PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN PROGRAMABLE SIMPLE PARA AUTOMATIZACIONES DE DOMÓTICA A TRAVÉS DE UNA PUERTA DE ENLACE TIPO GATEWAY

BENJAMÍN ISMAEL RIVERA VIDAL

Tesis para optar al grado de
Magíster en Ciencias de la Ingeniería

Profesor Supervisor:
MARCOS SEPÚLVEDA FERNÁNDEZ

Santiago de Chile, Enero 2022

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN PROGRAMABLE SIMPLE PARA AUTOMATIZACIONES DE DOMÓTICA A TRAVÉS DE UNA PUERTA DE ENLACE TIPO GATEWAY

BENJAMÍN ISMAEL RIVERA VIDAL

Tesis presentada a la Comisión integrada por los profesores:

MARCOS SEPÚLVEDA

VALERIA HERSKOVIC

ALESSIO BELLINO

JAVIER PEREDA

Para completar las exigencias del grado de
Magíster en Ciencias de la Ingeniería

Santiago de Chile, Enero 2022

A mi madre, a Ignacio, a Giovanni

y especialmente a Dios,

quien me ha traído hasta acá

AGRADECIMIENTOS

Quiero agradecer a las personas que han hecho posible este trabajo. En primer lugar, a mi profesor supervisor Marcos Sepúlveda, por un diferente, pero fructífero semestre de investigación y por recomendarme la temática de esta tesis. A la profesora Valeria Herskovic, por su increíble disposición a siempre entregar feedback, proponer mejoras y su atención al detalle. A César Meneses, compañero de investigación que aprendió, implementó y nos enseñó sobre interfaces de programación visuales, un punto muy relevante de la investigación. Y a Alessio Bellino, cuyo trabajo previo sirvió de inspiración para este proyecto. Igualmente quiero agradecer a Anid/Fondecyt, proyectos número 1211210 y 1200206, que hicieron posible la realización de este trabajo.

Por último, quiero agradecer de forma especial a Giovanni Caro e Ignacio Yousef, quienes me apoyaron y aconsejaron con cariño en los momentos difíciles de este último semestre y en el demandante proceso de tesis.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

RESUMEN

La domótica es el área de la computación que estudia y desarrolla el concepto de automatizaciones para las casas, convirtiéndolas en “casas inteligentes”. Sin embargo, se ha ido quedando atrás en la evolución de la facilidad de su uso, ya que las soluciones actuales se han enfocado en interactuar con los dispositivos de forma directa, pero acotada, o en hacerlo de forma totalmente configurable, pero altamente técnica. Así, aún hace falta un mayor foco en los usuarios, de tal forma que se permita realizar configuraciones programables arbitrariamente complejas sin que esto implique elevar la barrera de entrada.

En este trabajo se presenta una solución de domótica que está diseñada para tener una baja barrera de entrada para principiantes, pero que de igual forma permite realizar configuraciones y automatizaciones simples o complejas. Tiene bajo nivel de configuración *plug-and-play*, está basado en una aplicación Web multiplataforma, con un componente de procesamiento local tipo “gateway”, y con un fuerte enfoque de diseño en la facilidad de uso.

Se plantean además tres casos de uso solucionables a través de domótica, que son implementados a través de la solución propuesta. Se concluye de estos que la solución expuesta en este trabajo permite programar una automatización de domótica partiendo desde cero sin necesitar realizar ninguna instalación o configuración técnica, que es posible programar automatizaciones desde una interfaz visual con baja barrera de entrada o una interfaz de texto con opciones extendidas, y que es posible realizar toda la interacción necesaria a través de una única aplicación Web multiplataforma que no requiere instalación.

Palabras Claves: Domótica, Puerta de enlace, Programable, Internet de las Cosas, Red de Sensores Inalámbricos, Programable

ABSTRACT

Domotics is the area of computing that studies and develops the concept of home automation, turning houses into “smart homes”. However, this area is getting behind in the evolution of its ease to use as current solutions focus on a direct but limited set of features, or are fully configurable but have highly technical interaction with the devices in the domotic network. Thereby, a greater focus on users is still needed, in such a way that allows for arbitrarily complex programmable configurations but without raising the entry barrier.

In this research a domotic solution designed to have a low entry barrier for beginners is presented, which allows both simple and complex automatizations to be configured. It has a low level of plug-and-play initial configuration, it is based on a multiplatform Web application with a local processing component called “gateway”, and has a strong focus on ease of use.

This work also presents three use cases that can be solved using domotics and are implemented with the proposed solution. Based on them, it is concluded that the solution proposed in this thesis allows the user to program a domotic solution starting from scratch without the need of any installation or technical configuration. Also that it is possible to program automatizations with a visual interface with a low entry barrier or through a text interface with extended options, and that this solution allows to have all the necessary interaction with the devices through a single multiplatform Web application that does not require installation.

Keywords: Domotics, Gateway, Programmable, IoT, WSN

1. INTRODUCCIÓN

La domótica es el área de la computación que estudia y desarrolla el concepto de automatizaciones para las casas. Una vivienda que presenta esta clase de automatizaciones puede ser llamada "casa inteligente" o *"smart home",* pues diferentes dispositivos dentro de ella pueden comunicarse y realizar acciones de forma coordinada y congruente sin necesidad de interacción manual (Morais Bolzani et al., 2006). Un sistema de domótica puede monitorear y/o controlar componentes físicos de la casa como luces, sistemas de entretenimiento, electrodomésticos, elementos de seguridad como alarmas y sensores de movimiento, entre otros. También puede incluir componentes digitales como notificaciones, mensajes y conexión a internet (Oliveira et al., 2014). Al conjunto de todos estos elementos se le llamará desde ahora en adelante "red de domótica".

1.1. Motivación

Desde hace años que la computación ha ido cambiando su paradigma desde un enfoque meramente técnico a un enfoque centrado en el humano (Raskin, 1997). Hace 20 años era común necesitar un alto conocimiento técnico para poder utilizar de forma correcta un programa informático. Sin embargo, a lo largo de los años se ha ido cambiando esta mentalidad, lo que se observa en el fuerte énfasis que las aplicaciones y

tecnologías modernas ponen en la experiencia de usuario (UX), interfaz de usuario (UI), usabilidad y en que existan bajas barreras de entrada (Harper et al., 2008).

El problema radica en que el área de la domótica se ha ido quedando atrás en este aspecto ya que, pese a los avances en tecnologías de conexión, interoperabilidad, estandarización y facilidad de uso, las soluciones actuales de domótica se han enfocado en interactuar con los dispositivos de forma directa, pero acotada, o en hacerlo de forma totalmente configurable, pero altamente técnica (Woo & Lim, 2015). Así, aún hace falta un mayor foco en los usuarios, de tal forma que se permita realizar configuraciones programables arbitrariamente complejas sin que esto implique elevar la barrera de entrada, dado que las soluciones existentes que permiten esto no son fáciles de usar por principiantes (Marikyan et al., 2019).

Este trabajo se enmarca dentro de un proyecto de mayor alcance que busca enseñar pensamiento computacional a niños y jóvenes a través de la domótica. Esta tesis en particular se enfoca en la arquitectura e implementación específica de una solución de domótica altamente configurable que permita al usuario programar automatizaciones arbitrariamente complejas, pero sin dejar de ser fácil de usar, con tal de tener una baja barrera de entrada y una poco empinada curva de aprendizaje.

1.2. Hipótesis

En base al contexto descrito surge la siguiente pregunta de investigación: ¿Es posible desarrollar una solución de domótica que permita interactuar con los dispositivos

de forma detallada y profunda a través de un paradigma de programación visual de tal forma de disminuir la barrera de entrada?

A partir de la pregunta anterior y la base de que los lenguajes de programación visual son generalmente más fáciles de utilizar por usuarios nuevos (Dobesova, 2012), se plantea la siguiente hipótesis: Es posible desarrollar una aplicación que permita programar automatizaciones de domótica arbitrariamente complejas y con la profundidad deseada, pero además enfocándose en una baja barrera de entrada al posibilitar la configuración a través del paradigma de programación visual y no necesitar ninguna configuración técnica inicial o instalación.

1.3. Objetivos

El objetivo general de esta tesis consiste entonces en desarrollar una aplicación Web multiplataforma que permita programar automatizaciones de domótica simples o arbitrariamente complejas, sin necesidad de configuraciones técnicas avanzadas ni instalaciones.

En primer lugar, se quiere que sea una aplicación Web para que pueda ser ejecutada en cualquier dispositivo que tenga acceso a un navegador y para que no se necesite instalación para utilizarla. Luego, se busca que no requiera ninguna configuración técnica avanzada para comenzar, por lo que debe poseer la característica de ser *plug-and-play* (Santos et al., 2019). Por último, se quiere permitir realizar configuraciones de domótica simples y complejas, por lo que se busca implementar dos

interfaces de programación: una apuntada a usuarios novatos y otra a usuarios avanzados.

Se describen a continuación los objetivos específicos que esta tesis desarrolla en pos del objetivo general.

1.3.1. Implementación de hardware gateway

El primer objetivo parcial corresponde a la implementación de un componente de hardware llamado *"Gateway"* que es el centro de la arquitectura que se busca lograr para cumplir el objetivo general de este proyecto. Este componente está encargado de unificar la conexión con todos los dispositivos de la red de domótica, es decir, actuar como una "puerta de entrada" que procese de forma local las señales de la red del usuario.

Además, este gateway debe estar conectado a internet de tal forma de poder interactuar con él de forma bidireccional a través de una página Web. Sumado a esto, necesita implementar un componente de traducción entre protocolos, ya que los dispositivos de domótica se comunican a través de protocolos livianos como Zigbee (Ergen, 2004), mientras que la interacción con la página Web debe ser a través de un protocolo robusto como WebSocket basado en TCP (Fette & Melnikov, 2011).

1.3.2. Aplicación Web multiplataforma

Como segundo objetivo parcial, se busca desarrollar una aplicación Web que permita interactuar con la red de domótica a través del gateway desde cualquier lugar de

internet. De igual forma, se busca que esta aplicación pueda ser ejecutada indistintamente en un computador o en un smartphone con una experiencia de usuario similar.

1.3.3. Plug-and-play con configuración simple y avanzada

Por último, como tercer objetivo específico, se busca lograr una arquitectura totalmente plug-and-play. Primero, a nivel físico, es decir, que no requiera ninguna instalación de hardware adicional, sino que sea solamente necesario conectar a la red eléctrica y de internet, y ya esté listo para usarse. Y segundo, a nivel de aplicación, es decir, que no requiera ninguna configuración avanzada, sino que solamente se necesite registrar un dispositivo y ya pueda comenzar a utilizarse.

Sin embargo, a pesar de lo anterior, se busca que este bajo requerimiento de configuración inicial no afecte el nivel de profundidad real de configuración al que se puede llegar. Así, se busca implementar una interfaz de programación de automatizaciones con baja barrera de entrada al tener una cantidad limitada de opciones, pero también implementar una segunda interfaz de programación que permita realizar configuraciones arbitrariamente complejas o profundas.

1.4. Trabajo relacionado

En los últimos años ha habido una serie de trabajos que se han ido enfocando en construir redes de sensores inalámbricos (*Wireless Sensor Network*, WSN) cada vez más

robustas y de bajo consumo de batería (Gupta & Gupta, 2019). Lo anterior cobra gran importancia al considerar que un porcentaje alto de los dispositivos de domótica funcionan con batería y, por lo tanto, se desea que éstas duren la mayor cantidad de tiempo posible (Govindan et al., 2019). Así, se han desarrollado igualmente una serie de protocolos de comunicación como Z-Wave, Zigbee, RFID, UWB, entre otros, que buscan optimizar este consumo siendo lo más robusto posible, es decir, asegurando el correcto envío y recepción de cada señal y mensaje (Pei et al., 2008).

Se presentan a continuación tres áreas de desarrollo de literatura previa. Una primera área relacionada a la arquitectura física y protocolos utilizados en el *Gateway*, una segunda área relacionada al enfoque de la domótica en la interacción con el ser humano, y una tercera área relacionada a la barrera de entrada a la domótica y la facilidad de uso para novatos.

1.4.1. Gateway

Algunos trabajos presentan arquitecturas que solucionan problemas similares al enfrentado en este proyecto, por ejemplo, en la intercomunicación entre dispositivos. Sin embargo, no llegan al nivel de implementación, quedando en una propuesta teórica. Una de estas propuestas es una arquitectura de red orientada a servicios de 3 capas (dispositivos, servicios, y aplicación), en la que la capa de servicios disponibiliza de

forma estandarizada lo que puede entregar cada dispositivo, es decir, sus “servicios” (Aziz, 2013).

Otros trabajos plantean la importancia de una conexión a internet. En específico, el permitir una interacción desde cualquier lugar con conexión y no necesitar estar físicamente cerca de los dispositivos. Esto abre una nueva dimensión de interacción con los dispositivos de domótica, ya no solo limitándose a una red cerrada local, sino que pudiendo extender este concepto a un rango teóricamente ilimitado (Perumal et al., 2008; Rusli & Dianati, 2012).

Un importante aporte de los trabajos que plantean una apertura a internet es hacer visible la necesidad de no estar restringidos por la distancia física al momento de interactuar con una red de domótica. Sin embargo, la gran limitación que tienen estos trabajos es que para lograr esta apertura a la red necesitan usar el protocolo TCP para comunicarse con los dispositivos, un estándar exigente en procesamiento que puede dificultar el uso de dispositivos que funcionan con batería, un porcentaje alto de los componentes de domótica (Govindan et al., 2019).

