Sistema domótico

Pareja 13
Iker Manuel Pérez Villa
Lucía Pereira Menchero

Fecha: 12/05/2024

Índice

1. Introducción

En el siguiente documento se presenta un sistema de domótica orientado al hogar, con el cual, a través de la plataforma de Discord y un bot especialmente creado para esto, podremos conectarnos e interactuar con los dispositivos IoT del propio sistema.

Los dispositivos pueden ser de tres tipos: sensores de temperatura, relojes e interruptores, y cada uno de ellos cuenta con una funcionalidad diferente, pero todos ellos comparten la principal característica de poder cambiar de estado bien cada cierto tiempo o por petición del usuario.

Los actores principales para poder desarrollar la funcionalidad al completo son:

- **Controlador**
  Actor central del sistema. Se encarga de la comunicación entre el bot y los dispositivos IoT, teniendo en cuenta siempre las reglas del sistema ya definidas.

- **RuleEngine**
  Componente mediante el cual se registran reglas predeterminadas por el usuario en base a un cambio de estado de alǵun dispositivo del sistema.

- **Bridge**
  Actor encargado de la traducción de mensajes desde el bot de Discord al controlador.

- **Dispositivos IoT**
  Simulación de dispositivos con un determinado estado que puede cambiar.

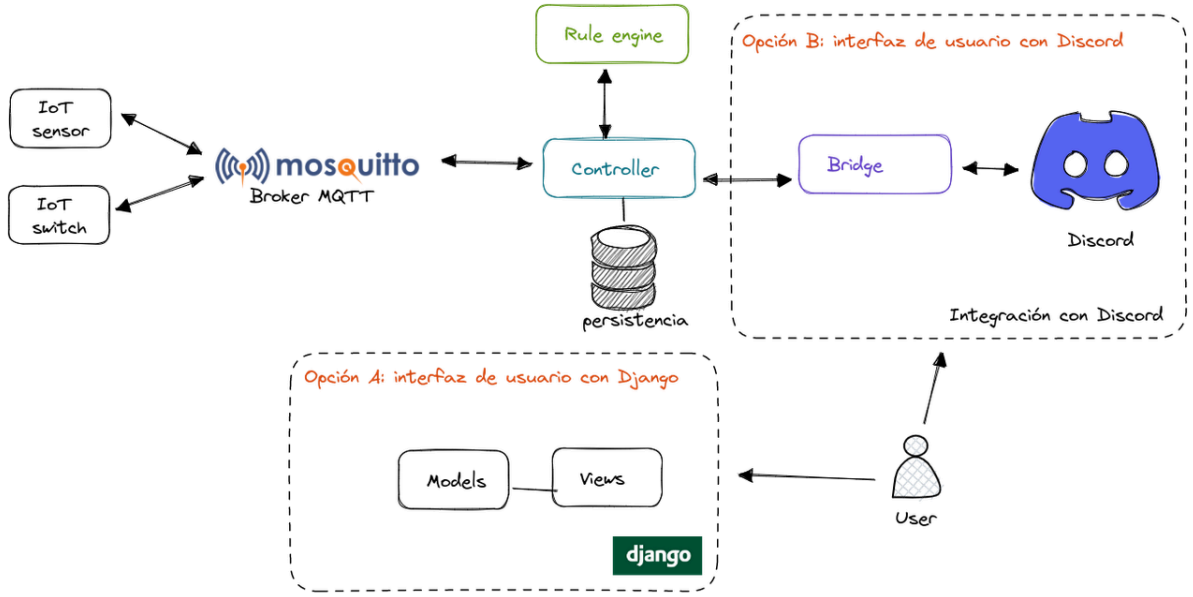
- **Bot de Discord**
  Interfaz para el uso del usuario, mediante el cual envía mensajes para realizar una serie de acciones posibles.

Para la comunicación entre los actores se utiliza el *broker* colas de mensajes MQTT, Mosquitto. Asimismo para poder utilizar el bot de Discord se hace uso de la API de Discord en Python.

Con este proyecto se entiende el funcionamiento de los dispositivos IoT como también se expanden los conocimientos de uso de colas de mensajes como medio de comunicación al utilizar un broker distinto y más ligero a la práctica anterior.

