Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos.

Itinerario de Ingeniería de Software 31105151

Sistema de propagación de averías en la Industria 4.0

Autor: Eduardo Buetas Sanjuan.

Director: Ismael Abad Cardiel.

JUNIO, 2017-2018

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos.

Itinerario de Ingeniería de Software 31105113

Sistema de propagación de averías en la Industria 4.0

Tipo de trabajo: B

Autor: Eduardo Buetas Sanjuan.

Director: Ismael Abad Cardiel.

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MÁSTER

Fecha: 30/05/2018.

Quién suscribe:

Autor(a): Eduardo Buetas Sanjuan.
D.N.I/N.I.E/Pasaporte.: 18046256L

Hace constar que es el autor del trabajo:

Título completo del trabajo.
Sistema de propagación de averías en la Industria 4.0

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo. BUETAS SANJUAN EDUARDO - 18046256L

Firmado digitalmente por BUETAS SANJUAN EDUARDO - 18046256L
Fecha: 2018.05.30 19:46:18 +02'00'

Eduardo Buetas Sanjuan

18046256L

IMPRESO TFdM05_AUTOR

AUTORIZACIÓN DE PUBLICACIÓN

CON FINES ACADÉMICOS

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del Autor:

BUETAS SANJUAN EDUARDO - 18046256L

Firmado digitalmente por BUETAS SANJUAN EDUARDO - 18046256L
Fecha: 2018.05.30 19:46:30 +02'00'

Eduardo Buetas Sanjuan

18046256L

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen

La industria, en un futuro próximo, sufrirá una gran evolución en los sistemas utilizados para la automatización de sus procesos. Con la llegada de la Industria 4.0, la llamada cuarta revolución industrial, proliferan ya y proliferarán los sistemas de automatización basados no en sistemas PLC (Programmable Logic Controller) como era habitual desde mediados de la década de los 80 hasta ahora, sino en pequeños sistemas inteligentes (SmartObject) interconectados entre ellos, creando pequeños sistemas de automatización trabajando de forma colaborativa.

Estos nuevos sistemas de control deberán convivir con los sistemas actuales de automatización y por ello será imprescindible buscar sistemas de comunicación comunes de estos nuevos controladores y de los controladores anteriores basados en PLCs, para el envío de los datos de planta a los sistemas superiores de información de las factorías, sistemas de control de producción (MES), de gestión empresarial (ERP), de propagación de averías, de calidad, etc.

Se defiende en este Trabajo Fin de Máster (en adelante TFM) que uno de los primeros sistemas que deberán evolucionar hacia sistemas de comunicación que permitan integrar los datos provenientes de controladores basados en PLCs y de los nuevos sistemas basados en SmartObjects serán los sistemas de propagación de averías. Esto es debido a la gran importancia de la correcta propagación de averías en el mantenimiento de la productividad, siendo clave para la reducción de los tiempos de paro de máquina en la industria.

Por ello en este trabajo se realiza una propuesta de un modelo de propagación de averías, incluyendo un servidor de propagación de averías capaz de recibir las averías provocadas por los diferentes sistemas, almacenarlas y distribuirlas, un controlador que envíe averías basado en PLC y dos basados en supuestos SmartObject, uno de ellos controlado por un microPC y otro controlado por un microcontrolador. También se propone en este trabajo un sistema móvil de recepción de averías para la recepción de las averías por parte del personal de mantenimiento correctivo de la factoría.

Antes de desarrollar estos elementos, en el presente trabajo se realiza un estudio del estado del arte existente en este campo, valorando las alternativas existentes y estudiando la conveniencia de su aplicación para conseguir los objetivos del presente estudio.

Keywords

Mantenimiento correctivo, protocolos de comunicación, industria 4.0, MQTT.

Í N D I C E

ÍNDICE FIGURAS

Í N D I C E T A B L A S

1. INTRODUCCIÓN

1.1. CONTEXTO

La propagación de averías en el mundo industrial es un tema capital para la productividad industrial. Esto implica que las averías producidas sean notificadas al personal de mantenimiento de las industrias, en tiempo y forma correcta, para que puedan ser solucionadas en el menor tiempo posible, reduciendo así al mínimo los tiempos improductivos de las máquinas o cadenas de producción.

Hasta ahora, la mayor parte de los controladores de automatización utilizados en la industria estaban basados en PLCs, pero esto era la industria 3.0.

En la actualidad y mucho más en los próximos años, con la llegada imparable de la industria 4.0, cada vez tenemos más elementos de control (automatización de máquinas, transportadores, Andon, Kanban, poka-yoke, gestión de almacenes, etc.) no basados en PLC, sino en los llamados Cyber-Physical-Systems (en adelante CPS por sus siglas en inglés).

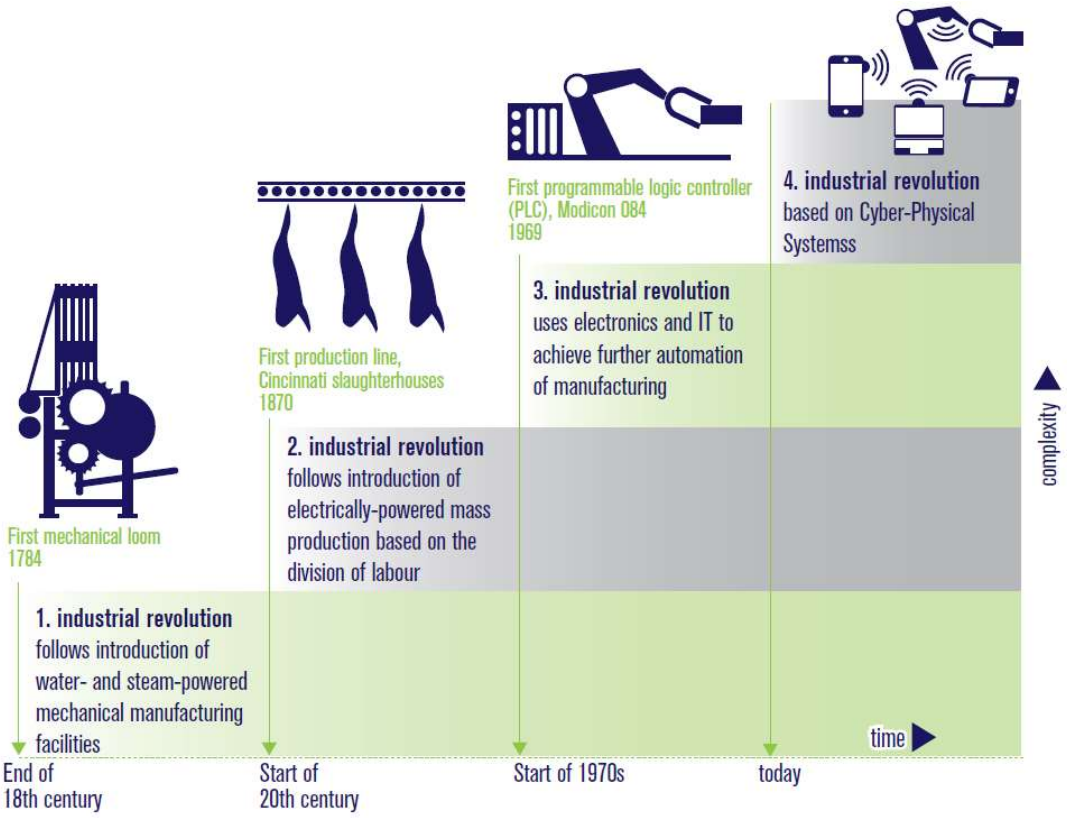

Fig. 1 Las cuatro fases de la revolución industrial. [1]

Los CPS están compuestos por sistemas inteligentes, sistemas de almacenamiento e instalaciones de producción capaces de intercambiar información de forma autónoma, desencadenar acciones y controlarse entre sí de forma independiente, según la definición aparecida en un informe de la Academia Nacional de Ciencia e Ingeniería de Alemania en 2013 [1].

Estos sistemas inteligentes generarán sus propios datos que deberán ser propagados a los sistemas superiores de las industrias: sistemas de control de producción (MES), de gestión empresarial (ERP), de propagación de averías, de calidad, etc.

Dichos dispositivos deberán convivir con los elementos de automatización actuales basados en PLC. Por esta razón, será crucial que se desarrollen sistemas de propagación de datos con protocolos de comunicación que puedan integrarse tanto en las tecnologías actuales como futuras.

Uno de los sistemas de comunicación más importantes para el mantenimiento de la productividad es el sistema de propagación de averías. Por ello, en este TFM se pretende estudiar la solución a la propagación de averías en la industria 4.0, que pueda integrar por un mismo canal los datos provenientes de sistemas de automatización basados en PLC y sistemas basados en las nuevas tecnologías que se implantarán con la llegada de la industria 4.0.

1.2. ESTADO DEL ARTE

En este punto se analizará el estado del arte existente por un lado para la intercomunicación de dispositivos entre los sistemas inteligentes embebidos y los sistemas superiores de control y por otro los protocolos ligeros de comunicación más utilizados para la comunicación entre dispositivos.

1.2.1. INTERCOMUNICACIÓN ENTRE SISTEMAS EN LA INDUSTRIA 4.0

Existen diversos artículos en los que se trata cómo estos nuevos sistemas inteligentes embebidos se deben comunicar con los sistemas superiores, por ejemplo [2], [3] y [4] entre otros, en estos artículos se plantea la utilización de *gateways* entre los SmartObjects y los servidores superiores.

Magdi S. Mahmoud et al en [2] proponen un sistema de *gateway* redundante entre los *Smart sensors* y los *Smart actuators* y los sistemas de control superiores (Fig. 3), sustituyendo al sistema de comunicación por capas actual (Fig. 2).

