se utilizarán las placas Protoboard y los cables Dupond de los 3 tipos: macho-macho, hembra-macho, hembra-hembra. Estos se pueden ver en la Figura 70.

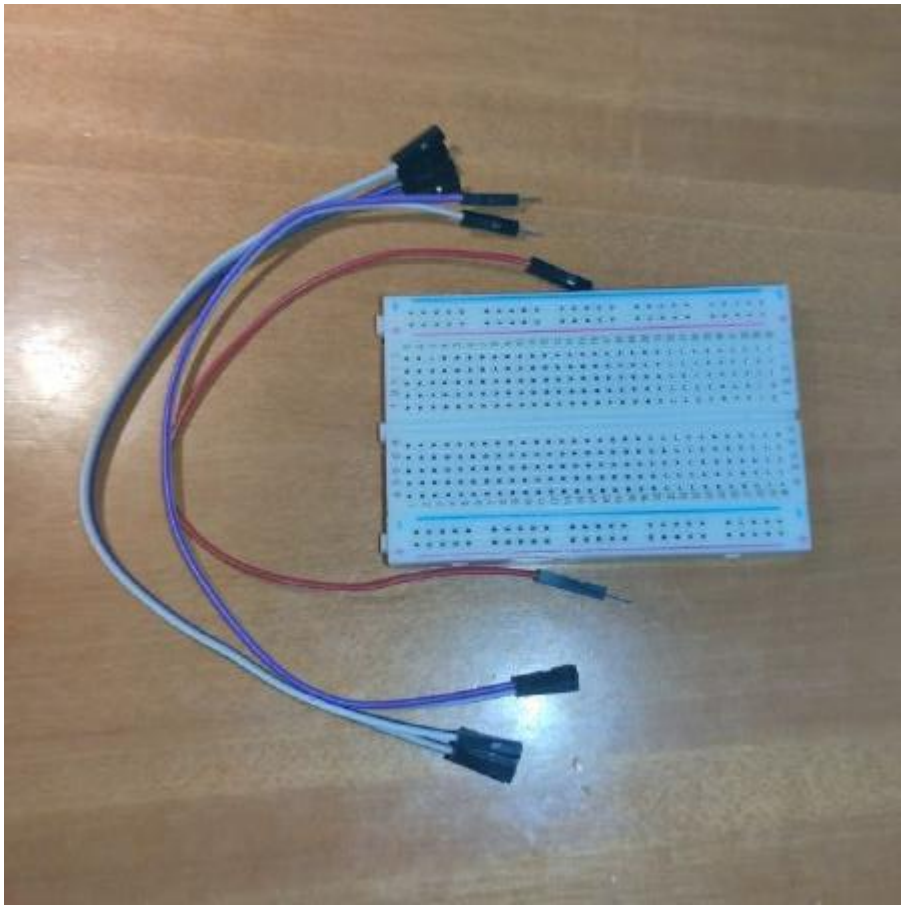

Figura 70.- Placa Protoboard y cables Dupond de los 3 tipos. Fuente: Elaboración propia.

Para la conexión se seguirá el esquema de la Figura 26. Lo primero que se montará será el sensor BME280, conectando para ello:

- Verde – GND – GND
- Naranja – 3.3 voltios – VCC
- Amarillo – D1 – SCL
- Azul – D2 – SDA

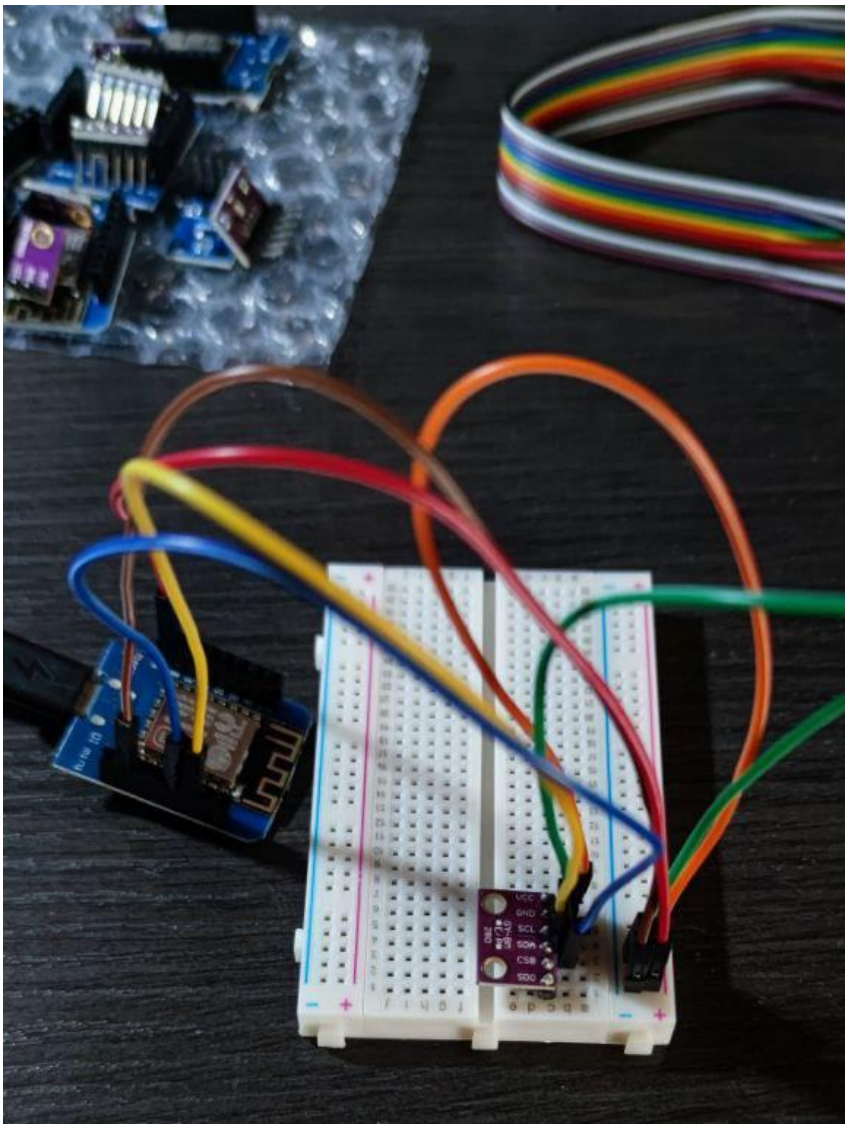

Figura 71.- Conexión del sensor BME280. Fuente: Elaboración propia.

