DISEÑO Y DESPLIEGUE DE UNA PLATAFORMA IOT COMPLETA BASADA EN COMPONENTES OPEN SOURCE

DESIGN AND DEPLOYMENT OF A COMPLETE IOT PLATFORM BASED ON OPEN SOURCE COMPONENTS

TRABAJO FIN DE MÁSTER
CURSO 2021-2022

AUTOR
ENRIQUE UGEDO EGIDO

DIRECTOR
FRANCISCO DANIEL IGUAL PEÑA

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DISEÑO Y DESPLIEGUE DE UNA PLATAFORMA IOT COMPLETA BASADA EN COMPONENTES OPEN SOURCE

DESIGN AND DEPLOYMENT OF A COMPLETE IOT PLATFORM BASED ON OPEN SOURCE COMPONENTS

TRABAJO DE FIN DE MÁSTER EN INGENIERÍA INFORMÁTICA
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

AUTOR
ENRIQUE UGEDO EGIDO

DIRECTOR
FRANCISCO DANIEL IGUAL PEÑA

CONVOCATORIA: JUNIO 2022
CALIFICACIÓN: 6

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

24 DE JUNIO DE 2022

DEDICATORIA

A mis tíos José y César.

A todas las personas que me han
acompañado durante mi dilatada
carrera como estudiante y representante.
Y a las que me han aguantado.

AGRADECIMIENTOS

A mi madre, mi hermana, mi tía, mis amigos Elena, Danis y Pablo, a mis compañeros Rita y Zarza, y a todas aquellas personas que me han insistido y presionado para terminar esto de una vez.

A mi hermana, que quiere este caso de uso para sus plantas.

A mi director, por la paciencia y el empuje.

A todos mis profesores del Grado en Ingeniería de Computadores, del Máster en Ingeniería Informática y del Máster en Internet de las Cosas, por los conocimientos que me han transmitido, algunos de los cuales se plasman en este proyecto, y por seguir saludándome por los pasillos.

A mis compañeros en Delegación de Alumnos y DCE durante todos estos años, y a mis Decanos y Vicerrectores que no preguntaban demasiado cuánto me faltaba para dejar de ser representante de estudiantes.

Al Tribunal de mi TFG, por los consejos que en su día me dieron y que me han servido con la elaboración de esta memoria.

A mis amigos que han aguantado sin verme durante las últimas semanas antes de la entrega del TFM.

A mis compañeres de otros saraos en los que estoy metido por entender mi desaparición temporal para centrarme en esto.

Finalmente, a la FDI y a la Complutense, por haber sido mi segunda casa.

ABSTRACT

Design and deployment of a complete IoT platform based on Open Source components

The Internet of Things is increasingly present in all kinds of commodity products and services. The present work demonstrates how it is possible to deploy a complete IoT platform, with its sensor/actuator nodes, gateway and clients. This project uses only Open Source components, which undoubtedly affects the cost of the infrastructure, making it accessible to all types of users and situations.

For the design of the platform, two components have been used: the Eclipse IoT community projects and the Zephyr project. Zephyr provides an operating system for nodes that can run on many boards on the market, many of them are low cost and with different types of communication interfaces. Zephyr also allows easy integration of Eclipse IoT projects.

Keywords

IoT, Open Source, Internet of Things, Internet, sensors, automation.

RESUMEN

Diseño y despliegue de una plataforma IoT completa basada en componentes Open Source

Internet de las Cosas está cada vez más presente en todo tipo de productos y servicios cotidianos. El presente trabajo demuestra cómo es posible desplegar una plataforma IoT completa, con sus nodos sensores/actuadores, su *gateway* y sus clientes, utilizando únicamente componentes Open Source, lo que sin duda repercute en el importe de la infraestructura, haciéndola accesible a todo tipo de usuarios y situaciones.

Para el diseño de la plataforma se utilizarán básicamente dos componentes, proyectos dentro de la comunidad Eclipse IoT y Zephyr Project. Zephyr proporciona un sistema operativo para nodos que puede ejecutarse en multitud de placas del mercado, muchas de ellas de bajo coste y con distintos tipos de interfaces de comunicaciones, permitiendo también integrar fácilmente los proyectos de Eclipse IoT.

Palabras clave

IoT, código abierto, Internet de las cosas, Internet, sensores, automatización.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

Desde su nacimiento durante la Guerra Fría como una herramienta de comunicación militar, Internet ha evolucionado a un ritmo vertiginoso revolucionando la vida mundial tal y como se conocía anteriormente. No hay más que ver la Figura 1-1 para darse cuenta de cómo en los últimos años el crecimiento de equipos conectados a Internet ha sido exponencial, más aun teniendo en cuenta que únicamente abarca hasta 2002, hace 20 años. [1] [2]

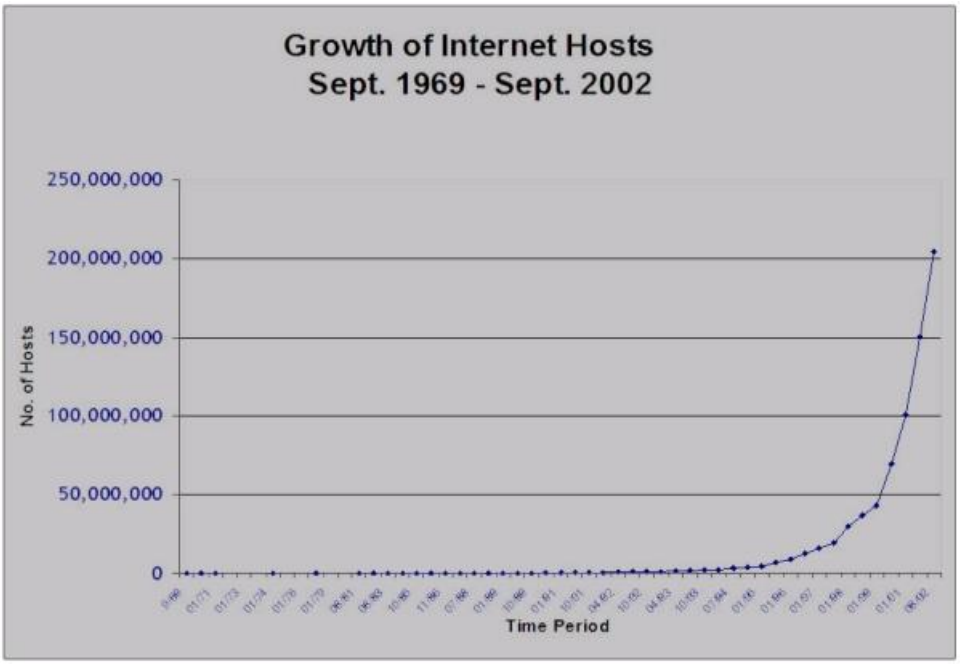

Figura 1-1. Gráfica de evolución de host conectados a Internet [2]

Tiene sentido el notable cambio de tendencia al alza al final de la gráfica, pues fue en 1982 cuando el Departamento de Defensa de los Estados Unidos decidió usar el protocolo TCP/IP en su red *Arpanet* creando así la red *Arpa Internet*, o *International Net*, que posteriormente quedaría con el nombre de Internet.

Más recientemente, en 1999, se empezó a utilizar el término "Internet de las Cosas", más conocido como IoT, por sus siglas en inglés (Internet Of Things). Fue el pionero británico Kevin Ashton quien empezó a utilizarlo para describir un sistema en el cual los objetos del mundo físico se podían conectar a Internet por medio de sensores. [3] [4] Y es que hoy en día es difícil pensar que algo, cualquier objeto cotidiano al que

se le puedan asignar unas funciones, no esté conectado a internet para poder automatizar, programar o activar y desactivar dichas funciones en remoto. Al contrario que hace unos años, no tantos, cuando era impensable que hasta unas zapatillas pudieran llevar un chip en la suela, o que electrodomésticos como nevera, o sobre todo equipos de climatización pudieran encenderse antes de llegar a casa. [5]

La definición de IoT podría ser la agrupación e interconexión de dispositivos y objetos a través de una red (bien sea privada o Internet, la red de redes), dónde todos ellos podrían ser visibles e interaccionar. Respecto al tipo de objetos o dispositivos podrían ser cualquiera, desde sensores y dispositivos mecánicos hasta objetos cotidianos como pueden ser el frigorífico, el calzado o la ropa. [6]

1.1 Motivación

El creciente uso de Internet en cosas de uso cotidiano y la proliferación de diferentes plataformas, la mayoría de ellas privativas, para crear el ecosistema que permita dar sentido a que esos objetos tengan internet, motiva la investigación objeto de este trabajo, que pretende demostrar cómo es posible dar cabida a todas las funciones típicas de una plataforma IoT utilizando únicamente dispositivos abiertos y *software libre*, es decir, con componentes Open Source, con las ventajas que ello conlleva.

La compatibilidad y escalabilidad de las plataformas es otra de las motivaciones, ya que muchos de los objetos con internet desarrollados por empresas utilizan *software privativo*, protocolos de comunicación no estándar o claves de cifrado en sus comunicaciones que no facilitan a sus usuarios. Incluso se da el caso de, aun usando protocolos estándar, se limita el uso de las antenas del dispositivo únicamente a aplicaciones del fabricante, como es el caso de la antena NFC de los dispositivos iPhone de Apple, cuestión que por cierto está en el punto de mira de la Comisión Europea por falta de competencia [7]. Con el uso de herramientas Open Source estos problemas no existen, si bien puede darse algún inconveniente de incompatibilidad, es evidente que resulta mucho más fácil comunicar dispositivos de diferentes fabricantes entre sí, a través de una red que puede ser también de otro fabricante diferente, si todos ellos utilizan tecnologías de comunicación IoT estándar, como pueden ser NB-IoT y LoRa, pero

también WiFi y Bluetooth, y protocolos estándar y abiertos, como MQTT. Algunos de estos conceptos se repasan en el Capítulo 2 - Estado de la cuestión.

1.2 Objetivos

El objetivo principal del proyecto es realizar una investigación detallada de la viabilidad del despliegue de una solución IoT completa basada en *software libre*. Para ello, en primer lugar, se debe definir de forma lo más específica posible el concepto de *framework*/infraestructura/despliegue IoT, identificando sus elementos principales (tanto *software* como *hardware*), los requisitos y limitaciones esperadas en cada uno de ellos, casos de uso que ilustren la interacción entre ellos, etc. Para cada uno de dichos actores, la segunda parte del estudio incluirá un repaso por el estado de la cuestión (alternativas) existentes en el mercado, haciendo especial hincapié en soluciones de *software libre* disponibles. Se valorará su madurez, funcionalidad, documentación, características reales, soporte, etc. Dicho estudio concluirá con la selección del producto de *software libre* que se considere más adecuado. La última parte del trabajo consistirá en la implementación real (bien sea en un entorno *hardware* real, o en un entorno virtualizado vía Máquinas Virtuales, contenedores, etc.) del entorno completo basado en las soluciones analizadas, ilustrando sus fortalezas y debilidades en base a casos de uso reales.

1.3 Plan de trabajo

Para la realización de este proyecto se estableció en el acuerdo de mínimos un plan de trabajo que, si bien es cierto que el comienzo del mismo se ha dilatado en el tiempo por diversos motivos, se ha ido cumpliendo de forma más o menos fidedigna desde el momento en que arrancó real y efectivamente la elaboración del mismo.

En el acuerdo alcanzado entre el director y el estudiante para la realización del presente Trabajo Fin de Máster se estableció la siguiente lista de tareas mínimas para presentarlo.

1) Estudio del estado de la cuestión, recopilación de información y documentación relativa a entornos/*frameworks* completos IoT. Se sugiere hacer hincapié en todos los proyectos disponibles en el marco de Eclipse IoT en niveles superiores y Zephyr a nivel

de nodo, ya que idealmente sería interesante que la solución propuesta se base en sus componentes.

2) División del trabajo en elementos principales que conformen un *framework* IoT según el estudio finalizado en 1). Al menos, se deben tratar soluciones a nivel de:

   a) Nodo sensor. La parte fundamental de todo despliegue IoT, es el dispositivo (o cosa) encargado de tomar las mediciones deseadas y, en caso de requerirse, activar los actuadores correspondientes.

   b) Gateway. Literalmente puerta de enlace, es el dispositivo en el borde de la red, o dispositivo frontera, habitualmente uno por despliegue IoT, que se comunica con todos los nodos del despliegue, ya sea por internet o por una red local, en cuyo caso servirá también de enlace de los nodos con internet. También puede servir para desplegar servicios de uso común por los dispositivos, habitualmente un panel de control o dashboard.

   c) Cloud. La nube donde se realiza la computación necesaria a partir de los datos obtenidos por los nodos así como se proveen de servicios de control de los datos y gestión de los nodos, entre otras cosas. Esta nube puede estar muchas veces en el propio *gateway*, en servidores remotos de internet, o contratarse a empresas especializadas en servicios Cloud IoT.

   d) Tecnologías de comunicación. Explorar las diferentes alternativas para interconectar los nodos entre sí o con su *gateway* y con Internet, así como los protocolos de red y comunicaciones.

      i) Protocolos de alto nivel. Relacionar las tecnologías de comunicación con los protocolos correspondientes por encima del nivel de transporte.

   e) Sistema Operativo. Tanto para los nodos, explorar RTOS (Sistemas Operativos de Tiempo Real) para elaborar su firmware, como para el *gateway* y el cliente, identificar el mejor tipo de sistema operativo para el propósito deseado. Procurar que el lado del cliente sea multiplataforma.

3) Para cada elemento identificado en 2), será necesaria su documentación, estudio y puesta en funcionamiento, identificando sus características principales, fortalezas y debilidades desde los puntos de vista que el alumno considere interesante.

4) En base a los elementos anteriores, se desarrollará e implementará un entorno completamente funcional basado exclusivamente en *software libre*, incluyendo casos de uso reales que ilustren las fortalezas/debilidades de cada componente (o del entorno de forma global).

5) Documentación y escritura de memoria.

Se acordó también establecer un sistema de seguimiento para comprobar el progreso del trabajo. Periódicamente, el estudiante debía dar cuenta del trabajo realizado durante el periodo correspondiente, los problemas encontrados y se acordará el trabajo a realizar durante el siguiente periodo. Esa comunicación podrá ser mediante reunión presencial/online (especialmente durante las primeras semanas), pero en cualquier caso mediante alguna herramienta de edición colaborativa a elección del alumno (Google Drive, Trello…)

Las modificaciones que se han realizado han sido fruto de las necesidades que se han ido viendo durante el desarrollo del proyecto y afectan fundamentalmente al orden y al tiempo dedicado a cada etapa. Adicionalmente se ha llevado un diario de bitácora, del que se presenta un resumen en el Apéndice B - , en el que se han ido anotando los avances, fundamentalmente con el objetivo de presentarlos en cada una de las reuniones que se ha mantenido en las que también se fijaban los siguientes pasos a seguir. De esta forma es más fácil evaluar el grado de cumplimiento del plan de trabajo.

Respecto a las reuniones, se ha tratado de mantener una cierta periodicidad dependiendo sobre todo de la dedicación al proyecto en cada época y de los hitos que se iban completando. En cualquier caso, en los periodos más largos se ha seguido haciendo un seguimiento vía correo electrónico. En la Tabla 1-1 se presenta un cronograma esquemático con las fechas en las que se mantuvo una reunión, se completó un hito, se encontró una dificultad o, en definitiva, se tomaron decisiones de calado para el transcurso del proyecto.

Fecha	Tipo de hito	Descripción
23-03-21	Comienzo	Se comienza la realización del TFM.
07-04-21	Planificación	Se establece el plan de trabajo.
25-10-21	Tarea completada	Presentación preliminar sobre el estado del arte.
10-11-21	Email	Mensaje de dudas simples y seguimiento de tareas.
13-12-21	Reunión	Se fija un calendario, continuar con el estudio y preparar un caso de uso con las plataformas vistas.
12-01-22	Reunión	Seguimiento de tareas y dudas simples.
14-02-22	Tarea completada	Caso de uso.
17-02-22	Reunión	Presentación del caso de uso.
03-03-22	Reunión	Seguimiento de tareas y resolución de dudas.
14-03-22	Reunión	Seguimiento de tareas y resolución de dudas.
25-03-22	Reunión	Seguimiento de tareas y resolución de dudas.
20-04-22	Inicio tarea	Se empieza a escribir la memoria.
08-05-22	Estado tarea	Se completan algunos epígrafes de la memoria.
30-05-22	Email relevante	Actualización de situación y análisis de diversas dificultades que se han encontrado en las últimas semanas. Se comentan también en el hilo otras dudas y cuestiones organizativas.
31-05-22	Reunión	Seguimiento de tareas y resolución de dudas.
10-06-22	Dificultad importante	Se descubre incompatibilidad de ADC para ESP32 con Zephyr.
13-06-22	Dificultad solventada	Se resuelve cambiando los sensores utilizados.
16-06-22	Tarea completada	Se completa el código de la aplicación.
17-06-22	Borrador de memoria	Borrador de la memoria en documento compartido.
24-06-22	Entrega final	Se realiza la entrega de la memoria.

Tabla 1-1. Cronología del plan de trabajo

Capítulo 2 - Estado de la cuestión

En este capítulo se realiza un repaso por algunas de las diferentes alternativas que existen actualmente para el tema del que versa este trabajo, tanto de plataformas completas ya existentes como de los componentes individualizados que las conforman, fundamentalmente software y hardware.

Las plataformas IoT se componen básicamente de dos tipos de componentes, *software* y *hardware*. Dentro de ambos tipos existen diferentes categorías dependiendo de las funciones que desempeñan o soluciones que ofrecen. Así por ejemplo en el caso de hardware se podría hablar fundamentalmente de los sensores y de los actuadores, pero también de los componentes que van asociados a ellos, como las placas de expansión a las que se ensamblan y que les dotan de conexión a Internet, pues no serían sino "cosas del internet de las cosas". Del lado del *software* está el propio *firmware* o controlador del *hardware*, así como también hay protocolos de comunicación, de seguridad, las funcionalidades que automatizan las respuestas del sistema ante determinadas situaciones, etcétera. En el mercado pueden encontrarse, además de multitud de componentes independientes, de cualquiera de esos dos tipos, y con mayor o menor compatibilidad con otros componentes, soluciones que presentan una plataforma completa, con compatibilidad garantizada y solución para todos los componentes básicos de una plataforma IoT y quizá algunos más.

