Implementación de un plugin para el análisis y control de dispositivos en una casa inteligente
Implementation of a plugin for the analysis and control of IoT devices in a Smart Home

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Sishi Chen
Shangyu Qiu
Xin Xiang Lin Zhou

Director
José Ignacio Requeno
Volker Stolz

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Dedicatoria

A los padres, por su amor incondicional, su apoyo y su infinita paciencia. A mis hermanos, por ser mi fuente de alegría y motivación. A toda mi familia, por su apoyo inquebrantable constante.

A los amigos, gracias por las noches de estudio interminables y por creer en mí cuando yo dudaba. Su amistad ha hecho que este viaje sea inolvidable.

A mis profesores y mentores, por vuestra orientación experta, su estímulo intelectual y su dedicación a mi crecimiento académico. Su sabiduría y consejos han sido fundamentales en mi formación como profesional.

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a todas las personas que colaboraron de alguna manera en el desarrollo y culminación de este trabajo de fin de grado.

En primer lugar, deseamos expresar nuestro agradecimiento a José Ignacio Requeno y Volker Stolz, nuestros tutores de Trabajo de Fin de Grado, por su invaluable orientación, apoyo y paciencia durante todo el proceso de investigación y desarrollo de este proyecto. Su experiencia y dirección fueron fundamentales para el éxito de este trabajo. Agradecemos sinceramente sus enseñanzas, su pasión por el conocimiento y su motivación para seguir aprendiendo.

También queremos agradecer a nuestros familiares y amigos por su constante apoyo, comprensión y aliento durante este desafiante pero gratificante proceso. Sus palabras de ánimo fueron una fuente de inspiración en los momentos más difíciles.

A la Facultad de Informática, queremos agradecer por brindarnos la oportunidad de realizar este Trabajo de Fin de Grado y por permitirnos contribuir con nuestro trabajo a la institución.

A todas las personas que de una forma u otra contribuyeron a la realización de este proyecto, les expresamos nuestro más sincero agradecimiento.

Este Trabajo de Fin de Grado no habría sido posible sin el apoyo y la colaboración de todos ustedes. Gracias por formar parte de este importante logro.

Resumen

Implementación de un plugin para el análisis y control de dispositivos en una casa inteligente

En un mundo donde la tecnología está cada vez más integrada en nuestra vida diaria, vemos claramente cómo la tecnología se está infiltrando en nuestros hogares. Esto requiere soluciones avanzadas para gestionar y analizar los dispositivos inteligentes que nos rodean. Este proyecto proporciona un plugin de Python para **Home Assistant** que se conecta a **TeSSLa**, una plataforma de análisis de datos de series de tiempo en tiempo real.

La importancia práctica de este trabajo radica en su capacidad para mejorar la calidad de vida optimizando la gestión de dispositivos en hogares inteligentes. Le permite tomar decisiones informadas sobre el uso de equipos (como la calefacción) en función de factores como la temperatura y las condiciones climáticas.

El objetivo principal de este proyecto es desarrollar un complemento que facilite la comunicación entre **Home Assistant** y **TeSSLa**, envíe y reciba datos de series temporales e implemente estrategias para optimizar el rendimiento del complemento. La funcionalidad principal de este complemento desarrollado es implementar soporte para la integración de hasta tres flujos de entidades, cubriendo la configuración de **TeSSLa** y el mapeo detallado de flujos de entidades. Admite múltiples integraciones de **TeSSLa**, permitiendo conexiones independientes entre ellas. Admite tipos de datos importantes como Int, String, Float y Boolean. **TeSSLa** desarrollo la integración con servicios AEMET y PVPC, incluyendo conversión detallada de tipos de datos, verificación de errores, sincronización de información y gestión eficiente de flujos de datos externos.

La implementación exitosa de este complemento cumple con los objetivos establecidos y logra una comunicación fluida y eficiente entre **Home Assistant** y **TeSSLa**, sentando las bases para una gestión más inteligente y eficiente de dispositivos en hogares inteligentes.

Palabras clave

Home Assistant, TeSSLa, plugin, AEMET, PVPC, integraciones, sensores, lenguaje de streaming, iot

Abstract

Implementation of a plugin for the analysis and control of IoT devices in a Smart Home

In a world where technology is increasingly integrated into our daily lives, we clearly see how technology is infiltrating our homes. This requires advanced solutions to manage and analyze the smart devices around us. This project provides a Python plugin for **Home Assistant** that connects to **TeSSLa**, a real-time time series data analysis platform.

The practical importance of this work lies in its ability to improve quality of life by optimizing device management in smart homes. It allows you to make informed decisions about the use of equipment (such as heating) based on factors such as temperature and weather conditions.

The main goal of this project is to develop a plugin that facilitates communication between **Home Assistant** and **TeSSLa**, send and receive time series data, and implement strategies to optimize the performance of the plugin. The main functionality of this developed plugin is to implement support for the integration of up to three entity flows, covering **TeSSLa** configuration and detailed entity flow mapping. Supports multiple **TeSSLa** integrations, allowing independent connections between them. Supports important data types like Int, String, Float, and Boolean. **TeSSLa** developed integration with AEMET and PVPC services, including detailed data type conversion, error checking, information synchronization and efficient management of external data flows.

The successful implementation of this plugin meets the established objectives and achieves fluid and efficient communication between **Home Assistant** and **TeSSLa**, laying the foundation for smarter and more efficient management of devices in smart homes.

Keywords

home Assistant, TeSSLa, plugin, AEMET, PVPC, integrations, sensors, streaming language, iot

Índice

1. Introducción **1**
1.1. Motivación 1
1.2. Objetivos 2
1.3. Plan de trabajo 3

2. Estado de la Cuestión: Home Assistant **5**
2.1. Sobre Home Assistant 5
2.2. Sobre plugins de control en Home Assistant 6

3. Estado de la Cuestión: lenguaje de Streaming **11**
3.1. Definición de Streaming 11
3.2. Herramientas y tecnologías que soportan Streaming 12
3.2.1. Apache Flink 12
3.2.2. Apache Spark 13
3.2.3. Kinesis Data Streams 13
3.2.4. Google Cloud Dataflow 14
3.3. Lenguajes de streaming 15

4. Tecnologías **17**
4.1. Python 17
4.2. Docker 17
4.3. GitHub 18
4.4. Jira 19

5. Conceptos previos **21**
5.1. Lenguaje de TeSSLa 21
5.2. Home Assistant 24
5.2.1. Arquitectura general de Home Assistant 24
5.2.2. Maneras de instalación de Home Assistant 27
5.2.3. Cómo se descarga un plugin en Home Assistant 29

6. Plugin en Home Assistant **31**

6.1. Descripción general 31
6.2. Configuración del entorno 31
6.3. Preparación e implementación de HA plugin 32
6.3.1. Verificación de coherencia entre sensores y streams de TeSSLa 33
6.3.2. Actualización del formulario permitiendo introducir tres entidades 39
6.3.3. Soporte de múltiples integraciones de TeSSLa 41
6.3.4. Soporte de multi-idioma 41
6.3.5. Retroalimentación de las integraciones 44
6.3.6. Soporte de diferentes tipos de datos 47
6.4. Ejemplo de recapitulación 59

7. Conclusiones y Trabajo Futuro **67**
7.1. Conclusión 67
7.2. Trabajo futuro 67

Introduction **69**
7.3. Motivation 69
7.4. Goals 70
7.5. Work plan 71

Conclusions **73**
7.6. Conclusion 73
7.7. Future work 73

Contribuciones Personales **75**

Bibliografía **77**

A. Manual sobre instalación de plugin **79**
A.1. Preparación para la instalación 79
A.2. Instalación del plugin 79
A.3. Uso del plugin 80

Índice de figuras

1.1. Diagrama de trabajo de Jira Fuente: Herramienta Jira 4

2.1. Interfaz de Home Assistant Fuente: interfaz de Home Assistant 6

3.1. Aplicación en eventos Fuente: Apache Flink 13
3.2. Principio de funcionamiento de Kinesis Fuente: AWS 14

4.1. Pantalla principal de Docker Fuente: Aplicación de Docker 18

5.1. TeSSLa web IDE Fuente: TeSSLa web IDE 22
5.2. Arquitectura de Home Assistant Fuente: Home Asisstant 26
5.3. Métodos de instalación de HA Fuente: Home Assitant 28
5.4. Tienda de plugin de Home Assistant Fuente: Home Assitant 29

6.1. Ejemplo de temperatura de TeSSLa Fuente: TeSSLa web IDE 32
6.2. Diagrama de output Fuente: TeSSLa web IDE 33
6.3. Mensaje de error mostrado en el formulario en caso de discrepancia Fuente: interfaz de Home Assistant 34
6.4. Notificación del mensaje de error Fuente: interfaz de Home Assistant 35
6.5. Mensaje del error concreto mostrado en notificación Fuente: interfaz de Home Assistant 35
6.6. Error ocurrido por restricción máxima de entities Fuente: interfaz de Home Assistant 36
6.7. Mensaje del error concreto Fuente: interfaz de Home Assistant 37
6.8. Error ocurrido por pasarse de input necesario Fuente: interfaz de Home Assistant 38
6.9. Mensaje mostrado Fuente: interfaz de Home Assistant 39
6.10. Formulario con 3 entidades de input Fuente: interfaz de Home Assistant 40
6.11. Seleccionar para elegir idioma a cambiar Fuente: interfaz de Home Assistant 42
6.12. Versión inglés Fuente: interfaz de Home Assistant 43
6.13. Versión español Fuente: interfaz de Home Assistant 44
6.14. Creación del stream a Fuente: interfaz de Home Assistant 45
6.15. Valor de output a_number Fuente: interfaz de Home Assistant 45

6.16. Creación del sensor b Fuente: interfaz de Home Assistant 46
6.17. Valor output de b_number Fuente: interfaz de Home Assistant 46
6.18. Test entity: Int sensor 1 Fuente: interfaz de Home Assistant 47
6.19. Test entity: Int sensor 2 Fuente: interfaz de Home Assistant 48
6.20. Test entity: int output Fuente: interfaz de Home Assistant 49
6.21. Test entity: Float sensor 1 Fuente: interfaz de Home Assistant 50
6.22. Test entity: Float sensor 2 Fuente: interfaz de Home Assistant 51
6.23. Test entity: Float output Fuente: interfaz de Home Assistant 52
6.24. Test entity: String sensor 1 Fuente: interfaz de Home Assistant 53
6.25. Test entity: String sensor 2 Fuente: interfaz de Home Assistant 54
6.26. Test entity: String output Fuente: interfaz de Home Assistant 55
6.27. Test entity: String on/off senssor 1 Fuente: interfaz de Home Assistant 56
6.28. Test entity: String on/off output Fuente: interfaz de Home Assistant . 56
6.29. Test entity: Unit events sensor 1 Fuente: interfaz de Home Assistant . 57
6.30. Test entity: Unit events output Fuente: interfaz de Home Assistant . . 57
6.31. Test entity: bool sensor Fuente: interfaz de Home Assistant 58
6.32. Test entity: Bool output 1 Fuente: interfaz de Home Assistant 58
6.33. Test entity: Bool output 2 Fuente: interfaz de Home Assistant 59
6.34. Ejemplo de recapitulación en playground Fuente: TeSSLa web IDE . . 61
6.35. Visualización del output Fuente: TeSSLa web IDE 62
6.36. Definición de sensores Fuente: interfaz de Home Assistant 63
6.37. Output del ejemplo V1 Fuente: interfaz de Home Assistant 64
6.38. Definición de sensores V2 Fuente: interfaz de Home Assistant 65
6.39. Output del ejemplo V2 Fuente: interfaz de Home Assistant 66

7.1. Jira work diagram Source: Jira Tool 72

A.1. Crear una nueva rama Fuente: GitHub 79
A.2. Configuración del entorno de desarrollo de HA Fuente: Home Assistant 80
A.3. Página web de Home Assistant Fuente: interfaz de Home Assistant . . 80
A.4. Página web de Home Assistant Fuente: interfaz de Home Assistant . . 81
A.5. Página web de Home Assistant Fuente: interfaz de Home Assistant . . 81
A.6. Página web de Home Assistant Fuente: interfaz de Home Assistant . . 82
A.7. Diálogo de formulario de TeSSLa Fuente: interfaz de Home Assistant 82

Índice de tablas

2.1. Diferencia entre los cinco plugins 9

5.1. Tipos fundamentales y constantes primitivas 23
5.2. Operadores de TeSSLa 23
5.3. Operadores para tipo Bool 23
5.4. Operadores para tipo Int 24
5.5. Operadores para tipo Float 24

Capítulo 1

Introducción

1.1. Motivación

En el mundo actual, la integración de la tecnología en nuestros hogares ha experimentado un cambio revolucionario. Los hogares inteligentes están equipados con una variedad de dispositivos conectados, lo que nos brinda más comodidad, eficiencia e innovación. Sin embargo, a medida que estos sistemas se vuelven cada vez más complejos, se necesitan soluciones avanzadas para analizar y controlar los numerosos dispositivos del hogar inteligente moderno. El crecimiento de estos dispositivos plantea un desafío importante: la necesidad de una plataforma unificada y eficiente para monitorear y evaluar continuamente estos dispositivos. Se propone implementar un plugin de Python como una solución para permitir una integración más fluida entre Home Assistant (Assistant (2023a))y TeSSLa (TeSSLa (2023)). Esta integración brinda la capacidad de enviar datos de series temporales de manera más eficiente y recibir resultados que pueden usarse para mejorar la administración de dispositivos domésticos.

La importancia práctica de esta investigación radica en su capacidad para mejorar significativamente la calidad de vida. En el contexto de la creciente presencia del Internet de las Cosas (IoT) en nuestro entorno, abordar la gestión y análisis eficiente de estos dispositivos no sólo aporta comodidad, sino que también ayuda a mejorar la eficiencia energética, la seguridad y la interoperabilidad de los dispositivos del hogar. **TeSSLa** proporciona una plataforma para integrar y analizar datos de series temporales de manera más eficiente, en lugar de depender únicamente de Python o los lenguajes integrados en **Home Assistant**. Esto no sólo mejora la toma de decisiones sino que también optimiza el funcionamiento de equipos como la calefacción, teniendo en cuenta factores como las previsiones solares y su impacto en la temperatura ambiente. Este programa tiene un impacto directo en la vida diaria y puede optimizar los recursos familiares. Ante las preguntas como: ¿Por qué encender la calefacción ahora si el sol saldrá más tarde y calentará la habitación? **TeSSLa** facilita la integración de datos de manera más efectiva, de manera que el plugin intenta ayudar a solucionar estos problemas con automatizaciones y creando así un hogar más eficiente y cómodo.

Desde que comenzamos nuestras carreras académicas en ingeniería informática,

hemos desarrollado un gran interés en la interacción entre la tecnología y la vida cotidiana. El concepto de hogar inteligente siempre ha sido una fuente de inspiración, especialmente en el contexto del Internet de las cosas. En este marco, nuestra pasión se centra en desarrollar integraciones para asistentes del hogar. La integración de **TeSSLa** recibe mensajes de otros sensores integrados en **Home Assistant**, los analiza y pasa los resultados a **Home Assistant**, permitiendo el control de dispositivos inteligentes. (Kallwies et al. (2022)).

Este proyecto es coherente con nuestras ambiciones e intereses académicos en el Internet de las cosas. Buscamos aplicar nuestros conocimientos teóricos adquiridos en la formación académica a situaciones del mundo real. No es solo un hito académico en nuestras carreras, sino también una oportunidad única para contribuir a los campos de rápido crecimiento de la automatización del hogar y la IoT. Nuestro interés por la innovación tecnológica y nuestro deseo de cambiar la forma en que interactuamos con la tecnología en nuestra vida diaria encajaron perfectamente con los desafíos y posibilidades que presenta este proyecto.

Este trabajo se basa en investigaciones anteriores que destacaron la necesidad de soluciones avanzadas para la gestión y el análisis de dispositivos en hogares inteligentes. A lo largo del proyecto, nuestro enfoque será claro y preciso, centrándonos en el diseño e implementación de complementos funcionales para satisfacer las necesidades específicas de gestión y análisis de dispositivos IoT en hogares inteligentes.

Considerándolo todo, el desarrollo de este plugin de Python es un esfuerzo apasionante para mejorar la integración y gestión de datos en hogares inteligentes. La combinación de tecnologías como **Home Assistant** y **TeSSLa** proporciona soluciones más avanzadas y eficientes para la toma de decisiones basadas en datos de series temporales. Esta iniciativa no sólo refleja nuestra pasión por la tecnología y la innovación, sino que también tiene como objetivo mejorar la experiencia práctica de vivir en un hogar inteligente.

1.2. Objetivos

Como nuestro principal objetivo, creamos un plugin de Python para **Home Assistant** empezando desde el principio para facilitar la comunicación con **TeSSLa**. El plugin debería poder transferir datos de series de tiempo a **TeSSLa** y recibir los resultados producidos por **TeSSLa** para su posterior procesamiento en **Home Assistant**. Con este plugin, nos aseguramos de que los datos enviados y recibidos se interpreten y procesen de manera consistente, lo que permite una comunicación fluida y eficiente entre **TeSSLa** y otros sensores del sistema. Además, implementaremos estrategias y técnicas para optimizar el rendimiento del plugin para garantizar operaciones eficientes y rápidas entre **Home Assistant** y **TeSSLa**. Esto incluye, por ejemplo, la compatibilidad con la integración simultánea de varios integraciones de **TeSSLa** en la plataforma. Todo esto es fundamental para garantizar una experiencia de usuario fluida e ininterrumpida.

Resumiendo todo, consideramos que nuestros objetivos principales son los siguientes:

- Establecer la creación de un plugin para **Home Assistant** que facilite la

comunicación con **TeSSLa**.

- El plugin debe enviar datos de series temporales a **TeSSLa** y recibir los resultados generados para su procesamiento en **Home Assistant**.
- Asegurar que los datos enviados y recibidos se interpreten y permiten comunicación fluida y eficiente entre **TeSSLa** y otros sensores del sistema.
- Implementar estrategias y técnicas para optimizar el rendimiento del plugin, garantizando una ejecución eficiente y rápida de las operaciones entre **Home Assistant** y **TeSSLa**.
- Soportar múltiples integraciones de **TeSSLa** en la plataforma de manera concurrente para asegurar una experiencia de usuario fluida e ininterrumpida.

Posteriormente, prepararemos una documentación detallada que explique el proceso de instalación, configuración y uso del plugin (Manual adjuntado en Apéndice: A). Además, elaboramos un tutorial detallado que servirá como guía para futuros desarrolladores interesados en trabajar con el plugin y comprender su funcionalidad.

1.3. Plan de trabajo

Para el desarrollo de este proyecto se ha seguido una metodología específica, inspirada en los principios Ágiles y facilitada por la herramienta Jira (Jira (2023),) con el objetivo de gestionar de manera eficiente las tareas y fomentar la colaboración entre todos los implicados.

En una primera fase, durante el verano se llevó a cabo un profundo estudio sobre el uso de Python, siendo el lenguaje principal para el desarrollo del posterior trabajo. Además, se profundizó en el conocimiento de dos elementos fundamentales para nuestro caso: **Home Assistant** y **TeSSLa**.

Posteriormente, se procedió a la configuración de los entornos necesarios para el desarrollo del trabajo, aspecto que será abordado en detalle en el capítulo 6.2.

Una vez establecidos los entornos de trabajo, se realizó una formación específica para adquirir los conocimientos necesarios en Python, **TeSSLa** y **Home Assistant**, los cuales son fundamentales para el desarrollo del proyecto. Estos procesos formativos se describirán con detalle en el capítulo 4.

El desarrollo del plugin de **Home Assistant** se llevó a cabo de manera progresiva, enfrentándose a nuevas subtareas y generando nuevas ideas a medida que avanzaba el proceso. Por lo cual esta fase forma la contribución principal al trabajo. Estas tareas se analizarán en profundidad en el capítulo 6.3 destinado al desarrollo del proyecto.

Para garantizar un avance continuo y resolver cualquier duda o problema que surgiera, se establecieron reuniones semanales de aproximadamente una hora y media de duración con los tutores. Estas reuniones se realizan de forma híbrida, tanto presencialmente como a través de la plataforma de Google Meet.

Además, es importante destacar que comparando con el código original de **Home Assistant**, nuestra implementación cuenta con un total aproximado de 200 líneas

de código más y 52 commits en la plataforma de Github. Este aspecto se detallará en el capítulo correspondiente de análisis y resultados.

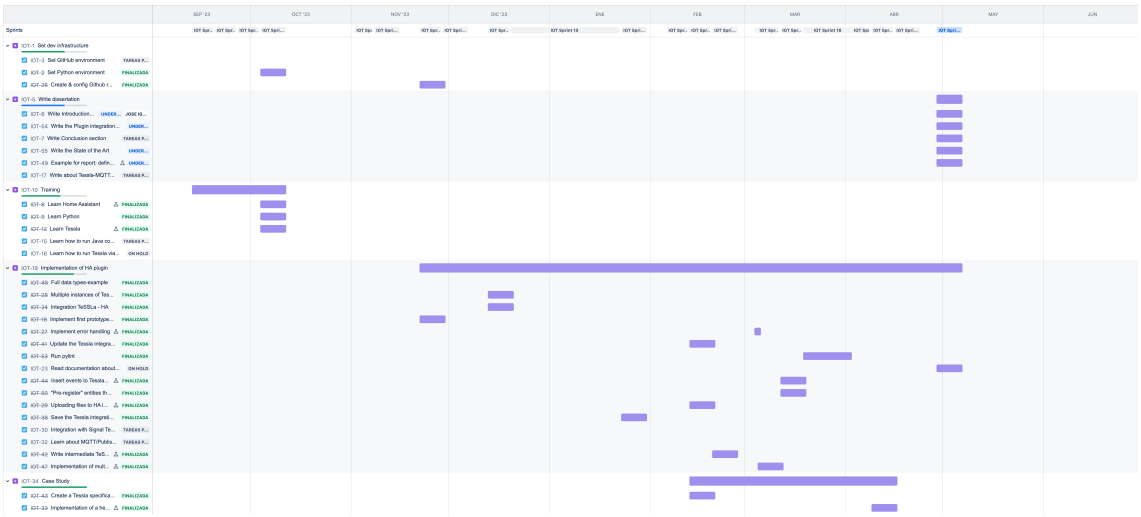

Figura 1.1: Diagrama de trabajo de Jira
Fuente: Herramienta Jira

Capítulo 2

Estado de la Cuestión: Home Assistant

2.1. Sobre Home Assistant

Home Assistant es un software de automatización del hogar gratuito y de código abierto diseñado como un sistema de control central para dispositivos inteligentes, centrándose en el control local y la privacidad. Se puede acceder a él a través de una interfaz de usuario web, aplicaciones de Android e iOS, o mediante comandos de voz a través de un asistente virtual compatible como Google Assistant o Amazon Alexa. Está desarrollada en Python, lenguaje popular por odrecer una interfaz unificada para interactuar con múltiples dispositivos y servicios, razón por la cual la integración en **Home Assistant** se programa utilizando este lenguaje.

Home Assistant cuenta con el soporte de muchos de los principales fabricantes de dispositivos inteligentes. Algunos ejemplos incluyen Xiaomi, Philips, Nest, IKEA, LG y Sonos... Todos estos fabricantes ofrecen integración con **Home Assistant**, permitiendo gestionar sus dispositivos desde este sistema central. **Home Assistant** admite una variedad de dispositivos y tecnologías, desde luces y termostatos hasta cámaras de seguridad y sensores de temperatura. Esto le permite crear un entorno doméstico integrado utilizando dispositivos de diferentes marcas, lo que facilita controlarlos todos desde una sola aplicación sin tener que usar diferentes aplicaciones para cada dispositivo. Además, permite crear automatizaciones que involucran dispositivos de diferentes fabricantes, como un sensor Xiaomi que puede detectar la presencia de una persona y encender una bombilla de Philips Hue.

La interfaz de usuario de **Home Assistant** destaca por su accesibilidad y facilidad de uso. Proporciona a los usuarios un panel intuitivo y personalizable que les permite monitorear y controlar fácilmente sus dispositivos inteligentes. Los usuarios pueden personalizar el panel a su gusto, organizando y resaltando los dispositivos y áreas del hogar que más les importan, lo que resulta en una experiencia personalizada y eficiente. Se puede acceder a la interfaz de usuario a través de cualquier dispositivo con conexión a Internet, ya sea un ordenador, tableta o teléfono inteligente, brindando la flexibilidad y conveniencia de monitorear y controlar el hogar desde cualquier lugar. También proporciona una vista instantánea del estado de todos los dispositivos conectados, lo que permite a los usuarios obtener información instantánea sobre el estado y la condición de su hogar.

Además, **Home Assistant** destaca por su capacidad de realizar automatizaciones avanzadas, permitiendo a los usuarios programar acciones específicas en función de reglas y condiciones predefinidas. Esto está relacionado con el desarrollo de nuevas funciones en nuestro proyecto. Los usuarios pueden crear reglas personalizadas según sus necesidades, desde condiciones y eventos climáticos específicos hasta el estado del dispositivo y la detección de presencia. Para aquellos con conocimientos más avanzados, **Home Assistant** permite la programación de automatización a través de YAML, lo que proporciona personalización y control adicionales.

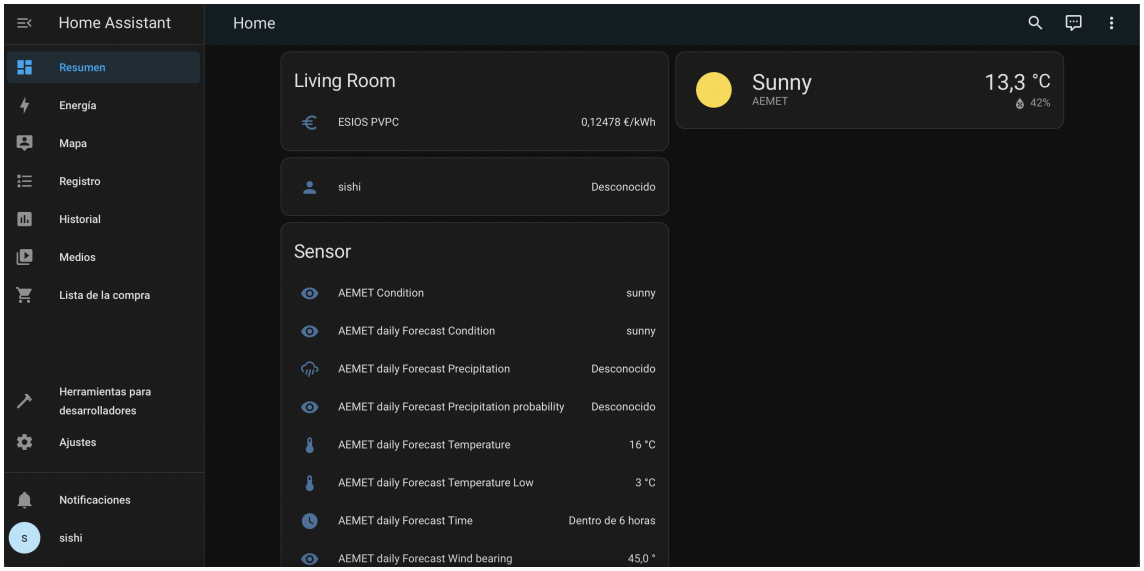

Figura 2.1: Interfaz de Home Assistant
Fuente: interfaz de Home Assistant

2.2. Sobre plugins de control en Home Assistant

En la era de la automatización del hogar, integrar dispositivos y servicios en plataformas centralizadas es fundamental para crear un entorno inteligente y eficiente. En esta sección, se analizará la integración dentro de **Home Assistant** que se pueden utilizar para gestionar dispositivos relacionados con el clima, el consumo de energía y el control de la calefacción. Estas son algunas de las integraciones disponibles:

Node-RED (Sinelec (2023)) es una herramienta de programación visual para dispositivos controladores de hardware. Está diseñado para facilitar la conexión de dispositivos y servicios de IoT a través de la representación visual de relaciones y funcionalidades, permitiendo la programación sin escribir código. La herramienta es compatible con múltiples sistemas operativos, incluidos Windows, Linux y servidores en la nube. Aunque Node-RED fue diseñado originalmente para facilitar la comunicación con hardware, se ha convertido en un estándar de código abierto para el procesamiento de datos en tiempo real. Su enfoque en la programación visual simplifica enormemente el proceso entre la producción y el consumo de información,

facilitando la programación del lado del servidor. Node-RED, un motor de streaming enfocado al Internet de las Cosas (IoT), permite definir flujos de servicios sobre protocolos como MQTT, convirtiéndolo en una herramienta versátil y adaptable para una variedad de aplicaciones. y proyectos.
IoTivity (IoTivity (2023)) es un marco de conectividad de código abierto diseñado para garantizar la interoperabilidad entre dispositivos de IoT. No es tan conocido como otras opciones, ofrece la capacidad de administrar flujos de datos y puede integrarse con **Home Assistant** mediante configuraciones personalizadas. Es una implementación de referencia de código abierto de OCF Secure IP Device Framework. El desafío para el ecosistema de IoT es garantizar que estos dispositivos emergentes puedan conectarse a Internet y entre sí de forma segura y confiable. El proyecto IoTivity se lanzó para unir a la comunidad de código abierto y acelerar el desarrollo de los marcos y servicios necesarios para conectar estos miles de millones de dispositivos.
Apache Kafka (hat (2022), Kafka (2023)) es una plataforma de transmisión de datos descentralizada que no solo le permite publicar, almacenar y procesar transmisiones de eventos instantáneamente, sino también suscribirse a ellos. Está diseñado para gestionar flujos de datos de múltiples fuentes y distribuirlos a diferentes usuarios. En definitiva, puede transmitir grandes cantidades de datos no sólo del punto A al punto B, sino también del punto A al punto Z y cualquier otro destino necesario simultáneamente.
Dark Sky (Apple (2023)) es una plataforma de pronóstico del tiempo que proporciona pronósticos meteorológicos hiperlocales y pronósticos precisos. Destaca por brindar información detallada sobre el clima local. Sin embargo, esta aplicación del tiempo no se puede probar en España porque solo muestra la última previsión meteorológica para Estados Unidos y Reino Unido. Dark Sky se puede integrar en **Home Assistant** para obtener información meteorológica detallada y automatizar acciones en función de las condiciones meteorológicas. Además, hay varias integraciones disponibles en **Home Assistant** para gestionar la energía y el consumo de energía. Estos incluyen monitorear el consumo de energía de los dispositivos conectados, monitorear la producción de energía solar y controlar termostatos y bombas de calor para optimizar la eficiencia energética. Ayudan a controlar el uso de energía, facilitan la transición a una energía sostenible y ahorran dinero. De esta forma, **Home Assistant** puede mostrar cuánta energía consume y produce el usuario, y hacia dónde fluye esa energía. Además, puede rastrear instantáneamente el uso de energía de dispositivos individuales para comprender su impacto en el consumo general.