Se prosigue con el sensor de luz TEMT6000 y se realizan las siguientes conexiones:

- Blanco – A0 – S
- Gris – GND – G
- Morado – 3.3 voltios – V

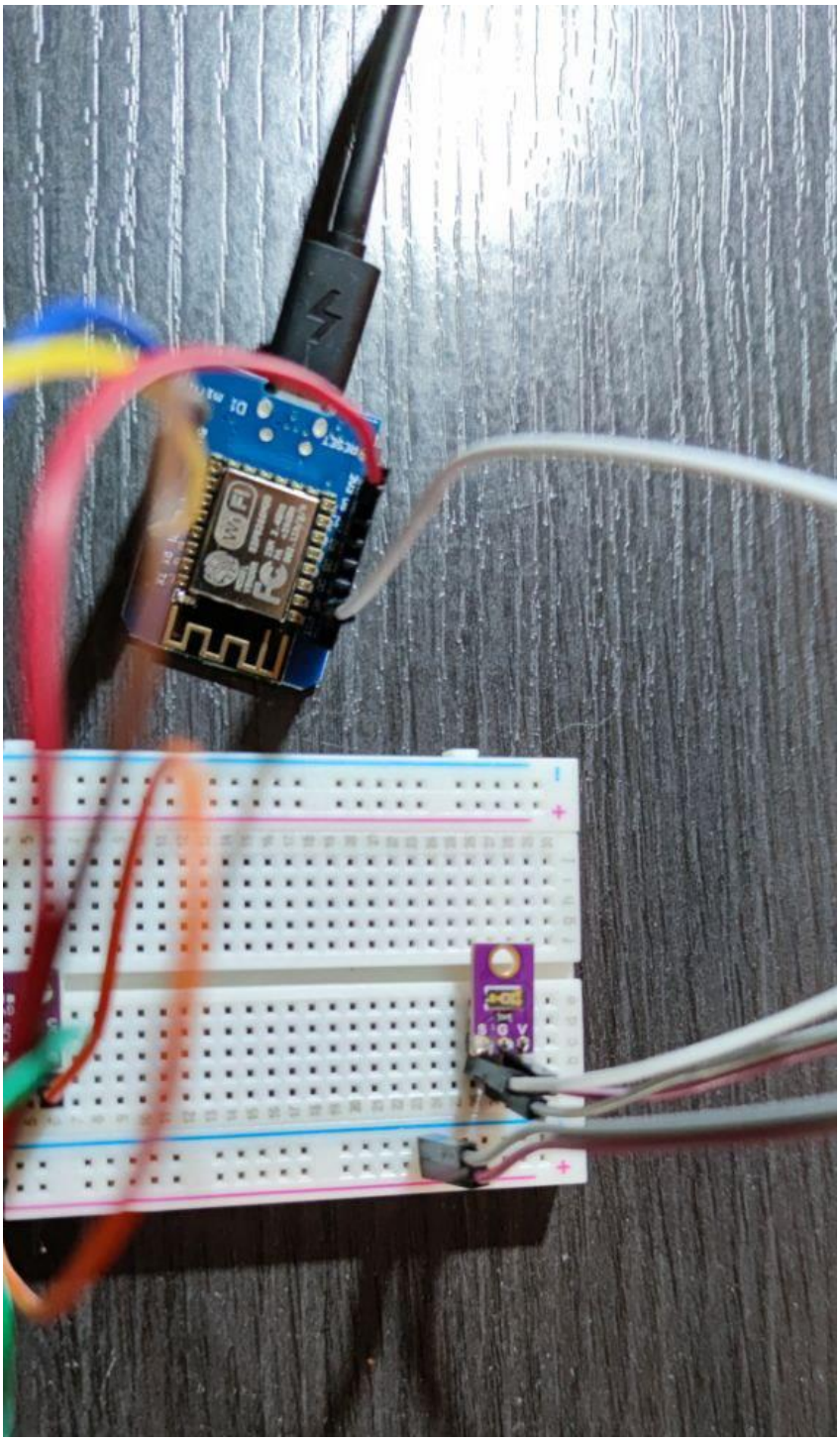

Figura 72.- Conexión del sensor TEMT6000. Fuente: Elaboración propia.

Acto seguido se procede con las conexiones para montar el sensor Senseair S8 encima de la placa. Se conectan los 4 cables que son necesarios de la manera siguiente:

- Marrón – 5V – VCC
- Negro – GND – GND
- Negro (unido al de color blanco) – TX – RX
- Blanco – RX – TX

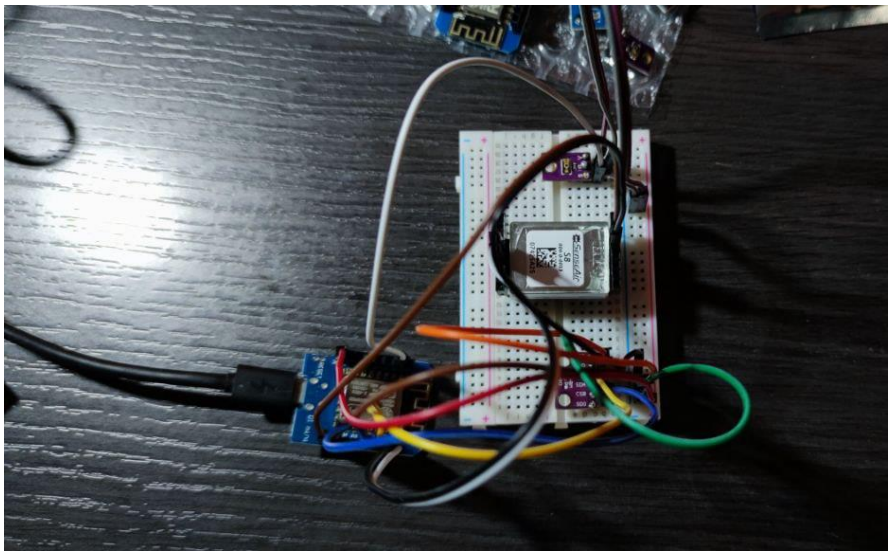

Figura 73.- Conexión del sensor Senseair S8 (junto con TEMT6000 y BME280). Fuente: Elaboración propia.

Finalmente se conectan los dos leds que representarán los actuadores finales. En este trabajo, simularán los distintos dispositivos finales que se quieran manejar de manera automática, como, por ejemplo: aires acondicionados o ventiladores, persianas eléctricas o las propias bombillas que tengan las luces de la casa.

Además, se han interconectado de dos maneras distintas los prototipos, uno está más orientado a que esté todo en la placa Protoboard y el otro está más orientado a poder disponer los distintos sensores de una manera más libre.

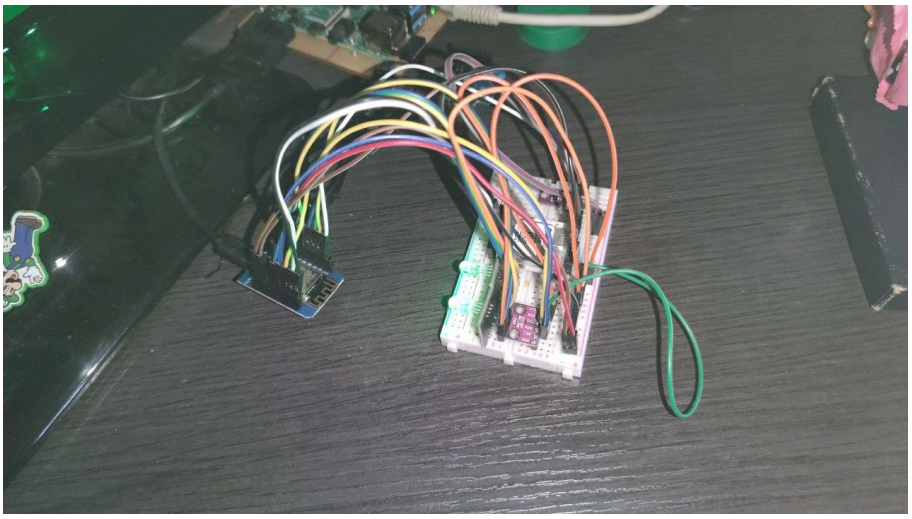

Figura 74.- Disposición 1 de Wemos D1 Mini. Orientado a integración en la placa. Fuente: Elaboración propia.