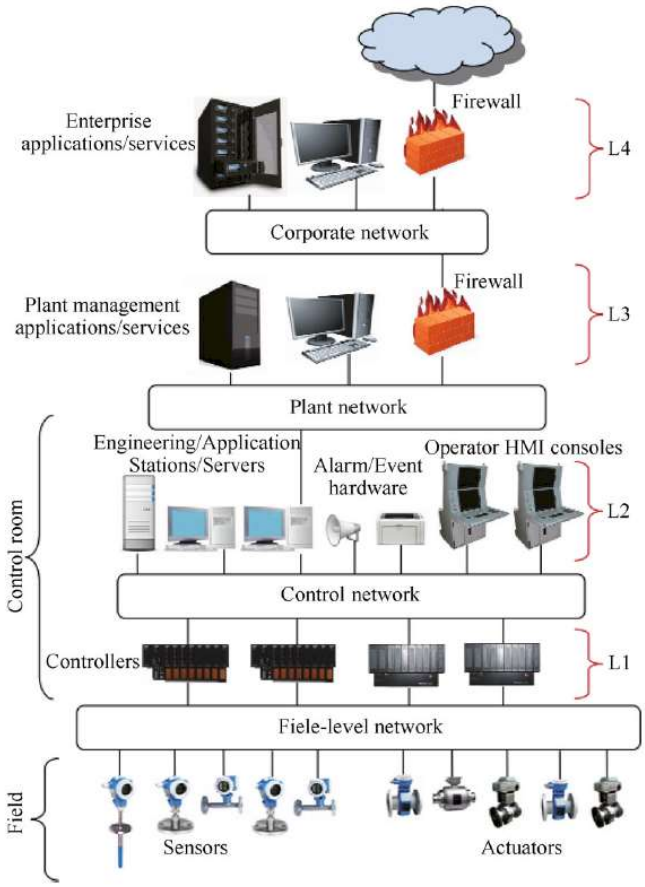

Fig. 2 Arquitectura multicapa [2].

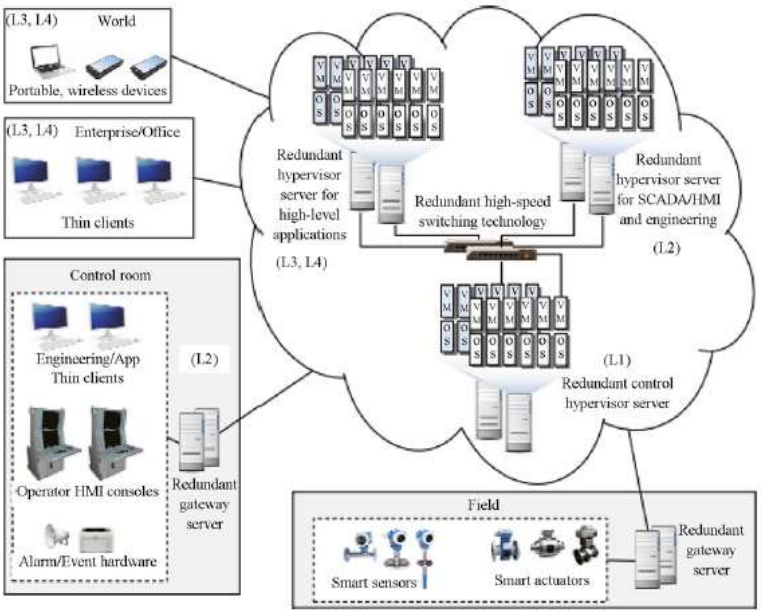

Fig. 3 Propuesta de comunicación de los Smart Objects a través de gateway redundante [2].

Por su parte Iveta Zolotová et al en su artículo [3], proponen la creación de una *gateway* que cree una representación virtual del mundo físico comunicándose con la capa de SmartObjets (mundo físico) y creando una interface de comunicación uniforme con la capa de los servidores superiores.

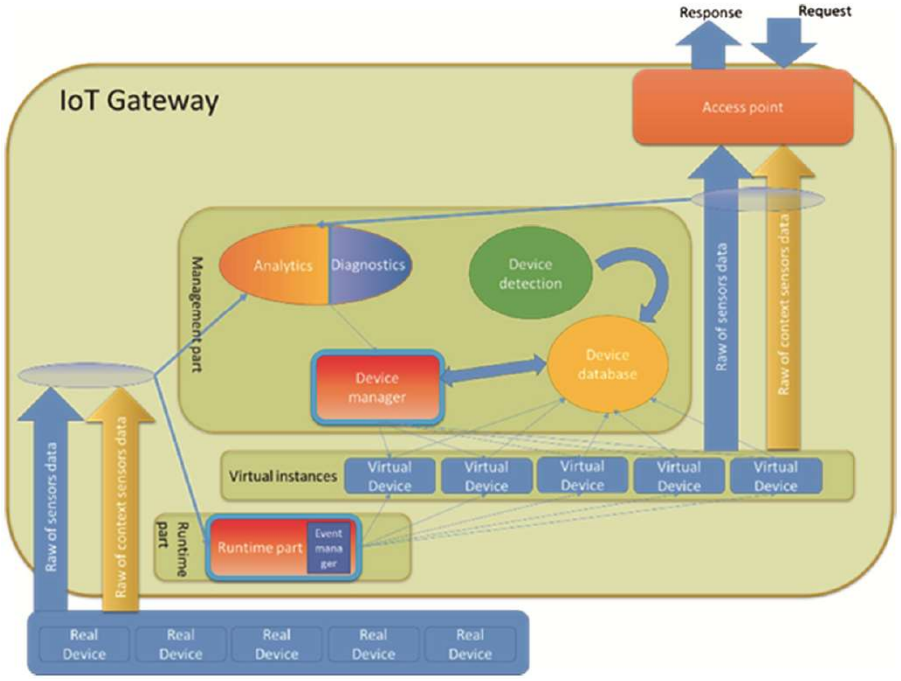

Fig. 4 IoT Gateway. [3]

En el artículo de Armando Astarloa et al [4], se describe una *gateway IP* (Fig. 5) creada por los autores del documento para la interconexión de los diferentes elementos de automatización con los servidores superiores Fig. 6.

Fig. 5 Smart-Gateway CPPS-Gate 40. [4]

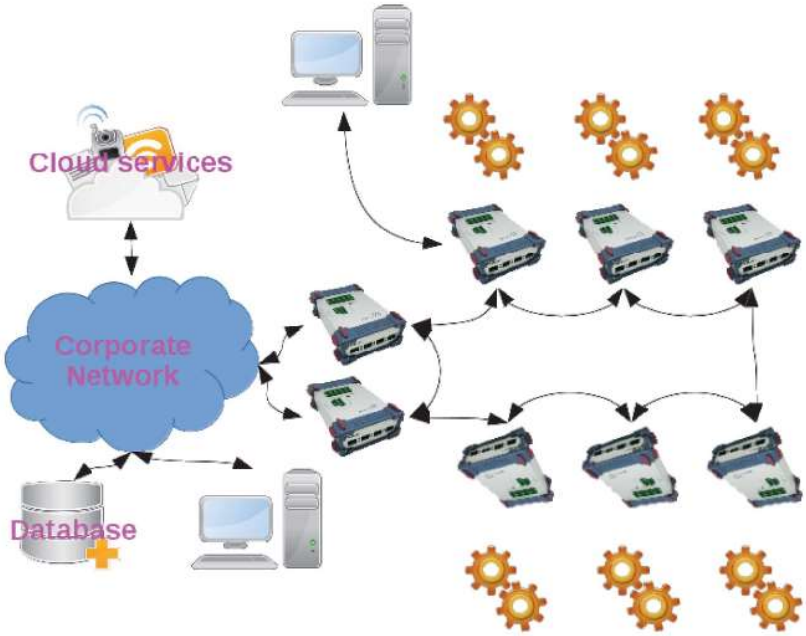

Fig. 6 Interconexión equipos a través de CPPS-Gate 40. [4]

Estos y otros artículos existentes en el estado del arte actual plantean la utilización de *gateways* entre los *SmartObject* y los sistemas superiores.

1.2.2. PROTOCOLOS DE COMUNICACIÓN LIGEROS

En este momento existen diversos protocolos de comunicación que se pueden implementar en los sistemas inteligentes embebidos, sobre todo utilizados en el ámbito del Internet of Things (en adelante IoT por sus siglas en ingles). En este punto se realiza un análisis sobre los protocolos más utilizados en este momento para la implementación de la comunicación de los SmartObjects en el ámbito del IoT.

Debido a la estandarización de las comunicaciones a través del protocolo de transporte TCP, tanto en los sistemas basados en PLC como en los sistemas SmartObject, todos los protocolos de aplicación analizados en este documento tienen como protocolo de transporte dicho protocolo.

1.2.2.1. REST

Es un protocolo cliente/servidor TCP sin estado, esto quiere decir que cada mensaje enviado por el protocolo debe contener toda la información necesaria para la comprensión del mensaje. No existe una sesión abierta que pueda darnos información sobre el mensaje fuera del propio mensaje.

En el IoT se suele emplear enviando datos con una codificación JSON (JavaScript Object Notation), introduciendo el nombre del elemento que proporciona el dato + el dato propiamente dicho:

{sensor:45, valor:33.4}

REST usa métodos HTML para implementar sus métodos CRUD (create, retrieve, update, delete):

- POST (crear)
- GET (consultar)
- PUT (editar)
- DELETE (eliminar)

En la actualidad muchos servicios web implementan su API REST para poder dar acceso a terceros a utilizar su información (Twitter, YouTube, los sistemas de identificación con Facebook, etc…).

Al utilizar este protocolo los métodos HTML para su implementación, no se entrará al detalle en este caso en la composición de sus mensajes.

1.2.2.2. CoAP

El protocolo CoAP (Constrained Application Protocol) está siendo especificado por la organización Internet Engineering Task Force, en su especificación RFC725 [5], esta definición se puede consultar a través de su página web (https://tools.ietf.org/html/rfc7252) .

