# SISTEMA DE EVENTOS REACTIVO CENTRADO EN EL USUARIO PARA LA GESTIÓN DE ALERTAS

**Planells Alba, Alejandro**

**Curs 2016-2017**

**Directores: CARLOS BOCK**

**DANIEL SOTO**

**GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS**

upf. Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

**Treball de Fi de Grau**

# SISTEMA DE EVENTOS REACTIVO CENTRADO EN EL USUARIO PARA LA GESTIÓN DE EVENTOS

Alejandro Planells Alba

TRABAJO FINAL DE GRADO

GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS

ESCOLA SUPERIOR POLITÈCNICA UPF
2017

DIRECTORES:

CARLOS BOCK

DANIEL SOTO

*Para ellos, por darme esta oportunidad.*

*Para ellos, por apoyarme en todo momento.*

*Para ellos, por confiar en mí.*

*Para ellos.*

*Para mis padres.*

# Agradecimientos

En estas líneas me gustaría agradecer a Daniel Soto por su compromiso y dedicación que ha mostrado durante la elaboración de este proyecto. He aprendido mucho de él durante estos meses. Gracias por orientarme y ayudarme a salir adelante en situaciones de bloqueo.

Por otro lado me gustaría agradecer a mis padres el apoyo recibido durante estos 4 años de carrera, especialmente estos últimos meses, donde me han tranquilizado en muchas ocasiones.

Gracias a todos los familiares y amigos que me han dado soporte moral para conseguir los objetivos propuestos.

Por último, no me puedo olvidar de Marina, que ha estado siempre ahí apoyándome y me ayudado mucho a conseguir que no me rinda.

¡Gracias a todos!

# RESUMEN

El uso de la tecnología para aumentar el bienestar y la comodidad de las personas es un fin que debería motivar a todo desarrollo tecnológico. En ese sentido las tecnologías de IoT son muy interesantes para conseguir dicho fin. Por ejemplo, poder controlar el entorno del usuario a través de un dispositivo electrónico o informar de lo que está sucediendo alrededor de él, son tareas útiles que deberían poder proyectarse en nuestra vida cotidiana. En este proyecto se analizan tecnologías incipientes para diseñar y modelar un sistema de eventos reactivo que es capaz de recibir diferentes tipos de eventos para que, una vez procesada la información, se presente cuando sea necesario una alerta al usuario. Dicha alerta será enviada y presentada de forma diferente dependiendo del estado del cliente. Las variables para presentar la alerta de una u otra forma son la localización del usuario y los dispositivos disponibles en ese momento. Por tanto, un punto clave de este proyecto es la posibilidad de integrar diferentes interfaces, tanto de entrada como de salida, a la vez que se centra completamente en el propio usuario.

# RESUM

L'ús de la tecnologia per augmentar el benestar i la comoditat de les persones és un fi que hauria de motivar a tot desenvolupament tecnològic. En aquest sentit les tecnologies d'IoT son molt interessants per tal d'aconseguir aquest fi. Per exemple, poder controlar l'entorn de l'usuari a través d'un dispositiu electrònic o informar del que està succeint al voltant d'ell, son tasques útils que haurien de poder projectar-se a la nostra vida quotidiana. En aquest projecte s'han analitzat tecnologies incipients per tal de dissenyar i modelar un sistema reactiu d'esdeveniments que és capaç de rebre diferents tipus d'events per a que, una vegada processada la informació, es presenti quan sigui necessari una alerta a l'usuari. Aquesta alerta serà enviada i presentada de forma diferent depenent de l'estat del client. Les variables per presentar l'alerta d'una forma o una altra son la localització de l'usuari i els dispositius disponibles en aquell moment. Per tant, un punt clau d'aquest projecte és la possibilitat d'integrar diferents interfícies, tant d'entrada com de sortida, a la vegada que es centra completament en el propi usuari.

# ABSTRACT

The use of technology in order to improve the well-being and comfort of the people is an end that should motivate every technological development. In this sense, IoT technologies are really interesting for achieving this goal. For example, being able to control the user's environment through an electronic device or informing about what is happening around him, are useful tasks that should be projected in our daily life. This project analyses emerging technologies to design and model a reactive event system that is capable of receiving different types of events so that, once the information is processed, an alert is presented to the user. This alert will be sent and presented differently depending on the client's status. The variables to present the alert in one form or another are the user's location and the available devices at that time. Therefore, a key point of this project is the possibility of integrating different interfaces, both input and output, while focusing entirely on the user.

# ÍNDICE

# 1. INTRODUCCIÓN

La tecnología ha sido muy importante en el desarrollo de muchos sistemas durante los últimos años. La calidad de vida de las personas aumenta considerablemente gracias a las nuevas mejoras e investigaciones en este ámbito tecnológico, presente y futuro de las nuevas generaciones.

Actualmente es uno de los campos de mayor inversión, ya que también tiene repercusiones tanto en ámbitos económicos como sociales. La creación de dispositivos físicos conectados a Internet favorece la elaboración de este proyecto, como las pulseras inteligentes o luces LED, entre muchos otros.

En este primer capítulo introductorio se presenta primero de todo las motivaciones por parte del autor, donde se explica brevemente el dominio del problema y aquello que se quiere solucionar y proponer. Seguidamente se presenta la idea original del proyecto y su evolución. A continuación se definen los objetivos que se quieren conseguir y por último se presenta la estructura de la memoria.

## 1.1. Motivaciones

Una de las motivaciones del presente proyecto es el profundo interés del autor por la domótica desde su infancia, sobre todo después de realizar una visita a una vivienda automatizada mientras cursaba en el Instituto, hará más de 6 años. Poder controlar mediante dispositivos electrónicos (en aquel caso una *tablet*) una casa fue uno de los factores que dieron pie a profundizar en este campo tecnológico.

Hoy en día la vida de las personas está conectada, de una manera u otra, directa o indirectamente, con Internet. Redes sociales, cuentas de correo electrónico y del banco, entre otras, seguridad del hogar… Un largo etcétera que hace el día a día impensable sin Internet. ¿Por qué no poder gestionar tu entorno y no únicamente tu casa?

Muchos sistemas permiten automatizar y actuar sobre objetos de una casa, como las persianas o las luces, entre otros. El principal problema actual es que no existe ninguna aplicación ni sistema que sea capaz de recolectar información de un usuario para tomar las decisiones pertinentes y actuar tanto en dispositivos domésticos como en dispositivos personales teniendo en cuenta una serie de variables por parte del usuario.

Con la aparición de nuevos dispositivos inteligentes en el mercado, se abre un abanico de posibilidades para la implementación de sistemas y nuevos modelos tecnológicos, a través de la programación y configuración de estos dispositivos.

Estando presente en una evolución considerable del *Internet of Things* durante los últimos años, se encuentra una gran oportunidad de seguir innovando en este

ámbito y aprender más sobre la automatización de procesos administrando una lógica razonable aplicada en segundo plano.

## 1.2. Idea original

La mayoría de las personas tienen contratado con el mismo operador tanto la línea de telefonía móvil como el servicio de Internet ADSL o fibra en casa. La idea original perseguía vincular esta relación para poder tener una fácil interconexión digital de objetos cotidianos con Internet y poder alertar al usuario de diferentes eventos.

Esta idea principal estaba enfocada en un entorno residencial, teniendo en cuenta todas las posibilidades disponibles en esta área, desde el operador del servidor hasta cualquier dispositivo capaz de alertar o comunicarse con las personas de diferentes maneras: televisión, teléfono móvil, ordenador, etc. Con este entorno definido, la propuesta era combinar el servicio de telecomunicaciones con otro secundario.

Los campos abarcados dentro del área definida eran varios, entre los que estaban Servicio Móvil, Automatización de Hogar, Sincronización de calendarios, Seguridad, Servicio de Internet y Servicios *Healthcare*, entre otros.

Empezado a trabajar, se hicieron evaluaciones previas de algunos productos, conceptos y tecnologías y se vio que esa vía no podía conducir al desarrollo de un Proyecto Final de Grado por diferentes causas: era demasiado ambicioso, insuficientemente concreto y difícil de implementar. Entonces se reformuló la idea y cambió en algo más específico, siguiendo la idea de gestión de alertas al usuario.

Por lo tanto, el enfoque principal de este proyecto es analizar tecnologías incipientes para diseñar y modelar un sistema de eventos reactivo centrado en el usuario para proporcionarle una serie de alertas cuando sea necesario según unos eventos de entrada mediante el uso de diferentes tecnologías integradas.

Actualmente se dispone de muchas tecnologías para la recolecta de información. Entre algunas más destacadas están los relojes inteligentes, las pulseras cuantificadoras, algunos servidores de internet o información proveniente de sensores, entre otras.

La idea es recoger información de estas entradas, procesarla y gestionarla en su estructura interna. El sistema debe ser inteligente, ya que dependiendo de los eventos de entrada debe decidir si es necesario alertar al usuario teniendo en cuenta varias variables que determinan el estado actual del usuario, entre las que están la localización, la hora en que se produce la alerta, las tecnologías disponibles, etc.

La idea es que el usuario pueda ser informado de cualquier suceso esté en el lugar que esté. Si resulta que se encuentra en su domicilio, se procede a una salida local en su propia casa, ya sea a través de una luz, un altavoz o la televisión, entre otras. Sino, la idea es transmitir el mensaje de alerta a otras

tecnologías que el usuario lleve consigo en ese momento, como por ejemplo el teléfono móvil o incluso alguna salida a través del vehículo si resulta que está conduciendo.

Por lo tanto, la idea general es conseguir un sistema que sea capaz de interconectar diferentes dispositivos y tecnologías entorno al usuario para estar siempre informado de los eventos que suceden a su alrededor. Para ello es importante entender y estudiar cómo diferentes procesos, aplicaciones u objetos pueden comunicarse entre ellos y auto gestionarse para proporcionar información útil y válida en tiempo real.

## 1.3. Objetivos

En este Proyecto Final de Grado el objetivo es realizar un estudio general de evaluación de la tecnología enfocada en el usuario y aplicada a la gestión de eventos en un entorno muy concreto. Por tanto, no se trata ni de un producto ni de un proyecto de investigación, sino un mero estudio al cual se le acompaña un diseño concreto con una implementación de prueba.

La meta de este proyecto es crear un sistema reactivo, robusto y consistente que sea capaz de recibir diferentes tipos de entrada, las gestione y las procese de tal manera que pueda generar salidas determinadas teniendo en cuenta información del usuario.

Para conseguir este propósito, se han definido una serie de tareas a cumplir que son las típicas cuando se hace una evaluación y exploración de una tecnología para ver cómo se podría aplicar esta tecnología a casos reales y útiles. La lista de objetivos es la siguiente:

- Describir claramente la idea.
- Evaluar las tecnologías que se podrían utilizar.
- Describir casos de uso que tengan sentido.
- Analizar casos de uso.
- Proponer sistema para los casos de uso.
- Implementar un *mockup* para probar el sistema.

Dado que es un proyecto de fin de grado y se dispone de restricciones de tiempo y recursos limitados, otro objetivo es demostrar que se ha creado un sistema modular capaz de integrar diferentes tecnologías. De esta manera, cualquier adaptación que se desee implementar y añadir al proyecto se podrá realizar de manera efectiva, según los diferentes casos de uso que convengan.

## 1.4. Organización de la memoria

La memoria del proyecto está estructurada en siete capítulos.

En este primer capítulo introductorio se han presentado las motivaciones por parte del autor, la idea general y una breve contextualización al tópico general del proyecto. Por último se han marcado los objetivos a cumplir dentro del proyecto.

El segundo capítulo muestra una descripción de la funcionalidad del proyecto, donde se presentan diferentes casos de uso analizados para dar relevancia al sistema que se desea implementar.

El tercer capítulo analiza el Estado del Arte de las diferentes tecnologías que se aplican en este sistema, donde se intenta estudiar qué hay disponible en el mercado y cómo se pueden adaptar a la idea del proyecto.

En el cuarto capítulo se presenta el diseño y la arquitectura lógica del sistema. Para entender el comportamiento de éste se analiza el flujo de ejecución, donde se evalúan los elementos necesarios para su implementación y la lógica aplicada.

En el quinto capítulo se explica cómo se ha procedido a la implementación del diseño lógico propuesto previamente. El entorno de desarrollo para la configuración, las funciones implementadas, las tecnologías y protocolos utilizados, etc. Se trata del capítulo más técnico donde se describe la programación del proyecto. Además, se presenta una maqueta de prueba del sistema para la evaluación del sistema.

El sexto capítulo describe las conclusiones, el trabajo realizado, qué se puede mejorar, el trabajo futuro y la experiencia por parte del autor.

Por último se presenta la bibliografía con todas las fuentes y referencias consultadas para la elaboración de este proyecto.

# 2. DESCRIPCIÓN DE LA FUNCIONALIDAD

En la Introducción se ha visto que el objetivo del proyecto es una idea algo difusa, ya que no se trata ni de un producto ni de una investigación. Aun así, este proyecto tiene como meta definir y diseñar un sistema de alertas enfocado en el usuario. El contenido de este capítulo se basa en concretar esa idea en requerimientos y funcionalidades que se le quieren dar al sistema.

Una parte muy significante de este proyecto es la funcionalidad que éste puede aportar a los diferentes usuarios. Por esta razón, una de las fases de mayor trascendencia y la que da valor al sistema es el análisis y el estudio de diferentes casos de uso, es decir, ponerse en el lugar de un usuario real en unas condiciones reales y ver qué funcionalidades querría tener.

Para contextualizar y englobar la idea general y aportar una visión práctica y real de la utilidad de este proyecto, se proponen diferentes situaciones donde se evalúa el comportamiento del sistema a determinados eventos. Estos eventos se proponen como entrada de datos al sistema simulando un posible caso real que le puede suceder a un usuario.

De esta manera, dadas diferentes situaciones se puede concretar el sistema y determinar qué elementos y tecnologías son necesarias.

## 2.1. Multimodalidad

Dado que el sistema que se quiere construir necesita estar alimentado por más de una fuente (tanto de entrada como de salida), es necesario adentrarse dentro del término “Multimodalidad”.

Este concepto proporciona al usuario diferentes maneras de interactuar con un sistema. Un ejemplo práctico se muestra a continuación. Antiguamente la relación persona-máquina era más estricta. En el caso de utilizar un ordenador, era necesario emplear un ratón, tener conocimientos sobre el teclado y saber medianamente cómo era el sistema operativo. Ahora, la multimodalidad permite no tener conocimientos informáticos, ya que es posible usar el ordenador mediante otro tipo de interacción, como puede ser la voz o los gestos.

La interacción multimodal habilita esta comunicación entre la persona y el sistema de una manera más libre y natural. Para ello es necesario una interfaz multimodal para interconectar a los diferentes usuarios con sistemas automatizados, tanto para las herramientas que ofrece como entradas como para las salidas del sistema.

Esta característica de disponer más de una posible entrada y salida puede provocar ambigüedades en los datos, por la cual cosa deben de tratarse correctamente para evitar desestabilizar el sistema.

Aplicando este concepto al proyecto, se puede disponer de diferentes medios y que cada medio aporte una determinada información. Un usuario interactúa con el sistema no únicamente utilizando un método concreto de entrada, sino a través de diferentes tecnologías de su entorno, de la misma manera que puede ser informado y alertado a través de una o varias tecnologías diferentes.

El entorno del usuario puede estar en continua comunicación con el sistema, aportando más flexibilidad y mejorando el rendimiento y la usabilidad. Por lo tanto, es importante tener en cuenta esta herramienta para crear un nexo entre la relevancia del proyecto y los diferentes casos de uso mediante la utilización de diferentes modalidades como tecnologías distintas.

## 2.2. Entorno de aplicación

Para captar información de los diferentes eventos de entrada y para proporcionar la salida de alerta adecuada se estudian y analizan diferentes tecnologías que pueden aportar riqueza al sistema.

Como se ha visto previamente, la idea es disponer de entradas y salidas multimodales, ya que permite diseñar un sistema que no es trivial y tiene mayor utilidad de cara al usuario. De este modo, se puede procesar información que provenga de diferentes servicios o dispositivos, al igual que se pueden sacar salidas que vayan a diferentes tipos de tecnologías.

Dentro de los campos que abarca el proyecto, se pueden agrupar componentes (tanto para las entradas como para las salidas del sistema) en diferentes categorías, aunque estas tecnologías se analizan en el próximo capítulo más en detalle cuando se estudia el estado del arte.

- Dispositivos físicos.
  Dentro de esta categoría aparecen dispositivos como altavoces que funcionan vía WiFi, bombillas LED configurables, o incluso sensores capaces de transformar magnitudes físicas en datos lógicos para poder procesarlos. Entre los sensores más interesantes para este proyecto aparecen, por ejemplo, sensores que miden la intensidad lumínica o sensores que detectan movimiento, entre otros.

- Servicios.
  El hecho de disponer de esta multimodalidad permite no sólo integrar dispositivos físicos para proceder a la alerta al usuario sino también otro tipo de tecnología en la nube, como por ejemplo cualquier servicio de mensajería que sea útil para avisar al usuario. Además, se contemplan otra serie de servicios como puede ser cualquier servidor de internet que pueda proporcionar información útil para el usuario.

- *Healthcare*.
  Existen dispositivos que permiten promover, proteger e incluso prevenir anomalías en la salud de las personas. Se considera una nueva categoría ya que es una mezcla de dispositivos y servicios que conjuntamente intentan proporcionar información relacionada con la salud y enfermedades del usuario. Un claro ejemplo se puede encontrar en las pulseras cuantificadoras que determinan datos sobre diferentes temas de salud del usuario que la lleva puesta.

Aceptando esta diversidad de tecnologías como entradas y salidas, hay que tratar esta multimodalidad en el sistema. Para ello, hay que crear procesos que estén en continua comunicación con el entorno entre los diferentes servicios y dispositivos. Cuando se trate de un dispositivo se conectará mediante su API para configurar las diferentes implementaciones. En cambio, cuando se trate de un servicio puede ser que sea a través de la API haciendo PULL o con una API que reciba eventos, o incluso a través de un modelo abstracto que se hará a través de un bróker de comunicaciones. Estos detalles se ejemplifican más adelante.

## 2.3. Ventajas del modelo

Para realizar un despliegue de los distintos posibles casos se necesita una propuesta previa que dé importancia a su funcionalidad.

Como se ha comentado anteriormente, el diseño del sistema tiene su enfoque principal en el usuario. Se necesita tener conocimiento de varios aspectos principales, es decir, varias variables que proporcionan información del usuario en cada momento para que el sistema actúe en consecuencia a estos datos.

Entre las variables más importantes se encuentran las tecnologías disponibles, la continua localización del usuario y la fecha y hora del momento de la alerta, entre otras.

Con estas variables a tener en cuenta, el usuario puede ser alertado en cualquier momento, sin tener que preocuparse de nada ya que estas alertas las recibe sin “darse cuenta” y sin tener que estar pendiente de aquello que le puede llegar a preocupar.

También es interesante para aquellos usuarios que cuidan algunos detalles, como el control de consumo, o que simplemente les interesa información útil no tan urgente pero sin la necesidad de acceder a ella personalmente.

Este sistema que se diseña y se desarrolla intenta aumentar el confort, la comodidad y la tranquilidad de las personas, otro aspecto importante de cara a la relevancia del proyecto.

## 2.4. Casos de Uso

Una vez comentadas las diferentes tecnologías utilizadas para procesar los eventos de entrada y salida, es importante describir algunos casos de uso posibles, los cuales tienen dos funciones. Por una parte, ayudar a entender la funcionalidad que proporciona el sistema. Por otra parte, ayudar a definir las funcionalidades que debe implementar.

Hay muchos casos de uso que se pueden aplicar a la idea del proyecto, y más teniendo en cuenta que el entorno tecnológico interactúa con el sistema de forma multimodal. Gracias a este concepto se pueden realizar muchas variaciones, por lo que no está limitado a un número fijo de casos de uso, sino que dependiendo las variables que se tengan en cuenta y las tecnologías disponibles se pueden proponer multitud de escenarios diferentes. Teniendo esto en cuenta, se ha trabajado en describir sólo unos pocos que sean suficientemente ejemplares y que además sean variados de tal manera que no se solapen demasiado. Además, deben ser útiles para que sirva de ejemplo en la implementación de la diversidad de áreas en las cuales se puede aplicar.

Seguidamente se listan tres casos de uso que se han considerado adecuados para cubrir un abanico amplio de posibilidades:

- Caso de uso 1: Monitorización de la Glucosa.
- Caso de uso 2: Control de Consumo de Energía en el Hogar.
- Caso de uso 3: Seguimiento de la Climatología.

### 2.4.1. ¿Por qué usar casos de uso?

El principal objetivo de la utilización de diferentes casos de uso se basa principalmente en acabar de definir los requerimientos y las funcionalidades que se desean obtener a través de este sistema. Es importante un estudio exhaustivo de varios casos reales para detallar situaciones y crear un marco de trabajo que sirva como guía para el desarrollo y las pruebas.

Para cada caso de uso, primero se define un escenario detallado del usuario con sus necesidades para no perder detalle de la ejecución y el comportamiento que el sistema debe soportar. Para cada situación, se definen como mínimo un actor que interactúa con una o varias interfaces, de tal manera que reciba una salida según unos eventos de entrada. Ayuda a la elección de diferentes elementos dentro del sistema y a crear una visión global de la lógica funcional que el sistema debe administrar acorde con las especificaciones creadas en los diferentes casos de uso.

Por otro lado, es bastante útil para que, una vez esté el diseño y la implementación del sistema finalizado, se realicen las pruebas pertinentes para comprobar que todo funciona correctamente y que el sistema responde como se espera sin errores.

### 2.4.2. Descripción de los casos de uso

A continuación se presentan diferentes escenarios que, aunque no se implementen todos los casos de uso expuestos, es una buena práctica describirlos para entender qué se espera del sistema y qué necesita para funcionar correctamente.

Para exponer los diferentes casos de uso se van a describir siguiendo la metodología expuesta a continuación en la siguiente tabla.

| CATEGORÍA | DESCRIPCIÓN |
|---|---|
| Escenario | Se presenta el entorno, la situación del usuario. |
| Datos a procesar | Se explica qué datos se procesan y se presentan posibles entradas, posibles salidas… |
| Respuesta del sistema | Se describe cómo debe responder el sistema ante tal entrada de datos al sistema |
| Objetivo | Se explica qué se pretende hacer y por qué. |
| Diagrama de interacción | Resumen de forma visual del caso de uso. |

*Tabla 1. Categoría y descripción de cada Caso de Uso*

Por último, si se considera necesario se añaden algunos comentarios adiciones del caso de uso particular que se está analizando.

#### 2.4.2.1. Caso de uso 1: Monitorización Nivel de Glucosa

##### Escenario

Se imagina que el usuario padece de diabetes. Una persona diabética necesita tener controlado y regulado el nivel de azúcar en sangre en todo momento. Estos niveles deben estar comprendidos entre 70 y 120 mg/dl para que no haya ningún problema de salud. El usuario tiene una rutina de vida normal, se mueve constantemente por su casa y tiene un trabajo de jornada completa por lo que pasa fuera de su domicilio la mitad del día. El usuario debe extraerse una muestra de sangre para analizarla constantemente durante el día para regular la azúcar en sangre.