Cada uno de ellos destaca por sus características y funcionalidades únicas. En el caso de Node-RED, se caracteriza por proporcionar un entorno visual y flexible para conectar dispositivos y servicios de IoT, mientras que IoTivity se centra en garantizar la interoperabilidad entre varios dispositivos. Apache Kafka destaca por sus capacidades de procesamiento de datos en tiempo real, lo que lo convierte en una opción valiosa para gestionar los flujos de información en entornos dinámicos como los hogares inteligentes. Finalmente, Dark Sky es conocido por la precisión de sus pronósticos meteorológicos, proporcionando datos hiperlocales y pronósticos a corto plazo.

En esta comparativa analizaremos aspectos relevantes de estos cuatro plugins

con el objetivo de justificar nuestra elección de **TeSSLa** como el más adecuado. Aquí hay una comparación descriptiva de estos cuatro complementos seleccionados:

1. **Node-RED:** El plugin proporciona una interfaz visual fácil de entender que permite a los usuarios diseñar flujos de trabajo personalizados sin la necesidad de conocimientos avanzados de programación. Ofrece amplio soporte para diferentes dispositivos y protocolos, puede conectarse fácilmente con **Home Assistant** y es gratuito y de código abierto, lo que lo hace popular entre quienes buscan una interfaz fácil de usar respaldada por una comunidad activa.

2. **IoTivity:** El plugin se centra en la interoperabilidad, facilitando la comunicación entre dispositivos de diferentes fabricantes. Es útil en entornos donde hay dispositivos de diferentes marcas y tecnologías. Al igual que Node-RED, es de código abierto, lo que significa que no hay que pagar licencias para utilizarlo.

3. **Apache Kafka:** La estructura adaptativa del plugin le permite gestionar eficientemente grandes cantidades de datos. Al igual que las funciones adicionales mencionadas anteriormente, este también es de código abierto, lo que significa que no hay que pagar ninguna licencia para instalarla.

Tras este análisis comparativo de las ventajas de los diferentes plugins disposibles, hemos seleccionado a **TeSSLa** como nuestro plugin objetivo para desarrollar el posterior proyecto. Esta decisión se basa en las siguientes fortalezas:

- Adaptabilidad: TeSSLa presenta un entorno adaptable y personalizable que se ajusta a las necesidades específicas de cada usuario.

- Compatibilidad: Similar a IoTivity, TeSSLa simplifica la comunicación entre dispositivos de distintos proveedores, asegurando una integración fluida en entornos IoT diversos.

- Análisis de datos: TeSSLa integra capacidades de análisis de datos en tiempo real, posibilitando una evaluación eficaz de la información proveniente de los dispositivos conectados.

Plugin	Funcionalidad principal	Ventajas	Desventajas	Licencia
Node-RED	Flujo de datos visual. Amplia gama de nodos disponibles	Entorno flexible y visual. Amplia comunidad. Altamente personalizable.	Curva de aprendizaje moderada.	Gratuito - Código abierto
IoTivity	Facilita la interoperabilidad. Soporta múltiples protocolos de comunicación. Seguridad integrada.	Facilita la interoperabilidad. Seguridad integrada.	Menos nodos disponibles que Node-RED. Curva de aprendizaje similar a Node-RED.	Gratuito - Código abierto
Apache Kafka	Alta escalabilidad y rendimiento. Soporta múltiples lenguajes de programación.	Altas capacidades de procesamiento de datos en tiempo real. Alta escalabilidad.	Complejidad adicional para implementar en Home Assistant.	Gratuito - Código abierto
TeSSLa	Gestión de dispositivos y servicios IoT. Interoperabilidad entre dispositivos de diferentes proveedores. Procesamiento de datos en tiempo real.	Combinación de las ventajas de los demás plugins. Altamente personalizable. Interfaz gráfica intuitiva.	Comunidad menos extensa que Node-RED o IoTivity.	Gratuito - Código abierto

Tabla 2.1: Diferencia entre los cinco plugins

Capítulo 3

Estado de la Cuestión: lenguaje de Streaming

3.1. Definición de Streaming

Los lenguajes de streaming son idiomas de codificación que le permiten administrar flujos de datos en tiempo real. Están diseñados para procesar información generada continuamente, como eventos de sensores, transacciones financieras o mensajes en redes sociales.

Los lenguajes de streaming tienen las siguientes características:

- Procesamiento en tiempo real: los datos se procesan a medida que se generan, lo que permite una respuesta inmediata a los eventos.
- Escalabilidad: Están diseñados para manejar grandes cantidades de datos de manera eficiente.
- Tolerancia a fallos: Incluso si ocurre un problema con el sistema, continúan funcionando.

Los lenguajes de streaming se pueden utilizar para una variedad de tareas, como:

- Consultar datos en tiempo real: Los lenguajes de streaming se pueden utilizar para analizar información en tiempo real, como datos de sensores o evolución financiera.
- Gestión de eventos: el lenguaje de transmisión se puede utilizar para gestionar eventos en tiempo real, como publicaciones en redes sociales o situaciones de seguridad.
- Creación de sistemas de streaming: los lenguajes de streaming son útiles para crear sistema de streaming, como pipelines de datos o plataformas de análisis en tiempo real.

Beneficios de usar lenguajes de Streaming:

- Reducir el intervalo de tiempo de respuesta: El lenguaje de streaming permite el procesamiento de datos en tiempo real, acortando así el tiempo de respuesta a los eventos.

- Optimizado para la efectividad: los lenguajes de streaming están diseñados para administrar de manera eficiente grandes cantidades de datos.

- Escalabilidad mejorada: el lenguaje de transferencia puede escalarse para manejar grandes cantidades de datos, como en hogares con múltiples dispositivos.

- Alta adaptabilidad: el lenguaje de streaming se puede adaptar a las necesidades específicas del usuario, permitiendo crear una aplicación personalizada para cada situación.

- Comunicación bidireccional con Home Assistant: el lenguaje de streaming admite la interacción bidireccional, no solo recibiendo mensajes del sistema, sino también enviando comandos para controlar o automatizar dispositivos.

3.2. Herramientas y tecnologías que soportan Streaming

3.2.1. Apache Flink

Debido a su amplio conjunto de funciones específicas, Apache Flink es una excelente opción para desarrollar y ejecutar una variedad de aplicaciones. Las características de Flink incluyen soporte continuo y por lotes, gestión de estado avanzada, semántica de procesamiento en tiempo de eventos y garantías únicas de coherencia del estado. Recibe eventos de uno o más flujos de eventos y responde a ellos realizando cálculos, actualizaciones de estado u operaciones externas. Las aplicaciones basadas en eventos representan una evolución del diseño de aplicaciones tradicionales, con jerarquías separadas para el procesamiento y almacenamiento de datos. En esta arquitectura, la aplicación lee datos y los almacena en una base de datos de transacciones remota. Por otro lado, las aplicaciones basadas en eventos son aplicaciones de procesamiento de flujo con estado. En este diseño, los datos y el procesamiento están ubicados conjuntamente, lo que facilita el acceso a los datos locales (ya sea en la memoria o en el disco). La tolerancia a fallos se logra escribiendo periódicamente puntos de control en un almacenamiento persistente remoto. El siguiente diagrama ilustra las diferencias entre la arquitectura de aplicaciones tradicionales y las aplicaciones controladas por eventos.

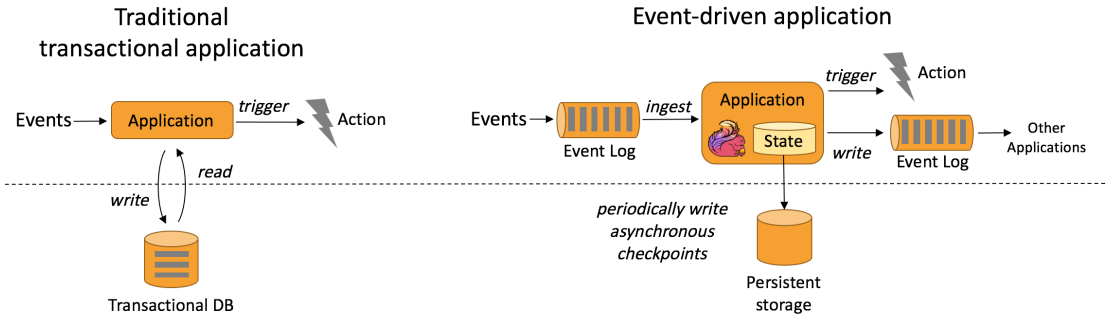

Figura 3.1: Aplicación en eventos
Fuente: Apache Flink

3.2.2. Apache Spark

Apache Spark es un motor multilenguaje para ejecutar trabajos de ingeniería de datos, ciencia de datos y aprendizaje automático en una máquina o clúster de un solo nodo. Sus principales características son las siguientes:

- Procesamiento de datos por lotes/transmisión: unifique el procesamiento de datos por lotes y en tiempo real en el lenguaje de su elección: Python, SQL, Scala, Java o R.

- Análisis con SQL: ejecute consultas ANSI SQL rápidas y descentralizadas para producir paneles e informes ad hoc. Funciona más rápido que la mayoría de los almacenes de datos.

- Ciencia de datos a escala: realice análisis de datos exploratorios (EDA) en conjuntos de datos de petabytes sin muestreo.

- Aprendizaje automático: entrene algoritmos de aprendizaje automático en computadoras portátiles a grupos tolerantes a fallas de miles de máquinas usando el mismo código.

3.2.3. Kinesis Data Streams

Amazon Kinesis Data Streams es un servicio de transmisión sin servidor que simplifica la ingesta, el procesamiento y el almacenamiento de flujos de datos de cualquier tamaño.

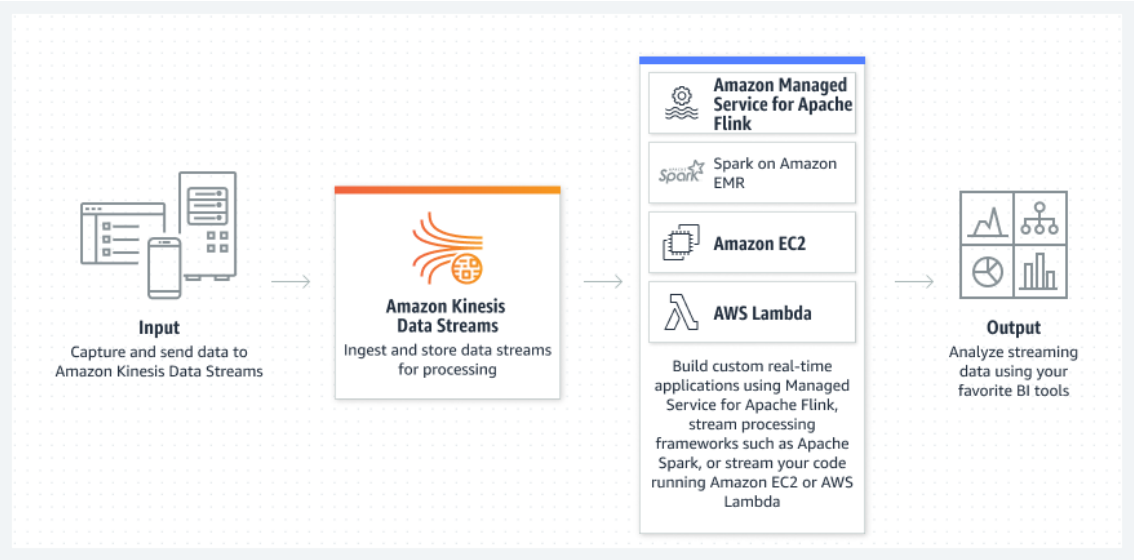

Figura 3.2: Principio de funcionamiento de Kinesis
Fuente: AWS

Entre sus características principales, podemos destacar los siguientes casos de usos:

- Transmitir registros y datos de eventos: Ingresa y recopile datos a nivel de TB de registros de aplicaciones y servicios, datos de flujo de clics, datos de sensores y eventos de usuario dentro de la aplicación todos los días, conduzca el panel de control en tiempo real, genere indicadores y transmita datos al grupo de datos.
- Ejecutar análisis en tiempo real: Utiliza los servicios de alojamiento de AWS Lambda o Amazon para Apache Flink para crear aplicaciones para datos de eventos de alta frecuencia, como datos de flujo de clics, y obtener información en segundos en lugar de días.
- Impulsar aplicaciones basadas en eventos: Independientemente del tamaño, empareja rápidamente con AWS Lambda para responder o ajustar eventos instantáneos en aplicaciones basadas en eventos en el entorno.

3.2.4. Google Cloud Dataflow

Google Cloud Dataflow es un procesamiento de datos de transmisión unificado sin servidor, rápido y rentable y procesamiento de datos por lotes. Comparando con los otros, se puede destacar las siguientes ventajas de Google Cloud Dataflow:

- Análisis rápido de datos de transmisión: El flujo de datos puede lograr un desarrollo de canalizaciones de datos de transmisión rápido y simplificado con una latencia de datos más corta.
- Simplificar la operación y la gestión: El método sin servidor de Dataflow elimina la sobrecarga operativa de las cargas de trabajo de ingeniería de datos, lo

que permite a los equipos centrarse en la programación sin tener que gestionar clústeres de servidores.

- Reducir el coste total de propiedad: La función de expansión automática de recursos y la función de procesamiento por lotes optimizada para el costo permiten a Dataflow proporcionar una capacidad casi ilimitada para gestionar las cargas de trabajo estacionales y máximas sin gastarle en exceso.

3.3. Lenguajes de streaming

Entre todos los lenguajes de streaming, cabe mencionar el lenguaje de programación basado en arquitectura lógica (RT-Lola), un lenguaje de programación de dominio específico (DSL) desarrollado para la especificación y verificación de sistemas integrados en tiempo real. RT-Lola se utiliza para modelar sistemas que operan en entornos críticos donde la velocidad de respuesta y la precisión son fundamentales. Este lenguaje permite a los ingenieros de sistemas describir el comportamiento de un sistema en términos de eventos, condiciones y acciones utilizando una sintaxis basada en la lógica del tiempo. Con RT-Lola, los diseñadores pueden establecer reglas y restricciones sobre cómo responde el sistema a diferentes eventos y estados, lo que facilita la verificación formal del comportamiento del sistema e identifica posibles fallas o violaciones de seguridad. Se utiliza ampliamente en diversos campos, como sistemas de control de vehículos autónomos, sistemas de gestión de tráfico, sistemas de comunicación críticos y equipos de aviónica. Al proporcionar un lenguaje de alto nivel para describir sistemas en tiempo real, RT-Lola ayuda a los ingenieros a desarrollar sistemas más seguros, confiables y eficientes. Por otro lado, se destaca a la lógica temporal de señales (STL), siendo una manera de lógica temporal que se centra en examinar señales reales. STL puede expresar características de cómo evolucionan ciertas propiedades físicas, como la velocidad o la temperatura.

Entre todas estas aplicaciones, hemos decidido usar **TeSSLa** porque ofrece un enfoque más flexible para analizar el funcionamiento de sistemas más allá de la simple medición de métricas de rendimiento. Mientras que otros lenguajes de streaming como **nfer(Kauffman (2021))** y **RT-Lola(Baumeister et al. (2020)**), se especializan en analizar aspectos específicos de las secuencias de datos, como la minería de intervalos temporales o la detección de eventos periódicos en tiempo real. Por otro lado, **TeSSLa** proporciona un marco de trabajo flexible para manejar eventos discretos, con funcionalidades como ventanas de tiempo adaptables y un tratamiento integral del tiempo como un componente fundamental. Además del monitoreo basado en secuencias, también se utiliza un conjunto diverso de técnicas basadas en lógicas temporales en tiempo real, como la Lógica Temporal de Señales **STL(Maler y Nickovic (2004)**). Estos lenguajes ofrecen una forma concisa de describir comportamientos temporales, aunque a menudo se limitan a declaraciones cualitativas, es decir, veredictos booleanos.

Capítulo 4

Tecnologías

4.1. Python

En el contexto de **Home Assistant**, las conexiones desempeñan una función crucial al posibilitar la comunicación y la interacción entre la plataforma y una diversidad de dispositivos y servicios inteligentes. Es relevante resaltar que estas vinculaciones se desarrollan mayormente en Python (John (2019)), un lenguaje de codificación reconocido por su sencillez, versatilidad y amplia aceptación en el ámbito del desarrollo de programas. En nuestro caso, hemos decidido emplear la versión 3 de Python para elaborar estas conexiones. Esta decisión se fundamenta en diversos aspectos, tales como la adaptación a las últimas funcionalidades y bibliotecas, además de la estabilidad y el respaldo continuo proporcionado por la comunidad de programadores. Python 3 nos ofrece una base sólida y fiable para crear conexiones que sean eficaces, seguras y altamente operativas dentro de **Home Assistant**.

4.2. Docker

Docker(Docker (2023)) es una plataforma abierta para crear, publicar y ejecutar aplicaciones. Docker permite separar las aplicaciones de la infraestructura, lo que le permite distribuir software rápidamente. Con Docker, puede administrar su infraestructura al igual que sus aplicaciones. Si sigue el enfoque de Docker para enviar, probar e implementar código, puede reducir drásticamente el tiempo entre escribir su código y ponerlo en uso. Proporciona la capacidad de empaquetar y ejecutar aplicaciones en entornos poco aislados llamados contenedores. El aislamiento y la seguridad permiten que varios contenedores se ejecuten simultáneamente en el mismo servidor. Los contenedores son ligeros y contienen todo lo necesario para ejecutar una aplicación sin depender de lo que esté instalado en el servidor. Puede compartir contenedores mientras trabaja y asegurarse de que todos los usuarios que reciben el contenedor tengan el mismo contenedor que opera de la misma manera. Además, la plataforma basada en contenedores de Docker permite una amplia portabilidad de cargas de trabajo. Los contenedores Docker pueden ejecutarse en la computadora portátil de un desarrollador, una máquina física o virtual en un centro de datos, un

proveedor de nube o una combinación de entornos. La portabilidad y la naturaleza ligera de Docker también facilitan la administración dinámica de cargas de trabajo, escalando aplicaciones y servicios casi instantáneamente según las necesidades comerciales. Por estos motivos, decidimos utilizar Docker para implementar **Home Assistant**. Creemos que es más fácil configurar un entorno HA que una instalación nativa en nuestra máquina. Docker proporciona un entorno aislado y consistente para ejecutar aplicaciones, lo que simplifica enormemente el proceso de configuración y administración de **Home Assistant**. Además, al utilizar contenedores Docker, garantizamos una mayor coherencia entre los entornos de desarrollo y producción.

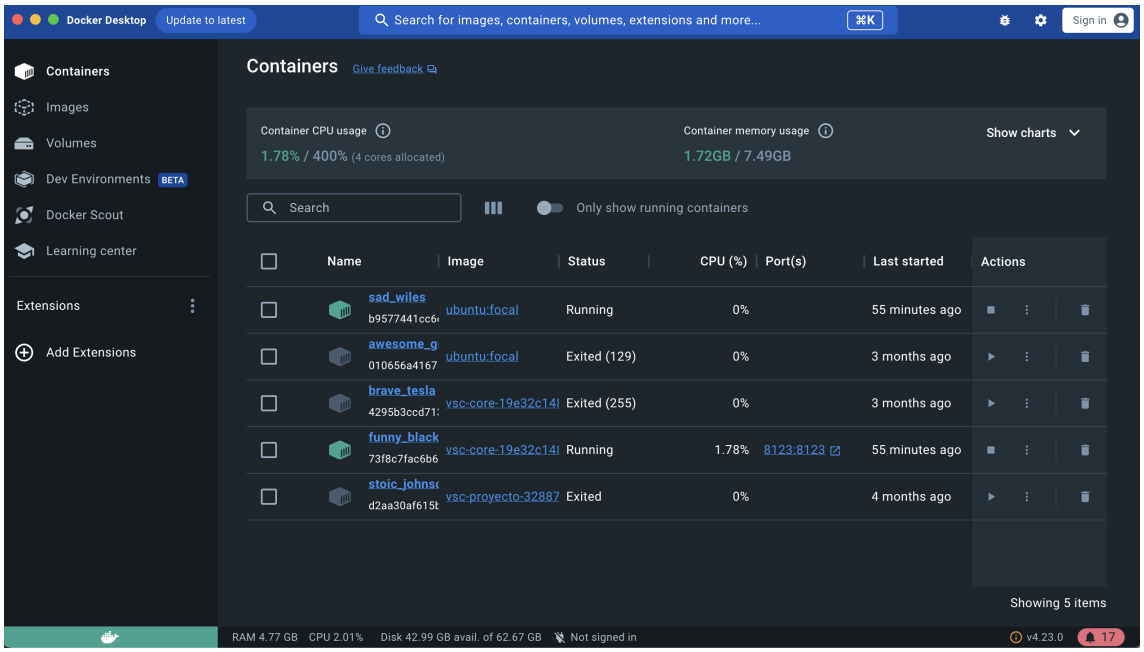

Figura 4.1: Pantalla principal de Docker
Fuente: Aplicación de Docker

4.3. GitHub

Se ha utilizado la plataforma de Github(GitHub (2023))para facilitar la escritura de código y transmisión de estos mismos. Permite compartir el código almacenado y trabajar en proyectos de equipo. El uso constante de esta herramienta se da principalmente en la revisión y corrección de los códigos. Se puede encontrar el repositorio en: https://github.com/XinXiangLinZhou/tessla.

Cabe mencionar también el uso de Visual Studio Code como herramienta del entorno de desarrollo integrado (IDE) desarrollado por Microsoft, un editor de código fuente que facilita el desarrollo de software con sus características avanzadas y soporte para extensiones.

4.4. Jira

Jira (2023)Jira es una herramienta flexible de gestión de proyectos que se adapta a cualquier metodología ágil, como Scrum y Kanban. Se ha empleado esta herramienta para la organización de sprints y la coordinación de reuniones de planificación de sprints, donde se decide qué tareas debe completar el equipo en el siguiente sprint a partir del backlog, que es la lista de tareas pendientes, permitiendo estimar historias, ajustar el alcance de los sprints, verificar la velocidad del equipo y reordenar la prioridad de las tareas en tiempo real.

Capítulo 5

Conceptos previos

5.1. Lenguaje de TeSSLa

TeSSLa (TeSSLa (2022)) es un lenguaje de transformación de flujos diseñado para diversos propósitos. **TeSSLa** en sí no es un lenguaje de programación, sino un lenguaje de especificación basado en flujos. Es particularmente adecuado para describir el sistema de una manera sencilla y proporcionar una visión diferente y posiblemente más abstracta del sistema que la implementación real.

No obstante, **TeSSLa** aún puede ser utilizado por programadores sin experiencia en lógica matemática. Con **TeSSLa**, los algoritmos de monitoreo se pueden describir directamente. Esto no sólo facilita a los ingenieros de software el uso de **TeSSLa** en la práctica, sino que también pueden usarlo para filtrar eventos y realizar análisis cuantitativos y estadísticos, así como sesiones de depuración sobre la marcha.

Está diseñado específicamente para examinar y examinar la dinámica de los sistemas ciberfísicos, con especial atención a la crítica temporal. **TeSSLa** se creó específicamente para la verificación del tiempo de ejecución de transmisiones (SRV) y está equipado con herramientas que cumplen con este objetivo principal. El soporte integrado para eventos con marca de tiempo hace que **TeSSLa** sea particularmente hábil en el manejo de flujos caracterizados por la dispersión y el grano fino.

- Verificación en Tiempo de Ejecución de Flujos (SRV): combina elementos del procesamiento de eventos complejos (CEP) y los enfoques tradicionales de verificación en tiempo de ejecución (RV). En **TeSSLa**, un flujo se somete a un proceso de transformación que no produce una única conclusión concluyente, sino que produce un flujo de salida continuo que encapsula las propiedades evaluadas para cada cambio en el flujo de entrada. **TeSSLa** se adapta bien a los estilos de programación declarativa y es muy adecuado para expresar propiedades y especificaciones de corrección.

- El tiempo como elemento crítico: **TeSSLa** no requiere eventos simultáneos para todas las transmisiones, lo que significa que puede manejar transmisiones dispersas y de alta frecuencia. En **TeSSLa**, cada evento tiene una marca de tiempo a la que se puede acceder mediante operadores de tiempo. El uso de marcas de tiempo de eventos tiene un doble propósito: le permite acceder al

orden global de eventos comparando marcas de tiempo y facilita la realización de cálculos basados en esas marcas de tiempo.

- Arquitectura Modular: **TeSSLa** conecta un conjunto de flujos de entrada a otro conjunto de flujos de salida correspondientes a través de ecuaciones que se referencian mutuamente, lo que permite hacer referencia a estados anteriores. Al admitir ecuaciones recursivas, **TeSSLa** puede expresar varias funciones utilizando solo un conjunto de operadores básicos. También contiene un sistema avanzado de tipos y macros, que incluye bibliotecas especializadas para diferentes dominios.

- Procesamiento paralelo eficiente: el diseño de **TeSSLa** sigue dos principios clave para promover el procesamiento paralelo eficiente: gestión de memoria explícita y composición de operadores locales. Al procesar flujos que contienen datos de tamaño fijo, los operadores de **TeSSLa** solo requieren memoria limitada porque cada operador debe almacenar como máximo un valor de datos. Este enfoque de diseño permite la implementación en sistemas con acceso limitado a la memoria, como FPGA o sistemas integrados.