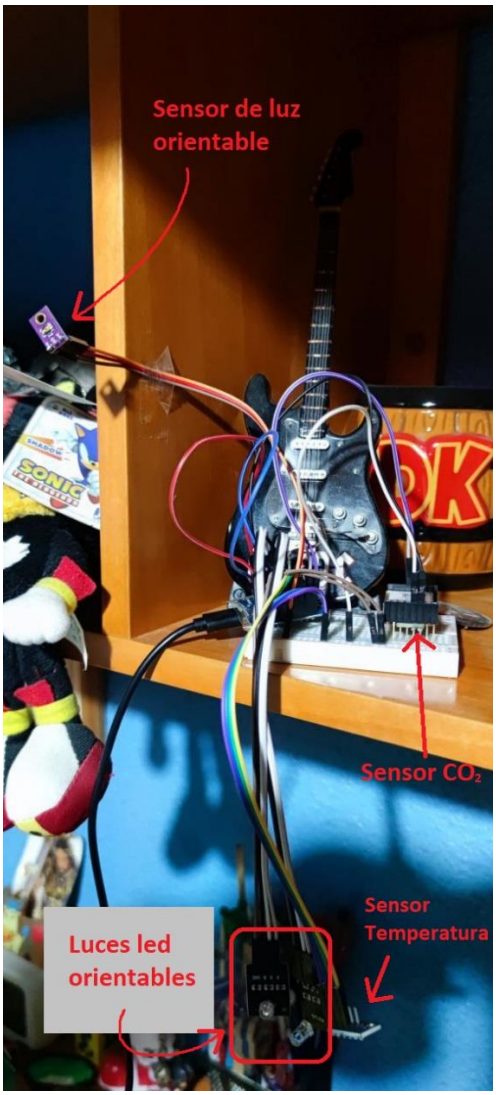

Figura 75.- Disposición 2 de Wemos D1 Mini. Orientado a libertad de posicionamiento de los sensores. Fuente: Elaboración propia.

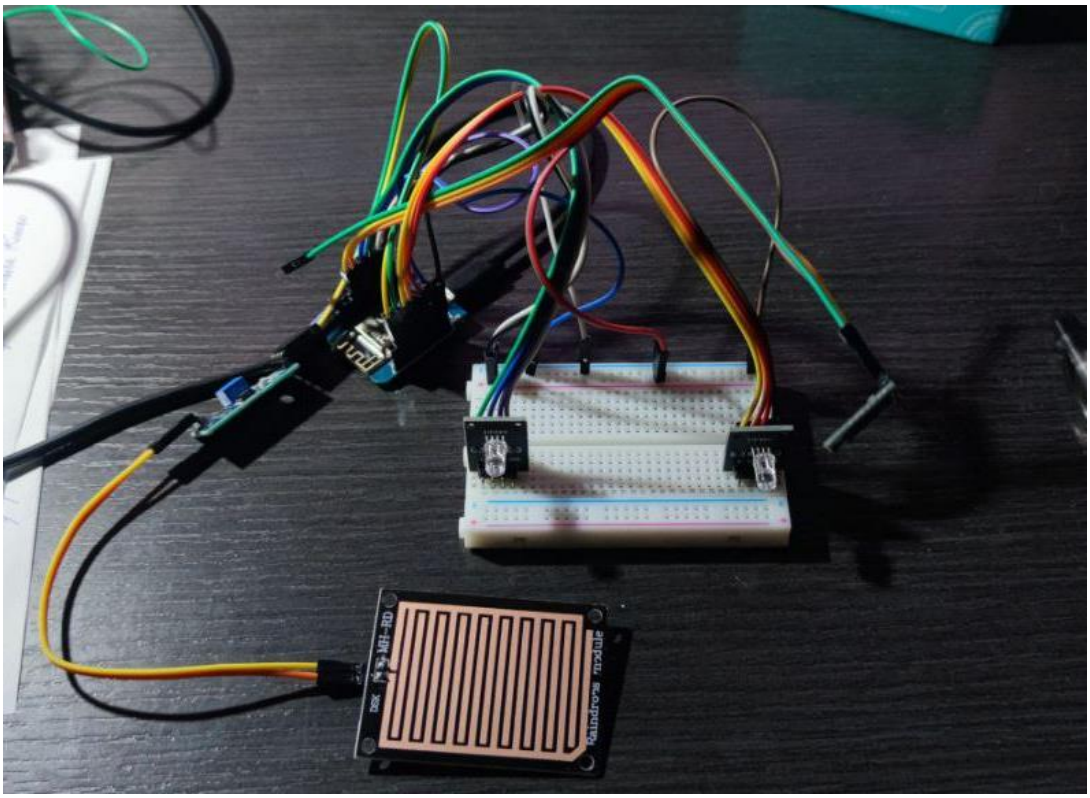

Figura 76.- Prototipo con sensor de luz y de lluvia con dos "actuadores". Fuente: Elaboración propia.

Se continua con el prototipo de estación que incorpora el sensor de lluvia. El resultado será el mostrado en la Figura 76. Para ello, se conectan los siguientes cables como se mostró en el diagrama de la Figura 27.

Lo primero será conectar el sensor de lluvia:

- Morado – 5V – VCC
- Azul – GND – GND
- Verde – D0 – D0
- Amarillo – A0 – A0

Lo segundo será conectar el sensor de luz BH1750:

- Rojo – 3.3V – VCC
- Marrón – GND – GND
- Naranja – D1 – SCL

- Amarillo – D2 – SDA

Al igual que en el caso anterior, el último paso sería conectar los dos leds que representarán los actuadores finales, por ejemplo, uno para la luz y otro para el sensor de lluvia.

8.2.4. Paso 4: Incorporación de sensores tanto a la placa Wemos como al Home Assistant mediante ESPhome.

Una vez flasheado, conectado a la red, y con todos los sensores pertinentes conectados. Se procede a escribir el "`.yaml`" que incorpore la configuración necesaria:

1. Se añade lo siguiente al fichero, debido a que el sensor BME280 que se va a usar requiere de conexión I2C:

```
i2c:
    scan: True
    id: bus_a
```

Código 37.- Uso del I2C del sensor BME280.

2. Y lo referente al sensor en sí mismo:

```
sensor:
  - platform: bme280
      temperature:
        name: "Temperatura Habitacion"
        oversampling: 16x
      pressure:
        name: "Precision Habitacion"
      humidity:
        name: "Humedad Habitacion"
      update_interval: 15s
      i2c_id: bus_a
      address: 0x76
```

Código 38.- Configuración de la placa para el sensor BME280.

Se definen las 3 magnitudes que se medirán y el nombre que se quiera proporcionar.

1. Para el sensor de luminosidad TEMT6000 solamente se debe añadir lo siguiente debajo de la otra plataforma:

```
- platform: adc
  pin: A0
  name: "Luminosidad Habitacion"
  unit_of_measurement: lx
  filters:
    - lambda: |-
        return (x / 10000.0) * 2000000.0;
```

Código 39.- Configuración de la placa para el sensor TEMT6000.