Si el usuario dispone de una pulsera cuantificadora que determina su nivel de azúcar, estos datos se analizan automáticamente cada cierto tiempo y se envían a un servidor donde se van almacenando continuamente.

### Datos a procesar

| ENTRADAS | SALIDAS |
|---|---|
| Nivel de Glucosa | Luz de la habitación |
| Localización del usuario | Altavoz de casa |
| Hora | Teléfono móvil |
| Estado de la bombilla | |

*Tabla 2. Resumen de los datos a procesar para el Caso de Uso 1*

El sistema recoge los datos del servidor continuamente y los va almacenando en una base de datos. Cada entrada de estos datos al sistema se considera un evento de entrada.

El sistema aplica una inteligencia a este evento para decidir si es necesario alertar al usuario o no dependiendo el nivel de glucosa del usuario. Si el sistema decide que es necesario disparar una alerta, el siguiente paso es procesar la información de geolocalización del usuario para decidir cómo responder ante tal situación.

Los niveles mínimos y máximos deben ajustarse a cada usuario, ya que cada persona dispone de un metabolismo diferente y puede ser más o menos grave, aunque los valores estándar oscilan entre 70 y 120 mg/dl.

Para este caso, se disponen de unas bombillas LED que pueden cambiar de color para alertar de forma local al usuario. Otra posible salida es alertar al usuario a través de un altavoz mediante un sonido de alarma. Si resulta que se necesita una salida no local, se dispone de un Servicio de Mensajería para notificar al usuario a través de un mensaje a su dispositivo móvil.

### Respuesta del Sistema

Si el nivel de azúcar es elevado (supera el límite), dependiendo de la relevancia y la situación del usuario, se espera que el sistema le envíe un mensaje de alerta para que se inyecte insulina, siempre y cuando no se provoque en horas de sueño. Para ello, el sistema localiza al usuario en ese momento. Si resulta que el usuario está localizado en su propio domicilio, se le avisa mediante una luz que se enciende de color rojo. Si el usuario está fuera de casa, recibe un mensaje de alerta mediante el Servicio de Mensajería instalado en su móvil.

En cambio, si el nivel de azúcar está por debajo del límite aceptable, el usuario debe ser alertado para ingerir azúcares. De nuevo, si está en su casa visualizará una luz de color azul y sino, un mensaje a su dispositivo móvil con el mensaje de alerta.

Dentro de todos los posibles valores de nivel de glucosa, se establecen 5 rangos diferentes que determinan la gravedad del evento. Si resulta que el nivel de azúcar es peligrosamente bajo o elevado, la bombilla, además de iluminar el color correspondiente, hará un efecto de parpadeo para remarcar la gravedad de la situación.

Estos casos suceden siempre y cuando el usuario no se encuentra en horas de sueño. Por otro lado, si el usuario está durmiendo y los niveles de azúcar se disparan (superan el doble de lo habitual o están por debajo de la mitad), el sistema despertará al usuario mediante un sonido de alarma a través del altavoz para alertar de la gravedad de la salud del usuario.

## Objetivo

Este escenario planteado es realmente útil para aquellas personas que constantemente deben extraerse una muestra de sangre y analizarla para ver si es necesario regular los niveles de azúcar. Con este sistema, el usuario podría despreocuparse de este aspecto ya que estará siempre alertado de cualquier subida y bajada de azúcar, se encuentre donde se encuentre.

Además, no se deberá extraer personalmente una muestra de sangre cada cierto tiempo, ya que la pulsera analiza el nivel de glucosa del usuario, por la cual cosa aumenta su comodidad en el día a día.

El usuario puede estar en peligro si se olvida de analizar su nivel de azúcar en sangre. De esta manera, el usuario está al tanto de su situación constantemente gracias a este sistema.

## Diagrama de interacción

A continuación se presenta de manera visual este caso de uso específico.

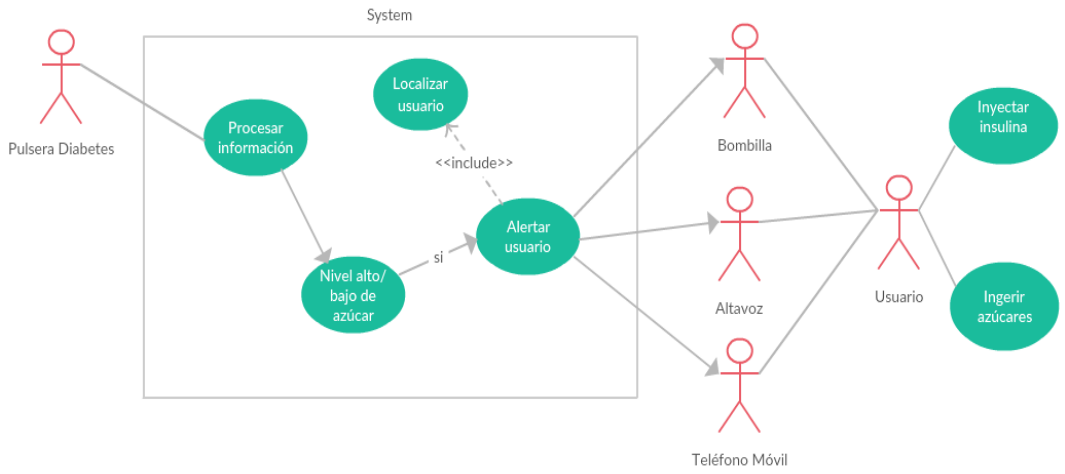



*Figura 1. Diagrama de interacción para Caso de Uso 1*

### Comentarios adicionales

Algunas modificaciones pueden aplicarse a este caso. Si el usuario está en su casa, también podría decidir si cuando los niveles de azúcar están controlados positivamente, el usuario desea ver una luz de color verde constantemente o simplemente que la bombilla esté apagada.

Dado que son datos relativos para cada persona, otra posible utilidad sería avisar con el nivel de azúcar para las horas de desayuno, comida y cena. O crear otros rangos con diferentes colores para tener aún más controlada la situación.

Por otro lado, este mismo escenario planteado puede aplicarse perfectamente a otros casos que estén relacionados con pulseras cuantificadoras y que proporcionen información del estado de salud del usuario, como puede ser el caso de la tensión arterial o la frecuencia cardíaca, entre otras.

## 2.4.2.2. Caso de uso 2: Control de Consumo de Energía en el Hogar.

### Escenario

Se imagina un usuario que vive en un domicilio particular. El usuario no tiene un control del consumo de energía en su domicilio y algunos meses la factura de recursos energéticos se dispara debido a este descontrol. El usuario quiere tener conocimiento del consumo una vez se llegue a unos límites marcados por él mismo para poder ser más ahorrador en función a la información que tiene disponible.

El usuario además es bastante despistado en algunos aspectos y se suele dejar las luces encendidas de las habitaciones, el aire encendido con las ventanas abiertas o algún grifo abierto mínimamente desperdiciando así gran cantidad de agua. Estos factores provocan un malgasto de energía que se refleja en los gastos ocasionados a final de mes.

### Datos a procesar

| ENTRADAS | SALIDAS |
|---|---|
| Nivel de consumo de agua | Luz de la habitación |
| Nivel de consumo de gas | Altavoces de casa |
| Nivel de consumo de electricidad | Teléfono móvil |
| Estado de las luces de casa | Correo electrónico |
| Localización del usuario | Actuadores |
| Hora | |

*Tabla 3. Resumen de los datos a procesar del Caso de Uso 2*

Se disponen de varios sensores en el domicilio, que determinan el consumo de recursos como agua, gas y electricidad. Además, se obtiene información sobre el movimiento que hay en diferentes zonas de la casa y el estado de las bombillas de casa. La información se introduce en el sistema y se procesa continuamente.

Otra información importante a procesar es la localización del usuario y la hora en qué ocurre la alerta para decidir qué tipo de alerta se dispara y de qué manera.

### Respuesta del Sistema

Si se detecta que las luces de casa están encendidas y no se detecta movimiento en esa zona de la casa, si se dispone de luces programables, una opción sería apagarlas automáticamente. Sino, alertar al usuario para que las apague de diferentes maneras. Si se encuentra en su domicilio, avisarle mediante el asistente virtual, como por ejemplo: “Atención. La luz de la “habitación 1” lleva encendida 5 minutos y no se detecta movimiento en la zona”. Si resulta que no se encuentra en su domicilio, enviarle un mensaje a su dispositivo móvil para alertarle que se ha dejado luces encendidas y que por lo tanto está consumiendo recursos energéticos.

Si por otro lado el aire acondicionado está encendido y se detectan que las persianas están abiertas, una opción sería o bien apagar el aire acondicionado o bien cerrar las persianas si se disponen de dispositivos programables. Sino, alertar al usuario de la situación mediante un sonido de alerta a través del altavoz. El mismo caso sucede para el grifo, que si queda abierto durante 3 minutos sin que haya presencia en la zona, se intente cerrar si se dispone de la tecnología necesaria o bien alertar al usuario de la misma manera que el caso anterior.

Por otro lado, si el sistema detecta que ha llegado a los límites establecidos de consumo de gas, agua o electricidad, una opción sería informarle a través del altavoz, si se encuentra en su domicilio, advirtiendo al usuario que ha sobrepasado los límites establecidos para prevenir gastos mayores. Otra posible opción sería que el sistema enviara un correo electrónico al usuario como mensaje de alerta ya que este tipo de notificación no es tan urgente y es mejor que quede registrada para asegurarse que el usuario puede consultar en cualquier momento los datos informativos acerca de este consumo.

### Objetivo

Este escenario propuesto alerta al usuario siempre que ocurran anomalías en el estado del consumo energético. Es útil para aquellas personas que quieren tener un control de los recursos consumidos.

Además, sirve para mantener al corriente al usuario de todos aquellos recursos energéticos que se están consumiendo sin que sea necesario, o bien porque no haya presencia en la zona o bien porque se está malgastando sin sentido.

Por otro lado, ayuda al usuario a ahorrar recursos energéticos y, en consecuencia, a regular de mejor manera la factura de cada compañía que le proporciona los recursos, ya que es alertado en todo momento de la situación actual de su consumo.

### Diagrama de interacción

A continuación se presenta de manera visual este caso de uso específico.

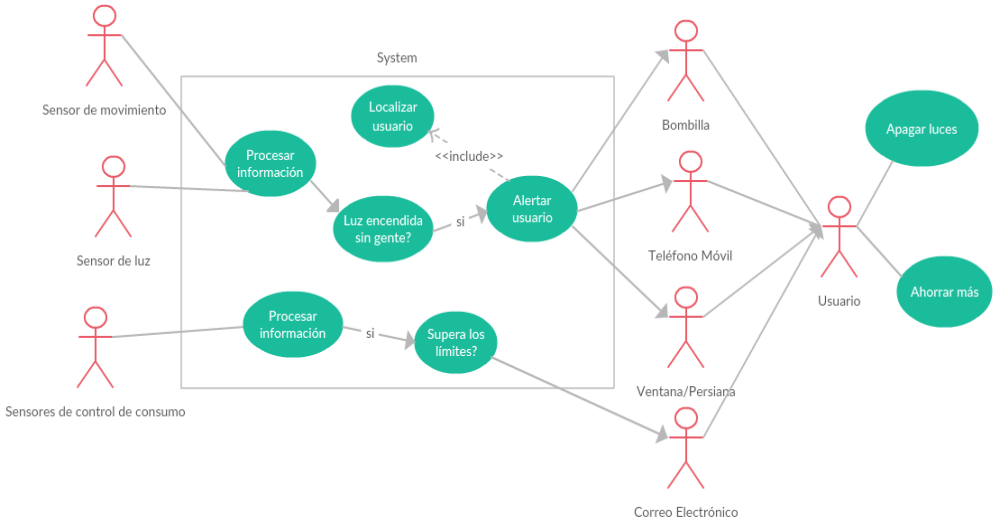



*Figura 2. Diagrama de interacción para el Caso de Uso 2*

### 2.4.2.3. Caso de uso 3: Seguimiento de la Climatología

### Escenario

Se imagina a un usuario que es alérgico al polen. Este tipo de alergia puede provocar lagrimeo, enrojecimiento en los ojos, algún que otro picor o incluso provocar otras enfermedades como la conjuntivitis. Las personas alérgicas deben ir con cuidado y procurar evitar zonas de alta concentración de polen ya que pueden derivar otras alergias y sentir todo este tipo de síntomas.

Se imagina que el usuario siente malestar con algunas reacciones que sufre por culpa de la alergia, que le provocan fatiga y dolores de cabeza cuando sale de su casa sin saber cuánta concentración de polen hay en la zona en la cual está.

La concentración de polen se mide en porcentaje. Normalmente se determinan 5 posibles niveles: nulo, bajo, moderado, alto y muy alto. A la concentración de polen hay que sumarle otros factores climatológicos que influyen directamente, como es el caso de la humedad o el viento en la zona. Cuanta mayor humedad y mayor fuerza de viento haya, peor será para las personas alérgicas.

### Datos a procesar

| ENTRADAS | SALIDAS |
|---|---|
| Concentración de polen | Luz de la habitación |
| Localización del usuario | Altavoces de casa |
| Fecha | Teléfono móvil |
| Porcentaje de humedad | Ventanas/Persianas de casa |

*Tabla 4. Resumen de los datos a procesar para el Caso de Uso 3*

El sistema debe recoger información de un servidor de meteorología para introducir en el sistema la concentración actual de polen y almacenarlo en la base de datos, considerando cada recogida de datos como un evento de entrada, donde se tiene constancia de la fecha de esta recolección de información.

Además, es importante conocer la posición en el mapa del usuario, ya que los datos provenientes del servidor deberán ser recogidos de la zona donde se encuentre el usuario.

El sistema debe aplicar una inteligencia al evento para decidir si es necesario alertar al usuario dependiendo el nivel de concentración de polen en la zona. Depende la localización actual del usuario, la alerta de salida se verá reflejada de varias maneras.

Para salida local, se puede alertar al usuario a través del color de luz de una bombilla, un altavoz inteligente (asistente virtual) o incluso el manejo de persianas o ventanas. Para salida fuera del domicilio, se puede enviar un mensaje a través de un Servicio de Mensajería al dispositivo móvil.

### Respuesta del Sistema

Una vez se obtiene la concentración de polen de la zona donde se encuentra el usuario, si es suficientemente elevada y puede provocar malestar al usuario, se procede a alertarle. Para completar la información de la alerta, se procede a actualizar la concentración de humedad ya que es un factor que influye en la gravedad de la situación.

Si resulta que el usuario se encuentra en su domicilio, y la concentración de polen es muy elevada (por ejemplo, mayor al 85%), una primera opción, si se tiene el control de ventanas o persianas, es cerrarlas automáticamente y alertar al usuario o bien a través de un dispositivo lumínico (donde se cambia el color a uno concreto) o bien a través del altavoz, avisándole de que los niveles de polen son muy elevados y que debido a la concentración de humedad puede ser más o menos grave y le pueden provocar malestar.

Dependiendo la hora del disparo de la alerta, se procede a alertar al usuario de una forma u otra. Si son horas diurnas, y el altavoz está encendido, se procede a alertarle a través de este medio, utilizando el asistente virtual que avise al usuario

"hablándole directamente a él", como por ejemplo: "Atención. Nivel de concentración de polen muy elevado y alta concentración de humedad. Evite estar en la calle lo máximo posible para prevenir efectos de malestar". Si por el contrario, resulta que son horas nocturnas, se procede a una salida lumínica, donde el color de la bombilla indicará el nivel de alerta teniendo en cuenta la información del polen y la humedad, por ejemplo, en seis estados diferentes.

En caso de que sean horas de sueño, el sistema únicamente alertará al usuario si resulta que las persianas están abiertas y el usuario no dispone de persianas programables. Sino, una opción sería cerrar las persianas automáticamente.

Si el usuario no se encuentra en su domicilio, se le alerta a través de su teléfono móvil con un mensaje de notificación, donde se incluye la concentración de polen en la zona y la peligrosidad de permanecer en la calle.

### Objetivo

Este escenario planteado es útil para aquellas personas que son bastante alérgicas y no tienen controlada la meteorología del día a día. Con este sistema, se consigue que el usuario esté alertado en todo momento de la concentración de polen en el área en la cual reside y no tiene por qué preocuparse de posibles reacciones a la alergia. Además, puede ser útil para poder programar actividades sin riesgo a sentir los síntomas o incluso elegir medio de transporte para ir al trabajo gracias al conocimiento de la concentración de polen en la zona.

### Diagrama de interacción

A continuación se presenta de manera visual este caso de uso específico.

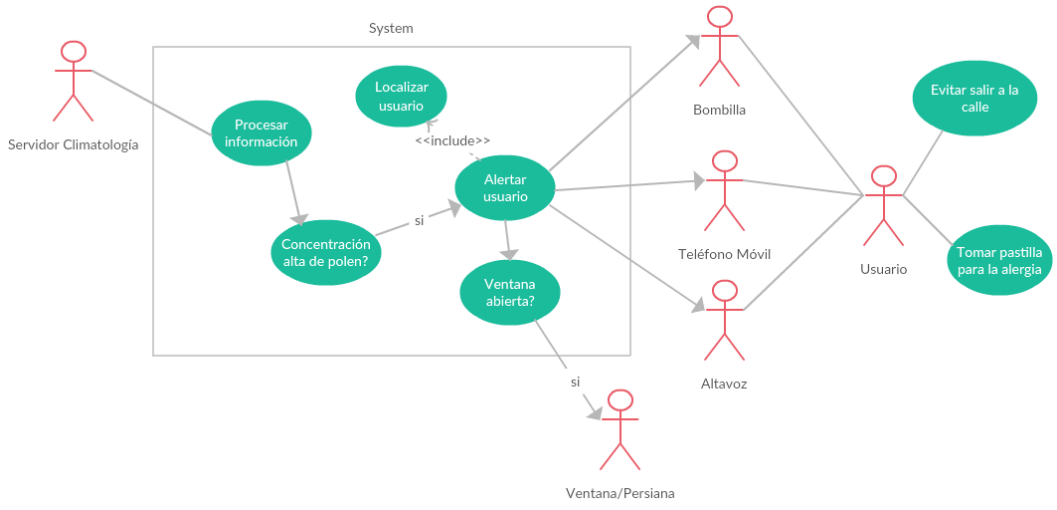



*Figura 3. Diagrama de interacción para el Caso de Uso 3*

#### Comentarios adicionales

Este mismo escenario planteado puede aplicarse perfectamente a otro tipo de alergias, como por ejemplo la radiación Ultravioleta. Con otros datos de entrada, se puede alertar al usuario de los riesgos que puede tener en la zona en ese preciso momento. O incluso aquellas personas que son alérgicas únicamente a la humedad y no al polen, siguiendo el mismo procedimiento.

Por otro lado, otra utilidad dentro de este ámbito podría ser la predicción. Alertar al usuario del tiempo que hará a la mañana siguiente para que así pueda programar la alarma a una hora concreta, estimar el tráfico o incluso elegir vehículo de transporte para ir al trabajo, ya que estas condiciones varían según si llueve o hace sol.

### 2.4.3. Conclusiones de los casos de uso

Los casos de uso anteriores son algunos de los posibles escenarios que el sistema puede integrar y ajustar a cada usuario. Existen muchos más, cualquier evento que recoja el sistema como dato se puede procesar y alertar al usuario según las condiciones que se barajen, o incluso hacer una mezcla entre diferentes casos.

El proyecto está enfocado a conseguir un sistema robusto y adaptable a cualquier tecnología. No se pueden llegar a implementar todos por los recursos y tiempo limitados en que se envuelve el proyecto, pero se demuestran algunos de ellos, donde la base del procesado de información está bien trabajada.

Por este motivo, cabe destacar la flexibilidad que se integra en el diseño ya que sirve para adaptarse a múltiples escenarios que pueden o no estar relacionados entre ellos.

Este análisis de diferentes casos de uso proporciona una visibilidad de los recursos necesarios para poder realizar el diseño del sistema siguiendo la funcionalidad que debe tener en base a estos escenarios. Una vez trabajados, se procede al análisis de diferentes tecnologías que cumplan con los requisitos que se desean implementar en el sistema.

Gracias a este estudio con unos objetivos más detallados se puede proponer un flujo de ejecución y establecer una base para el diseño abstracto del sistema, donde se puede conseguir una visión más concreta y una estructura más definida.

# 3. ESTADO DEL ARTE

Para diseñar el prototipo inicial se han analizado diferentes tecnologías necesarias para lograr un sistema compacto que cumpla con los requisitos y objetivos del proyecto. Aparecen tres elementos importantes a examinar, el primero de ellos es el sistema reactivo como estructura y funcionalidad. El segundo, las tecnologías disponibles (servicios y dispositivos) englobados dentro de la tecnología IoT para eventos de entrada y acciones de salida. Por último, el tipo de comunicación interna entre los componentes para la transmisión de datos.

Con estos conceptos claros, queda por conocer más en detalle estas tecnologías, pero, ¿qué es un sistema reactivo?, ¿qué hay actualmente en el mercado?, ¿cómo se pueden comunicar diferentes elementos dentro del sistema?, ¿de qué protocolos se dispone?

Estas cuestiones tienen que estar claras para construir la base del proyecto, por lo que se procede a una descomposición y comparación de cada ingrediente a analizar.

## 3.1. Sistemas Reactivos

La base del funcionamiento del sistema que se pretende analizar y diseñar son los Sistemas Reactivos, aquellos que cuando aparece un evento de entrada se realiza una acción de salida. Dentro de ese principio básico es necesario conocer qué son y cómo funcionan. A continuación se presenta una breve descripción y algunos ejemplos de este concepto para ayudar al lector a contextualizar el proyecto.

### 3.1.1. ¿Qué son? Descripción

Un sistema reactivo es un estilo arquitectural que permite la combinación en una sola unidad de múltiples aplicaciones o procesos individuales. Está en continua interacción con su entorno, reaccionando ante los estímulos externos. Puede funcionar de manera asíncrona, por lo que es capaz de equilibrar la carga en el sistema, lo que se conoce como "*load balancing*".

El fundamento principal de un sistema reactivo es el paso de mensajes, que permite tener temporalmente componentes desacoplados en el tiempo y en el espacio, aspectos que permiten la concurrencia y la distribución. Este desacoplamiento es básico para aislar completamente los diferentes componentes.

Los sistemas reactivos se caracterizan por ser, entre otras cosas:

- Responsivos.
  Se necesita conseguir una respuesta rápida a determinadas entradas de datos al sistema y que actúe de manera iterada y consistente. Cualquier entrada tiene que ser procesada inmediatamente para realizar la acción correspondiente, y si es necesario, instantáneamente.

- Orientado a mensajes.
  Una de las ventajas en un sistema de este tipo es la agilidad en la comunicación interna y el intercambio de mensajes asíncronos entre diferentes procesos, que ayudan a su ejecución en paralelo, provocando así mayor eficiencia y menor carga de datos en el sistema.