Más recientemente, otros trabajos plantean un modelo basado en 2 protocolos distintos que interactúan a través un *middleware* (componente intermediario) físico tipo *gateway* (Serdaroglu & Baydere, 2016). Para los dispositivos de domótica es deseable un protocolo liviano como 6LoWPAN o equivalente de bajo consumo de batería, pero para lograr esto se necesita sacrificar parte de la robustez de comunicación (Pei et al., 2008). Este tipo de protocolo resulta entonces útil para comunicación liviana de corta

distancia, pero no es adecuado para una comunicación de larga distancia propensa a pérdidas de paquetes de información como lo es internet (Borella et al., 1998). Por otro lado, para la salida a internet se usa un protocolo como TCP que asegura robustez, pero implica un alto costo de procesamiento por el peso de los mensajes. De esta forma, resulta útil para comunicación a través de medios de alta pérdida, pero no para dispositivos de bajo poder de procesamiento y dependientes de batería como los sensores y actuadores de domótica. Así, el uso de un componente intermedio que traduzca entre dos protocolos permite aprovechar los beneficios de cada uno en su contexto óptimo y no estar obligados a usarlos en donde no son eficientes.

Un último beneficio que también se plantea del uso de un gateway es que al haber un componente de procesamiento local, esto permite que la red de dispositivos continúe funcionando aunque no haya una conexión permanente a internet (Bonino et al., 2008).

1.4.2. Enfoque centrado en los humanos

Por otro lado, el área de la domótica también se ha desarrollado en cuanto a la interacción con humanos, cambiando de un enfoque centrado en la tecnología a uno centrado en el usuario (Singh et al., 2020). En particular, hay trabajos que se han enfocado en una interacción amigable con los dispositivos a través de interfaces visuales. Por ejemplo, centradas en la visualización online de datos locales de forma unidireccional local - en línea (Menéndez et al., 2020) o en una interacción completa

programable bidireccional a través de una aplicación de escritorio (Aguilar & Echeverría, 2020).

Lo anterior ha demostrado que efectivamente es posible interactuar con dispositivos de domótica a través de interfaces conceptuales y no solamente de texto. Sin embargo, estos trabajos presentan igualmente limitaciones relevantes como una interacción solamente unidireccional a través de internet o bidireccional, pero solo a través de una aplicación local de escritorio. Incluso cuando se logra una interacción bidireccional a través de internet, igualmente se requiere algún tipo de instalación, configuración técnica o hardware extra (Kiran et al., 2021).

1.4.3. Automatización sin necesidad de configuración

Por último, una tercera área de investigación de la domótica que resulta interesante para este trabajo trata sobre la facilidad de instalación inicial de una solución. Dicho de otra forma, qué tan fácil o difícil es lograr, a través de cierta solución de domótica, programar una automatización partiendo totalmente desde cero. Es deseable entonces lograr implementaciones que requieran muy bajo nivel de configuración inicial, de tal forma de ser amigables con usuarios nuevos y no implicar una alta barrera de entrada que traiga consigo resistencia al cambio o rechazo por desconocimiento (Marikyan et al., 2019).

En este ámbito, algunos trabajos han logrado generar una implementación que permite añadir dispositivos a una red de forma totalmente directa sin necesitar ninguna

configuración previa (Matthys et al., 2015) e incluso, además de eso, permitir interactuar con ellos a través de una aplicación Web amigable que abstrae toda la complejidad de la implementación de bajo nivel y tampoco requiere ninguna configuración (Santamaria et al., 2014). Con esto se ha logrado una experiencia sin necesidad de configuración que puede entenderse como "conectar y está listo" o "*plug-and-play*".

La limitación de los dos trabajos anteriores es enfocarse en una interacción directa con los dispositivos y no en una arbitrariamente compleja. Para lograr un inicio sin necesidad de configuración y una programación que se abstrae de lo técnico, sacrifica parte de la profundidad de interacción que se puede lograr.

1.5. Metodología

Se presenta a continuación, de forma secuencial, la metodología de investigación utilizada para lograr los objetivos definidos y comprobar el cumplimiento de la hipótesis planteada en este proyecto a través de la implementación de una solución de domótica.

1.5.1. Revisión de literatura

En base a la literatura científica y el estado del arte revisados en la sección 1.4, se determinó para este trabajo que es deseable plantear una solución que presente las siguientes características:

- Que esté separada en capas de abstracción, en la que cada capa solo interactúe con los "servicios" que exponen las otras capas. De esta manera, se separan las responsabilidades de procesamiento y se puede utilizar distintas tecnologías y protocolos de comunicación según lo que sea más conveniente (Aziz, 2013).
- Que utilice un componente intermediario tipo "Gateway" conectado a internet que comunique a los dispositivos con los usuarios, de tal forma de poder utilizar un protocolo para la red interna de dispositivos y otro protocolo distinto para la red externa de internet, y aprovechar así los beneficios de ambas en sus contextos óptimos (Serdaroglu & Baydere, 2016).
- Que presente una muy baja barrera de entrada para usuarios nuevos, necesitando poca o ninguna configuración inicial (Matthys et al., 2015; Santamaria et al., 2014).
- Que implemente una interacción amigable con el usuario, sin exigir configuraciones avanzadas o alto conocimiento técnico (Aguilar & Echeverría, 2020), pero que igualmente permita realizar configuraciones arbitrariamente complejas o profundas.

1.5.2. Arquitectura propuesta

Así, teniendo en consideración los puntos deseables anteriores, en este trabajo se plantea una implementación específica de una arquitectura orientada a servicios (Aziz, 2013) que es accesible desde cualquier parte de internet (Perumal et al., 2008; Rusli &

Dianati, 2012), pero no utiliza el protocolo TCP para comunicarse con los dispositivos, sino que usa un doble protocolo: Zigbee liviano de bajo consumo de batería (Salam et al., 2019) para comunicarse con la red de dispositivos y WebSocket TCP robusto para comunicarse con internet. La traducción entre ambos está hecha a través de un *middleware gateway* (Serdaroglu & Baydere, 2016) con una interacción amigable con el usuario (Aguilar & Echeverría, 2020; Menéndez et al., 2020), extendiendo esta idea para soportar cualquier dispositivo compatible con el protocolo Zigbee y permitir una interacción bidireccional a través de una aplicación Web multiplataforma. Por último, presenta la característica de ser *plug-and-play* (Matthys et al., 2015; Santamaria et al., 2014), pero sin limitar el nivel de configuración posible, ya que esta solución permite realizar automatizaciones arbitrariamente complejas.

Específicamente, para la realización de este trabajo se implementó una arquitectura separada en tres capas de abstracción: una capa WSN de bajo nivel en donde se encuentran todos los dispositivos de domótica, una capa de aplicación de alto nivel en donde los usuarios interactúan con los dispositivos, y una capa intermedia *gateway* que traduce entre la capa de aplicación y WSN, y funciona como un servidor de procesamiento local para las señales de los dispositivos de la red.

Esta arquitectura de tres capas presenta un total de cuatro componentes: un "servidor local" y un módulo de traducción entre la red interna WSN y la red externa de internet en la capa gateway; y un servidor central de procesamiento y un cliente Web multiplataforma en la capa de aplicación. A esto se le suman los dispositivos de

domótica de la capa WSN, formando así las tres capas del modelo. Se muestra en la Figura 1 un diagrama de la arquitectura en el que se observan los cuatro componentes mencionados.

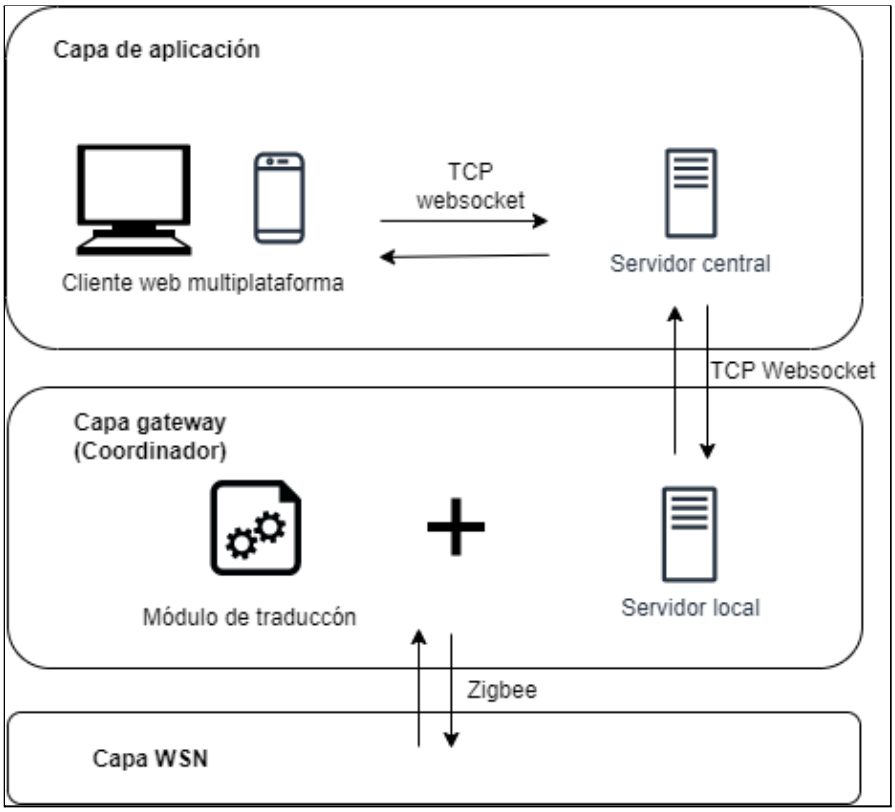


Figura 1: Diagrama arquitectura de 3 capas con énfasis en los 4 componentes del modelo

1.5.3. Implementación de arquitectura

Se presenta a continuación el detalle técnico de cada una de cada uno de los cuatro componentes que forman la arquitectura propuesta.

a) Servidor local

Para el cumplimiento del primer objetivo específico del proyecto se implementó un componente de "servidor local" que interactúa directamente con los dispositivos en la capa WSN. Tiene como función principal procesar de forma local los mensajes recibidos de los dispositivos, es decir, ejecutar toda la lógica que el usuario configuró al momento de crear una automatización.

A nivel físico está constituido por dos elementos: una Raspberry Pi 3 con sistema operativo Raspberry Pi OS buster que funciona como hardware principal y un receptor de señales Zigbee tipo CC2652 conectado por USB. Al conjunto de estas dos partes se le denominó "coordinador".

Por encima del hardware, se implementó un servidor en nodeJS (Tilkov & Vinoski, 2010) programado en lenguaje JavaScript. Este proceso corriendo en la Raspberry escucha los mensajes que los diferentes dispositivos emiten, guarda información contextual de ellos como tiempos de respuesta y calidad de conexión, procesa los mensajes tomando decisiones según las instrucciones que fueron programadas por el usuario y, según éstas, emite respuestas para que los actuadores de la red ejecuten las instrucciones especificadas.

Todo el procesamiento anterior ocurre de forma local en el coordinador y no necesita interactuar en ningún momento con la red de internet. Sin embargo, a pesar de lo anterior, el Coordinador sí está conectado a internet a través de un módulo aparte, de

tal forma de poder comunicarse con el componente de servidor central de forma bidireccional a través del protocolo WebSocket.

b) Módulo de traducción

También como parte de la implementación del gateway para cumplir el primer objetivo del proyecto, hay un segundo proceso que está corriendo constantemente en el coordinador y es aquel que está encargado de traducir mensajes entre los dispositivos de la capa WSN y las instrucciones que el usuario configura en la capa de aplicación.

La capa WSN utiliza el protocolo Zigbee liviano mientras que la capa de aplicación utiliza el protocolo WebSocket robusto y bidireccional. La traducción de mensajes entre ambos protocolos es realizada por el software Zigbee2MQTT (Koenkk, November 2021) que transforma señales Zigbee en mensajes MQTT construidos sobre TCP, compatibles con el protocolo WebSocket (Naik, 2017).

c) Servidor central

Se implementó igualmente en nodeJS un servidor corriendo en los servicios en línea de Amazon Web Services (AWS), lo que le permite estar disponible desde cualquier dispositivo conectado a internet (Baron & Kotecha, 2013). Este servidor presenta dos características relevantes para la arquitectura de la solución. En primer lugar, almacena toda la información contextual relevante de los distintos dispositivos de domótica permitidos en el sistema, de tal forma de poder realizar de forma automática toda la configuración técnica necesaria cuando una persona añade un dispositivo a su red. Así, el usuario solo ve que conectó el dispositivo y ya está listo para usarse. En

segundo lugar, este servidor central también construye de forma dinámica toda la implementación técnica necesaria cuando una persona programa una automatización de domótica. De esta forma, para el caso de configuración simple, el usuario solo ve una interfaz de programación de opciones concretas y reducidas, siendo que realmente por detrás el servidor arma toda la implementación de bajo nivel para que esta configuración funcione. Si el usuario quiere realizar una configuración más profunda, puede utilizar otra interfaz de programación que se salta este paso de transformación y así puede acceder a opciones que en la interfaz simple no están disponibles.

Este tercer componente de la arquitectura está apuntado a cumplir el tercer objetivo del proyecto: que la solución sea completamente *plug-and-play*, pero que permita igualmente realizar configuraciones simples y avanzadas, sin reducir la profundidad de la programación que se puede lograr.

d) Cliente Web

Por último, un cuarto componente de la arquitectura es una aplicación Web multiplataforma con la que interactúan directamente los usuarios. Esta aplicación está construida en JavaScript reactJS (Fedosejev, 2015) e igualmente corre en AWS, por lo que puede ser accedida indistintamente desde un computador o un smartphone. Se comunica solamente con el servidor central y es el medio unificado que tiene un usuario para interactuar con los dispositivos de su red.

Tiene dos funciones principales. En primer lugar, administrar los dispositivos de la red del usuario, pudiendo añadir, eliminar, ver el estado o cambiar la configuración de

estos. Y en segundo lugar, permitirle al usuario programar la automatización que su red ejecutará.