2.1 Entornos de Internet de las Cosas

Dado que lo que este proyecto plantea es la creación de una plataforma IoT completa, procede fijarse en qué plataformas o entornos similares existen en el mercado. Cabe decir en este punto que la plataforma que se presenta en este proyecto es en realidad la combinación del uso de distintas plataformas software en distinto hardware compatible, pues así es como se puede llegar a un entorno completo. Lo cierto además es que resulta complicado encontrar soluciones comerciales completas, que incluyan tanto su hardware como su software, si bien lo más habitual es encontrar conjuntos de herramientas software con compatibilidad con el hardware más común en el mercado, sin que éste forme necesariamente parte de la solución.

2.1.1 Bosch IoT Suite

La plataforma IoT de la empresa Bosch, constituye la base de las soluciones y servicios IoT que ofrece Bosch, con un claro afán de encabezar el Internet de las Cosas para Bosch y sus clientes. Esta Suite proporciona soluciones IoT para la industria, agricultura, energía, construcción, domótica y movilidad, además de servir de plataforma para proyectos personalizados.

La Bosch IoT Suite se presenta como "una plataforma de software IoT abierta para todos los dominios", sin embargo, la existencia de un extenso catálogo de dispositivos[1] desarrollados en su mayoría por la propia Bosch, que también los vende y tutoriza cómo conectarlos a la Suite, hace que se pueda considerar este entorno como una plataforma completa. Además, la amplia compatibilidad y soporte que proporciona para otros productos, servicios y protocolos la convierten en una solución heterogénea y muy versátil. [8]

Bosch ofrece además su propia nube de servicios, denominada Bosch IoT Cloud, como un marketplace en la que disponer de aplicaciones basadas en la nube o en activos, aplicaciones distribuidas para IoT, *Digital Twin* o herramientas sociales para IoT. Conformando así la pata *IaaS* (Internet as a Service: Internet como servicio) del despliegue IoT. Esto no impide, sin embargo, que puedan utilizarse servicios o plataformas de terceros, como por ejemplo Amazon Web Services, ya que Bosch IoT Suite es compatible con ellos.

Toda la Bosch IoT Suite está basada en Open Source y Open Standards, lo que aumenta notablemente su compatibilidad y soporte. Algunas de las aplicaciones utilizadas por las diferentes herramientas de Bosch IoT Suite, destacadas en la Figura 2-1, se describen más adelante. [9] [10]

Actualmente Bosch y la Universidad Complutense de Madrid cuentan con una Cátedra extraordinaria en Inteligencia Artificial aplicada a Internet de las Cosas.[2]

---

[1] https://bosch-iot-suite.com/iot-devices/

[2] https://www.ucm.es/catedrabosch/

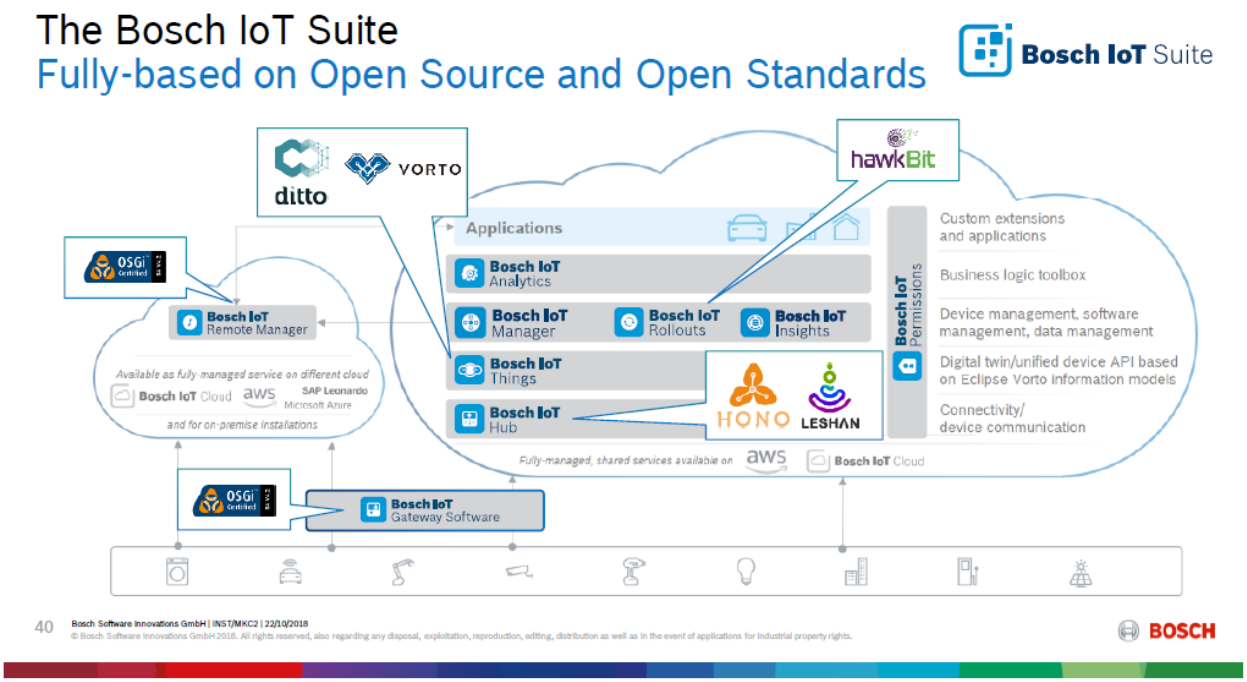

Figura 2-1. Esquema general del software y servicios en la nube de Bosch IoT Suite. [8]

2.1.2 Arduino

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso. [11]

Está destinada a la creación de proyectos interactivos, como puede ser un proyecto IoT. Si bien Arduino no presenta por sí sola una plataforma IoT completa como tal, lo cierto es que ofrece las herramientas suficientes para poder crear casi cualquier despliegue IoT que se plantee desarrollando tanto el *hardware* como el *software* íntegramente en Arduino, o utilizando además algún otro proyecto IoT como de hecho suelen integrar en sí las demás plataformas completas.

El proyecto nació en 2003, cuando varios estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de facilitar el acceso y uso de la electrónica y programación. Lo hicieron para que los estudiantes de electrónica tuviesen una

alternativa más económica a las populares BASIC Stamp[3], unas placas que por aquel entonces valían más de cien dólares, y que no todos se podían permitir.

El resultado fue Arduino, una placa con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y que puede ser programada tanto en Windows como macOS y GNU/Linux. Un proyecto que promueve la filosofía *'learning by doing'*, que viene a querer decir que la mejor manera de aprender es cacharreando. [11]

Las placas Arduino[4] pueden leer entradas (luz en un sensor, un dedo en un botón o un mensaje de Twitter) y convertirlo en una salida (activar un motor, encender un LED, publicar algo en línea). Para enviar el conjunto de instrucciones necesario al microcontrolador en la placa se utiliza el lenguaje de programación Arduino[5] (basado en Wiring[6]), y el Software Arduino (IDE)[7], (basado en Processing[8]). [12]

2.1.3 Raspberry Pi

De forma similar a Arduino, Raspberry Pi proporciona herramientas tanto software como hardware que pueden ser empleadas en el desarrollo de despliegues IoT completos.

Con su eslogan "Informática para todos", Raspberry Pi pretende que la informática sea accesible y asequible para todos, desde industrias grandes y pequeñas, hasta uso doméstico. [13]

Raspberry Pi para el hogar se describe como "El gran ordenador que cabe en tu mano". Un pequeño y poderoso ordenador de bajo coste. Raspberry Pi tiene todas las

---

[3] https://es.wikipedia.org/wiki/BASIC_Stamp

[4] https://www.arduino.cc/en/Main/Products

[5] https://www.arduino.cc/en/Reference/HomePage

[6] http://wiring.org.co/

[7] https://www.arduino.cc/en/Main/Software

[8] https://processing.org/

funciones esperables en un ordenador: conectividad inalámbrica a Internet, puertos HDMI para monitores y puertos USB para accesorios, junto con una amplia potencia de procesamiento y RAM para todo su uso diario. Además, funciona con un consumo energético de tan solo 15 vatios, mucho menos que cualquier PC tradicional. [14]

2.1.3.1 Raspberry Pi OS

Respecto al *software*, Raspberry Pi OS (anteriormente llamado Raspbian) es el Sistema Operativo compatible oficial.

Es gratuito y proporciona un entorno de escritorio familiar con todo lo habitual en un SO, como un navegador web, un procesador de textos y una aplicación de hoja de cálculo. Lo que permite utilizar el sistema operativo Raspberry Pi de manera intuitiva desde el primer momento. Además, recibe actualizaciones periódicas gratuitas, lo que garantiza que el dispositivo Raspberry Pi cuente con las últimas opciones de accesibilidad, actualizaciones y parches de seguridad. [15]

2.2 Software

2.2.1 Eclipse IoT

Es una iniciativa de la Fundación Eclipse para impulsar las soluciones IoT comerciales líderes en el mundo.

El grupo de trabajo de eclipse IoT es una comunidad de proveedores neutrales pertenecientes a la Fundación Eclipse que desean colaborar en el desarrollo de nuevas tecnologías en el ámbito del IoT. Se puede participar en las reuniones y actividades de la comunidad sin ser miembro.

Los proyectos de Eclipse IoT son de código abierto y pueden ayudar a la creación de otros proyectos. Existe un buscador para encontrar el más adecuado a cada objetivos o necesidad. Muchos de los proyectos aún se encuentran en estado de "incubación", lo que demuestra que Eclipse IoT está en constante crecimiento. Pero también hay proyectos en estado "regular" suficientes como para constituir una plataforma IoT completa y funcional. [16]

Existen diferentes categorías o etiquetas en las que se enmarcan los diferentes proyectos en función de las soluciones que ofrecen. Estas categorías coinciden en gran medida con los elementos típicos que componen un despliegue IoT, tanto a nivel físico como de red, comunicaciones y seguridad. Si bien, el continuo avance y desarrollo de herramientas y la aparición de nuevas necesidades y nuevos proyectos que las cubren hace que esta lista de categorías no sea algo cerrado, sino que se adapte a la vertiginosa evolución de Internet de las Cosas.

A continuación, se presentan algunos de los proyectos más destacados ya que la lista completa[9] consta de más de cuarenta proyectos.

2.2.1.1 Eclipse Leshan™

Servidor y cliente *OMA Lightweight M2M* en Java. *OMA Lightweight M2M* es un protocolo de Open Mobile Alliance para la gestión de dispositivos M2M o IoT. Leshan proporciona bibliotecas que ayudan a desarrollar un cliente y servidor M2M ligero. Forma parte de los proyectos Eclipse desde 2014. [17]

Es uno de los proyectos más completos para gateways, que incluye standards IoT y políticas de seguridad orientadas a ejecutarse en la nube.

2.2.1.2 Eclipse Kura™

Contenedor Java para aplicaciones M2M

Eclipse Kura™ es un IoT Edge Framework extensible de código abierto basado en Java/OSGi. Kura ofrece acceso API a las interfaces de hardware de IoT Gateways (puertos serie, GPS, watchdog, GPIO, I2C, etc.). Cuenta con protocolos de campo listos para usar (incluidos Modbus, OPC-UA, S7), un contenedor de aplicaciones y una programación de flujo de datos visual basada en la web para adquirir datos del campo, procesarlos en el perímetro y publicarlos en Plataformas líderes en la nube de IoT a través de la conectividad MQTT. [18]

[9] https://iot.eclipse.org/projects/

2.2.1.3 Eclipse Ditto™

Eclipse Ditto es un *framework* de código abierto para crear *Digital Twins* (gemelos digitales) de dispositivos conectados a Internet. Su diseño no se centra en un entorno de uso concreto y, por lo tanto, puede usarse en entornos industriales, residenciales, agrícolas y muchos otros entornos de IoT.

Ditto actúa como *middleware* de IoT, proporcionando una capa de abstracción para las soluciones de IoT que interactúan con dispositivos físicos a través de gemelos digitales como interfaz. Los dispositivos se integran a través de capas de conectividad como Eclipse Hono™ o *brokers MQTT* como Eclipse Mosquitto™.

Los gemelos digitales administrados en Ditto también se pueden integrar en otros sistemas *back-end* existentes mediante la creación de conexiones arbitrarias a dichos sistemas externos utilizando los protocolos admitidos. [19]

2.2.1.4 Eclipse Hono™

Proporciona interfaces de servicio remoto para conectar una gran cantidad de dispositivos IoT a un *back-end* e interactuar con ellos de manera uniforme, independientemente del protocolo de comunicación del dispositivo.

Está diseñado para conectar una gran cantidad de dispositivos IoT. Su arquitectura basada en micro servicios permite escalar horizontalmente. Conectando todos los dispositivos al *back-end* con Hono se puede comunicar con ellos mediante la misma API. Se admiten comunicaciones por protocolos comunes como HTTP, MQTT, AMQP y CoAP. La seguridad viene activada por defecto, admitiendo mecanismos de autenticación que van desde nombre de usuario y contraseña hasta certificados de cliente X.509[10] y utiliza la capa de seguridad de transporte (TLS[11]) para comunicarse con los dispositivos. Además, proporciona un mecanismo simple para soportar protocolos personalizados. [20]

---

[10] https://www.ssl.com/es/preguntas-frecuentes/¿Qué-es-un-certificado-x-509?/

[11] https://datatracker.ietf.org/doc/html/rfc8446

2.2.1.5 Eclipse Kapua™

Es una plataforma modular de integración en la nube para dispositivos IoT y sensores inteligentes que tiene como objetivo unir la tecnología de operación con la tecnología de la información. Sirve para administrar e integrar los dispositivos y sus datos. Una sólida base integrada de servicios de IoT para cualquier aplicación de IoT.

Los dispositivos pueden conectarse a Kapua vía MQTT y otros protocolos. Permite administrar de forma remota las aplicaciones, configuraciones y recursos de los dispositivos. Almacena e indexa los datos publicados por los dispositivos IoT para un rápido análisis y visualización en paneles (*dashboard*). Se puede integrar con otras aplicaciones TI mediante API REST. [21]

2.2.1.6 Eclipse Mosquitto™

Proporciona una implementación de servidor ligero del protocolo MQTT que es adecuada para todas las situaciones, desde máquinas de máxima potencia hasta máquinas integradas y de baja potencia. Los sensores y actuadores, que a menudo son las fuentes y los destinos de los mensajes MQTT, pueden ser muy pequeños y carentes de energía. Esto también se aplica a las máquinas integradas a las que están conectados, que es donde se podría ejecutar Mosquitto. [22]

2.2.1.7 Eclipse Californium

Se divide en cinco subproyectos. *Californium (Cf) Core* proporciona el marco central con la implementación del protocolo para crear aplicaciones de Internet de las cosas. El repositorio también incluye proyectos de ejemplo para comenzar. Todo el código fuente de Californium está alojado en GitHub, por lo que se puede contribuir fácilmente a través de *pull requests*.

Californium (Cf) tiene licencia dual bajo EPL y EDL. Esta última es una licencia similar a BSD, lo que significa que el marco Cf CoAP se puede usar junto con código propietario.

Cf está disponible en *Maven Central*[12] y es muy fácil de incorporar a cualquier proyecto Java. [23]

2.2.2 El proyecto Zephyr

The Zephyr® Project es un proyecto colaborativo de código abierto de The Linux Foundation® en el que desarrolladores y usuarios crean un Sistema Operativo en Tiempo Real (RTOS por sus siglas en inglés: *Real Time Operating System*).

Sus características más destacables es que es ligero, escalable, está optimizado para dispositivos con recursos limitados y existe para múltiples arquitecturas.

El código fuente de *The Zephyr® Project* está abierto en un repositorio *git*. Cuenta con una herramienta de línea de comando, llamada *West* (*Zephyr's meta-tool*), que facilita la instalación de múltiples repositorios Zephyr y características adicionales. [24]

En su página web oficial existen multitud de recursos que conviene destacar:

- *Demos* para múltiples usos, con su código fuente, descripción, casos de uso e instrucciones paso a paso.
- Listado[13], no escaso, de placas compatibles. Incluido QEMU, un emulador de procesadores gratuito y de código abierto basado en la traducción dinámica de binarios. [25]
- Documentación, tanto para la instalación del entorno y uso de los ejemplos como de la API para desarrollar nuevas aplicaciones.
- Información sobre reuniones de la comunidad.
- Herramientas como GCC, Renode, Nordic nrfx, NXP MCUXpresso SDK, Synopsys Designware…
- Vídeos
- Presentación de miembros y comunidad.

---

[12] https://mvnrepository.com/repos/central

[13] https://docs.zephyrproject.org/latest/boards/index.html

- Enseñanza. Disponen de un apartado donde se explican:
    - Aplicaciones en funcionamiento
    - Arquitectura del SO
    - Beneficios
        - Seguridad, Open Source, soporte, poderoso, modular y escalable, herramientas de configuración, conectividad.
- Preguntas frecuentes

Como es habitual en todo proyecto de código abierto, existe detrás una enorme comunidad que en este caso utiliza múltiples herramientas para comunicarse, tales como un blog, eventos, notas de prensa, *Discrod*, redes sociales, listas de correo, etcétera.