- Elásticos y resilientes.
  La capacidad de respuesta a bajo carga y la escalabilidad automática para satisfacer la demanda variable de manera proporcional a la que se añaden y se eliminan recursos son otras de las características que ofrece un sistema reactivo.

Algunas desventajas que tiene este tipo de sistema son que es más complejo de analizar debido a esta unificación y que por lo tanto está más sujeto a errores, por lo que puede llegar a existir problemas de seguridad en algunos casos. A continuación se presenta un esquema básico de un sistema reactivo [1].

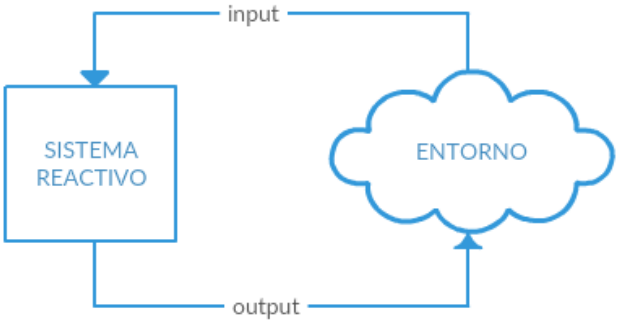



*Figura 4. Esquema general Sistema Reactivo*

Como se observa en la figura 4, dado un evento de entrada se genera una acción de salida. Se trata de un sistema lineal que puede funcionar asíncronamente aunque el modelo lógico es síncrono, ya que ante cada evento saca una respuesta. Por tanto aparece una causalidad aunque la comunicación pueda ser asíncrona.

### 3.1.2. Ejemplos de Tecnologías Reactivas.

Actualmente en el mercado hay varios estándares en cuanto al modelo acción-reacción sobre eventos. Entre los más famosos dentro de este entorno destacan *IFTTT* y *Microsoft Flow*. A continuación se presentan brevemente estos ejemplos.

#### 3.1.2.1. El servicio IFTTT

Una de las tecnologías más utilizadas y populares para este tipo de situaciones es IFTTT, acrónimo del inglés "*If This, Then That*", que en castellano significa "*Si ocurre esto, haz esto otro*", definiendo prácticamente su funcionalidad.

Se trata de un servicio web (también disponible como aplicación móvil) que se encarga de automatizar diferentes procesos y tareas a través de su propia web (incluyendo también su aplicación). Está basada en varios servicios diferentes, donde se puede actuar de manera distinta según las diferentes recetes activadas. Entendemos por recetas (o *applets*) todas las posibles acciones que se toman para comunicar diferentes servicios online entre ellos para realizar la tarea asignada que le pertoca [2].

Dispone de muchas recetas programables para muchos servicios, entre los que destacan las redes sociales como Facebook, Twitter o Instagram, pero que incluyen otros servicios como los de Google y Dropbox.

IFTTT funciona mediante el uso de cadenas de condiciones simples que se lanzan cuando se producen determinados cambios en otro servicio. Al tratarse de un software distribuido y de código abierto, cualquiera puede realizar nuevas recetas.

De uso fácil y rápido, proporciona un ahorra de tiempo para muchos usuarios, que además se despreocupan de muchos asuntos. El principal problema es que solo está permitida una acción para cada receta. Además, aunque hoy en día esté bastante más orientado a redes sociales y organización de diferentes cuentas, cada vez se están introduciendo más acciones a diferentes dispositivos inteligentes para englobar un campo de mayor cobertura respecto a su funcionalidad.

Algunos ejemplos de funcionalidad son cambiar a modo "Silencio" el teléfono móvil cuando el usuario entra en el trabajo, publicar automáticamente en Twitter la foto subida a Instagram o guardar una copia en Dropbox de las fotos subidas a Facebook, entre otros.

#### 3.1.2.2. Microsoft Flow

Microsoft Flow es otra herramienta de que permite configurar flujos de trabajo de manera automatizada entre diferentes aplicaciones y servicios. Está disponible tanto en servicio web como en aplicación móvil (iOS y Android). Permite recibir notificaciones de diferentes asuntos, sincronizar archivos y actuar según los eventos de entrada, entre otras muchas cosas [3].

Cada flujo de trabajo está dividido en varios bloques. Los desencadenadores son los eventos que disparan el flujo, las acciones son las tareas a realizar una vez se inicia el flujo, las condiciones aplican la lógica del flujo, los bucles iteran sobre las acciones y los servicios forman parte tanto del origen como del destino del flujo.

Entre los servicios más destacados aparecen algunos de Google como Google Drive y Google Calendars y otros como Salesforce, Azure y Dropbox, entre otros. También tiene a su disponibilidad realizar procesos automáticos en algunas redes sociales, como Facebook y Twitter, entre otras.

Como se ha visto, actualmente se dispone de servicios para realizar automatizaciones de procesos, pero no existe ningún sistema gestor que se encargue de relacionar diferentes tecnologías y aplicar varias condiciones lógicas entre ellas. La idea es que sea capaz de mezclar reglas para diferentes casos de uso. Además, este proyecto está más orientado a sistemas de alerta más que actuaciones en redes sociales o aplicaciones. En otras palabras, no sólo se necesita un modelo entrada-respuesta, sino uno que procese información de entrada y teniendo en cuenta varias variables, actúe en caso de que sea necesario, teniendo como enfoque principal la situación actual del usuario.

## 3.2. Dispositivos y servicios centrados en el usuario

En este apartado se analizan y se presentan algunos ejemplos de dispositivos inteligentes que pueden utilizarse tanto para eventos de entrada como para alertas de salida, como son las *Smart Lights*, las pulseras cuantificadoras y algunos sensores. Para ello, es importante contextualizar la tecnología que los engloba, el *Internet of Things*. Además, se presenta también diferentes ejemplos de servicios de mensajería para alertar al usuario.

### 3.2.1. Internet of Things

El término "*Internet of Things*" describe la interconexión de diferentes dispositivos conectados a Internet. Se trata de un concepto que define la capacidad de identificación entre dispositivos conectados a través de una red. De esta manera, un objeto puede ser controlado por otro objeto sin que ningún usuario tenga que interactuar con ellos de forma directa.

Para entender de manera clara el concepto, se ilustra con un reloj como ejemplo. Hace años, un reloj no era más que un simple dispositivo electrónico que proporcionaba la fecha y la hora, pero no tenía una utilidad más allá de medir el tiempo natural, sin la capacidad de conectarse con otras plataformas para enviar o recibir información. Hoy en día, existen relojes inteligentes que son capaces de salir de esta limitación y tienen la habilidad de relacionarse con otros dispositivos gracias a la conexión que establecen con la red de Internet.

Gracias al aumento de direcciones ocasionadas por la nueva versión del Protocolo de Internet, IPv6, se consigue viabilidad a la asignación de una IP a cualquier dispositivo electrónico de manera singular. De este modo, cualquier objeto puede tener asignado un identificador único para conectarse a la red y poder recibir y/o enviar información de manera inteligente, cosa que hace años era prácticamente impensable con la versión IPv4 [4, 5].

*Figura 5. Diagrama de una red IoT*

La figura 5 ilustra perfectamente las vías de comunicación que pueden llegar a existir entre diferentes objetos. La mayoría de estos dispositivos escuchan en un determinado puerto donde se intercambia información y datos con unos servidores externos y actúan teniendo en cuenta la información recogida de ellos.

Este mundo de interconexión es posible gracias a la comunicación que puede llegar a existir entre dispositivos simples como pueden ser sensores diminutos u otros más complejos como *smartphones*, *tablets* o incluso cámaras, relojes, pulseras inteligentes, impresoras…, un cúmulo de dispositivos que tienen una API a disposición para poder controlarlos y configurarlos.

Aunque sea una tecnología poco madura, está en alto grado de crecimiento e investigación. Por esta misma razón, cada vez son más las empresas que se dedican a crear nuevos dispositivos con esta capacidad de conexión y, aunque se trata de una tecnología sin experiencia, está llamada a ser una nueva revolución tecnológica que tendrá también grandes impactos económicos, sociales, e incluso ecológicos. Una de las frases que define el momento actual de Internet es:

"*La primera versión de Internet era sobre información creada por las personas, mientras que la siguiente versión es sobre información creada por cosas*"

Este proyecto se intenta adentrar en el *Internet of Things* para diseñar nuevas funcionalidades en un sistema de alertas al usuario.

### 3.2.2. Dispositivos IoT

Actualmente se disponen de varios dispositivos que son capaces de conectarse a Internet para recibir y/o enviar información. Algunos ejemplos útiles para este proyecto son las bombillas inteligentes, los asistentes virtuales (que también funcionan como altavoz) y algunos sensores, entre otros.

#### 3.2.2.1. Las Smart Lights

Los dispositivos lumínicos inteligentes son aquellos que permiten ser controlados o bien por una aplicación o bien por la API que disponen. Son dispositivos LED que permiten modificar los efectos lumínicos y que están conectados a Internet, recibiendo órdenes a través de la red. Normalmente necesitan un puente intermediario de mensajes. Algunos ejemplos analizados son *Philips Hue Lights* y *Lightify*, pero hay muchos en el mercado ya que es una tecnología en auge.

##### 3.2.2.1.1. Philips Hue Lights

Philips Hue es una tecnología que permite controlar bombillas o tiras de LED a través de una conexión Wi-Fi. Básicamente, son dispositivos lumínicos se pueden controlar con la aplicación móvil oficial de Philips o mediante simples solicitudes Http, utilizando la API web proporcionada por Philips que está basada en una notación JSON para enviar comandos a las bombillas.

*Figura 6. Philips Hue Kit*

Los ajustes de los diferentes parámetros de estos dispositivos están controlados mediante un *Hue Bridge*, que hace de intermediario y se encarga de recibir las órdenes y enviar información a sus dispositivos conectados a través del protocolo ZigBee. Por tanto, únicamente el bridge debe estar conectado a la red. A parte de encender y apagar los dispositivos, se pueden controlar otros parámetros como el color, la intensidad o incluso algún efecto de parpadeo. Además, permite agrupar diferentes dispositivos en escenas, donde cada escena puede programarse como se desee [6].

##### 3.2.2.1.2. Lightify

*Lightify*, de *Osram*, es otro sistema de iluminación que está conectado a la red. Permite la personalización del hogar a través de una aplicación móvil, desde donde los usuarios pueden interactuar cambiando el estado de las bombillas.

Las bombillas están conectadas al intermediario, llamado *Gateway*. Este aparato no necesita estar conectado al router sino que en cualquier enchufe de la casa sirve. El *Gateway* se comunica con las bombillas mediante el protocolo ZigBee,

como en el caso anterior. Entre los parámetros a modificar de estos dispositivos lumínicos están la temperatura de color, la intensidad y el color [7].

Aunque *Lightify* también permite la agrupación en escenas, éstas no se encienden instantáneamente y tienen un pequeño *delay* de medio segundo. Además, las transiciones entre los cambios de parámetros no son tan fluidas como en el caso de *Philips Hue*.

#### 3.2.2.2. Asistentes virtuales

Los asistentes virtuales son servicios en la nube que procesan información a través de comandos de voz. Son dispositivos formados básicamente por un altavoz y un micrófono que están siempre activados interactuando continuamente con el entorno.

##### 3.2.2.2.1. Amazon Echo "Alexa"

Amazon Echo es un dispositivo que permite a los usuarios interactuar con otros dispositivos inteligentes de manera intuitiva a través de la voz. Se trata de un asistente virtual basado en reconocimiento de voz que proporciona "habilidades" para realizar diferentes acciones, como reproducir música (*Spotify*, *Amazon Music, Pandora*), responder preguntas, leer las noticias, reportar acerca del tráfico o del tiempo, controlar otros dispositivos (*Philips Hue Lights, Samsung Smart Things, WeMo*), etc. Es necesaria su conexión a Internet para que el dispositivo responda, ya que el procesamiento de datos está basado en la nube, concretamente en los servicios de *Amazon Web Services*.


*Figura 7. Amazon Echo*

Se apoya en el desarrollo de Inteligencia Artificial y *Machine Learning*. Se pueden desarrollar nuevas "habilidades" o descargar algunas ya existentes para introducirlas en el dispositivo. Todo esto es posible gracias a la API propia que dispone, desde donde se pueden configurar funcionalidades y ajustes [8].

#### 3.2.2.3. Las Pulseras Cuantificadoras

Son pulseras electrónicas que se encargan de transformar información física en datos digitales. Están centradas en el usuario, recogiendo datos relevantes de él. Estos datos suelen enviarse al teléfono móvil o a un servidor del cual se puede extraer y analizar dicha información.

#### 3.2.2.3.1. Pulseras para controlar la diabetes

Existe un dispositivo creado por científicos mexicanos que permite analizar los resultados de los niveles de azúcar sin la necesidad de extraer una muestra de sangre regularmente. Estos datos se recogen en una aplicación en el móvil. La información de azúcar del usuario se recoge a través de un método químico que mide la glucosa en la piel, ya que son los mismos valores y parámetros que la glucosa en sangre.

*Figura 8. Pulsera glucosa*

#### 3.2.2.3.2. Pulseras *Fitness*

La mayoría de pulseras cuantificadoras hoy en día se encargan de medir la tensión del usuario, las horas de sueño, la distancia recorrida en un día o las calorías quemadas, entre las estadísticas más comunes.

Algunos ejemplos de este tipo de pulsera cuantificadora son *Polar*, *Runtastic*, o incluso marcas más conocidas como *Adidas* y *Nike*. Estas pulseras recogen los datos en una aplicación móvil, aunque cada vez más se están sumando a los servicios de "Google Fit", que ofrece una API abierta específica que monitoriza la actividad del usuario y muestra evolución de los resultados. Por lo tanto, una integración con Google Fit puede servir para recoger los resultados de algunas de estas pulseras y utilizarlo para otro tipo de programa según convenga [9].

### 3.2.2.4. Sensores

Los sensores, como bien se sabe, son objetos que transforman en variables eléctricas las magnitudes físicas o químicas del entorno. Son baratos, pequeños y bastante eficientes. Se pueden integrar a sistemas domóticos conectándolos a microprocesadores para gestionar y tratar la información.

Entre algunos sensores destacados para las alertas al usuario están los de intensidad lumínica, movimiento, humedad o temperatura, entre otros.

## 3.2.3. Servicios de Mensajería

Otras opciones de alerta que hay disponible para comunicar a los usuarios los eventos que suceden son los Servicios de Mensajería entre los diferentes

dispositivos electrónicos conectados, como *smarthphones, tablets,* ordenadores, etc.

Generalmente, el esquema del Servicio de Mensajería es como el de un sistema reactivo, pero aparece un *buffer*. El sistema es siempre remoto, entonces almacena el evento y cuando puede lo envía. Es un caso específico de un sistema reactivo, donde aparece un almacén y donde se dispone del servicio en la nube, apareciendo un acuse de recibo de notificación. El siguiente esquema ilustra de forma general el concepto.

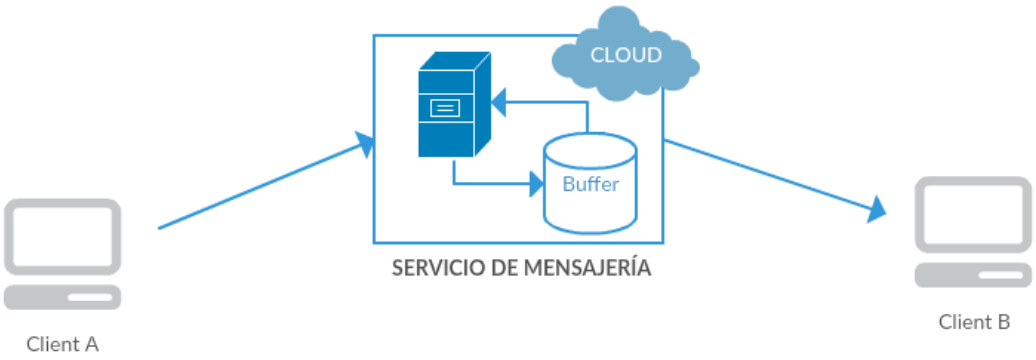



*Figura 9. Esquema general Servicio de Mensajería*

#### 3.2.3.1. Pushbullet

Se trata de una aplicación multiplataforma utilizada por millones de personas que está disponible para diferentes sistemas operativos. Pushbullet permite interconectar dispositivos propios (*smartphones*, *tablets*, ordenadores, etc.) mediante una cuenta de *Gmail*. De esta manera, el usuario puede visualizar las notificaciones en los diferentes dispositivos a medida que se van recibiendo. Además, Pushbullet permite, entre otras cosas, compartir fácilmente enlaces y archivos entre dispositivos, así como enviar y recibir mensajes (a través de una especie de chat interno dentro de la propia aplicación) o ver notificaciones entre dispositivos siempre y cuando esté activada esa funcionalidad [10].

#### 3.2.3.2. Hangouts

Hangouts es otra aplicación multiplataforma que surgió para unificar los servicios de Google Talk y Google Messenger. Proporciona un servicio de mensajería instantánea que permite la comunicación en tiempo real entre dos o más usuarios. Permite también la sincronización de dispositivos ya que las conversaciones quedan guardadas en la nube. Una buena característica es que permite visualizar si los mensajes han sido leídos o no. Además, acepta la realización de video llamadas [11].

## 3.3. Comunicaciones Asíncronas entre Agentes

La comunicación asíncrona es aquella en la cual el emisor no espera una respuesta del receptor antes de continuar realizando otras tareas, ya sean de comunicación o de otra naturaleza, ya que el procesamiento de un mensaje o evento ocurre en algún momento arbitrario, que posiblemente sea en el futuro. Esto permite incrementar la velocidad en las aplicaciones y procesos evitando bloqueos de los recursos del sistema durante periodos de tiempo no productivos.

El concepto clásico de agente está definido como un proceso completamente desacoplado de otros procesos de su entorno que interactúa con ellos de forma automática. En el caso de este proyecto, se entiende por agente un proceso desacoplado que no tiene conocimiento del resto de procesos del entorno ni de su estructura.

En cualquier sistema reactivo la comunicación entre diferentes procesos es básica. Se necesita que el intercambio de datos se realice de manera rápida y eficiente. Esta carga de datos al sistema no debe provocar sobrecarga ni congestión para permitir un intercambio fluido de información. En un entorno de agentes, múltiples procesos desacoplados colaboran a través de una red.

Una buena metodología para cumplir estos requisitos es la utilización de un “bróker” de comunicación.

### 3.3.1. Modelo de comunicación con Bróker

El intercambio de información dentro del propio sistema se realiza mediante el uso de mensajes definidos. Un bróker de mensajería, también conocido como “*middleware*”, es simplemente un intermediario entre el envío y la recepción de diferentes aplicaciones o procesos.

Este patrón arquitectónico redirecciona los datos entre sus clientes, traduciendo/transformando la información del protocolo de mensajería utilizado tanto en el remitente como en el receptor. Esta técnica permite una modularidad en el sistema que ayuda a tratar y adaptar diferentes paquetes de programa o procesos de manera independiente, sin el hecho de existir una relación directa entre ellos. Algunas de las características más interesantes son:

- Seguridad en la entrega.
- Evita entrega duplicada.
- Orden de los mensajes.
- Flexibilidad en la programación.
- Ayuda a la ejecución en paralelo.
- Desacopla información entre procesos.
- Convierte entre diferentes protocolos de transporte.

Este modelo de comunicación permite a cada proceso ser independiente del resto, ya que la información de cada uno va dirigida a un nodo central que se

encarga de gestionar y distribuir los mensajes, como se muestra en la siguiente figura [12].

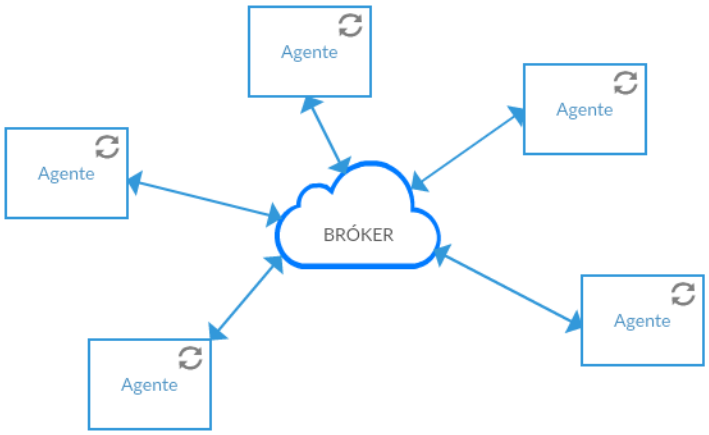



*Figura 10. Esquema de un Bróker de comunicaciones*

De este modo se obtiene un entorno compuesto por agentes sin necesidad de que cada uno de ellos tenga conocimiento de la existencia de los otros, estando totalmente desacoplados.

Por los motivos anteriormente expuestos se considera que el uso de un bróker de comunicación es una buena solución para aquellos escenarios donde se requiere gestionar eventos en tiempo real integrando diferentes tecnologías, ya que de esta manera los procesos no necesitan tener conocimiento de la estructura interna del sistema.

### 3.3.2. Ejemplo de protocolo: MQTT

Un ejemplo que utiliza el modelo de comunicación con bróker para transmitir la información y fluir la comunicación entre diferentes procesos es MQTT (*Message Queue Telemetry Transport*).

MQTT es un protocolo muy simple y ligero en el mundo de *Internet of Things* que utiliza una comunicación machine-to-machine (M2M) y que se lleva a cabo a través de TCP/IP. Fue inventado por Andy Stanford-Clark (IBM) y Arlen Nipper (Arcom) en 1999. Se utiliza para transmitir información y fluir la comunicación entre diferentes procesos. A día de hoy es un protocolo realmente útil para aquellos escenarios que requieren redes de bajo ancho de banda, alta latencia o no confiables.

Se trata por lo tanto de un sistema de mensajería extremadamente ligero basado en publish/subscribe que minimiza el ancho de banda y los recursos requeridos del dispositivo [13].

Como se ha mencionado anteriormente, se trata de un protocolo basado en publish/subscribe. Los dispositivos finales que están conectados a la red de

MQTT son los “Clientes”. Aparecen dos tipos de clientes, dependiendo si envían o reciben mensajes.

- Publicadores (*publishers*), encargados de enviar mensajes.
- Suscriptores (*subscribers*), encargados de recibir mensajes.

La comunicación entre diferentes clientes se realiza a través de temas, “topicos”. Esta comunicación puede ser 1:1 o 1:M. Los tópicos disponen de una estructura jerárquica en la cual cada jerarquía viene separada por una barra (‘/’). De esta manera, los publicadores envían mensajes a un tópico concreto (que ellos mismos pueden crear) y los clientes suscritos a dicho tópico reciben el mensaje.