En cuanto al segundo punto de creación de automatizaciones, esta aplicación Web ofrece dos opciones. La primera es una interfaz de programación visual basada en Google Blockly (Pasternak et al., 2017) que expone una cantidad reducida de "bloques" que representan lecturas de sensores, acciones ejecutables en los actuadores, o reglas de control de flujo. Así, el usuario crea su automatización uniendo bloques para armar la lógica deseada. La segunda opción es una interfaz de programación de texto en la que el usuario puede acceder a todas las funcionalidades de JavaScript sin estar restringido por bloques concretos predefinidos.

Este cuarto componente de la arquitectura apunta al cumplimiento del segundo y tercer objetivo de este proyecto: que la solución pueda ser utilizada desde múltiples plataformas, pero que esto no implique reducir la libertad de configuración posible.

1.5.4. Comprobación a través de casos de uso

Como último paso de la metodología aplicada en este trabajo está la comprobación de los objetivos planteados. Esto se llevó a cabo a través del análisis de tres casos de uso concretos en donde se pudieran apreciar si se cumplieron o no las características deseadas de esta solución y, por lo tanto, si se cumple o no el objetivo

general y los objetivos específicos de este proyecto. Se presentan a continuación los casos de uso desarrollados:

a) Aire acondicionado con múltiples sensores

Se tiene la siguiente situación: "En una habitación de una puerta y una ventana, se quiere hacer uso eficiente del aire acondicionado, teniéndolo encendido solo cuando hay una persona adentro y está la puerta y la ventana cerrada". Para implementar este caso se necesita entonces un enchufe inteligente que pueda encender y apagar el aire, un sensor de movimiento que detecte la presencia de una persona, y dos sensores de proximidad (uno en la puerta y otro en la ventana) que detecten si están abiertas o cerradas. Se busca comprobar a través de este caso de uso si efectivamente la aplicación es *plug-and-play*, pues se necesitan añadir un total de cuatro dispositivos más el coordinadir para poder programar la automatización.

b) Alarma desactivable con variables

Se tiene la siguiente situación: "Se quiere recibir una notificación cuando alguien abre una puerta que debería estar cerrada (por ejemplo, la puerta de una habitación cuando el dueño no está). Sin embargo, no se quiere que se active esta "alarma" cuando el mismo dueño es el que abre la puerta, por lo que se desea poder 'activar' o 'desactivar' esta funcionalidad a voluntad". Este segundo caso de uso se desea configurar a través de un smartphone, por lo que se busca comprobar si efectivamente la solución propuesta es multiplataforma. Además, este caso implica guardar un estado local de la alarma ('activada' o 'desactivada') por lo que se busca comprobar si

efectivamente la implementación de gateway realiza el procesamiento local sin necesidad de conectarse a internet.

c) **Café por la mañana con configuración avanzada**

Por último, el tercer caso de uso plantea la siguiente situación: “Se quiere prender el hervidor de agua durante 3 minutos de forma automática cuando amanezca para tomarse un café al despertar, pero solamente de lunes a viernes”. No existe actualmente un bloque en la interfaz de programación visual que cambie de estado dependiendo de si hoy es o no día de semana, por lo que para este caso se necesita programar la automatización directamente en JavaScript en la interfaz de texto de la aplicación. Se busca comprobar a través de este caso de uso si se cumple el objetivo de permitir configuraciones avanzadas arbitrariamente complejas, no limitadas por los bloques previamente definidos en la interfaz visual. Es importante mencionar que la situación planteada en este caso de uso busca mostrar que a través de la interfaz de texto es posible implementar funcionalidades que no se pueden hacer con los bloques existentes en ese momento del tiempo en la interfaz visual. Para este caso particular, los bloques necesarios podrían haber sido creados y puestos a disposición con anterioridad, pero no resulta factible pensar que toda posible situación que un usuario quiera configurar, haya sido pensada e implementada previamente en bloques por los autores del proyecto. Así, se hace necesario tener la libertad de poder programar automatizaciones no limitadas por los bloques existentes.

1.6. Resultados

Una vez implementada la arquitectura descrita en el punto 1.5.3, se construyeron a través de la plataforma los tres casos de uso del punto 1.5.4 con el fin de comprobar la correcta completitud de los objetivos planteados y la hipótesis propuesta en este trabajo. Se detallan a continuación los resultados obtenidos.

1.6.1. Plug-and-play

Para los tres casos de uso descritos anteriormente se pudo observar la característica de plug-and-play de la plataforma, sin embargo, se hace especialmente notorio para el caso del "aire acondicionado con múltiples sensores", ya que se necesitaba un mayor número de dispositivos.

Para construir este ejemplo, fueron necesarios los siguientes pasos: 1) Conectar el Coordinador a la red eléctrica y a internet; 2) ingresar a la aplicación Web y crearse una cuenta (o entrar con una creada previamente); 3) registrar el identificador único de 6 caracteres del coordinador y de los 4 dispositivos que se encuentra impreso en cada uno a través del panel de administración, como se observa en la Figura 6; 4) acceder a la opción de programación visual y construir en base a los bloques existentes la lógica deseada para la automatización, como se observa en la Figura 2; y 5) guardar la configuración.

Se observó entonces que, realizando el procedimiento anterior, un usuario puede tener corriendo en su hogar una automatización programada por él mismo, sin necesidad de hardware extra, instalaciones o configuraciones técnicas, solamente conectar y usar, es decir, una experiencia *plug-and-play*.

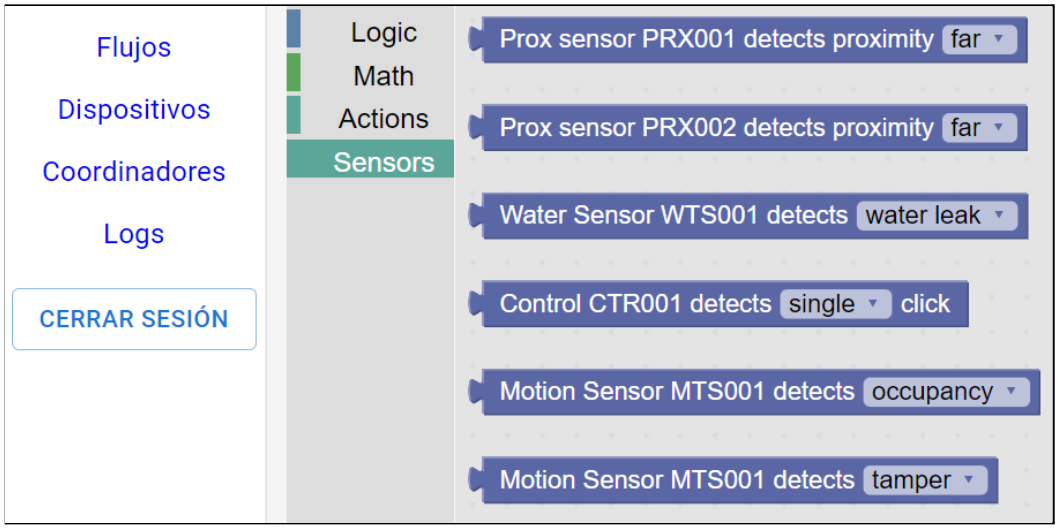


Figura 2: Bloques tipo “Sensors” (sección abierta) y “Logic”, “Math” y “Actions” (secciones cerradas) disponibles en la interfaz de programación visual. Estos bloques se utilizan dentro de un componente de lógica “when <señal de sensor>, do <acción>”

1.6.2. Multiplataforma y procesamiento local

El caso de uso “alarma desactivable con variables” fue construido a través de la interfaz visual de la plataforma utilizando un dispositivo móvil. En específico, se utilizó el modelo Xiaomi Redmi 8T, Android 9, con navegador Google Chrome en sentido horizontal como se muestra en la Figura 3. Se observó entonces que, aunque la disposición de opciones y responsividad no fuera óptima, efectivamente se pudo

configurar una automatización de domótica desde un *smartphone*. Por lo tanto, considerando el caso de uso anterior y el aquí descrito, se obtuvo que la aplicación sí es multiplataforma.

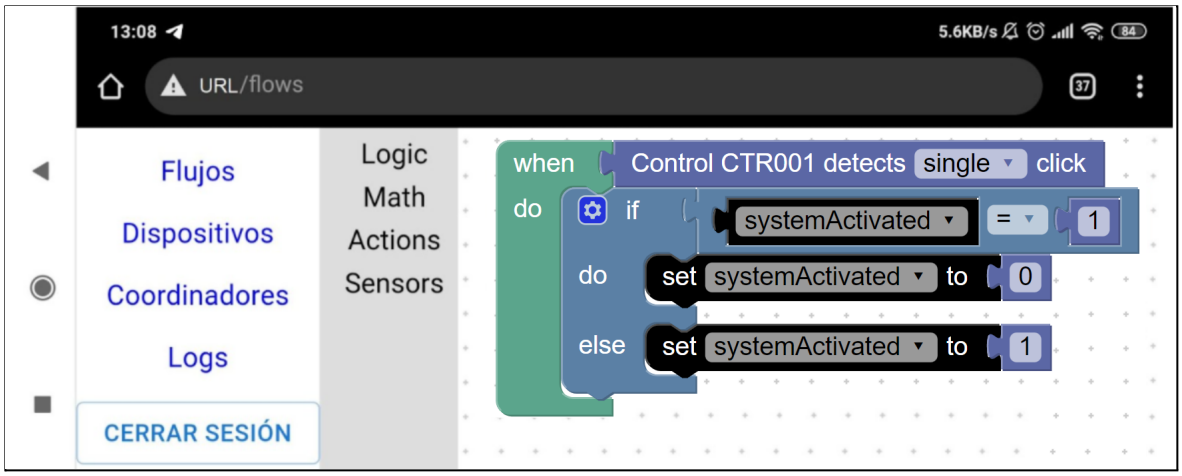


Figura 3: Lógica manejo de estados para caso de uso “Alarma desactivable con estados” construida en un Xiaomi Redmi Note 8T, Android 9, Google Chrome en horizontal

Además de lo anterior, este caso de uso implica manejar una variable de estado de la alarma: ‘activada’ o ‘desactivada’. El estado de esta variable no es manejado por el servidor central, sino por el coordinador de forma local, específicamente guardado en su memoria. Se observó entonces que el coordinador efectivamente realizó un procesamiento local que le permitió manejar estados de variables propias, como se observa en el historial de estados (*logs*) que el coordinador generó, mostrados en la Figura 4.

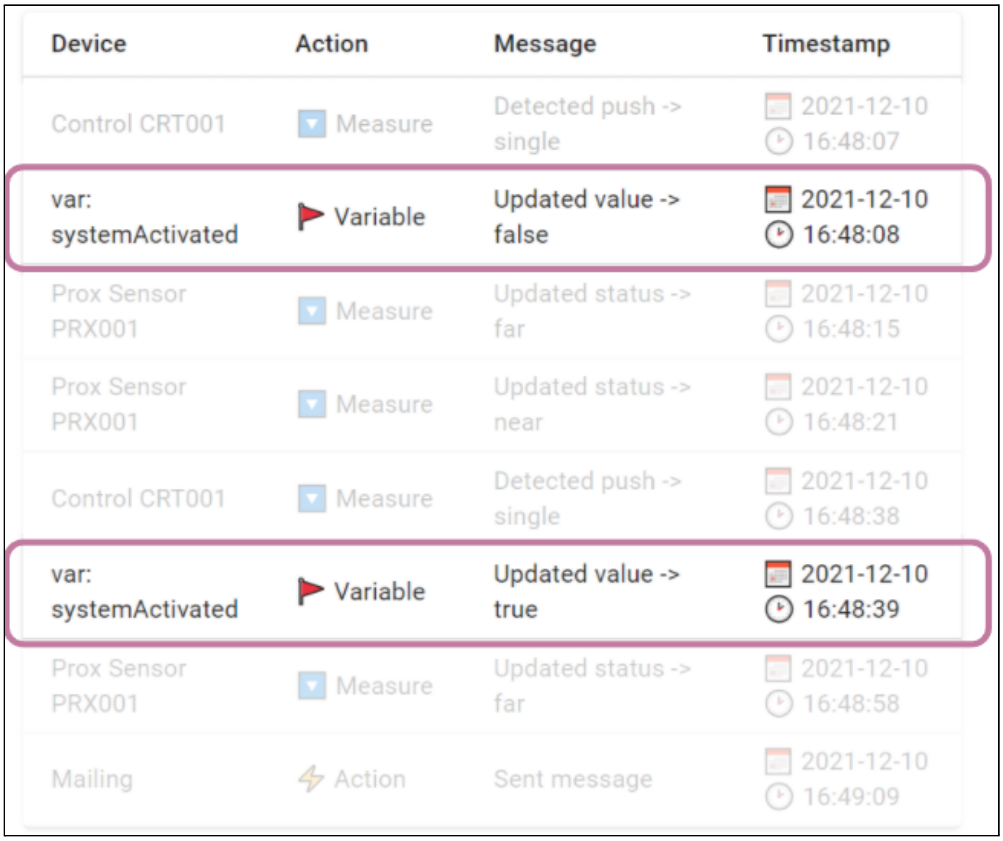


Figura 4: Historial de estados (logs) generados por el caso de uso "alarma desactivable con variables" con las actualizaciones del estado de la variable utilizada marcados en rosado

1.6.3. Automatizaciones a través de interfaz visual y de texto

Se puede observar en los dos casos de uso anteriores que es posible programar una automatización de domótica partiendo desde cero, sin necesitar en ningún momento entrar en una configuración técnica, sino que puede simplemente usarse la interfaz de programación visual. Sin embargo, pueden existir momentos en los que se busque

realizar una configuración más avanzada o implementar funcionalidades que no están disponibles por defecto en los bloques.