En este caso, se especifica que la plataforma será el conversor analógico-digital, estableciendo el único pin analógico que dispone la placa y el filtro que convertirá la medición en un valor interpretable en Lux.

2. Para el sensor Senseair S8, se debe definir también un UART, ya que usa RX y TX:

```
uart:
  rx_pin: RX
  tx_pin: TX
  baud_rate: 9600
```

Código 40.- Definición de la conexión UART para el sensor Senseair S8.

```
- platform: senseair
  co2:
    name: "CO2 Habitacion"
  update_interval: 15s
```

Código 41.- Configuración de la placa para el sensor Senseair S8.

3. Quedando el fichero de configuración de esta manera:

```
esphome:
    name: wd1m_habitacion
    platform: ESP8266
    board: d1_mini
    arduino_version: 2.5.1

wifi:
    ssid: "VodafoneC770"
    password: "--------------"

    # Enable fallback hotspot (captive portal) in case
    ap:
        ssid: "Wd1M Habitacion Fallback Hotspot"
        password: "BHRC9RU4Pq2i"

captive_portal:

# Enable logging
logger:

# Enable Home Assistant API
api:

ota:

i2c:
    scan: True
    id: bus_a

uart:
  rx_pin: RX
  tx_pin: TX
  baud_rate: 9600


sensor:
  - platform: bme280
    temperature:
      name: "Temperatura Habitacion"
      oversampling: 16x
    pressure:
      name: "Precision Habitacion"
    humidity:
      name: "Humedad Habitacion"
    update_interval: 15s
    i2c_id: bus_a
    address: 0x76

  - platform: adc
    pin: A0
    name: "Luminosidad Habitacion"
    unit_of_measurement: lx
    filters:
      -  lambda: |-
           return (x / 10000.0) * 2000000.0;
    update_interval: 15s
```

```
  - platform: sensair
    co2:
      name: "CO2 Habitacion"
    update_interval: 15s
```

Código 42.- Fichero final con la configuración de la placa.

Nota: Si la subida va a ser realizada por USB, los pines RX y TX del Wemos deben de estar desconectados.

4. Hasta aquí sería la entrada y recogida de datos. Para continuar, se prepararán las salidas mediante los leds.

```
light:
  - platform: rgb
    name: "Luz Temperatura Cocina"
    red: salida_roja_1
    green: salida_verde_1
    blue: salida_azul_1
  - platform: rgb
    name: "Luz CO2 Cocina"
    red: salida_roja_2
    green: salida_verde_2
    blue: salida_azul_2
```

Código 43.- Configuración de la placa para las luces RGB.

```
output:
  - platform: esp8266_pwm
    id: salida_roja_1
    pin: D0
  - platform: esp8266_pwm
    id: salida_verde_1
    pin: D5
  - platform: esp8266_pwm
    id: salida_azul_1
    pin: D6
  - platform: esp8266_pwm
    id: salida_roja_2
    pin: D7
  - platform: esp8266_pwm
    id: salida_verde_2
    pin: D8
  - platform: esp8266_pwm
    id: salida_azul_2
    pin: D3
```

Código 44.- Definición de los pines para las luces RGB.

El Código 42 es válido para todas las placas que dispone el proyecto. Lo único que habría que cambiar son los nombres para que correspondan con el nombre de la placa en cuestión.

Una vez configurado los sensores, Home Assistant notificará que ha detectado al dispositivo automáticamente. Lo único que habrá que hacer será añadir y configurar la integración, indicando en qué sala se encuentra el dispositivo.

Por otra parte, para la placa con el sensor de lluvia el proceso será análogo salvo para los dos sensores nuevos.

1. Para el sensor de luz BH1750 se necesita establecer lo siguiente:

```
sensor:
  - platform: bh1750
    name: "Luminosidad Exterior"
    measurement_time: 40
    update_interval: 15s
    i2c_id: bus_a
```

Código 45.- Configuración de la placa para el sensor BH1750. [12]

2. Para el sensor de lluvia propiamente dicho se establecerá lo siguiente:

```
binary_sensor:
  - platform: gpio
    pin:
      number: D0
      inverted: True
    name: "Sensor de Lluvia"
    device_class: moisture
```

Código 46.- Configuración de la placa para el sensor de lluvia.

3. Por último, solamente queda esperar a que la plataforma lo reconozca y añadirlo como se hizo anteriormente.

Tras realizar la configuración, se instalará el add-on File editor, para poder gestionar más adelante los ficheros de configuración de Home Assistant y poder integrar Apache Kafka, entre otras cosas. Esto se realiza de la misma manera que se puede apreciar en la Figura 56.

[12] El valor 40 se especifica ya que estará en un exterior y se requiere que tenga una sensibilidad máxima, si estuviera en interior podría subirse. Siempre en el rango [31-254]

8.2.5. Paso 5: Control remoto de aire acondicionado.

Lo primero que se tiene que hacer es flashear el Sonoff con una imagen de ESPEasy. Para ello se conectará el programador y el Sonoff como muestra la Figura 77.

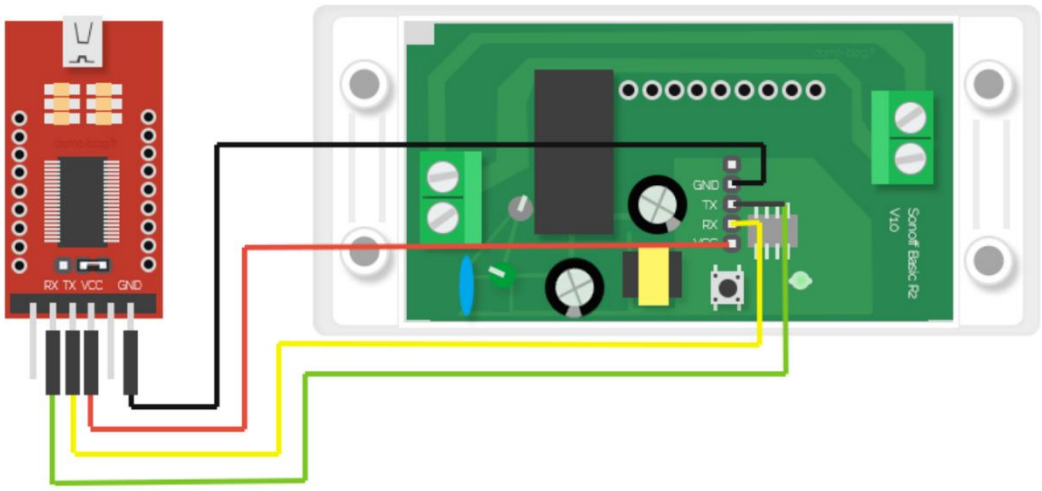

Figura 77.- Esquema de conexión programador-Sonoff. Fuente: Elaboración propia.

Se pulsa el botón que trae y se conecta al PC. Una vez conectado se suelta el botón. Usando la herramienta ESP Easy Flasher (letscontrolit.com, 2017) se carga el binario a introducir.