La arquitectura de MQTT está basada en una topología de estrella. No existe comunicación directa entre clientes. La gestión de la comunicación entre diferentes clientes se lleva a cabo a través de un servidor, el *Bróker*, que se encarga de hacer de nodo central entre clientes. Su función se basa principalmente en captar los mensajes enviados por los publicadores con el tópico asignado redirigirlos únicamente a los clientes suscritos a dicho tópico.

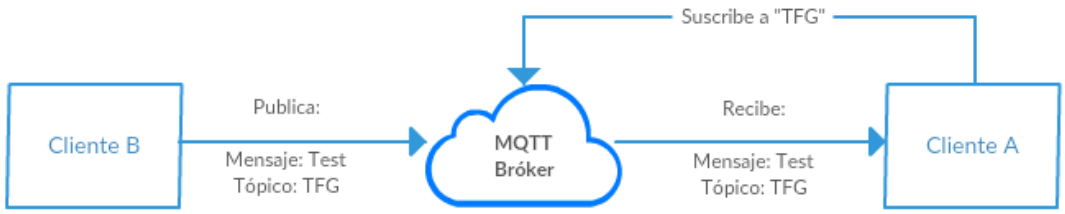



*Figura 11. Esquema de funcionamiento del bróker MQTT*

Un ejemplo sencillo se muestra en la figura 11. El cliente A se suscribe al tópico “TFG”. Otro cliente B publica un mensaje con el tópico “TFG”. Éste lo gestiona el Bróker, que lo redirecciona a todos aquellos clientes que están suscritos a ese tópico. Por lo tanto, el cliente A recibe el mensaje.

MQTT intenta a su vez garantizar la confiabilidad y seguridad en la entrega de mensajes. Para ello dispone de un parámetro importante, la Calidad de Servicio (*Quality of Service*, *QoS*), que ajusta los niveles de garantía en la transición de mensajes entre publicadores y suscritores. Tenemos tres posibles opciones de configuración de la calidad de servicio:

- QoS0
  El emisor envía el mensaje sin guardarlo en memoria y no espera ningún tipo de mensaje de confirmación. Proporciona la misma garantía que la capa TCP inferior.


*Figura 12. Calidad de servicio QoS0 en MQTT*

- QoS1
  El emisor envía el mensaje y lo guarda en memoria hasta que reciba confirmación (PUBACK) por parte del receptor. Si no se recibe este mensaje de confirmación se vuelve a reenviar el mensaje, y así continuamente hasta que el receptor envíe la confirmación, que hará borrar de la memoria del emisor el mensaje y se dará por concretado. Por este motivo puede haber duplicación en el envío, pero se asegura la llegada del mensaje en el receptor.

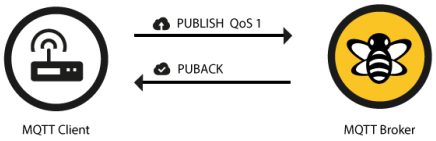



*Figura 13. Calidad de servicio QoS1 en MQTT*

- QoS2
  Es el nivel más seguro ya que evita duplicados pero también el más lento debido a todas las confirmaciones. El emisor envía el mensaje pero esta vez con un identificador que es único para cada emisión y lo guarda en memoria. Cuando el receptor lo recibe, guarda el identificador y responde con un PUBREC. Cuando el emisor recibe esta confirmación, borra el mensaje de la memoria porque ya ha sido captado con éxito pero envía al receptor una confirmación de recepción (PUBREL). En este momento se elimina el identificador en el receptor y vuelve a confirmar al emisor con un PUBCOMP. Una vez recibido, se elimina también el identificador en el emisor.

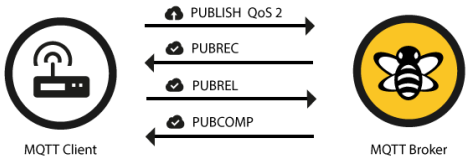



*Figura 14. Calidad de servicio QoS2 en MQTT*

Para la utilización del servicio en local, es aceptable una calidad de servicio *QoS0* ya que se conoce la red y se tiene una conexión estable entre los diferentes clientes, tanto publicadores como suscritores.

MQTT aparece como protocolo de mensajería en diferentes servicios como Facebook Messenger y Amazon Web Services, entre otros.

#### 3.3.2.1. Ejemplo de implementación: Mosquitto

Un ejemplo de implementación que está basado en el protocolo MQTT para un modelo de comunicación con bróker es *Mosquitto*, un servicio de mensajería de código abierto que se implementa sobre el protocolo MQTT. Propone un método de distribución de mensajes basado en *publish/suscribe*. *Mosquitto* es un proyecto de Eclipse que está escrito en C, pero dispone de varios módulos y librerías para diferentes lenguajes de programación [14].

Dispone de las siguientes ventajas:

- Tamaño reducido.
- Paquetes de información con carga mínima.
- Distribución de datos a diferentes destinos de manera eficiente.
- Bajo consumo de recursos.
- Rapidez en la entrega.
- Acepta multitud de clientes.
- Fácil de configurar y utilizar.
- Continúo feedback entre clientes suscritores y bróker central.
- Soporta todos los parámetros de QoS.

# 4. DISEÑO LÓGICO

El análisis del diseño lógico es una parte fundamental para conocer los requerimientos que debe tener el proyecto. Además, ayuda a la elección de las diferentes tecnologías dentro del sistema, según las funcionalidades que sean necesarias.

En este capítulo se desglosa el proyecto en diferentes componentes generales y se establece la lógica funcional. Para ello, primero de todo se procede al análisis de su estructura interna, que se descompone en varias partes diferenciadas para analizarlas separadamente. Para entender el funcionamiento interno de los diferentes procesos que existen se presenta el flujo de ejecución del programa. Seguidamente se muestra la arquitectura lógica del sistema, con las distintas tecnologías que se necesitan para su futura implementación. Por último se presentan algunos procesos lógicos internos del sistema que son necesarios entender para visualizar mejor el comportamiento global de los diferentes procesos.

Un primer diagrama que muestra el diseño más general se representa de la siguiente manera, donde el sistema recibe diferentes entradas de distintos dispositivos y servicios, procesa la información y efectúa una salida si corresponde a través de otras tecnologías.

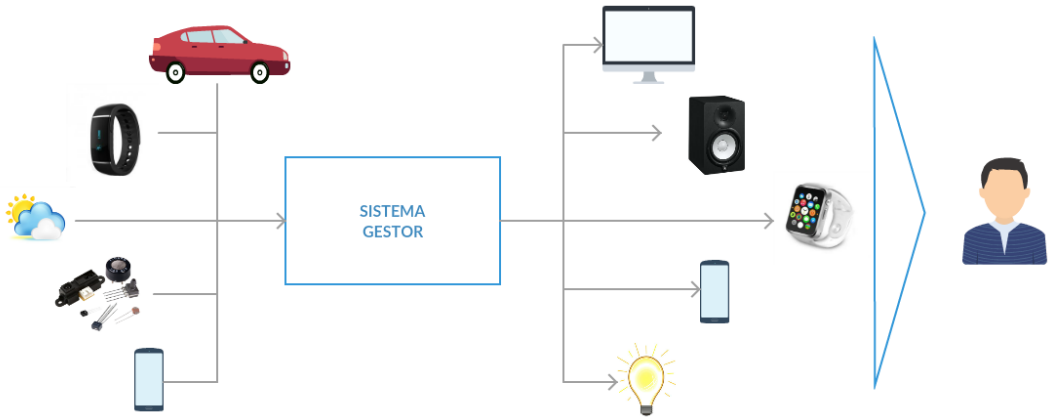



*Figura 15. Esquema abstracto del sistema*

Como se ha comentado anteriormente, el usuario es un factor determinante. Sin usuario no se toman decisiones. Por eso mismo la salida está conectada directamente con él. Esto no quiere decir que todas las acciones de salida se disparan al usuario, sino que dependiendo del estado y las características del evento se va a utilizar el mejor o el conjunto mejor de dispositivos o servicios que se vayan a usar para alertarle.

## 4.1. Modelo del sistema

A gran escala, el esquema más abstracto que representa la idea del sistema es el siguiente.

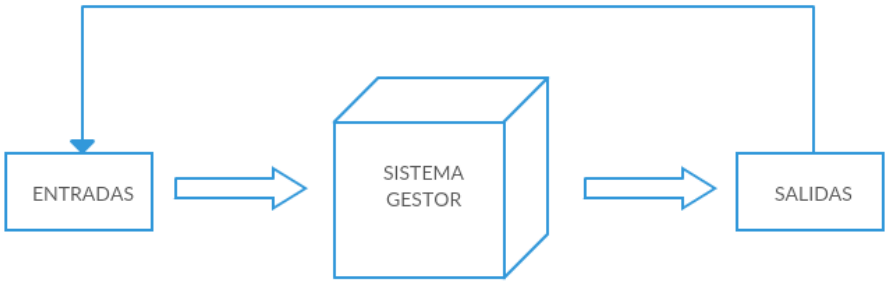



*Figura 16. Esquema general del sistema*

Como se observa en la figura 16, este esquema se aplicaría a cualquier sistema de control general, con unas entradas, unas salidas y una retroalimentación, pero no lo es. Se trata de un sistema reactivo, que dado un input saca un output si lo considera necesario.

Para desglosar este esquema se procede a analizar su estructura para descomponerlo en tres capas diferenciadas:

- Interfaz con el entorno.
  Los eventos de entrada se captan mediante fenómenos físicos (comentados en apartados anteriores) y se convierten en eventos lógicos una vez entran en el sistema. De la misma manera, se necesita una transformación de evento lógico a evento físico para la acción de alerta de salida. La inclusión en el sistema de estas interfaces permite esta transformación del mundo físico a lógico y viceversa.

  En el primer esquema de este capítulo, la figura 15, se pueden observar algunas interfaces tanto de entrada como de salida.

- Núcleo reactivo.
  El núcleo reactivo es el eje principal del sistema. Se encarga de revisar datos almacenados y actualizarlos y generar acciones en función de unas reglas de negocio iterativas. En esta capa aparecen todos los elementos que se ocupan de la lógica del sistema.

- Manejo de datos.
  Otro de los aspectos a considerar se basa en la comunicación interna del sistema y el procesado de información y datos que el núcleo reactivo solicita.

Para especificar más el modelo, se reformula el esquema anterior y se mapean algunos elementos para ser más específico en algunos aspectos.

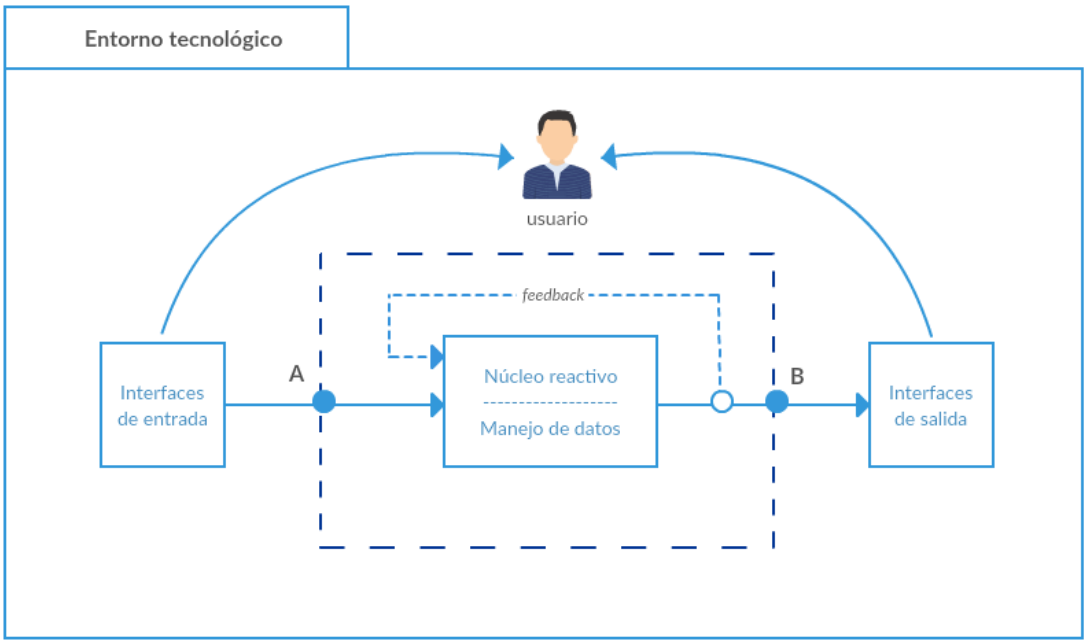



*Figura 17. Diagrama general detallado*

Como se observa en la figura 17, los únicos puntos de interacción con el exterior del sistema son la entrada de datos y las alertas de salida que forman parte del entorno tecnológico del usuario. Aunque las interfaces no forman parte íntegramente del sistema, sí que es necesario adaptar la información que proporcionan los diferentes agentes para introducirla mediante unos procesos de conversión. Estos procesos delimitan la frontera del sistema, tanto para los eventos de entrada (frontera A) como para las alertas de salida (frontera B).

Así como antes se veía una retroalimentación directa entre la entrada y la salida, en este esquema se puede ver cómo la retroalimentación queda claramente dentro del sistema, donde la salida retro acciona el núcleo reactivo mediante el manejo de datos.

Además, se puede identificar al usuario como pieza fundamental del proyecto, ya que puede afectar a determinadas entradas y recibirá las correspondientes salidas según su situación en el momento del disparo de la alerta, estando en continua interacción con el entorno tecnológico.

Teniendo esta estructura definida, este proyecto centra más atención en el núcleo y el manejo de datos, de tal manera que sea adaptable a cualquier tipo de interfaz con el entorno físico y que sus propias reglas fluyan dentro del sistema de manera correcta mediante el tránsito de información que circula dentro del mismo.

Para entrar más en detalle del modelo interno del sistema, a continuación se amplía el núcleo reactivo, donde se pueden observar diferentes elementos.

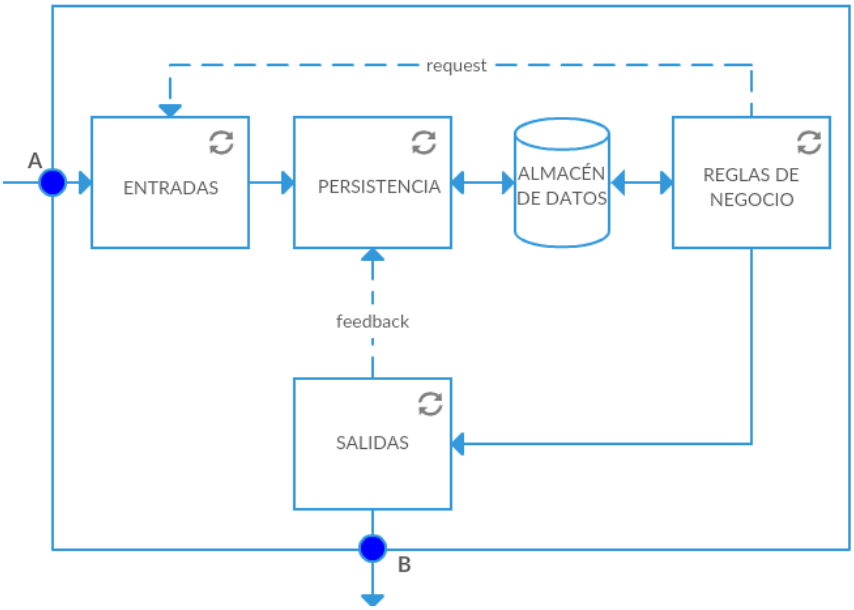



*Figura 18. Esquema lógico del núcleo del sistema*

Como se observa en la figura 18, las fronteras que delimitan las entradas y salidas entre el sistema y el entorno tecnológico quedan bien marcadas. En este diagrama se pueden visualizar los elementos lógicos más generales: el proceso de entrada, el proceso de persistencia, el almacén de datos, las reglas de negocio y el proceso de salida. Además, se puede ver claramente dónde queda definida la retroalimentación del sistema, y cómo el propio sistema puede pedir actualizaciones al proceso de entrada.

Se trata de un esquema puramente lógico y por lo tanto no aparecen elementos físicos. El manejo de datos y la comunicación interna del sistema se definen más adelante, cuando se analiza el flujo de ejecución. Aun así, se adelanta que todas estas comunicaciones viajan físicamente a través de un bróker de comunicaciones.

Por otro lado, es importante explicar que este diagrama en un entorno complejo requiere de varios elementos que pueden estar distribuidos. Por este motivo hay que mapearlo de una forma más compleja. Para tener claro cómo realizar este mapeo, dado que esto se hará más adelante, conviene analizar bien el flujo de ejecución.

## 4.2. Flujo de ejecución

El flujo de ejecución ayuda a estructurar y los elementos principales básicos del modelo del sistema. Estos elementos se proponen y se diseñan teniendo en cuenta los requerimientos, funcionalidades y restricciones que se han analizado más en detalle cuando se definieron los Casos de Uso. La lista de elementos es la siguiente:

- Interfaz de entrada
- Receptor de entrada
- Canal
- Proceso Almacén
- Almacén
- Motor de decisión
- Receptor de salida
- Interfaz de salida

El siguiente diagrama detalla el flujo de ejecución, donde se pueden visualizar estos componentes y las relaciones entre ellos.

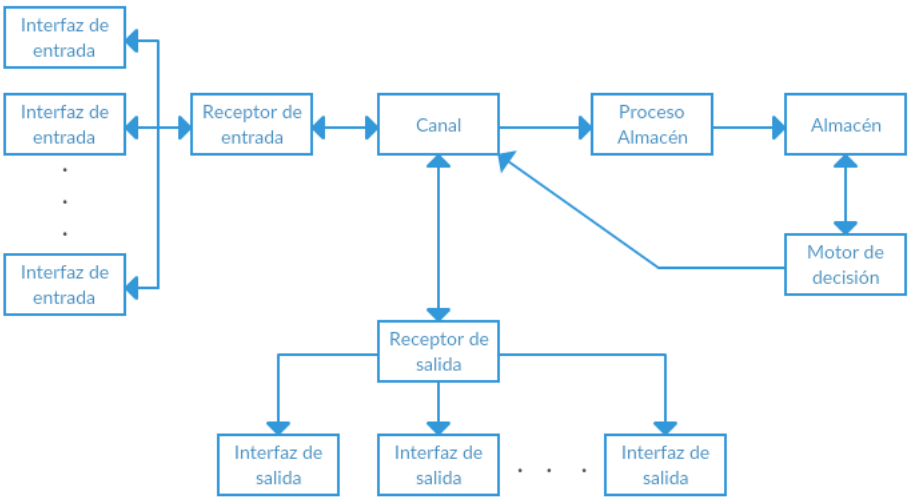



*Figura 19. Diagrama del flujo de ejecución.*

Como se observa en la figura 20, se dispone de varias interfaces de entrada que captan los diferentes eventos. La información lógica se introduce en el sistema cuando los procesos de recepción de entrada reciben esta información (única y diferente para cada tecnología que aporta datos) y los envían al canal de comunicación interno del sistema.

El Canal no es más que un distribuidor de mensajes dentro del propio sistema. Se necesita que sea rápido, en tiempo real, y capaz de desacoplar mensajes entre elementos, ya que éstos no tienen por qué conocer cómo se trata la información entre ellos.

Este primer flujo de mensajes que proviene directamente de los datos del evento los recibe el Proceso Almacén. Se encarga de recibir datos y guardarlos en el almacén. Para conseguir un almacén sólido, debe ser persistente en el tiempo para no perder información en caso de cualquier interrupción en el sistema. El almacén debe reunir y acumular toda la información necesaria en torno a la cual gira el núcleo del proyecto.

El Motor de Decisión del sistema contiene las diferentes reglas de negocio que provocan la decisión de alertar al usuario. Está en continua iteración accediendo

a los datos del almacén, editando y eliminando cuando lo considera necesario. Este motor contiene la lógica principal del sistema reactivo.

Dado que es un proyecto centrado en el usuario, se necesita saber su localización en todo momento, así como otras variables. A pesar de que la información de localización del usuario se va recolectando de algunas de las interfaces de entrada periódicamente, también es necesario poder pedir actualizaciones en cualquier momento si los datos que se tienen en el almacén no son fiables (o no están actualizados). Por lo tanto, este motor de decisión debe ser capaz, a su vez, de poder enviar peticiones al Receptor de Entradas.

Si durante la revisión y exploración de datos el Motor de Decisión obtiene los requisitos necesarios para disparar una alerta, éste vuelve a utilizar el canal para enviar la información de la alerta junto con la localización del usuario y los parámetros convenientes al receptor de salidas. Si resulta que no dispone de la información necesaria, pide actualizaciones a través del Canal.

El Receptor de Salidas debe decidir cómo actuar y realizar la salida correspondiente a la interfaz de salida definida para esa alarma, según las variables en aquel momento que le han llegado a través del Motor de Decisión. Otra función de este receptor es enviar nueva información al Proceso Almacén a través del Canal, que actualiza campos en función de los datos por parte del emisor.

Para ver en detalle y de manera más visual el comportamiento y las funciones que tiene cada componente se presenta la siguiente tabla.

| COMPONENTE | FUNCIONES |
|---|---|
| Interfaz de entrada | Proporcionar información y datos de entrada |
| Receptor de entrada | Captar información y datos<br>Enviar información/datos al Proceso Almacén<br>Recibir información del Canal |
| Canal | Re direccionar información |
| Proceso Almacén | Recibir información y datos del Canal<br>Crear, editar datos del almacén |
| Almacén | Crear, editar, eliminar datos<br>Persistir en el tiempo<br>Recuperar información |
| Motor de decisión | Contener reglas de negocio (lógica)<br>Actuar sobre Reglas de Negocio<br>Acceso a los datos (Crear, modificar, eliminar)<br>Pedir actualizaciones<br>Enviar información por el Canal |
| Receptor de salidas | Recibir del canal<br>Tomar decisión de salida<br>Enviar datos por el Canal |
| Interfaz de salida | Enviar información/datos a salidas |

*Tabla 5. Componentes lógicos con sus respectivas funciones*

## 4.3. Arquitectura lógica

Una vez definido el funcionamiento interno del sistema se procede a la implementación, pero no se puede implementar directamente si no que se necesitan diferentes componentes específicos que cumplan los requerimientos de los elementos del sistema.

Por lo tanto, estos elementos vistos anteriormente en el flujo de ejecución deben mapearse para conseguir un conjunto coherente de patrones que definan un marco determinado y evidente para posteriormente interactuar con el código fuente del software.

Para conseguir este objetivo se procede a construir la arquitectura lógica del sistema, es decir, el diseño de más alto nivel en cuanto a la estructura de un sistema se refiere, que es el conector que une el diseño con la implementación, determinando de manera abstracta todos aquellos componentes que suponen un cargo de trabajo o que realizan alguna tarea determinada. Para cada componente debe existir información acerca de las interfaces que posee y la comunicación que existe entre diferentes elementos.

Por un lado, la persistencia de datos en el sistema es vital para que éste sea robusto. Se necesita de un almacén de datos que tenga una conectividad segura, que sea capaz de indexar debido a la cantidad de eventos que se pueden llegar a acumular y que el manejo de los elementos del almacén sea fácil de manipular y gestionar. El almacén por lo tanto se mapea en una base de datos relacional, la cual estará gestionada por un Sistema Gestor de Base de Datos (SGBD) SQL.