Así, por ejemplo, podemos observar que para el caso de uso "Café por la mañana con configuración avanzada", se quiso usar una funcionalidad extra que pudiera determinar si el día actual es día de semana o no. Esta funcionalidad debió ser implementada a través de la interfaz de texto escribiendo directamente código JavaScript como se muestra en la Figura 14. Se observó entonces que la aplicación permitió realizar una configuración avanzada arbitrariamente compleja (solo limitada por la capacidad misma de JavaScript) sin reducir la libertad de acción como en el caso de la interfaz de programación visual. Se tienen entonces las dos opciones posibles: configuración simple limitada o configuración avanzada técnica.

1.7. Artículo científico

El detalle técnico de la implementación y discusión de resultados específicos obtenidos de cada caso de uso mencionado se encuentra plasmado en el artículo científico "SIPGate: Simple Programmable Gateway Application for Wireless Sensor Networks" presentado en el siguiente capítulo. Luego, en el capítulo 3 se presentan las principales conclusiones y limitaciones encontradas, además de plantear posibles opciones de trabajo futuro.

2. SIPGATE: SIMPLE PROGRAMMABLE GATEWAY APPLICATION FOR WIRELESS SENSOR NETWORKS

2.1. Introducción

El área de domótica estudia la computación y electrónica aplicada a las casas, convirtiéndolas en *casas inteligentes (Morais Bolzani et al., 2006)*. Esto permite, por ejemplo, poder apagar las luces desde el celular o configurar que se enciendan automáticamente cuando anochezca, ver de forma remota la cámara de la puerta frontal o recibir notificaciones al celular cuando se detecta movimiento en la sala de estar y no estamos nosotros en casa (Oliveira et al., 2014). Las funcionalidades anteriores se llevan a cabo a través de sensores (dispositivos que obtienen datos del mundo real) y actuadores (dispositivos que ejecutan acciones), los que forman una red que no tiene la necesidad de una infraestructura rígida (Jangra, 2010).

Desde hace años que la computación ha ido cambiando su paradigma desde un enfoque centrado en la tecnología a un enfoque centrado en el humano (Raskin, 1997). Esto se puede observar en el fuerte énfasis que las aplicaciones y tecnologías modernas ponen en la experiencia de usuario (UX), interfaz de usuario (UI), usabilidad, intuitividad, bajas barreras de entrada y poco empinadas curvas de aprendizaje (Harper et al., 2008). Sin embargo, en cuanto a soluciones de domótica, aún hace falta un mayor foco en los usuarios, dado que las soluciones existentes no son fáciles de usar por principiantes (Marikyan et al., 2019).

Pese a los avances en tecnologías de conexión, interoperabilidad, estandarización y facilidad de uso, las soluciones actuales se han enfocado en interactuar con los dispositivos de forma muy directa, pero acotada, o en hacerlo de forma totalmente configurable, pero altamente técnica (Woo & Lim, 2015). Existen soluciones como los paneles de administración propietarios de Sonoff (disponible en https://sonoff.tech/) o My Touch Smart (disponible en https://mytouchsmart.com/) que ofrecen una experiencia de usuario muy amigable y conectada a internet, pero reducen la libertad de acción solo a interacciones directas simples. Existen otras soluciones como Home Assistant (disponible en https://www.home-assistant.io/) o Node-Red (disponible en https://nodered.org/) que permiten total libertad de configuración, pero requieren altos niveles de configuración, tienen empinadas curvas de aprendizaje o no permiten la conexión con internet. Y existen otras soluciones como Zigbee2MQTT (Koenkk, November 2021) que intentan atacar ambos problemas simultáneamente, pero de igual forma requieren configuración de hardware externo y conocimiento técnico avanzado para poder iniciar la primera vez. Por lo tanto, no existe actualmente una solución enfocada en la experiencia de usuario, que pueda ser usada por principiantes, pero a la vez sea altamente configurable. Así, el presente artículo plantea una implementación con estas características deseadas.

Este trabajo se enmarca dentro de un proyecto de mayor alcance que busca enseñar pensamiento computacional a niños y jóvenes a través de la domótica. El presente artículo se organiza de la siguiente forma. La sección 2 revisa los trabajos

relacionados en esta área, en particular, en cuanto a la usabilidad y flexibilidad de cada plataforma. La sección 3 presenta en detalle la arquitectura y funcionamiento de la solución propuesta. La sección 4 expone tres casos de uso que fueron construidos en base a esta solución. La sección 5 discute los resultados obtenidos y las limitaciones de esta propuesta. Finalmente, la sección 6 concluye el aporte de nuestra solución y plantea el trabajo futuro que se considera relevante.

2.2. Trabajos relacionados

En los últimos años ha habido una serie de trabajos que se han ido enfocando en construir Redes de Sensores Inalámbricos (*Wireless Sensor Networks*, WSN) cada vez más robustas, con bajo consumo de batería y aprovechando nuevos protocolos de comunicación (Pei et al., 2008).

Algunos trabajos presentan arquitecturas que solucionan problemas similares al enfrentado en este proyecto, por ejemplo, en la intercomunicación entre dispositivos. Sin embargo, no llegan al nivel de implementación, quedando en una propuesta teórica. Una de estas propuestas es una arquitectura de red orientada a servicios de 3 capas (dispositivos, servicios, y aplicación) en la que la capa de servicios disponibiliza de forma estandarizada lo que puede entregar cada dispositivo, es decir, sus “servicios” (Aziz, 2013).

Algunos trabajos plantean el abrir una red WSN a internet para permitir la interacción desde cualquier lugar con conexión y no estar restringidos por la necesidad de estar físicamente cerca de los dispositivos (Perumal et al., 2008; Rusli & Dianati, 2012). Sin embargo, tienen la limitación de que para lograr esto necesitan usar el protocolo TCP para comunicarse con cada dispositivo de forma particular, un estándar exigente en procesamiento que puede dificultar el uso de dispositivos que funcionan con batería, un porcentaje alto de los componentes de domótica (Govindan et al., 2019). Más recientemente, se planteó un modelo basado en dos protocolos distintos que interactúan a través un middleware físico tipo gateway. Para los dispositivos se usa un protocolo liviano como 6LoWPAN de bajo consumo de batería y para la salida a internet se usa un protocolo robusto como TCP. Además, al haber un gateway de procesamiento local, esto permite que la red de dispositivos continúe funcionando aunque no haya una conexión permanente a internet (Bonino et al., 2008; Serdaroglu & Baydere, 2016).

Se ha trabajado no solo en la conectividad externa de la red, sino también en lograr que las redes sean *plug-and-play* y *ready-to-use*, es decir, modelos que no requieran altos niveles de configuración previa para comenzar (Santamaria et al., 2014). Sin embargo, estos trabajos se enfocan en la interacción directa con los dispositivos y no permiten configurar automatizaciones arbitrariamente complejas.

La necesidad de una interfaz de usuario amigable también ha sido foco de la investigación en esta área, cambiando desde un enfoque centrado en la tecnología a uno centrado en el usuario (Singh et al., 2020). Por ejemplo, este enfoque ha sido usado para

lograr la visualización en línea de datos locales de forma unidireccional local - en línea (Menéndez et al., 2020).

Otros trabajos toman la misma idea del *middleware gateway* e implementan una interfaz de usuario amigable que permite una interacción bidireccional. Sin embargo, se limitan únicamente a una aplicación de escritorio, no a una interacción Web desde cualquier tipo de dispositivo (Aguilar & Echeverría, 2020) o requieren igualmente algún tipo de instalación y hardware extra (Kiran et al., 2021).

Por último, se ha trabajado también sobre la posibilidad de utilizar la cámara de un celular con reconocimiento de imágenes como sensor de cambios de estados más complejos no fácilmente modelables con sensores clásicos (por ejemplo, saber si un enchufe está ocupado o si un paragüas está o no reposado al lado de la puerta) (Bellino, 2016). Esto aporta en la complejidad soportada de cambios de estado, pero sólo puede detectar cambios a través del celular, y asimismo, sólo puede reaccionar mediante el celular, sin, por ejemplo, poder provocar cambios físicos en el ambiente.

En base a lo anterior, en este trabajo se plantea SIPGate, una implementación específica de una arquitectura orientada a servicios (Aziz, 2013) que es accesible desde cualquier parte de internet (Perumal et al., 2008; Rusli & Dianati, 2012), pero no utiliza el protocolo TCP para comunicarse con los dispositivos, sino que usa un doble protocolo: Zigbee liviano de bajo consumo de batería (Salam et al., 2019) para comunicarse con los dispositivos y TCP robusto para comunicarse con internet. La traducción entre ambos está hecha a través de un *middleware gateway* (Serdaroglu &

Baydere, 2016), con una interacción amigable (Aguilar & Echeverría, 2020; Bellino, 2016; Menéndez et al., 2020), extendiendo esta idea para soportar cualquier dispositivo compatible con el protocolo Zigbee y permitir una interacción bidireccional a través de una aplicación Web multiplataforma. Por último, presenta la característica de ser *plug-and-play* (Santamaria et al., 2014) al solo necesitar registrar un código alfanumérico de 6 caracteres impreso en cada dispositivo para comenzar a usarlo, pero esto sin limitar el nivel de configuración posible, ya que nuestra solución permite realizar automatizaciones arbitrariamente complejas.

2.3. SIPGate: un gateway simple programable para redes de sensores inalámbricos

Nuestra propuesta, llamada SIPGate (*Simple Programmable Gateway*, Puerta de enlace Programable Simple) es una solución de domótica programable enfocada en la simpleza de uso, basada en la arquitectura orientada a servicios, implementada a través de un *middleware* físico tipo *gateway*. Se describe a continuación la arquitectura construida.

2.3.1 Arquitectura

En total existen 3 capas de abstracción: una capa de la aplicación, donde el usuario interactúa con la plataforma; una capa WSN, que maneja todos los dispositivos físicos; y una capa intermedia Gateway, que comunica las dos anteriores. Esta capa

situada en el punto medio de la arquitectura cumple un doble propósito: por un lado provee un servidor local de procesamiento y almacenamiento de información de los diferentes dispositivos y, por el otro, funciona como una capa de traducción que permite que interactúen los protocolos Web externos de la capa de aplicación con los protocolos internos de la WSN. La Figura 5 muestra la arquitectura general de SIPGate con enfoque en la separación de capas.

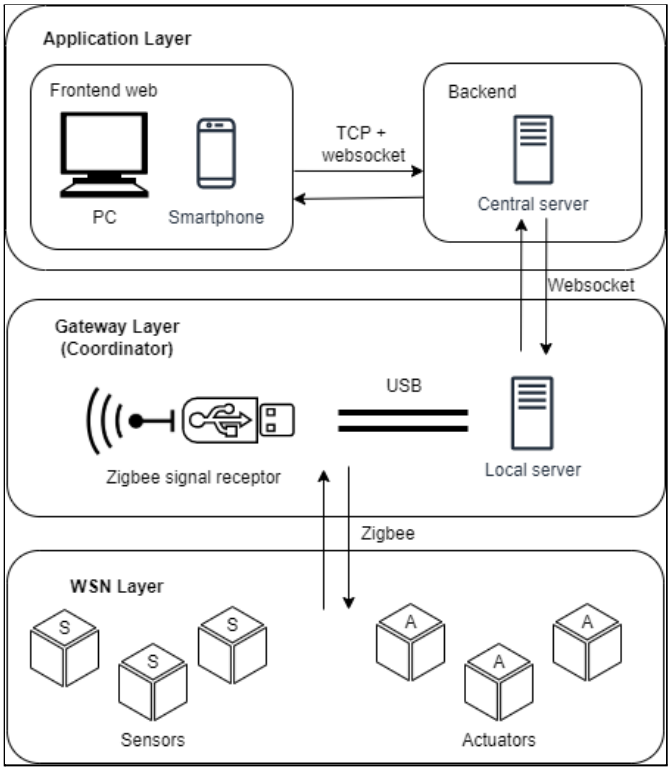


Figura 5: Diagrama de arquitectura de 3 capas de SIPGate con énfasis en las 3 capas de abstracción

A continuación, se presenta en detalle la estructura y funcionamiento de cada capa en particular.

2.3.1.1 Capa WSN

La capa WSN es la de más bajo nivel del modelo, la que interactúa directamente con cada dispositivo. La comunicación ocurre a través del protocolo Zigbee, una especificación basada en IEEE 802.15.4 enfocada específicamente en un bajo peso de mensajes y bajo consumo de energía por transmisión, lo que lo hace un protocolo ideal para la conexión de dispositivos de domótica (Salam et al., 2019).

El protocolo Zigbee tiene un alcance promedio dependiente del hardware utilizado para su implementación (Salam et al., 2019). Para este trabajo se utilizó un receptor de señales tipo CC2652 que concentra el flujo de mensajes de la red con un radio de alcance de 30 metros, considerado prudente para la implementación en un contexto de vivienda particular (Salam et al., 2019).

2.3.1.2 Capa gateway

Esta segunda capa del modelo se basa en un *middleware* físico conectado permanentemente a la red eléctrica y a la red de internet, y que es necesario posicionar físicamente de tal forma que todos los dispositivos estén a una distancia menor al alcance máximo del receptor utilizado.

Se describen en las subsecciones a continuación los tres componentes que conforman esta capa.

a) Hardware y sistema local

La capa gateway fue implementada a través de una Raspberry Pi 3 corriendo el sistema operativo Raspberry Pi OS buster, similar a (Kiran et al., 2021). Esta Raspberry tiene conectada mediante puerto USB el receptor de señales CC2652 y un modem 4G (o equivalente WiFi), que le permite no necesitar una conexión de red fija. A la unión de estos elementos se le llamó "Coordinador" y desde ahora en adelante se tratarán como una unidad indivisible.