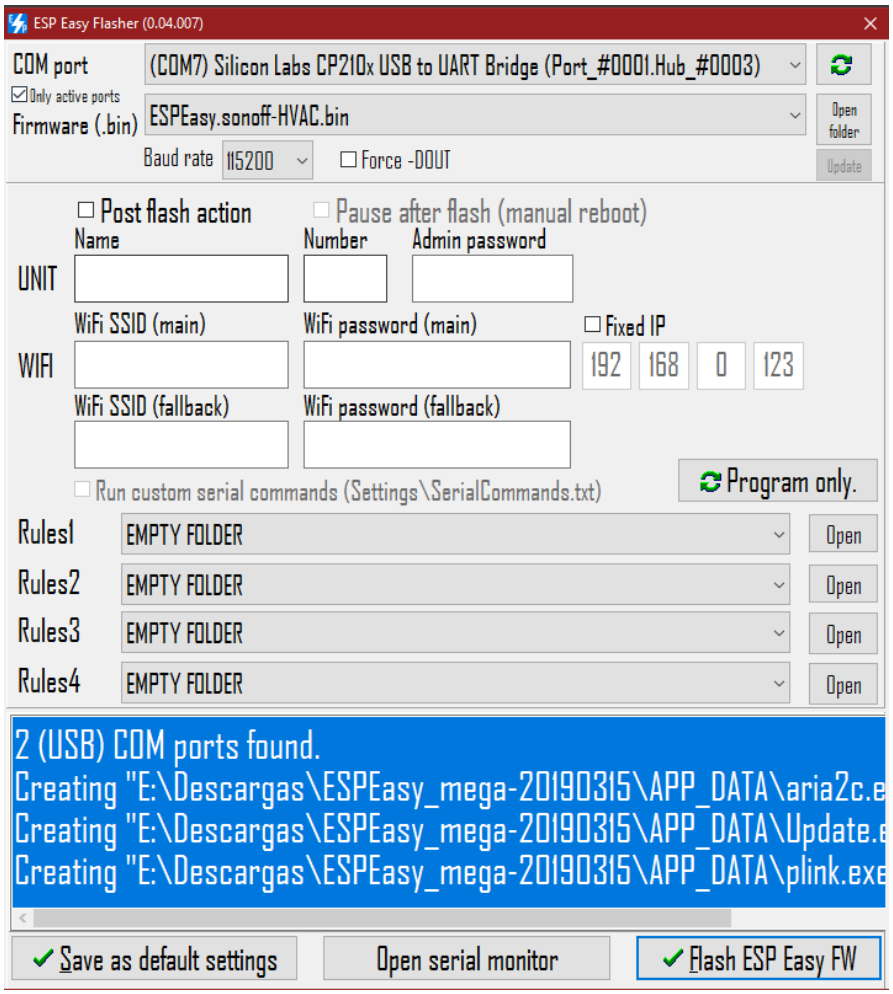

Figura 78.- ESP Easy Flasher con configuración del binario que se va a cargar en el Sonoff. Fuente: Elaboración propia.

Mientras se flashea, se instalará en Home Assistant el add-on "*Mosquitto Broker*" como hasta ahora y en la configuración del add-on se establecerá un usuario y una contraseña como se muestra en la Figura 79. Este add-on se requiere para poder comunicar el Sonoff con Home Assistant mediante el protocolo MQTT.

Opciones

```
logins: [{
  "username": "mqtt_user",
  "password": "mqtt_2021_S#B#R"
}]
anonymous: false
customize:
  active: false
  folder: mosquitto
certfile: fullchain.pem
keyfile: privkey.pem
require_certificate: false
```

Figura 79.- Configuración de MQTT. Fuente: Elaboración propia.

Se guarda el proceso y se le da a "*start*".

Una vez haya terminado y se haya flasheado el Sonoff, se desconecta y vuelve a conectar otra vez el usb y se busca en las redes Wi-Fi disponibles del ordenador. Debería de haber una nueva llamada ESP_0:

Figura 80.- Red local que crea el Sonoff Basic. Fuente: Elaboración propia.

Hay que conectarse a ella y acceder a la IP 192.168.4.1. La clave por defecto es "`configesp`", sin comillas. Se accede a la configuración y se modifica para que quede tal y como se presenta en la Figura 81.

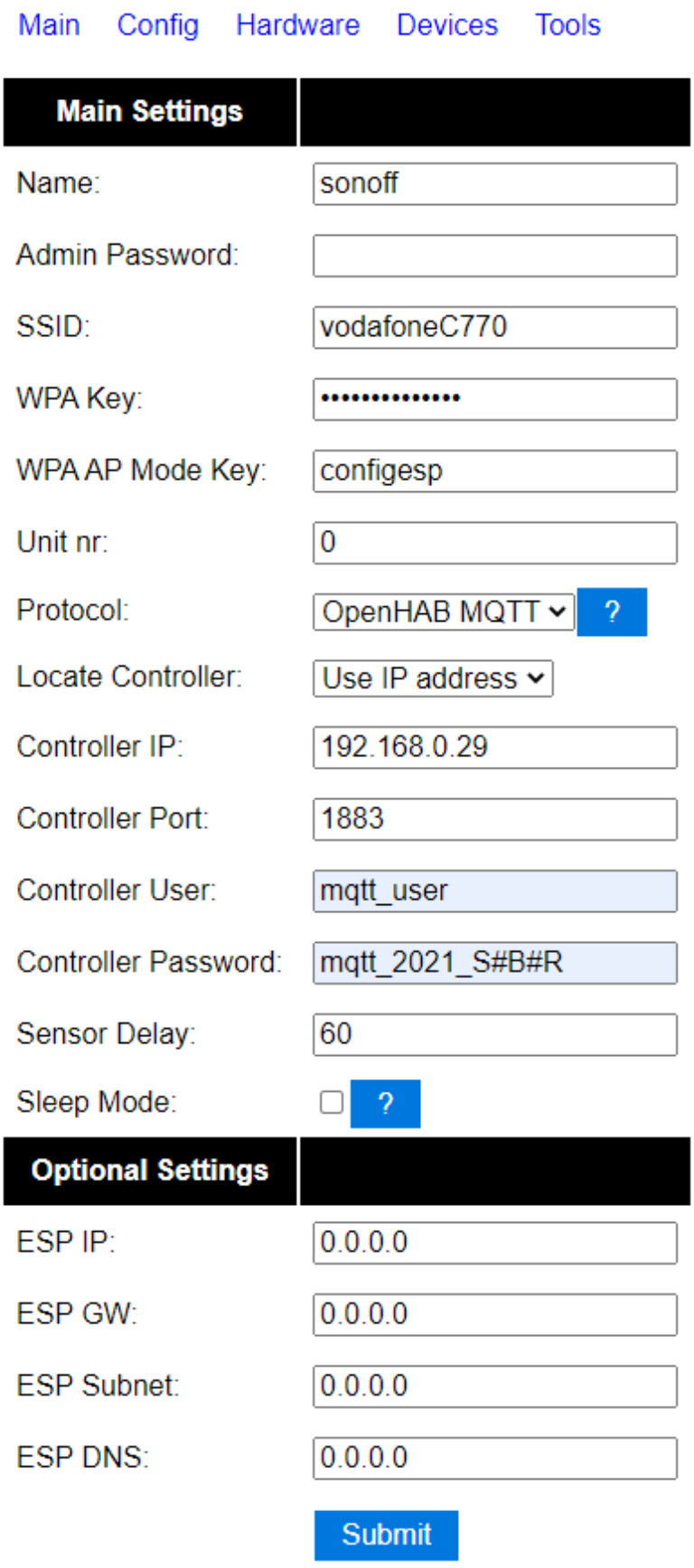

Figura 81.- Configuración del Sonoff. Fuente: Elaboración propia.