Por otro lado, dado que aparecen múltiples procesos y se necesita intercambiar información fluidamente entre ellos, se necesita un bróker de comunicaciones ya que ayuda a la ejecución en paralelo, desacopla información entre procesos y aporta flexibilidad. El canal debe ajustarse a cualquier situación que el sistema requiera. Por este motivo el canal se mapea en un bróker de comunicaciones, que estará gestionado bajo el protocolo MQTT.

Los elementos restantes son procesos que actúan o bien con la base de datos o bien con el bróker de comunicaciones.

Con estos elementos definidos, se mapea el flujo de trabajo en la siguiente arquitectura lógica, donde se observa cómo se organizan y cómo se integran los diferentes componentes lógicos del sistema:

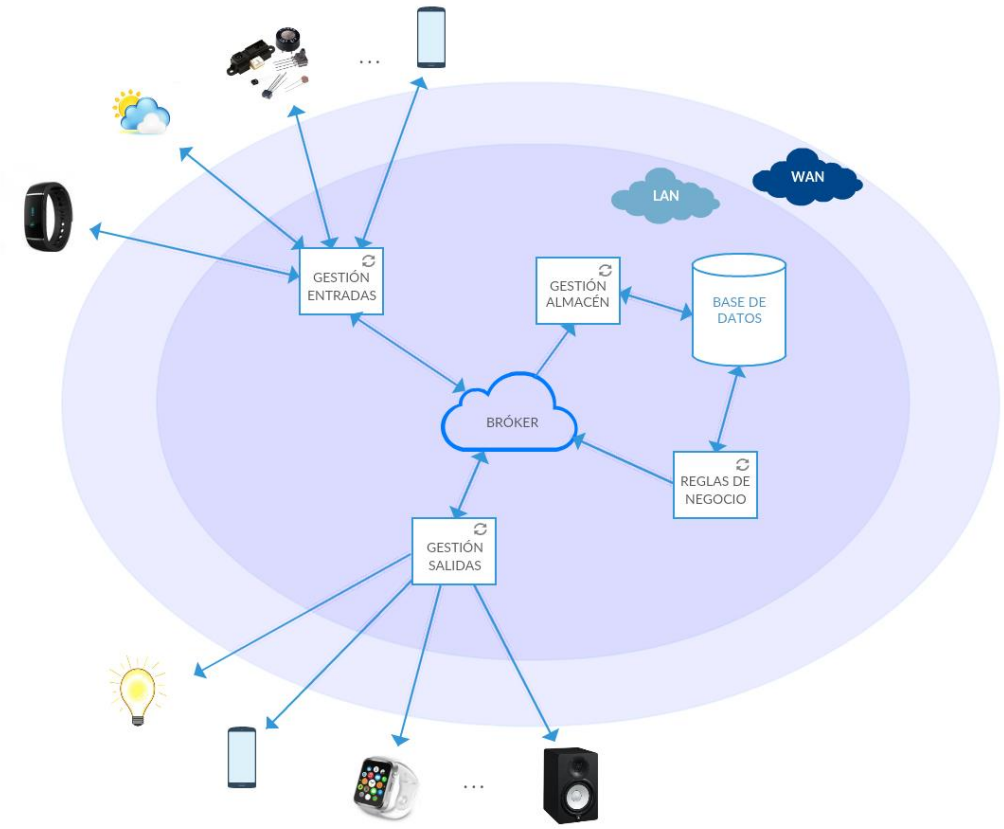



*Figura 20. Arquitectura lógica del sistema*

## 4.4. Procesos lógicos

A parte de visualizar la arquitectura con sus elementos y componentes estructurados, dentro del núcleo reactivo se aplica otra lógica al sistema que procesa información y decide qué acciones realizar en cada momento.

### 4.4.1. Entrada de datos al Almacén

Uno de los puntos a tener en cuenta es la lógica que se aplica en el elemento Gestión Almacén, que recibe multitud de eventos a través del canal proveniente del elemento Entradas.

Para cada tipo de servicio puede haber más de una fuente de información. Por ejemplo, el sistema puede comprobar el nivel de concentración de polen en varios servicios integrados, al igual que la Localización del usuario, comentado más adelante en el apartado 4.4.3. Por este motivo, el sistema debe ser capaz de decidir cuál es la información más exacta o la que más se ajusta en ese momento para insertar el evento en la base de datos. Para ello, se necesita tener varios procesos que determinen las características del evento. A continuación se muestra un esquema general de esta estructura.

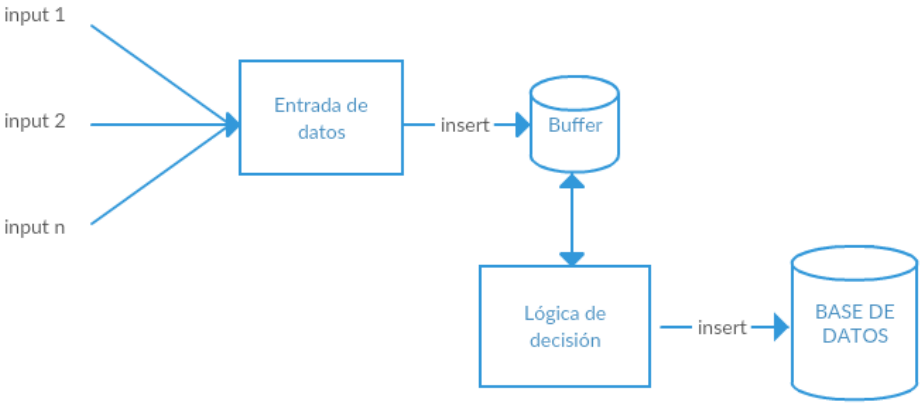



*Figura 21. Esquema general de la lógica de decisión de eventos*

Como se observa en la figura 21, en un primer lugar aparece la recepción de datos, que se encarga únicamente de almacenar los datos de entrada que le llegan en una cola de eventos clasificados según el tipo de evento de entrada. Por otro lado, aparece otro componente que aplica la lógica de decisión del sistema. Éste se encarga de analizar periódicamente estas colas y decidir entre toda la información disponible. Una vez toma una decisión, inserta el evento en la base de datos. Además, se encarga de modificar las colas, eliminando elementos cuando considere necesario.

De todos los tipos de servicio que proporcionan información sobre el mismo evento, únicamente uno es insertado en la base de datos. Es importante aplicar una buena lógica de decisión ya que las Reglas de Negocio comprobarán y gestionarán los eventos que se han decidido como correctos en ese momento.

### 4.4.2. Reglas de Negocio

Este sistema no es puramente determinista, en el sentido de que ante tal evento siempre va a suceder una alerta, sino que toma una decisión teniendo en cuenta una serie de variables, entre las que están las siguientes, aunque luego se pueden añadir más.

- Tipo de la alerta
- Localización del usuario
- Hora de la alerta
- Gravedad de la alerta

Para esta toma de decisiones complejas habrá un proceso que se encargue de gestionarlas. Existe por tanto un componente que se creará como lo que se conoce “Reglas de Negocio”.

Estas Reglas de Negocio deben revisar continuamente los eventos de entrada en el sistema, analizarlos y decidir si cumplen con los requisitos indispensables para disparar una alerta. Este componente es uno de los más importantes dentro del sistema, ya que a parte debe actualizar cada evento tratado. Si resulta que el

evento cumple con las condiciones necesarias para alertar al usuario, revisa y compara las diferentes variables para decidir si es necesario actuar o no. En caso de no disponer de información suficiente, pedirá actualizaciones a través del bróker.

Estas reglas son variables en el tiempo, no son fijas. Además, hay una cierta inteligencia. La implementación no resulta trivial y habrá componentes que permitan definir estos conceptos. Por otro lado, cabe destacar que la toma de decisiones está distribuida entre los componentes de salida y los componentes de las Reglas de Negocio, donde cada componente revisará una serie de variables.

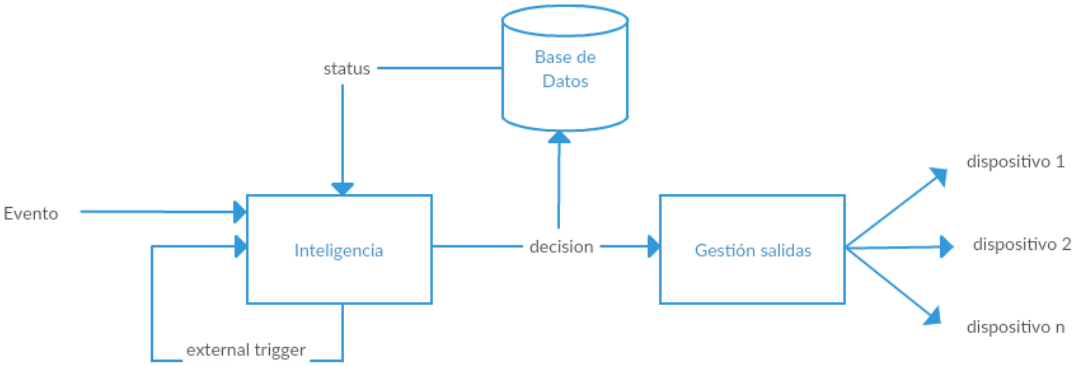



*Figura 22. Esquema lógico Reglas de Negocio*

La figura 22 muestra de manera visual la estructura lógica de las Reglas de Negocio, que se explica a continuación.

Dado un evento de entrada, se procesa la información con una determinada inteligencia. La regla se alimenta del estado, que está almacenado en la base de datos y que pueden ser varias variables. Entonces puede realizar una retroalimentación o tomar ya la decisión de salida. Una vez tome la decisión, actualiza el estado en el almacén de datos. El evento llega por lo tanto a la gestión de salidas, que también revisa una serie de condiciones, entre las que está el momento en que sucede, para escoger la mejor alerta posible para el usuario, según las condiciones recibidas en ese momento.

Por lo tanto, se puede ver que el sistema no es reactivo determinista, sino que depende de una inteligencia y que además se puede realimentar a sí mismo.

### 4.4.3. Localización del usuario

El mensaje de alerta se dispara por una de las interfaces de salida, pero ¿cómo sabe el sistema cuál de todas escoger? Como se ha comentado previamente, la localización del usuario juega un papel muy importante dentro de este proyecto, y es vital conocer su posición en el mapa para efectuar la salida oportuna. Esta localización se puede saber mediante varios dispositivos y/o servicios, por lo tanto tiene que haber varios procesos que se encarguen de recuperar toda esa

información y proporcionar unos datos de localización del usuario para almacenarla en el sistema. Todo este proceso no es trivial y puede provocar inconsistencia, por lo que es necesario un análisis detallado.

Se procede a conseguir esta información a través de varias maneras diferentes, como se muestra en la figura 23.

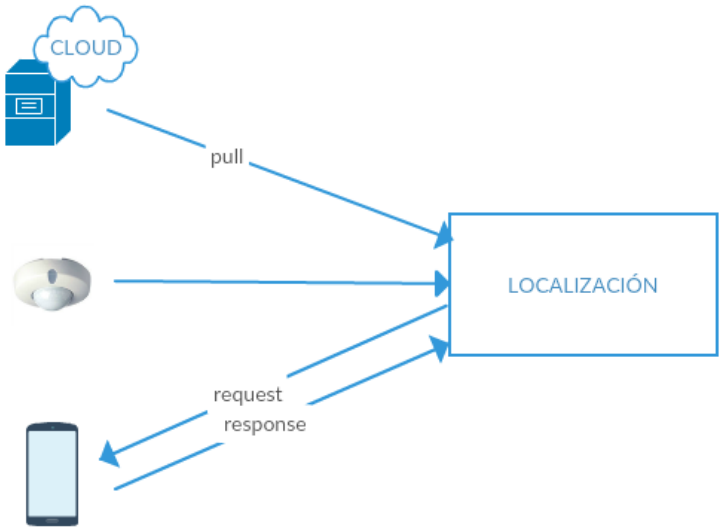



*Figura 23. Maneras de conseguir localización del usuario*

- Sensores.
  Disponer de varios sensores de movimiento en el domicilio del usuario que vayan recolectando información cada cierto tiempo, entrando estos datos en el sistema.

- Servicio de Geolocalización.
  Cualquier *Smartphone* dispone de aplicaciones como *Google Maps* que permiten recuperar información de localización a través de unas coordenadas GPS. Puede haber varios servicios que proporcionen información de geolocalización. Se puede conseguir de dos maneras diferentes:

  - Tecnología PULL.
    Recibir información cada cierto tiempo por parte del servicio de geolocalización a través del dispositivo móvil.

  - Petición.
    Si el sistema procede a ejecutar una alerta, en caso de no tener información de localización actualizada o no sea fiable, el sistema deberá solicitar esta información directamente al dispositivo para actualizar su localización inmediatamente.

Tanto la información proveniente de los sensores como de los servicios en la nube introducen datos de geolocalización del usuario cada cierto tiempo. En

cambio, si el sistema lo considera necesario, cuando la información que obtiene para la variable localización es insuficiente se procederá a realizar peticiones directamente.

Con estos métodos se obtiene mayor fiabilidad para saber dónde está ubicado el usuario siempre que se considere oportuno para tomar una decisión respecto a la acción de salida.

# 5. IMPLEMENTACIÓN FÍSICA

Una vez se dispone de un marco claro de trabajo con unos requerimientos físicos y lógicos del sistema y con la estructura bien definida se procede a la implementación del mismo.

En este capítulo primeramente se expone el entorno de desarrollo en el cual el proyectista ha estado trabajando en la elaboración del sistema y las tecnologías utilizadas. Luego, se representa la planificación que se ha seguido durante el proyecto. Posteriormente se presenta la descripción técnica del sistema, donde aparece la especificación de los diferentes componentes de una manera más detallada y técnica, así como la elaboración de una maqueta para el testeo y análisis de resultados.

## 5.1. Entorno de desarrollo

Con la estructura lógica clara, y una vez identificados los elementos que se necesitan para la implementación del proyecto, toca decidir qué tecnologías son las más adecuadas para cumplir con los requisitos funcionales que el diseño lógico propone y ponerlas en funcionamiento, teniendo en cuenta el entorno de desarrollo escogido.

De todos los posibles dispositivos y servicios analizados, sólo algunos de ellos se han integrado en el sistema, ya que se dispone de recursos y tiempo limitado. Aun así, se ha intentado combinar algunos servicios con algunos dispositivos para cubrir diferentes escenarios.

La implementación del sistema se ha desarrollado utilizando una máquina virtual bajo el sistema operativo Ubuntu. Dentro de este ámbito de desarrollo, se utilizan las siguientes tecnologías:

- Plataforma Linux para ejecución.
- Python como lenguaje de programación.
- MySQL para la gestión de base de datos.
- MQTT como protocolo de comunicación entre agentes.
- Philips Hue Lights y Pushbullet como dispositivos y servicios IoT.
- Servicios de Climatología y Meteorología como interfaces de entrada.

A continuación se describen los elementos más importantes.

### 5.1.1. Lenguaje de programación Python

Dentro de este entorno definido, el lenguaje de programación utilizado para la creación de diferentes procesos es Python, que es un lenguaje de programación interpretado, fácil de leer y entender, es multiplataforma y tiene a disposición extensiones para muchas librerías. Dado que el mundo de IoT tiene su base en la utilización de servidores y procesos en la nube, Python tiene buena relación

directa con este tipo de necesidades, por lo que una buena práctica es implementar el sistema en este lenguaje. Además, existen muchos proyectos diferentes de código abierto donde se puede aprovechar la conexión de diferentes tecnologías con Python, como se verá más adelante.

Se utiliza la versión Python 2.7.12 y los procesos y funciones se crean a través de scripts, que son ejecutados a través de la consola de Ubuntu. De este modo, a través de la consola se pueden obtener y visualizar datos relevantes mientras se ejecuta el código en tiempo real.

### 5.1.2. Base de datos MySQL

Una base de datos es un conjunto de información agrupada por diferentes tópicos o temática que comparten datos relativos entre los elementos que la forman. Permite clasificar información, ordenarla y acceder a ella de una manera rápida y sencilla. En este proyecto se utiliza un Sistema Gestor de Base de Datos gestionado por SQL (*Structure Query Language*). Se trata de un lenguaje declarativo que ayuda al manejo de datos y a efectuar consultas para acceder, recuperar y modificar la información relacional que aparece en la base de datos, así como realizar cálculos básicos con estos datos. En concreto, se utiliza MySQL, que funciona de manera rápida y es de código abierto. Además, la integración con Python es relativamente sencilla.

La versión con la que se trabaja es mysql-server 5.7.18-0ubuntu0.16.04.1.

Para integrarlo en Python, se descarga la librería *python-mysqldb*, y se importa *MySQLdb* en cada script o proceso que esté en contacto con los datos, tanto para añadir, editar o eliminar [15]. Una explicación detallada de esta integración se puede ver en el ANEXO 1.

### 5.1.3. Comunicación interna MQTT: Mosquitto

Como se ha analizado previamente en el Estado el Arte, un buen protocolo analizado que funciona sobre el concepto de un bróker central es MQTT. Entre los diferentes servicios que utilizan este protocolo, se necesita de uno que consuma pocos recursos, que sea rápido en la entrega y que acepte multitud de clientes. Un servicio que cumple con estos requisitos es Mosquitto.

Para integrarlo en Python, se utiliza la librería “paho-mqtt” [16].

En este proyecto, dado que la comunicación interna entre procesos tiene una importancia sustancial, una explicación más detallada del funcionamiento y la utilización de este servicio se presenta en el ANEXO 2.

### 5.1.4. Interfaz de entrada/salida: Philips Hue Lights

Para alertas de uso local, una de las tecnologías que se ha decidido integrar es el set de *Philips Hue Lights*. Como se ha comentado en el apartado 3.2.2.1.1., la

bombilla (o conjunto de bombillas) está conectada mediante el protocolo ZigBee con el *Hue Bridge*, que hace de intermediario. Éste dispone de un portal interno, su API, que no es más que un panel de control *web-based* que se conecta con Internet. Desde este portal se permite controlar los ajustes de las luces en el sistema. Esta API necesita acceso directo al bridge, por la cual cosa es necesario estar en la misma red local.

La API es una interfaz que funciona mediante comandos REST a través de HTTP dentro de la misma red local. A cada dispositivo lumínico se le asigna una URL dentro de esa red para controlar sus parámetros.

Los parámetros a modificar son varios. La notación para realizar un "PUT" a la URL de la bombilla está en formato JSON. Los parámetros más utilizados dentro de este proyecto son para apagar o encender la bombilla, para cambiar su color (siguiendo el sistema HSV) o incluso para configurar un efecto de parpadeo.

La configuración del bridge, las URLs, el funcionamiento general y la creación de usuarios para el *bridge* se detallan en el ANEXO 3.

### 5.1.5. Interfaz de entrada/salida: Pushbullet

Como servicio de mensajería se ha optado por *Pushbullet*, que puede gestionar información del usuario sincronizando sus dispositivos mediante una cuenta de Gmail.

Se ha utilizado la librería "pushbullet.py", que permite conectarse a la API de Pushbullet [17]. Para ello, se necesitan las credenciales del usuario, un *token* personal de la cuenta utilizada que puede conseguirse fácilmente a través de la página web oficial de Pushbullet. De esta manera, se importa el paquete *pushbullet* donde está definida la clase y se crea una instancia de la clase Pushbullet, pasándole como parámetro el *token* del usuario.

Con esta librería se pueden enviar mensajes entre los dispositivos integrados en la cuenta de manera sencilla a través de la función PUSH_NOTE. Por otro lado, para poder recibir diferentes notificaciones y tratar Pushbullet como interfaz de entrada, se ha utilizado un pequeño proyecto de código abierto que permite esta conexión. Aunque el proyecto incluye multitud de funciones, para este proyecto sólo son necesarias algunas de ellas, entre las que está GET_PUSHES, que devuelve la lista de notas o mensajes que el usuario dispone en su cuenta [18].

### 5.1.6. Interfaces de salida: Servicios de Meterología y Climatología

Algunas interfaces de entrada que se han implementado en el sistema para obtener información real acerca de diferentes servicios de meteorología y climatología son las siguientes:

- Para capturar el porcentaje de humedad se ha utilizado la API de "OpenWeatherMap" [19].

- Para capturar el nivel de polen se ha utilizado un servicio web que proporciona la UAB [20].
- Para capturar la Radiación ultravioleta se ha utilizado un servicio web que proporciona "WeatherOnline" [21].

## 5.2. Planificación del desarrollo

Haciendo referencia al desarrollo de todo el trabajo realizado se presenta un seguimiento del proyecto. No se presenta una planificación inicial, solo un diagrama de seguimiento ya que como había una parte de evaluación y de definir el marco del proyecto, no se definieron tareas iniciales sino que se siguió un modelo iterativo de desarrollo. A continuación se describe de forma general este seguimiento del proyecto.

La primera toma de contacto con el director del proyecto fue a principios de Diciembre de 2016, donde se propusieron unas ideas principales y se empezó a definir el entorno de trabajo.

El desarrollo del proyecto comenzó a mediados de Enero, donde se trabajó un mes entero realizando una búsqueda exhaustiva de diferentes tecnologías y frameworks IoT. Debido a la escasa relación que existía entre los resultados obtenidos y lo que se quería encontrar, se volvió a realizar otro análisis cambiando el modo de búsqueda por entornos que se habían definido.

A mediados de Febrero se empezó a analizar la persistencia del sistema. Para ello, se decidió trabajar en Java y se hizo la integracón con MySQL, creando un *mockup* sencillo con alguna condición lógica introducida manualmente en el sistema.

A principios de Marzo se volvió a iterar en el análisis de tecnologías, esta vez para el modelo de comunicación interna, donde se estuvo revisando el bróker de comunicaciones y los protocolos y servicios que se implementaban en dicho modelo de comunicación durante dos semanas. Después de realizar varias pruebas con Mosquitto, se decidió cambiar el lenguaje de programación a Python para su integración ya que resultaba más sencillo. Por este motivo, se buscaron librerías para integrar también con MySQL.

A finales de Marzo y a principios de Abril se volvió a iterar en la búsqueda de tecnologías pero esta vez de manera más concreta, buscando tecnologías de entrada y salida de datos, dispositivos IoT y servicios. Se descubrió Pushbullet, que al principio se analizó para tratarlo como evento de entrada al sistema ya que muestra notificaciones entre dispositivos, incluido de las redes sociales y correos electrónicos, entre otros, y se decidió integrar en Python, por lo que hubo una semana de pruebas e interacción.

Teniendo ya un servicio de entrada se empezó a realizar un primer diseño lógico del sistema, donde aparecían elementos como la entrada de datos, la persistencia y la inteligencia del sistema.