Es un protocolo cliente/servidor UDP realizado para utilizarse en dispositivos con pocos recursos, son habituales nodos con microcontroladores de 8-bits y pequeñas cantidades de memoria RAM y ROM, que se pueden comunicar con otros dispositivos a través de redes con limitados recursos o bien con dispositivos que estén en redes sin problemas de recursos.

El protocolo CoAP se coloca entre la capa de transmisión (UDP) y la capa de aplicación, como muestra la Fig. 7.

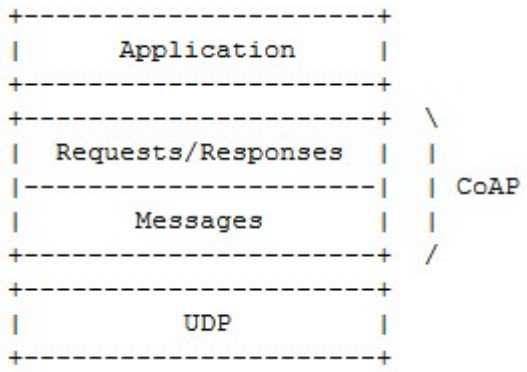

Fig. 7 Capas protocolo CoAP [5].

Dependiendo de la cabecera del mensaje podemos generar mensajes con acuse de recibo (Fig. 8) (en los cuales, si no existe ese acuse, se repite el mensaje pasado un timeout), o sin acuse de recibo. (Fig. 9).

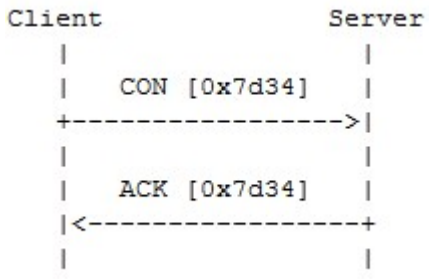

Fig. 8 Mensaje CoAP con Acuse [5].

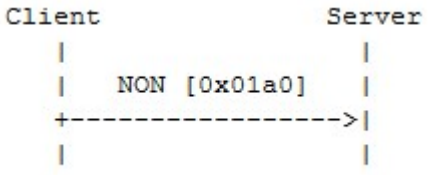

Fig. 9 Mensaje CoAP sin Acuse [5].

En este protocolo existen dos tipos de mensaje: requests (petición de información) y responses (envío de información). En Fig. 10 y Fig. 11 , podemos ver un intercambio de mensajes, con y sin acuse de recibo.

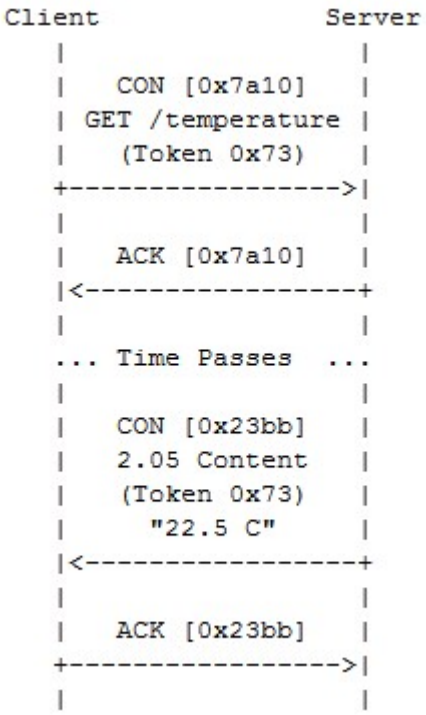

Fig. 10 Intercambio de mensajes Request--Response con acuse de recibo [5].

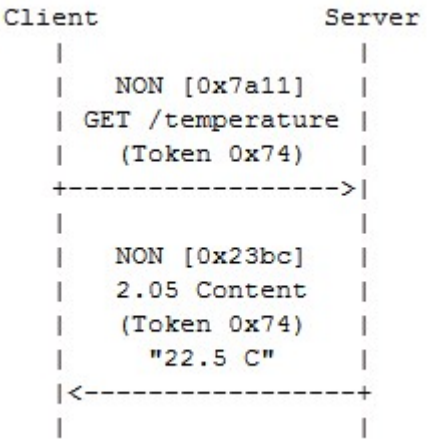

Fig. 11 Intercambio de mensajes Request--Response sin acuse de recibo [5].

Los mensajes están compuestos por una cabecera y un cuerpo, la longitud del cuerpo está especificada en la cabecera. El formato del mensaje lo podemos ver en su especificación en la Fig. 12.

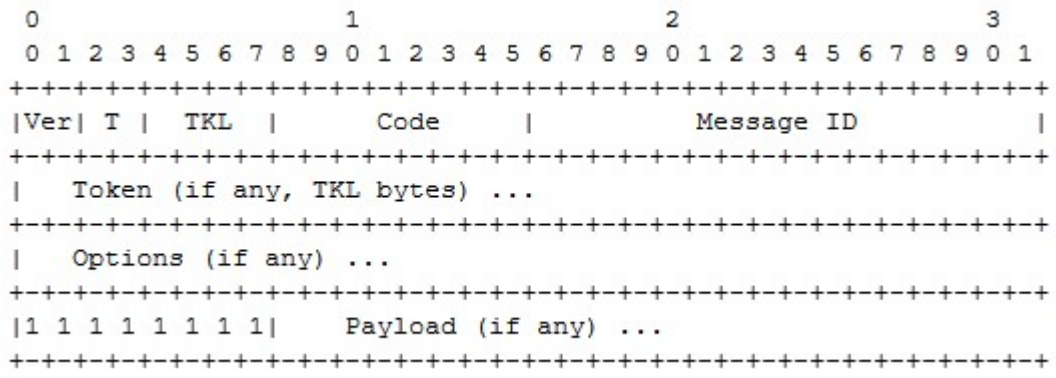

Fig. 12 Composición de mensaje CoAP [6].

Donde en su cabecera tenemos:

Ver: Dos bits. Versión del protocolo. Con la versión actual debe ser 01.

T: Dos bits. Tipo de mensaje (00)→Confirmable; (01)→No Confirmable; (10)→Respuesta; (11)→Reset.

TKL: Cuatro bits. Indica la longitud de la variable Token.

Code: Ocho bits. Se divide en dos secciones, los tres primeros bits son la clase y los tres siguientes indican su subclase (c.dd). Las clases de estos códigos se definen en [6] con la siguiente distribución:

0.00: Mensaje vacío
0.01-0.31: Request.
1.00-1.31: Reservados.
2.00-5.31: Response
6.00-7.31: Reservados.

Las subclases están definidas en [5] como muestran en Fig. 13 y Fig. 14.

Code	Name	Reference
0.01	GET	[RFC7252]
0.02	POST	[RFC7252]
0.03	PUT	[RFC7252]
0.04	DELETE	[RFC7252]

Fig. 13 Códigos de mensajes Request protocolo CoAP [5].

Code	Description	Reference
2.01	Created	[RFC7252]
2.02	Deleted	[RFC7252]
2.03	Valid	[RFC7252]
2.04	Changed	[RFC7252]
2.05	Content	[RFC7252]
4.00	Bad Request	[RFC7252]
4.01	Unauthorized	[RFC7252]
4.02	Bad Option	[RFC7252]
4.03	Forbidden	[RFC7252]
4.04	Not Found	[RFC7252]
4.05	Method Not Allowed	[RFC7252]
4.06	Not Acceptable	[RFC7252]
4.12	Precondition Failed	[RFC7252]
4.13	Request Entity Too Large	[RFC7252]
4.15	Unsupported Content-Format	[RFC7252]
5.00	Internal Server Error	[RFC7252]
5.01	Not Implemented	[RFC7252]
5.02	Bad Gateway	[RFC7252]
5.03	Service Unavailable	[RFC7252]
5.04	Gateway Timeout	[RFC7252]
5.05	Proxying Not Supported	[RFC7252]

Fig. 14 Códigos de mensaje Response protocolo CoAP [5].

Message ID: Número de mensaje utilizado para la no duplicidad de las lecturas y para la contestación en los mensajes confirmables.

Después de la cabecera tenemos el valor del token que, de existir, sirve para relacionar la respuesta con la consulta por parte del cliente.

A continuación, tenemos las opciones o bien el payload, las opciones (mensaje request) posibles las tenemos designadas en [5], como muestra Fig. 15.

Number	Name	Reference
0	(Reserved)	[RFC7252]
1	If-Match	[RFC7252]
3	Uri-Host	[RFC7252]
4	ETag	[RFC7252]
5	If-None-Match	[RFC7252]
7	Uri-Port	[RFC7252]
8	Location-Path	[RFC7252]
11	Uri-Path	[RFC7252]
12	Content-Format	[RFC7252]
14	Max-Age	[RFC7252]
15	Uri-Query	[RFC7252]
17	Accept	[RFC7252]
20	Location-Query	[RFC7252]
35	Proxy-Uri	[RFC7252]
39	Proxy-Scheme	[RFC7252]
60	Size1	[RFC7252]
128	(Reserved)	[RFC7252]
132	(Reserved)	[RFC7252]
136	(Reserved)	[RFC7252]
140	(Reserved)	[RFC7252]

Fig. 15 Posibles opciones en mensaje CoAP [6].

En los mensajes Response, tendremos el valor pedido en la parte del mensaje designada como payload. El siguiente ejemplo (Fig. 16) muestra un mensaje CoAP.