Para comprender mejor, hemos procedido a realizar algunos casos de estudio de **TeSSLa** utilizando TeSSLa web IDE:

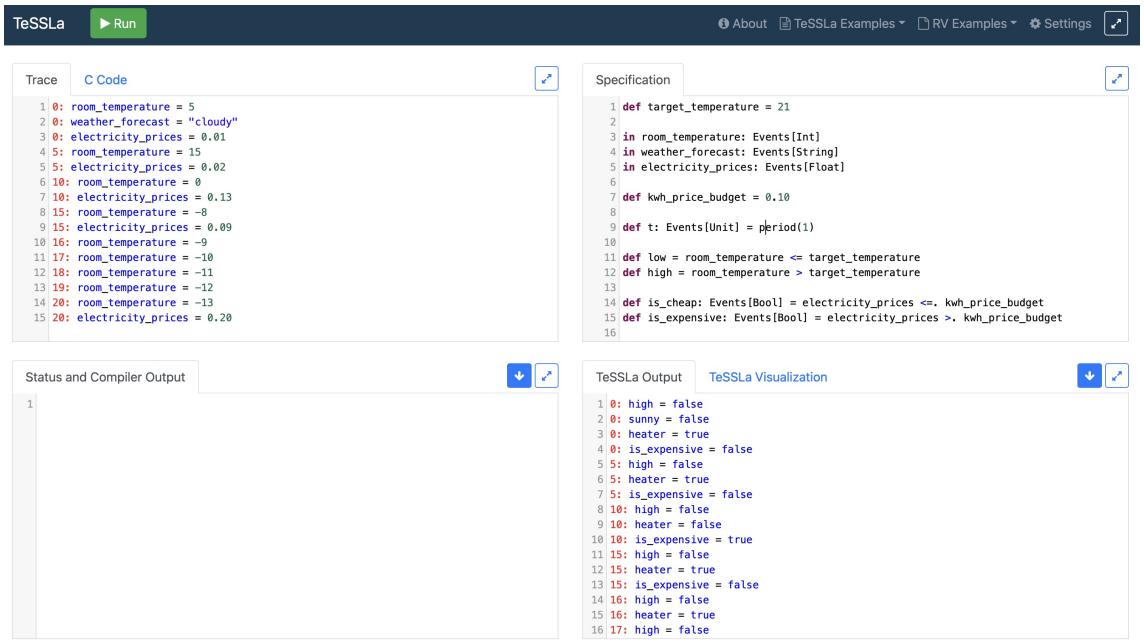

Figura 5.1: TeSSLa web IDE
Fuente: TeSSLa web IDE

Donde nos permite visualizar cómo el código de **TeSSLa** genera trazas de outputs. En el panel arriba derecho se encuentra el código de `specification`. En la columna de la izquierda tenemos la secuencia de trazas ordenadas por timestamp (marca de tiempo). En la imagen de abajo a la derecha tenemos una ilustración con

los eventos que genera **TeSSLa** como salida. **TeSSLa** permite procesar trazas de ejecución o código anotado (p.ej: código C).

En cuanto al lenguaje de **TeSSLa** (Tessla (2023)), la nomenclatura más usada tiene esta forma: *Events[T]*, donde T es un tipo de valor, tanto valores primitivos: `int, bool, string, float, Unit`, como otros tipos más complejos. Entre estos, se destaca el tipo `Unit` por tener un único valor ().

Tipo	Significado
Bool	un Boolean puede ser true o false
Events	tipo básico para streams
Float	TeSSLa utiliza un número de punto flotante IEEE-754 de 64 bits
Int	Los valores máximos y mínimos dependen del motor de TeSSLa
String	puede almacenar secuencias de caracteres arbitrariamente largas
Unit	() es el único valor del tipo Unit, se utiliza para representar eventos sin valores
Anotación @raw	especifica que output flujo de salida debe imprimirse en un formato sin procesar, solo contenga el valor del evento sin marca de tiempo o nombre de flujo
False	representa que una proposición dada es falsa
True	representa que una proposición dada es verdadera

Tabla 5.1: Tipos fundamentales y constantes primitivas

Para los tipos primitivos, el lenguaje de **TeSSLa** admite muchas operaciones lógicas usadas como en los otros lenguajes de programación pero con unos pequeños matices:

Operador	Significado
==	Devuelve True si los dos tipos de valores son iguales
!=	Devuelve True si los dos tipos de valores son diferentes

Tabla 5.2: Operadores de TeSSLa

Operadores para tipo `Bool`:

Operador	Significado
&&	"y" lógico
||	"o" lógico
!	negación

Tabla 5.3: Operadores para tipo Bool

Operadores para tipo `Int`:

Los operadores de tipo `Float` son muy similares a los de `Int`, con una sola diferencia en representación de sintaxis para diferenciarlos, que consiste en añadir

Operador	Significado
<	Menor que (devuelve bool)
>	Mayor que (devuelve bool)
<=	Menor igual que (devuelve bool)
>=	Mayor igual que (devuelve bool)
+	Suma
-	Resta
*	Multiplicación
/	División

Tabla 5.4: Operadores para tipo Int

un `.` como sufijo antes de los operadores. A través de esto, permite la inferencia automática de los tipos de datos.

Operador	Significado
.<	Menor que (devuelve bool)
.>	Mayor que (devuelve bool)
.<=	Menor igual que (devuelve bool)
.>=	Mayor igual que (devuelve bool)
.+	Suma
.-	Resta
.*	Multiplicación
./	División

Tabla 5.5: Operadores para tipo Float

Además, para diferenciar entre estos dos tipos, se usa 1 como `Int`, y 1.0 en caso de `Float`.

5.2. Home Assistant

5.2.1. Arquitectura general de Home Assistant

La arquitectura de **Home Assistant** se basa en tres partes principalmente:

- Sistema operativo (SO): proporciona una plataforma Linux eficiente creada específicamente para alojar **Home Assistant**. Su función principal es gestionar los recursos de hardware como la unidad central de procesamiento (CPU), la memoria y el almacenamiento. Como analogía, podemos pensar en esta parte del sistema operativo como la base de una casa: es la base sobre la que se construye todo lo demás.

- Supervisor (home assistant (2023)): administrar los sistemas operativos para garantizar un funcionamiento óptimo. Supervise el funcionamiento de Home Assistant Core y las integraciones de terceros. Administre las actualizaciones,

la seguridad y las copias de seguridad de Home Assistant Core. En comparación, podríamos decir que este rol es similar al de un administrador del hogar, responsable de mantener todo actualizado y seguro, y facilitar los cambios o ajustes necesarios.

- Core: la parte principal de **Home Assistant** que interactúa con los usuarios, supervisores y dispositivos y servicios disponibles. Se encarga de realizar la automatización y procesar datos de sensores y dispositivos. Podemos equiparar esta función al cerebro de la casa, encargado de procesar la información y tomar decisiones.

En términos más simples, el sistema operativo establece la base, el Supervisor mantiene las cosas en marcha y Core es el centro central que interactúa con los usuarios y con tu sistema de Home Assistant.

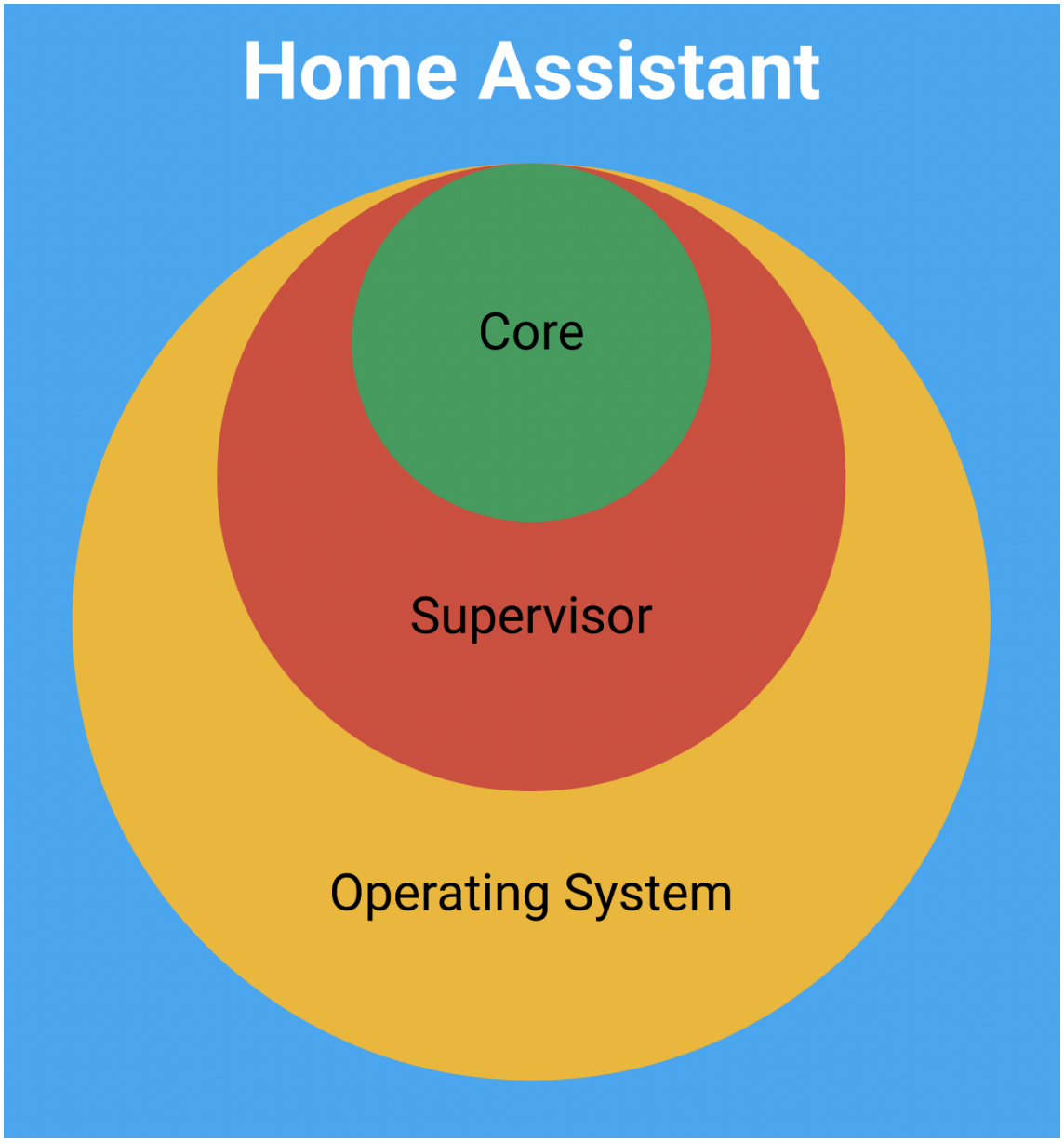

Figura 5.2: Arquitectura de Home Assistant
Fuente: Home Asisstant

Además de sus funcionalidades actuales, **Home Assistant** también ofrece la opción de ampliar su funcionalidad mediante el uso de integraciones. Estas integraciones (Assistant (2023)) son responsables de detectar o desencadenar eventos, proporcionar servicios y mantener el estado. Son el vínculo que conecta a **Home Assistant** con el resto del mundo, permitiendo que el sistema se comunique con una variedad de dispositivos inteligentes, servicios en la nube y sistemas locales, abriendo una variedad de posibilidades para automatizar y controlar su hogar. Por ejemplo, en nuestro caso utilizamos extensiones como AEMET y PVPC...

Las entidades sirven como vínculo entre el mundo físico de los dispositivos y sensores y el mundo digital de las interfaces de usuario y la automatización de los asistentes domésticos. Estos son los componentes básicos que componen la interfaz

de usuario y automatización de **Home Assistant**.

Las entidades son principalmente de este tipo:

- Sensores: Son similares a los ojos y oídos de un asistente doméstico. Son entidades que recopilan datos inmediatos sobre el entorno, como temperatura, tiempo, probabilidad de lluvia, velocidad del viento, etc... y los convierten en información valiosa para la automatización de dispositivos y la personalización del espacio.

  - `sensor.aemet_temperature`
  - `sensor.aemet_daily_forecast_condition`
  - `sensor.aemet_rain_probability`
  - `sensor.aemet_daily_forecast_wind_speed`

Según el estado de la entidad, se pueden establecer reglas para habilitar o deshabilitar acciones específicas. Las entidades pueden reaccionar a las acciones resultantes. Por ejemplo, podría construir una automatización que encienda la calefacción de su hogar si `sensor.aemet_temperature` cae por debajo de cierto nivel. Estos resultados se pueden mostrar en paneles y vistas dentro de la interfaz de usuario de **Home Assistant**, lo que le permite ver el estado actual de cada entidad, monitorear sus valores y administrar su configuración.

5.2.2. Maneras de instalación de Home Assistant

En esta sección vamos a hablar de las diferentes maneras de instalación de **Home Assistant** (Assistant (2022)). **Home Assistant** ofrece cuatro maneras diferentes de instalación:

1. Sistema Operativo Home Assistant: Esta opción es la más recomendada ya que proporciona una experiencia optimizada y simplificada sin necesidad de instalar manualmente el sistema operativo y configurar sus dependencias.

2. Home Assistant Supervisor: proporciona un entorno completo de Home Assistant, que incluye Home Assistant Core y otras herramientas como Docker y Supervisor. Se ejecuta en un sistema operativo básico (generalmente una distribución de Linux) y utiliza Docker para administrar el contenedor Home Assistant y otras dependencias. Su función principal es la gestión de contenedores, permitiendo la instalación de complementos y la gestión del sistema a través de la interfaz web de Home Assistant.

3. Home Assistant Container: Esta es una instalación independiente basada en contenedores de Home Assistant Core. Proporciona un entorno similar a Home Assistant Core, pero utiliza contenedores Docker para ejecutar Home Assistant y sus dependencias. Sin embargo, no incluye un Supervisor para gestionar contenedores ni permite la instalación de complementos para ampliar la funcionalidad de Home Assistant.

4. Home Assistant Core: es una instalación manual utilizando un entorno virtual Python. Básicamente, es un programa Python que puede ejecutarse en diferentes sistemas operativos. Es por el cual hemos optado en nuestro proyecto ya que brinda la capacidad de monitorear, controlar y automatizar equipos.

   Home Assistant incluye Home Assistant Core y herramientas para facilitar su instalación en dispositivos como la Raspberry Pi y otras plataformas sin configuración previa del sistema operativo. Se trata de una solución completa que incluye una interfaz de usuario para la gestión, accesible desde el front-end de Home Assistant, aunque esta característica no está disponible en la configuración base de Home Assistant Core. Sin embargo, al igual que Supervisor, esta opción no incluye otras herramientas como Docker o Supervisor, ni permite la instalación de complementos.

Vemos ahora una tabla de comparación entre estas diferentes maneras de instalación:

	HA OS[1]	Container[1]	Core[1]	Supervised[1]
Automations	✅	✅	✅	✅
Dashboards	✅	✅	✅	✅
Integrations	✅	✅	✅	✅
Blueprints	✅	✅	✅	✅
Uses container	✅	✅	❌	✅
Supervisor	✅	❌	❌	✅
Add-ons	✅	❌	❌	✅
Backups	✅	✅[2]	✅[2]	✅
Managed Restore	✅	❌[3]	❌[3]	✅
Managed OS	✅	❌	❌	❌

Figura 5.3: Métodos de instalación de HA
Fuente: Home Assitant

5.2.3. Cómo se descarga un plugin en Home Assistant

Tenemos principalmente dos maneras para descargar plugin en Home Assistant, la primera es mediante la tienda disponible de Add-ons en los métodos de instalación disponibles (Assistant (2023b)):

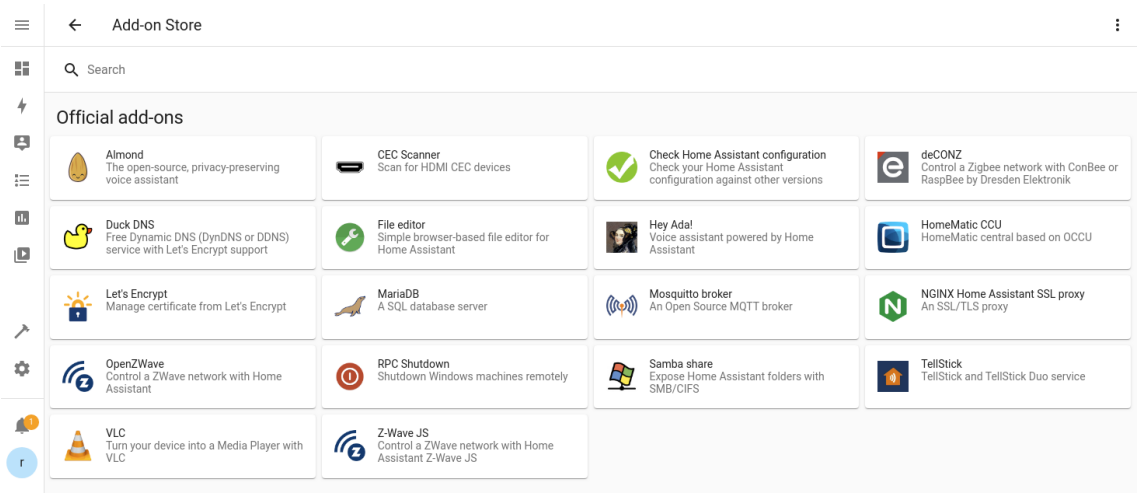

Figura 5.4: Tienda de plugin de Home Assistant
Fuente: Home Assitant

Ésta es una tienda de plugins integrada que permite instalar y administrar plugins fácilmente directamente desde la interfaz web de Home Assistant. Para instalar plugins:

- Acceda al panel de Configuración->Plugins en la pestaña principal de la interfaz de **Home Assistant**.
- Haga click en Tienda de complementos, todos los plugins incluyen una documentación correpondiente.
- Buscar plugins: navegar por los plugins disponibles por categoría o usar la barra de búsqueda para encontrar un plugin específico por nombre.
- Instalar un plugin: haz clic en el botón de Instalar una vez encontrado el plugin que desee. **Home Assistant** descargará e instalará el plugin automáticamente.

Algunos plugins más avanzados solo serán visibles en Modo avanzado, que se puede cambiar en la página de perfil de usuario. Haga clic en un complemento que le interese, para leer la documentación o para instalar el plugin. En esta forma de instalación, solo se podrá utilizar plugins disponibles en la tienda, para más funcionalidades adicionales se realizará por la otra forma de descarga. Sin embargo, la descarga de plugin por este método resulta ser más fácil y también el Supervisor comprueba e instala automáticamente las actualizaciones disponibles.

La segunda forma de descargar plugin (por el cual hemos optado en nuestro caso, ya que por la elección de descarga de Core no disponemos de Supervisor en la interfaz de **Home Assistant**) ya es para usuarios más avanzados o plugins no disponibles en

la tienda del complementos. En este segundo caso se descarga el plugin manualmente, descargando los archivos del plugin y colocándolos en el directorio correspondiente de Home Assistant.

Para esto, hay que seguir los siguientes pasos:

- Descargamos los archivos del plugin necesarios, generalmente un archivo zip de código Python.
- Acceder al sistema de archivos de Home Assistant.
- Colocar la carpeta de archivos de plugin descargados en el directorio correspondiente de Home Assistant, generalmente `/config/custom_components` o `/components`.
- Una vez que los archivos del plugin estén en su lugar, reinicia el sistema de Home Assistant para que los cambios se hagan efectos.

En esta forma de descarga, nos permite obtener un control total sobre la instalación y la configuración, de manera que nos ofrece la posibilidad de personalizar nuestro plugin con más libertad.

Capítulo 6

Plugin en Home Assistant

6.1. Descripción general

En este apartado se detalla las características principales desarrolladas en este trabajo, que serán explicadas a continuación con más detalle en el apartado de Preparación e implementación de HA plugin.

1. Implementación de soporte para hasta tres entidades-stream, abordando aspectos como la configuración de TeSSLa, el mapeo detallado de las entidades-stream, así como la implementación de un riguroso proceso de detección y corrección de errores.

2. Soporte de múltiples integraciones de TeSSLa de manera que permite conectarse a diferentes integraciones sin que una afecte a la otra.

3. Soportamos todos los tipos más importantes como `int` (sensor.random), `string` (sensor.aemet), `float` (sensor.aemet), `boolean` (sensor.random_sensor).

4. Desarrollo e integración de TeSSLa con los servicios de AEMET (Agencia Estatal de Meteorología) y PVPC (Precio Voluntario para el Pequeño Consumidor), incluyendo tareas como la conversión exhaustiva de tipos de datos, chequeo de errores, sincronización de información y la gestión eficiente de los flujos de datos provenientes de estas fuentes externas.

6.2. Configuración del entorno

Esta sección detalla el proceso de configuración del entorno de desarrollo necesario para implementar el proyecto.

Para garantizar un entorno de trabajo eficiente y coherente, hemos implementado una serie de configuraciones. En primer lugar, se configura la infraestructura necesaria para el desarrollo del proyecto, incluyendo la preparación y ajuste de los sistemas y herramientas necesarios. Dentro de este marco, se establece el entorno Python necesario para el desarrollo y ejecución del proyecto, lo que implica la instalación y

configuración de bibliotecas y dependencias específicas necesarias. Adopta un enfoque sistemático para garantizar la coherencia y compatibilidad entre las diferentes bibliotecas utilizadas en el entorno Python de un proyecto.

Además, se creó un repositorio dedicado en la plataforma GitHub para almacenar y gestionar el código fuente del proyecto. El repositorio sirve como una plataforma centralizada para colaborar, compartir y rastrear cambios en el código base. Se establecen prácticas de control de versiones para promover la colaboración y flujos de trabajo de desarrollo eficientes. Presta atención al detalle y está diseñado para optimizar la productividad y eficiencia de los equipos de desarrollo. Estas configuraciones proporcionan una base sólida para el avance eficiente y organizado del proyecto.

A través de este repositorio, el estado de nuestro código de implementación se actualiza diariamente, lo que refleja el progreso continuo de nuestro trabajo.

6.3. Preparación e implementación de HA plugin

Durante esta fase, se llevó a cabo la preparación y la implementación del plugin para **Home Assistant**. Se realizaron una serie de pasos fundamentales para asegurar una integración efectiva y coherente del sistema **TeSSLa** en el entorno de **Home Assistant**.
En primer lugar, se procedió a la integración de **TeSSLa**, un componente imprescindible para el funcionamiento del plugin. Se realizó una fase inicial de prueba utilizando un ejemplo sencillo de temperatura para simular el comportamiento de **TeSSLa**, y también nos permite visualizar el output como muestra la figura 6.1. Esto permitió verificar la viabilidad y funcionalidad de la integración antes de proceder con la implementación completa y también nos ayuda a entender sobre qué se basa **TeSSLa**.

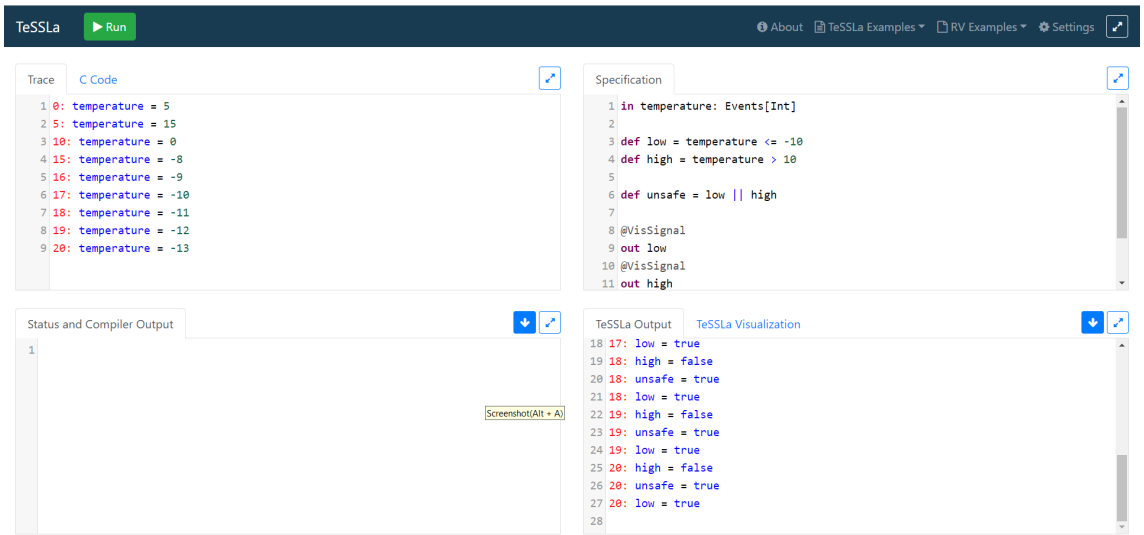

Figura 6.1: Ejemplo de temperatura de TeSSLa
Fuente: TeSSLa web IDE

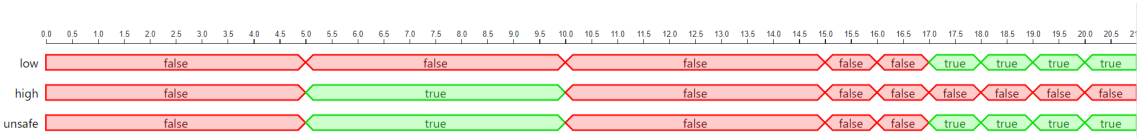

Figura 6.2: Diagrama de output
Fuente: TeSSLa web IDE

Para la fase de preparación, se desarrolló también un prototipo inicial del plugin para **Home Assistant** utilizando Java y Popen. Esta implementación proporcionó una base para la construcción posterior. Para ello se llevó a cabo un estudio sobre Popen, se exploraron sus características y funcionalidades para comprender su uso efectivo dentro del contexto del proyecto.

6.3.1. Verificación de coherencia entre sensores y streams de TeSSLa

Se implementó código para verificar la consistencia de datos entre los sensores que establecemos en `specification.tessla` y los streams de entity de la entrada introducidos a **TeSSLa**. En caso de discrepancia entre ambos, se configuró un mensaje de error en el panel de notificaciones (Figura: 6.4) de **Home Assistant** para alertar al usuario sobre la incompatibilidad y facilitar la resolución del problema. En caso contrario, procedemos a la carga de datos correctamente.