2.2.2.1 La herramienta West

El proyecto Zephyr incluye una herramienta de línea de comando llamada West. Los comandos integrados de West proporcionan un sistema de administración de múltiples repositorios con funciones inspiradas en la herramienta Repo[14] de Google y los submódulos Git. West también es "conectable": se pueden escribir comandos personalizados de extensión West que agregan características adicionales a West. Zephyr usa esta herramienta para brindar conveniencia para crear aplicaciones, actualizarlas y depurarlas. [26]

2.2.3 AWS IoT Greengrass

AWS IoT Greengrass es un servicio Open Source de *edge runtime* en la nube. Permite crear, implementar y administrar aplicaciones IoT en los dispositivos. Se puede utilizar para crear software que permita a los dispositivos actuar localmente sobre los datos que generan, ejecutar predicciones basadas en modelos de aprendizaje automático y filtrar y agregar datos de dispositivos.

[14] https://gerrit.googlesource.com/git-repo/

Permite también que los dispositivos recopilen y analicen datos más cerca del lugar en el que se generan los datos, reaccionen de forma autónoma a eventos locales y se comuniquen de forma segura con otros dispositivos de la red local. Los dispositivos Greengrass también pueden comunicarse de forma segura con AWS IoT Core[15] y exportar los datos de la nube y a la nube de AWS. Permite crear aplicaciones perimetrales utilizando módulos de software preconstruidos, llamados componentes, que pueden conectar los dispositivos perimetrales a servicios AWS de terceros. También se puede utilizar AWS IoT Greengrass para empaquetar y ejecutar el software utilizando funciones *Lambda*[16], contenedores *Docker*[17], procesos nativos del sistema operativo o tiempos de ejecución personalizados de elección del usuario o desarrollador. [27]

2.2.4 Azure IoT

Microsoft también proporciona una serie de productos y servicios IoT comprendidos en distintas plataformas dentro de Azure IoT[18] de entre las que destacan las siguientes:

- Azure IoT Hub

Habilita una comunicación segura y confiable entre la aplicación IoT y los dispositivos que esta administra. Azure IoT Hub ofrece un back-end de solución hospedado en la nube que permite conectarse prácticamente a cualquier dispositivo. [28]

- Azure Digital Twins

Ofrece información que permita mejorar los productos y optimizar las operaciones y los costes, para crear experiencias de cliente innovadoras.

---

[15] https://docs.aws.amazon.com/iot/?id=docs_gateway

[16] https://aws.amazon.com/lambda/

[17] https://www.docker.com/

[18] https://azure.microsoft.com/es-es/overview/iot/#overview

Junto con Azure IoT Hub proporcionan los bloques de creación para que las empresas construyan soluciones personalizadas para escenarios de IoT complejos. [29]

- Azure IoT Central

Permite a las empresas empezar a crear rápidamente aplicaciones de IoT con una oferta de soluciones de IoT totalmente administrada.

Es altamente seguro, escalable, garantiza que las inversiones sean repetibles y se integra con las aplicaciones empresariales ya existentes. [30]

- Azure IoT Edge

Permite ampliar la inteligencia y los análisis de la nube moviendo las cargas de trabajo y la lógica de negocios de la nube a los dispositivos perimetrales. [31]

También proporciona dos tipos de Sistemas Operativos:

- Windows para IoT: Sistema operativo a largo plazo y servicios para administrar los dispositivos. [32]
- Azure RTOS: que realmente es un conjunto de aplicaciones para el desarrollo de soluciones insertadas que incluye un sistema operativo pequeño que proporciona un rendimiento ultrarrápido y confiable para los dispositivos con recursos limitados. Admite microcontroladores de 32 bits y las herramientas de desarrollo empotrado más populares. [33]

2.2.5 Google Cloud IoT Core

Es un servicio totalmente gestionado que permite conectar y administrar dispositivos de IoT de manera segura, desde unos pocos hasta millones. Transfiere datos desde dispositivos conectados de forma segura y sencilla y compila aplicaciones sofisticadas que se integran con los otros servicios de macrodatos de Google Cloud Platform [34]. Esto pone a disposición del desarrollador una solución de lo más completa para recopilar, procesar, analizar y visualizar datos del Internet de las cosas en tiempo real, lo que se traducirá en una mejora de la eficiencia operativa. [35]

2.3 Hardware

2.3.1 ESP32

Desarrollada por Espressif, al igual que otras placas como ESP8266 y las series que evolucionan de la propia ESP32, su versatilidad, características y reducido coste, unido a la alta disponibilidad, asunto no tan obvio en estos tiempos de escasez de chips, la han convertido quizás en la placa más usada a la hora de desarrollar dispositivos IoT autónomos y de bajo coste para todo tipo de soluciones y, sin duda, la mejor opción para el desarrollo de este trabajo.

La ESP32 es una MCU rica en funciones con conectividad Wi-Fi y Bluetooth integradas para una amplia gama de aplicaciones. Cuenta con un diseño robusto, capaz de funcionar de forma fiable en entornos industriales, con una temperatura de funcionamiento que oscila entre –40 °C y +125 °C. Logra un consumo de energía ultra bajo con una combinación de varios tipos de software patentado. ESP32 también incluye características de última generación, como activación de reloj de grano fino, varios modos de potencia y escalado dinámico de potencia. Puede funcionar como un sistema independiente completo o como un dispositivo esclavo de una MCU anfitriona, lo que reduce la sobrecarga de la pila de comunicación en el procesador de la aplicación principal. ESP32 puede interactuar con otros sistemas para proporcionar funcionalidad Wi-Fi y Bluetooth a través de sus interfaces SPI/SDIO o I2C/UART. [36]

2.3.2 STM32 Nucleo

La placa STM32 Nucleo-64 proporciona una forma asequible y flexible para que los usuarios prueben nuevos conceptos y construyan prototipos eligiendo entre las diversas combinaciones de funciones de rendimiento y consumo de energía que proporciona el microcontrolador STM32. Para las placas compatibles, el SMPS externo reduce significativamente el consumo de energía en modo *Run*. El soporte de conectividad ARDUINO® Uno V3 y los encabezados *ST morpho* permiten la fácil expansión de la funcionalidad de la plataforma de desarrollo abierta STM32 Nucleo con una amplia variedad de escudos especializados. No requiere ninguna sonda separada ya que integra el depurador/programador ST-LINK. [37]

2.3.3 Sensortag

Los kits SensorTag de Texas Instruments son kits de demostración de IoT de última generación que combinan datos de sensores con conectividad en la nube. Estos kits facilitan el desarrollo IoT, con una conexión segura a la nube. Cuenta con una aplicación SensorTag (disponible para iOS y Android) que sirve para conectar sus datos a la nube en solo unos minutos. Estos kits proporcionan acceso a múltiples tecnologías de conectividad inalámbrica. Están disponibles en tres variantes. El CC2650STK cubre múltiples estándares compatibles con *Bluetooth Low Energy*, 6LoWPAN y ZigBee. El CC1350STK cuenta con conectividad *Bluetooth Low Energy* e inalámbrica de largo alcance Sub-1GHz. El CC3200STK-WIFIMK utiliza *Wi-Fi SimpleLink*, tecnología inalámbrica de baja potencia. Cada uno de estos dispositivos funciona con muy bajo consumo por lo que requieren poca o ninguna batería (puede ser suficiente una pila de botón convencional), lo que le permite estar funcionando durante meses o años. [38]

2.3.4 Apple AirTag

El Apple AirTag es un sencillo rastreador con tecnología IoT que convierte cualquier objeto que se quiera localizar, por ejemplo, en caso de pérdida, en un objeto IoT con la función de rastreo y localización.

Utiliza la tecnología inalámbrica de banda ultra-ancha (UWB)[19], que permite a los dispositivos compatibles, fundamentalmente los iPhone, pero también algunos iPads y otros productos de Apple, detectar con precisión de centímetros la distancia a la que se encuentra el AirTag y con un rango de decenas de metros. [39]

Existen varios dispositivos como este que funcionan únicamente con Bluetooth, tecnología que también incorpora el AirTag, sin embargo, lo interesante en este caso es que, los AirTag utilizan una red creada por los más de 1650 millones de dispositivos Apple activos en el mundo para ubicar con precisión cualquier AirTag, se encuentre a la distancia que se encuentre de su propietario. En resumen, el AirTag se comunica con los iPhones y otros dispositivos Apple cercanos enviando de forma encriptada un

[19] https://blogthinkbig.com/ultra-wideband

identificador único. Estos dispositivos, que están conectados a Internet y que están geolocalizados, comunican con los servidores de Apple la detección de ese AirTag y su ubicación. Esto permite a su propietario localizarlo a través de esa red creada por los dispositivos Apple activos [40].

Sin duda un caso muy interesante de aplicación de Internet de las Cosas.

2.4 Tecnologías y protocolos de comunicación

2.4.1 RFID

RFID son las siglas en inglés de *Radio Frecuency Identification* o Identificación por Radiofrecuencia en español. Tras estas siglas, que definen lo que permite hacer RFID, está una tecnología capaz de almacenar, recuperar y re-grabar una inmensa cantidad de datos (hasta cuatro millones de caracteres y millares de bytes) en un pequeño chip y transmitirlos a través de ondas de radio o radiofrecuencia.

RFID permite incorporar además a ese chip un Electronic Product Code (Código Electrónico de Producto o EPC) único para cada etiqueta. [41]

Resulta interesante recordar en este punto como la tecnología RFID fue la primera con la que empezó a utilizarse el término "Intertet de las Cosas", cuando el comerciante de P&G Kevin Ashton realizó una presentación para expender el uso de etiquetas RFID en los productos a la venta de las tiendas. [42]

2.4.2 MQTT

Es un protocolo de mensajería estándar de OASIS para Internet de las cosas (IoT). Está diseñado como un transporte de mensajería de publicación/suscripción extremadamente ligero que es ideal para conectar dispositivos remotos con un espacio de código pequeño y un ancho de banda de red mínimo. MQTT hoy en día se utiliza en una amplia variedad de industrias, como la automotriz, la manufactura, las telecomunicaciones, el petróleo y el gas, etc. [43]

Sus siglas significan *Message Queue Telemetry Transport*. La gestión de los mensajes se realiza en un *broker*, y sus clientes publican los mensajes hacia el *broker* y/o se suscriben a él para recibirlos. Los mensajes se organizan por *topics*, que básicamente son etiquetas que actúan como un sistema para determinar a qué temas un cliente se suscribe o envía mensajes. [44]

2.4.3 HTTP

El *Hypertext Transfer Protocol* (Protocolo de transferencia de hipertexto en español) es un protocolo de nivel de aplicación para sistemas de información hipermedia distribuidos y colaborativos. Es un protocolo genérico, sin estado, que se puede utilizar para muchas tareas más allá de su uso para hipertexto, como servidores de nombres y sistemas de gestión de objetos distribuidos, a través de la extensión de sus métodos de solicitud, códigos de error y encabezados. Una característica de HTTP es la escritura y negociación de la representación de datos, lo que permite que los sistemas se construyan independientemente de los datos que se transfieren. La iniciativa de información global de la World-Wide Web utiliza HTTP desde 1990. [45]

2.4.4 AMQP

AMQP son las siglas de *Advanced Message Queuing Protocol*, Protocolo Avanzado de Colas de Mensajes en español. Es un estándar abierto para un protocolo de mensajería asíncrona interoperable a escala empresarial, relativamente sencillo pero muy potente. [46]

Conecta sistemas, alimenta los procesos empresariales con la información que necesitan y transmite de forma fiable las instrucciones que logran sus objetivos. [47]

2.4.5 CoAP

Constrained Application Protocol (CoAP), es un protocolo cliente-servidor que, a diferencia de MQTT, aún no está estandarizado al 100%. Con CoAP, un nodo cliente puede comandar a otro nodo vía un paquete CoAP. El servidor CoAP lo interpretará, extraerá la información del paquete, y decidirá qué acción realizar dependiendo de su lógica. El servidor no necesariamente realizará la confirmación de la petición. [44]

Capítulo 3 - Caso de uso

Este capítulo tiene se explica el caso de uso que se ha pensado y sobre el que se ha basado todo el desarrollo del proyecto para dar solución al mismo. El objetivo de este caso de uso es plantear una situación realista con una aplicación útil sobre alguna tarea cotidiana, pero que a la vez ilustre de forma completa el tipo de necesidades que tendría cualquier otro caso de uso de análoga finalidad. Sirva por tanto este caso de uso para ejemplificar cualquier otra situación en la que un nodo o red de nodos con una serie de sensores y actuadores y conectados a un sistema central pueda automatizar o simplificar tareas o simplemente monitorizar ciertas variables.

La situación es la de, en un cultivo cualquiera, hacer un seguimiento de la temperatura y la humedad, así como automatizar la apertura y cierre de los sistemas de regadío existentes. Para ello se tienen en cuenta diferentes casuísticas que van más allá de los simples *flags* de apertura o cierre de un grifo en función de determinados umbrales de temperatura, sino que se pueden establecer también niveles de alerta, tiempos máximos de regado seguido para evitar encharcamiento etcétera.

Una de las alarmas que se puede programar es un umbral de temperatura mínima, por debajo del cual se estime que existe riesgo de congelación de las tuberías y, en caso de alcanzarse, abrir mínimamente y de forma intermitente la llave de paso para mantener un pequeño flujo de corriente que impida dicho congelamiento, pero que tampoco sea muy alto para evitar el derroche de agua, así como la congelación de los cultivos regados a esas bajas temperaturas. De hecho, también se podría establecer un umbral de temperatura mínimo por debajo del cual no se activa el riego a pesar de que el índice de humedad sea bajo, pues regar a temperaturas bajas puede ser más perjudicial para el cultivo, por el riesgo de congelamiento, que pasar unas horas con humedad inferior a la deseable. Aunque este último caso parece físicamente difícil de darse, no está de más ser precavido y dicha situación se puede simular para comprobar el correcto funcionamiento del sistema.

Un entorno real donde podría utilizarse este caso de uso es en un invernadero. En un lugar de esas características tiene sentido que la humedad sea similar en toda la estancia y los aspersores o sistemas de pulverización se activen de forma simultánea en toda ella, por lo que podría bastar con un único nodo para realizar las mediciones de humedad, o si se desea más precisión utilizar más nodos y establecer que la humedad deba disminuir en varios de ellos antes de activar el riego. Sin embargo, en otro posible entorno real como serían cultivos al aire libre, tiene sentido sectorizar la sensorización, sobre todo si no toda la extensión del terreno soporta las mismas condiciones, en lo que a sombras solares o velocidad del viento se refiere, factores ambientales que más pueden afectar tanto a la humedad relativa del aire como a la de la tierra.

Se detallan a continuación los diferentes componentes de que consta el sistema.

3.1 Nodo

Su función principal en este caso de uso será mantener los niveles mínimos de humedad relativa del lugar en el que se instale, que puede ser un invernadero o un cultivo en una zona seca.

Al obtener lecturas de humedad por debajo de un umbral establecido abrirá la llave de paso del aspersor correspondiente, ayudando así a subir la humedad hasta otro umbral indicado, o durante un tiempo preestablecido.

Igualmente transmitirá las lecturas obtenidas por sus sensores al gateway para que el usuario pueda conocerlas en tiempo real. Así mismo comunicará los momentos en los que se activa o desactiva un actuador por haber alcanzado los niveles de alerta establecidos.

Como se ha comentado anteriormente, se puede complementar su función con otros niveles de alerta de los parámetros medidos por sus sensores según se desee o se estime oportuno. Pero en todo caso será en el nodo donde estén programadas esas decisiones.

3.1.1 Sensores y actuadores

El nodo consta de un sensor de temperatura y humedad y estará conectado a la llave de paso de uno o varios aspersores cercanos. Idealmente el nodo se colocará junto al aspersor, pero en una zona ciega del mismo, para preservarlo del agua. Si bien el diseño del nodo debería ser estanco y soportar inclemencias meteorológicas, este aislamiento obedece más a la necesidad de no tergiversar las mediciones cuando el aspersor está abierto. Esto es porque en ese momento la humedad podría subir repentinamente si se moja, arrojando unas medidas irreales, lo que daría como resultado un funcionamiento intermitente del sistema de riego que seguramente no sea lo deseable.

Para esta función se puede optar por cualquiera de los sensores disponibles en el mercado, habiendo una enorme variedad Open Source que por un bajo coste ofrecen unos resultados precisos con poco margen de error. Se recomienda por ejemplo sensores digitales desarrollados por Sensirion[20] que cuentan con interfaz I2C[21].

Adicionalmente también puede incluir un sensor de humedad de la tierra, o contar únicamente con este sensor si se considera que los factores ambientales no son relevantes y sólo se desea controlar el estado de la tierra en sí. En este caso lo más habitual es un sensor capacitivo o resistivo con salida analógica, pero también pueden utilizarse los referidos anteriormente, montados en una estructura que permita insertarlos en la tierra.

Respecto al actuador, existen en el mercado algunos mecanismos de llaves de paso de agua o, directamente aspersores, preparados para funcionar como actuadores de dispositivos IoT.

---

[20] https://sensirion.com/products/product-categories/humidity/

[21] https://www.i2c-bus.org/

Por simplicidad, y con el objetivo de aprovechar una instalación de riego que seguramente ya exista, se puede optar por reemplazar el sistema de activación de ese sistema de riego por un relé controlado por el nodo mediante alguno de sus puertos GPIO.

Otra opción más avanzada pasaría por instalar un actuador que permita regular el nivel de flujo de agua que se permita pasar.

3.1.2 Alimentación

En principio el nodo puede ser ubicado en lugares donde difícilmente habrá conexión eléctrica, por lo que es necesario que dispongan de una batería y una fuente de alimentación renovable, como puede ser una placa solar o una dinamo conectada al circuito de agua, que permita recargar su batería cada vez que se active el actuador.

3.1.3 Firmware

El sistema operativo del nodo será el que contenga la información de comunicación con el *gateway*, así como los umbrales de activación y desactivación de los actuadores. Es conveniente que se puedan configurar estos 3 parámetros de forma remota y, adicionalmente, se puede valorar la opción de actualizar el firmware también de forma remota.

En este punto cobra importancia tener en consideración los requisitos de mantenimiento del nodo, señales que alerten de un funcionamiento errático o de alguna avería en el mismo, tanto en el hardware como en el software, por lo que sería recomendable que contase con actualizaciones OTA que permita realizar actualizaciones a distancia. Del mismo modo debe existir algún mecanismo de reactivación en caso de que un nodo pierda completamente la alimentación y se apague, para que pueda volver a encenderse sin necesidad de intervenir sobre él.