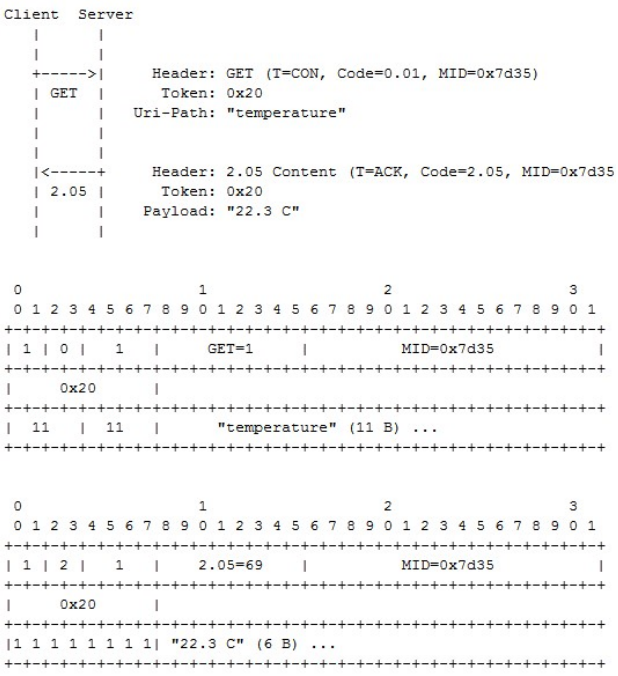

Fig. 16 Petición dato y respuesta protocolo CoAP [5].

Como hemos visto, este protocolo economiza la cantidad de datos utilizados en su comunicación y es un protocolo desconectado, es decir, cuando necesitamos un dato lo pedimos, sin tener que realizar ninguna negociación previa para la adquisición de un dato.

1.2.2.3. MQTT

Este protocolo ya está implantado para la comunicación de datos de sensórica y varios gigantes digitales ya están apostando por él. Así, Amazon tiene su propia plataforma online AWS IoT [7], también la tiene IBM, IBM Watson IoT Platform [8] o Google, aunque todavía en una versión beta de su API [9].

También desde el ámbito de la Industria se han hecho algunas aproximaciones para la utilización de este protocolo. Hay varios estudios que tratan la utilización del protocolo MQTT en este ámbito [10], [11] o [12].

Es un protocolo basado en publicaciones←→subscripciones, ligero (con un consumo de datos reducido) que se utiliza sobre el protocolo TCP/IP. Este estándar fue creado por el Dr. Andy Stanford-Clark de IBM y por Arlen Nipper de Arcom (ahora Eurotech) en 1999 y está definido en un estándar ISO (ISO/IEC PRF 20922).

A partir de marzo de 2013, el protocolo MQTT está bajo proceso de normalización por parte de OASIS (*Advancing Open Standards for the Information Society*) y, en este momento, la versión de estandarización en vigor es la 3.1.1, publicada el 29 de octubre de 2014 [13].

Este protocolo se basa en una topología en estrella, donde un nodo central actúa como controlador de la red (*broker*), el cual puede gestionar una gran cantidad de clientes (con valores típicos por encima de 10.000 clientes). Este *broker* es el encargado de gestionar las subscripciones de los clientes y sus publicaciones.

Las publicaciones de los clientes, llamadas tópicos, se estructuran de forma jerárquica, de modo que podemos establecer relaciones entre tópicos del tipo padre-hijo, como muestra la Fig. 17.

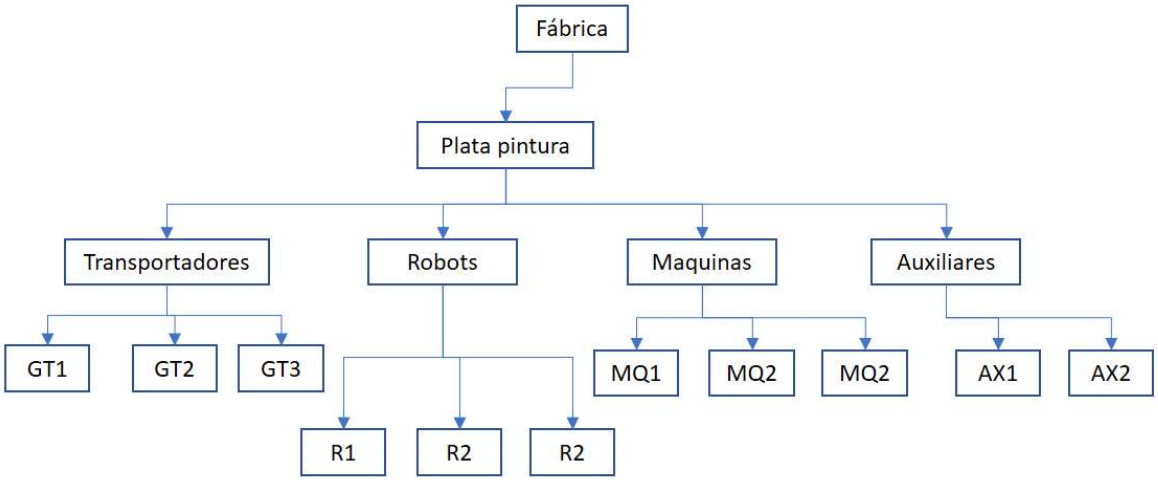

Fig. 17 Estructura publicaciones MQTT.

Cada cliente se puede suscribir a uno o varios tópicos, recibiendo todas las publicaciones que estén por debajo de ese tópico utilizando la *wildchar* #, es decir, un suscriptor que se suscriba al tópico "*Fabrica/#*" recibirá todas las publicaciones de la Fig. 17, un suscriptor que se suscriba al tópico "*Fabrica/Planta Pintura/Transportadores/#*" únicamente recibirá las notificaciones de "GT1", "GT2" y "GT3" y un suscriptor que se suscriba al tópico "*Fabrica/Planta Pintura/Robots/#*" únicamente recibirá las publicaciones de "R1", "R2" y "R3".

Este protocolo implementa tres tipos de calidad de servicio (QoS) (Fig. 18):

- *QoS 0 At most once delivery*: El mensaje se entrega de acuerdo con las capacidades de la red. No se envía ninguna respuesta por parte del receptor y el remitente no realiza ningún reintento. El mensaje llega al receptor una vez o ninguna.
- *QoS 1 At least once delivery*: El mensaje se entrega al menos una vez al receptor, el receptor debe enviar una respuesta al remitente, el remitente realiza reintentos si no

llega la respuesta del receptor, por lo que el receptor puede recibir más de una vez el mensaje.

- *QoS 2 Exactly once delivery*: El mensaje se entrega una y solo una vez al receptor. El receptor envía una respuesta al remitente, el remitente realiza reintentos si no llega la respuesta del receptor. Cuando llega la respuesta del receptor, el remitente envía una confirmación de recepción de respuesta y el receptor debe enviar una respuesta a esta confirmación. De este modo, los dos interlocutores saben que el mensaje ha sido correctamente entregado.

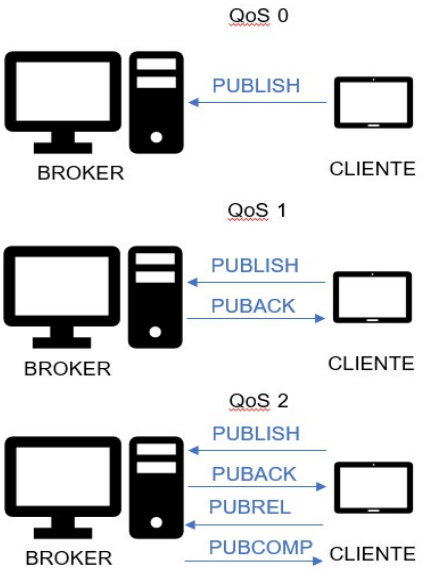

Fig. 18 Tipos de calidad de servicio (QoS) en MQTT.

Los mensajes MQTT tienen tres partes diferenciadas como nos muestra la Fig. 19.

Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Fig. 19 Estructura mensajes MQTT [13]

La cabecera fija, que está en todos los mensajes MQTT y nos dice de qué tipo de mensaje se trata, tiene la estructura mostrada en la Fig. 20.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

Fig. 20 Estructura cabecera fija MQTT [13].

En esta cabecera se informa del tipo de mensaje, este tipo de mensaje puede ser uno de los tipos de mensaje mostrados en la Fig. 21.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Fig. 21 Tipos de mensaje MQTT [13].

Como podemos ver por la estructura y cantidad de mensajes que puede contener el protocolo MQTT, este protocolo es fácilmente implementable y su huella de programa es muy reducida, por lo que será posible su implantación tanto en los dispositivos de control industrial actuales (PLCs), como en los nuevos sistemas de control industrial de la industria 4.0.

1.3. PROPUESTA DEL AUTOR

La inclusión de gateways genera utilización de recursos para: crear, instalar y mantener dichas gateways y por otro lado la inclusión de un nuevo componente en la comunicación susceptible de generar fallos y retrasos en la comunicación.

Es por ello por lo que la inclusión de componentes de tránsito entre redes o protocolos para las comunicaciones entre sistemas supone un gasto de recursos, que cuando es necesario está justificado, pero si podemos comunicar los dispositivos de un sistema en la misma capa de red, con el mismo protocolo, este enfoque supone un ahorro de recursos tanto en su implementación como en su mantenimiento.

Teniendo estas consideraciones en cuenta, la mejor aproximación a este nuevo reto en las comunicaciones dentro de la industria no es la inclusión de *gateways* entre los diferentes sistemas sino la inclusión de protocolos de comunicación comunes en los diferentes elementos del sistema, para la comunicación entre ellos, y así poder realizar una comunicación directa entre todos los componentes, sin necesidad de incluir nuevos puntos de salto entre capas.

En este trabajo se desarrolla un estudio para crear un sistema de propagación de averías para la industria 4.0, basado en un protocolo común entre las diferentes capas de los sistemas de control.

Este sistema será capaz de recibir las averías generadas por los diferentes sistemas de automatización, tanto de los basados en PLC como de los basados en sistemas inteligentes embebidos (SmartObject). Distribuirá dichas averías a los sistemas de alerta móviles que portarán los diferentes técnicos de mantenimiento correctivo y almacenará dichos datos en un SGBD (Sistema de Gestión de Bases de Datos) para la creación de informes y análisis forense de las incidencias.

2. SOLUCIÓN DESARROLLADA

La estructura propuesta en este TFM se presenta en la siguiente figura (Fig. 22).