Sobre el sistema operativo del Coordinador se ejecutan tres procesos importantes para la capa Gateway, uno de los cuales es necesario para la funcionalidad de traducción y dos para el procesamiento local.

b) Traducción

El receptor de señales únicamente escucha y envía mensajes según el protocolo Zigbee, pero para procesamiento posterior y compatibilidad de comunicación, es deseable utilizar un protocolo altamente estándar. En específico, se utilizó el protocolo MQTT que es liviano, basado en la arquitectura publicador-suscriptor y corre sobre TCP (Naik, 2017). Así, el primer proceso que es necesario ejecutar en el gateway es aquel que traduce entre señales Zigbee y mensajes MQTT. Para este punto se utilizó el traductor Z2M: Zigbee2MQTT, software de código abierto diseñado específicamente para esta traducción (Koenkk, November 2021).

c) Procesamiento local

El procesamiento local ocurre en dos procesos separados que comparten información.

Al primero de estos procesos se le denominó **“automatización”** y es el que está encargado de recibir las señales que los dispositivos entregan, realizar operaciones de control de flujo, tomar decisiones y enviar instrucciones de vuelta a esos u otros dispositivos de la red. Esta funcionalidad está implementada a través de un servidor de nodeJS (Tilkov & Vinoski, 2010) que corre un script de código JavaScript puro, pero incorpora además una librería para trabajar con mensajes MQTT. Este archivo script está construido con la arquitectura publicador-suscriptor basada en tópicos y está en todo momento escuchando a los tópicos de los dispositivos que se encuentran actualmente en la red. Además, este archivo es el que se ve modificado cuando el usuario cambia la configuración de su automatización en la capa de la aplicación y corre automáticamente cuando se enciende la Raspberry.

Al segundo proceso necesario para llevar a cabo el procesamiento local se le denominó **“servidor local”**. Esta funcionalidad permite escuchar las señales recibidas de los dispositivos con el objetivo de almacenar su información y llevar un registro de cuando se escuchó de cada uno por última vez. Además, este servidor local está conectado mediante protocolo WebSocket TCP a la capa de aplicación para permitir comunicación asíncrona bidireccional sin aumentar en gran manera el consumo de red

(Skvorc et al., 2014). De esta forma, el servidor local puede recibir las instrucciones que el usuario le mande desde la capa de la aplicación y enviar de vuelta los estados o información de la red o los dispositivos, según corresponda.

2.3.1.3 Capa de Aplicación

La última de las capas de la arquitectura es la capa de la aplicación con la que interactúa directamente el usuario. Esta capa está constituida por un servidor central *backend* y por un cliente *frontend* que corre en el navegador Web.

El servidor central *backend* está construido como una aplicación en nodeJS con una base de datos PostgreSQL alojada en su totalidad en los servicios en línea de Amazon Web Services (AWS) (Baron & Kotecha, 2013). Este servidor *backend* es el que realiza toda la gestión de cuentas, configuraciones, sesiones, historial de estados de los dispositivos y respaldos de lo que realizan los usuarios. También funciona como puente de comunicación bidireccional entre la aplicación *frontend* y el servicio de procesamiento local de la capa *Gateway*.

El cliente *frontend* está construido como una aplicación Web en ReactJS (Fedosejev, 2015), alojada en AWS, que se comunica únicamente con el servidor central *backend*. En este cliente se encuentran disponibles las interfaces que permiten configurar los dispositivos de la red WSN y crear y configurar las automatizaciones que se deseen aplicar para estos dispositivos anteriores.

a) Configuración de dispositivos

Desde la plataforma el usuario puede, en primer lugar, añadir su coordinador. Para esto, solo necesita registrar en la aplicación el código alfanumérico de 6 caracteres que se encuentra físicamente impreso en el componente. Una vez hecho esto, el usuario puede añadir una cantidad arbitraria de dispositivos, igualmente ingresando sus códigos únicos de 6 caracteres, como se observa en la Figura 6.

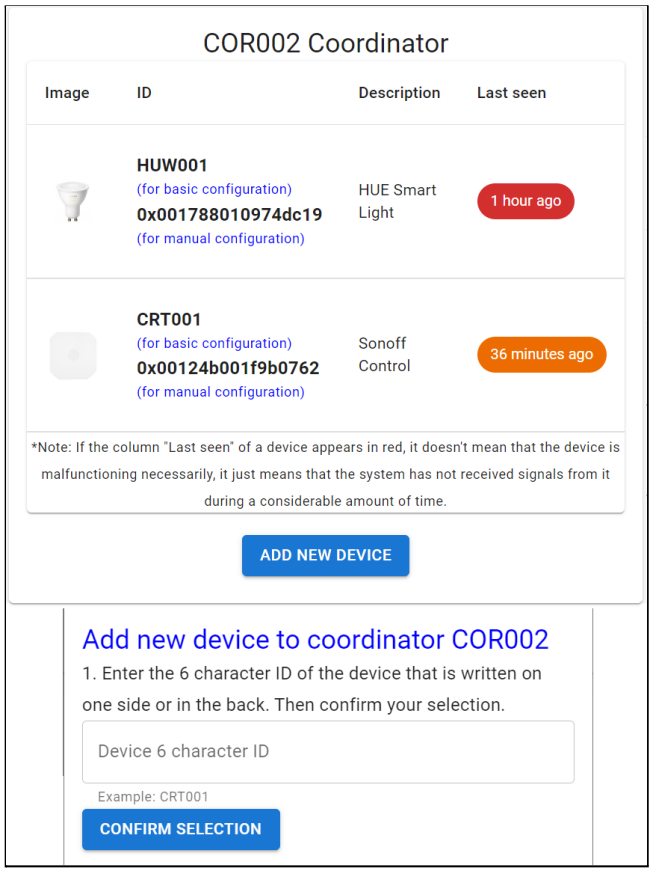


Figura 6: Interfaz utilizada para añadir dispositivos a partir de sus códigos únicos en plataforma SIPGate

Cada vez que se añade un dispositivo a un Coordinador a través de su ID de 6 caracteres, la aplicación *frontend* manda una solicitud al *backend*, el que obtiene el ID físico del dispositivo (número hexadecimal) de su base de datos, y envía a la capa *Gateway* del Coordinador la instrucción de añadir este dispositivo a su lista de "dispositivos permitidos", de tal forma de autorizar que solamente éste se comunique con el Coordinador y no cualquier otro. Si luego desde la aplicación se elimina un dispositivo, se manda la instrucción de sacarlo de esa misma lista. Este modelamiento añade una barrera de seguridad al impedir que un dispositivo cualquiera pueda conectarse a la red.

Además, a través de la plataforma es posible ver el estado de conexión de todos los dispositivos que se tiene agregados (la última vez que fueron vistos en línea).

b) Creación de automatizaciones

La creación de automatizaciones es la función principal de la capa de aplicación. Se quiere poder configurar cómo deberían funcionar en conjunto los dispositivos que se tienen añadidos. Esta configuración se puede realizar a través de una interfaz de texto o a través de la herramienta de programación visual Google Blockly (Pasternak et al., 2017) y se estructura según el paradigma de programación basado en eventos (Faison, 2006) en la que un "evento" es una señal recibida por un sensor.

En la opción de la interfaz de texto (caso denominado "configuración avanzada"), los dispositivos deben ser manejados de forma manual con sus ID

hexadecimales, mientras que en la herramienta Blockly (caso “configuración simple”) se encuentran pre-configurados automáticamente bloques con los ID de 6 caracteres que representan a cada uno de los posibles dispositivos. Cuando el usuario abre la pestaña con la interfaz de programación visual, únicamente ve los bloques relacionados a los dispositivos que actualmente tiene conectados a su coordinador.

Así, por ejemplo para el caso de Blockly, si el usuario quiere que automáticamente se realice una acción cuando un sensor de proximidad detecta cercanía, entonces lo que debería hacer es añadir a su automatización el bloque asociado a este sensor y seleccionar que quiere “escuchar” a la señal (*when* “evento”) de cercanía y no a otras señales que pueda generar. Una vez hecho esto, sólo necesita añadir la acción que se debería ejecutar al escuchar la señal de cercanía, lo que se logra conectando el bloque de acción deseado a este bloque anterior de “escuchar”. Se pueden añadir indistintamente reglas de control de flujo como “if” y “for” que impliquen otros sensores o variables arbitrarias al igual que se pueden añadir toda la cantidad de acciones que se desee, ya sea que estas impliquen actuadores u otras variables. En la Figura 7 se presenta un ejemplo de la situación anterior en la interfaz visual Blockly y en la Figura 8, el mismo ejemplo en la interfaz de programación de texto.

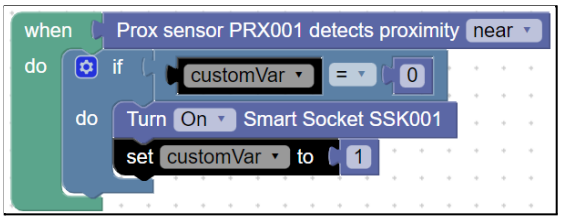


Figura 7: Construcción en Blockly de automatización “escuchar sensor y realizar acción sobre variable y actuador si una variable está en cierto estado”

```
// La siguiente lógica es la función “function” que corre dentro de un
client.on(‘message’, function(topic, payload)). Esta función se ejecuta
cada vez que el coordinador recibe un mensaje (evento) de cualquier
tópico ‘topic’ al que está suscrito. (Las suscripciones se manejan de
forma automática).
if (topic === 'zigbee2mqtt/0x002726ac25ff') {
  if (customVar === 0) {
    client.publish('zigbee2mqtt/0x002726ac25ff', {
      state: 'ON',
    });
    customVar = 1;
  }
}
```

Figura 8: Construcción en interfaz de texto de automatización “escuchar sensor y realizar acción sobre variable y actuador si una variable está en cierto estado”

Una vez que el usuario ha añadido todos los bloques en la interfaz visual o escrito todo lo deseado en la interfaz de texto para su automatización, puede guardar esta configuración, la que se envía al servidor central para ser respaldada y reenviada a la capa *gateway* del Coordinador. El Coordinador recibe esta nueva configuración, detiene la automatización que actualmente está ejecutando y comienza a ejecutar la nueva versión, sin necesitar ninguna acción manual.

2.3.2 Características relevantes

Se presentan a continuación cuatro puntos relevantes que la arquitectura de SIPGate permite.

2.3.2.1 Multiplataforma

Todo el proceso descrito anteriormente de configuración de componentes de la red de domótica y creación de automatizaciones se realiza a través de la aplicación Web multiplataforma. Todas las configuraciones que el usuario haga están respaldadas en el servidor central y, por lo tanto, no son dependientes del dispositivo desde el que se realicen.

2.3.2.2 Uso de 2 protocolos

Como se menciona en las secciones anteriores, se utiliza el protocolo Zigbee para comunicar la capa WSN con la capa Gateway y el protocolo WebSocket TCP para comunicar la capa Gateway con la capa de aplicación. El uso de dos protocolos de forma simultánea permite aprovechar los beneficios de cada uno en su contexto particular sin tener que utilizarlos en otros contextos en los que no son óptimos.

El protocolo Zigbee es liviano y de bajo consumo energético. Sin embargo, no es un protocolo diseñado con un alto nivel de fiabilidad, comprobación de errores o comunicación de larga distancia como TCP, por lo que no resulta adecuado para ser utilizado como comunicación entre un cliente y un servidor físicamente muy lejanos a través de internet, una red propensa a pérdidas de paquetes de información (Borella et al., 1998). Por otro lado, TCP provee todas las herramientas necesarias para ser altamente confiable en redes de alta pérdida, pero lo logra teniendo que añadir procesamiento extra que asegure la transmisión correcta de la información. Todo este

procesamiento adicional lo hace ser un protocolo pesado y no de bajo consumo, lo que impactaría negativamente la duración de las baterías de los dispositivos (Govindan et al., 2019).

Basado en las características de cada uno de los protocolos, se puede aprovechar lo liviano del protocolo Zigbee con lo confiable del protocolo TCP aplicando cada uno de estos en el contexto en el que resulta más favorable y comunicando ambos a través de una traducción realizada en la capa Gateway en el coordinador.

2.3.2.3 Plug-and-play

Uno de los puntos importantes de esta solución propuesta es el bajo nivel de configuración necesario para iniciar una primera automatización. Para que un usuario que parte totalmente desde cero pueda tener funcionando en su hogar una red compleja de dispositivos, necesita seguir los siguientes pasos: 1) Conectar el Coordinador a la red eléctrica y a internet a través de 4G, WiFi o internet por cable. El Coordinador está configurado para ejecutar automáticamente todos los procesos necesarios al momento de encenderse. 2) Ingresar a la dirección Web de la aplicación desde cualquier dispositivo y crearse una cuenta (o ingresar a la suya si ya tiene). 3) Registrar en la interfaz Web el identificador de 6 caracteres del Coordinador y de todos los dispositivos que se quiera incluir. 4) Crear una automatización desde la interfaz de programación visual o a través de la interfaz de texto con la complejidad arbitraria que desee y guardarlo. Con este último paso, las distintas capas de la arquitectura realizan la gestión necesaria para

actualizar la automatización en el coordinador y, de esta forma, la configuración hecha por el usuario queda automáticamente corriendo y lista para usarse.

Es importante notar que en todo este proceso, el usuario nunca tuvo que instalar ni configurar nada relacionado a los protocolos, a la comunicación, el manejo de mensajes o cualquier otro detalle técnico de bajo nivel. Solamente interactuó con la configuración de alto nivel a través de una plataforma unificada y guiada, y con esto logró implementar una automatización arbitrariamente compleja.

2.3.2.4 Configuración simple y avanzada

Por último, todo el proceso de *plug-and-play* y programación en Blockly permite empezar desde cero y lograr una automatización funcionando con todas las configuraciones por defecto en el caso "configuración simple". Sin embargo, es posible que haya situaciones en las que se desee realizar una "configuración avanzada", caso para el cual se necesita usar la interfaz de texto de SIPGate.