2. Definición del proyecto

En la siguiente imagen se puede observar la estructura principal del proyecto a desarrollar.



Este sistema ofrece una interfaz gráfica para el usuario mediante un bot en la plataforma de Discord. Dicho usuario puede crear y eliminar dispositivos IoT del sistema, como también solicitar un listado de ellos, saber el estado de ellos y establecer un nuevo estado. También puede crear, eliminar y listar las reglas de configuración del sistema.

En el caso de los interruptores y relojes, los estados pueden cambiar cada cierto tiempo (tiempo configurable si se quisiera).

2.1. Objetivos y funcionalidad

El objetivo principal del proyecto es crear un sistema distribuido y sincronizado. Para ello, se utiliza el protocolo de colas de mensajes mediante MQTT y el broker Mosquitto, como también la API que proporciona Discord para la implementación de la comunicación con el bot. El lenguaje de programación es Python en todo momento.

¿Cómo funciona?

Una determinada acción comienza en el usuario que la solicita al comunicárselo al bot de Discord. Será el Bridge quien reciba el mensaje del bot y lo interprete, comprobando que está bien escrito y que es un comando aceptado por el sistema. Realiza la traducción del mensaje y lo envía al Controlador. Este será el encargado de redirigir el mensaje al dispositivo concreto, y a su vez, consultar con el RuleEngine si hay acciones que realizar debido a alguna regla del sistema. En caso de que así sea, el Controlador comunica a los dispositivos involucrados estas acciones nuevas.

Todos los resultados llevados a cabo en el sistema son comunicados de vuelta al usuario por la interfaz de Discord.

2.2. Limitaciones del sistema

El sistema cumple con la funcionalidad descrita en el apartado anterior, pero debe tenerse en cuenta que tanto las acciones a realizar por los dispositivos como las reglas posibles, están limitadas a lo descrito en esta memoria. Por lo tanto, no es un sistema domótico de ámbito general.

También, el sistema no se encarga de instanciar y poner en marcha nuevos dispositivos con un mensaje por el bot de agregar un dispositivo; dicho mensaje simplemente comunica que se registre el nombre del dispositivo en el listado de dispositivos. El dispositivo en sí debe estar funcionando en la máquina que ejecuta todo el proyecto, y debe *lanzarse* manualmente.

2.3. Requisitos Funcionales

Lógica del Controlador

[RF-C-1] El controlador puede recibir mensajes traducidos del bridge, estando seguro de que no puede haber fallo en el formato de estos.

[RF-C-2] El controlador hace uso del RuleEngine como un atributo más, utilizando sus funciones cuando lo necesite.

[RF-C-3] El controlador puede enviar mensajes tanto al bridge como a los dispositivos del sistema.

Lógica del Bridge

[RF-B-1] El bridge puede recibir mensajes del bot de Discord.

[RF-B-2] El bridge puede recibir mensajes de respuesta del controlador.

[RF-B-3] El bridge debe poder traducir los mensajes del bot de manera correcta, respondiendo con mensaje de consultar los comandos en caso de un mal formato.

[RF-B-4] El bridge puede enviar mensajes al controlador para solicitar determinadas acciones.

[RF-B-5] El bridge puede enviar mensajes al bot de Discord.

Lógica del RuleEngine

[RF-RE-1] El RuleEngine puede listar los dispositivos del sistema.

[RF-RE-2] El RuleEngine puede listar las reglas del sistema.

[RF-RE-3] El RuleEngine puede recoger el estado de los dispositivos del sistema.

[RF-RE-4] El RuleEngine puede establecer el estado de los dispositivos del sistema.

[RF-RE-5] El RuleEngine puede agregar los dispositivos nuevos del sistema.

[RF-RE-6] El RuleEngine puede eliminar los dispositivos del sistema.

[RF-RE-7] El RuleEngine puede agregar las reglas nuevas del sistema.

[RF-RE-8] El RuleEngine puede eliminar las reglas nuevas del sistema.

Lógica de los Dispositivos IoT

[RF-D-1] Un dispositivo IoT puede ser de tres tipos: sensor, reloj o interruptor.

[RF-D-2] Un dispositivo IoT puede registrarse en el fichero del sistema al instanciarse en el dispositivo.

[RF-D-3] Un dispositivo IoT puede eliminar su entrada en el fichero del sistema al terminar su ejecución en el dispositivo.