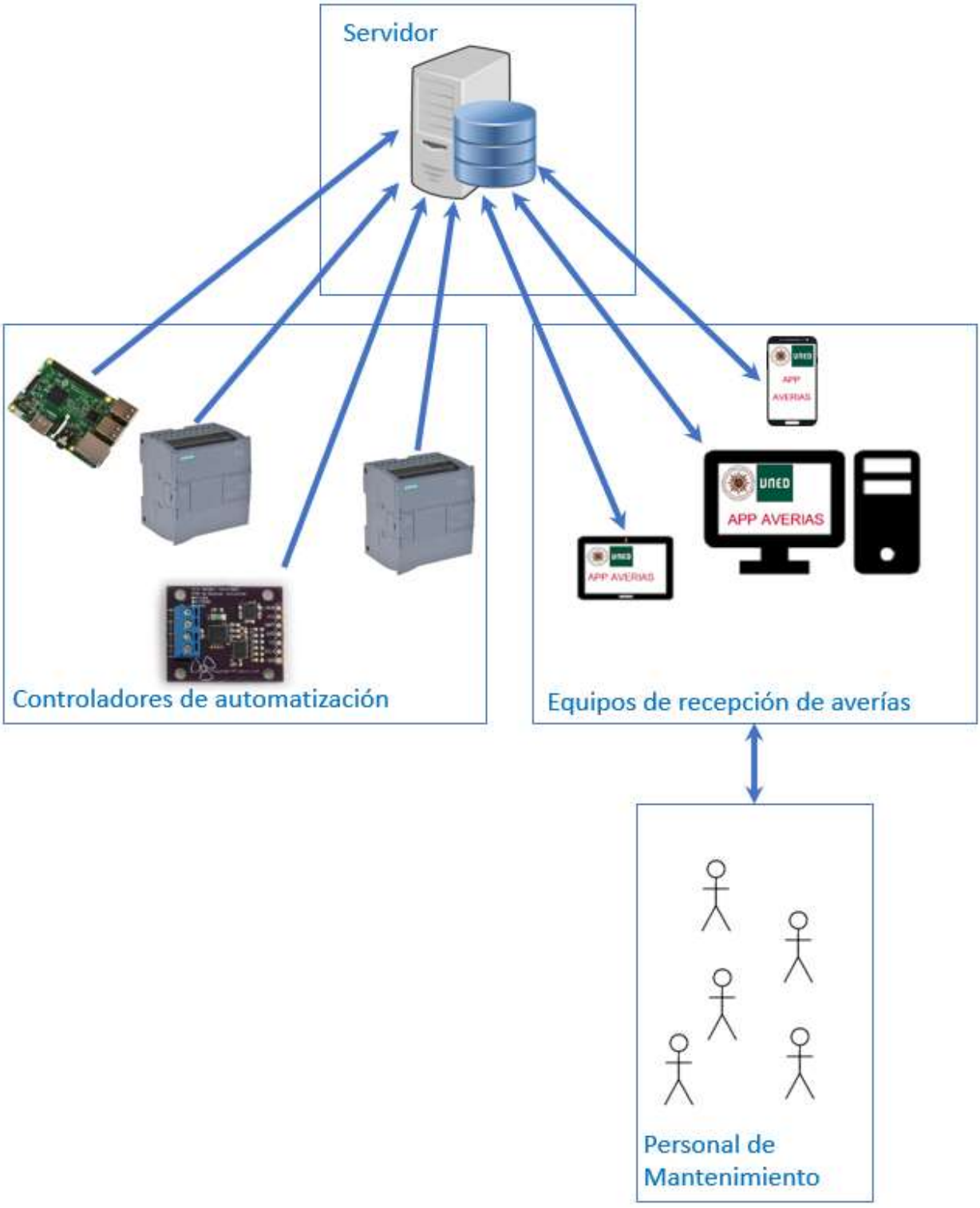

Fig. 22 Estructura sistema de propagación de averías.

El primer punto que dilucidar, según el estudio realizado, es el protocolo de comunicación utilizado para las comunicaciones entre los diferentes interlocutores del sistema, en este sistema tenemos los siguientes canales de comunicación principales:

1. Controladores de automatización → Servidor: envío de las incidencias producidas en los sistemas, así como el momento de su resolución.
2. Servidor → Equipos de recepción de averías: Envío de las incidencias a los equipos móviles para la información del personal de mantenimiento.
3. Equipos de recepción de averías → Servidor: Envío de los acuses de recibo de las incidencias por parte del personal de mantenimiento y petición de información extra.
4. Servidor → Equipos de recepción de averías: Envío de información adicional de una avería concreta bajo petición del usuario.

Para que esta intercomunicación entre los diferentes elementos del sistema pueda implementarse en todos los participantes y tenga un funcionamiento óptimo, el protocolo debe tener ciertas características:

- Baja huella de programa, para que en cualquier dispositivo ligero se pueda implementar.
- Capacidad para interconectar gran cantidad de dispositivos en la misma infraestructura.
- Facilidad de configuración y puesta en marcha.
- Baja o nula incidencia sobre la configuración del sistema, al añadir un participante.
- Facilidad de enrutamiento.
- Calidad de servicio (QoS) que asegure la comunicación.

2.1. COMPARACIÓN DE PROTOCOLOS

En este punto se realiza un análisis de los protocolos ligeros más utilizados, expuestos en el punto 1.2.2, para la elección del más adecuado para la consecución de los objetivos planteados. Teniendo en cuenta las restricciones con las que tenemos que trabajar, para que sean válidos para integrar los sistemas de propagación de averías, tanto en los sistemas actuales basados en PLC, como en los sistemas futuros basados en CPS.

Para poder implementar un sistema de transmisión de averías óptimo, que pueda implementarse en los sistemas de control propios de la industria 3.0 y en los de la industria 4.0, el protocolo a utilizar debe cumplir con ciertas características, expuestas en el punto anterior de este TFM.

Los tres protocolos cumplirán todas las características exigidas al protocolo excepto la calidad de servicio, que en el caso del REST no tienen ninguna posibilidad de gestionar calidad de servicio de ningún tipo y en el caso de CoAP solo tienen la posibilidad de controlar que el mensaje ha sido entregado al menos una vez sin poder controlar si un mensaje ha sido entregado más de una vez.

El protocolo MQTT tiene la capacidad de configurarse como un protocolo que controle si un mensaje ha sido recibido una y solo una vez por el receptor de este (QoS 2).

En cuanto a la iniciativa del mensaje, tanto el protocolo REST como el protocolo MQTT el emisor del mensaje envía la información por iniciativa propia, sin embargo, el protocolo CoAP está diseñado para que la información sea requerida por parte del receptor de esta (Request → Response).

El protocolo MQTT también tiene la ventaja de ser un protocolo de publicación / suscripción estructurado, lo que facilita el hecho que, al añadir un nuevo publicador por debajo de un nodo determinado, sin más configuración en los suscriptores la información de ese publicador llegue a dichos suscriptores.

Por estos motivos esta investigación se decanta por escoger el protocolo MQTT para el desarrollo del sistema de distribución de averías basado en el protocolo MQTT.

2.2. COMPONENTES DEL SISTEMA

Se describen a continuación los principales componentes de la solución propuesta en este TFM.

2.2.1. SERVIDOR DE PROPAGACIÓN DE AVERÍAS

El servidor de propagación de averías estará compuesto por tres sistemas bien diferenciados que se complementarán para la consecución de los objetivos de esta parte del sistema: el *broker* MQTT, el sistema de gestión de bases de datos y el software de propagación de averías.

2.2.1.1. BROKER MQTT Y ORGANIZACIÓN TÓPICOS

Como se ha descrito en el punto 1.2.2.3, el protocolo MQTT tiene una topología en estrella (Fig. 23), donde todos los publicadores envían la información a un nodo central llamado *broker* y es este nodo el que envía la información a todos los suscriptores.

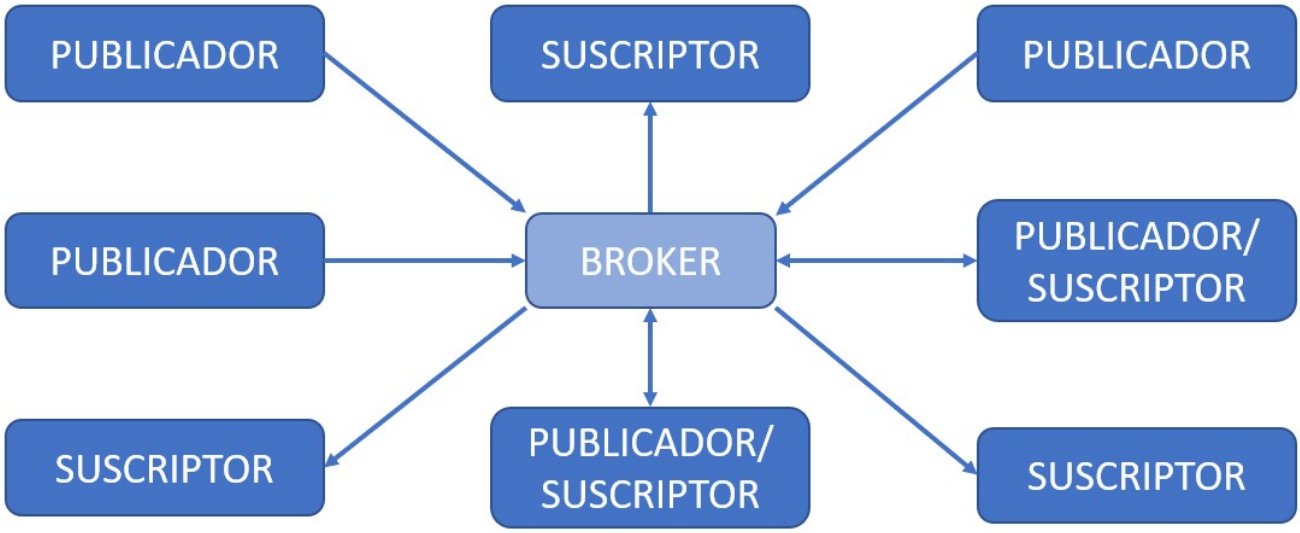

Fig. 23 Topología MQTT.