3.2 Gateway

Consistirá en un equipo o dispositivo frontera el cual recibirá los datos de todos los nodos que conforman el despliegue. Será también el encargado de mantener en marcha el servidor de comunicaciones. Por ejemplo, el *broker* MQTT, utilizado tanto por los nodos como por los clientes. Sería posible que albergará también una base de datos para mantener un histórico de las mediciones y actualizaciones, así como un panel tipo *dashboard* que permitiera visualizar las diferentes medidas en tiempo real.

Idealmente este equipo deberá mantenerse encendido y en línea, sin embargo, como la computación de las actualizaciones estará programada en los propios nodos, su caída o desconexión momentánea no resulta crítica. También puede implementar sistemas de detección de caídas de los nodos para alertar sobre ello y, si es posible, tratar de reconectar o levantar los nodos que estén fallando.

3.3 Clientes

De modo muy similar al *gateway* el cliente mostrará los datos que el *gateway* recopile de los nodos. La forma más sencilla sería mediante una simple suscripción al *topic MQTT*, o algún otro protocolo abierto de los listados en el Estado de la Cuestión, en el que los sensores envían sus mediciones al *broker* del *gateway*, pero también, si se ha instalado un *dashboard*, el cliente podrá conectarse al mismo y visualizar datos y realizar las mismas acciones y administración que desde el *gateway* con la diferencia de que no actúa también como servidor.

3.4 Comunicaciones

Mediante LoRa, WiFi o la infraestructura de red que se estime oportuna, y por MQTT, HTTP, AMQP o CoAP, el nodo podrá enviar a su *gateway* cada lectura obtenida de los sensores o simplemente los flags de activación de los actuadores, según se configure. Así como dar las últimas lecturas al ser solicitadas bajo demanda por el *gateway* o alguno de sus clientes.

Los nodos tienen capacidad para conectarse de forma autónoma a redes, si bien se puede hacer un despliegue en el que los nodos se conecten directamente a Internet mediante una red WiFi con salida al exterior, del mismo modo que el *gateway*,

o, por el contrario, se conecten a una red LAN, creada por el *gateway*, que puede ser por una tecnología más adecuada para las dimensiones del despliegue y la distancia de éste a los distintos nodos que lo conformen. En este caso la salida a Internet será proporcionada por el gateway para que puedan acceder los clientes externos.

3.5 Posible implementación Open Source

En el siguiente capítulo se abordará en profundidad la implementación realizada, si bien, en este punto y vistos los requisitos que se plantean, se propone una posible implementación de los mismos.

La implementación, completamente Open Source, constará de un firmware para el nodo basado en Zephyr RTOS, al cual se añadirán los proyectos de Eclipse IoT necesarios para las funcionalidades descritas, fundamentalmente para las comunicaciones.

Para las comunicaciones se sugiere el uso del protocolo MQTT por ser más escalable y ajustarse más a las necesidades del despliegue propuesto.

En cuanto al *gateway*, que deberá ser un equipo con Sistema Operativo abierto, como por ejemplo Ubuntu, correrán en él las herramientas Eclipse IoT pertinentes para el funcionamiento y control de los nodos. Fundamentalmente Eclipse Mosquitto, ya que actuará como *broker* al que los nodos publicarán los datos. Los clientes también utilizarán Eclipse Mosquitto para suscribirse al *broker* MQTT del *gategay*.

Capítulo 4 - Implementación y despliegue

Antes de comenzar con este capítulo es conveniente indicar que en el Apéndice A se explica detalladamente el procedimiento para crear el entorno de trabajo.

Partiendo de la base del Caso de uso descrito en el capítulo anterior, se pretende con esta implementación justificar el potencial de las plataformas y herramientas elegidas para su desarrollo e implementación, tanto de las que se presentan en funcionamiento como de las que se plantean como ampliaciones o trabajo futuro. Esta implementación y propuestas de herramientas para solucionar las diferentes necesidades del caso de uso pueden ser utilizadas como hilo conductor en el desarrollo de una solución para otros casos de uso con requisitos similares.

4.1 Aplicación del nodo

La aplicación para el nodo[22] se ha desarrollado utilizando El proyecto Zephyr y algunos de sus ejemplos para implementar la lectura de los datos obtenidos por los sensores, la activación de los actuadores y el envío de dichos datos al *broker* MQTT, así como todos los requisitos de red y adicionales que requiere.

Se desarrolla bajo los términos de la licencia Apache 2.0[23]. Se ha tomado esta decisión ya que el propio proyecto Zephyr usa esta licencia para lograr un equilibrio entre la contribución abierta y permitir usar el software. La licencia Apache 2.0 es una licencia permisiva de código abierto que permite usar, modificar, distribuir y vender libremente los productos propios que incluyen software con licencia Apache 2.0 [48].

Para su desarrollo se han utilizado como base los ejemplos de Zephyr MQTT Publisher[24] y SHT3XD: High accuracy digital I2C humidity sensor[25].

---

[22] https://bitbucket.org/tfmeue/nodo-iot

[23] https://www.apache.org/licenses/LICENSE-2.0

[24] https://docs.zephyrproject.org/3.0.0/samples/net/mqtt_publisher/README.html

[25] https://docs.zephyrproject.org/3.0.0/samples/sensor/sht3xd/README.html

Consta de tres partes. En primer lugar se establecen las configuraciones de los servicios que se van a utilizar y algunos parámetros necesarios para los mismos, en el fichero *prj.conf*, para el uso del bus I2C es necesario configurar la información de los sensores y sus direcciones en el fichero *esp32.overlay* dentro del directorio *boards*.

Mientras, el código principal del programa realiza un flujo que comienza inicializando la interfaz de red inalámbrica y esperando a que ésta se conecte. Una vez establecida la conexión, se enciende el LED correspondiente y comienza un buble infinito durante el cual el nodo está leyendo periódicamente los valores de los sensores y enviándolos al *broker* MQTT.

Cabe destacar que desde 2016, la implementación de MQTT en Zephyr se realiza mediante el uso del paquete de librerías del proyecto Eclipse Paho de Eclipse IoT.[26]

Una vez completada la aplicación es necesario construir los archivos binarios para la ESP32 y *flashearlos* en la placa. Esto se realiza con los siguientes comandos desde el directorio *zephyrproject/Zephyr*:

```
$ west build -p auto -b esp32 nodo-iot
$ west flash
```

Donde `-p auto` indica que se remplace cualquier aplicación previamente construida, `-b esp32` indica la placa para la que se quieren generar los binarios y `nodo-iot` es la carpeta donde se encuentra la aplicación.

4.1.1 Conexión con sensores y actuadores

La placa de desarrollo ESP32 utilizada consta de treinta y ocho pines, entre los cuales se encuentran uno de alimentación de 3.3V y otro de 5V, tres puertos de conexión a tierra, y 29 puertos de entrada salida GPIO que comparten también otras funciones como la de transmisión (TX) y recepción (RX) de la UART o la línea de reloj (SCL) y línea de datos (SDA) del bus I2C.

[26] https://github.com/zephyrproject-rtos/zephyr/issues/1847

Este último caso es el relevante a la hora de conectar los sensores de temperatura y humedad mediante I2C. En concreto se corresponden con los pines GPIO21 y GPIO22 (marcados en la placa como G21 y G22), que actúan como SDA y SCL respectivamente. [49]

Los sensores de temperatura y humedad SHT30 y SHT31 constan cada uno de cuatro pines. Dos de ellos de alimentación (VIN y GND) y otros dos los de conexión al bus I2C (SCL y SDA). El SHT30 cuenta con los 4 pines directamente en un peine listo para insertar en una placa de prototipado, el SHT31, por su parte, viene montado en una palca con un conector JST del que sale el cableado.

En la placa de prototipado se han montado los dos sensores en una misma fila, a la cual se añaden los cables que unen esa conexión con los puertos de alimentación y del bus I2C de la placa ESP32, tal y como se aprecia en la Figura 4-1.

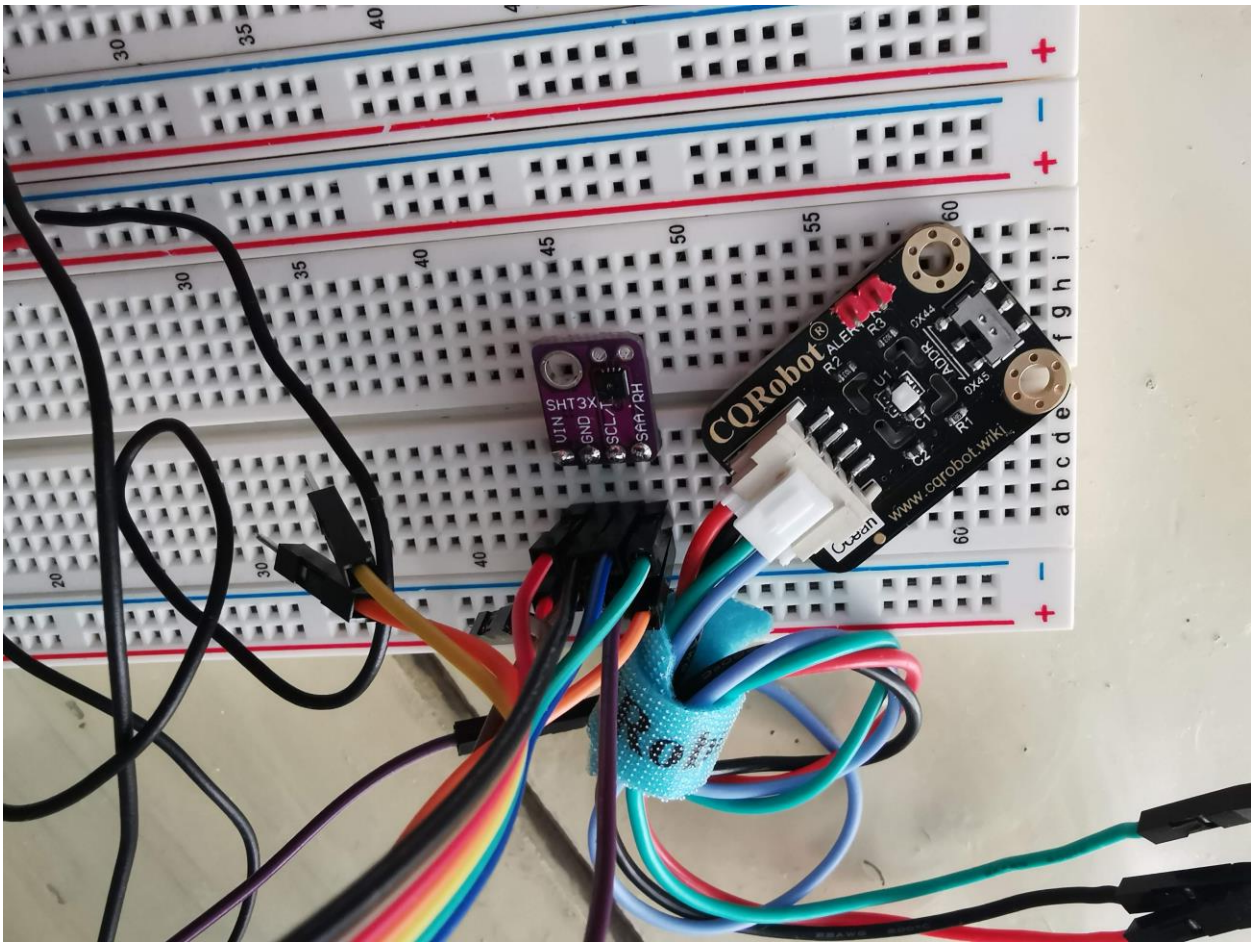

Figura 4-1: Fotografía de los sensores conectados en la placa de prototipado

La placa del sensor SHT31 cuenta también con un interruptor que permite configurar la dirección del dispositivo en el puerto I2C, entre la 0x44 y la 0x45, puesto que el sensor SHT30 viene configurado por defecto en la dirección 0x44, este interruptor debe situarse en el lado donde indica 0x45 para el correcto funcionamiento de ambos sensores.

Para el funcionamiento de estos sensores es necesario contar con el driver[27] de los mismos, el cual viene incluido en la versión del SDK de Zephyr utilizada.

Por otro lado, se utilizan tres LEDs, de los cuales uno actuará a modo de indicador de estado de la conexión WiFi y los otros dos simulan la respuesta de los actuadores. Para el funcionamiento de estos LEDs se han 3 puertos GPIO como puertos de salida, los cuales han sido, los puertos GPIO12, GPIO 13 y GPIO19 los elegidos para conectar los diodos.

Siguiendo estas indicaciones, el cátodo se conectará a un pin de tierra (GND) en los 3 casos, y el ánodo del LED de color verde, utilizado para indicar la conexión a un punto de acceso WiFi en el pin G12, mientras que los ánodos de los LEDs de color azul, utilizados para representar el estado del actuador, o lo que es lo mismo, el paso del agua o no, se conectará a los pines G13 y G19, cada uno. En la Figura 4-2 se pueden identificar los tres LEDs descritos conectados según se acaba de indicar, así como la conexión del bus I2D y la alimentación que llega a los sensores vistos anteriormente.

[27] https://esp-idf-lib.readthedocs.io/en/latest/groups/sht3x.html

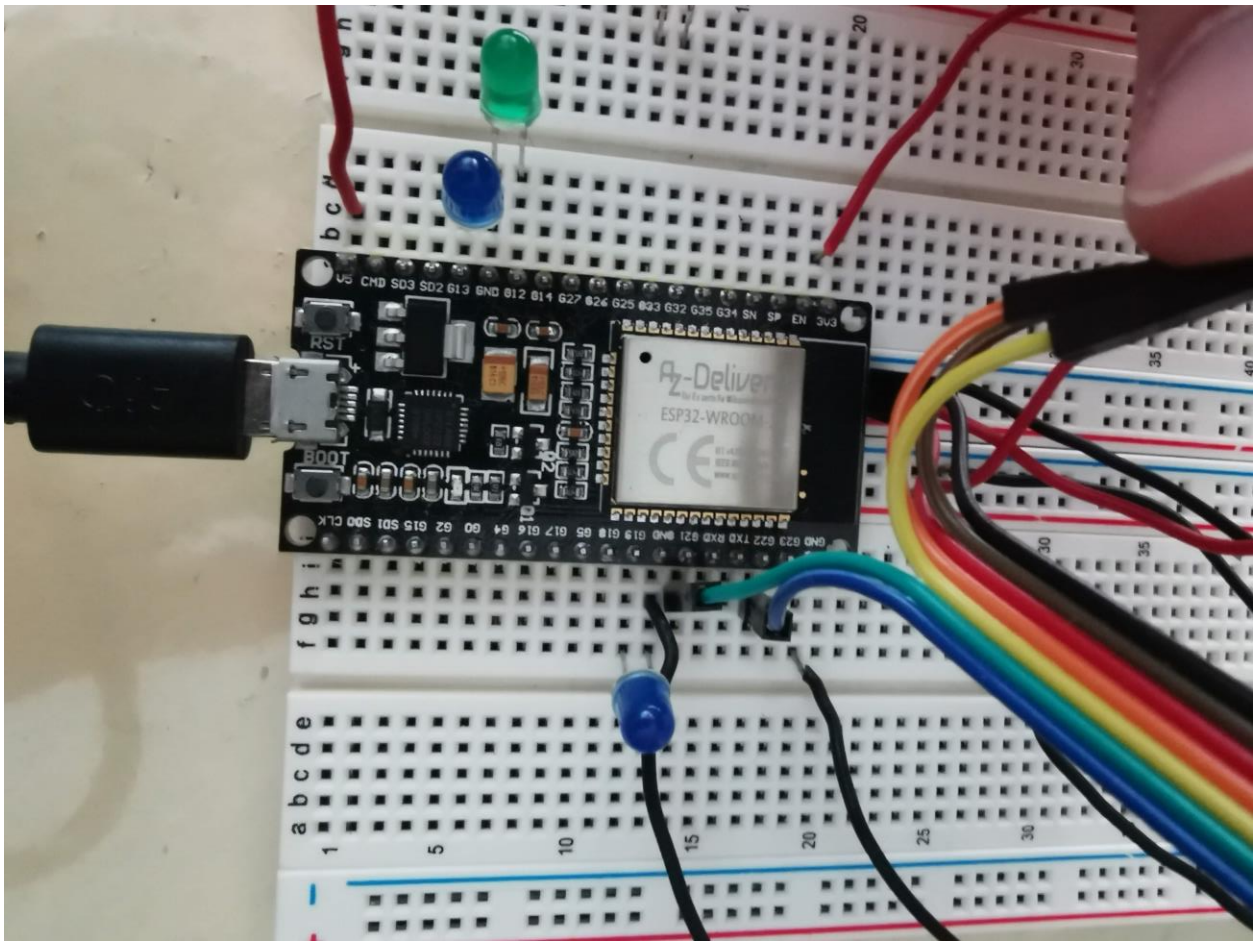

Figura 4-2. Fotografía de las conexiones a la ESP32

4.1.2 Digital twins

El uso de digital twins permite la creación de gemelos digitales de los nodos físicamente existentes. Estos pueden ser utilizados como interfaz para interactuar con los nodos reales a los que representan o para realizar simulaciones. En este sentido existe la posibilidad de realizar más de un gemelo digital del mismo nodo y simular en cada uno de ellos situaciones diferentes, incluso situaciones que son poco probables, especialmente cuando lo que se mide son magnitudes físicas como en este caso.

De este modo se pueden realizar pruebas de funcionamiento del código para comprobar que ante esas situaciones el comportamiento que se da en los actuadores es el esperado. Igualmente puede ser utilizado para realizar pruebas de estrés.

La creación de estos gemelos digitales se haría mediante Eclipse Ditto™, otro de los proyectos de Eclipse IoT que utiliza a su vez Eclipse Hono™ y Eclipse Mosquitto™.