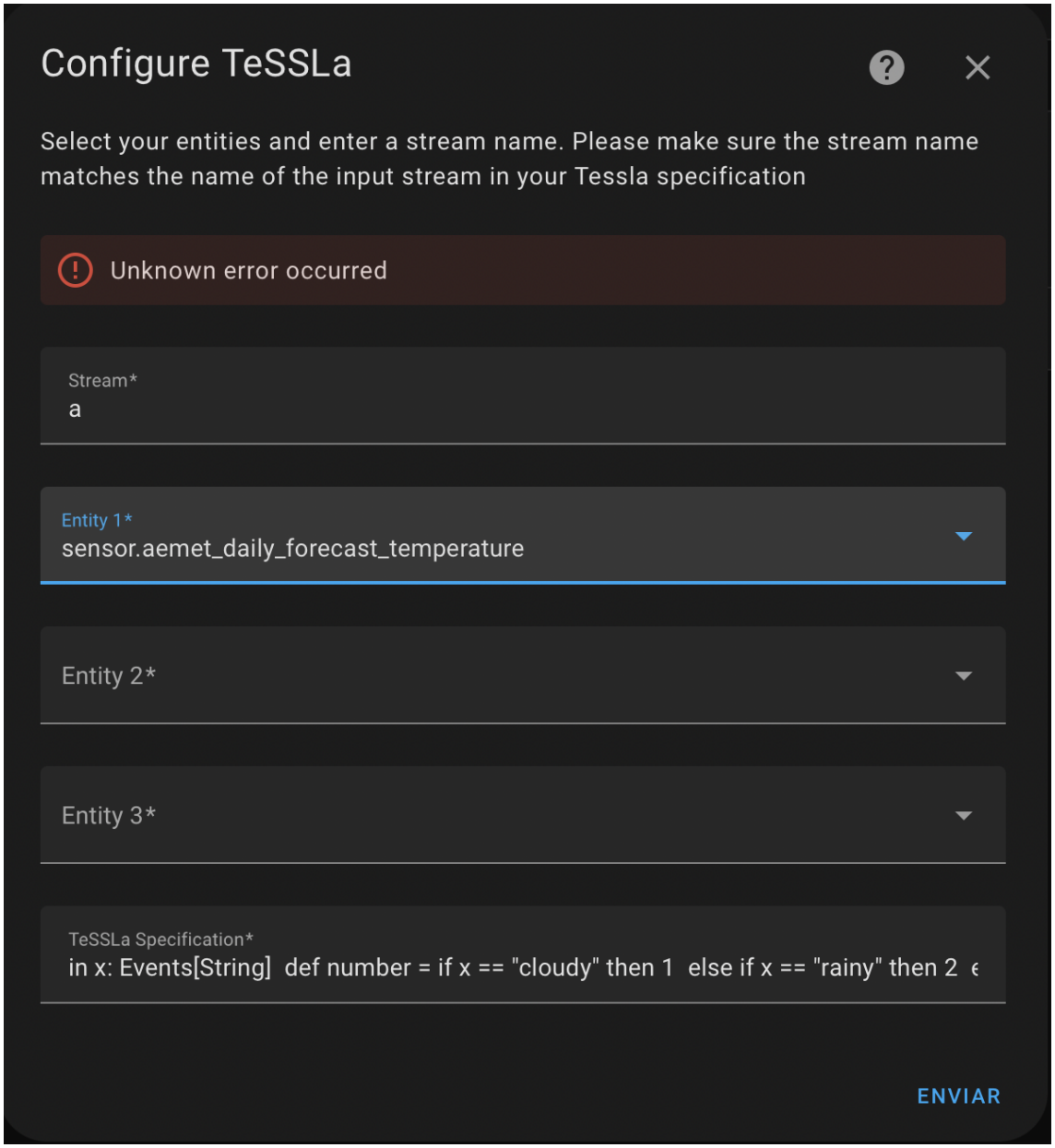

Figura 6.3: Mensaje de error mostrado en el formulario en caso de discrepancia
Fuente: interfaz de Home Assistant

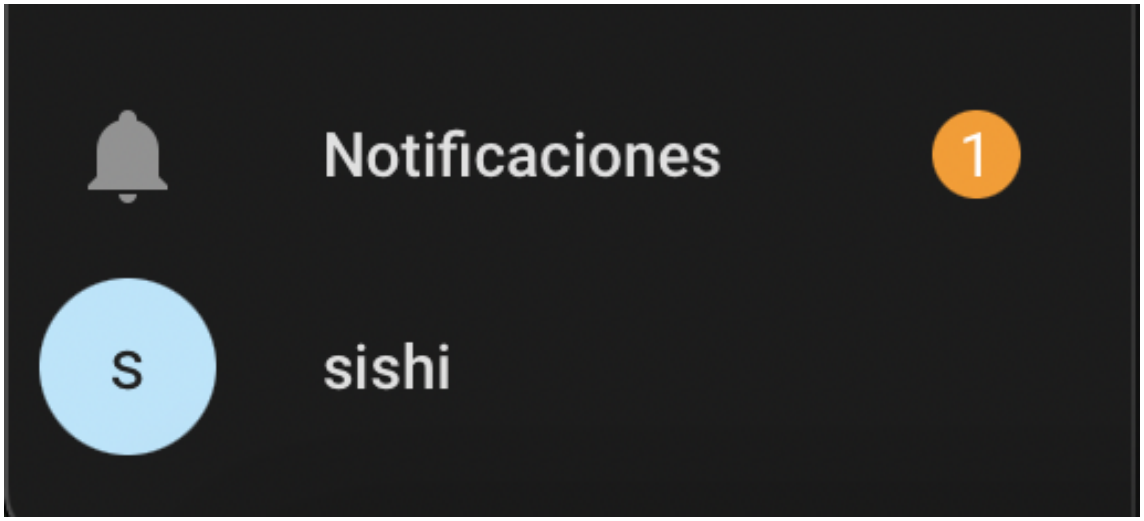

Figura 6.4: Notificación del mensaje de error
Fuente: interfaz de Home Assistant

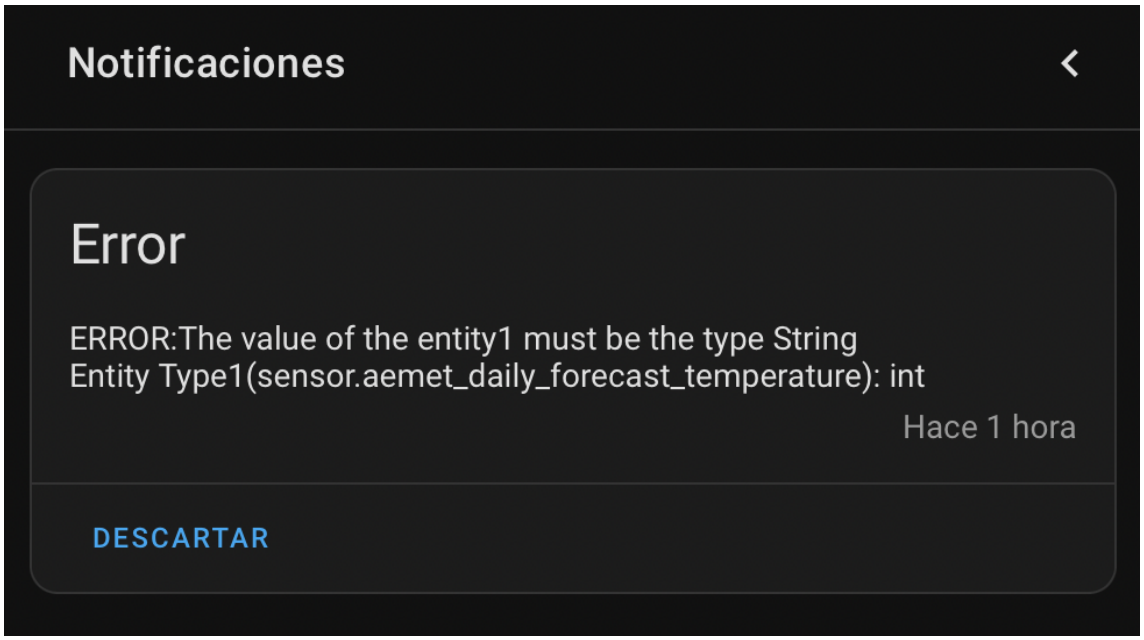

Figura 6.5: Mensaje del error concreto mostrado en notificación
Fuente: interfaz de Home Assistant

Aparte de la concordancia de los tipos de datos de entrada, se ha implementado funcionalidades como comprobar el número de input stream es <= 3, ya que en el formulario actualizado, permitimos introducir como máximo 3 entradas. En caso contrario, se indicará una alerta como en el caso anterior al usuario como se muestra en la imagen siguiente.

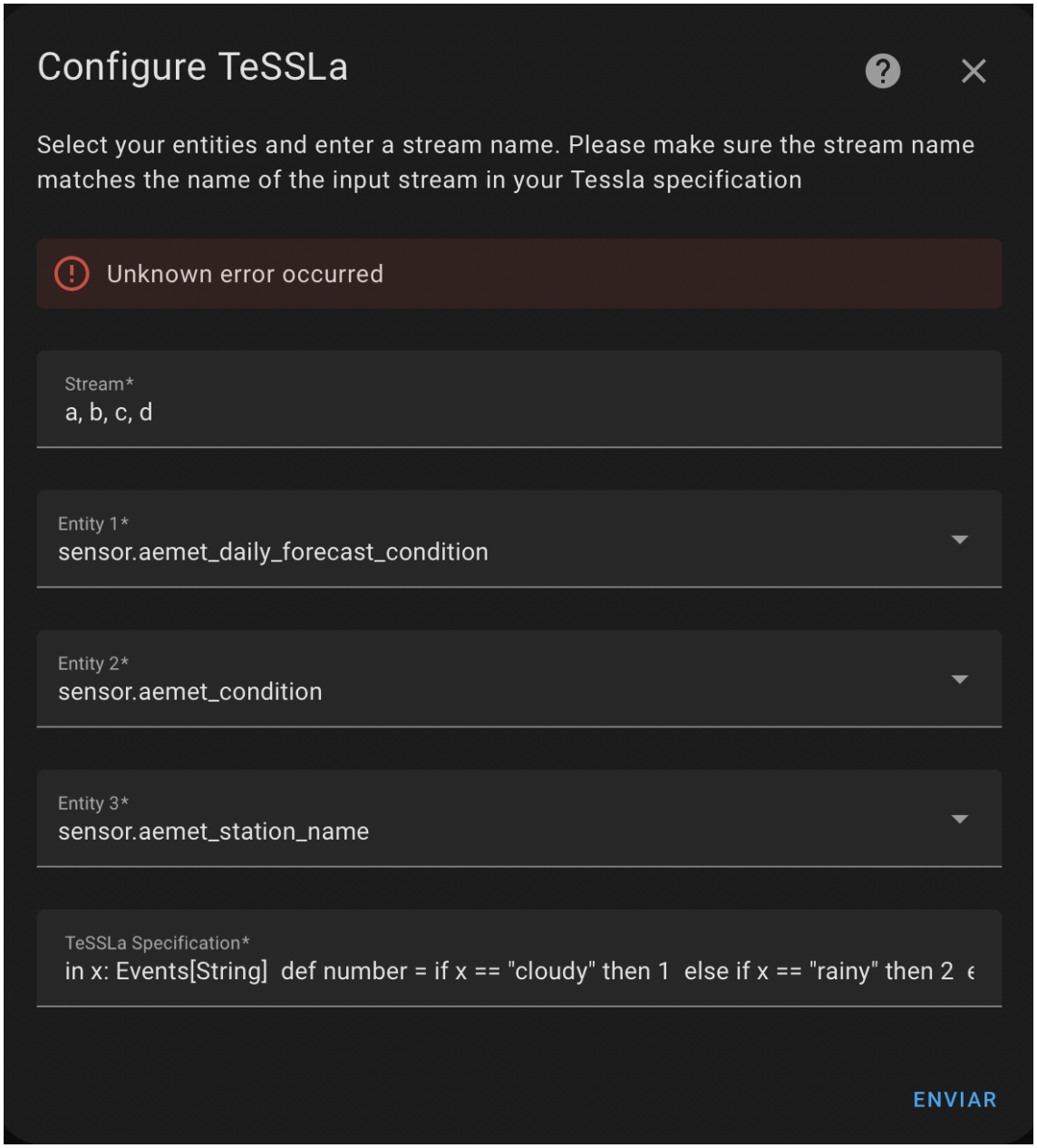

Figura 6.6: Error ocurrido por restricción máxima de entities
Fuente: interfaz de Home Assistant

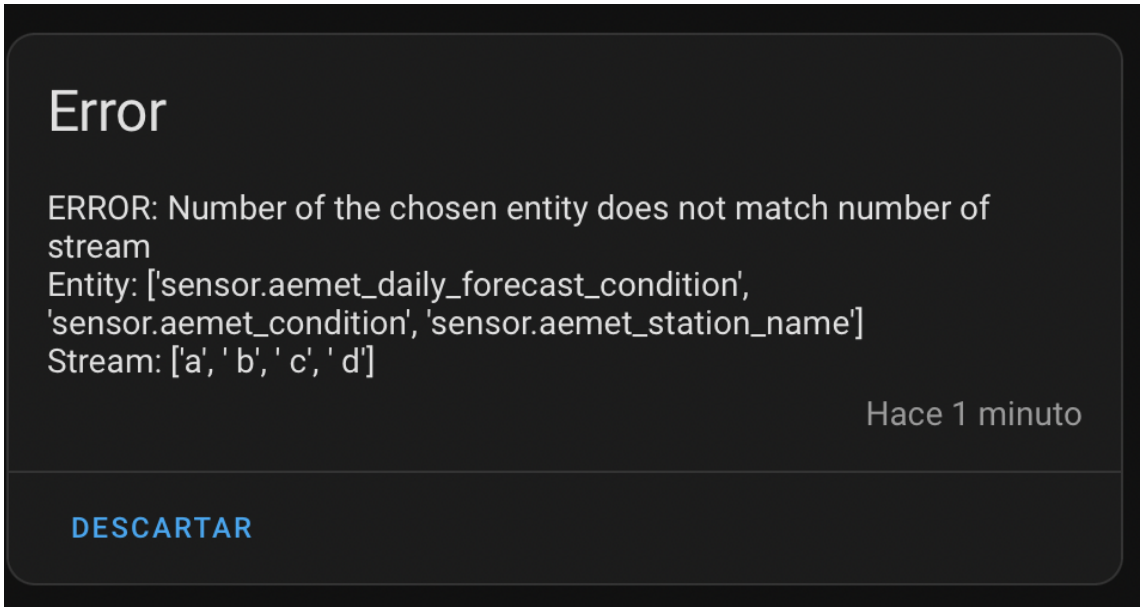

Figura 6.7: Mensaje del error concreto Fuente: interfaz de Home Assistant

También cabe mencionar que existe otra comprobación de alertas de error que nos permite garantizar que el número de los entities establecidos como entrada coincide exactamente con el número de input necesitado por `specification.tessla`. En otro caso, en la pestaña de notificaciones se mostrará otro tipo de alerta como muestra la figura 6.8.

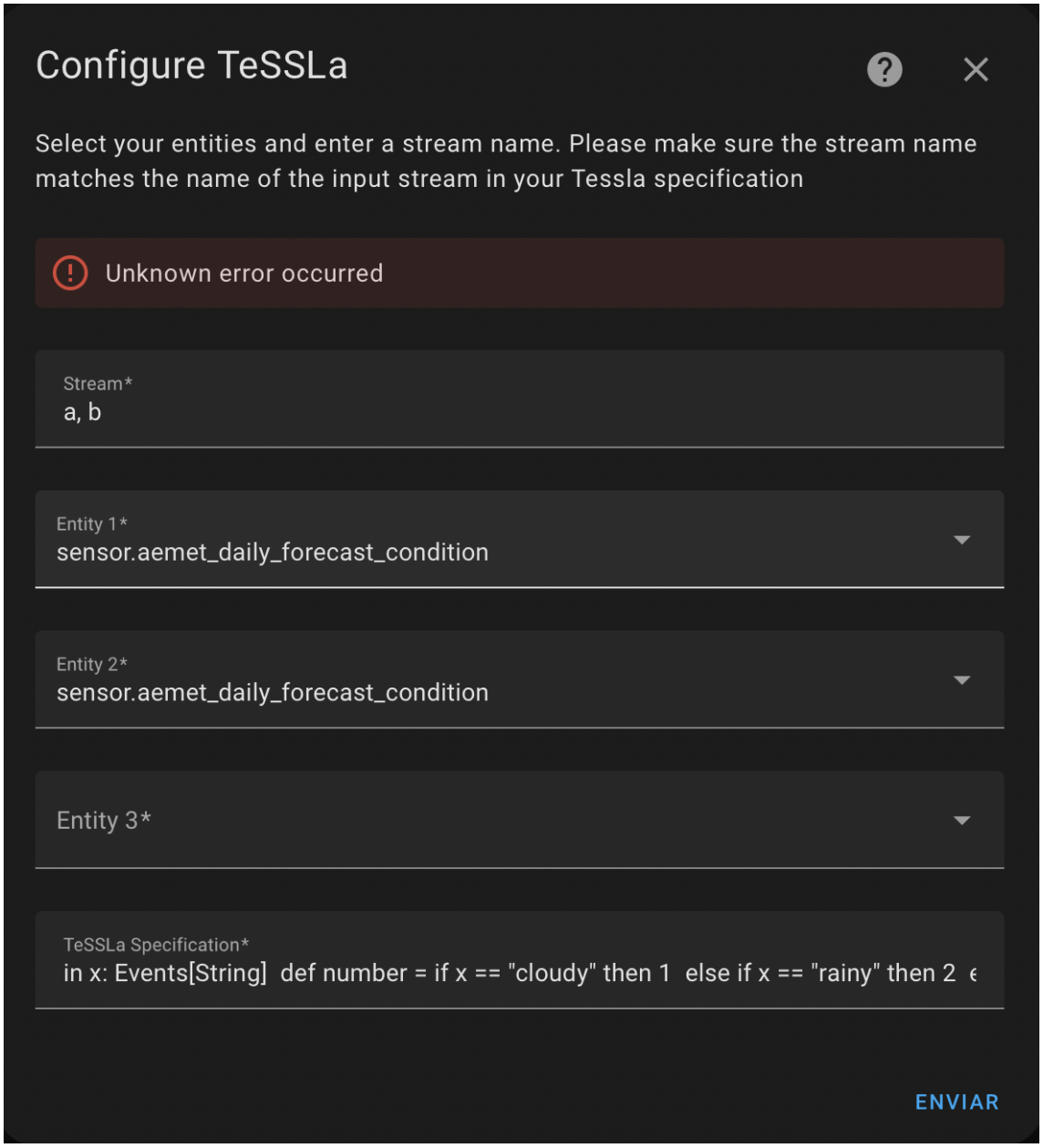

Figura 6.8: Error ocurrido por pasarse de input necesario
Fuente: interfaz de Home Assistant

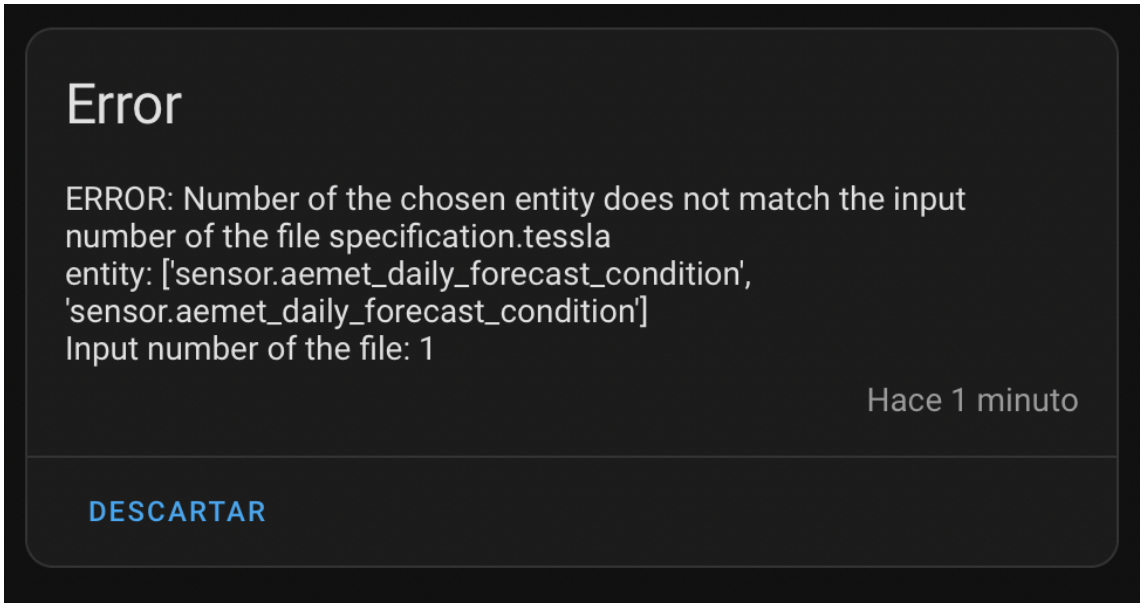

Figura 6.9: Mensaje mostrado
Fuente: interfaz de Home Assistant

Vemos en detalle del error en la notificación del mensaje, ya que en el formulario introducimos dos entradas, sin embargo en la `specification` solo recibe una entrada de input.

6.3.2. Actualización del formulario permitiendo introducir tres entidades

Como se mencionó anteriormente, hemos agregado hasta 3 entidades en el formulario de integración de **TeSSLa**. Estas entidades se asocian de manera secuencial a los streams de entrada de **TeSSLa**, permitiendo un control eficiente de múltiples tipos de sensores para el plugin de AEMET como por ejemplo. Se mostrarán alertas en caso de errores.

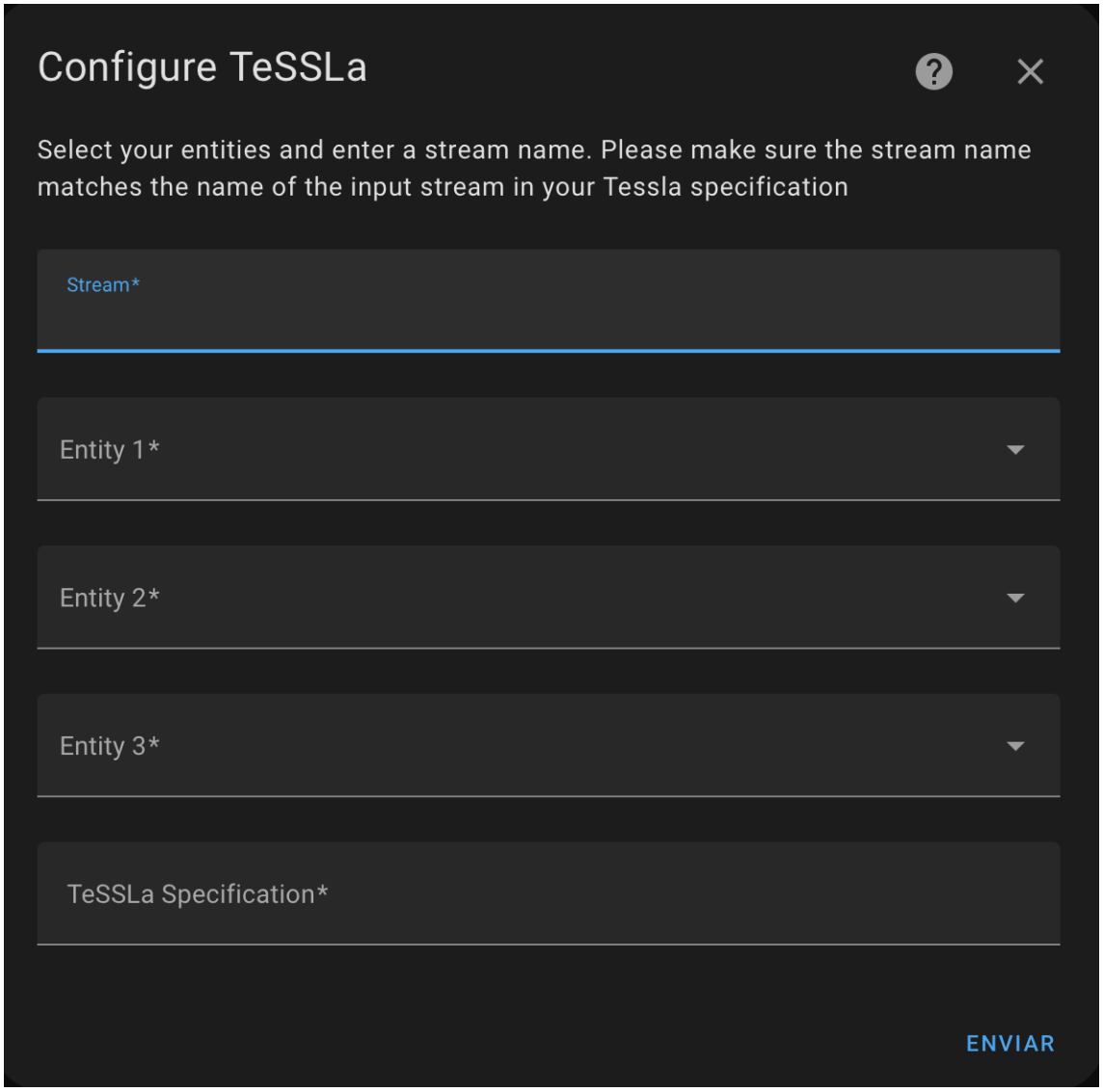

Figura 6.10: Formulario con 3 entidades de input
Fuente: interfaz de Home Assistant

A parte de esto, hemos intentado agregar la funcionalidad de file upload para subir `specification` en el envío del formulario, pero debido a problemas de integración al final hemos decidido en introducir las especificaciones manualmente para el envío de formulario.

La razón por la cual se ha establecido un límite de hasta tres integraciones se basa en la necesidad de gestionar eficientemente el precio de la electricidad, la temperatura y el tiempo. Además, esta decisión está basa en ejemplos análogos observados en sistemas de **Home Assistant**, donde se integran sensores de temperatura y humedad, iluminación inteligente y cámaras de seguridad.

6.3.3. Soporte de múltiples integraciones de TeSSLa

En este apartado, se centra en la contribución basada en desarrollo de múltiples integraciones en **TeSSLa**.

Para ello, se ha desarrollado una nueva funcionalidad que aborda la necesidad de gestionar información de configuración en **Home Assistant**. En este sentido, se ha introducido el uso de un archivo temporal `tempfile`, que permite almacenar información relevante sobre la configuración en **Home Assistant**, así como la información de `specification.tessla`. Asegurando que la información se mantenga incluso después de reiniciar el sistema. Esto se logró aprovechando la capacidad de **Home Assistant** para guardar automáticamente la información almacenada en el objeto `configflow`, garantizando una experiencia sin interrupciones para el usuario. Éste actúa como un repositorio temporal para la especificación de **TeSSLa**. En lugar de escribir directamente en el directorio de configuración de **Home Assistant**, donde podría haber problemas de colisión o conflicto en lectura de datos cuando tenemos múltiples instancias de integraciones de **TeSSLa**, la especificación de **TeSSLa** se almacena en este archivo temporal. Después de que **TeSSLa** lea la primera trace, este fichero permanecerá abierto, posteriormente se produce la eliminación automática del archivo temporal. Para el próximo inicio de **Home Assistant**, se creará otro fichero temporal automáticamente. Esto garantiza un manejo más seguro y eficiente de la información crítica de configuración.

La adopción de un archivo temporal proporciona una solución flexible y escalable para gestionar múltiples integraciones de **TeSSLa**. Al mantener la especificación de **TeSSLa** en un archivo separado y temporal, se facilita la gestión de la configuración, permitiendo así usar diferentes integraciones sin causar problemas de colisión de lectura de archivos.

6.3.4. Soporte de multi-idioma

Cabe mencionar que con la posibilidad que ofrece **Home Assistant** sobre el cambio de país, lo cual permite un cambio de idioma dependiendo de la región elegida. De esta manera **Home Assistant** también cabe la posibilidad de traducir automáticamente los mensajes de error, formularios de texto... a otros idiomas. De manera que **Home Assistant** se adapta automáticamente al lenguaje de entorno y traduce los mensajes dependiendo del lenguaje del entorno.