En nuestro sistema el *broker* se ejecutará en el servidor de gestión de propagación de averías (Fig. 22).

Este *broker* será el encargado de recibir y distribuir todos los mensajes del sistema. A él se enviarán todas las publicaciones de los participantes y será este *broker* el encargado de enviar las publicaciones a los suscriptores.

Los mensajes se publican y se suscriben organizándose por los llamados nodos, organizados jerárquicamente formando tópicos. Para la organización de dichos tópicos, en este TFM se ha ideado una estructura con dos grandes grupos organizados como nos muestran las figuras Fig. 24 y Fig. 26.

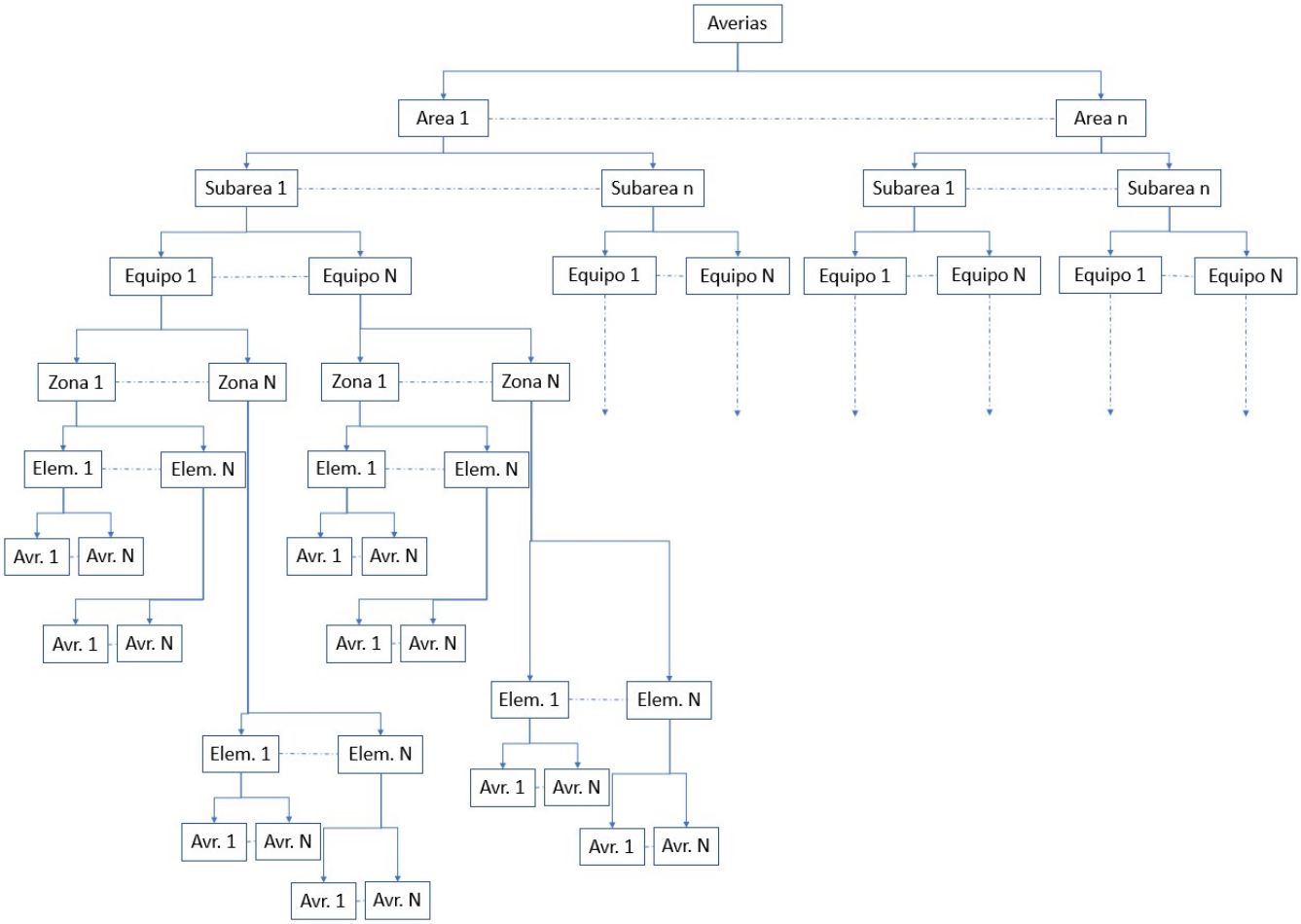

Fig. 24 Estructura tópico Averías, propuesta de autor.

Como podemos ver en la Fig. 24 por debajo del nodo "*Averias"*, que siempre tendrá este nombre, se organizarán todos los nodos de todas las averías del sistema. Esta estructura, ideada para la consecución de los objetivos de este TFM, estará organizada en los siguientes nodos:

- **Área:** la primera subdivisión de los nodos de las averías será el área al que pertenece el sistema generador de la avería. Esta primera subdivisión servirá para organizar las averías dependiendo de en qué zona de la fábrica se produce la avería, por ejemplo, en una organización habitual de una fábrica de automoción podríamos asignar un área para cada una de las zonas de producción:

  - Chapistería.
  - Fosfatación y Cataforesis.
  - Pintura.
  - Montaje Final.

- **Subárea:** La segunda subdivisión de estos nodos servirá para realizar subdivisiones organizativas dentro de un área general, siguiendo con el ejemplo de la factoría de automoción, es habitual que los equipos de mantenimiento estén divididos por funcionalidad de las instalaciones, que podrían corresponder con las subáreas definidas dentro de cada área:
  - Transportadores
  - Máquinas (máquinas de llenado, células robotizadas, etc...)
  - Infraestructura (Andon Calidad, Andon Material, Poka-Yoke, etc…)

  No será necesario que todas las áreas se subdividan en las mismas subáreas, dependiendo de la organización que cada factoría quiera definir, las áreas podrán estar subdivididas en las mismas subáreas o no.
- **Equipo:** Cada uno de los equipos controlados por un único sistema de automatización, continuando con nuestro ejemplo de una factoría de automoción, un equipo sería un grupo de transportadores que sean controlados por un solo sistema de control, una máquina concreta, una célula de soldadura robotizada, etc…
- **Zona:** Cada uno de los equipos puede estar dividido en una o varias zonas, por ejemplo, es habitual que en un transportador tengamos divididos los elementos por zonas de emergencia, o por cota de la instalación. En general la subdivisión en zonas de los elementos de un equipo se realiza para la rápida localización de un elemento dentro de los sistemas controlados por un equipo.
- **Elemento:** En este nodo se introducirá el nombre del elemento que ha producido la avería, un elemento podría ser dentro de un transportador cada una de las mesas que lo componen (elevadores, mesas giratorias, caminos de rodillos, etc…) dentro de una célula robotizada cada uno de los robots y mesas que la componen, en una máquina de llenado cada una de las bridas, válvulas que la componen, etc…

- **Nombre avería:** como último nodo del tópico tendremos el nombre de la avería producida en el elemento, debe ser un nombre descriptivo de la avería, siguiendo con nuestro ejemplo, las averías de una mesa de rodillos podrían ser:
  - Tiempo de tránsito a B1.
  - Tiempo de tránsito a B2.
  - Fallo Motor M1.
  - Detección B1 fuera de ciclo.
  - Etc…

Todas estas subdivisiones organizativas de cada avería deberán ser complementadas para el correcto funcionamiento de nuestro sistema. En el caso de un nodo no tenga subdivisiones, se podrá crear un único nodo hijo, pero deberá ser creado.

Por ejemplo, si un equipo no está subdividido en zonas, se deberá crear un único nodo hijo de este elemento (ejemplo en Fig. 25).

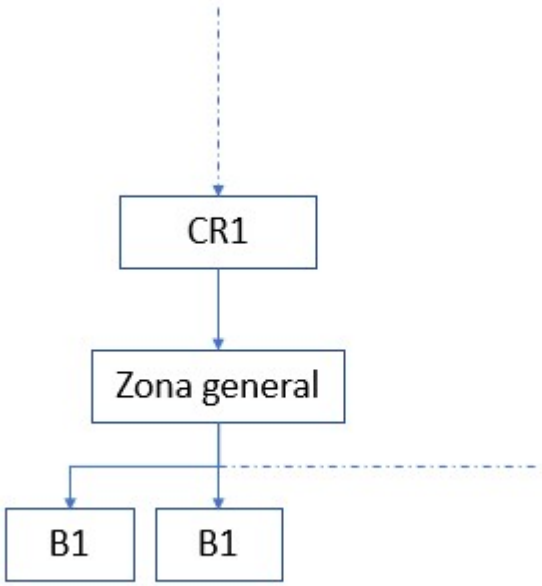

Fig. 25 Ejemplo tópico elemento con una sola zona.

Los nombres de estos nodos deberán ser significativos para la localización y comprensión de las averías producidas.

Los mensajes posibles en estos nodos serán los siguientes:

Finalización de la avería "0": Este mensaje será publicado por parte del controlador que controle el elemento que genere esta avería, este mensaje se producirá en el momento que una avería sea solucionada.

Activación de la avería "1": Este mensaje será publicado por parte del controlador que controle el elemento que genere esta avería, este mensaje se producirá en el momento que una avería se genere.

Acuse de la avería "2-equipo que acuse la avería": Este mensaje será enviado por parte del equipo de recepción de averías cuando el técnico de mantenimiento que trabaje con este equipo acuse la avería. En el mensaje además del indicador del acuse *"2"* y separado por un guion, se introducirá el nombre del equipo de recepción de averías. Este mensaje contiene el nombre del equipo de recepción de averías, que estará relacionado en base de datos con el técnico de mantenimiento que trabaje con él, para que se pueda almacenar en la base de datos el usuario que realiza el acuse de recibo de esta avería, así como el instante en el que se realice dicho acuse.