[RF-D-4] Un dispositivo IoT puede cambiar de estado si el usuario lo solicita.

[RF-D-5] Un dispositivo IoT puede enviar su estado como respuesta si el usuario lo solicita.

[RF-D-6] Un dispositivo IoT envía mensajes por defecto cada cierto tiempo (si es un sensor o un reloj; en caso contrario no).

2.3. Decisiones de funcionalidad

A continuación se responden a una serie de preguntas acerca de las decisiones tomadas en la funcionalidad de nuestro proyecto.

- **¿Controller y RuleEngine han de ser aplicaciones separadas?, ¿por qué?, ¿qué ventajas tiene una y otra opción?**

No tiene por qué, ya que la funcionalidad de reglas que implementa el RuleEngine podría unificarse con la del controlador directamente. Al carecer de conexión con una cola de mensajes, y simplemente que el RuleEngine sea instanciado como un atributo en el Controlador, hace que el registro y comprobación de funciones de un fichero pueda hacerse en una misma clase de Python.

- **¿Cómo se comunican Controller y RuleEngine en la opción escogida?**

Controller es una clase de Python que mantiene conexión con el broker de colas de mensajes para comunicarse tanto con los dispositivos IoT como con la clase Bridge.

RuleEngine, en cambio, es otra clase que no requiere de la conexión con ningún componente del sistema y contiene una funcionalidad para leer y escribir contenido en el fichero de reglas y comprobar si alguna de ellas se activa frente a un cambio de estado.

Controller instancia un RuleEngine como un atributo más y llama a sus funciones para agregar, eliminar y comprobar las reglas.

- **¿Tiene sentido que alguno de estos componentes compartan funcionalidad?, ¿qué relación hay entre ellos?**

Sí, los dispositivos IoT funcionan con una misma base lógica. Se conectan al broker para recibir y enviar mensajes de la misma forma, incluso los nombres de los topics son iguales, exceptuando la última palabra que corresponde al propio id del dispositivo que actúa.

Por esta razón lo más sencillo es implementar una **clase padre** de Python de la que hereden una misma funcionalidad todos los dispositivos IoT, y únicamente se diferencien en algunos métodos (como en la manera de responder ante un mensaje).

- **¿Cuántas instancias hay de cada componente?**

Hay un único Controlador, RuleEngine y Bridge, junto con un único bot de Discord. En cambio, pueden instanciarse tantos dispositivos como se quieran, siempre y cuando después se comunique al bot los nuevos dispositivos a agregar.

- **¿Controller y Bridge han de ser aplicaciones separadas?, ¿por qué?, ¿qué ventajas tiene una y otra opción?**

Ambas deben ser aplicaciones separadas por facilidad de implementación, ya que Bridge es el que se comunica con la API de Discord, por lo que es el intermediario entre el sistema en sí y la interfaz gráfica. Gracias a ese componente intermedio se puede asegurar que los mensajes entrantes son correctos y no requieren una comprobación previa. Podría verse como una barrera de seguridad.

Realmente, si hablamos en términos de código e implementación, de nuevo la funcionalidad del Bridge podría estar integrada en el Controlador, pero entonces el ámbito de acción de la clase Controlador sería demasiado difuso. Ya no sería un punto de conexión entre los distintos elementos necesarios en el sistema, sinó que más bien él mismo sería todos los elementos en uno.

- **¿Cómo se comunican Controller y Bridge en la opción escogida?**

En nuestro caso, al utilizar MQTT para la comunicación de colas de mensajes entre los dispositivos y el Controlador, hemos decidido utilizarlo también entre el Controlador y el Bridge, ya que es muy sencillo de llevar a cabo y de igual manera requiere esa comunicación asíncrona del sistema.

- **¿Qué tipo de persistencia tiene el sistema?**

Nuestro sistema basa la persistencia en el uso de ficheros (system_rules.txt, iot_devices.txt y data). Se utilizan ficheros como mecanismo más simple que una base de datos, ademas de poder acceder a los datos de inmediato y editarlos de manera sencilla. Los dispositivos se registran en el fichero cuando se inicia su ejecución en la máquina, mientras que el registro de estos en el sistema como tal se hace mediante el RuleEngine y un diccionario que guarda los dispositivos y sus estados en cada momento. El RuleEngine, al ser un atributo del sistema, cuando este guarda sus datos en *data*, los dispositivos quedan registrados en el sistema al volver a iniciarse. Las reglas se registran en el fichero cuando se comunica al bot que se quiere agregar una nueva.