Ahora, dado que se requiere el pin RX para poder transmitir datos con el emisor de infrarrojos, hay que desactivar el puerto serie para poder usar tanto RX como TX. Para ello, se accede a la pestaña de Tools > Advanced:

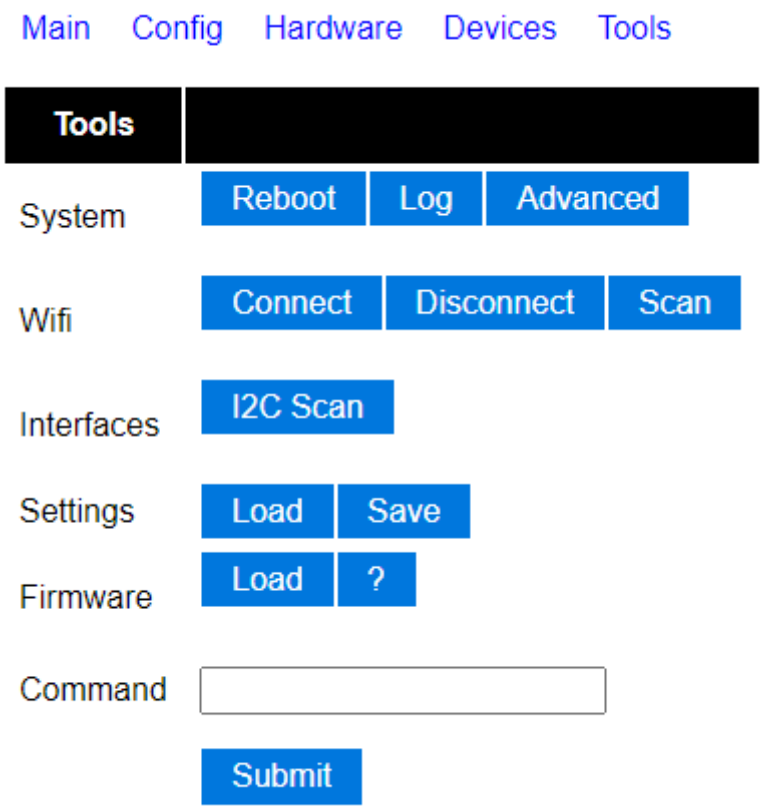

Figura 82.- Tools y luego en la pestaña de Advanced. Fuente: Elaboración propia.

Se deja toda la configuración por defecto, pero se desmarca la casilla del puerto serie. A continuación, se reinicia el dispositivo:

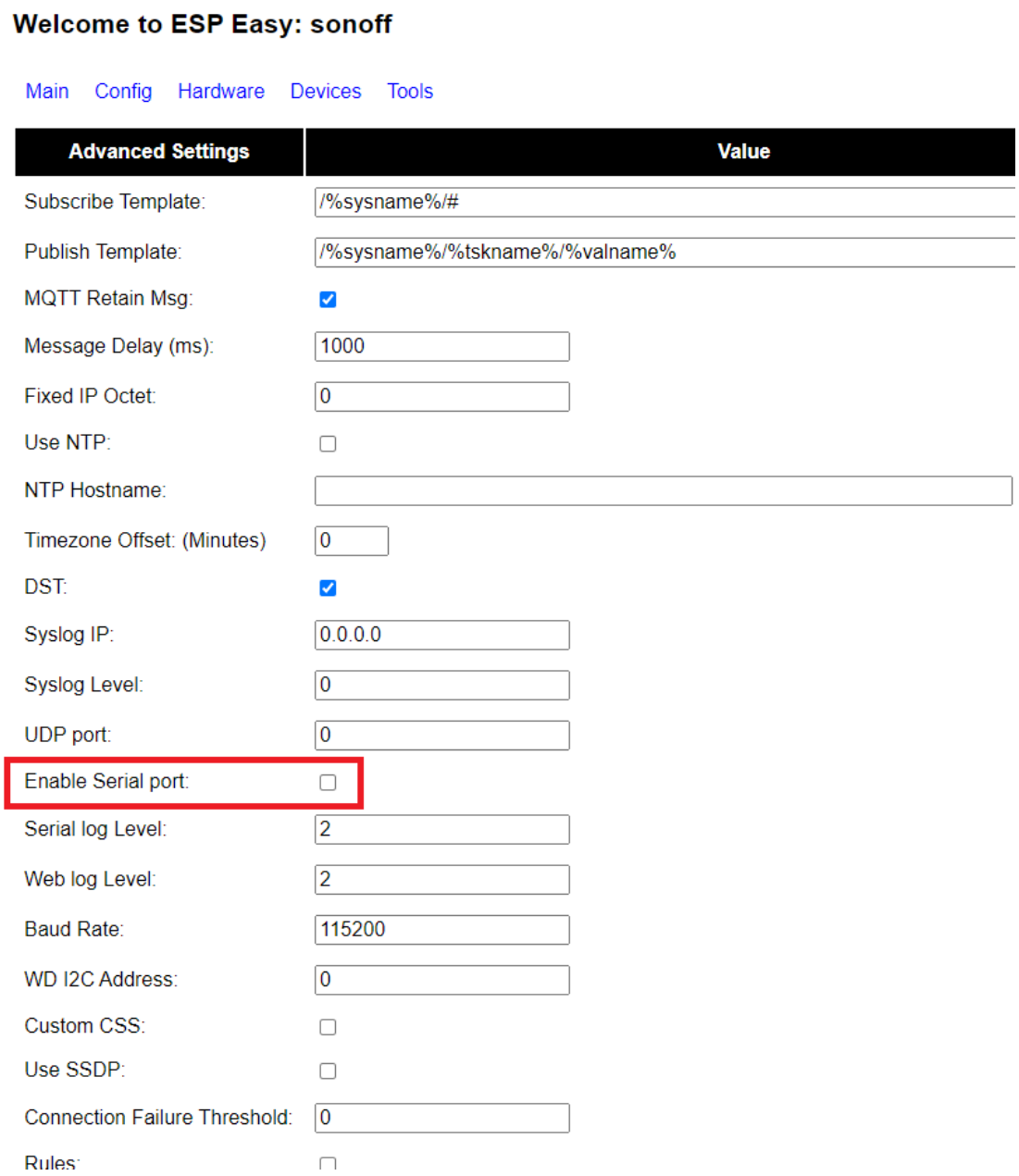

Figura 83.- Se desactiva el check en "Enable Serial Port" para poder usar los pines RX y TX como salidas/entradas. Fuente: Elaboración propia.

En la pestaña de Hardware se pueden configurar los pines que dispone la placa Sonoff. La configuración se realizará de la siguiente manera como se muestra en la Figura 84. En este caso, se establece que el led de estado estará asignado al pin 7, y que el SDA y SCL serán el pin 2 y 1 respectivamente.