Petición de información "3-tipo de información requerida": Este mensaje será enviado por parte de un equipo de recepción de averías a petición del usuario para la descarga de información adicional sobre dicha avería, esta información se enviará a un nodo propio de este equipo de recepción de averías, estos nodos estarán organizados como veremos en el siguiente árbol de nodos mostrado en la Fig. 26. Los "*tipos de información requerida"* válidos para la petición de información, que deberán ser codificados detrás del número *"3"* y un guion *"-"*, son los siguientes:

- **"*mensaje*":** Esta petición de información dará como resultado el envío de un mensaje corto que describe brevemente la avería.
- **"*descripcion*":** Esta petición de información dará como resultado el envío de una descripción detallada de la avería.
- **"*actuacion*":** Esta petición de información dará como resultado el envío de una propuesta de actuación ante la avería. Esta información ayudará al técnico de mantenimiento que la requiera a tomar decisiones ante las actuaciones a realizar.
- **"*prioridad*":** Esta petición de información dará como resultado el envío de la prioridad codificada para esta avería. Estas prioridades serán un número entero

positivo, cuanto menor sea este valor mayor será su prioridad, siendo 0 la prioridad máxima configurable.

- **"todo":** Ante esta petición de información se enviará al peticionario todas las anteriores informaciones.

Por otro lado, tendremos el árbol de tópicos para la recepción de informaciones extra enviados por el servidor de propagación de averías a los equipos de recepción de averías, organizados como vemos en la Fig. 26.

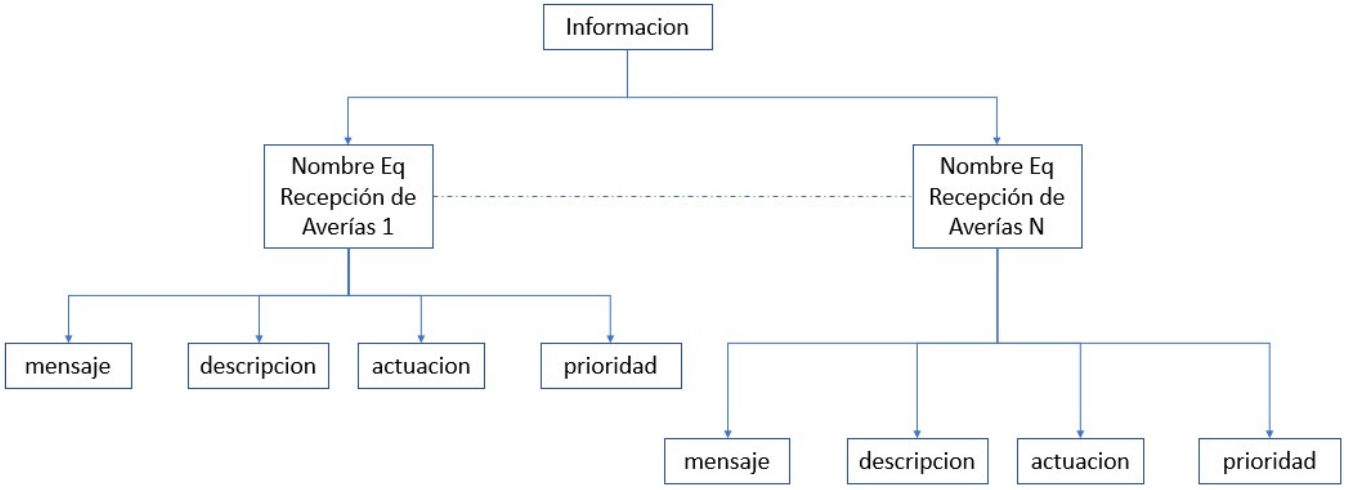

Fig. 26 Estructura tópico Información, propuesta para la realización de este trabajo.

Para la recepción de las informaciones extra se genera un nuevo nodo que se llamará siempre "*Informacion"*, por debajo de este nodo existirá un nodo hijo por cada uno de los equipos de recepción de averías, que tendrá como nombre el nombre de este equipo (almacenado en base de datos). Cada uno de los nodos que se referencien a los equipos de recepción de averías tendrá cuatro nodos hijos, en cada uno de estos nodos es donde se recibirá cada una de las informaciones que se pueden consultar desde estos equipos.

- **"mensaje".**
- **"descripcion".**
- **"actuacion".**
- **"prioridad".**

Para la generación de estas estructuras no es necesaria ninguna configuración en el servidor donde se ejecuta el *"broker"*, únicamente se crearán las publicaciones con estos tópicos en los diferentes publicadores de la instalación.

2.2.1.2. BASE DE DATOS DEL SISTEMA

El sistema de gestión de bases de datos (en adelante SGBD) contendrá la base de datos del sistema con la lista de averías, con todos sus datos, y almacenará las incidencias relacionadas con cada avería, aparición y desaparición de cada una de ellas, acuse por parte del personal de mantenimiento correctivo, etc...

Estos datos se deben almacenar para la generación de informes de análisis de las averías, que ayudan a planificar el mantenimiento preventivo, modificar los protocolos de actuación del personal de mantenimiento, etc.. Por otro lado, estos datos almacenados también se utilizarán para un análisis forense de una avería (o cadena de ellas) concreta.

Podemos ver el resumen de la base de datos del sistema en la Fig. 27.

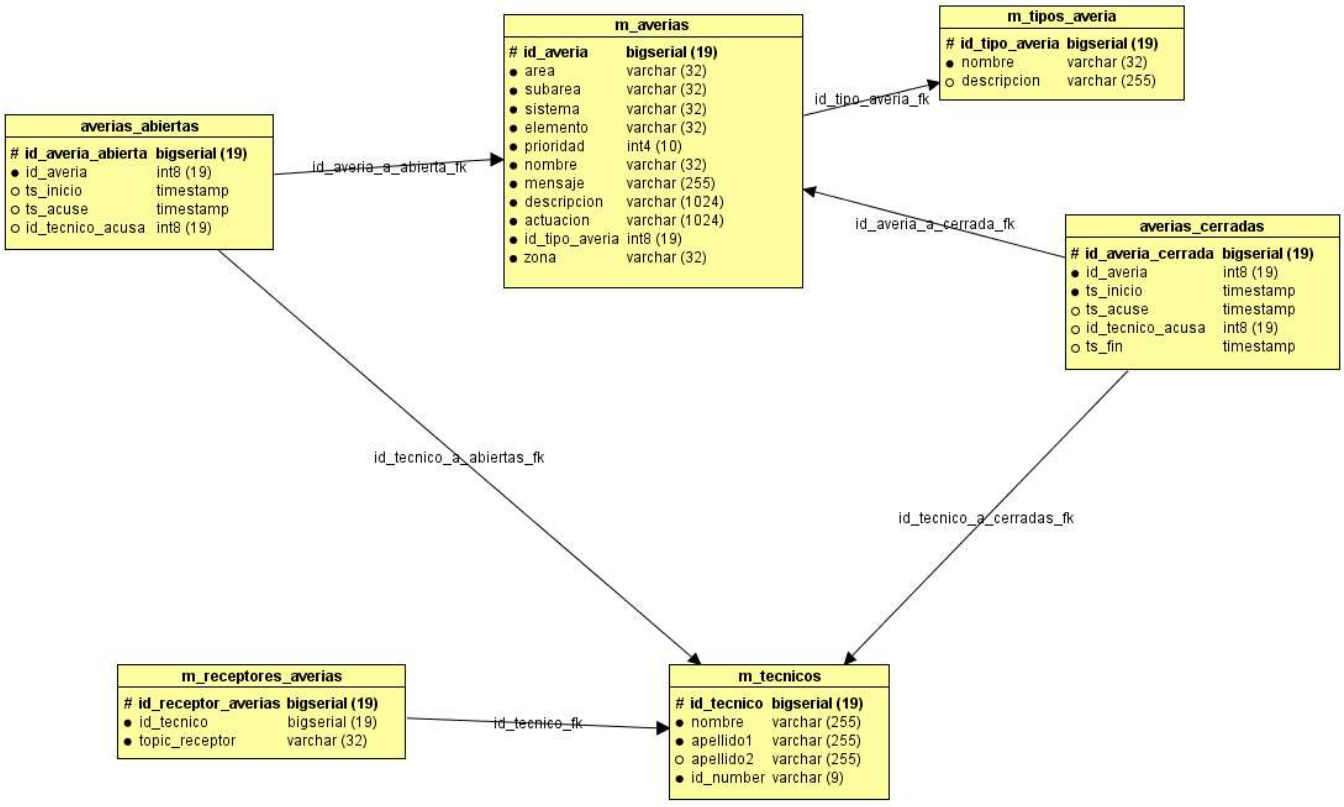

Fig. 27 Esquema base de datos Servidor del sistema propagador de averías.

Como podemos ver la base de datos del sistema estará compuesta por seis tablas que se detallan a continuación.

2.2.1.2.1. TABLA m_averias

Esta tabla será la tabla maestra de averías del sistema, en ella se contendrán todos los datos de las averías configuradas en el sistema. Cuando se dé una nueva alta de un elemento automatizado en el sistema su lista de averías deberá ser incluida en esta tabla para poder gestionar las averías de manera adecuada por el sistema.

En esta tabla se contendrán tanto los datos que conforman los seis niveles de nodos dependientes jerárquicamente del nodo "*Averias"* descritos en el punto 2.2.1.1 y representados en la Fig. 24:

- Área: máximo 32 caracteres.
- Subárea: máximo 32 caracteres.
- Sistema: máximo 32 caracteres.
- Zona: máximo 32 caracteres.
- Elemento: máximo 32 caracteres.
- Nombre: máximo 32 caracteres.