3. Implementación

3.1. Reglas del sistema

En esta sección se define el formato de las reglas del sistema que un usuario puede establecer.

Una regla está compuesta de dos partes; la condición y la acción a realizar.
En nuestro caso, las reglas pueden estar compuestas por más de una condición. estando estas divididas por el operando **“and”** o el comando **“or”**. La acción al contrario solo puede ser una.

A continuación se muestra el formato completo de una regla:

`if` *`condicion`* `and/or` *`condicion`* `and/or` *`condicion`* … `then` *`accion`*

3.2. Mensajes del bot

En esta sección se definen todos los mensajes posibles que pueden usarse para comunicarse con el bot, además de los mensajes de respuesta o formas de reaccionar del bot frente a determinadas situaciones.

Los mensajes posibles son:

`IoT help`
Mostrar el mensaje de ayuda. Informa acerca de los comandos posibles.

`IoT add_rule <rule>`
Agregar una regla al sistema.

`IoT remove_rule <rule>`
Eliminar una regla del sistema.

`IoT list_rules`
Mostrar las reglas del sistema.

`IoT add_device <device_type> <device_id>`
Agregar un dispositivo al sistema.

`IoT remove_device <device_type> <device_id>`
Eliminar un dispositivo del sistema.

`IoT list_devices`
Listar los dispositivos del sistema.

`IoT set_state <device_type> <device_id> <state>`

Establecer el estado a un determinado dispositivo.

`IoT get_state <device_type> <device_id>`

Recoger el estado de un determinado dispositivo.

Cuando un dispositivo o una regla es agregado o eliminado correctamente del sistema, el bot informa de ello con un mensaje satisfactorio, mientras que si el id no es válido para alguna de estas opciones, en el sistema no cambia nada y el bot informa del error.

Cuando se intenta establecer o recoger el estado de un dispositivo, el bot informa con mensaje de error de nuevo si el id no es válido, y en caso contrario responde con el mensaje directamente enviado desde los dispositivos, pasando por el controlador.

Siempre antes de comunicarse con el controlador, es decir, antes de que el mensaje entre en el sistema, el bot con la ayuda del bridge comprueba que el formato del mensaje sea válido, y en caso de no serlo informa con un mensaje de recomendación para mirar los mensajes posibles del sistema.

3.2. Ejecución

Todo lo relacionado con la forma de ejecutar el proyecto, se puede encontrar en la Wiki de gitlab. Aquí puede encontrarse tanto los pasos a seguir para ejecutar el sistema básico, con comunicación desde Discord, como también la resolución de problemas que puedan surgir con los modulos de python y la ejecución de los tests. Asimismo, también se puede encontrar una pequeña guía de ejecución de un entorno virtual.

4. Conclusiones

En lo que respecta a su funcionalidad, hemos logrado implementar un sistema que permite al usuario crear, eliminar y gestionar sus dispositivos inteligentes a través de una interfaz desarrollada utilizando la API de Discord. Además, cada vez que se produce un cambio de estado en alguno de estos dispositivos, se activa una verificación basada en una serie de reglas. Si se satisfacen ciertas condiciones específicas, se ejecutan una serie de acciones que implican cambios de estado en otros dispositivos. Por último, ciertos dispositivos, como los sensores, son capaces de comunicar su estado periódicamente.

En resumen, durante el desarrollo de este proyecto hemos podido abordar diversos desafíos relacionados con la comunicación, los cuales hemos enfrentado de la mejor manera posible.

En nuestra opinión, el sistema que hemos creado cumple con las expectativas del proyecto tanto en términos de rendimiento como de funcionalidad. Además, ha sido un proceso bastante gratificante de desarrollar, especialmente la parte relacionada con el bot de Discord. Aunque esta herramienta forma parte de nuestra rutina diaria, comprender su funcionamiento y las funciones que ofrece la API ha sido un desafío considerable, al igual que el desarrollo del puente. Ha requerido tiempo entender estas funcionalidades y cómo se implementan.