A mediados de Abril se hizo de nuevo una revisión del sistema. Se añadieron elementos, como la localización del usuario, y se decidió utilizar Pushbullet como Servicio de Mensajería para alertar al usuario. Para evento de entrada, se analizaron pulseras cuantificadoras y se decidió aplicar el Caso de Uso 1, que se puede ver en el apartado 2.4.2.1. Dado que no se disponía de la pulsera, se procedió a su simulación.

A finales de Abril se generalizó el sistema para que fuese adaptable a cualquier evento de entrada y de salida. Además, volviendo a revisar el sistema, se modificó la lógica de la localización, donde se agregaron dos nuevos casos que, aunque no estén implementados, están contemplados en el sistema. Uno eran los sensores y el otro la tecnología PULL del servidor a nuestro sistema.

A principios de Mayo se volvieron a analizar tecnologías tanto de entradas como de salida. Se encontró Philips Hue, que se integró al sistema como acción de salida. Además se volvió a iterar en el diseño del sistema y se propusieron nuevas Reglas de Negocio que estaban distribuidas en dos componentes diferentes.

A mediados de Mayo se trabajó un mes dedicado exclusivamente al desarrollo de la documentación, arreglando algunas piezas de código que se necesitaba para el desarrollo de la maqueta de prueba.

Por último, se retocaron los últimos detalles y se añadieron nuevas reglas de negocio para nuevas interfaces de entrada, creando una parte del Caso de Uso 3 y añadiendo algunas mejoras. Además, se modificó una pequeña parte importante del diseño para conseguir un sistema consistente.

Como se ha visto, la planificación ha seguido un modelo iterativo donde se iba revisando, haciendo cambios en el diseño y agregando y eliminando algunos elementos para conseguir el sistema final.

## 5.3. Descripción técnica

Para analizar este apartado, primero se analiza la estructura técnica del sistema, donde se analizan de forma general los componentes que lo generan. A continuación se entra más en detalle en cada componente, analizando los procesos unitarios. Una vez analizado el sistema entero, se presenta una maqueta donde se evalúan los resultados obtenidos.

### 5.3.1. Arquitectura física de componentes

A gran escala, para clarificar la descripción técnica, se puede dividir el sistema en 4 grandes componentes que funcionan por separado, teniendo en cuenta la relación directa que existe entre ellos mediante el bróker de comunicaciones central, encargado de distribuir la información entre ellos.

La siguiente figura muestra de forma visual el marco de trabajo, dividido en estos 4 grandes bloques, que posteriormente se analizarán más en detalle.

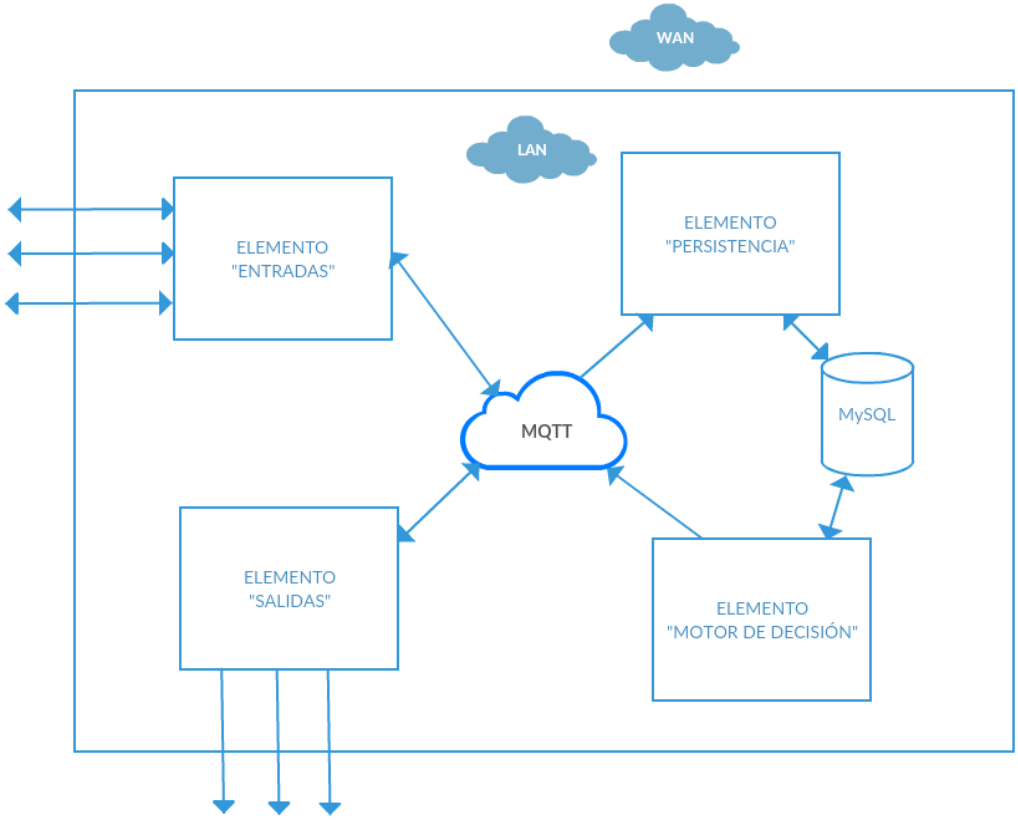



*Figura 24. Estructura global del sistema*

- Elemento Entradas.
- Elemento Persistencia.
- Elemento Motor de Decisión.
- Elemento Salidas.

Gracias al bróker de comunicaciones, cada elemento puede funcionar por separado y trabajar de forma asíncrona entre procesos. Esto permite el flujo de información más dinámica, ágil y eficaz. Además, permite que haya varias implementaciones de estos componentes y que estén distribuidas, que es la ventaja que proporciona.

Cada uno de estos elementos está formado por uno o varios procesos que contienen varios componentes. Por ejemplo, el elemento “Entradas” tendrá tantos procesos como tipos de evento de entrada estén configurados en el sistema. Mapeando el esquema anterior a uno más detallado se obtiene la arquitectura física del sistema, que es la implementación física del diagrama lógico que se puede ver en la figura 20.

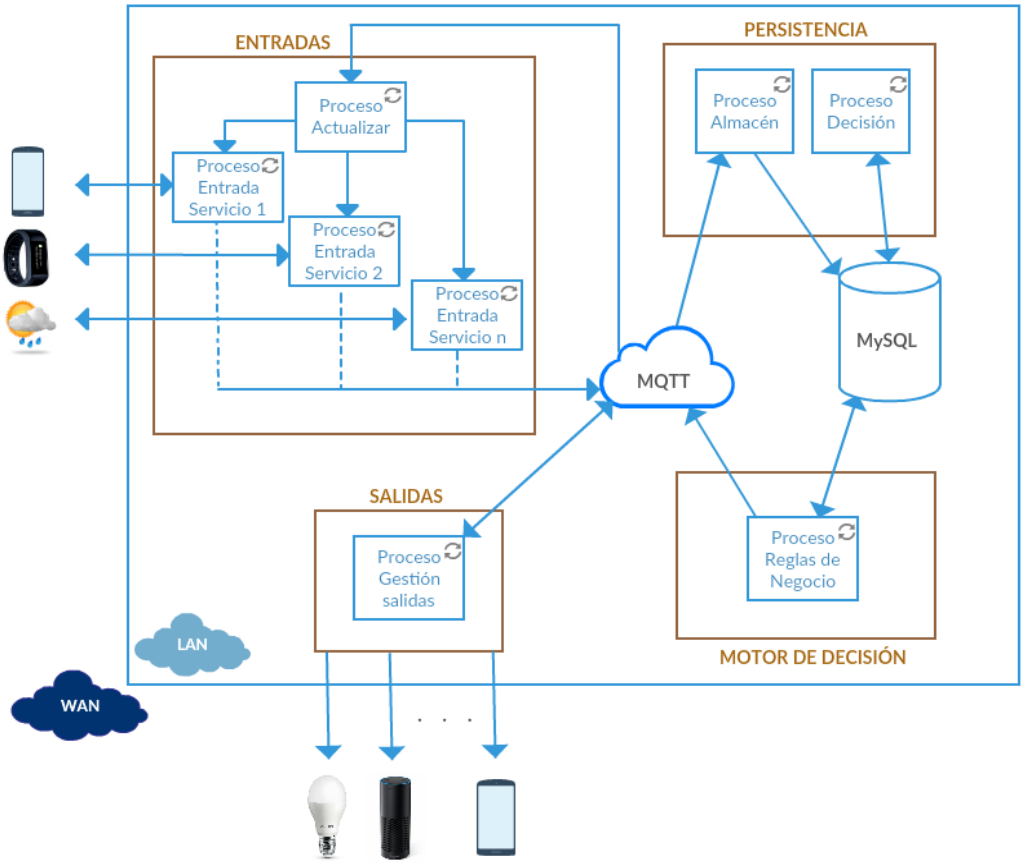



*Figura 25. Arquitectura física del sistema*

A continuación se va a analizar en un desarrollo *top-down* la descripción de cómo se han implementado los diferentes elementos que se presentan en el diagrama que muestra la arquitectura física, la figura 25, presentando dentro de cada componente algunas de las funciones más importantes.

#### 5.3.1.1. Elemento ENTRADAS

Este elemento es el encargado de conectar con los diferentes servicios y dispositivos del exterior que forman parte del entorno del usuario. Por tanto, es el responsable de traducir la información que proviene de estos servicios y dispositivos para introducirla en el sistema. Esta comunicación con el exterior traspasa la frontera entre el área WAN, donde habitan la mayoría de dispositivos IoT y servicios, y el área LAN propia del sistema.

Dentro de este elemento aparecen dos componentes principales. El primero de ellos es el "Componente Servicio de Entrada". Dentro de este componente aparecen tantos procesos como tipos de entrada haya, ya que cada una de ellas puede tener varias fuentes de información que proporcionan datos sobre el mismo evento. Por ejemplo, para la localización del usuario puede haber más de un método para recoger esta información, ya sea a través de sensores o de un servicio de geolocalización GPS a través del dispositivo móvil. Del mismo modo,

para recolectar información acerca de la concentración de polen o humedad, puede haber más de un servicio integrado para tener más fiabilidad.

Para clarificar y mejorar la legibilidad por parte del lector, de ahora en adelante, se entiende como “categoría” el conjunto de servicios y/o dispositivos de entrada que proporcionan la misma información acerca de un evento.

El segundo componente importante dentro de este elemento es el “Componente Actualización”. Este componente es un proceso que se encarga de realizar las acciones pertinentes para intentar recuperar nueva información de las diferentes entradas cuando se le solicita.

#### 5.3.1.1.1. Proceso Entrada Servicio

Como se ha comentado anteriormente, cada tipo de entrada al sistema aporta unos datos diferentes, por la cual cosa es necesario un proceso de conversión distinto, aunque todos ellos siguen una misma estructura. Estos procesos se encargan de empaquetar y encapsular la información proveniente de los diferentes dispositivos y servicios y enviarla al bróker de comunicaciones con un tópico específico para cada evento de entrada.

La estructura que sigue cada proceso es la siguiente, donde se ejecutan de manera secuencial una serie de funciones, entre las que están:

- Función ACTUALIZAR_EVENTO
- Función RECOGER_EVENTO
- Función PREPARAR_MENSAJE_EVENTO
- Función ENVIAR_MENSAJE_MQTT

##### Función ACTUALIZAR_EVENTO

Esta función se ejecuta periódicamente de manera automática cada cierto tiempo, según la periodicidad establecida para recoger información de cada categoría, donde se procede a recoger datos de todos los servicios integrados que proporcionan la misma información.

Una de las limitaciones de este componente es que el tiempo de periodicidad para cada categoría no es configurable por parte del usuario, y está definido estáticamente.

Esta función utiliza las siguientes tres funciones, ejecutadas de manera secuencial.

##### Función RECOGER_EVENTO

Cada tipo de entrada tiene unos servicios y/o unos dispositivos integrados que son accesibles o bien a través de la API o bien a través de un servidor. Esta función, distinta para cada caso, se encarga de comunicarse con ellos para extraer la información disponible en ese momento. Por ejemplo, para el caso de

la Localización serán unas coordenadas GPS y para el caso de la humedad será un porcentaje.

#### Función PREPARAR_MENSAJE_EVENTO

Una vez se consigue el evento en sí, hay que preparar una estructura de mensaje para que sea interpretable en el receptor. El envío de mensajes a través del bróker se ha realizado mediante mensajes JSON, siguiendo el concepto de clave-valor, para que se pueda enviar mucha información en un solo mensaje y que sea fácil de descifrar en el destino.

Entre las claves más comunes están el estado, el tipo de servicio desde donde proviene esa información, y el propio evento en sí. Aun así, dependiendo la categoría, cada mensaje puede ser diferente dependiendo el evento de entrada que se analice. Hay que tener en cuenta que desde este elemento siempre se envían mensajes con estado “open” para que el elemento Persistencia pueda tratarlos correctamente.

#### Función ENVIAR_MENSAJE_MQTT

Esta función recibe como parámetros el tópico al cual se desea enviar el mensaje y el mensaje en sí que se desea enviar, que proporciona la función anterior. Dentro de esta función se instancia un cliente publicador de MQTT y se crea una conexión con el bróker, donde envía el mensaje con el tópico definido.

Una explicación más detallada de esta función aparece en el ANEXO 2, ya que tiene mucha relevancia en el proyecto porque se usa constantemente para enviar datos entre procesos internos del sistema. Además, incluye también una explicación general del funcionamiento necesario para aquellos procesos que están suscritos a MQTT.

### 5.3.1.1.2. Proceso Actualización

Además de existir una conexión periódica con las interfaces de entrada, también es necesario que el sistema sea capaz de pedir nueva información cuando los datos almacenados no estén actualizados o no sean fiables.

Por este motivo, este proceso está conectado a MQTT como cliente suscriptor, y está suscrito a todos los tópicos que se utilizan para las peticiones de actualización. Dependiendo del tópico con el que llegue el mensaje, se ejecutará la función ACTUALIZAR_EVENTO para la categoría correspondiente, de tal manera que realizará una nueva búsqueda de datos entre los diferentes servicios y/o dispositivos integrados para introducir en el sistema la nueva información que se ha solicitado, siguiendo los pasos comentados en el apartado anterior.

## 5.3.1.2. Elemento PERSISTENCIA

Este elemento es el encargado de gestionar, en mayor parte, la base de datos. Su función principal es introducir y actualizar eventos en la base de datos.

Además, de todos los eventos que llegan de entrada, tiene que ser capaz de decidir para cada categoría cuál es el valor adecuado en ese momento, aplicando una lógica razonable.

Dentro de este elemento aparecen dos componentes principales. En primer lugar el “Componente Almacén”, que es un proceso que está conectado a MQTT como cliente suscriptor y está suscrito a tantos tópicos como categorías de entrada hay. A parte, también está suscrito a los tópicos con los que publica el “Elemento Salidas”, ya que se encarga de actualizar valores en la base de datos a través de la retroalimentación que recibe de él. En segundo lugar, el “Componente Decisión”, que es el encargado de aplicar la lógica razonable para decidir el valor correcto del evento en ese instante de tiempo, teniendo en cuenta toda la información recogida por parte del componente anterior.

#### 5.3.1.2.1. Proceso Almacén

Este proceso está suscrito a varios tópicos a través de MQTT. Cada mensaje que recibe tiene que tratarlo de una manera u otra. Como el mensaje llega codificado en JSON, primero de todo se deserializa y se mira qué valor tiene para la clave “Status”. Si es “open”, significa que se debe insertar por la cual cosa se utiliza la función INSERTAR_DATOS_BDD. Si resulta que el estado no es “open”, se debe actualizar un evento existente, por la cual cosa se utiliza la función ACTUALIZAR_DATOS_BDD.

Todos los eventos que recibe con estado “open” los inserta en unas tablas específicas que guardarán esos valores de manera temporal, desde donde el “Proceso Decisión” decidirá el valor correcto.

Para algunos casos concretos, como por ejemplo la Localización, simplemente se utiliza la función de insertar, ya que no es necesario actualizar el evento ya que no se trata como alerta, sino de manera informativa para tomar una decisión.

##### Función INSERTAR_DATOS_BDD

Esta función recibe dos argumentos. El primero de ellos es la tabla a la cual se van a insertar los datos. El segundo, los datos a insertar dentro de esa tabla.

Para ello se crea una conexión con la base de datos a través de MySQLdb. Una explicación más detallada de la integración de MySQLdb con Python se puede ver en el ANEXO 1.

##### Función ACTUALIZAR_DATOS_BDD

Como en la función anterior, primero se crea una conexión con la base de datos. Esta función es aún más sencilla, ya que simplemente cambia el estado de la tabla que recibe con el id correspondiente.

#### 5.3.1.2.2. Proceso Decisión

Por otro lado, como se ha comentado previamente, para cada categoría puede aparecer información proveniente de diferentes servicios. Este proceso sirve para decidir qué valor se considera correcto en ese preciso instante de tiempo. Para ello, este proceso se ejecuta periódicamente cada cierto tiempo, donde se llama a la función COMPROBAR_TABLAS_DECISION.

##### Función COMPROBAR_TABLAS_DECISION

Esta función itera sobre todas las tablas de decisión de la base de datos. Si encuentra alguna tabla con algún elemento, se utiliza la técnica de *threading* para ejecutar en paralelo la función DECISION_SOBRE_CATEGORIA después de un tiempo definido. Se espera este tiempo ya que algunos eventos de entrada para la misma categoría pueden tardar más en llegar ya sea por la conexión que se realiza en la entrada o por algún *delay* del bróker.

Una de las limitaciones que presenta este proceso es que tanto el tiempo de ejecución del proceso en sí como el tiempo de espera definido para esta función están marcados de manera estática.

##### Función DECISION_SOBRE_CATEGORIA

Esta función vuelve a conectarse con la base de datos para la tabla correspondiente, ya que después de varios segundos de espera puede haber habido nuevas inserciones en la tabla que se está tratando.

Esta función es diferente para cada categoría, ya que se aplica una lógica distinta. Aun así, para la elección del valor que se considera como “correcto” en ese momento, se itera sobre todos los elementos disponibles en ese momento y según unos criterios definidos para cada caso se procede a una elección. Entre los parámetros de elección más comunes están el tipo de servicio (ya que puede tener más o menos fiabilidad) y la ventana de actualización, entre otros.

Una vez se decide el valor, se utiliza la función INSERTAR_DATOS_BDD para introducir en la tabla correspondiente el valor final de esa categoría, el cual el “Elemento Reglas de Negocio” tratará para decidir si es necesario disparar una alerta o no.

Por último, una vez se ha tomado una decisión y se ha introducido en la base de datos “original”, es decir, la que no es temporal, se procede a eliminar todos los elementos en la tabla de decisión correspondiente, llamando a la función ELIMINAR_ELEMENTOS_CATEGORIA_DECISION, pasándole como parámetro la tabla de decisión correspondiente, para que no vuelva a iterar sobre los mismos elementos.

##### Función ELIMINAR_ELEMENTOS_CATEGORIA_DECISION

Esta función simplemente se conecta con la base de datos y elimina todos los elementos que haya disponibles para la tabla que le llega como argumento.

#### 5.3.1.3. Elemento MOTOR DE DECISIÓN

Este elemento se encarga de aplicar la lógica más importante del sistema. Va revisando continuamente la base de datos y en caso de detectar un evento nuevo lo procesa y decide si el sistema debe actuar para alertar al usuario o no.

El elemento está formado por un proceso principal, “Proceso Reglas de Negocio”, que se va ejecutando iterativamente sobre cada una de las tablas (todas aquellas que no son de decisión) de la base de datos para revisar los eventos que hay disponibles y el estado en que se encuentran.

##### 5.3.1.3.1. Proceso Reglas de Negocio

Este proceso se ejecuta iterativamente cada cierto tiempo, que está definido de manera estática, no configurable. Es el encargado de ir revisando todas las tablas de la base de datos continuamente. Cada tupla, es decir, cada evento, tiene varios campos según la categoría que sea. Los más comunes son el valor o nivel del evento (por ejemplo, si es humedad, el porcentaje), el estado (que puede ser abierto, pendiente, procesando o cerrado) y la fecha de registro del evento. Estos campos son claves a la hora de analizar los diferentes eventos.

Dentro de este proceso se ejecutan varias funciones, que siguen un orden lógico y que se explican a continuación.

###### Función COMPROBAR_ESTADO

Esta función se conecta con la base de datos, la inspecciona y recoge, para cada tabla, los eventos que tienen como estado “open” y la fecha del evento está comprendida en un intervalo definido que recibe como argumento la función. Este intervalo está definido estáticamente y cada intervalo es una ventana de tiempo donde se tiene en cuenta la hora actual.

Se realiza de esta manera para asegurarse que no va a quedarse ningún evento en estado “open”, ya que siempre se va a poder recoger a la siguiente iteración siempre y cuando el evento entre.

Si resulta que encuentra algún evento que cumple con estos requisitos, actualiza el estado del evento a pendiente, para que sea tratado correctamente en otra función. Además, añade la fecha actual al campo de la tabla “fecha_pendiente”, ya dispone de un tiempo máximo en este estado, como se verá más adelante.

###### Función REVISAR_EVENTOS_PENDIENTES

Esta función se encarga de revisar por separado todos los eventos que están pendientes de tratar. Aunque cada evento que esté pendiente se gestiona a través de otra función, GESTIONAR_EVENTOS_PENDIENTES, esta función se encarga de iterar sobre todos los eventos que tienen estado “pending” y modificar, en caso de que sea necesario, el estado actual del evento una vez se retorne el estado de la función GESTIONAR_EVENTOS_PENDIENTES, que puede ser “processing”, si se está procesando la alerta al usuario, o “closed”, en

caso de que se decida que no hace falta alertar. En caso de que sea "processing", añade la fecha actual al campo "fecha_procesado", ya que también dispone de un tiempo máximo en este estado.

### Función GESTIONAR_EVENTOS_PENDIENTES

Esta función es la encargada de decidir si se debe lanzar una alerta al usuario o no. Tanto si se dispara como si no, debe devolver el estado en el que se encuentra el evento ya que una vez tratado no puede quedarse pendiente, a no ser que las funciones intermedias soliciten actualizaciones, que entonces se volverá a revisar en la siguiente iteración.

Por lo tanto, para cada categoría de evento se dispone de unas funciones específicas para gestionar ese evento en concreto. Por ejemplo, para el nivel de azúcar se harán unas comprobaciones para determinar, si es necesario, qué grado de alerta disparar dependiendo el nivel de glucosa del usuario.

Otras reglas de negocio pueden relacionar diferentes tipos de eventos, como por ejemplo el polen y la humedad, que pueden ir relacionados.

En el caso de que se decida que se debe alertar al usuario, se procede a comprobar su localización. Para ello se utiliza la función COMPROBAR_LOCALIZACION, que se explica más adelante. El siguiente paso es preparar el mensaje de alerta en un JSON concreto y enviarlo a través de la función ENVIAR_MENSAJE_MQTT al tópico correspondiente, siempre y cuando la función COMPROBAR_LOCALIZACION haya devuelto un valor (hay que tener en cuenta que si la información no está actualizada se pedirá una actualización de manera asíncrona, la cual cosa no se podrá resolver ese evento en esa iteración). Seguidamente, devuelve el nuevo estado para ese evento, "processing", para que la función REVISAR_EVENTOS_PENDIENTES lo actualice.