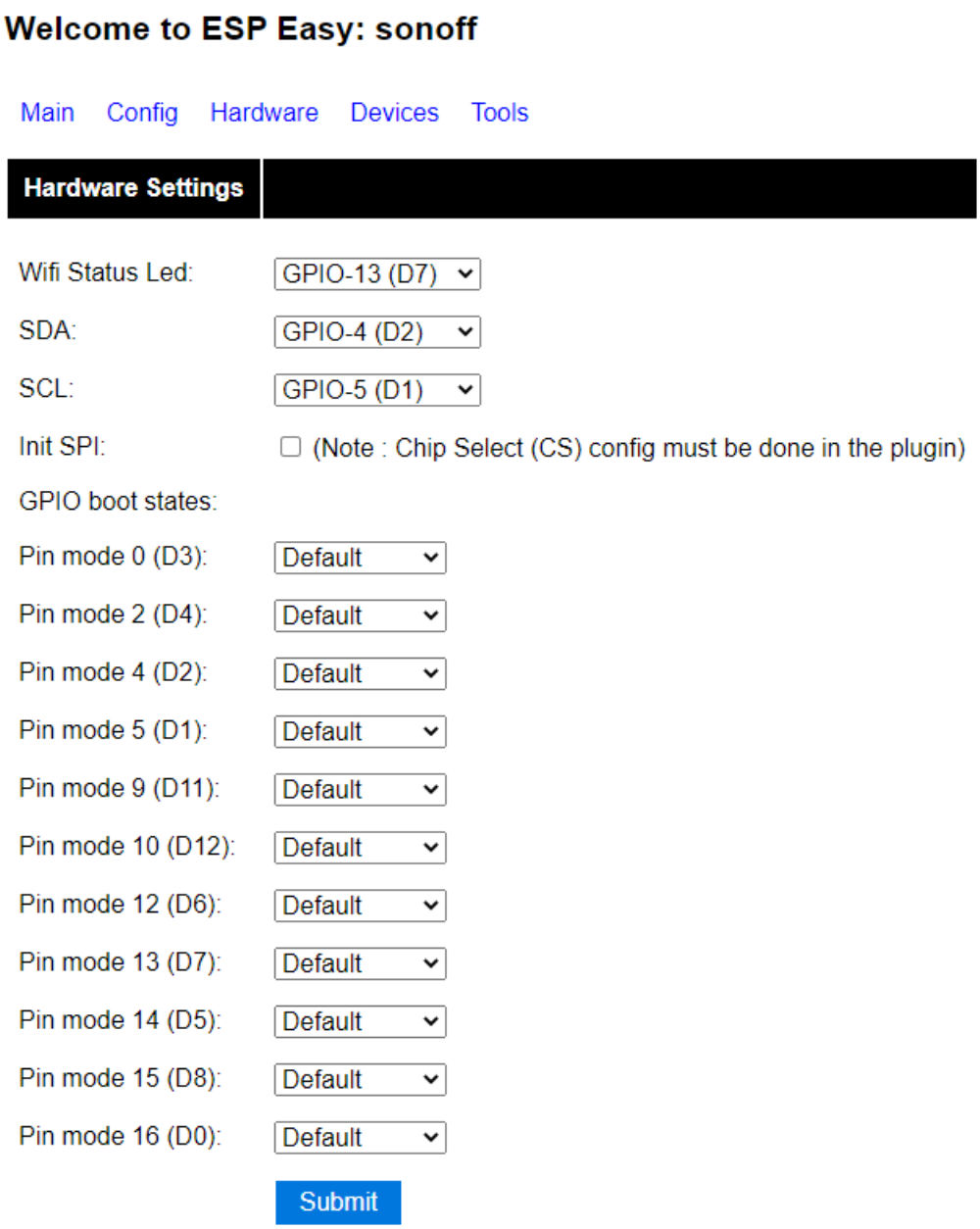

Figura 84.- Pestaña Hardware de Sonoff Basic. Fuente: Elaboración propia.

Para finalizar con la configuración de la placa Sonoff, habrá que desplazarse hasta el apartado Devices y se editará la primera sección.

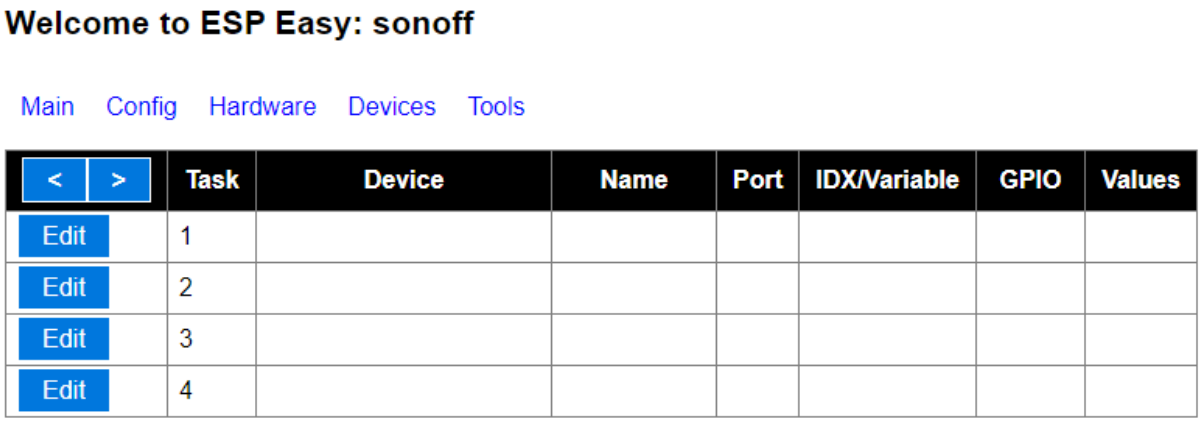

Figura 85.- Apartado Devices del Sonoff Basic. Fuente: Elaboración propia.

Se presiona en el botón de "*Edit*" y se rellena la configuración con lo que se muestra en la Figura 86.