Para cambiar el idioma de **Home Assistant**, se puede realizarlo desde perfil de cada uno y seleccionando el cambio de idioma:

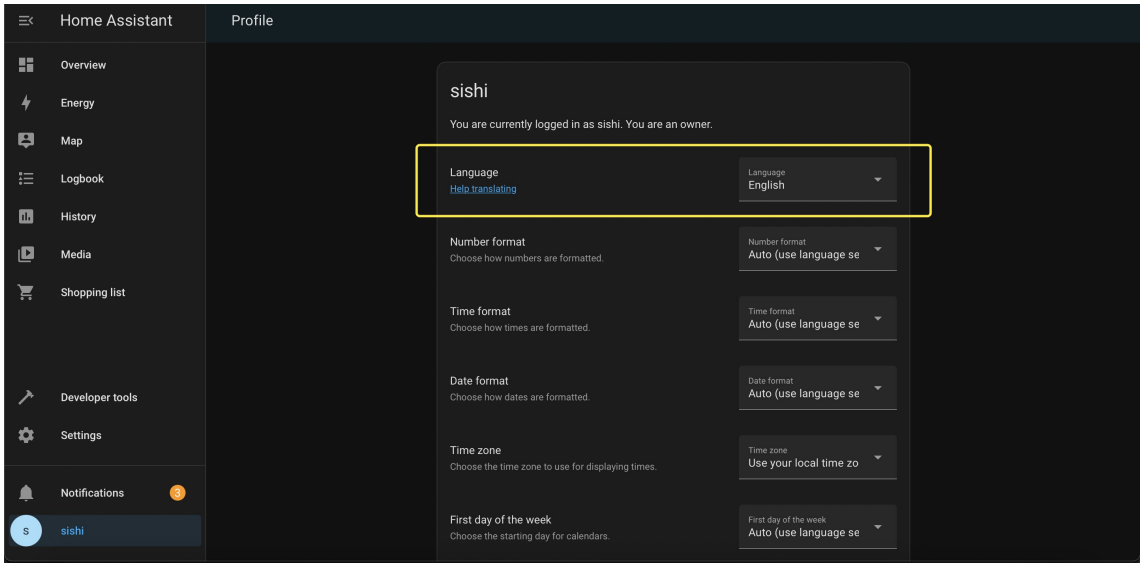

Figura 6.11: Seleccionar para elegir idioma a cambiar
Fuente: interfaz de Home Assistant

Actualmente disponemos de la traducción automática entre español e inglés, de manera que al seleccionar el idioma se mostrará el formulario con el lenguaje correspondiente. En la siguiente imagen, vemos cómo se muestra la imagen en versión inglés:

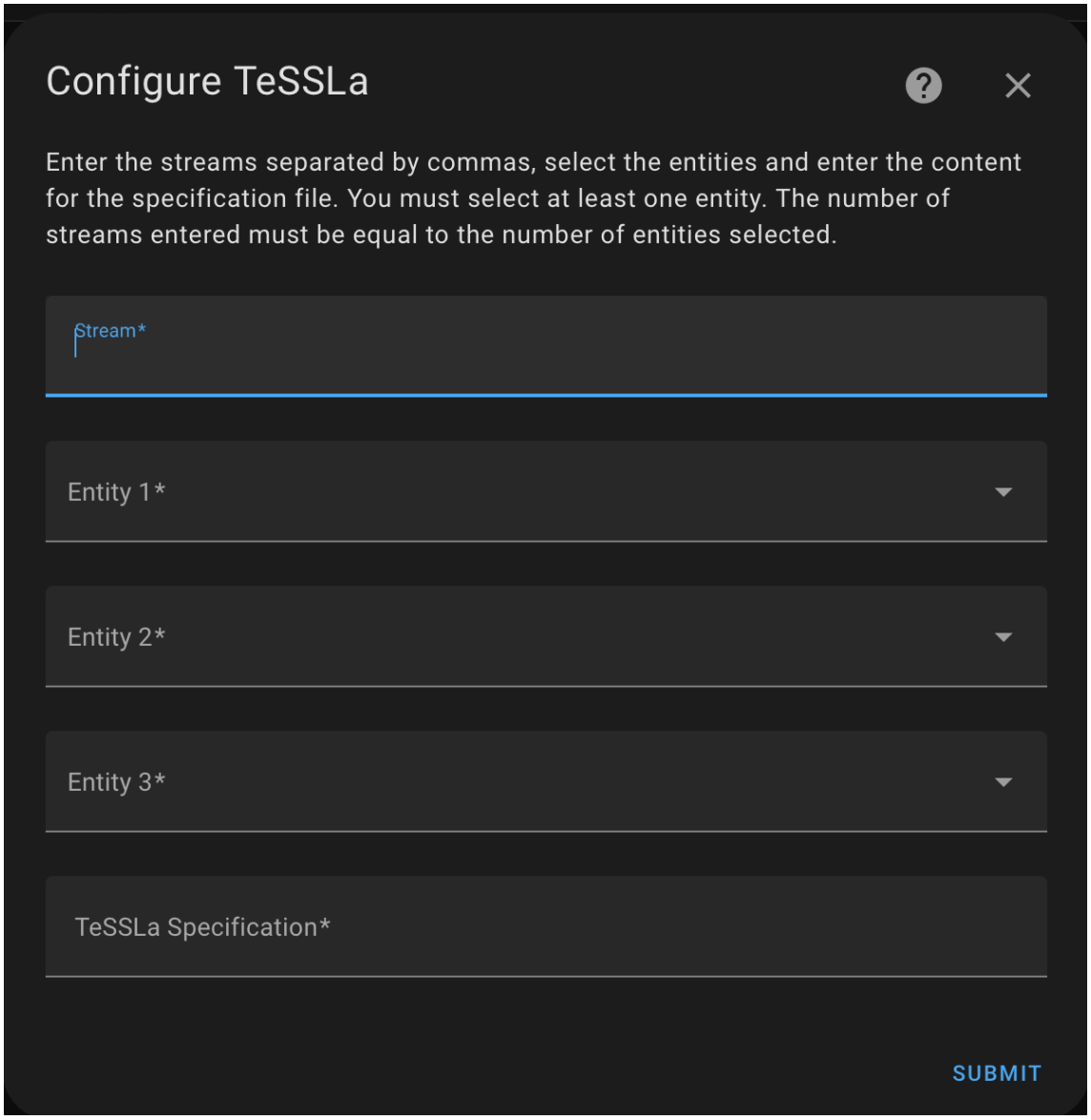

Figura 6.12: Versión inglés
Fuente: interfaz de Home Assistant

Sin embargo, si seleccionamos la ubicación al territorio español, el formulario se muestra en español como se ilustra en la imagen 6.13

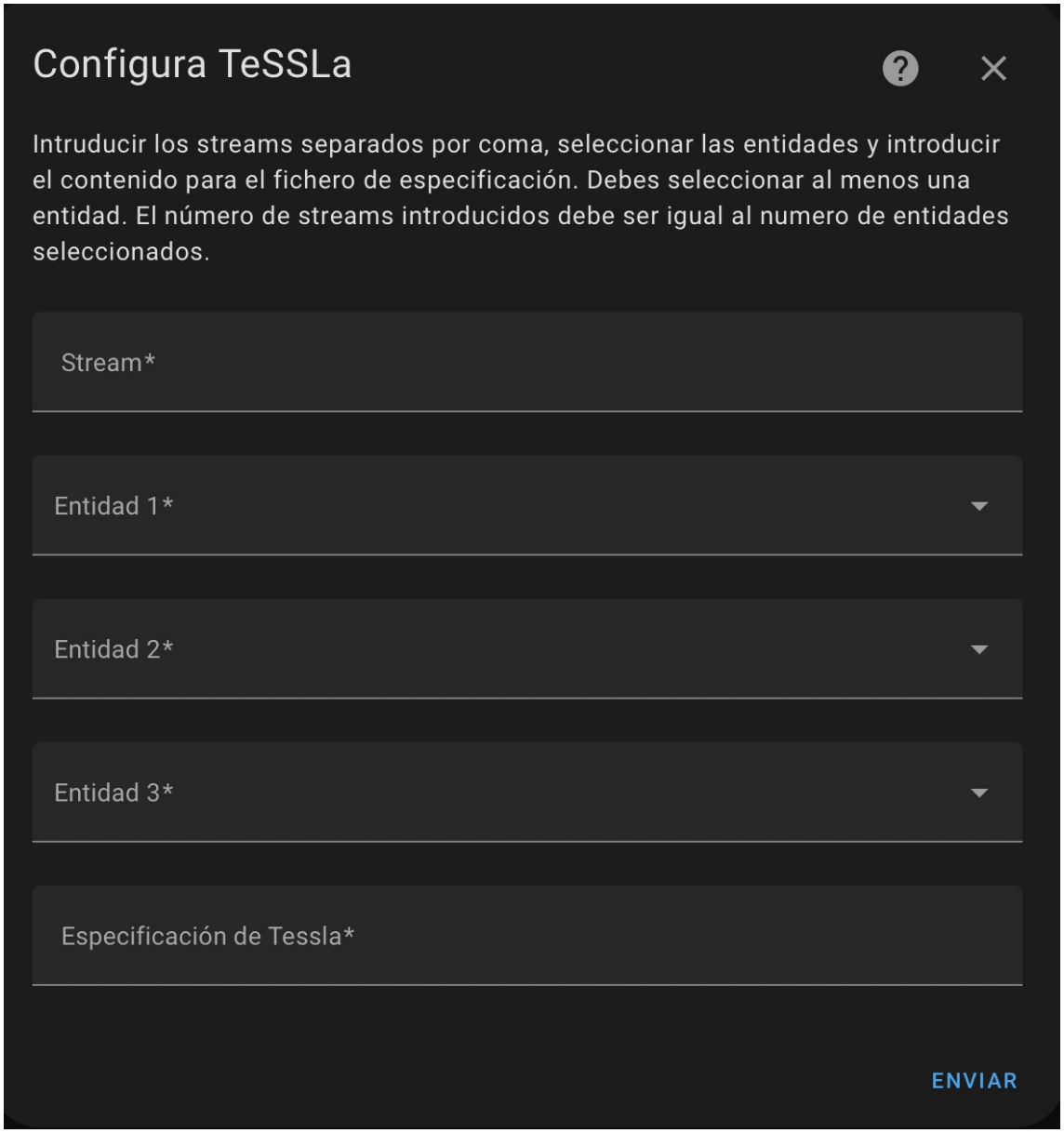

Figura 6.13: Versión español
Fuente: interfaz de Home Assistant

6.3.5. Retroalimentación de las integraciones

Se destaca también la posibilidad de poder retro-alimentar una integración a la otra, de manera que podemos utilizar una salida de output de una integración como la entrada input de otra integración. Observamos en el siguiente ejemplo:

Aprovechado el uso de AEMET, nos permite obtener el valor de `sensor.aemet_temperature` como 18.3º como la temperatura de la ubicación correspondiente de ese momento, creamos entonces un stream llamado a con la entidad de `sensor_aemet_temperature`, y en la `specification` le aumentamos 1.1 como se muestra en la figura siguiente:

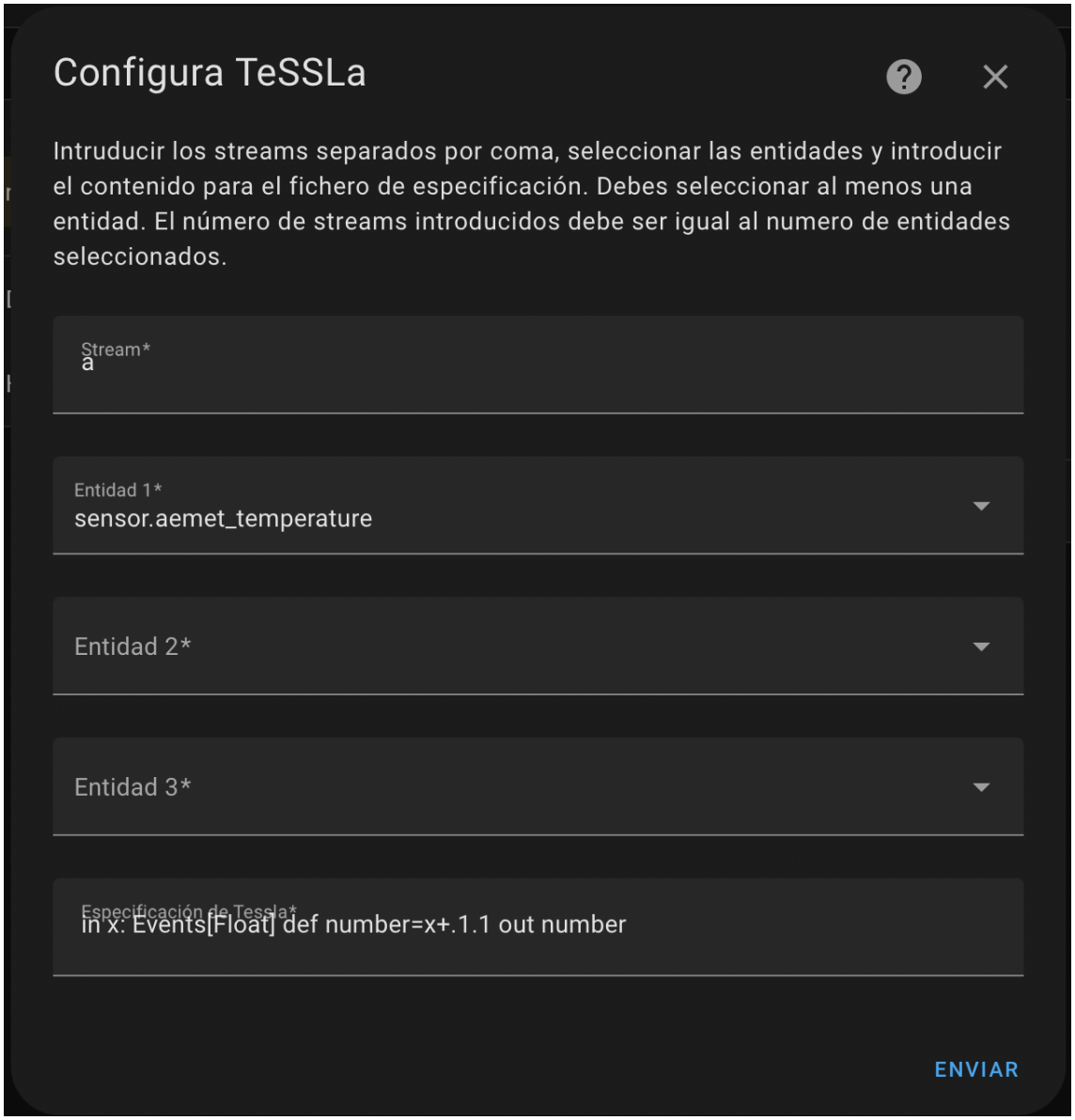

Figura 6.14: Creación del stream a
Fuente: interfaz de Home Assistant

El valor de `tessla.a_number` después de ejecutarlo es el siguiente:

Figura 6.15: Valor de output a_number
Fuente: interfaz de Home Assistant

Creamos ahora otro stream llamado b con la misma `specification` para probar la correcta función de retroalimentación. Usamos como entrada `tessla.a_number` generado por el proceso anterior:

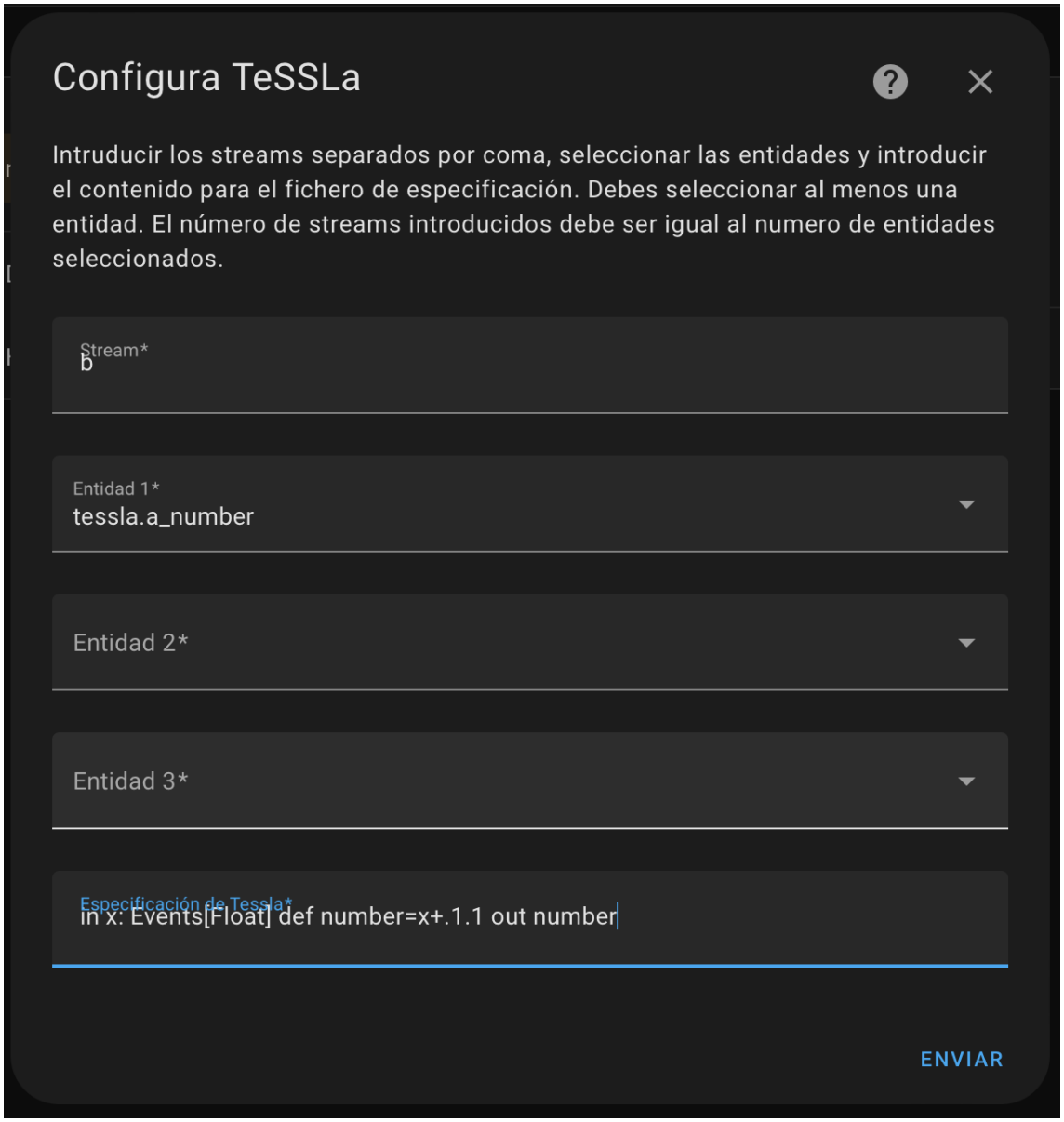

Figura 6.16: Creación del sensor b
Fuente: interfaz de Home Assistant

Vemos en la figura 6.17 cómo muestra el output del proceso de retroalimentación:

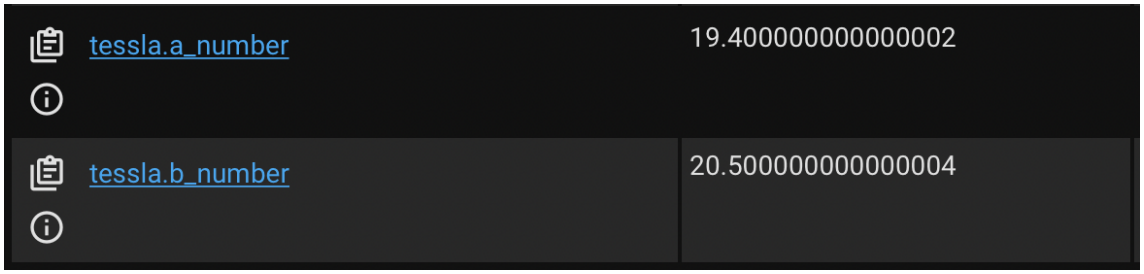

Figura 6.17: Valor output de b_number
Fuente: interfaz de Home Assistant

6.3.6. Soporte de diferentes tipos de datos

Relacionado con el apartado anterior, como se ha mencionado anteriormente, el plugin soporta la retroalimentación de la mayoría de los tipos de datos: `Int, Float, String, Unit Events, Bool`. En las próximas imágenes, ilustramos cómo integrar **TeSSLa** en **Home Assistant** con cada uno de los 5 tipos de datos. De manera que se observa en los siguiente ejemplos el uso de retroalimentación de cada uno de los tipos de datos:

En primer lugar, probamos con un test entity de tipo `Int`, usando `sensor.random_sensor_1` como su entidad de entrada y simplemente en la `specification` le mandamos sumar un entero a su valor original:

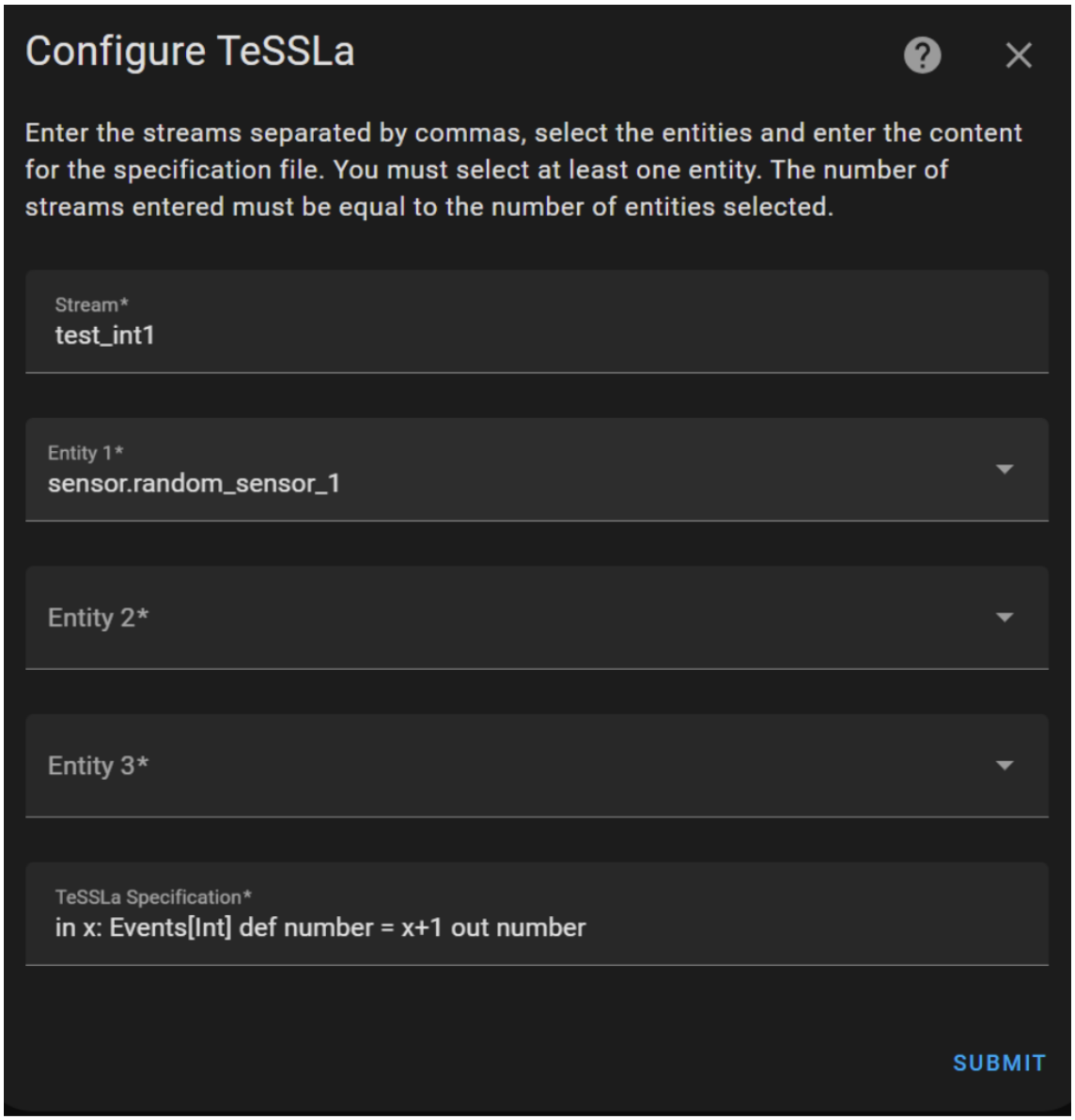

Figura 6.18: Test entity: Int sensor 1
Fuente: interfaz de Home Assistant

Se hace lo mismo con el otro sensor, pero en este caso se usa `tessla.test_int1_number` como la entidad del sensor, aprovechando nuestra funcionalidad de retroalimentación:

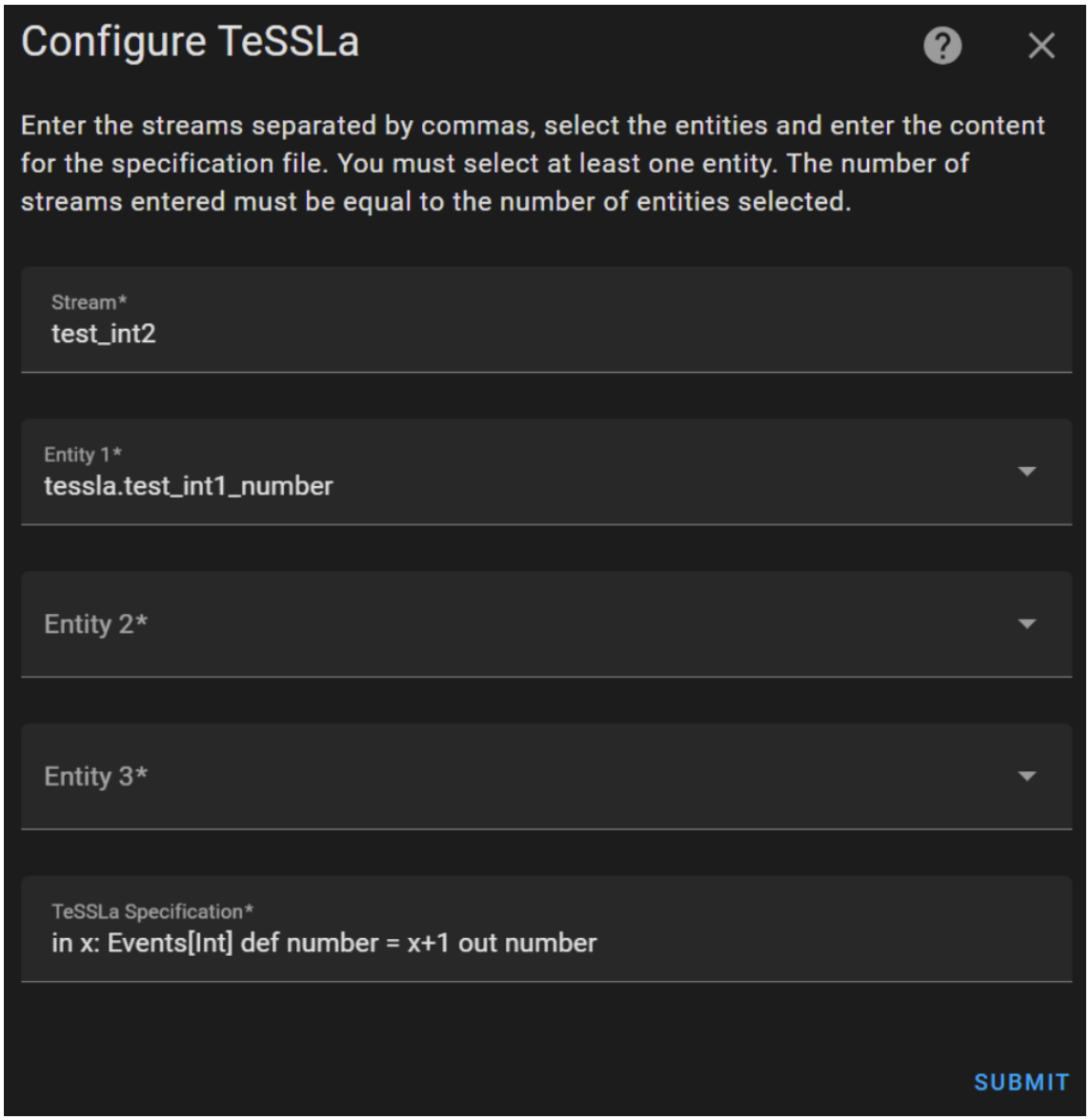

Figura 6.19: Test entity: Int sensor 2
Fuente: interfaz de Home Assistant

Como output de estos dos sensores, se observa en la siguiente figura cómo es el resultado después de la ejecución correcta con el tipo `Int` sumando dos veces un entero al valor original:

Figura 6.20: Test entity: int output
Fuente: interfaz de Home Assistant

Se observa en las próximas imágenes qué ocurre en el caso de tipo `Float`:

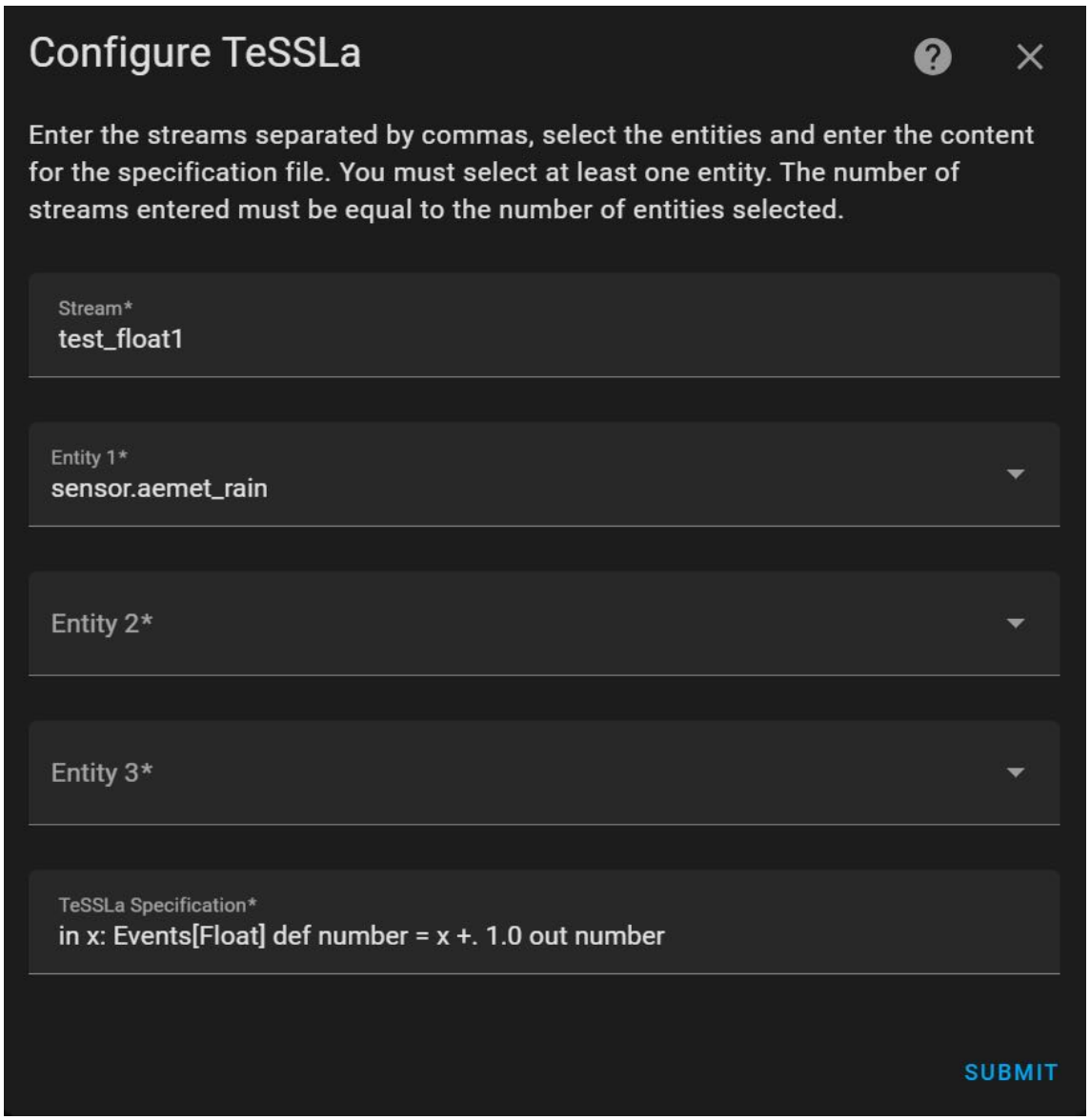

Figura 6.21: Test entity: Float sensor 1
Fuente: interfaz de Home Assistant

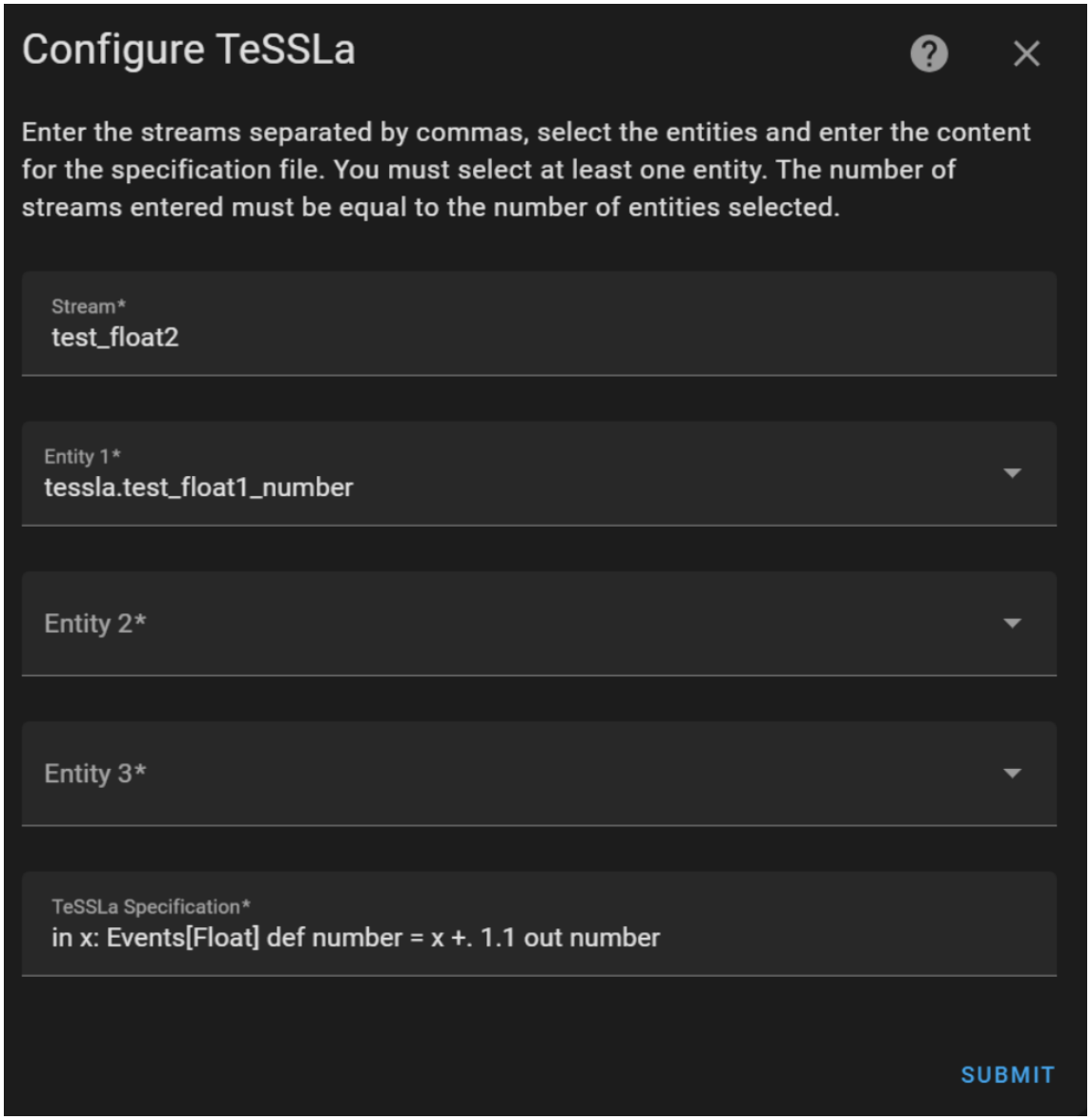

Figura 6.22: Test entity: Float sensor 2
Fuente: interfaz de Home Assistant

Como resultado, se observa que el output de cada sensor se ha ejecutado de manera correctamente siguiente a las especificaciones correspondientes:

Figura 6.23: Test entity: Float output
Fuente: interfaz de Home Assistant

Como ejemplo de la integración con una cadena de texto (`String`), usamos la AEMET. Mediante `sensor.aemet_daily_forecast_condition` , obtenemos la previsión en formato texto, como: cloudy, rainy, sunny... Posteriormente, la previsión la codificamos como un entero, tal y como se ve en el siguiente fragmento de código de especificación:

```
in x: Events[String]
def number =
    if x == "cloudy" then 1
    else if x == "rainy" then 2
    else 0
out number
```

Listing 6.1: Código de especificacion

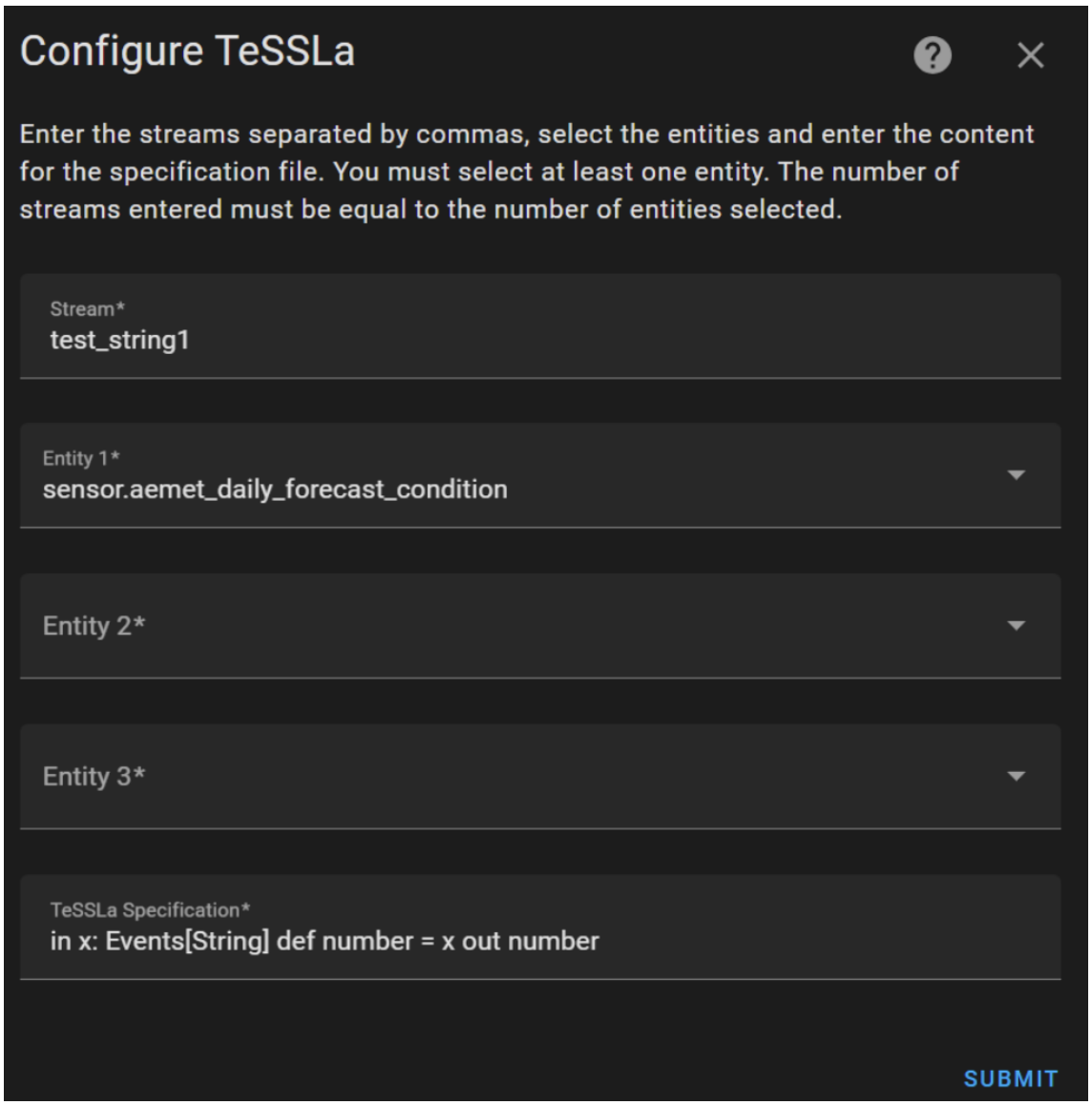

Figura 6.24: Test entity: String sensor 1
Fuente: interfaz de Home Assistant

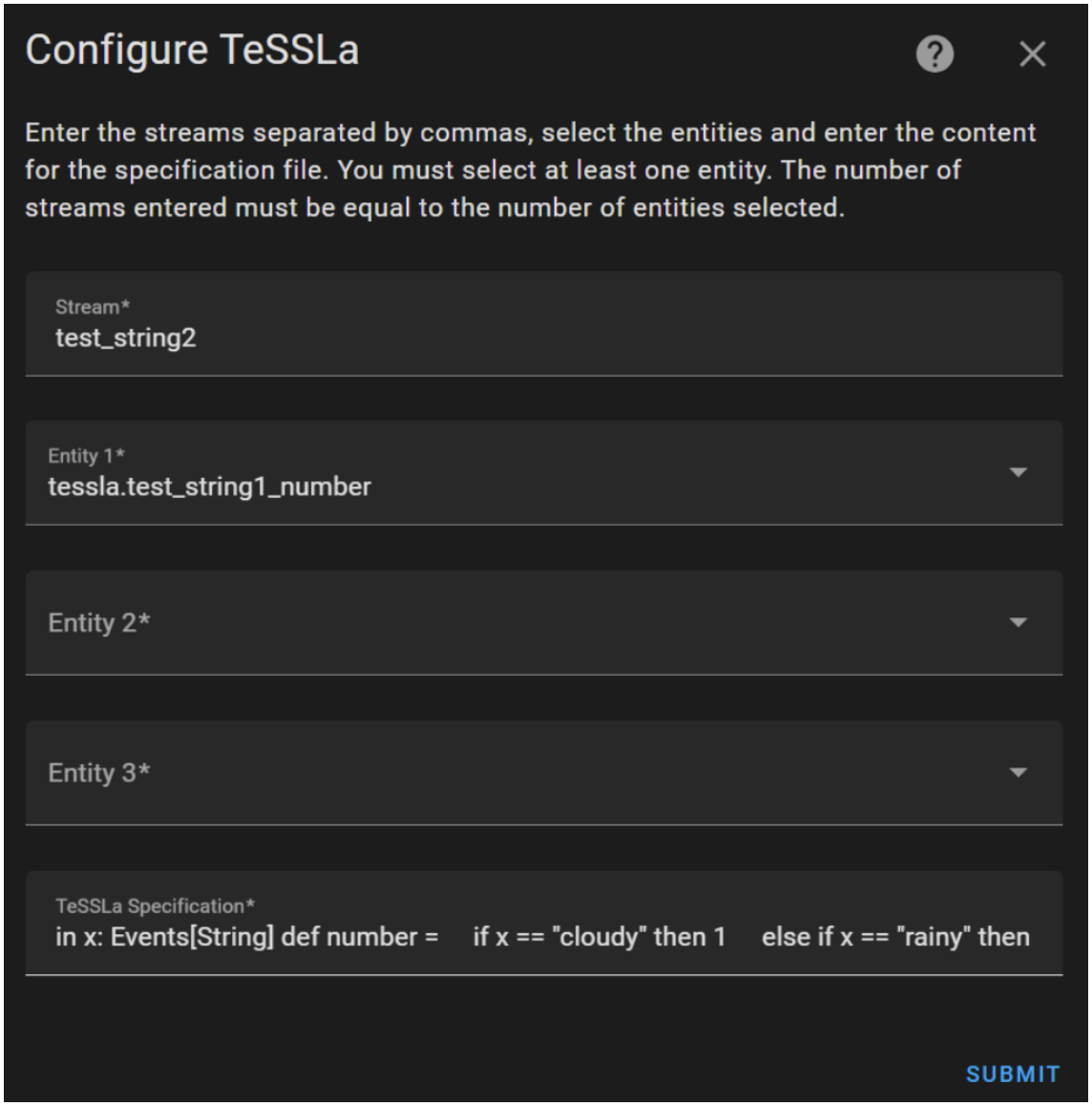

Figura 6.25: Test entity: String sensor 2
Fuente: interfaz de Home Assistant

Se observa en la tabla de resultados, como partly cloudy no corresponde ni a cloudy ni a rainy, por lo cual se obtiene el valor de 0 siguiendo a la especificación.

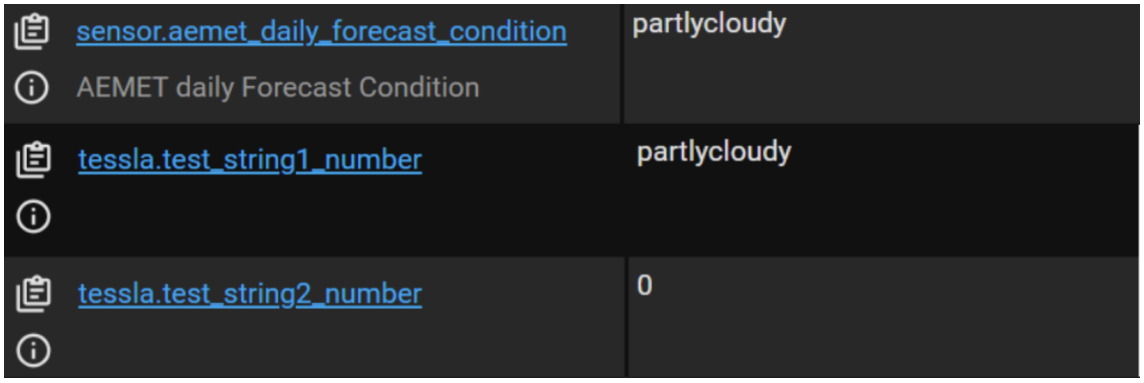

Figura 6.26: Test entity: String output
Fuente: interfaz de Home Assistant

Probamos con otro ejemplo de tipo `String` :

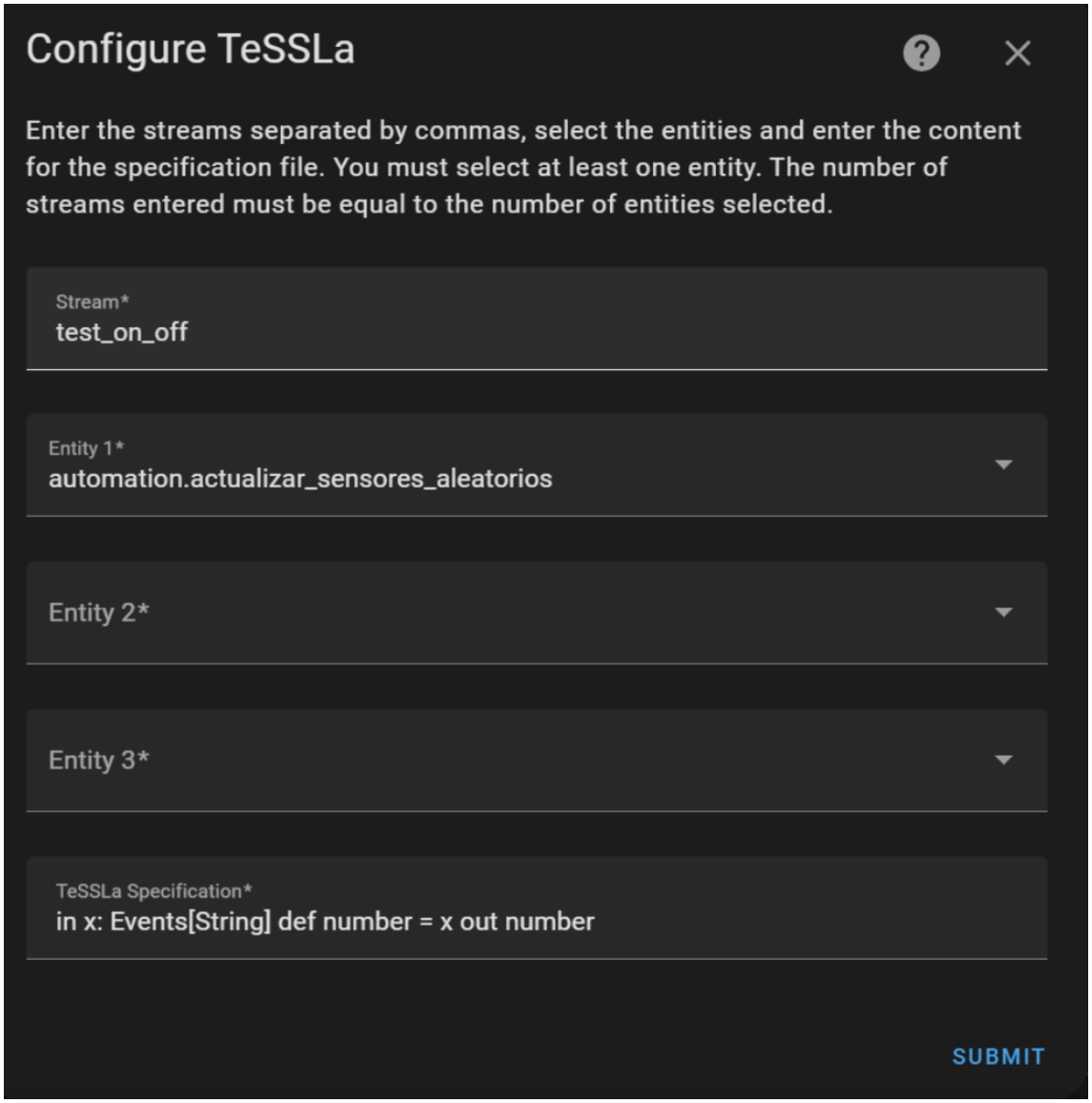

Figura 6.27: Test entity: String on/off senssor 1
Fuente: interfaz de Home Assistant

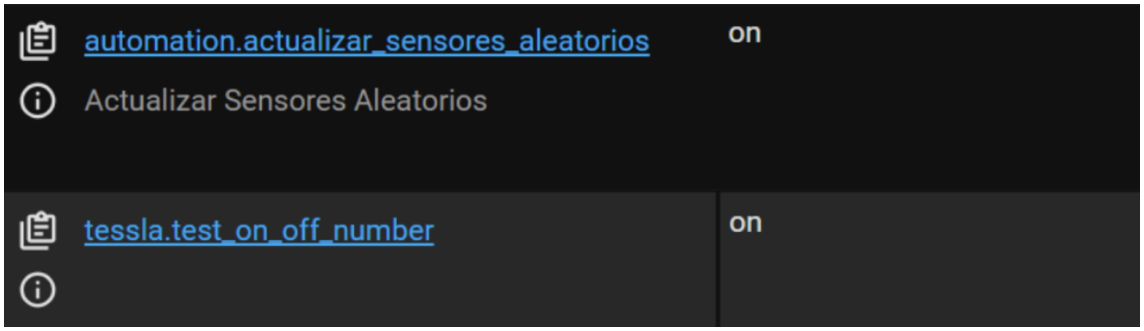

Figura 6.28: Test entity: String on/off output
Fuente: interfaz de Home Assistant

Se hace un ejemplo con el tipo de `Unit events`, que toma evento de tipo entero, lo incrementa a uno, y produce un resultado de 0 en **Home Assistant** en caso de que la salida sea ():

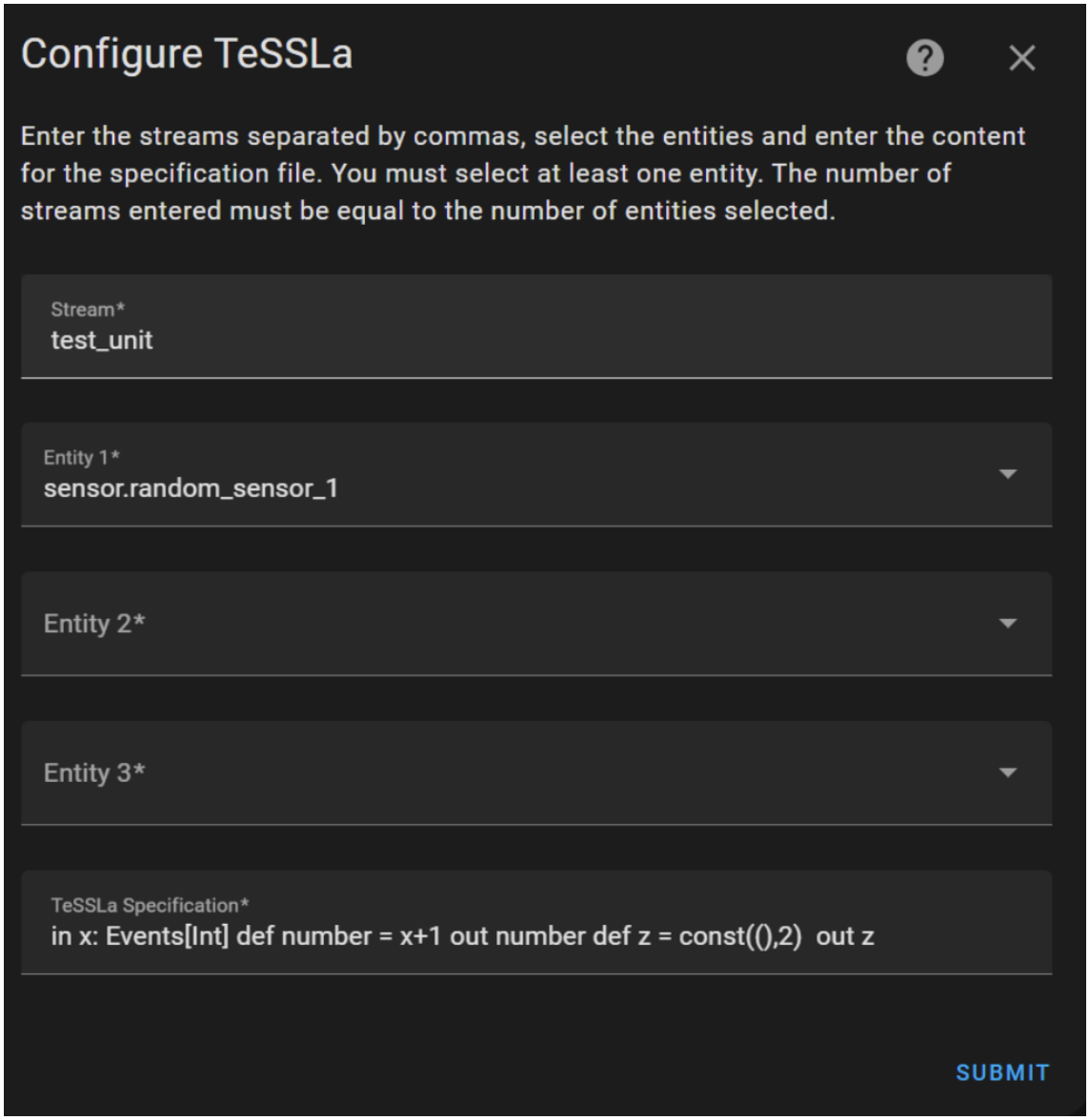

Figura 6.29: Test entity: Unit events sensor 1
Fuente: interfaz de Home Assistant

Figura 6.30: Test entity: Unit events output
Fuente: interfaz de Home Assistant

Por último, se hace un ejemplo con el tipo `Bool`:

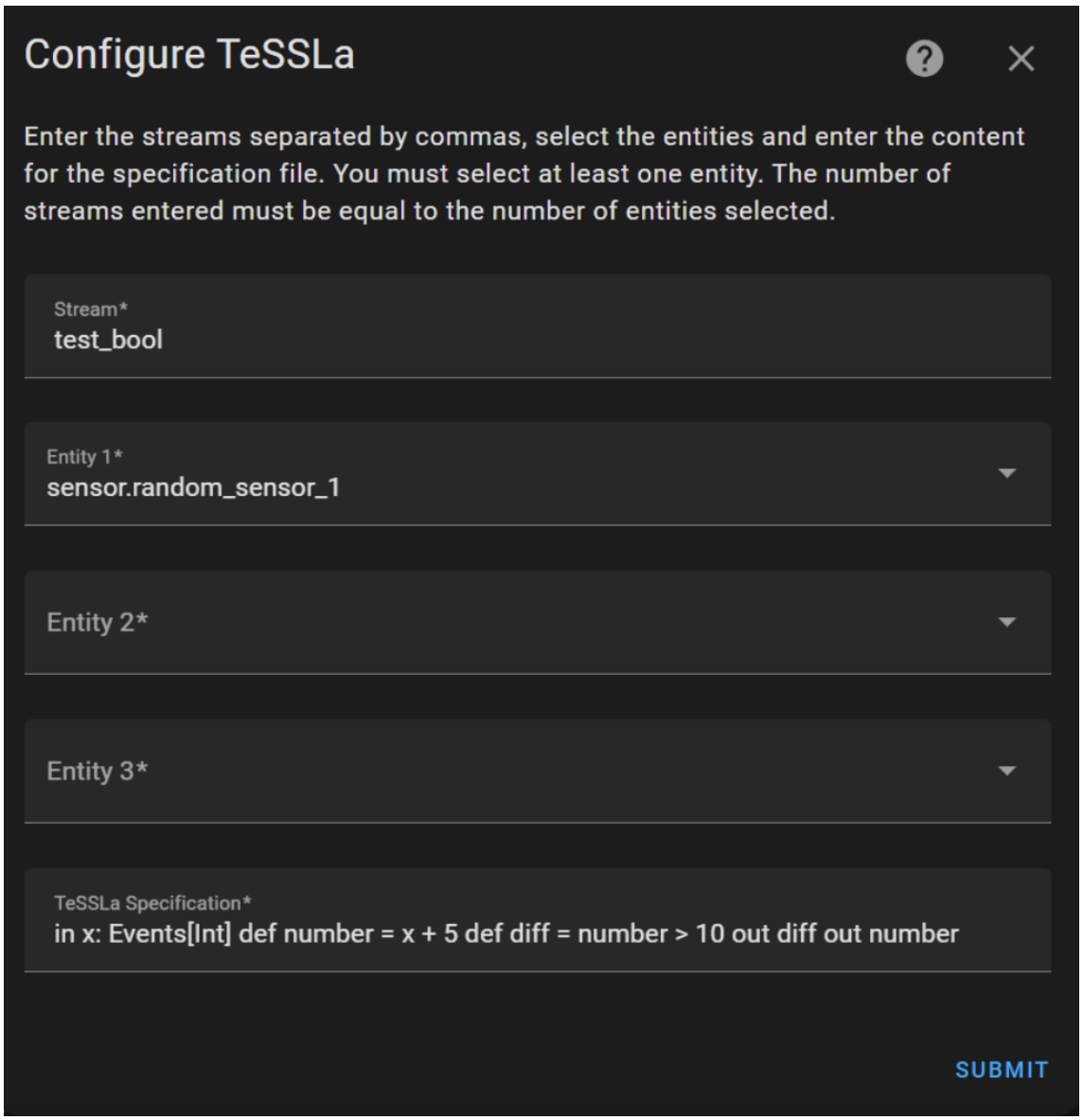

Figura 6.31: Test entity: bool sensor
Fuente: interfaz de Home Assistant

Figura 6.32: Test entity: Bool output 1
Fuente: interfaz de Home Assistant

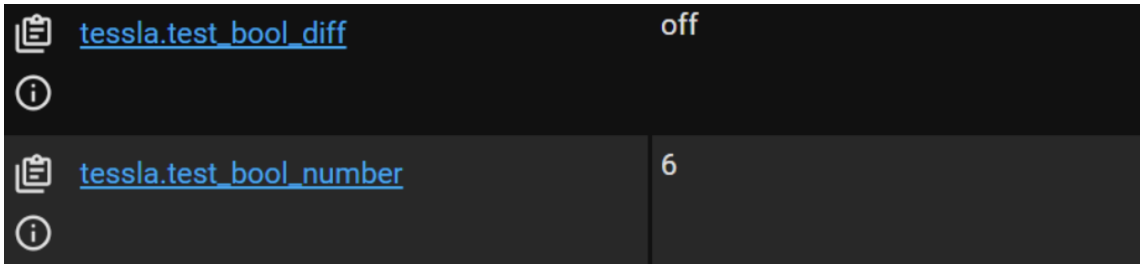

Figura 6.33: Test entity: Bool output 2
Fuente: interfaz de Home Assistant

6.4. Ejemplo de recapitulación

Para llevar a cabo todas las funcionalidades implementadas en el proyecto, se realiza posteriormente un ejemplo de recapitulación que engloba todas las integraciones que se habían comentado anteriormente:

- Predicción de clima
- Precio de electricidad
- Chequeo de tipos

En este ejemplo, contamos con información que recibimos de varias fuentes (AEMET para los datos de entrada de información de predicción meteorológica, precio voluntario para el pequeño consumidor para los consumos para obtener datos sobre el precio de electricidad). Vamos a explicar cómo funciona la integración de **TeSSLa** con varias fuentes diferentes:

```
1  def target_temperature = 21
2
3  in room_temperature: Events[Int]
4  in weather_forecast: Events[String]
5  in electricity_prices: Events[Float]
6
7  def kwh_price_budget = 0.10
8
9  def t: Events[Unit] = period(1)
10
11 def low = room_temperature <= target_temperature
12 def high = room_temperature > target_temperature
13
14 def is_cheap: Events[Bool] = electricity_prices <=.
       kwh_price_budget
15 def is_expensive: Events[Bool] = electricity_prices >.
       kwh_price_budget
16
17 def sunny = weather_forecast == "sunny"
```

```
18
19 def switch_off_heater = high || sunny || is_expensive
20 def heater = !switch_off_heater
21
22 out high
23 out sunny
24 out is_expensive
25 out heater
```

Listing 6.2: Código de recapitulación Versión 1

El código proporciona un sistema de control de calefacción que utiliza eventos para tomar decisiones basadas en diferentes variables, como la temperatura ambiente, el pronóstico del tiempo y los precios de la electricidad. La temperatura ambiente es un evento de tipo `Int`, el pronóstico del tiempo es un evento de tipo `String`, y los precios de la electricidad son eventos de tipo `Float`. Se definen también dos variables de condiciones High y Low para determinar si la temperatura ambiente es baja (menor o igual a la temperatura objetivo) o alta (mayor que la temperatura objetivo). Además, se definen eventos que indican si los precios de la electricidad son considerados baratos (menores o iguales al presupuesto) o caros (mayores al presupuesto). Se define una condición para apagar el calentador, que ocurre cuando la temperatura es alta, el pronóstico del tiempo es soleado o los precios de la electricidad son caros. Dependiendo de si se cumple la condición `switch_off_heater`, el evento de `heater` indicará si el calentador será encendido o apagado. Finalmente, las salidas del sistema son los eventos `high, sunny, is_expensive y heater`, que indican si la temperatura es alta, si el pronóstico del tiempo es soleado, si los precios de la electricidad son caros y si el calentador está encendido, respectivamente. En esta primera versión, indicamos explícitamente los tramos para los que el consumo de energía son caros o baratos.