Por otro lado, si el sistema decide que no es necesario alertar al usuario, se devuelve como nuevo estado "closed" para cerrar el evento.

### Función COMPROBAR_LOCALIZACIÓN

Esta función localiza al usuario. Para ello, primero de todo se conecta a la base de datos y revisa el último valor introducido. Si la fecha de ese evento es menor a un tiempo definido, se considera actualizada la localización del usuario. En este caso, se recogen los valores de latitud y longitud y se revisa si forman parte del rango establecido como "vivienda local" del usuario.

Si por el contrario la fecha de ese evento no está actualizada, se utiliza la función ENVIAR_MENSAJE_MQTT al tópico concreto de actualización, al cual está escuchando un proceso en el elemento Entradas.

El valor que devuelve esta función es simplemente un Booleano que indica si el usuario está en su casa o no.

Otra de las limitaciones en este caso es que tanto la localización del usuario como el rango de error permitido están introducidos de manera estática, y por el momento no son programables.

#### Función CANCELAR_EVENTOS_LIMITE_EJECUCION

Esta función revisa todos los eventos que están en estado procesando y los elimina si supera un tiempo delimitado.

#### Función CANCELAR_EVENTOS_LIMITE_PENDIENTE

Igual que la función anterior, elimina los eventos que están pendientes y superan un límite establecido.

Estos límites son importantes ya que pueden congestionar el sistema y cargar

#### Función ELIMINAR_EVENTOS_EXPIRADOS

Esta función revisa todos los eventos que están en estado cerrado y los elimina si supera un cierto tiempo, como por ejemplo, 2 días, ya que no aportan información al sistema y se necesita liberar espacio.

### 5.3.1.4. Elemento SALIDAS

Este elemento es el encargado de recibir las alertas provenientes de las Reglas de Negocio y actuar con las interfaces de salida según los parámetros recibidos, entre los que están el tipo de evento y la localización del usuario. También realiza algunas comprobaciones teniendo en cuenta algunas variables, como la hora de ejecución de la alerta. Por último, hay que destacar que este elemento se encarga de retroalimentar el sistema para confirmar que se ha proporcionado una salida al evento recibido.

El elemento está formado por un componente principal, el “Proceso Gestión Salidas”.

#### 5.3.1.4.1. Proceso Gestión Salidas

Este proceso está conectado a MQTT como cliente suscriptor a todos aquellos tópicos con los que envía información los componentes de las Reglas de Negocio. Dependiendo el tópico con el que llegue el mensaje, se ejecuta una función u otra, ya que cada categoría de evento tiene definido unas alertas dependiendo de unas variables. Más adelante se presenta la función de manera abstracta.

El mensaje que recibe está encapsulado en JSON, por lo tanto se deserializa y se comprueban las variables. Las más comunes son la gravedad de la alerta y la localización del usuario. Además de éstas, en este proceso se comprueba otra variable, la hora de disparo de la alerta, ya que han establecido unas “horas de sueño” del usuario de manera estática.

### Función TOMAR_DECISION_SALIDA

Esta función recibe un mensaje de tipo JSON, por lo cual lo primero que realiza es su deserialización, donde se obtiene datos del evento de alerta: la gravedad del evento, la localización del usuario y el identificador del evento de la tabla correspondiente, necesario para realizar el *feedback*.

Cada categoría tiene definida una función concreta, donde se evalúan una serie de condiciones según los parámetros que le llegan. Además, comprueba si la alerta se encuentra en horas de sueño del usuario o no, ya que también es un factor determinante para decidir una de las salidas posibles.

### Función COMPROBAR_HORA_ALERTA

Esta función devuelve un Booleano indicando si el momento de disparo de la alerta está comprendida entre las horas de sueño del usuario o no, ya que esta información es importante para decidir el tipo de salida, en caso de que haya que realizarla.

### Función ALERTAR_VIA_PHILIPS

Esta función enciende o apaga la bombilla Philips mediante la conexión con su API. Recibe dos parámetros. El primero es el color de la luz que se desea iluminar. El segundo, si se debe realizar efecto parpadeo o no.

Para configurar el estado de la bombilla se procede a realizar la llamada correspondiente a la URL específica. Los detalles de configuración y obtención de claves de usuario se pueden ver en el ANEXO 3.

### Función ALERTAR_VIA_PUSHBULLET

Para la integración de Pushbullet, como se ha comentado previamente, se ha utilizado la librería “pushbullet.py”.

Esta función se utiliza para enviar un mensaje a través de la aplicación Pushbullet. La función recibe dos parámetros: el título y el cuerpo del mensaje. El mensaje es enviado a todos los dispositivos asociados a la cuenta del usuario.

### Función RETROALIMENTAR_SISTEMA

Por último, una vez se ha enviado la alerta a las diferentes interfaces de salida, se realiza una retroalimentación al sistema. Se encapsula en un JSON el identificador que corresponde ese evento, deserializado previamente, y el nuevo estado, que es “closed” ya que se ha realizado la alerta correctamente. Esta información se envía como mensaje mediante la función ENVIAR_MENSAJE_MQTT, con el tópico correspondiente. El “Proceso Almacén” es el receptor de estos mensajes, que se encarga de actualizar la base de datos con esta información.

### 5.3.2. Maqueta de prueba

El código fuente del proyecto se puede descargar del siguiente repositorio de Github:

*http://github.com/APlanells/alert-management-system*

En el README de este repositorio se pueden visualizar los prerrequisitos necesarios, así como la instalación y la configuración de los diferentes servicios integrados. Dado que el sistema creado no es trivial, y tiene integrados varios servicios diferentes, para ponerlo en marcha es necesario seguir una serie de pautas para poder ejecutar el sistema.

Los prerrequisitos necesarios para este sistema se pueden ver en la sección "Prerequisites" del README.

Dentro del abanico de posibilidades que hay tanto para las interfaces de entrada como de salida, sólo se han implementado algunas y se han simulado otras para cubrir los máximos escenarios posibles, dentro de las reglas de negocio implementadas.

Entre las interfaces reales de entrada están:

- Nivel de concentración de polen, que se obtiene a través de un servidor web.
- Porcentaje de humedad, que se obtiene a través de "OpenWeatherMap", conectándose a la API.
- Índice de radiación UV, que se obtiene a través de un servidor web.

Para las interfaces reales de salida aparecen:

- Luces LED de Philips Hue Lights.
- Servicio de Mensajería Pushbullet.

Para las interfaces reales tanto de entrada como de salida, primero es necesario conseguir el API KEY de todos aquellos servicios que lo requieren. La información detallada para cada tipo de servicio puede verse en el apartado "Installation and Configuration" del README de Github.

Como se ha comentado previamente, para cubrir más posibilidades se ha procedido a la simulación de algunos servicios de entrada. Esta simulación se ha realizado a través de un servidor externo para simular que estos datos forman parte del exterior del sistema y no de manera local. Se necesita tener un servidor que soporte PHP y que disponga de una base de datos SQL, ya que la lógica del servidor se ha programado a través de unos scripts PHP con unas tablas definidas en SQL. La información necesaria, es decir, los archivos o scripts SQL y PHP para cargar en el servidor están disponibles en el repositorio de Github, y los ajustes necesarios están documentados en el README.

Para este proyecto se ha utilizado como servidor externo Cloud Hosting 000webhost [22]. Entre los simuladores creados aparecen:

- Pulsera cuantificadora que proporciona el nivel de azúcar del usuario.
- Servicio de localización GPS.
- Sensor de presencia en el domicilio.
- Porcentaje de humedad.
- Índice UV.

Además de utilizar estos simuladores de entrada a través de un servidor externo, también se ha procedido a simular el funcionamiento de un asistente virtual como salida del sistema mediante el uso de una librería que traduce el texto en habla. Concretamente, se ha utilizado "pyttsx" [23]. Aunque funcione de manera local, simula el comportamiento de un altavoz inteligente que ayuda a probar el sistema en diferentes situaciones, aumentando así el número de salidas disponibles.

Una vez está todo configurado e instalado, para proceder a probar el sistema, se ha seguir un orden para poner en marcha los diferentes procesos ejecutables en el sistema. Este orden está definido en la sección "Execution Order" del README, en el repositorio del proyecto.

### Ejemplo de prueba

Para probar el sistema y comprobar que los procesos funcionan de manera correcta, se procede a realizar un ejemplo basado en el Caso de Uso 1, suponiendo que el usuario tiene una pulsera cuantificadora que determina el nivel de azúcar.

Se supone que el usuario está fuera de casa y el nivel de glucosa supera el límite establecido de 120 mg/dl. En este caso, el usuario debería recibir una notificación en su teléfono móvil a través de Pushbullet que le informe acerca de su situación actual, recomendando al usuario que ingiera insulina dado que los niveles son elevados.

Para poner en marcha el sistema y ejecutar este caso específico, primero se ha de crear una cuenta en Pushbullet y modificar algún script del proyecto para configurar la cuenta personal. En el apartado "Pushbullet Service", dentro de la sección "Installation and Configuration" del README está detallada la modificación necesaria. En segundo lugar, hay que configurar el simulador tanto para el nivel de azúcar como para la localización del usuario. Los detalles necesarios para llevar a cabo esta actividad están explicados en el apartado "External Server", dentro de la misma sección anterior. Una vez se tiene creado el servidor, se han cargado los archivos PHP y SQL necesarios y se dispone del dominio (MY_DOMAIN), se realizan las siguientes llamadas al servidor para ajustar los valores deseados.

- MY_DOMAIN/set_sugar.php?level=140
- MY_DOMAIN/set_location_serviceGPS.php?home=0
- MY_DOMAIN/set_location_sensor.php?home=0

Por último, una vez se dispone del sistema configurado, hay que ponerlo en marcha teniendo en cuenta un orden específico, siguiendo los pasos descritos en la sección “Execution Order” del README.

Para este caso, de entre todos los procesos que recogen información de las diferentes interfaces de entrada, es suficiente con ejecutar únicamente el proceso “input_sugar.py”, que es el que proporciona información acerca el nivel de azúcar del usuario.

Realizados estos pasos, una vez se ejecuta el último script, “input_sugar.py”, el sistema empieza a procesar los datos. Dado que una vez se detecte la necesidad de alertar al usuario el sistema no dispondrá de la geolocalización de éste, pedirá una actualización de su posición en el mapa para decidir un tipo de salida u otro. Como para este ejemplo se ha simulado que el usuario se localiza fuera de su domicilio, se recibirá un mensaje a través de Pushbullet con título “SUGAR LEVEL” y cuerpo de mensaje: “Be careful! Your sugar level is high. Please inject some insuline”.

# 6. CONCLUSIONES

## 6.1. Objetivos alcanzados

El objetivo principal del proyecto era el de conseguir un sistema robusto y consistente que sea adaptable a múltiples dispositivos e interfaces, tanto de entrada como de salida, y que gestione una serie de eventos para alertar al usuario en caso de que sea necesario. A partir de aquí, la idea era integrar el máximo de tecnologías posibles para cubrir los diferentes escenarios expuestos donde puede ser útil el sistema.

En primer lugar, cabe destacar que se han conseguido la mayoría de los objetivos propuestos en cuanto al diseño e implementación del sistema, a excepción de algunos de ellos descritos en el apartado 6.2 (Trabajo futuro). Se ha conseguido un sistema robusto y consistente que no es trivial en cuanto a las decisiones de salida.

Aunque alguno de los procesos en ejecución caiga, no se pierden los datos del sistema ya que éstos se almacenan de manera persistente. Esto ha sido posible gracias al gestor de base de datos *MySQL* que se ha utilizado. Además, gracias a la indexación utilizada, la búsqueda de resultados cuando hay multitud de eventos se realiza de forma rápida.

En cuanto a la comunicación, MQTT es un protocolo bastante ágil que permite desacoplar procesos y obtener mayor eficiencia en el sistema. Dado que se basa en un sistema de alertas en tiempo real (aunque también pueden programarse para un tiempo determinado), el hecho de tratar cada elemento por separado ha sido un punto clave en el sistema, ya que no colapsa ni bloquea los recursos cuando alguna llamada falla. El funcionamiento de Mosquitto en este proyecto ha proporcionado un hilo de comunicación perfecto a través de mensajes en formato JSON. Al actuar en una red local, no se han detectado fallos entre el envío y la recepción de datos. Además, ha sido muy útil para realizar tareas de forma asíncrona, como por ejemplo las peticiones de actualización entre procesos.

Por otro lado, cabe destacar que se ha conseguido un sistema modular y adaptable a diferentes interfaces de entrada y de salida. Se han podido integrar algunos dispositivos IoT como *Philips Hue Lights* y servicios como *Pushbullet*, que funcionan correctamente a través de peticiones directas con su API. El *bridge* de Philips, tal y como está configurado, funciona correctamente si el sistema se encuentra en la misma red local. Si se quisiera utilizar este tipo de alerta en otra área, se debería configurar el *bridge* con una IP estática y asignarle un puerto de escucha determinado para realizar peticiones a esa dirección. Para privatizarlo, se podría crear un servidor de tal manera que el sistema realice una petición de tipo “POST” a este servidor pasándole varios parámetros, donde uno de ellos haría de contraseña, y si coincide, el servidor se encargaría de realizar la llamada al bridge correspondiente con los parámetros del, o de los, dispositivos lumínicos que se deseen. Por otro lado, para el caso del servicio *Pushbullet*, hay que tener en cuenta un detalle, y es que el servicio gratuito de *Pushbullet* tiene

una limitación de 500 mensajes *push* por mes. Depende que casos sería conveniente utilizar la versión Pro o mirar de cambiar el servicio de mensajería por otro.

El número de interfaces físicas de entrada y salida es relativamente limitado en la implementación final. Esto es debido a que el objetivo del proyecto era el diseño y evaluación del mismo. Por tanto, se ha optado por simular algunas de estas interfaces para ampliar diferentes posibilidades en el sistema.

Se ha conseguido un sistema flexible y consistente que ayuda a la integridad de diferentes tecnologías y nuevos casos de alerta de manera sencilla. Esto ha sido posible ya que el sistema no es trivial, funciona de manera asíncrona entre procesos y elementos del sistema y la toma de decisiones está distribuida en diferentes elementos. Gracias a estas características del sistema, se ha conseguido relacionar diferentes eventos de entrada y proporcionar una salida correspondiente teniendo en cuenta información de varios tipos de servicios integrados.

Además, gracias a la modularidad comentada previamente, el sistema es capaz de recibir datos de varios dispositivos y servicios que proporcionan la misma información y decidir cuál es la mejor opción para ese momento en concreto. Esto ha sido posible gracias a la lógica que se ha aplicado en el Elemento Persistencia.

Hay que destacar que algunas de las variables del sistema, como los intervalos de comprobación para las Reglas de Negocio o la periodicidad con la que se recogen datos de las interfaces de entrada están implementadas de manera estática, lo que supone una limitación.

Por último, el sistema presentado no incluye todas las reglas de negocio necesarias para todos los Casos de Uso expuestos previamente ya que no ha dado tiempo. Aun así, se ha conseguido integrar el Caso de Uso 1 y gran parte del Caso de Uso 3, incluyendo información acerca de la humedad y el polen, entre otras. Las alertas restantes quedan pendientes para un futuro, así como otros detalles que se comentan a continuación.

## 6.2. Trabajo futuro

Para localizar al usuario, se probó de integrar la API de Google Maps [23]. Para ello era necesario crear un *token* personal en la cuenta de Google para recuperar información de geolocalización del usuario, que se puede conseguir realizando una llamada a la API de Google con el *token* conseguido. Se encontraron algunos problemas con esto. Por un lado, esta llamada te devuelve la geolocalización del punto desde donde se realiza la petición. Por otro, no es capaz de reconocer todos los dispositivos asociados a la cuenta.

También se miró de intentar crear una aplicación móvil para hacer la llamada desde ahí y recuperar esta información en el sistema, pero no es exactamente lo que se quería implementar. Queda pendiente seguir investigando sobre el tema

de la API de *Google Maps* para conseguir la geolocalización del usuario en todo momento que se requiera. Aun así, se ha creado un servidor externo que a través de una llamada devuelve una posición como la devuelve Google, en coordenadas, latitud y longitud para testear el sistema, ya que el tiempo era justo y no se ha podido terminar esta parte.

Por lo tanto, el primer problema a tratar es el tema de la geolocalización del usuario. Se ha de estudiar bien la API de Google para conseguir esta información real. Además, el siguiente paso es integrar unos sensores de movimiento en el domicilio para recuperar esta información. Aunque todos los casos de localización del usuario ya están contemplados en el sistema, no están implementados todos por lo que debe realizarse en un futuro inmediato, ya que la localización del usuario juega un papel muy importante en este proyecto.

Otro propósito futuro es el de incorporar nuevas tecnologías de salida. Para las alertas locales, se empezó a mirar el asistente virtual de Amazon, el *Echo*, que sirve como altavoz para alertar al usuario en su domicilio. Creando una cuenta en los servicios web de Amazon está la API del *Amazon Echo* a disposición, por lo que se podría configurar para realizar diferentes acciones de salida. Por el momento, el altavoz está simulado a través de la librería “pyttsx”.

A parte de las salidas, también es susceptible de ampliar la cantidad de eventos de entrada añadiendo nuevas tecnologías. La integración no tiene que ser complicada ya que el sistema está diseñado de forma general, por la cual cosa añadiendo nuevas tablas de diferentes eventos en la base de datos y realizando pequeños cambios en algunas funciones del sistema, así como añadir las correspondientes reglas de negocio, sería suficiente.

En el proyecto hay varios casos creados que no se han podido llegar a terminar. Otro trabajo futuro es acabarlos. Por ejemplo, está implementada la interfaz de entrada para recuperar el nivel de Radiación Ultravioleta, y hay definidas una serie de Reglas para este tipo de alerta, pero está incompleto debido a la falta de tiempo. Otro caso es el de Pushbullet. Como se ha comentado en el Estado del Arte, *Pushbullet* no es únicamente un servicio de mensajería, sino que además sirve para sincronizar cuentas e informar entre dispositivos notificaciones, entre las que están las redes sociales o recibimiento de correos electrónicos, entre otras. El sistema también tiene configuradas las tablas y algunas reglas para este tipo de evento de entrada, aunque no está terminado. La librería “pushbullet.py” no incluye herramientas para captar los mensajes de la cuenta del usuario. El siguiente paso es incluir algunas de las funciones del código abierto comentado en el apartado 5.1.5 para recoger eventos de entrada de Pushbullet, siempre que el evento que se reciba no sea el mismo que el que tiene como salida este sistema. Por lo tanto, como se ha visto, aunque haya parte de estas alertas implementadas, aún no están finalizadas, por lo que hay que acabar los últimos detalles para integrarlas por completo al sistema.

Otro de los aspectos a mejorar es la cantidad de variables que se tienen en cuenta para decidir la alerta al usuario. De momento hay dos principales, la localización y la hora de disparo de la alerta, pero se pueden ampliar

dependiendo si se encuentra en horario de trabajo, si son horas de desayuno, comida o cena, si está de vacaciones o si es fin de semana, entre muchas otras.

Además de éstas, aparecen otras variables como son la gravedad de la alerta y los rangos establecidos para cada "gravedad". Por ejemplo, para los niveles de azúcar del usuario, cada usuario puede tener una regulación diferente y por lo tanto los rangos definidos pueden ser diferentes. Por ello, otra buena propuesta para el futuro es poder modificarlas de forma dinámica.

Todas estas variables ya implementadas están fijas en el sistema, de manera estática, lo que supone una limitación. Por lo tanto, otra futura implementación sería poder ajustarlas a través de una interfaz de configuración, ya que el usuario así podría decidir su "propia" configuración del sistema, según sus necesidades.

Por otro lado, hay que comentar que el sistema recoge información de cada categoría de eventos al mismo tiempo, la procesa y la guarda temporalmente hasta que se toma una decisión respecto a los diferentes datos recibidos. Una vez se toma la decisión, se eliminan los datos analizados. En un futuro podría modificarse esta parte, y no eliminar todos estos datos analizados, sino guardarlos con un tiempo de vida particular para cada servicio de entrada. De esta manera, la recepción de información no tendría por qué realizarse a la vez para cada tipo de servicio.

Por último, hay que tener en cuenta varias situaciones respecto a la salida del sistema para las luces *Philips Hue Lights*. El sistema está configura con un único dispositivo lumínico. Por lo tanto, si se tiene muchos tipos de alerta distintos, con una única bombilla puede ser confuso para el usuario. Para un futuro, una buena opción es añadir varios dispositivos lumínicos o ampliar el rango de colores de la bombilla, para poder diferenciar de manera adecuada cada alerta de salida.

Dado que la base del sistema ya está creada, al ser un sistema de alertas a través de unos eventos de entrada que pueden venir de cualquier aplicación o servicio, a partir de ahora hay que seguir construyendo sobre esta base y abarcar el máximo de escenarios posibles para conseguir un sistema completo en todos los sentidos. El siguiente paso es empezar a incluir sensores en el domicilio para recoger otro tipo de información, y más a largo plazo, actuadores, para que realicen las acciones pertinentes cuando el sistema lo solicite.

# 7. BIBLIOGRAFÍA

## 7.1. Referencias

[1] Kowalski, R and Sadri, F: *A Logic-Based Framework for Reactive Systems*, Department of Computing, Imperial College London, p.1 (2011).

[4] Rose, K, Eldridge, S and Chapin, L: The Internet of Things: An Overview, *Understanding the Issues and Challenges of a More Connected World*, Internet Society, 2015.

[9] *The Internet of Things: disrupting insurance models*, EfmaDigest, Julio 2016.