4.1.3 Actualizaciones OTA

Las actualizaciones son un punto importante para garantizar la integridad y seguridad del nodo ante posibles vulneraciones que se descubran, así como para asegurar su durabilidad en el tiempo, manteniendo versiones del software, librerías etcétera actualizadas que estén en funcionamiento y no caigan en desuso.

La implementación de actualizaciones automáticas se puede realizar con un back-end independiente del dominio como es Eclipse hawkBit™. Ofrece una integración directa de dispositivos a través de HTTP o una API de administración de dispositivos que permite conectar dispositivos con diferentes protocolos. Se puede hacer uso de la interfaz gráfica de usuario y otros servicios pueden interactuar con hawkBit a través de la API de administración RESTful. [50]

4.2 El Gateway

En el entorno desarrollado el equipo que se ha utilizado para el desarrollo y para cargar el firmware en el nodo es el mismo que actúa como *Gateway*.

Concretamente lo único que se hace es mantener activo un broker MQTT. Para el correcto funcionamiento de este despliegue en redes LAN domésticas puede ser necesario realizar configuraciones en el *router* proporcionado por la operadora o proveedor de servicios de Internet (ISP), especialmente la apertura del puerto TCP 1883, que es el utilizado por MQTT por defecto y en el caso de esta implementación.

Para el funcionamiento vía internet de este servicio, y permitir que este broker tenga clientes conectados fuera de la red LAN también sería necesaria la configuración de redireccionamiento de puertos NATP correspondiente en el *router*.

La puesta en servicio del broker MQTT en el gateway se realiza mediante el siguiente comando en una terminal:

```
$ mosquitto -v -p 1883
```

Si hubiera algún problema, probablemente sea necesario reiniciar el servicio con:

```
$ sudo service mosquitto restart
```

Teniendo en cuenta que Eclipse Mosquitto™ es en sí mismo un proyecto de Eclipse IoT[28], utilizando un *broker* MQTT en el servidor se está implementando el uso de otro proyecto más de este conjunto de proyectos.

Con el fin de entregar un código usable y sin las dependencias expresadas, que pueden complicarse más con el uso de redes públicas o corporativas por las políticas de seguridad que incorporan, el que se entrega utiliza un servidor[29] que aloja un *broker* MQTT disponible públicamente. Este servidor es concretamente del proyecto Eclipse Mosquitto, que lo pone a disposición pública para realizar pruebas.

4.3 Los clientes

Para recibir los mensajes publicados en el *broker* MQTT por los nodos, cualquier equipo con acceso al servidor puede convertirse en cliente, como también lo son los nodos, y suscribirse al *topic* correspondiente.

En el caso de este despliegue el mismo equipo que actúa como Gateway se convierte en cliente en otra terminal introduciendo el siguiente comando:

```
$ mosquitto_sub -h test.mosquitto.org -p 1883 -t eue/tfm/sensors
```

Nuevamente se está utilizando el proyecto Eclipse Mosquitto™ de Eclipse IoT.

Del mismo modo, otros equipos de la misma red pueden convertirse en cliente utilizando comandos análogos según el sistema operativo, o cualquiera de las múltiples aplicaciones, incluso *on-line*, que existen para suscribirse a *brokers* MQTT, también para dispositivos móviles.

Dado que en el código proporcionado se utiliza un servidor de internet, cualquier dispositivo en el mundo puede convertirse en cliente por cualquiera de los métodos mencionados.

---

[28] https://projects.eclipse.org/projects/iot.mosquitto

[29] test.mosquitto.org

4.4 La red de comunicaciones

La placa ESP32 dispone de conectividad WiFi, por lo que se utiliza esta misma para conectarse a la red Wi-Fi local que le da, a su vez, conexión a internet.

Para indicarle a la ESP32 los datos de la red a la que debe conectarse, con Zephyr debe incluirse la información del SSID y contraseña en un fichero de configuración que, para el caso de ESP32 es específico y se encuentra en la ruta … del proyecto de la aplicación.

La conexión se realiza de manera automática al encenderse la placa, si bien se han incorporado una serie de comandos al inicio del código de la aplicación para mostrar en el *log* el estado de la interfaz de red y para activar el LED correspondiente una vez que se establece la conexión con el punto de acceso.

Como ya se ha comentado anteriormente, para el correcto funcionamiento de las comunicaciones puede ser necesario intervenir en la configuración de puertos o de redireccionamiento de puertos del *router* de la compañía telefónica, así como se recomienda el uso de un servicio de DNS dinámicas[30] que permita acceder a la IP pública de la red de forma intuitiva, con un dominio fácil de recordar, y que se actualice de forma automática en caso de que ésta cambie como suele ser habitual con las conexiones a Internet domésticas comercializadas en España.

Respecto al protocolo utilizado para el envío de los datos se ha utilizado MQTT, que es además uno de los protocolos más extendidos en IoT, por constar con comunicaciones M:N, frente a CoAP que utiliza comunicaciones 1:1. Esto es fundamental para el caso de uso que se plantea en el que es esperable que pueda haber uno o varios nodos comunicándose con el mismo servidor a la vez y con uno o varios clientes de forma simultánea. Además, esto facilita la escalabilidad del sistema permitiendo añadir nuevos nodos en cualquier momento de forma sencilla. Permite también implementar diferentes medidas de seguridad.

[30] https://www.noip.com/es-MX/remote-access

4.5 Funcionamiento

El comienzo de la rutina es automático una vez que se *flashea* el *firmware* desarrollado o también tras cada reinicio.

Como se ha visto previamente, lo primero que ocurre es que se activa la interfaz de red inalámbrica en busca del SSID configurado al cual intenta conectarse con la contraseña indicada. Este proceso es automático y mientras se realiza se muestra un log de estados por pantalla como se ve en la Figura 4-3. Se aprecia también que el tiempo de conexión es muy rápido, inferior a diez segundos.

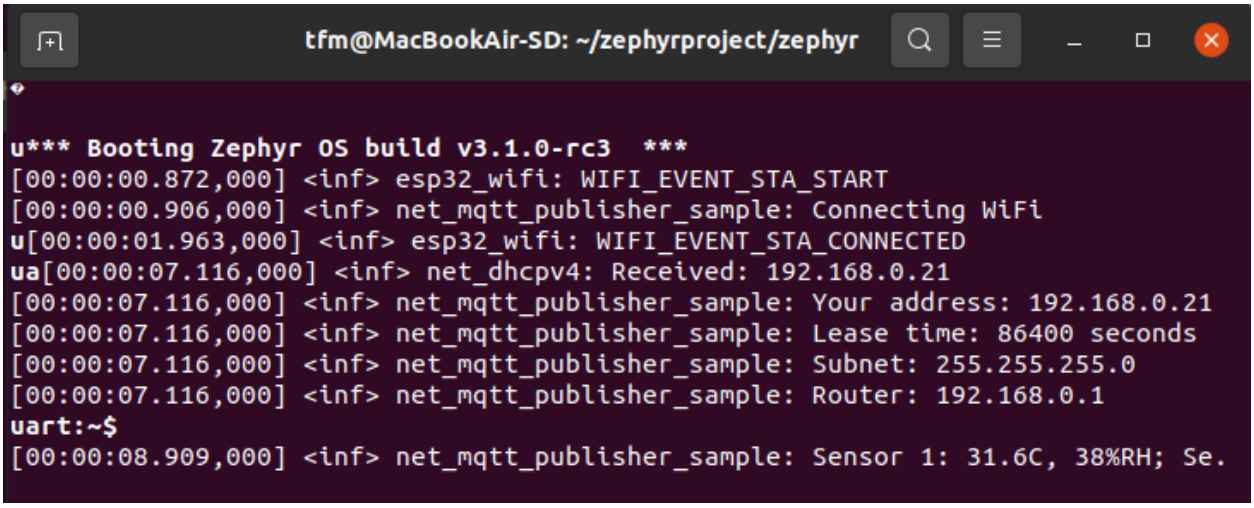

Figura 4-3. Captura de los mensajes de arranque de la ESP32

Esta captura se ha tomado desde una terminal del Gateway con la ESP32 conectada por USB y accediendo a su consola con el comanado

```
$ minicom -D /dev/ttyUSB0 -o
```

Para el cual es necesario disponer del minicom[31].

Inmediatamente después de conectarse a la red WiFi y obtener dirección IP por DHCP de la red local, el nodo comienza a realizar las lecturas de los sensores y transmitirlas por MQTT, como de hecho se ve en la Figura 4-3 que apenas dos segundos después de conectarse ya publica el primer mensaje en el *broker*.

[31] https://linux.die.net/man/1/minicom

```
[00:00:17.179,000] <inf> net_mqtt_publisher_sample: Sensor 1: 31.6C, 38%RH; Se.

[00:00:17.179,000] <inf> net_mqtt_publisher_sample: attempting to connect:
[00:00:17.179,000] <inf> net_mqtt_publisher_sample: try_to_connect: 0 <OK>
[00:00:17.180,000] <inf> net_mqtt_publisher_sample: mqtt_ping: 0 <OK>
[00:00:17.236,000] <inf> net_mqtt_publisher_sample: PINGRESP packet
[00:00:17.681,000] <inf> net_mqtt_publisher_sample: mqtt_publish: 0 <OK>
[00:00:23.183,000] <inf> net_mqtt_publisher_sample: Sensor 1: 32.2C, 55%RH; Se.

[00:00:23.183,000] <inf> net_mqtt_publisher_sample: attempting to connect:
[00:00:23.183,000] <inf> net_mqtt_publisher_sample: try_to_connect: 0 <OK>
[00:00:23.184,000] <inf> net_mqtt_publisher_sample: mqtt_ping: 0 <OK>
[00:00:23.260,000] <inf> net_mqtt_publisher_sample: PINGRESP packet
[00:00:23.685,000] <inf> net_mqtt_publisher_sample: mqtt_publish: 0 <OK>
[00:00:29.188,000] <inf> net_mqtt_publisher_sample: Sensor 1: 32.4C, 79%RH; Se.

[00:00:29.188,000] <inf> net_mqtt_publisher_sample: attempting to connect:
[00:00:29.188,000] <inf> net_mqtt_publisher_sample: try_to_connect: 0 <OK>
[00:00:29.188,000] <inf> net_mqtt_publisher_sample: mqtt_ping: 0 <OK>
[00:00:29.236,000] <inf> net_mqtt_publisher_sample: PINGRESP packet
[00:00:29.688,000] <inf> net_mqtt_publisher_sample: mqtt_publish: 0 <OK>
[00:00:35.191,000] <inf> net_mqtt_publisher_sample: Sensor 1: 32.0C, 50%RH; Se.

[00:00:35.191,000] <inf> net_mqtt_publisher_sample: attempting to connect:
```

Figura 4-4. Captura del funcionamiento normal del nodo

En la Figura 4-4 se observa cómo se va ejecutando en bucle cada cinco segundos el envío de lecturas de los sensores al *broker* MQTT. Paralelamente en la Figura 4-5 está la parte del cliente suscrito al topic y que va recibiendo las lecturas.

Se observa como las primeras lecturas son muy similares, pero en las últimas se ha forzado un aumento de la humedad relativa del Sensor 1 para comprobar cómo, en ese caso, cambia de estado el led correspondiente que se apaga al superar el umbral establecido.

```
Sensor 1: 31.6C, 38%RH; Sensor 2: 31.6C, 38%RH.

Sensor 1: 31.6C, 38%RH; Sensor 2: 31.6C, 38%RH.

Sensor 1: 31.7C, 38%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 31.7C, 38%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 31.6C, 38%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 31.6C, 38%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 31.6C, 38%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 32.2C, 55%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 32.4C, 79%RH; Sensor 2: 31.6C, 39%RH.

Sensor 1: 32.0C, 50%RH; Sensor 2: 31.6C, 39%RH.
```

Figura 4-5. Captura de los mensajes recibidos en un cliente MQTT

4.6 Dificultades encontradas

En un primer momento se planteó el uso de una placa NodeMCU con chip ESP8266 puesto que ya se disponía de ella. Sin embargo, en Zephpy el soporte para este procesador es únicamente para utilizarlo como *shield*, es decir, como placa de expansión o complementaria que se acople mediante sus pines a otra placa base ampliando sus capacidades [51] pero no se puede utilizar por sí sola. Por este motivo se tuvo que cambiar la placa utilizada, optando en esta ocasión por la sucesora de la ESP8266 y desarrollada por la misma empresa, la ESP32 que tanto se ha mencionado en esta memoria, y que sí cuenta con soporte[32].

Esto ralentizó durante un tiempo el desarrollo del proyecto ya que las primeras pruebas no fueron bien debido a este inconveniente con la ESP8266, con la consecuente confusión al no tener claro los motivos en un primer momento, y, posteriormente, por los tiempos de entrega de la ESP32 nueva que, afortunadamente no fueron muy dilatados gracias a que es una placa de uso muy extendido y con alta disponibilidad en varios proveedores que pueden servirla en cuestión de días.

La decisión de la palca ESP32 fue tomada en base a lo que se acaba de exponer, pero esto trajo más adelante un importante quebradero de cabeza, relacionado también con las decisiones tomadas en cuanto a los sensores a utilizar.

Inicialmente el caso de uso se planteó como se cuenta en el Capítulo 3 - , pero a la hora de empezar a desarrollarlo y de adquirir los componentes necesarios se pensó que sería más interesante medir la humedad relativa de la tierra en lugar del aire. Para ello se optó por un sensor capacitivo analógico distribuido por AZ-Delivery. El dispositivo tiene una interfaz PH 2.0 - 3P para realizar las conexiones y tiene unas dimensiones de 100 mm x 22.5 mm x 9.5 mm. Ésta se inserta en el suelo como muestra la siguiente figura, que sirve también para ilustrar la placa en cuestión. [52]

[32] https://docs.zephyrproject.org/2.7.0/boards/xtensa/esp32/doc/index.html

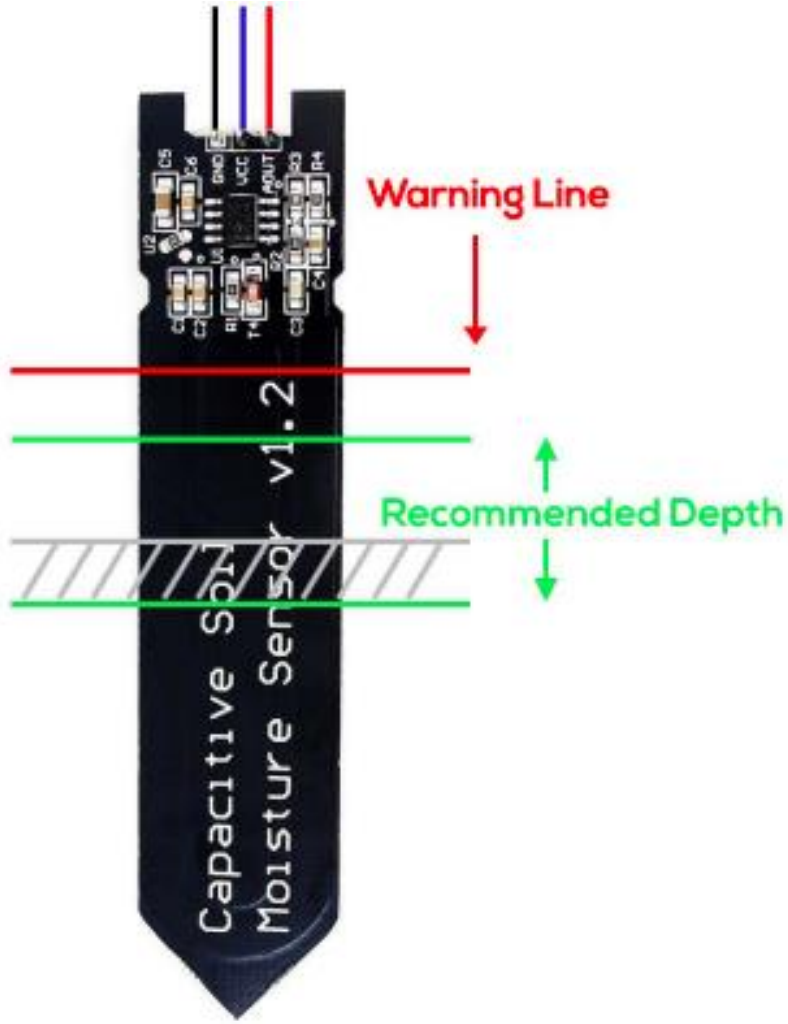

Figura 4-6. Módulo higrómetro capacitivo analógico. [52]

Sin embargo, tras muchos intentos por sacar adelante la aplicación que debería funcionar para leer la entrada analógica y la cual se llegó a completar pero sin conseguir compilarla para la placa ESP32, se descubrió que la compatibilidad de la conversión analógico-digital (ADC) de la ESP32, necesaria para obtener las lecturas de este sensor, aun no estaba implementada en Zephyr[33]. Esto obligó a descartar el uso de este sensor y utilizar otros que funcionasen por interfaces sí implementadas en Zephyr, con el consecuente cambio en toda la aplicación que se había desarrollado y la desmotivación que supone enfrentarse a una dificultad de estas.

[33] https://github.com/zephyrproject-rtos/zephyr/issues/44908

No obstante, se entrega también el código[34] que se desarrolló para el uso del sensor analógico, pues este debería ser funcional en otras placas compatibles o en la propia ESP32 cuando se implemente la compatibilidad ADC. Sin embargo se debe advertir de que dicho código no ha sido testeado por los motivos expuestos, si bien se considera relevante entregarlo como prueba del trabajo realizado en esta línea aunque finalmente no fuera fructífero. Este código fue inspirado por el código del artículo referenciado en [53] que expone un caso de uso similar al propuesto pero más completo.