En esta segunda versión más refinada equivalente del controlador anterior, que calcula el promedio del precio de kwh basado en precio medio de los anteriores para definir si las siguientes horas van a ser caras. El código corresponde al siguiente:

```
1  def target_temperature = 21
2
3  def target_temperature = 21
4  in room_temperature: Events[Int]
5  in weather_forecast: Events[String]
6  in electricity_prices: Events[Float]
7  def kwh_price_budget = 0.10
8  def t: Events[Unit] = period(1)
9  def low = room_temperature <= target_temperature
10 def high = room_temperature > target_temperature
11
12 def sum2(x: Events[Float]) = {
13   def s = merge(last(s, x) +. x, 0.0); s}
14 def count2(x: Events[Float]) = {
15   def s = merge(last(s, x) +. 1.0, 1.0); s}
```

```
16 def average2(x: Events[Float]) = sum2(x) /. count2(x)
17 def is_cheap: Events[Bool] = electricity_prices <=.
     average2(electricity_prices)
18 def is_expensive: Events[Bool] = !is_cheap
19 def sunny = weather_forecast == "sunny"
20 def switch_off_heater = high || sunny || is_expensive
21 def heater = !switch_off_heater
22 out electricity_prices
23 out average2(electricity_prices)
24 out high
25 out sunny
26 out is_expensive
27 out heater
```

Listing 6.3: Código de recapitulación Versión 2

Lo importamos en el playground de **TeSSLa** y observamos cómo es el output después de ejecutar el código:

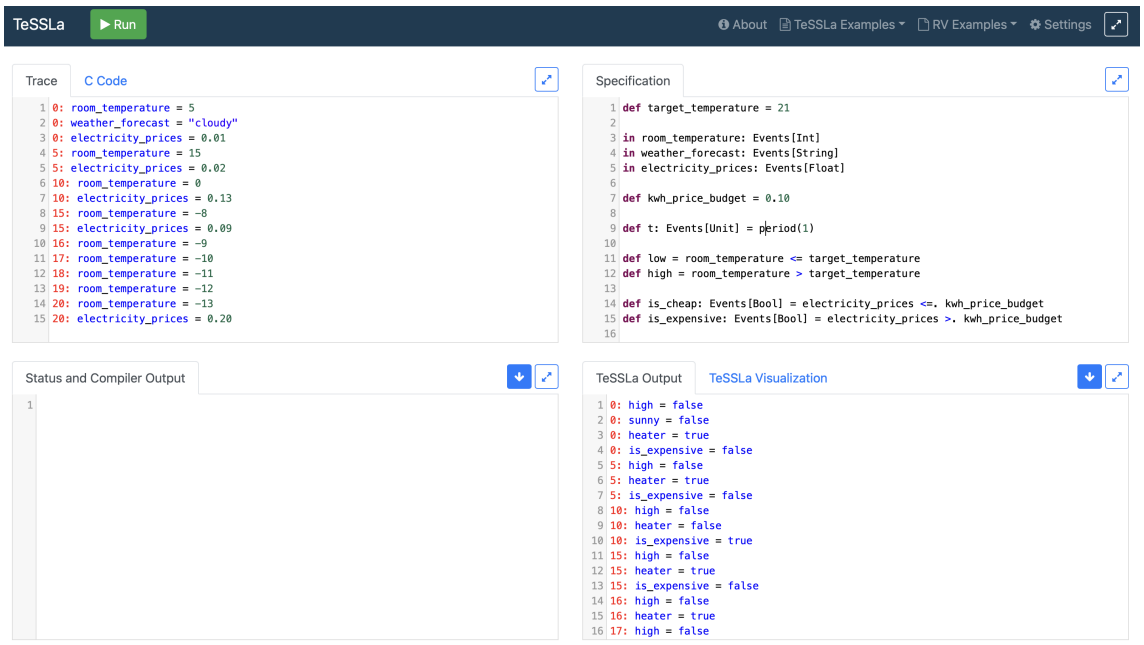

Figura 6.34: Ejemplo de recapitulación en playground
Fuente: TeSSLa web IDE

Podemos observar también cómo es la visualización en formato de flujo de los resultados. Teniendo al principio del flujo de las trazas los cuatro outputs definidos, y a medida que se avanza las trazas, se observa el cambio de estado de todos ellos. En esta pestaña nos permite visualizar mejor cuándo se enciende la calefacción:

Figura 6.35: Visualización del output
Fuente: TeSSLa web IDE

Ahora lo probamos en nuestras integraciones:
Definimos los tres sensores necesarios llamados `temperature, weather, prices` respectivamente. Cada uno relacionado con las integraciones correspondientes, las dos primeras entidades correponden a AEMET y el último se relaciona con PVPC. Introducimos nuestra especificación aportada:

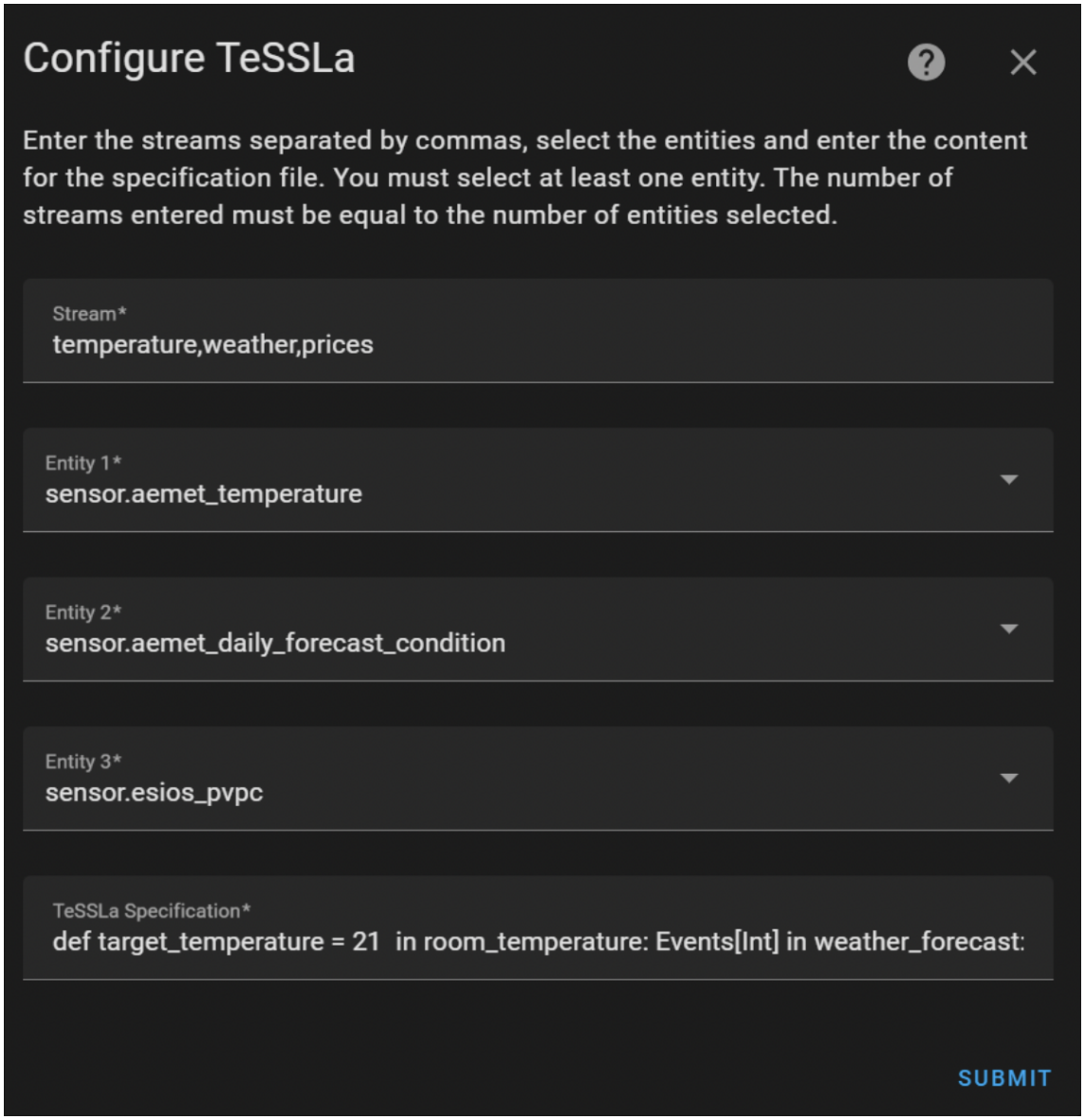

Figura 6.36: Definición de sensores
Fuente: interfaz de Home Assistant

Vemos ahora en **Home Assistant** la salida de los resultado:

- heater
- high
- is_expensive
- sunny

Cada uno de ellos muestran resultados dependiendo de los valores de entrada de ese momento.

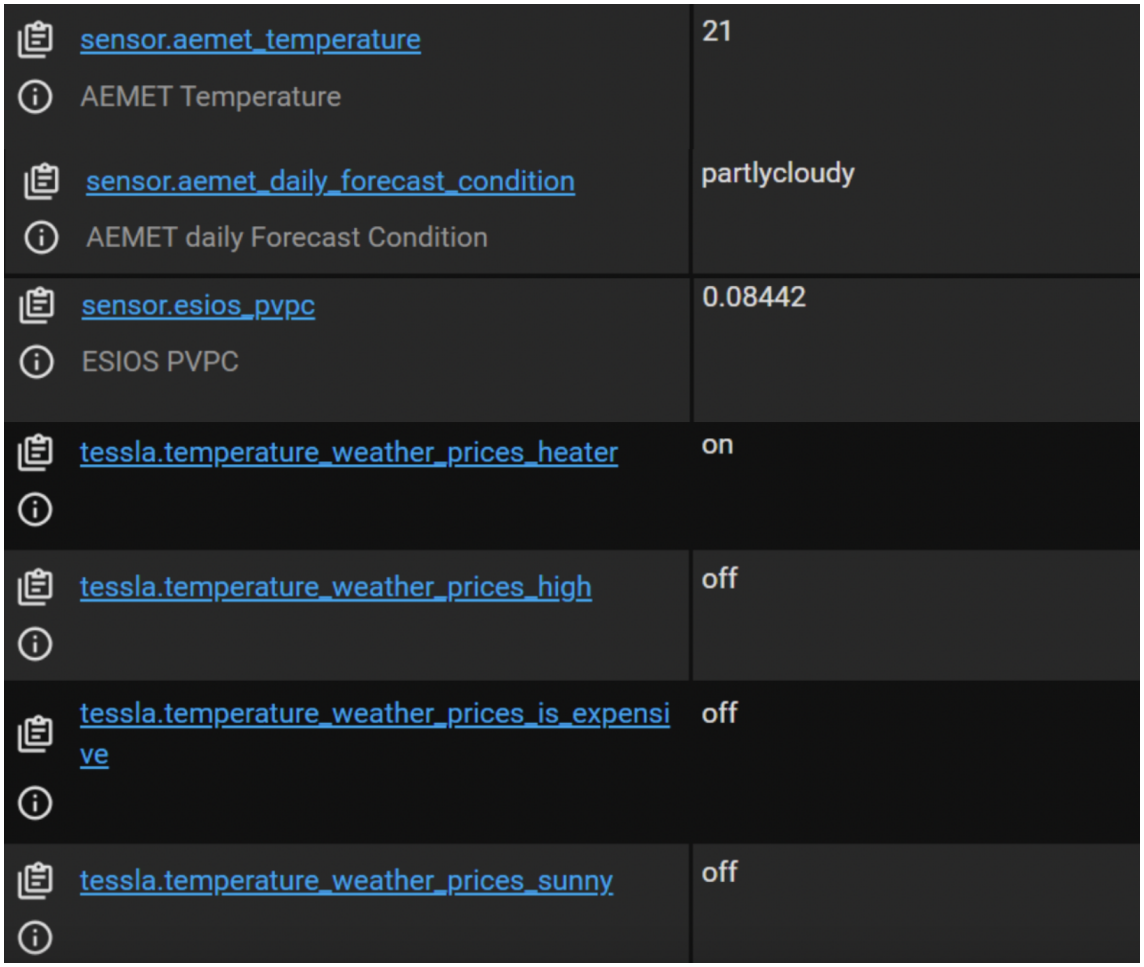

Figura 6.37: Output del ejemplo V1
Fuente: interfaz de Home Assistant

Vemos en las siguientes imágenes la salida de la segunda versión de ejemplo:

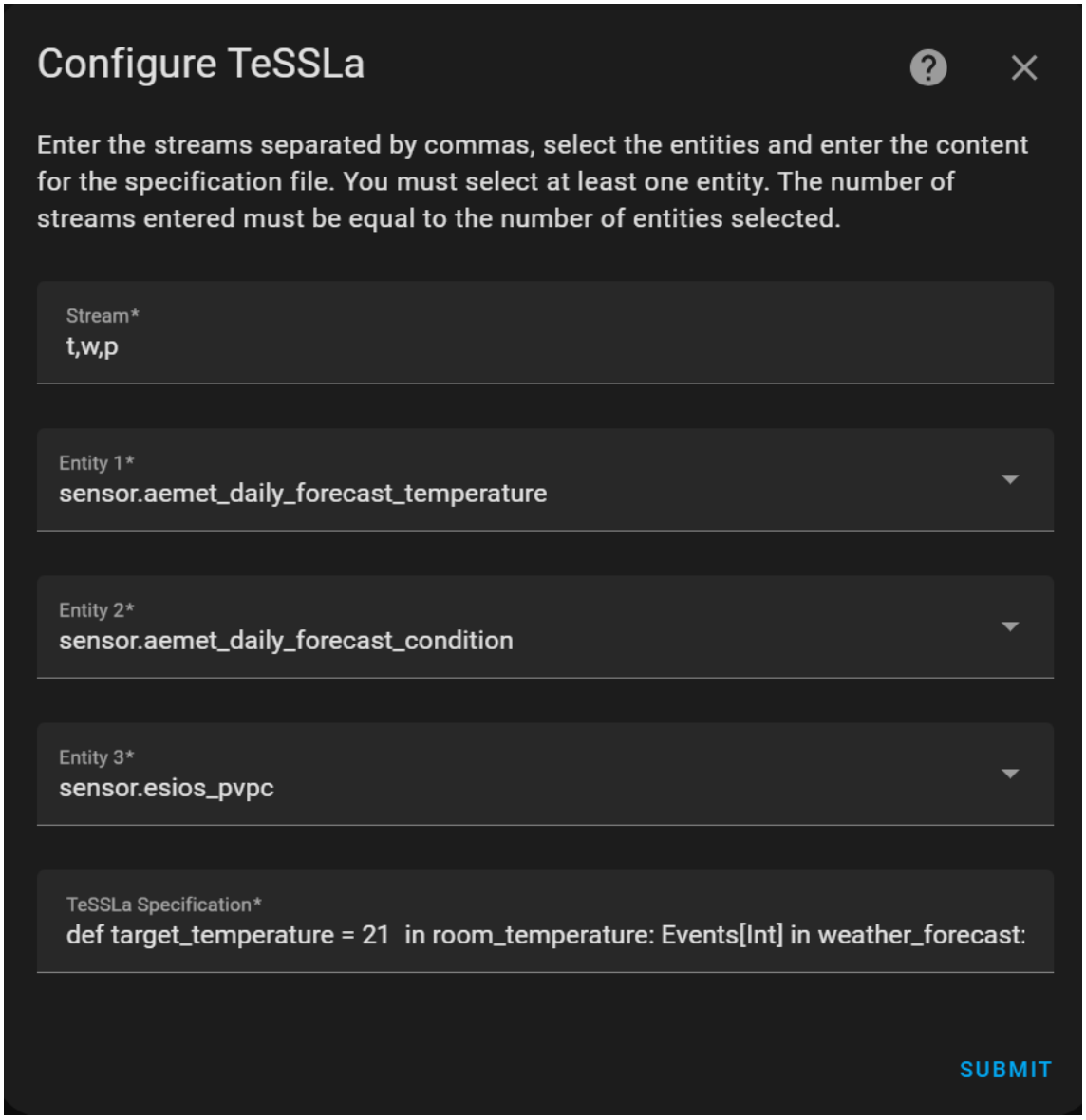

Figura 6.38: Definición de sensores V2
Fuente: interfaz de Home Assistant

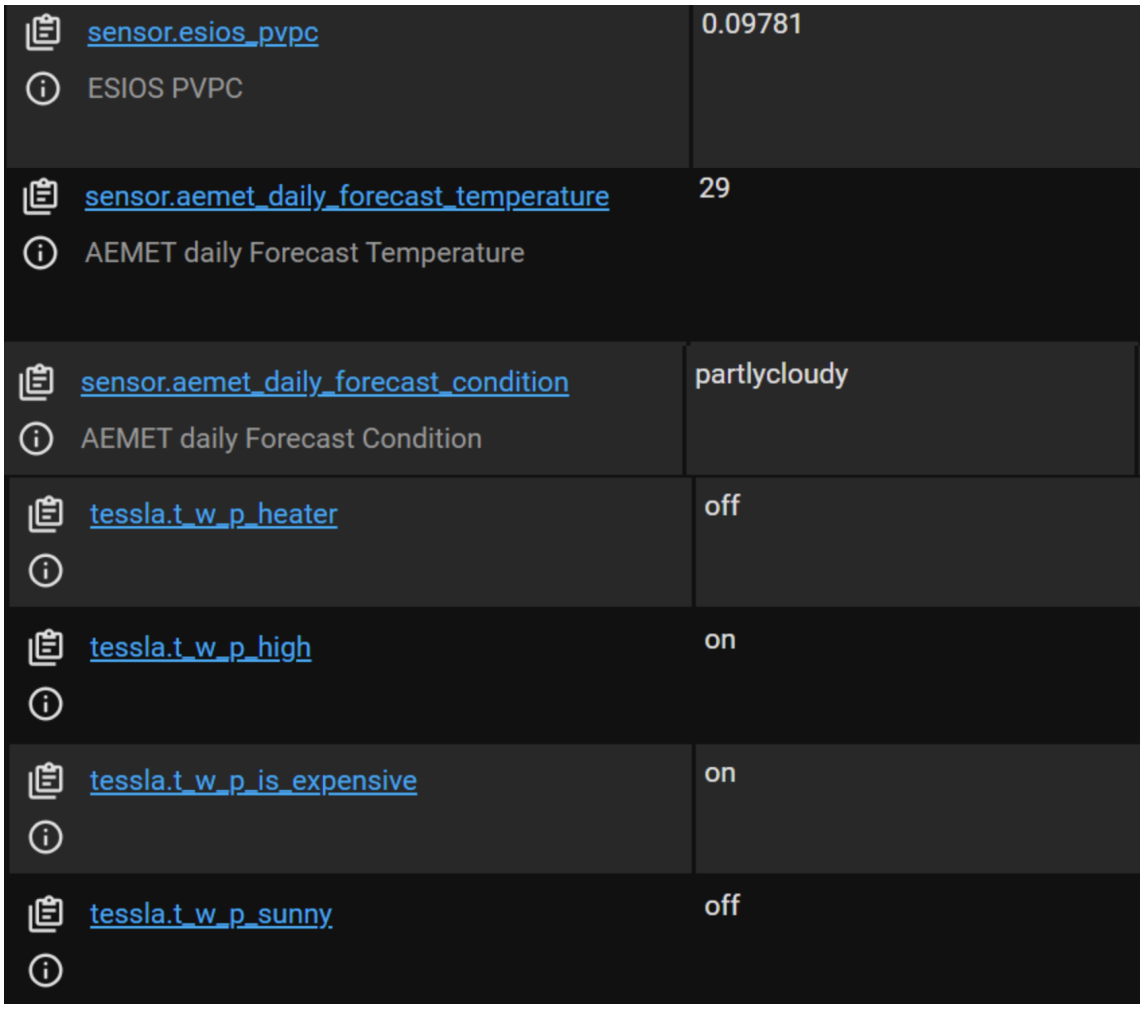

Figura 6.39: Output del ejemplo V2
Fuente: interfaz de Home Assistant

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusión

Al final del trabajo podemos concluir que hemos logrado con éxito los objetivos que nos propusimos para el desarrollo del plugin. En resumen, hemos implementado con éxito una funcionalidad que permite transmitir datos de series de tiempo a **TeSSLa** y recibir los resultados para procesarlos en **Home Assistant**. Nos aseguramos de que la interpretación de los datos enviados y recibidos facilite una comunicación fluida y eficiente entre **TeSSLa** y otros sensores del sistema. También abordamos el soporte para múltiples integraciones de **TeSSLa** simultáneamente, asegurando una experiencia de usuario perfecta. Brindamos soporte para hasta tres integraciones de flujos de entidades, incluida una configuración detallada de **TeSSLa**, un mapeo meticuloso de flujos de entidades y procedimientos rigurosos de detección y corrección de errores.

Además, desarrollamos e integramos con éxito **TeSSLa** con los servicios AEMET y PVPC, incluida una amplia conversión de tipos de datos, verificación de errores, sincronización de información y gestión eficiente de flujos de datos de estas fuentes externas. Se ha proporcionado soporte para múltiples integraciones de **TeSSLa**, lo que le permite conectarse a diferentes integraciones sin interferir entre sí. Además, hemos implementado todos los tipos de datos principales, como `Int, String, Float, Boolean`..., para satisfacer las necesidades del usuario y garantizar la versatilidad del plugin.

Sin embargo, como todavía no hay tiempo para resolverlo, aún quedan algunas tareas pendientes que debemos mejorar en el futuro:

7.2. Trabajo futuro

En cuanto al trabajo a mejorar en el futuro, se plantea posteriormente los puntos claves de mejora:

1. Signal Temporal Logic (STL)
2. Soporte de comentarios

3. MQTT

En cuanto a la integración con Signal Temporal Logic (STL), implica la capacidad de traducir las especificaciones en STL a consultas que el sistema pueda entender y evaluar sobre los datos disponibles..STL es un lenguaje formal para definir las propiedades temporales de las señales, comúnmente utilizado en sistemas de monitoreo y control. Desde la perspectiva de nuestro proyecto, STL representa un modelo de lenguaje diferente que nos permite establecer propiedades. En **TeSSLa** nos centramos en flujos de entrada y salida, mientras que en STL nos centramos en señales de entrada que contienen operadores lógicos específicos de estas señales.

Con respecto a la función de soporte de comentarios en el complemento, desafortunadamente no podemos implementarla en la tarea actual. Para abordar este problema, sería necesario descargar los complementos (Add-on), pero en nuestro método de instalación actual no se permite esta acción.

Por último, cabe mencionar el uso posterior de MQTT(MQ (2023), AWS (2023)). Es un protocolo de comunicación ampliamente utilizado en el campo del Internet de las cosas (IoT) para estandarizar cómo los dispositivos intercambian datos a través de la red. Facilita la comunicación entre dispositivos IoT y IIoT (Internet industrial de las cosas), como sensores integrados y PLC industriales. El protocolo opera con un enfoque basado en eventos y utiliza un modelo de comunicación llamado Publicación/Suscripción (Pub/Sub) para conectar dispositivos. En este modelo, el dispositivo que envía datos (editor) y el dispositivo que recibe datos (suscriptor) se comunican a través de temas y mantienen conexiones independientes entre sí. La gestión de esta conexión se realiza a través de un broker MQTT, que se encarga de filtrar los mensajes entrantes y distribuirlos a los suscriptores.

En nuestro proyecto, MQTT requiere importancia al reemplazar los buffers que creamos para resolver problemas relacionados con la recepción de eventos simultáneamente. Nuestro objetivo es utilizar MQTT para almacenar y leer eventos en el otro extremo de la cola para que podamos ordenar los eventos de los sensores individuales antes de enviarlos a **TeSSLa**, asegurando que estén organizados en orden cronológico según marcas de tiempo.

En la implementación actual, confiamos en un búfer en el que los eventos se ingresan manualmente, se clasifican según sus marcas de tiempo y luego se transfieren a **TeSSLa**. Sin embargo, este enfoque presenta desafíos porque **TeSSLa** requiere que los seguimientos de eventos se clasifiquen en orden creciente para evitar errores en el procesamiento. Por lo tanto, recomendamos usar MQTT para coordinar los eventos recibidos y permitir que **TeSSLa** los clasifique automáticamente según las marcas de tiempo. Esta alternativa promete proporcionar una solución más eficiente y ordenada que nuestro actual enfoque de amortiguación.

Introduction

7.3. Motivation

In today's world, the integration of technology into our homes has undergone a revolutionary change. Smart homes are equipped with a variety of connected devices, bringing us more convenience, efficiency and innovation. However, as these systems become increasingly complex, advanced solutions are needed to analyze and control the many devices in the modern smart home. The growth of these devices poses a significant challenge: the need for a unified and efficient platform to continuously monitor and evaluate these devices. It is proposed to implement a Python plugin as a solution to allow a more fluid integration between Home Assistant (Assistant (2023a)) and TeSSLa (TeSSLa (2023)). This integration provides the ability to send time series data more efficiently and receive results that can be used to improve home device management.

The practical importance of this research lies in its ability to significantly improve quality of life. In the context of the growing presence of the Internet of Things (IoT) in our environment, addressing the efficient management and analysis of these devices not only provides convenience, but also helps to improve energy efficiency, security and interoperability of the devices. home devices. **TeSSLa** provides a platform to integrate and analyze time series data more efficiently, rather than relying solely on Python or the languages built into **Home Assistant**. This not only improves decision making but also optimizes the operation of equipment such as heating, taking into account factors such as solar forecasts and their impact on ambient temperature. This program has a direct impact on daily life and can optimize family resources. When faced with questions like: Why turn on the heating now if the sun will rise later and heat the room? **TeSSLa** facilitates data integration more effectively, so the plugin tries to help solve these problems with automations and thus creating a more efficient and comfortable home.