Lo anterior se logra de dos formas. La primera es que, a través de la interfaz de texto, el usuario es capaz de programar manualmente su automatización utilizando directamente el lenguaje JavaScript, ya que lo que hace Blockly en el caso de "configuración simple" es traducir bloques visuales a trozos de código. Así, para el caso de "configuración avanzada", el usuario puede saltar esta transformación y utilizar todo el potencial que ofrece este lenguaje, sin estar restringido por las implementaciones

previamente definidas en bloques. Las especificaciones concretas necesarias para programar con este modo se encuentran en el anexo I.

La segunda forma de “configuración avanzada” es a través del envío directo de mensajes MQTT que el usuario declara y el Coordinador ejecuta sin ningún procesamiento intermedio. Están soportados todos los comandos existentes en la documentación de Zigbee2MQTT (disponible en https://www.zigbee2mqtt.io/), lo que incluye opciones que pueden eventualmente romper el funcionamiento normal del coordinador. Así, se debe tener en consideración que, para esta configuración avanzada, SIPGate permite la utilización de la totalidad de las opciones que provee JavaScript y Zigbee2MQTT, pero no asegura de ninguna forma la continuidad del funcionamiento.

2.4. Casos de uso

Se presentan a continuación 3 casos de uso de automatizaciones concretas implementadas a través de SIPGate.

2.4.1 Aire acondicionado con multisensor

En una habitación de 1 puerta y 1 ventana, se quiere hacer uso eficiente del aire acondicionado teniéndolo encendido solo cuando hay una persona adentro y está la puerta y la ventana cerrada. Se quiere entonces poder cumplir las siguientes condiciones

para el aire acondicionado: se apaga si se abre la puerta o la ventana; se enciende si hay movimiento en la habitación y la puerta y la ventana están cerradas.

Para llevar a la realidad el ejemplo anterior se necesitan los siguientes sensores y actuadores: un enchufe inteligente que pueda encender y apagar el aire, un sensor de movimiento que detecte la presencia de una persona, y dos sensores de proximidad (uno en la puerta y uno en la ventana) que detecten si están abiertas o cerradas.

Una vez que el usuario ha registrado su coordinador y sus cuatro dispositivos en la interfaz Web, puede crear su automatización en Blockly o en la interfaz de texto. La Figura 9 muestra una opción utilizando Blockly con dos bloques de condiciones.

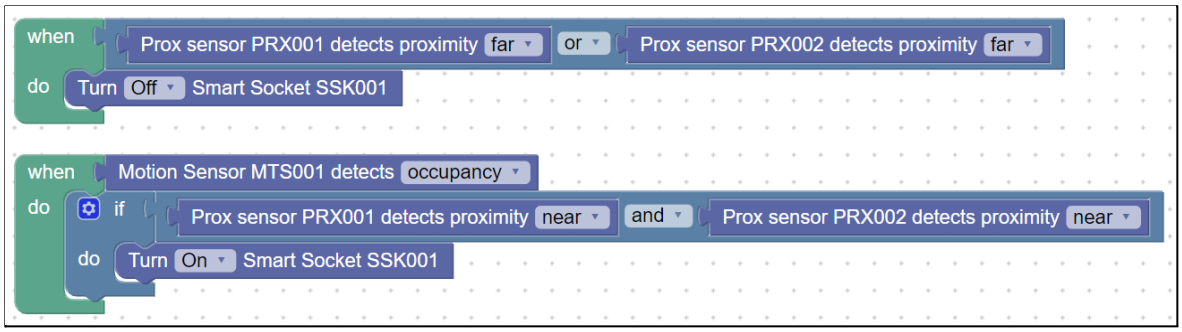


Figura 9: Implementación caso de uso “aire acondicionado con multisensor” en interfaz de programación visual Blockly.

Una vez que el usuario termina de crear su automatización, la guarda y ésta se actualiza de forma automática en su coordinador. En este momento, ya empieza a funcionar sin necesitar ningún paso extra.

Se muestra en la Figura 10 un diagrama explicativo de la implementación física y en la Figura 11 un trozo del historial de estados (*logs*) generados por los dispositivos cuando una persona abrió la puerta para salir de la habitación, la cerró tras de sí y luego volvió a abrirla para entrar nuevamente. Se observa de los *logs* que el sensor de proximidad pasó a estado "far" y el de movimiento a "unoccupied", por lo tanto, el enchufe pasó a estado "off". Luego, cuando se cerró la puerta, el sensor de proximidad pasó a estado "near", pero el sensor de movimiento seguía en "unoccupied". Finalmente, cuando la persona volvió a entrar, cambió el estado del sensor de proximidad a "near" y el de movimiento a "occupied", por lo que se pasó el enchufe a estado "on".

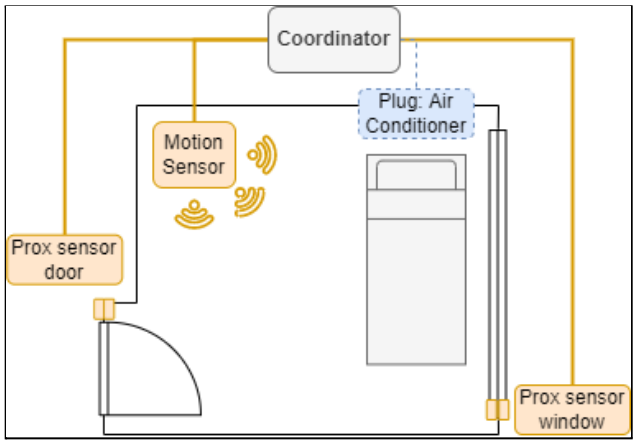
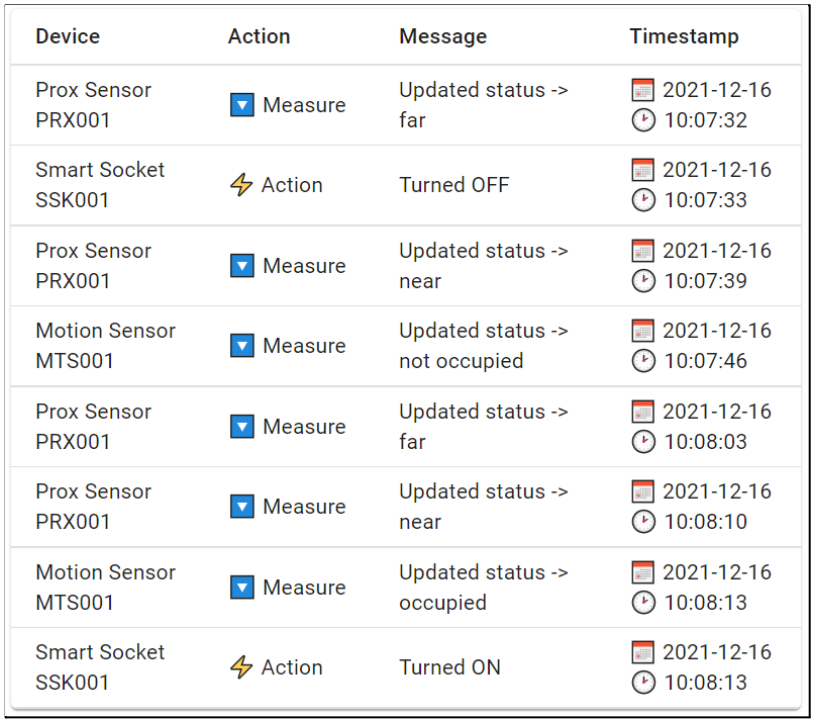


Figura 10: Diagrama posicionamiento de sensores (línea continua) y actuadores (línea punteada) conectados de forma inalámbrica para caso de uso "aire acondicionado con multisensor"



Device	Action	Message	Timestamp
Prox Sensor PRX001	Measure	Updated status -> far	2021-12-16 10:07:32
Smart Socket SSK001	Action	Turned OFF	2021-12-16 10:07:33
Prox Sensor PRX001	Measure	Updated status -> near	2021-12-16 10:07:39
Motion Sensor MTS001	Measure	Updated status -> not occupied	2021-12-16 10:07:46
Prox Sensor PRX001	Measure	Updated status -> far	2021-12-16 10:08:03
Prox Sensor PRX001	Measure	Updated status -> near	2021-12-16 10:08:10
Motion Sensor MTS001	Measure	Updated status -> occupied	2021-12-16 10:08:13
Smart Socket SSK001	Action	Turned ON	2021-12-16 10:08:13

Figura 11: Logs generados por el caso de uso “aire acondicionado con multisensor”

2.4.2 Alarma desactivable con variables

Se quiere recibir una notificación cuando alguien abre una puerta que debería estar cerrada (por ejemplo, la puerta de una habitación cuando el dueño no está). Sin embargo, no se quiere que se active esta “alarma” cuando el mismo dueño es el que abre la puerta, por lo que se desea poder “activar o desactivar” esta funcionalidad a voluntad.

Para implementar esta automatización se necesita un control, un sensor de proximidad y un nodo notificación por mail (el que está disponible para todos los usuarios de la plataforma). Para lograr la programación anterior, se utilizó nuevamente Blockly, pero esta vez con 2 bloques de condiciones junto con la funcionalidad de

declaración de variables de la siguiente forma: el primer conjunto se interpreta como “si se clickea una vez el control, activa el sistema si está desactivado o lo desactiva si está activado”. El segundo conjunto se interpreta como “si se abre la puerta y el sistema está en modo ‘activo’, entonces manda un email”.

Se muestra en la Figura 12 la implementación hecha en Blockly a través de un smartphone y, en la Figura 13, los *logs* generados al desactivar la alarma, abrir y cerrar la puerta, activar la alarma y volver a abrir la puerta. Se observa que la notificación solo se envió en el segundo caso.

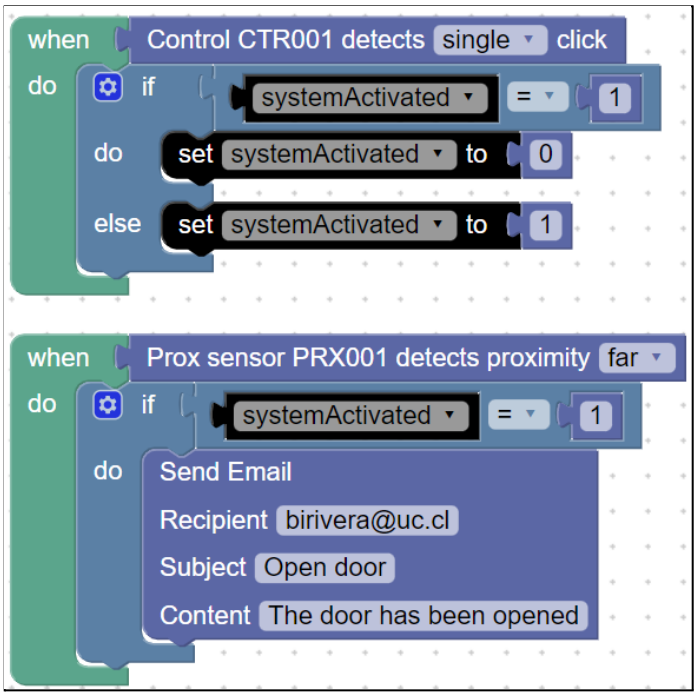
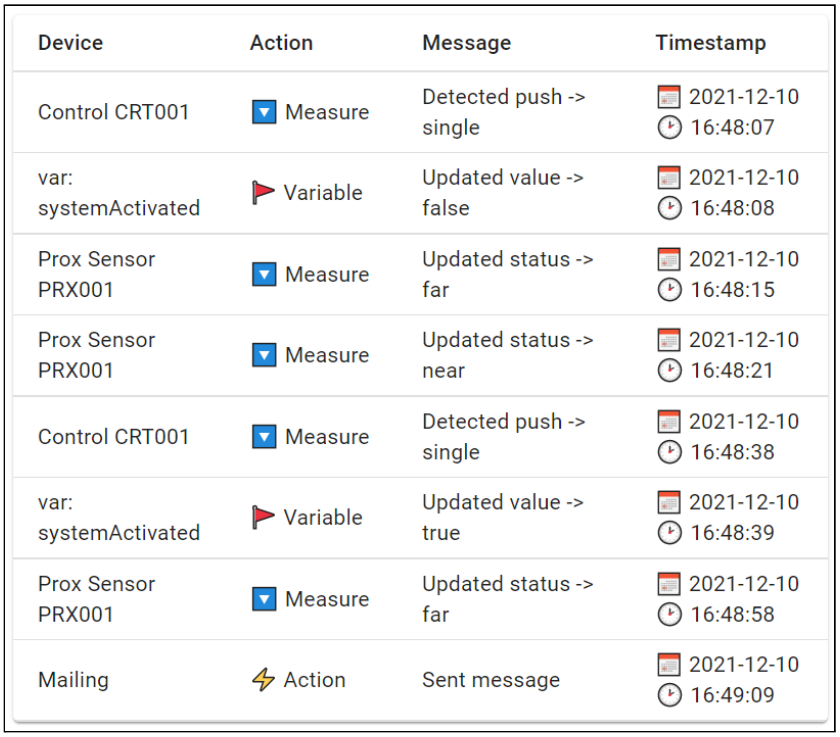


Figura 12: Implementación caso de uso “alarma desactivable con variables” en interfaz de programación visual Blockly.