Se optó entonces por volver al planteamiento original del caso de uso y utilizar sensores de temperatura y humedad (termohigrómetros) que sí tuvieran compatibilidad con Zephyr y utilizasen una interfaz disponible en la ESP32. Para asegurar este punto se consultó la lista de ejemplos[35] disponibles para Zephyr, entre los que se encontraban el SHT3XD y el SHT4X. También se contempló el SHT2X, pero los drivers para esa versión no estaban en el repositorio Zephyr y no se consiguieron encontrar.

El problema ahora estaba en conseguir que llegasen a tiempo estos sensores y ponerlos en funcionamiento. De hecho, el primer sensor que llegó lo hizo más tarde de lo previsto y resultó estar defectuoso. Se decidió entonces adquirir dos más, dando como resultado una mejora inesperada en el desarrollo ya que el mismo nodo ahora es capaz de leer los datos de dos sensores y no solo de uno.

Todas estas dificultades impidieron continuar con el desarrollo de la aplicación, y más en concreto con los proyectos Eclipse IoT con los que se pretendían implementar los *digital twins* y las actualizaciones OTA, así como, si hubiera quedado tiempo, continuar implementando otros proyectos de Eclipse IoT en la plataforma.

---

[34] https://bitbucket.org/tfmeue/analogico/

[35] https://docs.zephyrproject.org/3.0.0/samples/sensor/sensor.html

Capítulo 5 - Conclusiones y trabajo futuro

La primera y más evidente conclusión de este Trabajo Fin de Máster es que, tal y como se proponía demostrar, es que se puede diseñar y desplegar una plataforma de Internet de las Cosas completa utilizando únicamente componentes Open Source.

No solo en el caso del despliegue realizado para el caso de uso que aquí se proponía se basa en componentes Open Source, sino que, además, la mayoría de las herramientas *softwate* para el IoT que se han encontrado y explicado en el Estado de la cuestión son Open Source. Incluso las soluciones software que presentan empresas privadas para el uso de sus servicios, ya sea con su hardware propio o con otro hardware, implementan componentes software Open Source dentro de sus plataformas.

En cuanto al hardware también se ha presentado una solución completamente abierta que permite crear los dispositivos a demanda de forma además muy asequible económicamente hablando, como es Arduino. Para el desarrollo se ha utilizado ESP32 por simplicidad y por ser una alternativa perfectamente válida para el caso de uso, si bien es cierto que este procesador no es Open Hardware al no ser abiertas sus especificaciones de bajo nivel.

También es importante concluir que el desarrollo de una plataforma IoT no es algo trivial, como demuestran las dificultades encontradas que se han expuesto. Si bien es cierto que existen multitud de componentes software y hardware y que la compatibilidad entre ellos, especialmente los de uso más extendido, es muy amplia, aún hay muchas incompatibilidades a tener en cuenta y que conviene prever a la hora de hacer un análisis de requisitos previo a la implementación de una solución y a la toma de la decisión de los componentes a utilizar. Esto influye también en el caso de la infraestructura y tecnología de comunicaciones ya que todas tienen sus limitaciones en unos u otros aspectos y hay que buscar la más óptima para el despliegue que se pretendan realizar y las condiciones del mismo, sobre todo en lo que a ubicación de los nodos se refiere.

Para continuar con el trabajo en el futuro surgen tres formas alternativas. Por un lado, se puede continuar el desarrollo presentado ampliando las funcionalidades que tiene o presentando unas nuevas. Una de las primeras partes podría ser implementar mejores políticas de seguridad en las transmisiones o cambiar la infraestructura de comunicaciones a una que no requiera de un punto de acceso WiFi. Para la implementación de esas mejoras sin duda se pueden utilizar algunos de los proyectos Eclipse IoT que se hayan descrito o no, se han quedado sin utilizar en esta ocasión. Es por ejemplo el caso de Eclipse Ditto para los gemelos digitales que permitan hacer simulaciones de las condiciones físicas del entorno, así como Eclipse hawkBit para dotar a los nodos de actualizaciones automáticas.

En esta línea también se puede trabajar en el lado del usuario, creando una aplicación, o *front-end*, más amigable, tanto para la visualización de los datos como para la gestión y administración del *gateway* y de los nodos. Incluso si se considera que los datos pueden ser de utilidad para el público general se pueden distribuir de forma abierta.

Otra alternativa, que puede ser simultánea o también continuación de la anterior, es exportar esta implementación a otro caso de uso de similares características, pero quizás con otros sensores y actuadores u otra finalidad. Aunque más rápido incluso sería aprovechar el despliegue tal y como se plantea añadiendo otros escenarios o casos de uso, o ampliarlo solo con los nuevos sensores o actuadores que se necesiten, pero aprovechando también los ya existentes y sin perder la funcionalidad que ya tiene.

Finalmente, y si se obtiene la experiencia suficiente, quizás sea posible, a la vez que enriquecedor, participar en la comunidad de desarrolladores, tanto para Zephyr como para los proyectos de Eclipse IoT, incluso ayudando a desarrollar los proyectos que se encuentra en modo incubación, en este segundo caso, o en el caso de Zephyr, participando en el *issue* que pretende añadir compatibilidad ADC para la ESP32, que ha supuesto un escollo en el desarrollo de este Trabajo Fin de Máster y que, a su vez, abriría la puerta a, como trabajo futuro, desarrollar el caso de uso con ese sensor analógico de humedad en la tierra, tal y como estaba previsto.

Chapter 1 - Introduction

Since its birth during the Cold War as a military communication tool, the Internet has evolved at a dizzying pace, revolutionizing world life as it was previously known. You only must see Figure 1-1 to realize how in recent years the growth of equipment connected to the Internet has been exponential, even more so considering that it only covers up to 2002, 20 years ago. [1] [2]

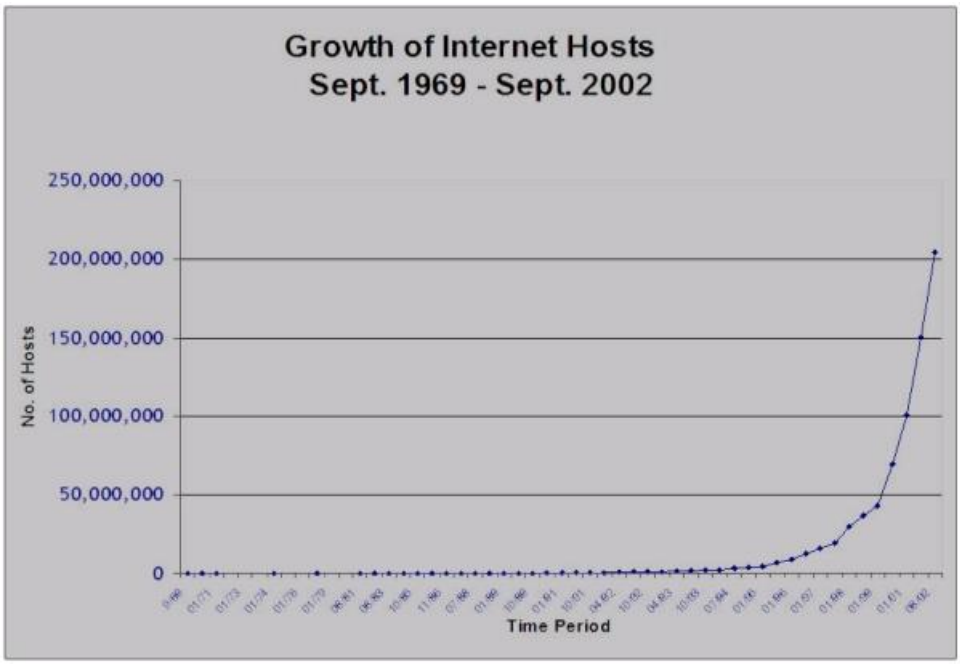

Figure 1-1 Graph of evolution of host connected to the Internet [2]

The notable change in the upward trend at the end of the graph makes sense, since it was in 1982 when the United States Department of Defense used the TCP/IP protocol in its Arpanet network, thus creating the Arpa Internet network, or International Net, which later it would remain with the name of Internet.

More recently, in 1999, the term "Internet of Things" began to be used, better known as IoT, for its acronym in English (Internet Of Things). It was the British pioneer Kevin Ashton who began to use it to describe a system in which objects in the physical world could be connected to the Internet through sensors. [3] [4] Nowadays it is difficult to think that something, any everyday object to which functions can be assigned, is not connected to the internet to be able to automate, program or activate and deactivate

those functions remotely. Contrary to a few years ago, not so many, when it was unthinkable that even some slippers could have a chip in the sole, or that electrical appliances such as a refrigerator, or above all the air conditioning equipment could be turned on before arriving home. [5]

The definition of IoT could be the grouping and interconnection of devices and objects through a network (can be private or the Internet, the network of networks), where all of them could be visible and interact. Regarding the type of objects or devices, they can be anything, from sensors and mechanical devices to everyday objects such as the refrigerator, footwear or clothing. [6]

1.1 Motivation

The growing use of the Internet in things of daily use and the proliferation of different platforms, most of them private, to create the ecosystem that allows making sense of these objects having the Internet, motivates the research object of this work, which aims to demonstrate how it is possible to accommodate all the typical functions of an IoT platform using only open devices and free software, that is, with Open Source components, with the advantages that this entails.

The compatibility and scalability of the platforms is another motivation, since many of the Internet objects developed by companies use proprietary software, non-standard communication protocols or encryption keys in their communications that they do not provide to their users. It is even the case that, even using standard protocols, the use of the device's antennas is limited only to manufacturer's applications, as is the case of the NFC antenna of Apple's iPhone devices, an issue that by the way is in the spotlight from the European Commission for lack of competence [7]. With the use of Open Source tools these problems do not exist, although there may be some incompatibility problem, it is evident that it is much easier to communicate devices from different manufacturers with each other, through a network that can also be from a different manufacturer, if all of them use standard IoT communication technologies, such as NB-IoT and LoRa, but also WiFi and Bluetooth, and standard and open protocols, such as MQTT. Some of these concepts are reviewed in the *Capítulo 2 - Estado de la cuestión*.

1.2 Objetives

The main objective of the project is to carry out a detailed investigation of the feasibility of using a complete IoT solution based on free software. To do this, first of all, the concept of IoT framework/infrastructure/deployment must be defined as specifically as possible, identifying its main elements (both software and hardware), the requirements and limitations expected in each of them, cases of use that illustrates the interaction between them, etc. For each of these actors, the second part of the study will include a review of the state of the topic (alternatives) existing in the market, with special emphasis on available free software solutions. Its maturity, functionality, documentation, real characteristics, support, etc. will be valued. That study will conclude with the selection of the free software product that is considered most suitable. The last part of the work will consist of the actual implementation (either in a real hardware environment, or in a virtualized environment via Virtual Machines, containers, etc.) of the complete environment based on the solutions analyzed, illustrating their strengths and weaknesses based on real use cases.

1.3 Work plan

In order to carry out this project, a work plan was established in the minimum agreement which, although it is true that its beginning has been delayed over time for various reasons, has been carried out more or less reliably since the moment when its elaboration really and effectively started.

In the deal reached between the director and the student for the realization of this Master's Thesis, the following list of minimum tasks was established to present it.

1) Study of the state of the art, collection of information and documentation related to complete IoT environments/frameworks. It is suggested to emphasize all the projects available in the Eclipse IoT framework at higher levels and Zephyr at the node level, since ideally it would be interesting that the proposed solution is based on their components.
2) Work division into main elements that make up an IoT framework according to the study completed in 1). At least, solutions should be addressed at the level of:

a) Sensor node. The fundamental part of any IoT deployment is the device (or thing) in charge of taking the desired measurements and, if required, activating the corresponding actuators.

b) Gateway. It is the device at the edge of the network, or border device, usually one per IoT deployment, which communicates with all the nodes of the deployment, either over the Internet or over a local network, in which case it will also serve connection of the nodes with the internet. It can also be used to display services commonly used by devices, usually a control panel or dashboard.

c) Cloud. The cloud where the necessary computation is carried out from the data obtained by the nodes, as well as data control and node management services, among other things. This cloud can often be at the gateway itself, on remote internet servers, or contract to companies specialized in Cloud IoT services.

d) Communication technologies. Explore the different alternatives to interconnect the nodes with each other or with their gateway and with the Internet, as well as the network and communications protocols.

e) High-level protocols. Relate communication technologies with the corresponding protocols above the transport level.

f) Operating System. Both for the nodes, explore RTOS (Real Time Operating Systems) to develop their firmware, and for the gateway and the client, identify the best type of operating system for the desired purpose. Ensure that the client side is cross-platform.

3) For each element identified in 2), it will be necessary to document, study and put it into operation, identifying its main characteristics, strengths and weaknesses from the points of view that the student considers interesting.

4) Based on the above elements, a fully functional environment based exclusively on *free software* will be developed and implemented, including real use cases that illustrate the strengths/weaknesses of each component (or of the environment as a whole).

5) Documentation and memory writing.

It was also agreed to establish a monitoring system to check the progress of the work. Periodically, the student had to give a report of the work done during the corresponding period, the problems encountered and the work to be done during the following period will be agreed. This communication may be through a face-to-face/online meeting (especially during the first weeks), but in any case, through a collaborative editing tool of the student's choice (Google Drive, Trello...)

The modifications made are the result of the needs that have been found during the development of the project and fundamentally affect the order and time dedicated to each stage. Additionally, a logbook has been kept, of which a summary is presented in *Apéndice B* - , in which the progress has been noted, fundamentally with the aim of presenting it in each of the meetings that have been held in which the next steps to follow were also set. This makes it easier to assess the degree of compliance with the work plan.

Regarding the meetings, an attempt has been made to maintain a certain periodicity, depending above all on the dedication to the project at each time and the milestones that were being completed. In any case, in the longest periods, follow-up has continued via email. Table X presents a schematic schedule with the dates on which a meeting was held, a milestone was completed, a difficulty was encountered or, in short, important decisions were made for the course of the project.

Date	Milestone	Description
23-03-21	Start	Realization of the TFM begins.
07-04-21	Planification	Se establece el plan de trabajo.
25-10-21	Completed task	Preliminary presentation on the state of the art.
10-11-21	Email	Simple question messages and task follow-up.
13-12-21	Meeting	A calendar is set, continue with the study and prepare a use case with the platforms seen.
12-01-22	Meeting	Follow-up of tasks and simple doubts.
14-02-22	Completed task	Use case.
17-02-22	Meeting	Use case presentation.
03-03-22	Meeting	Follow-up of tasks and resolution of doubts.
14-03-22	Meeting	Follow-up of tasks and resolution of doubts.
25-03-22	Meeting	Follow-up of tasks and resolution of doubts.
20-04-22	Start task	Report begins to be written.
08-05-22	State task	Some epigraphs of the report are completed.
30-05-22	Important email	Update on the situation and analysis of various difficulties encountered in recent weeks. Other doubts and organizational issues are also discussed in the thread.
31-05-22	Meeting	Follow-up of tasks and resolution of doubts.
10-06-22	High difficulty	ESP32 ADC incompatibility discovered for ESP32 with Zephyr.
13-06-22	Difficulty solved	This is solved by changing the sensors used.
16-06-22	Completed task	The application code is completed.
17-06-22	Draft of the report	Draft of the report in shared document.
24-06-22	Handover	Se realiza la entrega de la memoria.

Table 1-1. Work plan timeline

Chapter 5 - Conclusions and future work

The first and most obvious conclusion of this Master Thesis is that, as it was proposed to demonstrate, it is possible to design and deploy a complete Internet of Things platform using only Open Source components.

Not only is the use case proposed in this thesis based on Open source components, but also most of the IoT software tools that have been found and explained in the State of the Art are Open Source. Even software solutions presented by private companies for the use of their services, either with their own hardware or with third party hardware, implement Open Source software components within their platforms.

In terms of hardware, a complete Open Source option has been presented in this thesis. That solution allows the creation of devices on demand in a very affordable way, such as Arduino. The ESP32 has been chosen for the development due to its simplicity and because it is a perfectly valid alternative for this specific use case. Although it is true that this processor is not Open Hardware because its low level specifications are not open.

It is also important to conclude that the development of an IoT platform is not trivial, as shown by the difficulties encountered. While it is true that there are a multitude of software and hardware components and that the compatibility between them, especially the most widely used ones, is very hide. There are still many incompatibilities to consider and that should be foreseen when doing a requirements analysis prior to implementing a solution and making the decision on the components to be used. This also influences the case of communications infrastructure and technology, since they all have their limitations in some aspects and it is necessary to look for the most optimal one for the deployment to be carried out and its conditions, especially as far as the location of the nodes is concerned.

There are three alternative ways to continue the work in the future. On the one hand, the presented development can be continued by extending the existing functionalities or introducing new ones. One of the first parts could be to implement better security policies in transmissions or change the communications infrastructure to

one that does not require a WiFi access point. For the implementation of these improvements you can certainly use some of the Eclipse IoT projects that have been described or not, have been left unused on this occasion. This is for example the case of Eclipse Ditto for digital twins that allow simulations of the physical conditions of the environment as well as Eclipse hawkBit to provide nodes with automatic updates.

In this line, it is also possible to work on the user side, creating a more user-friendly application, or front-end, both for the visualization of the data and for the management and administration of the gateway and the nodes. Even if it is considered that the data can be of use to the general public, it can be distributed openly.

Another alternative, which can be simultaneous or also a continuation of the previous one, is to export this implementation to another use case with similar characteristics, but perhaps with other sensors and actuators or another purpose. Although even faster would be to take advantage of the deployment as it is proposed by adding other scenarios or use cases, or to extend it only with the new sensors or actuators that are needed, but also taking advantage of the existing ones and without losing the functionality it already has.

Finally, and if enough experience is gained, it may be possible, as well as enriching, to participate in the developer community, both for Zephyr and Eclipse IoT projects, even helping to develop projects that are in incubation mode, in the latter case, or in the case of Zephyr, participating in the issue that aims to add ADC compatibility for the ESP32, which has been a stumbling block in the development of this Master Thesis and, in turn, would open the door to, as future work, develop the use case with this analog humidity sensor in the ground, as planned.