Since we began our academic careers in computer engineering, we have developed a strong interest in the interaction between technology and everyday life. The concept of the smart home has always been a source of inspiration, especially in the context of the Internet of Things. Within this framework, our passion is focused on developing integrations for home assistants. The **TeSSLa** integration receives messages from other sensors integrated into **Home Assistant**, analyzes them and

passes the results to **Home Assistant**, allowing the control of smart devices. (Kallwies et al. (2022)).

This project is consistent with our ambitions and academic interests in the Internet of Things. We seek to apply our theoretical knowledge acquired in academic training to real-world situations. It is not only an academic milestone in our careers, but also a unique opportunity to contribute to the rapidly growing fields of home automation and IoT. Our interest in technological innovation and our desire to change the way we interact with technology in our daily lives were a perfect fit for the challenges and possibilities presented by this project.

This work builds on previous research that highlighted the need for advanced solutions for device management and analytics in smart homes. Throughout the project, our approach will be clear and precise, focusing on the design and implementation of functional add-ons to meet the specific management and analysis needs of IoT devices in smart homes.

All in all, the development of this Python plugin is an exciting effort to improve data integration and management in smart homes. The combination of technologies such as **Home Assistant** and **TeSSLa** provides more advanced and efficient solutions for decision making based on time series data. This initiative not only reflects our passion for technology and innovation, but also aims to improve the practical experience of living in a smart home.

7.4. Goals

As our main goal, we created a Python plugin for **Home Assistant** starting from the beginning to facilitate communication with **TeSSLa**. The plugin should be able to transfer time series data to **TeSSLa** and receive the results produced by **TeSSLa** for further processing in **Home Assistant**. With this plugin, we ensure that data sent and received are interpreted and processed consistently, allowing for smooth and efficient communication between **TeSSLa** and other sensors in the system. Additionally, we will implement strategies and techniques to optimize the performance of the plugin to ensure efficient and fast operations between **Home Assistant** and **TeSSLa**. This includes, for example, support for simultaneously integrating multiple **TeSSLa** integrations into the platform. All of this is essential to ensure a smooth and uninterrupted user experience.

Summing everything up, we consider that our main objectives are the following:

- Establish the creation of a plugin for **Home Assistant** that facilitates communication with **TeSSLa**.
- The plugin must send time series data to **TeSSLa** and receive the generated results for processing in **Home Assistant**.
- Ensure that data sent and received are interpreted and allow fluid and efficient communication between **TeSSLa** and other sensors in the system.
- Implement strategies and techniques to optimize the performance of the plugin, guaranteeing efficient and fast execution of operations between **Home**

Assistant and **TeSSLa**.

- Support multiple **TeSSLa** integrations into the platform concurrently to ensure a seamless and seamless user experience.

Subsequently, we will prepare detailed documentation that explains the process of installing, configuring and using the plugin (Manual attached in Appendix: A). Additionally, we have prepared a detailed tutorial that will serve as a guide for future developers interested in working with the plugin and understanding its functionality.

7.5. Work plan

For the development of this project, a specific methodology has been followed, inspired by Agile principles and facilitated by the Jira tool (Jira (2023),) with the aim of efficiently managing tasks and promoting collaboration between all those involved.

In the first phase, during the summer, an in-depth study was carried out on the use of Python, being the main language for the development of subsequent work. In addition, the knowledge of two fundamental elements for our case was deepened: **Home Assistant** and **TeSSLa**.

Subsequently, the environments necessary for the development of the work were configured, an aspect that will be addressed in detail in the chapter 6.2.

Once the work environments were established, specific training was carried out to acquire the necessary knowledge in Python, **TeSSLa** and **Home Assistant**, which are fundamental for the development of the project. These training processes will be described in detail in the chapter 4.

The development of the **Home Assistant** plugin was carried out progressively, tackling new subtasks and generating new ideas as the process progressed. Therefore this phase forms the main contribution to the work. These tasks will be analyzed in depth in the chapter 6.3 dedicated to the development of the project.

To ensure continuous progress and resolve any questions or problems that arose, weekly meetings lasting approximately one and a half hours were established with the tutors. These meetings are held in a hybrid way, both in person and through the Google Meet platform.

Furthermore, it is important to note that compared to the original **Home Assistant** code, our implementation has a total of approximately 200 more lines of code and 52 commits in the Github platform. This aspect will be detailed in the corresponding analysis and results chapter.

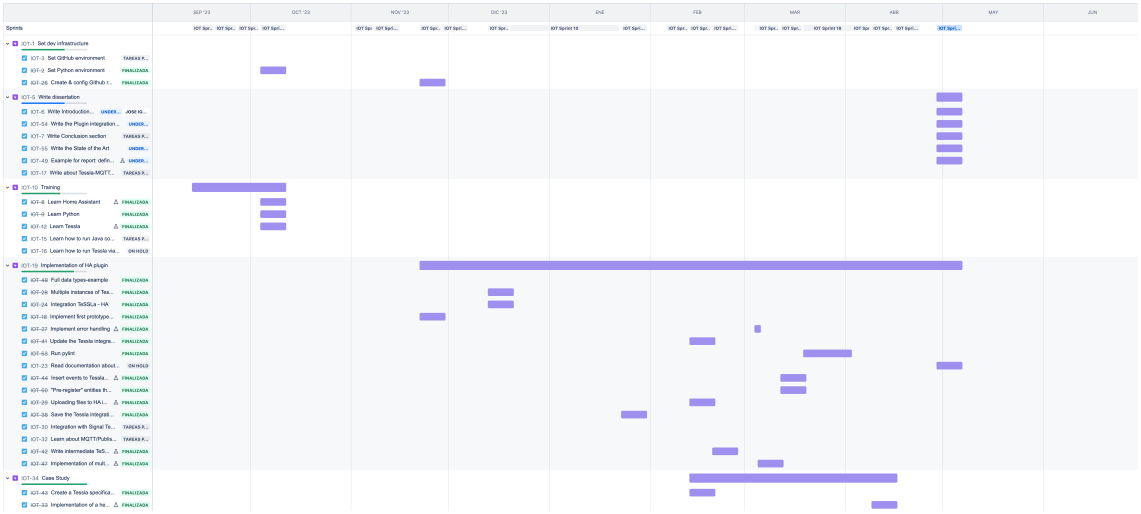

Figure 7.1: Jira work diagram
Source: Jira Tool

Conclusions and Future Work

7.6. Conclusion

At the end of the work we can conclude that we have successfully achieved the objectives we set for the development of the plugin. In summary, we have successfully implemented a functionality that allows transmitting time series data to **TeSSLa** and receiving the results for processing in **Home Assistant**. We ensure that the interpretation of sent and received data facilitates smooth and efficient communication between **TeSSLa** and other sensors in the system. We also addressed support for multiple **TeSSLa** integrations simultaneously, ensuring a seamless user experience. We provide support for up to three entity flow integrations, including detailed **TeSSLa** configuration, meticulous entity flow mapping, and rigorous error detection and correction procedures.

Additionally, we successfully developed and integrated **TeSSLa** with AEMET and PVPC services, including extensive data type conversion, error checking, information synchronization, and efficient management of data flows from these external sources. Support has been provided for multiple **TeSSLa** integrations, allowing you to connect to different integrations without interfering with each other. Furthermore, we have implemented all the main data types, such as `Int, String, Float, Boolean`..., to meet user needs and ensure the versatility of the plugin.

However, since there is still no time to solve it, there are still some pending tasks that we must improve in the future:

7.7. Future work

Regarding the work to improve in the future, the key points for improvement are subsequently raised:

1. Signal Temporal Logic (STL)
2. Comment support
3. MQTT

As for the integration with Signal Temporal Logic (STL), it implies the ability to translate specifications in STL into queries that the system can understand and

evaluate on the available data. STL is a formal language to define the temporal properties of signals , commonly used in monitoring and control systems. From our project perspective, STL represents a different language model that allows us to set properties. In **TeSSLa** we focus on input and output streams, while in STL we focus on input signals that contain logical operators specific to these signals.

Regarding the comment support feature in the plugin, unfortunately we cannot implement it in the current task. To address this issue, it would be necessary to download the Add-ons, but our current installation method does not allow this action.

Finally, it is worth mentioning the subsequent use of MQTT(MQ (2023), AWS (2023)). It is a communication protocol widely used in the Internet of Things (IoT) field to standardize how devices exchange data over the network. Facilitates communication between IoT and IIoT (Industrial Internet of Things) devices, such as embedded sensors and industrial PLCs. The protocol operates with an event-based approach and uses a communication model called Publish/Subscribe (Pub/Sub) to connect devices. In this model, the device sending data (publisher) and the device receiving data (subscriber) communicate through topics and maintain independent connections with each other. The management of this connection is carried out through an MQTT broker, which is responsible for filtering incoming messages and distributing them to subscribers.

In our project, MQTT requires importance by replacing the buffers we created to solve problems related to receiving events simultaneously. Our goal is to use MQTT to store and read events at the other end of the queue so that we can sort the events from individual sensors before sending them to **TeSSLa**, ensuring that they are organized in chronological order based on timestamps.

In the current implementation, we rely on a buffer where events are manually entered, sorted by their timestamps, and then passed to **TeSSLa**. However, this approach presents challenges because **TeSSLa** requires that event traces be sorted in increasing order to avoid errors in processing. Therefore, we recommend using MQTT to coordinate received events and allow **TeSSLa** to automatically classify them based on timestamps. This alternative promises to provide a more efficient and orderly solution than our current damping approach.

Contribuciones Personales

Xin Xiang Lin Zhou

1. Configuración de la infraestructura de desarrollo.
    - Configuración del entorno de Python.
    - Creación y configuración de un repositorio en GitHub.
2. Implementación del plugin Home Assistant.
    - Integración de TeSSLa con Home Assistant.
    - Actualización del formulario de integración de TeSSLa.
    - Implementación de manejo de errores.
    - Implementación de múltiples idiomas.
    - Inserción de eventos en el motor de TeSSLa con marcas de tiempo crecientes.
    - Ejecución de Pylint en todos los códigos Python.
3. Experimentación con un ejemplo de implementación de un controlador de calefacción basado en la temperatura de entrada y los precios de la electricidad.

Shangyu Qiu

1. Configuración de la infraestructura de desarrollo.
    - Configuración del entorno de Python.
2. Implementación del plugin Home Assistant.
    - Implementación del primer prototipo del plugin usando java.
    - Investigación de cargar archivos a HA en lugar de escribir las especificaciones de TeSSLa en el cuadro de entrada.
    - Implementación del guardado de la configuración de integración de TeSSLa cuando Home Assistant se reinicie.

- Implementación del fichero temporal de especificación de TeSSLa intermediario con HA.
- Implementación de múltiples idiomas.

3. Experimentación con un ejemplo de implementación de un controlador de calefacción basado en la temperatura de entrada y los precios de la electricidad.

Sishi Chen

1. Configuración de la infraestructura de desarrollo.
   - Configuración del entorno de Python.
2. Implementación del plugin Home Assistant.
   - Integración de TeSSLa con Home Assistant.
   - Implementación del primer prototipo del plugin usando java.
3. Redacción de memoria.

Bibliografía

Apple. How Dark Sky users can use the Apple Weather app. 2023. https://support.apple.com/en-us/102594.

home assistant. Home assistant supervisor. 2023. https://developers.home-assistant.io/docs/supervisor.

Assistant, H. Home assistant installation. 2022. https://www.home-assistant.io/installation/.

Assistant, H. Home Asisstant. 2023a. https://www.home-assistant.io/.

Assistant, H. Home assistant add-ons. 2023b. https://www.home-assistant.io/addons/.

Assistant, H. Home assistant integration architecture. 2023. https://developers.home-assistant.io/docs/architecture_components.

AWS. ¿qué es MQTT? 2023. https://aws.amazon.com/es/what-is/mqtt/.

Baumeister, J., Finkbeiner, B., Schirmer, S., Schwenger, M. y Torens, C. Rtlola cleared for take-off: Monitoring autonomous aircraft. En *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II* (editado por S. K. Lahiri y C. Wang), vol. 12225 de *Lecture Notes in Computer Science*, páginas 28–39. Springer, 2020.

Docker. Docker overview. 2023. https://docs.docker.com/get-started/overview/.

GitHub. About github and git. 2023. https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

hat, R. ¿qué es Apache Kafka? 2022. https://www.redhat.com/es/topics/integration/what-is-apache-kafka.

IoTivity. Iotivity. 2023. http://iotivity.org.

Jira. Jira: Herramientas ágiles para equipos de software. 2023. https://www.atlassian.com/es/software/jira/agile.

John, V. G. *Introduction to Computation and Programming using Python with Application to Computational Modeling and Understanding Data. Third edition*. Editorial del Libro, 2019.

Kafka, A. Apache kafka. 2023. https://kafka.apache.org/.

Kallwies, H., Leucker, M., Schmitz, M., Schulz, A., Thoma, D. y Weiss, A. Tessla - an ecosystem for runtime verification. En *Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings* (editado por T. Dang y V. Stolz), vol. 13498 de *Lecture Notes in Computer Science*, páginas 314–324. Springer, 2022.

Kauffman, S. nfer - A tool for event stream abstraction. En *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings* (editado por R. Calinescu y C. S. Pasareanu), vol. 13085 de *Lecture Notes in Computer Science*, páginas 103–109. Springer, 2021.

Maler, O. y Nickovic, D. Monitoring temporal properties of continuous signals. En *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings* (editado por Y. Lakhnech y S. Yovine), vol. 3253 de *Lecture Notes in Computer Science*, páginas 152–166. Springer, 2004.

MQ, H. MQTT essentials. 2023. https://www.hivemq.com/mqtt/.

Sinelec. ¿qué es Node-Red y para qué sirve? 2023. https://blog.gruposinelec.com/actualidad/que-es-node-red-y-para-que-sirve/.

TeSSLa. TeSSLa Tutorial. 2022. https://www.tessla.io/tutorial/.

TeSSLa. TeSSLa. 2023. https://tessla.io/.

Tessla. TeSSLa: TeSSLa Standard Library for TeSSLa 2.0.0. 2023. https://www.tessla.io/stdlib/2.0.0/types/.

Apéndice A

Manual sobre instalación de plugin

A.1. Preparación para la instalación

Antes de todo, es necesario comprobar los siguientes requisitos para poder empezar a instalar nuestro plugin:

- Docker instalado.
- Visual Studio Code instalado.
- Java y Python3 instalado en el sistema.
- El intérprete de TeSSLa en forma de un archivo JAR llamado tessla.jar.

A.2. Instalación del plugin

1. Ir al repositorio Core de Home Asisstant y crear una nueva rama.

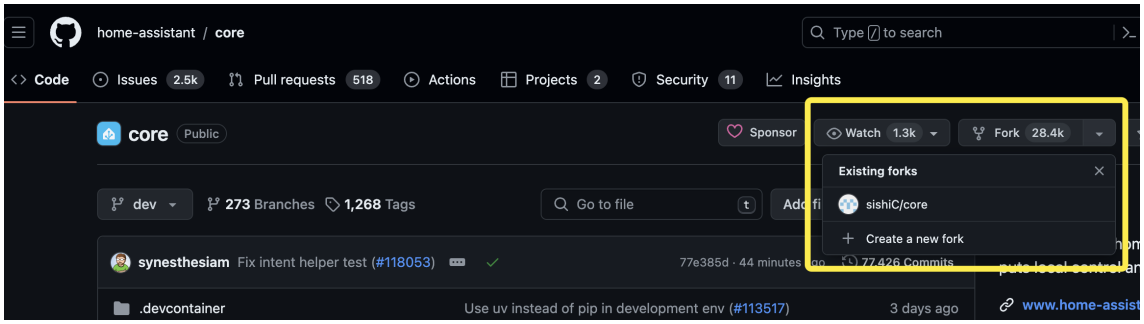

Figura A.1: Crear una nueva rama
Fuente: GitHub

2. Copiar el enlace de la nueva rama en configuración del entorno de desarrollo:

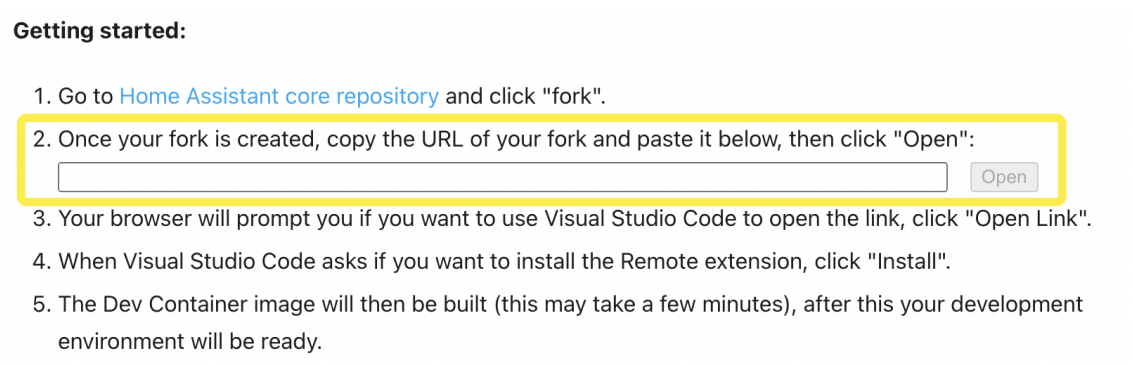

Figura A.2: Configuración del entorno de desarrollo de HA
Fuente: Home Assistant

3. Descargar nuestro código del plugin en un zip comprimido.

4. Colocar tessla.jar en la carpeta descargada llamada tessla (dentro de config/-custom_components).

5. Sustituir nuestra carpeta de config por la carpeta original de Core en Visual Studio Code.

A.3. Uso del plugin

1. Abre la página de Home Assistant y ve a Ajustes.

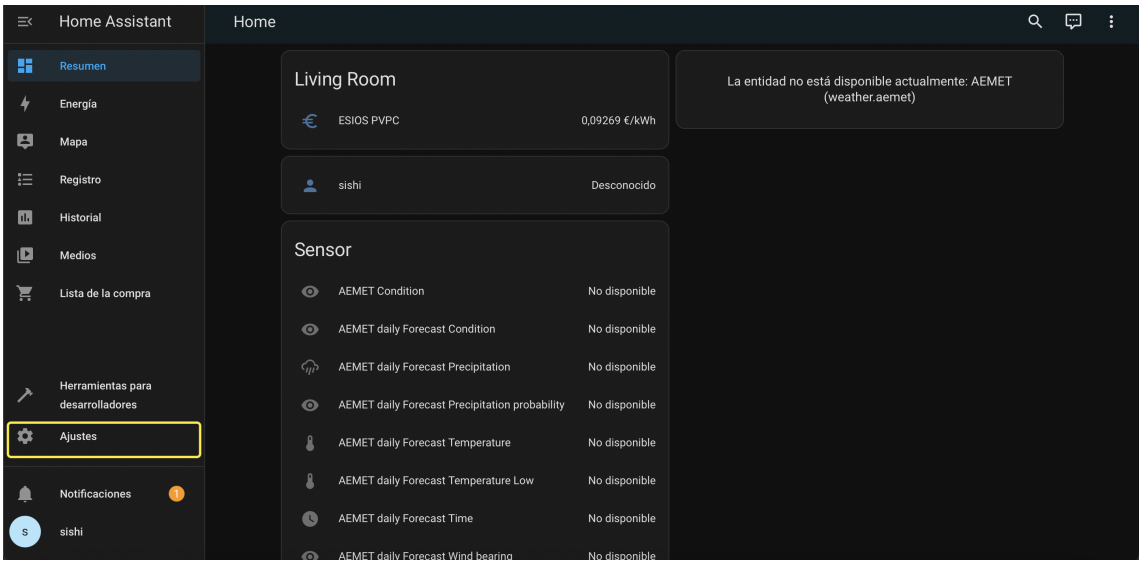

Figura A.3: Página web de Home Assistant
Fuente: interfaz de Home Assistant

2. Selecciona Integraciones en el menú.

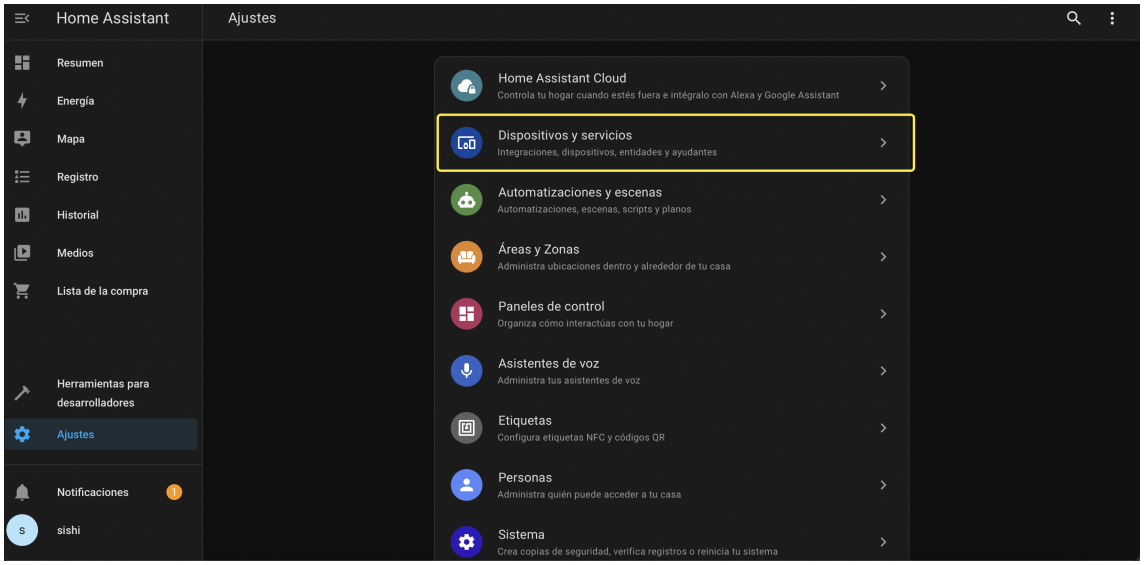

Figura A.4: Página web de Home Assistant
Fuente: interfaz de Home Assistant

3. Haz clic en Agregar Integraciones y busca TeSSLa.

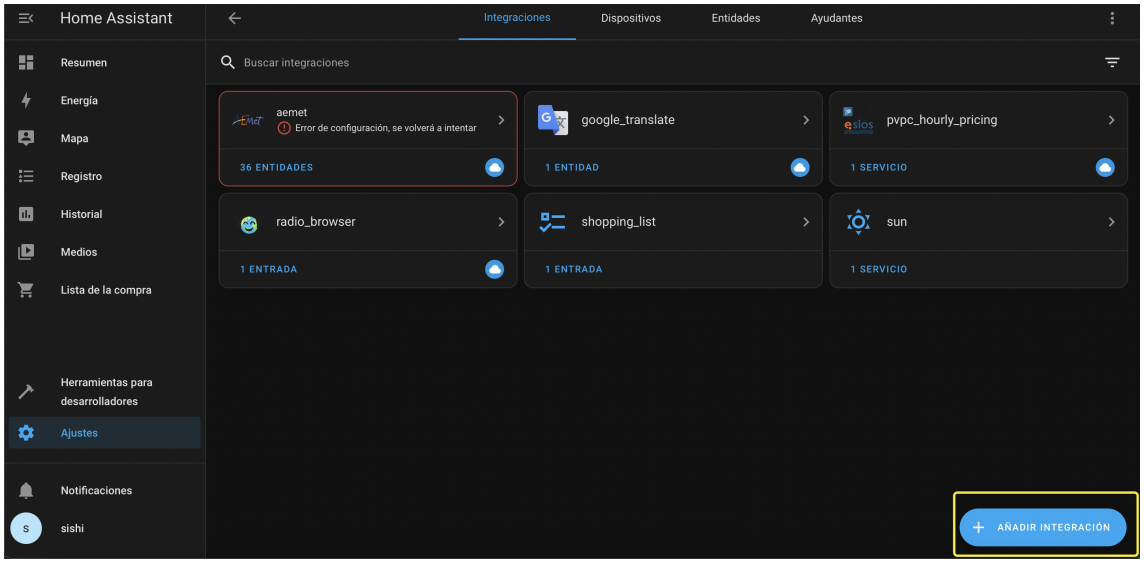

Figura A.5: Página web de Home Assistant
Fuente: interfaz de Home Assistant

4. Cuando aparezca la integración de TeSSLa en los resultados de la búsqueda, haz clic en ella.

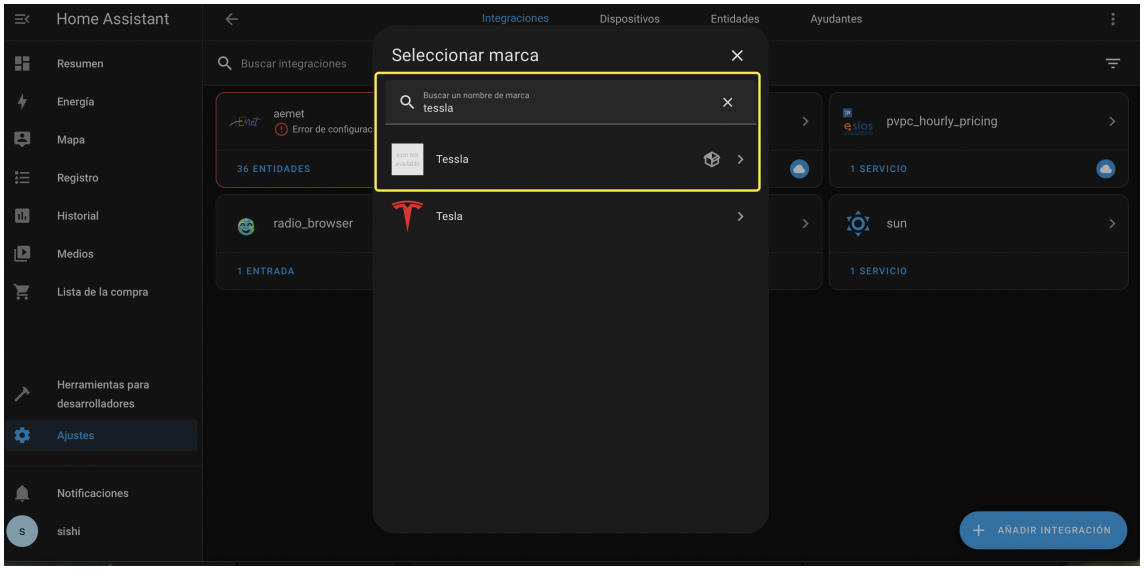

Figura A.6: Página web de Home Assistant
Fuente: interfaz de Home Assistant

5. Se te presentará el diálogo de integración de TeSSLa.

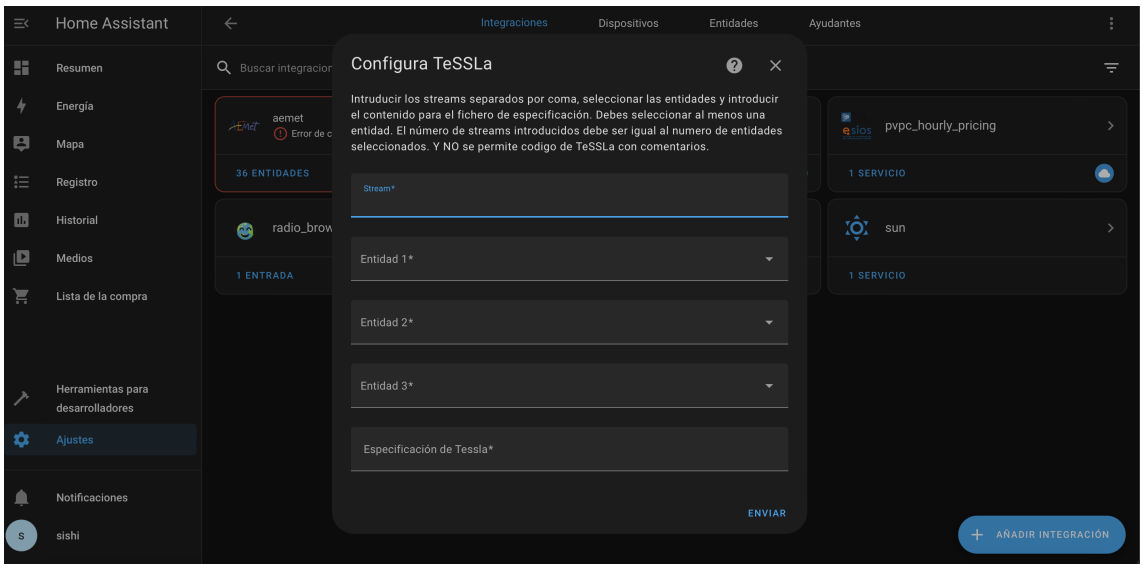

Figura A.7: Diálogo de formulario de TeSSLa
Fuente: interfaz de Home Assistant