Device	Action	Message	Timestamp
Control CRT001	Measure	Detected push -> single	2021-12-10 16:48:07
var: systemActivated	Variable	Updated value -> false	2021-12-10 16:48:08
Prox Sensor PRX001	Measure	Updated status -> far	2021-12-10 16:48:15
Prox Sensor PRX001	Measure	Updated status -> near	2021-12-10 16:48:21
Control CRT001	Measure	Detected push -> single	2021-12-10 16:48:38
var: systemActivated	Variable	Updated value -> true	2021-12-10 16:48:39
Prox Sensor PRX001	Measure	Updated status -> far	2021-12-10 16:48:58
Mailing	Action	Sent message	2021-12-10 16:49:09

Figura 13: Logs generados por el caso de uso “alarma desactivable con variables”

2.4.3 Café al amanecer con interfaz de texto

Se quiere prender el hervidor de agua durante 3 minutos de forma automática cuando amanezca para tomarse un café al despertar, pero solamente de lunes a viernes.

Para esta implementación se necesita un sensor de luz y un enchufe inteligente para el hervidor. Sin embargo, no existe por defecto un bloque “retorna ‘verdadero’ solo si hoy es día entre lunes y viernes”, por lo que es necesario crear esta automatización a través de la interfaz de texto para usar funcionalidades de JavaScript puro, lo que se considera una “configuración avanzada”.

Se muestra a continuación en la Figura 14 el trozo de código necesario para implementar la funcionalidad deseada. Es importante notar que los tópicos y mensajes que se escuchan o envían corresponden a los IEEE y servicios de los dispositivos, pero la función Date() es propia de JavaScript, no implementada directamente a través de los bloques existentes. Se observa entonces que, a través de esta interfaz de texto, se puede acceder a las funcionalidades de JavaScript puro, no limitando la libertad de programación únicamente a las capacidades de Blockly. La especificación de formato necesario para el código se encuentra descrito en el anexo I.

```
// La siguiente lógica es la función “function” que corre dentro de un
client.on(‘message’, function(topic, payload)). Esta función se ejecuta
cada vez que el coordinador recibe un mensaje (evento) de cualquier
tópico ‘topic’ al que está suscrito. (Las suscripciones se manejan de
forma automática).
const lgSensTopic = 'zigbee2mqtt/0x0012a4f5b0f7';
const lightLevel = payload.lightLevel;
const plugTopic = 'zigbee2mqtt/0x002726ac25ff/set';
const days = ["su", "m", "tu", "w", "th", "f", "sa"];
if (topic === lgSensTopic && lightLevel > 3) {
  const dayOfWeek = new Date().getDay();
  const dayToday = days[dayOfWeek];
  const coffeeDays = ["m", "tu", "w", "th", "f"];
  if (coffeeDays.includes(dayToday)) {
    // Variable 'client' predefinida en cada script
    client.publish(plugTopic, JSON.stringify({
      state: 'ON',
    }));
    // Función sleep(ms) predefinida en cada script
    sleep(180000);
    client.publish(plugTopic, JSON.stringify({
      state: 'OFF',
    }));
  }
}
```

Figura 14: Implementación caso de uso “café al amanecer con interfaz de texto” escrito en JavaScript

Por último, es importante notar que esta implementación de funcionalidades avanzadas a través de texto plano no incluye ningún log por defecto, sino que deben ser también implementados manualmente si se desea conforme a la especificación de la documentación disponible en el anexo I.

2.5. Discusión y limitaciones

En la sección anterior se presentan 3 casos de uso concretos construidos a través de SIPGate en los que se pudo programar, a través de la interfaz visual o la interfaz de texto, una automatización que utiliza uno o varios sensores y uno o varios actuadores y que puede incluir funcionalidades de JavaScript no soportadas actualmente en bloques de Blockly. Para lograr lo anterior, se necesitó ingresar a la aplicación *frontend*, ingresar a una cuenta, registrar el coordinador y los dispositivos que quería utilizar, programar la automatización y, finalmente, guardar la configuración.

Se observa que en SIPGate se planteó una arquitectura basada en servicios y separada en capas de abstracción (Aziz, 2013). Así, se pudo observar que nunca fuera necesario interactuar directamente con la parte técnica de bajo nivel de la implementación, sino que simplemente se veían los “servicios” (señales y acciones) que los dispositivos exponían. La diferencia con respecto al trabajo mencionado anteriormente es que SIPGate lleva esta arquitectura a una implementación concreta basada en un gateway, como se explica en la sección 3.

De los prototipos de ejemplo, se observó también que se interactuó en todo momento con los dispositivos de la red a través de una interfaz Web accesible desde computador o smartphone (Perumal et al., 2008; Rusli & Dianati, 2012). Sin embargo, a diferencia de esos trabajos, en SIPGate no se utilizó el protocolo TCP en todo momento, sino que se usó un doble protocolo WebSocket TCP para la interacción aplicación-gateway y Zigbee de bajo consumo de batería para la conexión gateway-dispositivos (Morais Bolzani et al., 2006).

Luego, SIPGate tuvo un proceso de inicio "*plug-and-play*", ya que se necesitó un nivel muy bajo de configuración para empezar a programar las automatizaciones (Santamaria et al., 2014). Solo se necesitó conectar el Coordinador a internet y registrarlo junto con los dispositivos en la interfaz Web. Sin embargo, nuestra solución además permitió realizar automatizaciones arbitrariamente complejas, como por ejemplo el aire acondicionado que dependía del estado de tres sensores de forma simultánea, no de algo modificable manualmente. Además, permitió configuraciones avanzadas que habilitaron usar funcionalidades propias de JavaScript puro, no disponibles en bloques, como Date(), a través de la interfaz de comandos de texto.

Por último, se observó que todo el proceso completo, desde el primer al último paso, fue realizado en una única aplicación, sin necesitar en ningún momento realizar alguna instalación o configuración manual. La automatización fue ejecutada en forma local por el gateway (Serdaroglu & Baydere, 2016), pero SIPGate además lo logra con esta experiencia de usuario con menor barrera de entrada. Se observó también un fuerte

énfasis en la interacción unificada usuario-dispositivo que no requiere un alto conocimiento técnico (Aguilar & Echeverría, 2020; Menéndez et al., 2020), pero además, SIPGate logró una interacción totalmente bidireccional y basada en tecnologías Web tal que permitiera acceder desde cualquier plataforma con un navegador, y no estar obligado a instalar una aplicación específica para cada sistema.

Es relevante reconocer las limitaciones de este trabajo. La primera tiene relación con la necesidad de conexión a internet. Gracias al procesamiento local del *gateway*, no es necesario que el coordinador esté conectado a internet en todo momento para ejecutar la automatización configurada. Sin embargo, para poder modificar esta automatización o revisar los historiales de estado (*logs*) generados por el proceso, se requiere obligatoriamente una conexión constante a internet. Por decisión de diseño, teniendo como objetivo la seguridad, no es posible conectarse de forma local al coordinador sin pasar por la interfaz Web, por lo que aunque se esté físicamente al lado del coordinador, no es posible acceder a los *logs* que está generando ni es posible actualizar la configuración guardada sin conexión a internet. Otra limitación es que, aunque el enfoque de SIPGate sea la facilidad de uso para principiantes, igualmente se necesita cierto conocimiento de programación que permita entender la lógica representada en bloques en la interfaz de programación visual (por ejemplo, condiciones “if” y uso de funciones). Dado que esta propuesta se enmarca en un proyecto que busca enseñar a programar, lo anterior es una característica de este trabajo; sin embargo, si se desea utilizar como solución domótica, se convierte en un requerimiento para su uso. Pese a

que en este trabajo mostramos que se pueden crear soluciones domóticas de manera sencilla, éstas fueron creadas por los autores del artículo, por lo que en el futuro es necesario evaluar la usabilidad y facilidad de uso con usuarios finales.

2.6. Conclusiones y trabajo futuro

En este trabajo se presenta SIPGate, que implementa una solución de creación y configuración de automatizaciones *plug-and-play*, a través de una aplicación Web multiplataforma y diseñado para tener un fuerte enfoque en una experiencia de usuario amigable, pero sin sacrificar libertad de complejidad deseada o de configuraciones avanzadas. Estas características se comprueban mediante 3 casos de uso implementados en SIPGate.

Como trabajo futuro, se plantea la necesidad de explorar mayores opciones de interacción fuera de línea con el coordinador, ya que actualmente para la actualización de flujos o revisión de historiales de acciones, se requiere obligatoriamente una conexión a internet. Una opción sería la incorporación de cachés que almacenen temporalmente los cambios o explorar la opción de conectarse directamente al coordinador a través de un protocolo de corto alcance.

Finalmente, como este trabajo se enmarca dentro de un proyecto mayor que busca enseñar pensamiento computacional a través de la domótica, se buscará probar si una solución con programación de bloques como esta permite enseñar conceptos de

programación como condiciones, variables y funciones a niños y jóvenes y a la vez mostrarles el impacto de la computación en el mundo real.

3. CONCLUSIONES Y TRABAJO FUTURO

En este último capítulo se extraen las principales conclusiones obtenidas del trabajo realizado. Se detallan en concordancia con los objetivos planteados, la implementación hecha de la solución propuesta y los casos de uso analizados. Finalmente, se discuten las limitaciones encontradas y posibles opciones de trabajo futuro.

3.1. Conclusiones

En primer lugar, se obtuvo que efectivamente fue posible implementar la solución propuesta a través de una arquitectura de 3 capas basada en un *gateway*. La capa Gateway solo ve los servicios que expone la capa WSN y el usuario solo interactúa en la capa de la aplicación con los servicios que expone la capa Gateway. De igual forma, se observó que toda la interacción hecha con los dispositivos estuvo unificada en una única aplicación y, para el caso de uso “alarma desactivable con variables”, la automatización realiza el manejo de estados de manera local. Se concluye entonces que el primer objetivo de implementación del sistema se cumple correctamente.

Luego, se observó que en los tres casos de uso fue posible construir una automatización partiendo desde cero, sin necesitar en ningún momento algún otro

hardware externo, instalación o configuración avanzada. Para el coordinador y para los dispositivos, la experiencia se resumió en “conectar y ya está listo para usarse”, es decir, se cumple efectivamente el objetivo de ser plug-and-play.

Del caso de uso “alarma con estados” y de la implementación del cliente Web en la capa de la aplicación, se desprende que efectivamente la solución propuesta es multiplataforma, pues puede ser ejecutada en un navegador sin requerimiento específico de un dispositivo o instalación y toda la información de la red de un usuario está respaldada en el servidor central, por lo que no depende del dispositivo desde el que se acceda.

Por último, del caso de uso “café por la mañana con configuración avanzada” se observó que fue posible crear, en código JavaScript a través de una interfaz de texto, una automatización utilizando funcionalidades programadas manualmente que no se encuentran disponibles en los bloques de la interfaz de programación visual. Es posible entonces realizar configuraciones y automatizaciones arbitrariamente complejas sin estar limitado por las opciones predefinidas en la interfaz visual. Se tiene entonces la posibilidad de realizar una configuración avanzada, pero teniendo también la opción visual amigable para usuarios nuevos. Se cumple entonces el objetivo de permitir configuraciones simples y avanzadas.

En resumen, en este trabajo se logró desarrollar una aplicación de domótica multiplataforma que no requiere ningún componente físico adicional, instalación o configuración técnica para usarse. Tiene además dos opciones de interacción: una

interfaz de programación visual para principiantes de tal forma de poder crear una automatización sin entrar en ningún apartado técnico; y una interfaz de programación de texto para usuarios avanzados que quieran tener total control sobre el comportamiento de su red. Se comprueba entonces el cumplimiento de la hipótesis planteada en este proyecto.

3.2. Limitaciones

Existen ciertas limitaciones para el trabajo presentado. En primer lugar, en cuanto a la arquitectura misma de la solución, se tiene el problema de la necesidad de conexión a internet para interactuar con algunas funcionalidades de la red de domótica. Si bien es cierto que una automatización puede correr sin que el coordinador esté conectado a internet, esta conexión es obligatoria al momento de querer actualizar alguna configuración u obtener los datos generados. Necesariamente se tiene que acceder a través de la interfaz Web.

Adicionalmente, parte importante de la arquitectura planteada se basa en el protocolo Zigbee y el software de traducción Zigbee2MQTT, por lo que no se tiene compatibilidad directa con otro tipo de protocolos de comunicación de domótica y se depende ineludiblemente de cómo evolucione la compatibilidad de este programa de traducción y la aceptación y mantenimiento del protocolo Zigbee.

Una tercera limitación es que, aunque el enfoque de este trabajo sea la facilidad de uso para principiantes, igualmente se necesita un conocimiento previo de

programación que permita entender la lógica representada en bloques en la interfaz de programación visual, por ejemplo, condiciones, control de flujo y funciones.

Finalmente, pese a que en este trabajo se muestra la posibilidad multiplataforma de ejecutar la aplicación Web desde un computador o un smartphone, esto solo se desarrolla a nivel de prototipo funcional, no como un producto final totalmente optimizado para cualquier tipo de pantalla, dispositivo o navegador por lo que no se puede asegurar su funcionamiento universal.

3.3. Trabajo futuro

Como trabajo futuro, se plantea la necesidad de explorar mayores opciones de interacción offline con el coordinador, ya que actualmente para la actualización de flujos o revisión de logs, se requiere obligatoriamente una conexión a internet. Una opción sería la incorporación de cachés que almacenen temporalmente los cambios o explorar la opción de conectarse directamente al coordinador a través de un protocolo de corto alcance.

De igual forma se plantea la posibilidad de explorar otros protocolos de comunicación livianos como 6LoWPAN o Z-Wave que se puedan incorporar de forma paralela a Zigbee y se plantea la necesidad de establecer estándares que permitan conectar a la infraestructura planteada dispositivos que no estén actualmente soportados. Los detalles de compatibilidad necesarios se encuentran descritos en el anexo II.

Finalmente, como este trabajo se enmarca dentro de un proyecto mayor que busca enseñar pensamiento computacional a través de la domótica, se plantea igualmente, a modo de trabajo futuro, la necesidad de probar si una solución con programación de bloques como esta permite enseñar conceptos de computación a niños y jóvenes y también la necesidad de experimentar con otros tipos de bloques o simplificar de mayor forma la interfaz de texto. Esto con el objetivo de encontrar la sintaxis, redacción e interacción de bloques que permita de mejor forma el aprendizaje, y disminuya el requisito de conocimiento previo de programación.