BIBLIOGRAFÍA

[1] L. Bahillo, «Historia de Internet: cómo nació y cuál fue su evolución,» marketing4ecommerce, 16 mayo 2022. [En línea]. Available: https://marketing4ecommerce.net/historia-de-internet/. [Último acceso: 20 mayo 2022].

[2] Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya), «Retro Informática. El pasado del futuro,» marzo 2008. [En línea]. Available: https://www.fib.upc.edu/retro-informatica/historia/internet.html. [Último acceso: 20 mayo 2022].

[3] J. Elder, «Cómo Kevin Ashton nombró El Internet de las Cosas,» 2 septiembre 2019. [En línea]. Available: https://blog.avast.com/es/kevin-ashton-named-the-internet-of-things.

[4] J. Normaln, «Kevin Ashton Invents the Term "The Internet of Things",» 1999. [En línea]. Available: https://www.historyofinformation.com/detail.php?id=3411. [Último acceso: junio 2022].

[5] K. Rose, S. Eldridge y L. Chapin, La Internet de las cosas - Una breve reseña, Internet Society (ISOC), 2015.

[6] Deloitte España; María Gracia, «¿Qué es IoT (Internet Of Things)?,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html. [Último acceso: 20 Mayo 2022].

[7] European Commission, "Antitrust: Commission sends Statement of Objections to Apple over practices regarding Apple Pay," 2 May 2022. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/ip_22_2764. [Accessed 31 May 2022].

[8] Bosch Software Innovations GmbH | INST/MKC2, *Introduction to Bosch IoT Suite for Computer Science Students,* 2018.

[9] Bosch.IO GmbH, «Bosch IoT Suite - A toolbox in the cloud for IoT developers,» 2022. [En línea]. Available: https://bosch-iot-suite.com/. [Último acceso: junio 2022].

[10] V. Denner, «Why a Bosch IoT Cloud?,» [En línea]. Available: https://blog.bosch-si.com/digital-transformation/why-a-bosch-iot-cloud/. [Último acceso: junio 2022].

[11] Y. Fernández, «Qué es Arduino, cómo funciona y qué puedes hacer con uno,» 3 agosto 2020. [En línea]. Available: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno. [Último acceso: 11 junio 2022].

[12] Arduino, «What is Arduino?,» 5 February 2018. [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction. [Último acceso: junio 2022].

[13] Raspberry Pi, «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.com. [Último acceso: junio 2022].

[14] Raspberry Pi, «Raspberry Pi for home,» [En línea]. Available: https://www.raspberrypi.com/for-home/. [Último acceso: junio 2022].

[15] Raspberry Pi, «Raspberry Pi OS,» [En línea]. Available: https://www.raspberrypi.com/software/. [Último acceso: junio 2022].

[16] The Eclipse Foundation, «Eclipse IoT,» [En línea]. Available: https://projects.eclipse.org/projects/iot. [Último acceso: junio 2022].

[17] The Eclipse Foundation, «Eclipse Leshan,» 2015. [En línea]. Available: https://www.eclipse.org/leshan/. [Último acceso: 2022].

[18] The Eclipse Foundation, «Eclipse Kura,» [En línea]. Available: https://www.eclipse.org/kura/. [Último acceso: 2022].

[19] The Eclipse Foundatio, «Eclipse Ditto,» 21 Apr 2022. [En línea]. Available: https://www.eclipse.org/ditto/. [Último acceso: junio 2022].

[20] The Eclipse Foundation, «Eclipse Hono,» 2019. [En línea]. Available: https://www.eclipse.org/hono/. [Último acceso: junio 2022].

[21] The Eclipse Foundation, «Eclipse Kapua,» 2016. [En línea]. Available: https://www.eclipse.org/kapua/. [Último acceso: junio 2022].

[22] The Eclipse Foundation, «Eclipse Mosquitto,» 2021. [En línea]. Available: https://projects.eclipse.org/projects/iot.mosquitto. [Último acceso: junio 2022].

[23] The Eclipse Foundation, «Eclipse Californium,» [En línea]. Available: https://www.eclipse.org/californium/. [Último acceso: junio 2022].

[24] The Linux Foundation, «Zephyr Project,» [En línea]. Available: https://www.zephyrproject.org/. [Último acceso: junio 2022].

[25] QEMU, «QEMU,» 2020. [En línea]. Available: https://wiki.qemu.org/Main_Page. [Último acceso: junio 2022].

[26] Zephyr Project, «West (Zephyr's meta-tool),» [En línea]. Available: https://docs.zephyrproject.org/latest/develop/west/index.html. [Último acceso: junio 2022].

[27] Amazon Web Services, Inc., «What is AWS IoT Greengrass?,» 2022. [En línea]. Available: https://docs.aws.amazon.com/greengrass/v2/developerguide/what-is-iot-greengrass.html. [Último acceso: 2022].

[28] Microsoft, «IoT Hub | Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/services/iot-hub/.

[29] Microsoft, «Digital Twins | Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/services/digital-twins/. [Último acceso: junio 2022].

[30] Microsoft, «Azure IoT Central,» [En línea]. Available: https://azure.microsoft.com/es-es/services/iot-central/. [Último acceso: junio 2022].

[31] Microsoft, «IoT Edge | Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/services/iot-edge/. [Último acceso: junio 2022].

[32] Microsoft, «Windows para IoT,» [En línea]. Available: https://azure.microsoft.com/es-es/services/windows-iot/. [Último acceso: junio 2022].

[33] Microsoft, «RTOS | Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/services/rtos/#overview. [Último acceso: junio 2022].

[34] Google, «Documentación de Google Cloud IoT Core,» [En línea]. Available: https://cloud.google.com/iot/docs?hl=es#docs. [Último acceso: junio 2022].

[35] Google, «Cloud IoT Core,» [En línea]. Available: https://cloud.google.com/iot-core. [Último acceso: junio 2022].

[36] ESPRESSIF SYSTEMS, «ESP32 Wi-Fi & Bluetooth MCU,» [En línea]. Available: https://www.espressif.com/en/products/socs. [Último acceso: junio 2022].

[37] STMicroelectronics, «Nucleo-L073RZ - STM32,» [En línea]. Available: https://www.st.com/en/evaluation-tools/nucleo-l073rz.html. [Último acceso: junio 2022].

[38] Mouser Electronics, Inc., «SensorTag Kits - TI,» [En línea]. Available: https://www.mouser.es/new/texas-instruments/ti-sensor-tag-kits. [Último acceso: junio 2022].

[39] B. X. Chen, «¿Qué es un Apple AirTag? Un sencillo rastreador con tecnología de punta,» 28 April 2021. [En línea]. Available: https://www.nytimes.com/2021/04/28/technology/personaltech/apple-airtag-review-tile.html. [Último acceso: junio 2022].

[40] P. Basappa, «The technology behind Apple's AirTag,» 16 Jul 2021. [En línea]. Available: https://medium.com/nerd-for-tech/the-technology-behind-apples-airtag-c7983f9322b5. [Último acceso: junio 2022].

[41] Bionix Supply Chain Technologies, «¿Qué es la tecnología RFID y cómo funciona?,» [En línea]. Available: https://bionixtechnologies.com/tecnologia-rfid/. [Último acceso: junio 2022].

[42] J. MACHUGH, «Attention, Shoppers: You Can Now Speed Straight Through Checkout Lines!,» 1 Jul 2004. [En línea]. Available: https://www.wired.com/2004/07/shoppers/. [Último acceso: junio 2022].

[43] MQTT, «MQTT: The Standard for IoT Messaging,» [En línea]. Available: https://mqtt.org/. [Último acceso: junio 2022].

[44] Pick Data, «MQTT vs CoAP, la batalla por ser el mejor protocolo IoT,» 21 Oct 2019. [En línea]. Available: https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot. [Último acceso: junio 2022].

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee, «Hypertext Transfer Protocol -- HTTP/1.1,» The Internet Society, 1999.

[46] S. Vinoski, «Advanced Message Queuing Protocol,» *IEEE Internet Computing,* vol. 10, nº 6, pp. 87-89, 2006.

[47] OASIS, «AMQP is the Internet Protocol for Business Messaging,» [En línea]. Available: https://www.amqp.org/about/what. [Último acceso: junio 2022].

[48] Zephyr Project, «Contribution Guidelines,» [En línea]. Available: https://docs.zephyrproject.org/latest/contribute/guidelines.html. [Último acceso: junio 2022].

[49] RandomNerdTutorials.com, «ESP32 I2C Communication: Set Pins, Multiple Bus Interfaces and Peripherals (Arduino IDE),» [En línea]. Available: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/. [Último acceso: junio 2022].

[50] Eclipse hawkBit™, «Eclipse hawkBit,» 2021. [En línea]. Available: https://www.eclipse.org/hawkbit/. [Último acceso: junio 2022].

[51] Zephyr Project, «Shields - Zephyr Project Documentation,» [En línea]. Available: https://docs.zephyrproject.org/latest/hardware/porting/shields.html. [Último acceso: junio 2022].

[52] AZ-Delivery Vertriebs GmbH, «Hygrometer Modul V1.2 Datenblatt».

[53] C. Hirsch, E. Bartocci y R. Grosu, «Capacitive Soil Moisture Sensor Node for IoT in Agriculture and Home,» de *IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, 2019.

[54] JIMBLOM, «ESP32 Thing Hookup Guide,» 7 March 2019. [En línea]. Available: https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide/resources--going-further. [Último acceso: junio 2022].

APÉNDICES

Apéndice A - Preparación del entorno

En la guía de inicio de la web de documentación de Zephyr Project[36] vienen detallados los primeros pasos para instalar el entorno de programación Zephyr y las dependencias necesarias tanto para sistemas operativos Linux como MacOS y como Windows. Para el desarrollo de este proyecto se ha utilizado la distribución de Linux Ubuntu versión 20.04.4 LTS de 64 bits, instalada directamente en una unidad externa de arranque de 256GB, conectada a un MacBook Air 13" de 2013 con procesador Intel Core i7 4650U con 4 núcleos a 1,70 GHz y 8 GB de memoria RAM. Del mismo modo, se ha utilizado la versión 3.0.0 de Zephyr, por lo que la información de este apéndice se centrará en las características y versiones indicadas, pudiendo variar en otras versiones.

Instalación del Sistema Operativo

Habitualmente este paso ya se habrá realizado en la mayoría de las ocasiones, sin embargo, antes de comenzar un desarrollo nuevo suele ser conveniente utilizar una máquina y un Sistema Operativo cuyo único fin vaya a ser este, en lugar de un equipo de uso más habitual y diverso. En estos casos se suele optar por la instalación de un Sistema Operativo limpio en una máquina virtual, debido a las ventajas y versatilidad que ello conlleva. Para el desarrollo de este proyecto se ha optado por instalar el Sistema Operativo directamente en una unidad de disco externa desde el cual arrancar el sistema, para contar así con todas las características hardware del equipo y no tener las limitaciones típicas de una máquina virtual.

El procedimiento de instalación, en cualquier caso, es muy similar y sencillo. Basta con descargar la imagen del Sistema Operativo, en este caso Ubuntu, para seguir con el uso de herramientas Open Source, ya que cualquier otro sistema operativo no cumpliría con esta premisa, si bien pueden servir otras distribuciones de Linux.

[36] https://docs.zephyrproject.org/latest/develop/getting_started/index.html

Instalación de Zehpy Project

Como siempre antes de empezar una instalación de cualquier característica o aplicación nueva es recomendable comprobar que el sistema y el repositorio están actualizados.

```
sudo apt update
sudo apt upgrade
```

Preparación previa

Para instalar Zephyr primero es necesario tener instaladas una serie de herramientas, en concreto *CMake*, *Python* y el compilador *Devicetree*.

```
wget https://apt.kitware.com/kitware-archive.sh
sudo bash kitware-archive.sh
sudo apt install --no-install-recommends git cmake ninja-build gperf \
ccache dfu-util device-tree-compiler wget python3-dev \
python3-pip python3-setuptools python3-tk python3-wheel xz-utils file \
make gcc gcc-multilib g++-multilib libsdl2-dev
cmake --version
python3 --version
dtc --version
```

El resultado de los últimos tres comandos debe dar algo similar a la figura.

Zephyr y dependencias Python

A continuación, se instala Zephyr y las librerías Python que necesita para funcionar:

```
pip3 install --user -U west
echo 'export PATH=~/.local/bin:"$PATH"' >> ~/.bashrc
source ~/.bashrc
west init ~/zephyrproject
```

```
cd ~/zephyrproject

west update

west zephyr-export

pip3 install --user -r ~/zephyrproject/zephyr/scripts/requirements.txt
```

En este último caso hay que estar pendiente de que no se produzcan errores de incompatibilidad de las dependencias requeridas con otras versiones de las librerías previamente instaladas en el equipo, como ocurre en la Figura A-1.

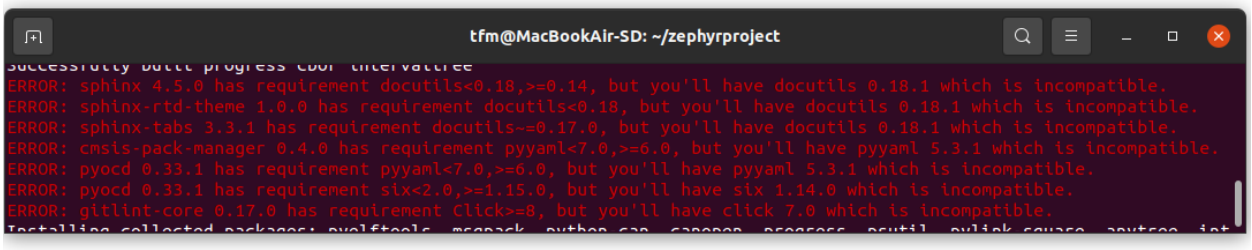

Figura A-1. Errores de compatibilidad de versiones dependientes

Para solucionar este tipo de casos hay que instalar la versión exacta correspondiente que requiere Zehpyr en todo el árbol de dependencias afectado. Esto se hace añadiendo == y el número de la versión que se desea instalar. Ésta viene indicada en el texto del error. Un ejemplo se ve en la Figura A-2, en la que la propia dependencia tiene otra que lo impide, por lo que hay que continuar el proceso por todo el árbol, con las librerías que en cada caso den error.

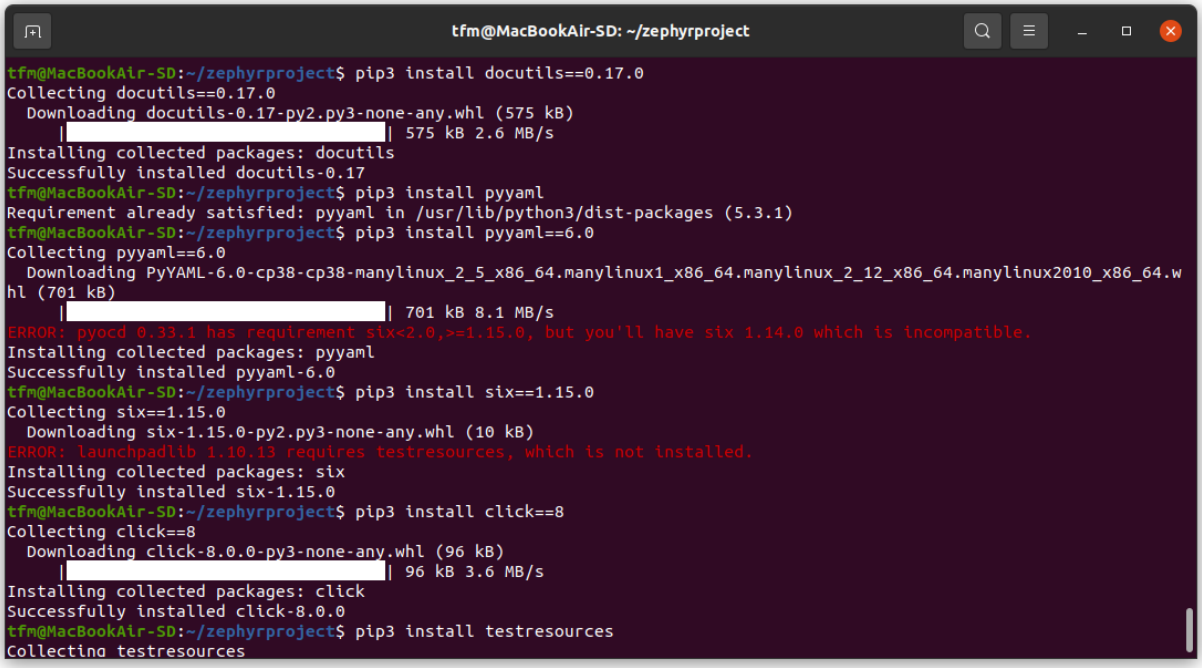

Figura A-2. Instalación de una versión determinada con pip3

Toolchain

A continuación, hay que instalar el SDK de Zephyr, que incluye las *toolchains* para soportar las diferentes arquitecturas. Por lo tanto posteriormente, habrá que instalar las correspondientes a la placa que se va a utilizar, la ESP32, ya que no vienen incluidas por defecto.

```
cd ~

wget https://github.com/zephyrproject-rtos/sdk-ng/releases/download/v0.14.2/zephyr-sdk-0.14.2_linux-x86_64.tar.gz

wget -O - https://github.com/zephyrproject-rtos/sdk-ng/releases/download/v0.14.2/sha256.sum | shasum --check --ignore-missing

tar xvf zephyr-sdk-0.14.2_linux-x86_64.tar.gz

cd zephyr-sdk-0.14.2

./setup.sh

sudo cp ~/zephyr-sdk-0.14.2/sysroots/x86_64-pokysdk-linux/usr/share/openocd/contrib/60-openocd.rules /etc/udev/rules.d
```

```
sudo udevadm control --reload
```