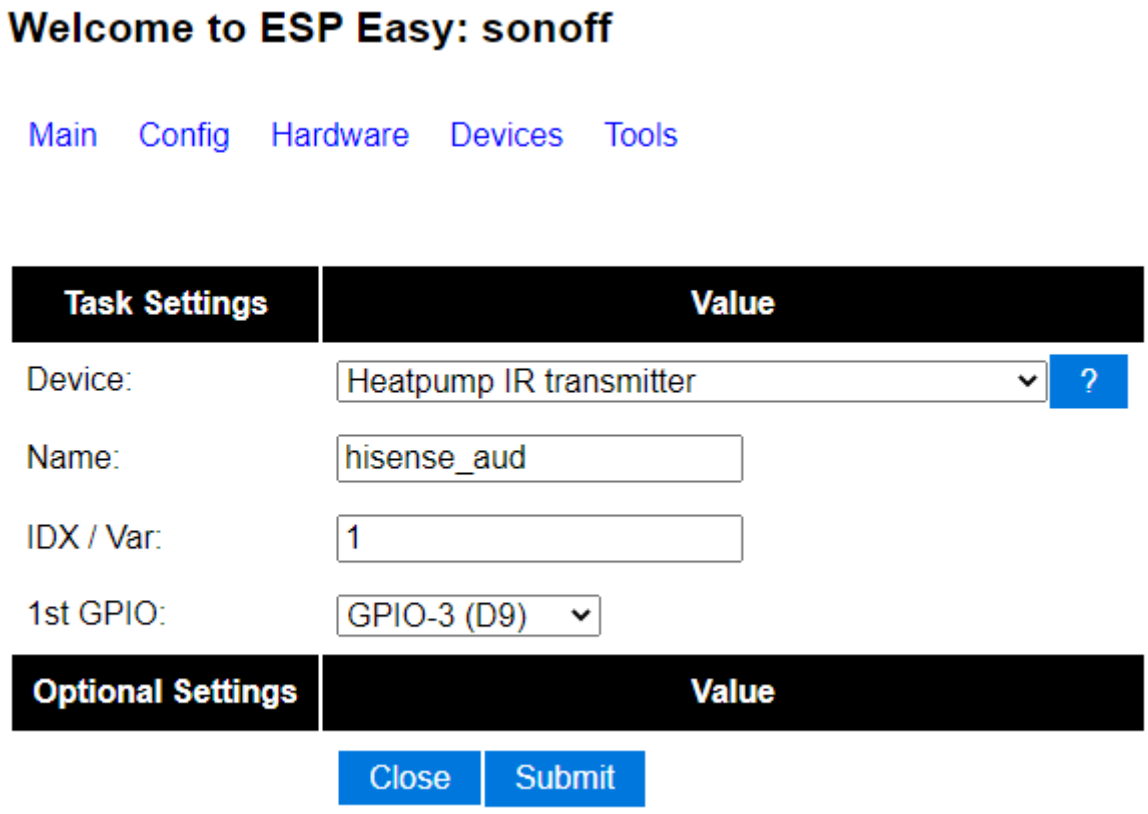

Figura 86.- Configuración necesaria para que el GPIO-3 (el RX) se establezca como salida estándar del dispositivo Heatpump IR transmitter [13]. Fuente: Elaboración propia.

[13] El nombre de "hisense_aud" es el que trae el binario configurado para el tipo de aires acondicionados de la marca Hisense.

Una vez se le dé a "*Submit*" se terminaría todo el trabajo respecto a esta parte del trabajo. El paso final será introducir en la plataforma de Home Assistant lo necesario para que aparezca representado el "mando a distancia", ya que al ser un dispositivo completamente personalizado no será detectable automáticamente. Con las siguientes líneas que se leen en el Código 47, y que se establecerán en el fichero de configuración general de la plataforma, se conseguirían dos tarjetas: una que solamente permite controlar el apagado/encendido del aire y otra bastante más compleja con los distintos modos de uso. Estas se presentan en las figuras 88 y 87, respectivamente.

```
switch:
- platform: mqtt
  name: "Aire acondicionado" # el nombre que queráis
  command_topic: "/sonoff/cmd" # el sonoff-daikin o el que hayáis
configurado en el apartado config de EspEasy.
  payload_on: "heatpumpir,1"
  payload_off: "heatpumpir,0"
  qos: 1
  retain: true

climate:
- platform: mqtt
  name: 'Aire acondicionado cuarto'
  initial: 27
  min_temp: 16
  max_temp: 30
#current_temperature_topic: /status/sonoff/HEATPUMPIR
  modes:
    - 'off'
    - 'on'
    - heat
    - cool
    - dry
    - fan
    - maint
    - auto
  swing_modes:
    - auto
    - swing
    - up
    - mup
    - middle
    - mdown
    - down
  fan_modes:
    - auto
    - maximum
    - high
    - medium
    - low
    - silent
  power_command_topic: /sonoff/HEATPUMPIR/POWER
  payload_on: 1
  payload_off: 0
  mode_command_topic: /sonoff/HEATPUMPIR/MODE
  temperature_command_topic: /sonoff/HEATPUMPIR/TEMP
  fan_mode_command_topic: /sonoff/HEATPUMPIR/FAN
  swing_mode_command_topic: /sonoff/HEATPUMPIR/HSWING
  qos: 1
```

Código 47.- Configuración en Home Assistant para el control del aire acondicionado.

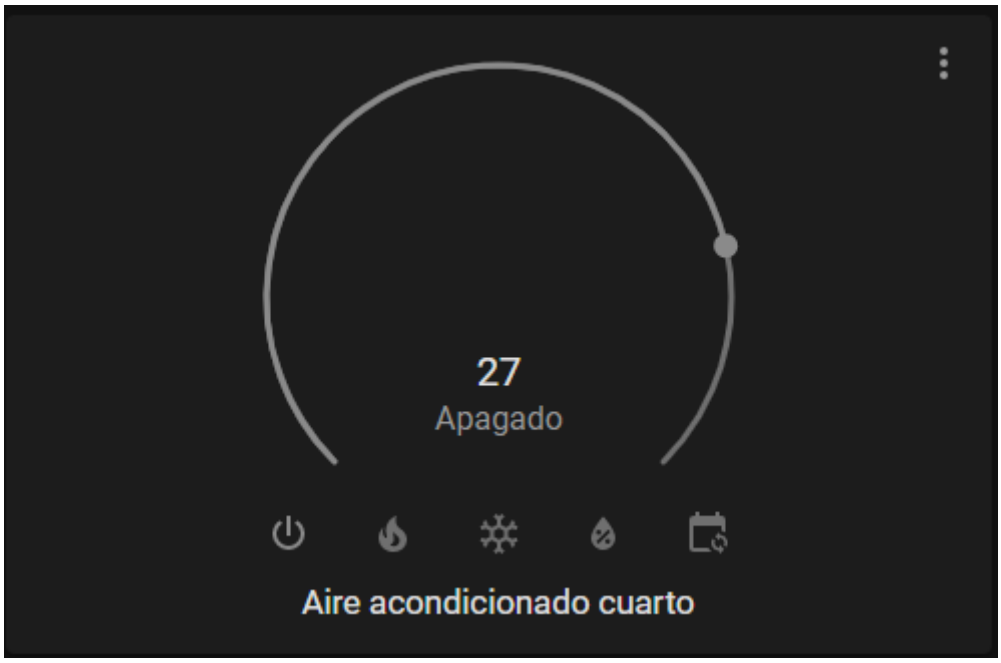

Figura 87.- Control más avanzado del aire acondicionado. Fuente: Elaboración propia.

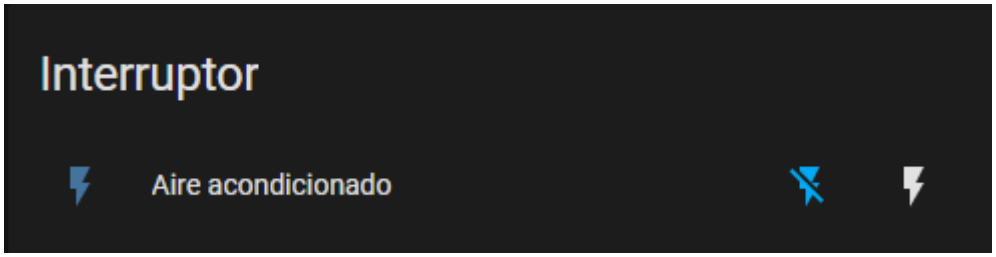

Figura 88.- Control simple del aire acondicionado. Fuente: Elaboración propia.

Una vez finalizados estos pasos, se podrá colocar el dispositivo Sonoff con el emisor de infrarrojos apuntando al aire acondicionado y, mediante la plataforma Home Assistant, conseguir que dicho emisor emita los códigos IR que transmitiría el mando original y que permiten encender, apagar y modificar el comportamiento del electrodoméstico.