BIBLIOGRAFÍA

Aguilar, R. P., & Echeverría, A. (2020). Diseño de un software de automatización de propósito general basado en Raspberry Pi. *Artículos, Foro Tecnológico*. https://www.researchgate.net/profile/Martin-Menendez-6/publication/354693004_Automatizacion_del_analisis_y_la_generacion_de_codigo_de_sistemas_de_enclavamiento_en_FPGA/links/6147e0973c6cb310697e0a54/Automatizacion-del-analisis-y-la-generacion-de-codigo-de-sistemas-de-enclavamiento-en-FPGA.pdf#page=48

Aziz, M. W. (2013). Service-Oriented Layered Architecture for Smart Home. *International Journal of Smart Home*, *7*(6), 409–418.

Baron, J., & Kotecha, S. (2013). Storage options in the aws cloud. *Amazon Web Services, Washington DC, Tech*. https://www.ncloud24.com/aws/img/file/AWS_Storage_Options.pdf

Bellino, A. (2016). Protobject: A sensing tool for the rapid prototyping of UbiComp systems. *Proceedings of the 2016 ACM International Joint*. https://dl.acm.org/doi/abs/10.1145/2968219.2971373

Bonino, D., Castellina, E., & Corno, F. (2008). The DOG gateway: enabling ontology-based intelligent domotic environments. *IEEE Transactions on Consumer Electronics*, *54*(4), 1656–1664.

Borella, M. S., Swider, D., Uludag, S., & Brewster, G. B. (1998). Internet packet loss: measurement and implications for end-to-end QoS. *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications Flexible Communication Systems. Wireless Networks and Mobile Computing (Cat. No.98EX206)*, 3–12.

Ergen, S. C. (2004). ZigBee/IEEE 802.15. 4 Summary. *UC Berkeley, September*, *10*(17), 11.

Dobesova, Z.(2012): Visual programming for novice programmers in geoinformatics. SGEM 2012 12th International Multidisciplinary Scientific GeoConfrenceConference, Proceedings Volume III, STEF92 Technology Ltd., Sofia, Bulgaria, 433-440p. ISSN 1341-2704, DOI: 10.5593/sgem2012

Faison, T. (2006). *Event-Based Programming*. Springer.

Fedosejev, A. (2015). *React.js Essentials*. Packt Publishing Ltd.

Fette, I., & Melnikov, A. (2011). *The websocket protocol*. RFC 6455, December. http://www.hjp.at/doc/rfc/rfc6455.html

Govindan, K., A, K., Ppallan, J. M., Jaiswal, S., & Subramaniam, K. (2019). TCP Closure Optimization for Enhanced Battery Life in Smart Devices. *IEEE Transactions on Mobile Computing*, *18*(3), 645–657.

Gupta, S., & Gupta, S. (2019). Comparative Analysis of Energy Consumption in Sensor Node Scheduling Heuristics in Wireless Sensor Network. *Engineering Vibration, Communication and Information Processing*, 399–406.

Harper, E. R., Rodden, T., Rogers, Y., Sellen, A., & Human, B. (2008). *Human-Computer Interaction in the year 2020*. Citeseer.

Jangra, A. (2010). Wireless Sensor Network (WSN): Architectural Design issues and Challenges. *(IJCSE) International Journal on Computer Science and Engineering*, *2*(09), 3089–3094.

Kiran, S., Sai, S. M., Shivani, C. H., Srilekha, S., & Yashwanth, H. (2021). *Web Controlled Home Automation Using Raspberry Pi*. https://login.easychair.org/publications/preprint_download/xhP7

Koenkk. (November 2021). *Home | Zigbee2MQTT*. Zigbee2MQTT. https://www.zigbee2mqtt.io/

Marikyan, D., Papagiannidis, S., & Alamanos, E. (2019). A systematic review of the smart home literature: A user perspective. *Technological Forecasting and Social Change*, *138*, 139–154.

Matthys, N., Yang, F., Daniels, W., Michiels, S., Joosen, W., Hughes, D., & Watteyne, T. (2015). µPnP-Mesh: The plug-and-play mesh network for the Internet of Things. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 311–315.

Menéndez, M., Lutenberg, A., Alvarez, N., Larosa, F., & Ghignone, R. (2020). Automatización del análisis y la generación de código de sistemas de enclavamiento en FPGA. *Artículos, Foro Tecnológico Y Reportes*, 25.

Morais Bolzani, C. A., Montagnoli, C., & Netto, M. L. (2006). Domotics Over IEEE 802.15.4 - A Spread Spectrum Home Automation Application. *2006 IEEE Ninth*

International Symposium on Spread Spectrum Techniques and Applications, 396–400.

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Systems Engineering Symposium (ISSE)*, 1–7.

Oliveira, L. M. L., Rodrigues, J. J. P. C., Elias, A. G. F., & Zarpelão, B. B. (2014). Ubiquitous monitoring solution for wireless Sensor Networks with push notifications and end-to-end connectivity. *Mobile Information Systems*, *10*(1), 19–35.

Pasternak, E., Fenichel, R., & Marshall, A. N. (2017). Tips for creating a block language with blockly. *2017 IEEE Blocks and Beyond Workshop (B B)*, 21–24.

Pei, Z., Deng, Z., Yang, B., & Cheng, X. (2008). Application-oriented wireless sensor network communication protocols and hardware platforms: A survey. *2008 IEEE International Conference on Industrial Technology*, 1–6.

Perumal, T., Ramli, A. R., Leong, C. Y., Mansor, S., & Samsudin, K. (2008). Interoperability for smart home environment using web services. *International Journal of Smart Home*, *2*(4), 1–16.

Raskin, J. (1997). Looking for a humane interface: will computers ever become easy to use? *Communications of the ACM*. https://dl.acm.org/doi/pdf/10.1145/253671.253737

Rusli, S., & Dianati, M. (2012). Mobile Access to Smart Home Network. *2012*. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.210.6381&rep=rep1&type=pdf

Salam, A., Hoang, A. D., Meghna, A., Martin, D. R., Guzman, G., Yoon, Y. H., Carlson, J., Kramer, J., Yansi, K., Kelly, M., Skvarek, M., Stankovic, M., Le, N. D. K., Wierzbicki, T., & Fan, X. (2019). *The Future of Emerging IoT Paradigms: Architectures and Technologies*. https://doi.org/10.20944/preprints201912.0276.v1

Santamaria, A. F., De Rango, F., Falbo, D., & Barletta, D. (2014). SmartHome: a domotic framework based on smart sensing and actuator network to reduce energy wastes. *Wireless Sensing, Localization, and Processing IX*, *9103*, 63–69.

Santos, J. C. S., Sejfia, A., Corrello, T., Gadenkanahalli, S., & Mirakhorli, M. (2019, August 12). Achilles' heel of plug-and-Play software architectures: a grounded theory based approach. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE '19: 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn

Estonia. https://doi.org/10.1145/3338906.3338969

Serdaroglu, K. C., & Baydere, S. (2016). WiSEGATE: Wireless Sensor Network Gateway framework for internet of things. In *Wireless Networks* (Vol. 22, Issue 5, pp. 1475–1491). https://doi.org/10.1007/s11276-015-1046-5

Singh, A. P., Luhach, A. K., Gao, X.-Z., Kumar, S., & Roy, D. S. (2020). Evolution of wireless sensor network design from technology centric to user centric: An architectural perspective. *International Journal of Distributed Sensor Networks*, *16*(8), 1550147720949138.

Skvorc, D., Horvat, M., & Srbljic, S. (2014). Performance evaluation of Websocket protocol for implementation of full-duplex web streams. *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1003–1008.

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, *14*(6), 80–83.

Woo, J., & Lim, Y. (2015). User experience in do-it-yourself-style smart homes. *Proceedings of the 2015 ACM International Joint*. https://dl.acm.org/doi/abs/10.1145/2750858.2806063

ANEXOS

I. Documentación uso de interfaz de texto

1. Uso general

La interfaz de texto de SIPGate permite programar automatizaciones directamente en código JavaScript. Para que lo anterior funcione, SIPGate toma el código que escribió el usuario y le añade toda la configuración necesaria antes de enviarlo al coordinador para ser ejecutado.

La forma general que tiene una automatización .js ejecutable de SIPGate es la siguiente.

```
// Imports
const mqtt = require('mqtt');
const axios = require('axios');
require('dotenv').config();
// Setup and status management init
const client = mqtt.connect(process.env.MQTTSERVER);
const devicesStatus = {};

// Subscriptions
client.on('connect', function () {
  // Automatic subscriptions to all devices that the controller has
  // client.subscribe('zigbee2mqtt/0x00124b002269e5d9', function (){});
  // ...
})
// Trigger that activates with EVERY message from any device
client.on('message', async function(topic, payload) {
  // Message parsing and status save
  const msg = JSON.parse(payload.toString());
  devicesStatus[topic] = msg;
  // -------------------
  // USER'S CODE ADDED IN THE WEB APPLICATION
  // -------------------
})
```

Luego, para escuchar una señal específica de un topic en específico, se añade como bloque **“IF”** dentro del client.on(“message”), de la siguiente forma

```
// Do something if received status1 = bar in topic1
if (topic === 'topic1' && msg.status1 === 'bar') {
  // Do something
}
```

Y para publicar un mensaje en un tópico, se hace de la siguiente forma (importante notar que el payload debe enviarse como string)

```
// Send payload state: OFF to topic2
client.publish('topic2', JSON.stringify({
  state: 'OFF',
}))
```

2. Uso de LOGS

Además del envío de mensajes MQTT normal, puede ser de interés registrar LOGS (historial de eventos) de lo que están haciendo los dispositivos. Para esto, se puede usar la siguiente función, que envía la información al servidor central de SIPGate, el que la almacena y asocia al coordinador y usuario que lo envía.

```
const logsUrl = `${process.env.WEBSOCKET_URL}/api/logs/add`;
const response = axios.post(logsUrl, {
  coordinatorId: process.env.RASPBERRY_ID,
  deviceId, // INTEGER: Unique device identifier
  flowId,   // INTEGER: Unique flow identifier
  action,   // STRING: action to log
  log,      // STRING: details to log
})
```

3. Uso de notificaciones

Las notificaciones que permite SIPGate son manejadas directamente por el servidor central y no son enviadas directamente desde el coordinador. Para gatillar una notificación se necesita utilizar la siguiente función. (Nota: ejemplo para el caso mailing. Otros tipos de notificaciones futuras pueden tener leves variaciones)

```
const response = await axios.post(mailRoute, {
  to: 'example@example.com',
  subject: 'Email subject',
  text:'Body of the email',
})
```

II. Documentación compatibilidad con otros protocolos

El servidor de procesamiento local del coordinador es capaz de recibir y enviar únicamente **mensajes según el protocolo MQTT**. Para esto utiliza un segundo proceso local que funciona como "servidor MQTT" y requiere tener un **MQTT broker** instalado (recomendado Mosquitto). Por defecto este servidor MQTT corre en "localhost", pero puede ser cambiado desde el **.env** del proyecto **domotics_raspberry**.

```
MQTTSERVER='mqtt://localhost'
```

Así, para permitir conectar **cualquier tipo de dispositivo** a SIPGate, se necesitan seguir los siguientes pasos:

1. Traducir los mensajes desde el protocolo utilizado por el dispositivo a mensajes MQTT con un topic y un payload. Para esto se pueden usar herramientas similares a Zigbee2MQTT, compatibles para el protocolo particular de interés. El formato de los mensajes debe ser el siguiente:
    - **"topic"** debe ser un STRING único para cada dispositivo
    - **"payload"** debe ser un STRING. Se recomienda para este paso armar un mensaje en JSON y luego transformarlo a STRING. De esta forma se puede incluir mayor cantidad de información de forma ordenada.

2. Publicar los mensajes MQTT obtenidos en la dirección declarada en el **.env** de domotics_raspberry.

Se muestra a continuación un ejemplo de mensaje MQTT publicado en el servidor MQTT "MQTTServer"

```
MQTTServer.publish('topic/0x04cf8cdf3c789384', JSON.stringify({
  state1: 'changeState1',
  otherInfo: {
    var1: 'thing1',
    var2: 1,
  }
}))
```

3. Para que el servidor local de domotics_raspberry pueda reconocer estos mensajes, es necesario suscribirse a los tópicos que se están publicando en los pasos anteriores. Esto se lleva a cabo añadiendo una nueva fila de client.subscribe() a la función en el client.on("connect"), como se muestra a continuación.

```
client.on('connect', function () {
  client.subscribe('topic1', function (){});
  client.subscribe('topic2', function (){});
  client.subscribe('topic/0x04cf8cdf3c789384', function (){});
})
```

4. Una vez hecho esto, se puede trabajar libremente con los mensajes dentro de la función client.on("message", function (topic, payload) { }), como se muestra a continuación.

```
client.on('message', async function(topic, payload) {
  // Transform payload into JSON
  const msg = JSON.parse(message.toString());
  // Turn off if door open
  if (topic === 'topic1' && msg.contact === false) {
    client.publish('topic2', JSON.stringify({
      state: 'OFF',
    }))
  }
})
```

5. Por último, para publicar mensajes a los dispositivos externos que conectamos, debemos publicar un mensaje en el cliente de MQTT configurado por defecto en la interfaz de texto. Este cliente automáticamente está conectado al servidor MQTT declarado en el .env del proyecto. Luego de esto, el mismo traductor usado en el paso 1 debería transformar estos mensajes MQTT salientes en el formato deseado de comunicación.

```
client.publish('topic1321', JSON.stringify({
  state: 'OFF',
}))
```