[12] Rodriguez-Dominguez, C, Benghazi, K, Noguera, M, Garrido, JL, Rodriguez, ML and Ruiz-Lopez, T: *A Communication Model to Integrate the Request-Response and the Publish-Suscribe Paradigms into Ubiquitous Systems*, Multidisciplinary Digital Publishing Institute, 2012

## 7.2. Librerías y fuentes de código

[15] Librería MySQLdb para Python:
*http://mysql-ython.sourceforge.net/MySQLdb.html*

[16] Librería paho-mqtt para Python:
*http://pypi.python.org/pypi/paho-mqtt/1.1*

[17] Librería pushbullet.py para Python:
*http://pypi.python.org/pypi/pushbullet.py*

[18] Código abierto para integrar Pushbullet con Python:
*http://github.com/randomchars/pushbullet.py*

[18] Librería pushbullet.py para Python:
*http://github.com/randomchars/pushbullet.py*

[23] Librería pyttsx para Python:
*http://pypi.python.org/pypi/pyttsx*

## 7.3. Otras fuentes

[2] Documentación oficial de IFTTT:
*http://platform.ifttt.com/docs*

[3] Documentación técnica de Microsoft Flow:
*http://flow.microsoft.com/es-es/guided-learning/learning-flow-parts/*

[6] Documentación técnica de la API de Philips Hue Lights:
*http://developers.meethue.com/documentation/how-hue-works*

[7] Documentación técnica de la API de Lightify:
*https://eu.lightify-api.org/*

[8] Documentación técnica de la API de Amazon Echo “Alexa”:
*http://developer.amazon.com/alexa*

[9] Documentación técnica de la API de Google Fit:
*https://developers.google.com/fit/rest/*

[10] Documentación técnica de la API de Pushbullet:
*http://docs.pushbullet.com/*

[11] Documentación técnica de la API de Hangouts:
*http://developers.google.com/+/hangouts/*

[13] Página oficial del proyecto MQTT:
*http://mqtt.org/*

[14] Página oficial del Proyecto Mosquitto:
*http://mosquitto.org/*

[19] Documentación técnica de la API de OpenWeatherMap:
*http://openweathermap.org/current*

[20] Servicio web informativo sobre la concentración de Polen:
*http://lap.uab.cat/aerobiologia/es/forecast/catalunya*

[21] Servicio web informativo sobre el índice UV:
*http://www.woespana.es/Espana/Barcelona/IndiceUV.htm*

[22] Página oficial de Cloud Hosting 000webhost:
*http://es.000webhost.com/*

[24] Documentación técnica de la API de Google Maps:
*http://developers.google.com/maps/documentation/geolocation/intro?hl=es-419*

# ANEXO 1

En este Anexo se presenta la integración de MySQL con Python, detallando el funcionamiento general de esta conexión.

Como se ha comentado previamente, para conectarse con la base de datos se utiliza la librería *MySQLdb*, que debe importarse siempre y cuando se use.

El funcionamiento es el siguiente. Primero de todo se necesita abrir una conexión con la base de datos. Para ello se utiliza la función "*connect()*", pasándole como parámetros el host, el usuario, la contraseña y el nombre de la base de datos.

Una vez se consigue esta conexión, el siguiente paso es crear un cursor para ejecutar las consultas, a través de la función "*execute()*".

```
import MySQLdb

DB_HOST = 'localhost'
DB_USER = 'root'
DB_PASS = 'root'
DB_NAME = 'Project'

datos = [DB_HOST, DB_USER, DB_PASS, DB_NAME]
conn = MySQLdb.connect(*datos)
cursor = conn.cursor()
```

La ventaja de esta librería es que permite realizar consultas dinámicas. La consulta debe ser un *string* en formato SQL, pero permite dejar información incompleta añadiendo "*(%s)*", ya que MySQLdb lo convertirá más adelante en un valor literal SQL. La función "*execute()*" toma como primer parámetro el string de la consulta y como segundo parámetro un array de dimensión igual al número de "(%s)" que haya en dicha consulta. MySQLdb convierte estos valores de manera ordenada, substituyendo cada ("%s)" por el valor del array correspondiente.

La consulta puede ser de dos tipos: o bien es un "SELECT" para recuperar información o un "DELETE" o "UPDATE" para modificar datos.

En el primer caso, para visualizar la información retornada se tiene que guardar la información en una variable mediante la función "*fetchall()*" que proporciona el cursor. Esta información es un array *nxm*, donde *n* es el número de tuplas que ha devuelto la consulta y *m* es el número de campos que se han seleccionado en dicha consulta.

```
query = ("SELECT * FROM Geoloc WHERE id = (%s)")
identifier = 1
cursor.execute(query, [identifier])
data = cursor.fetchall()
```

Para el segundo caso, después de ejecutar la consulta se debe actualizar la base de datos. Esto se realiza a través de la función "*commit()*" que proporciona la conexión.

```
query = ("UPDATE Geoloc SET latitud = (%s) WHERE id = (%s)")
identifier = 1
value = 23.534345
cursor.execute(query, [value, identifier])
conn.commit()
```

Por último, se cierra el cursor y la conexión con la base de datos.

```
cursor.close()
conn.close()
```

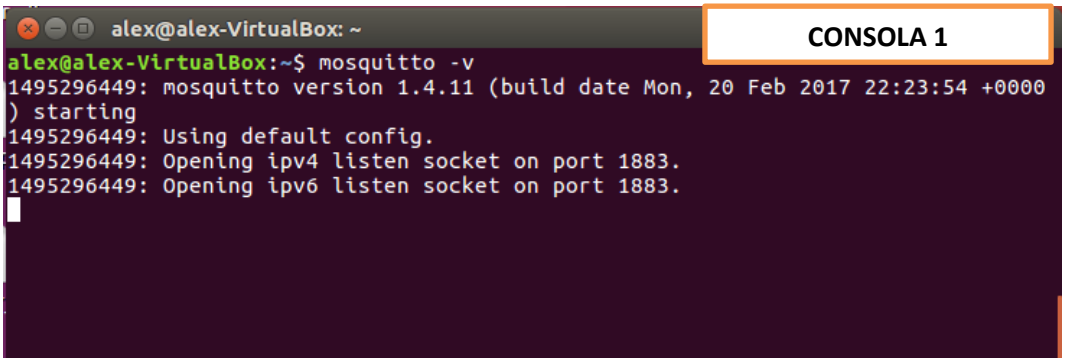
# ANEXO 2

En este Anexo se presenta en la primera parte el funcionamiento de Mosquitto, con algunos ejemplos visibles para comprender cómo funciona. Después, en la segunda parte, se analizan las funciones y el código general utilizado para integrarlo en el proyecto.

## PARTE 1

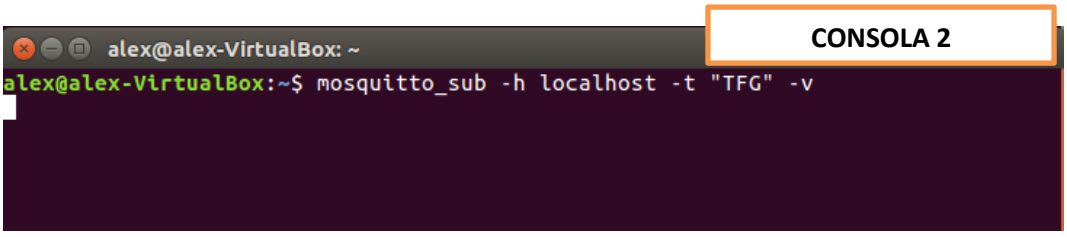
A continuación se explica el funcionamiento de Mosquitto, ya que es la base de comunicación interna entre procesos de este proyecto. Para mostrarlo de forma visual, se abre una consola, donde se puede visualizar los detalles del bróker con el comando:

*mosquitto –v*

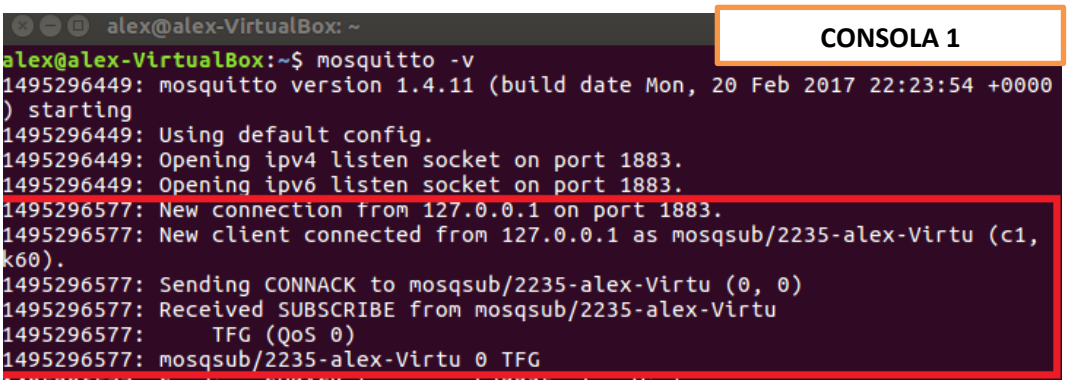



Como se observa en la Consola 1, el puerto 1883 está abierto para este servicio. Desde este momento ya se tiene el servicio en funcionamiento. Cualquier operación que se realice en él se podrá visualizarla en esa consola.

El siguiente paso es crear un cliente que se suscriba a un tópico concreto. Para ello, se abre una nueva terminal, Consola 2, y se suscribe al tópico “TFG” de la siguiente manera:

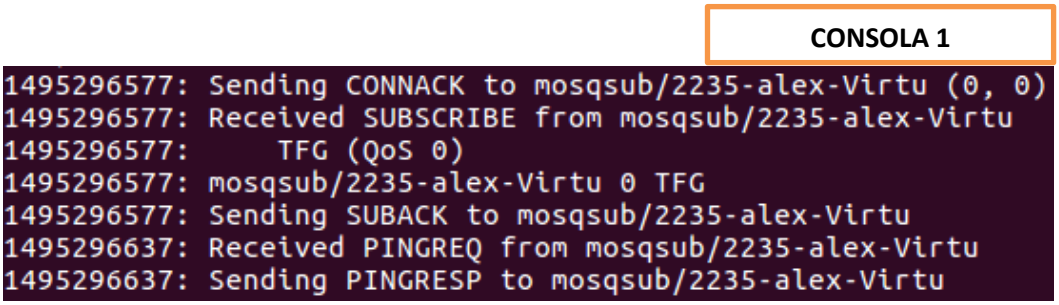



Prestando atención a la Consola 1, se puede observar este nuevo cliente registrado, enmarcado en rojo:

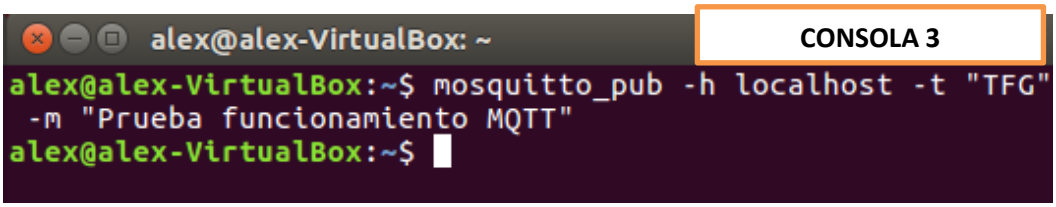



Aparece un nuevo cliente conectado en el puerto 1883 desde 127.0.0.1. Se envía un CONNACK al nuevo cliente y se recibe un SUSCRIBE al tópico “TFG” con una calidad de servicio de nivel 0 (QoS0).

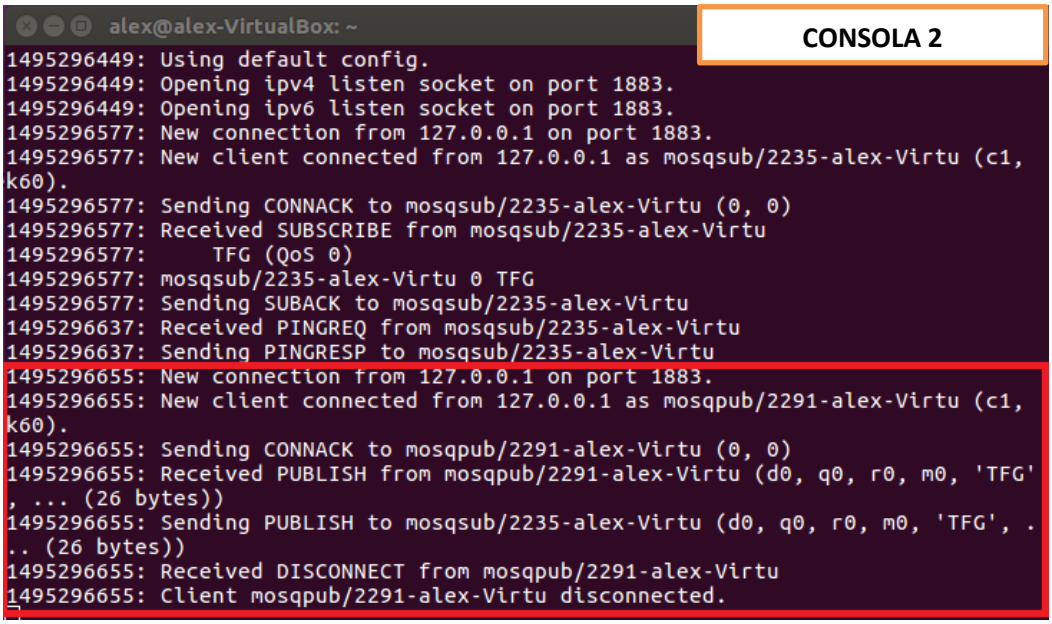
Una vez conectado, el bróker central envía continuamente PINGREQ y recibe PINGRESP para mantener conexión con el cliente. Si no llega el PINGRESP, se desconecta de esa suscripción.



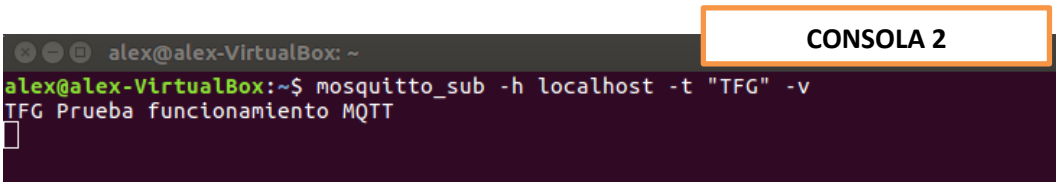

Para que el cliente suscrito al tópico “TFG” reciba un mensaje, otro cliente debe publicar un mensaje a ese tópico. Se abre una nueva terminal, Consola 3, y se prueba de la siguiente manera:




De nuevo, como se ha registrado un nuevo cliente, en este caso publicador, se puede visualizar la actividad en la Consola 1, registrado de la siguiente manera:




Como se observa, se recibe la conexión de un nuevo cliente, de tipo PUBLISH con tópico “TFG”, por lo que redirecciona el mensaje al suscritor, como se muestra en la Consola 2.




Por último, comentar que los clientes que publican un mensaje no están conectados permanentemente, si no que se crea una nueva conexión cada vez que se desea publicar algo.

## PARTE 2

A continuación se presenta con más detalle el código utilizado en la implementación.

El funcionamiento es muy sencillo. En primer lugar se crea una instancia de “Client”, que es la clase principal, utilizando su constructor.

*mqttc = mqtt.Client()*

Seguidamente hay que crear una conexión entre el cliente creado y el bróker. Para ello se utiliza la función “*connect()*”, que toma como parámetros:

- Host del bróker
- Puerto que utiliza el servidor host para conectarse
- KeepAliveInterval, que es el máximo tiempo permitido (en segundos) entre comunicaciones con el bróker.

Para este proyecto, los ajustes de MQTT son siempre los mismos. Se utiliza “localhost”, el puerto 1883 y un intervalo de mantenimiento de 45 segundos.

Ahora ya se puede pueden utilizar los métodos "*suscribe()*", para suscribirse a un tópico, "*publish()*", para publicar a un tópico determinado, y/o "*disconnect()*", para cerrar la conexión.

Como se tratan de manera distinta, a continuación se detalla el uso de los dos tipos de clientes que MQTT dispone: cliente publicador y cliente suscritor.

### Cliente publicador

Las siguientes líneas de código definen la función "*pub_msg()*", que en la documentación está descrita como ENVIAR_MENSAJE_MQTT, que es la función utilizada en cualquier proceso del sistema para enviar un mensaje al bróker de comunicaciones.

La función toma como primer argumento el tópico al cual se desea publicar el mensaje y como segundo argumento el mensaje en sí.

Como se ha comentado previamente, primero se instancia la clase y se crea una conexión con el bróker. Para publicar el mensaje, simplemente se utiliza la función "*publish()*", que toma como parámetros los mismos que "*pub_msg()*" toma como argumentos, tópico y mensaje. En este momento el mensaje se envía al bróker, encargado de redireccionarlo. Dado que el mensaje tiene que ser de tipo texto, se envía información en formato JSON, ya que permite estructurar en un mismo mensaje multitud de contenido disperso, siguiendo la noción de clave-valor.

Por último, se cierra la conexión con MQTT mediante la función "*disconnect()*".

```
import paho.mqtt.client as mqtt

# MQTT Settings
MQTT_Broker = "localhost"
MQTT_Port = 1883
MQTT_KeepAliveInterval = 45

def pub_msg(topic, message):
    mqttc = mqtt.Client()
    mqttc.connect(MQTT_Broker, int(MQTT_Port), int(Keep_Alive_Interval))
    mqttc.publish(topic, message)
    mqttc.disconnect()
```

### Cliente suscritor

Igual que en el caso anterior, primero se instancia la clase y se crea una conexión cliente-broker. Una vez creada esta conexión, a diferencia del cliente publicador, hay que mantener el flujo de tráfico de la red con el bróker. Para conseguir esto se utiliza el método "*loop_forever()*".

Cuando el cliente recibe un mensaje CONNACK por parte del bróker en respuesta a la conexión creada, se crea una función *callback* "*on_connect()*". En este *callback* es donde se suscribe al tópico correspondiente ya que en caso de

pérdida de conexión, cuando se reconecte, la suscripción se renovará sin perderla.

Dentro de "*on_connect()*" se puede suscribir a los tópicos que se deseen mediante la función "suscribe()", que toma como argumentos el tópico y la calidad de servicio (QoS), comentada previamente. Para este proyecto, dado que se realiza en red local, se utiliza un *QoS0*.

Por último, cualquier mensaje que llegue a este cliente a través de los tópicos que está suscrito, aparece en la función "*on_message()*". El parámetro que interesa es "msg" ya que es una estructura que contiene el mensaje (msg.payload) y el tópico (msg.topic).

```
import paho.mqtt.client as mqtt

# MQTT Settings
MQTT_Broker = "localhost"
MQTT_Port = 1883
MQTT_KeepAliveInterval = 45
MQTT_Topic = "TOPIC"

#Subscribe to all desired topics
def on_connect(mosq, obj, rc):
    mqttc.subscribe(MQTT_Topic, 0)

def on_message(mosq, obj, msg):
    message = msg.payload
    topic = msg.topic

def on_subscribe(mosq, obj, mid, granted_qos):
    pass

mqttc = mqtt.Client()

# Assign event callbacks
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_subscribe = on_subscribe

# Connect
mqttc.connect(MQTT_Broker, int(MQTT_Port), int(MQTT_KeepAliveInterval))

# Continue the network loop
mqttc.loop_forever()
```

Las acciones correspondientes se realizan a través de la función "*on_message()*", que dependiendo del tópico se llama a unas funciones u otras.

# ANEXO 3

En este Anexo se presenta la estructura y el funcionamiento de *Philips Hue Lights*, ya que es una de las tecnologías IoT que se han integrado en la elaboración del proyecto. El escenario de *Philips Hue Lights* es el siguiente.

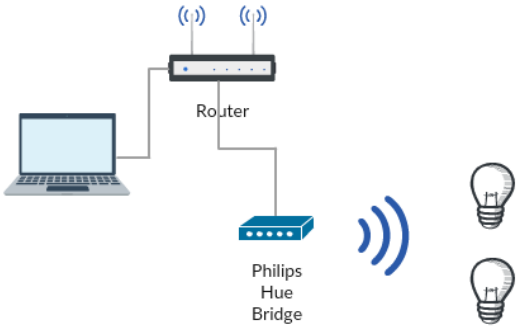
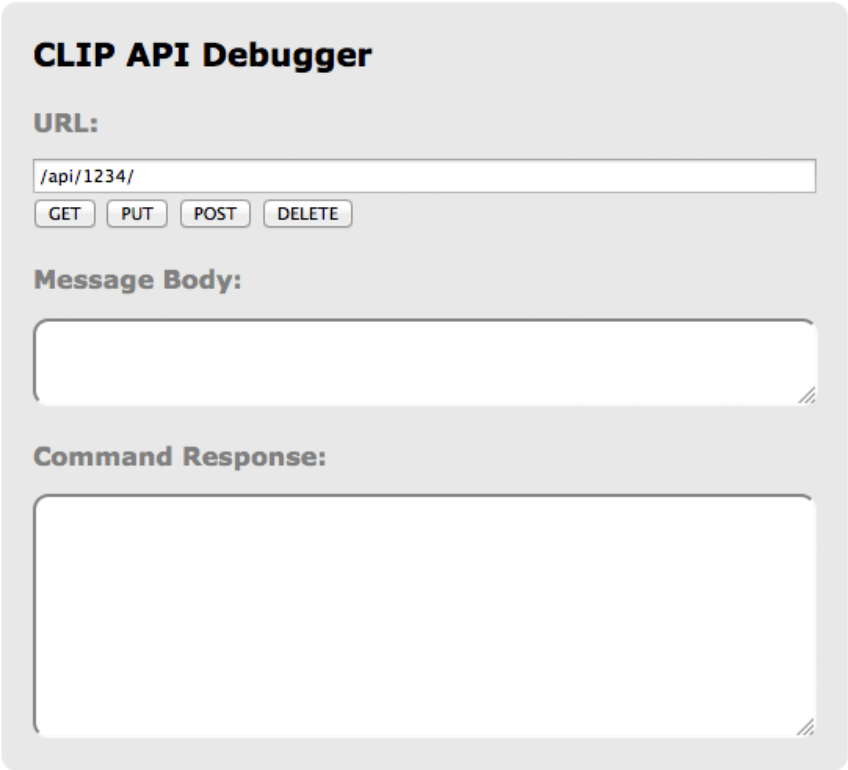



*Figura 26. Esquema Philips Hue Lights*

La API disponible para modificar los parámetros de los dispositivos se utiliza a través de una interfaz accesible a través de:

*http://<bridgeIPadress >/debug/clip.html*,

donde se carga el siguiente contenido.



Para poder enviar comandos a la bombilla, es obligatorio saber la dirección IP del *Hue Bridge*, ya que es el encargado de comunicarse con todos los dispositivos Philips. Una vez está conectado en la red local, es fácil recuperar esta información a través de la siguiente URL:

*http://www.meethue.com/api/nupnp*

Esta dirección devuelve la dirección IP y la dirección MAC del *Hue Bridge.* Una vez conocida la dirección IP, es necesario obtener un *username* para ser capaces de enviar comandos al router. Para ello se realizan los siguientes pasos:

- Se presiona el botón del bridge.
- Se llama a la siguiente dirección: *http://<bridgeIPadress>/api/* con un "BODY" tal que así: "{devicetype":"my_hue_app#<*username*>"}", realizando un "GET request".
- Se obtiene el *username* en la respuesta.

Con este *username* ya se puede conectar con el bridge para modificar parámetros. Cada dispositivo lumínico dispone de su propia URL. Se pueden saber qué dispositivos hay conectados llamando a la siguiente dirección:

*http://<bridge ip address>/api/<username>/lights*

Cada dispositivo tiene un número único asociado al bridge para cada usuario. Por lo tanto, para cambiar el estado del dispositivo lumínico 1, se realiza a través de:

*http://<bridgeIPaddress>/api/<username>/lights/1/state*

En el "BODY" se aplican los parámetros que se desean modificar en formato JSON y se realiza un "PUT request". Por ejemplo:

*{"on":true, "sat":254, "bri":254,"hue":10000}*