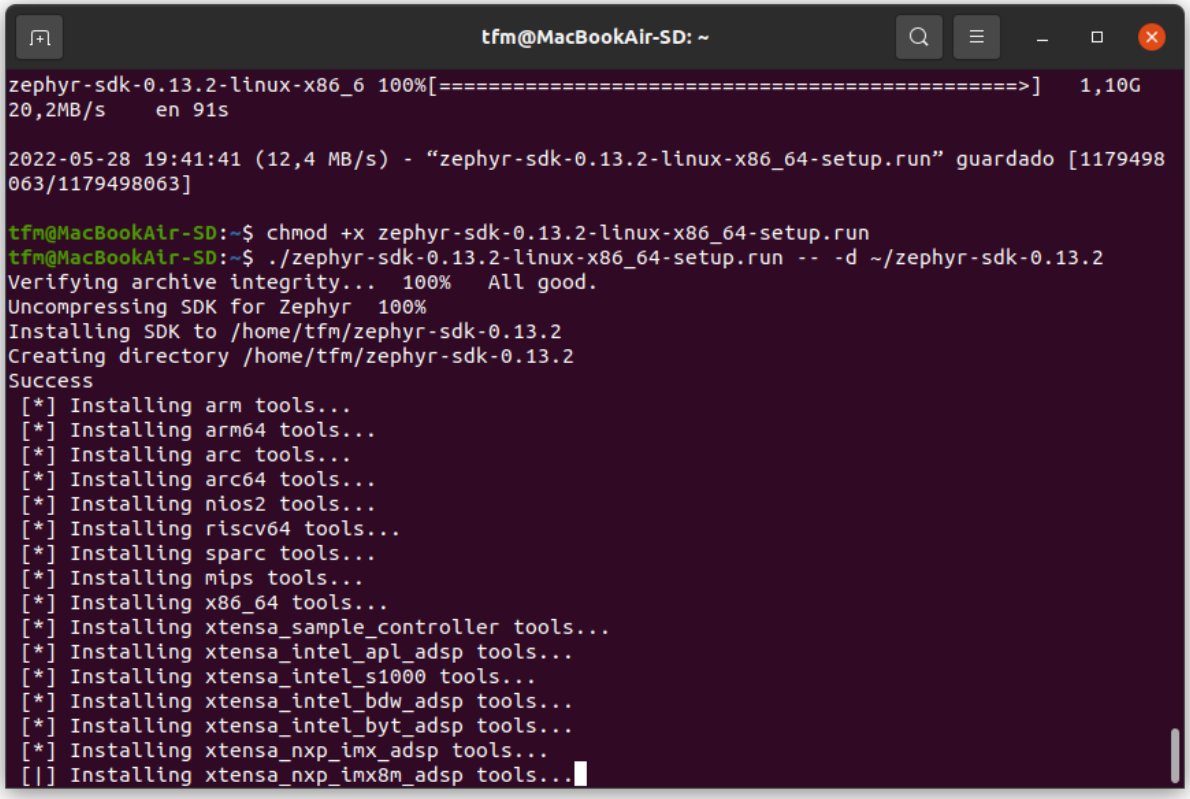

Figura A-3. Instalación del SDK

La figura muestra la vista de la consola durante la instalación del SDK de Zephyr.

Toolchain para ESP32

Este paso es un poco delicado ya que se deben tener muy en cuenta las diferentes versiones existentes y la compatibilidad entre ellas. Además, las ubicaciones por defecto no siempre coinciden con las que Zephyr requiere en su PATH, por lo que puede hacerse necesario copiar, mover o enlazar la *toolchain* a otra ubicación o incluso renovar los ficheros que la componen.

En primer lugar se realiza la instalación como debería ser, lo cual se muestra en la Figura A-4 y se realiza siguiendo las indicaciones de la guía web.[37]

[37] https://docs.zephyrproject.org/3.0.0/boards/xtensa/esp32/doc/index.html

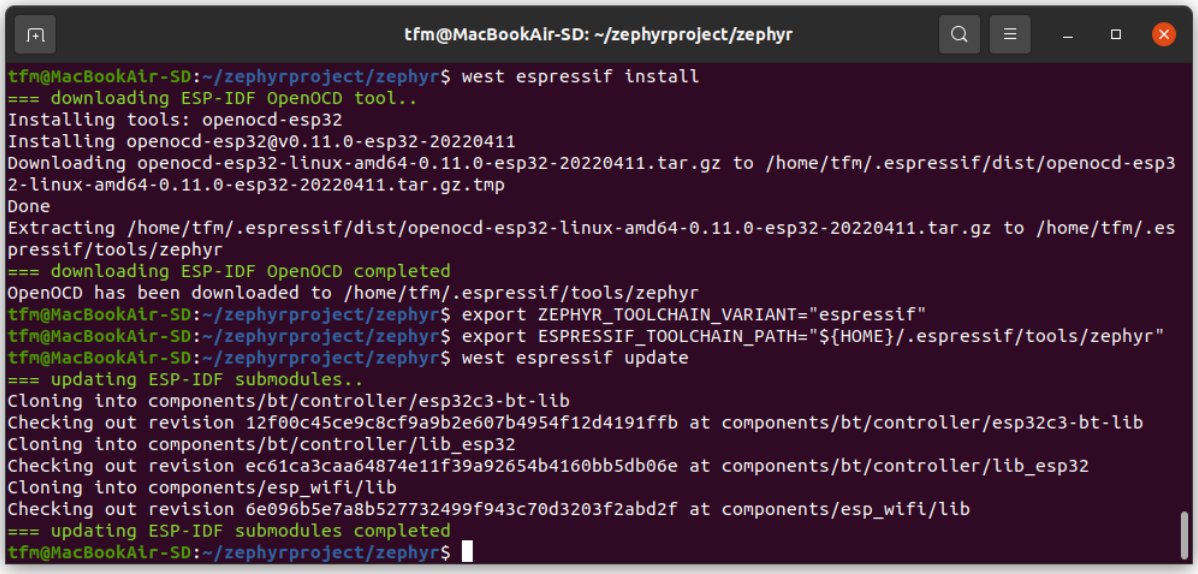

Figura A-4. Instalación del toolchain para ESP32

ESPtool y flasheo de ESP32

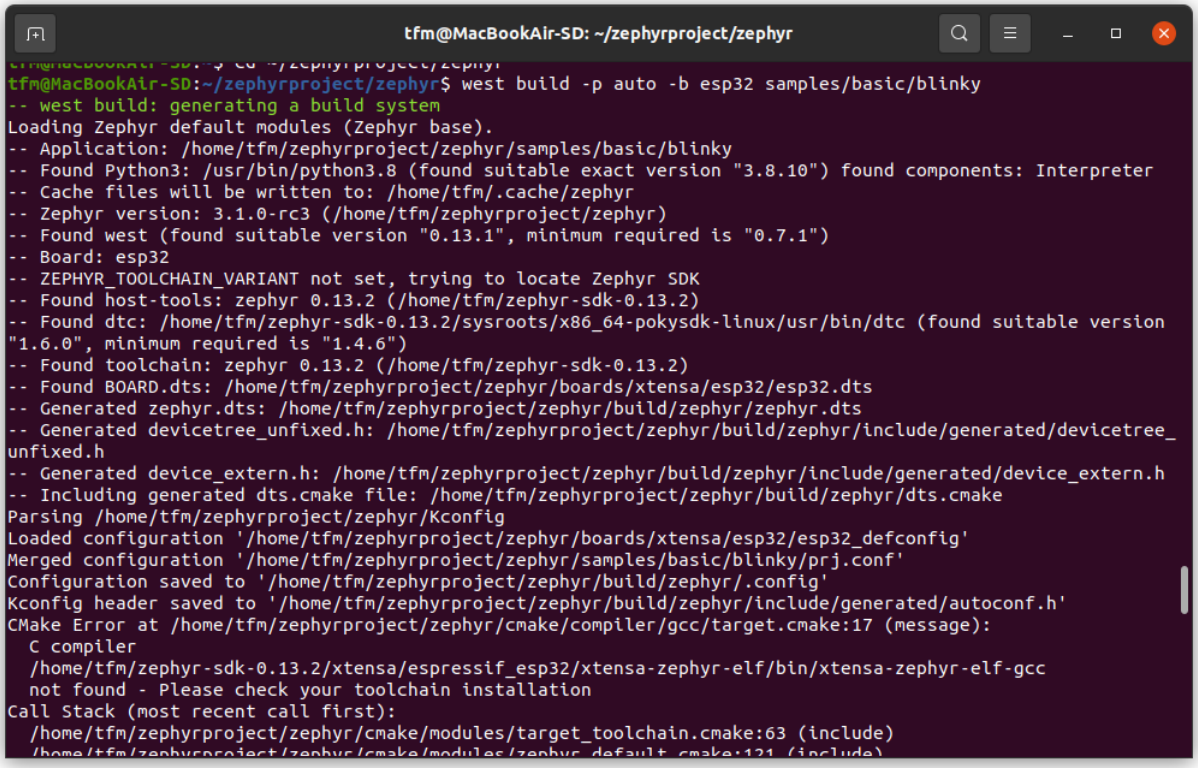

Figura A-5. Ejemplo de flasheo de aplicación en la ESP32

Si a la hora de construir una aplicación diera error por ausencia de las toolchains, cuestión que puede suponer un verdadero quebradero de cabeza, hay que comprobar en qué directorio están las toolchains necesarias (Figura A-6) coincide con el directorio en el que Zephyr las está buscando, el cual se muestra en el error.

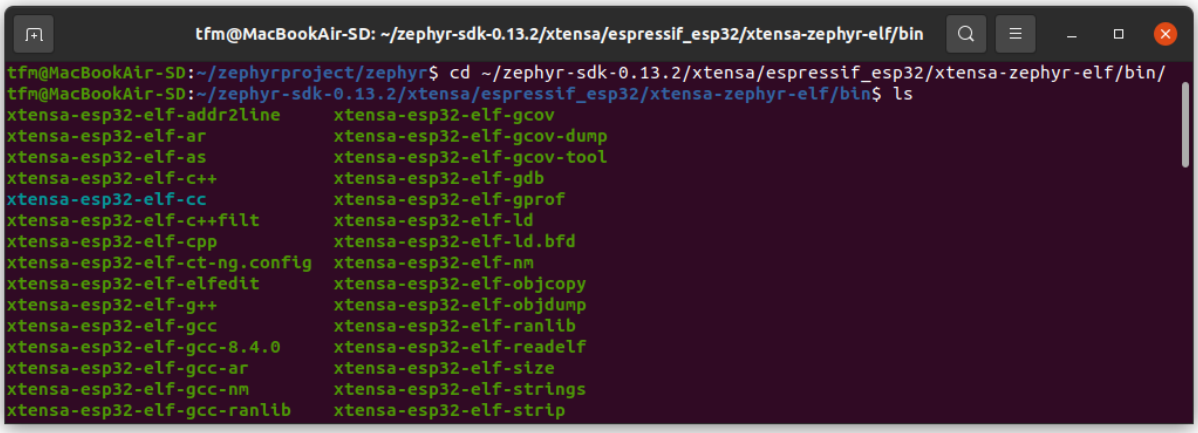

Figura A-6. Ubicación de las toolchains pasa ESP32

Para solucionarlo es necesario ubicar las toolchains en ese directorio, mejor copiando que moviendo por si alguna otra herramienta las tiene bien ubicadas, y renombrarlas con el nombre que espera tener, que en ocasiones no coincide. Esto se puede hacer con la herramienta rename, por ejemplo como se muestra en la Figura A-7.

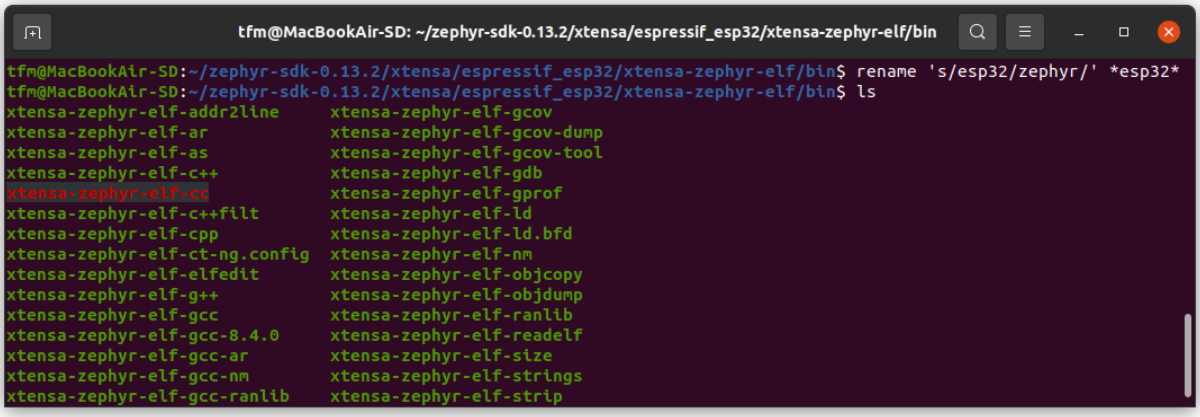

Figura A-7. Reubicación y renombre de las toolchains de ESP32 para su uso por Zephyr SDK

Apéndice B - Diario de bitácora

El presente Trabajo Fin de Máster fue concebido inicialmente para el curso 2019-2020, pero por diversos motivos, principalmente derivados de la pandemia, su ejecución ha sido aplazada en varias ocasiones hasta que se pudo establecer el mes de marzo de 2021 como fecha de inicio o reanudación.

En los primeros meses el trabajo se limita a una preparación previa de lo ya acordado por parte del estudiante, también teniendo en cuenta la situación de baja por paternidad del director.

Durante este tiempo se prepara, fruto de la investigación de las dos plataformas principales sobre las que versará el proyecto y que se explican en el capítulo de "Estado de la Cuestión", una presentación sobre Eclipse IoT y Zephyr Project, que será presentada en la próxima reunión, previa la Navidad.

Ya en diciembre se tiene una reunión en la que se expone la presentación anteriormente indicada y se pone en común la situación hasta ese momento. Se empiezan a establecer las primeras tareas concretas e hitos a completar de cara a próximas reuniones. En esa fecha se establece la necesidad de inventar o pensar un caso de uso que pueda utilizar cuantos más proyectos posibles mejor y de realizar un estudio en profundidad de los distintos proyectos de Eclipse IoT que vayan a ser de aplicación al caso de uso.

La siguiente reunión sería a mediados de febrero, en la que se tratan varios asuntos ya que el avance respecto a la anterior no fue menor, aunque principalmente en investigación del estado de la cuestión y menos en desarrollo tangible. Se empiezan a poner los conceptos sobre la mesa y a abordar el desarrollo del nodo. Es a partir de aquí cuando el estudiante prepara una máquina con todo el entorno de desarrollo necesario y se empieza a preparar el primer nodo sensor con un chip ESP32 montado en una placa de desarrollo NodeMCU. Se fija la siguiente reunión para dos semanas después.

Por algunas dificultades con la placa, ya que inicialmente se utilizó un chip ESP8266, que no es del todo compatible con las aplicaciones utilizadas, la reunión se mantuvo de forma rápida y se pospuso para una semana y media después.

Ya a mediados de marzo los avances con las pruebas con Zephyr y el nodo eran notables. En una reunión un poco más larga de lo habitual se explicaron los avances y las dificultades encontradas hasta el momento y se buscó conjuntamente solución a las mismas. En este momento también se mira el calendario que hay por delante con vistas a planificar la entrega del proyecto en la convocatoria de junio. De momento el avance es lo suficientemente adecuado y progresivo como para que esta posibilidad sea factible.

A finales de mes en otra reunión en la que se plantea la forma de crear el firmware del nodo, en dos patas: sensorización y publicación. En un principio la publicación podría ser con datos aleatorios en lo que se completa la sensorización. Se propone el uso del proyecto paho de Eclipse para los clientes y empezar a ver qué sensores se van a utilizar, cómo se van a programar y cómo se van a cablear.

Ya en abril se ve que se ha avanzado sobretodo en el funcionamiento de MQTT en el nodo pero sin funcionalidad asociada al caso de uso, se ha atascado un poco la parte de sensorización por lo que se deja temporalmente aparcada, mientras se intentará despejar la incertidumbre sobre si el uso de Ditto es válido para los objetivos que se buscaban y se empiezan habla de otros proyectos de Eclipse que podrían incorporarse en el nodo si diera tiempo como Hawkbit.

Tras el periodo vacacional de Semana Santa se reanuda la actividad y se empieza a redactar la memoria. En este periodo el trabajo se realiza de forma un poco más autónoma, realizándose el seguimiento fundamentalmente a través de correo electrónico y mediante comentarios y correcciones sobre los apartados de la memoria que se van redactando. También se realiza un breve parón por descanso de algo más de una semana a mediados de mayo. A la vuelta del mismo se continúa con la redacción de la memoria y atendiendo a los comentarios y sugerencias del director sobre las partes ya escritas y la estructura propuesta.

No es hasta el 31 de mayo cuando se mantiene otra reunión de larga duración en la que se toman decisiones de calado sobre la estructura de la memoria, el objetivo de aplicación que se va a cumplir y qué quedará como parte de trabajo futuro. Se retoma aquí la programación de la aplicación con el objetivo de finalizar las partes acordadas y poder describirla en la memoria con sus correspondientes muestras de funcionamiento. Unos días después se descubre un importante inconveniente de compatibilidad que se detallará convenientemente en la memoria. Debido al poco margen de maniobra se valoran diferentes alternativas para solventarlo, que también se describen más adelante. Simultáneamente se sigue avanzando en la escritura de esta memoria, en las partes que no requieren de haber finalizado el desarrollo, fundamentalmente documentando en el Estado de la cuestión las otras tecnologías existentes que no se utilizan en el proyecto.

Ya en la segunda semana de junio se da por fin por completada la aplicación del prototipo que será entregada, se documenta en la memoria y se incluyen muestras de funcionamiento, también con las instrucciones para desplegar el *gateway* y el uso de aplicaciones de cliente. Se finaliza por tanto la memoria y se realiza un repaso general sobre lo que podría considerarse el borrador previo, las correspondientes correcciones, fundamentalmente de erratas, y la entrega para su calificación.