Así como las informaciones adicionales que se publicaran en los nodos dependientes jerárquicamente del nodo *"Informacion"* que podemos ver en la Fig. 26, como se describe en el punto 2.2.1.1:

- Mensaje: mensaje corto descriptivo de la avería, máximo 255 caracteres.
- Descripción: descripción detallada de la avería, máximo 1024 caracteres.
- Actuación: actuación recomendada ante la avería, máximo 1024 caracteres.
- Prioridad: nivel de prioridad de la avería, número entero.

Para su utilización en la generación de informes y estadísticas también se incluye en esta tabla el tipo de avería, para ello se incluye una clave externa a la tabla auxiliar de tipos de avería (*m_tipos_averia*).

2.2.1.2.2. TABLA m_tipos_averia

Esta tabla será la tabla maestra de los tipos de avería únicamente utilizada para su tratamiento en la generación de informes, estadísticas, etc..

Esta tabla, además de su clave principal, contendrá el nombre y la descripción del tipo de avería.

2.2.1.2.3. TABLA m_receptores_averias

Esta tabla contendrá los datos de los diferentes equipos receptores de averías que estarán utilizándose en el sistema, descritos en el punto 2.2.3. de este documento.

Esta tabla únicamente contendrá el tópico que utilizará el equipo, que será el nombre del equipo y una clave externa a la tabla *m_tecnicos* que relacionará el equipo con el técnico que en este momento tenga asignado este equipo.

2.2.1.2.4. TABLA m_tecnicos

Esta tabla contendrá los técnicos de mantenimiento dados de alta en el sistema, con sus datos identificativos, nombre, apellido1, apellido2 y su número de DNI o tarjeta de residente.

2.2.1.2.5. TABLA averias_abiertas

Esta tabla tendrá un listado actualizado de las averías que se encuentran en cada momento activas. Tendrá los siguientes datos:

- Clave que relacione la avería con sus datos en la tabla *m_averias*.
- Sello de tiempo del instante en que la avería sea creada, este dato se creará en el momento de crear el registro en la tabla.
- Sello de tiempo del instante en que la avería ha sido acusada, si la avería no ha sido acusada este campo estará vacío.
- Clave que relacione el acuse de recibo de la avería con el técnico que ha acusado la avería en la tabla *m_tecnicos*, si la avería no ha sido acusada este campo estará vacío.

En esta tabla solo se encontrarán las averías activas, por lo que cuando una avería sea cerrada, desaparecerá de esta tabla y será almacenada como histórico en la tabla *averias_cerradas*.

Esta tabla únicamente podrá contener un registro por cada avería abierta, por lo que en esta tabla nunca se podrá repetir el campo clave que relaciona el registro con los datos de la avería en la tabla *m_averias*.

2.2.1.2.6. TABLA averias_cerradas

Esta tabla tendrá un histórico de todas las averías producidas en la instalación y que ya han sido resueltas, esta tabla se utiliza como histórico para la generación de informes, estadísticas y análisis forense de las averías producidas en la instalación.

Tendrá los mismos campos que la tabla *averias_abiertas*, que se rellenarán con los valores almacenados en la tabla *averias_abiertas* en el momento de producirse la conclusión de la avería, además tendrá un campo para almacenar el instante en el que la avería se ha resuelto y que se rellenará cuando se cree un registro rellenando este campo con el sello de tiempo del instante en el que la avería ha sido resuelta.

2.2.1.3. SOFTWARE PROPAGADOR DE AVERÍAS

Este software será otro componente del servidor, estará corriendo en él y se encargará de las publicaciones y suscripciones que el servidor debe realizar para el correcto funcionamiento del sistema.

Este software estará suscrito al tópico *"Averias/#"* recibiendo todos los mensajes cuyo tópico comience por el nodo "*Averias"*, o dicho de otro modo todos los mensajes relacionados con las averías de la instalación.

Cuando reciba un mensaje tipo *"1"*, leerá del tópico por el que le llega el mensaje los diferentes elementos que identifican una avería de manera unívoca; área, subárea, nombre del equipo, zona, elemento y nombre de avería. Con estos datos incluirá esta incidencia en el base de datos del sistema como "Avería abierta", incluyendo el sello de tiempo del instante en el que la avería es producida.

En el momento de recibir un mensaje de tipo *"2-nombre del equipo de envío"*, igual que en el caso anterior leerá la información para la identificación unívoca de la avería del tópico por el que llega el mensaje y añadirá en la base de datos a la avería abierta, el sello de tiempo del instante del acuse de recibo de la avería, así como el técnico de mantenimiento que ha realizado el acuse de recibo de la avería. Para ello consultará en la base de datos el técnico asociado en ese momento al equipo que ha enviado el acuse de recibo.

Por último, al recibir un mensaje de tipo *"3-tipo de información pedida"*, desencadenará la publicación de la información requerida.

Este software publicará esta información concreta de una avería pedida por un "*Equipo de recepción de averías"*, consultando esta información en la base de datos identificando la avería de manera unívoca a partir de la información obtenida del tópico del cual se recibe la petición de información de la avería y lo hará sobre un tópico compuesto de la siguiente manera *"Informacion/Nombre del equipo que ha pedido la informacion/tipo de informacion"* y en el contenido del mensaje enviará la información requerida por el peticionario de esta.

2.2.2. CONTROLADORES DE AUTOMATIZACIÓN

Estos sistemas, que serán los controladores de los diferentes sistemas automatizados de la instalación, únicamente actuarán como publicadores de mensajes en el sistema sin estar suscritos a ningún nodo.

Estarán configurados con el área y subárea donde se encuentran ubicados, el nombre del sistema y con sus diferentes zonas y elementos dentro de cada zona, así como el nombre de cada avería de los elementos.

Cuando se produzca una avería el sistema construirá la cadena correspondiente al tópico de esa avería "*Averias/Area/Subarea/Nombre Equipo/Zona/Elemento/Nombre Averia*" y enviará un mensaje "*1*" al sistema, cuando se resuelva la avería enviará un mensaje "*0*" al sistema con el mismo tópico.

Cuando el sistema se conecte al *broker* se enviará un mensaje especial para informar al servidor de propagación de averías y a los suscriptores del nodo de este sistema que el sistema está conectado, este mensaje tendrá el contenido "*0*" en el tópico "*Averias/Area/Subarea/Nombre Equipo/Mqtt/Mqtt/Conexion*", siendo su zona, elemento y nombre de avería fijo para todos los controladores.

También cuando el *broker* de la instalación detecte la desconexión de uno de estos equipos, enviará un mensaje "*1*" al tópico "*Averias/Area/Subarea/Nombre Equipo/Mqtt/Mqtt/Conexion*". Esta avería especial estará configurada en todos los controladores para que el servidor sepa en todo momento qué controladores han perdido la conexión con el servidor, y por lo tanto no están enviando averías al sistema.

Para que se produzca el envío de esta información cuando un controlador se desconecte del sistema se utilizará el *flag* de conexión del protocolo MQTT "*will"* creado para este uso como estándar del protocolo.

2.2.3. EQUIPOS DE RECEPCIÓN DE AVERÍAS

Estos equipos estarán suscritos a dos tópicos diferentes, corriendo dos clientes MQTT distintos. El primero de los clientes estará suscrito a el nodo superior del área que deba controlar el técnico de mantenimiento al que se le ha asignado. Por ejemplo, un técnico de mantenimiento que deba atender las averías de transportadores de la nave de pintura se suscribirá al tópico *"Averias/Nave Pintura/Transportadores/#"* recibiendo de este modo todos los mensajes de las averías que estén jerárquicamente por debajo de ese nodo, es decir todas las averías de la subárea de transportadores dentro de la nave de pintura.

Mostrarán una lista de averías, cuando un mensaje *"1"* de una avería sea recibido la información de esta avería será mostrada en rojo en la lista de averías. Cuando la avería sea acusada, por el mismo equipo, o por otro, se recibirá un mensaje *"2-nombre del equipo de envío"* de esta avería, con la información de dicha avería obtenida de manera unívoca por el tópico por el que se ha recibido el mensaje se buscará la avería en la lista y se cambiará su estado y color, pasando este a amarillo.

Cuando el equipo reciba un mensaje *"0"* de una avería el usuario, por configuración tendrá la oportunidad de que la avería se mantenga en la lista hasta su borrado manual, en cuyo caso se mantendrá en la lista en color verde o bien desaparecerá de manera inmediata.

Las averías en la lisa se ordenarán siempre según el orden de llegada de la incidencia, siendo las más recientes las que aparecerán en primer lugar.

El usuario podrá pedir información adicional de una avería en concreto como se ha comentado en el punto 2.2.1.1, para ello el equipo enviará un mensaje por el mismo tópico por el que ha recibido la avería para pedir la información deseada sobre ella con uno de los siguientes mensajes:

- "*3-mensaje*"
- "*3-descripcion*"
- "*3-actuacion*"
- "*3-prioridad*"
- "*3-todo*"

Cuando esto suceda el equipo estará a la espera de la información con su segundo cliente MQTT suscrito al tópico *"Informacion/Nombre del equipo/#"* recibiendo en esta

suscripción todas las informaciones adicionales que le sean enviadas por el servidor de la instalación.

Por otro lado, para realizar el acuse de recibo de la avería enviará un mensaje *"2-nombre del equipo"* con el mismo tópico por el que ha recibido la avería

Estos equipos deberán ser equipos portátiles que los técnicos de mantenimiento puedan llevar encima durante toda su jornada laboral, al ser equipos móviles como smartphone, tablets, etc. Deberán producir los correspondientes avisos cuando aparezca alguna incidencia en el sistema, avisos sonoros, vibraciones, notificaciones, etc…

2.2.4. EJEMPLO PROPAGACIÓN DE AVERÍAS

Como ejemplo de intercambio de mensajes supondremos un sistema de automatización llamado *Grupo Funcional 1* (*GF1*), que pertenece a los grupos funcionales de *Transportadores* de la *Nave de Pintura*.

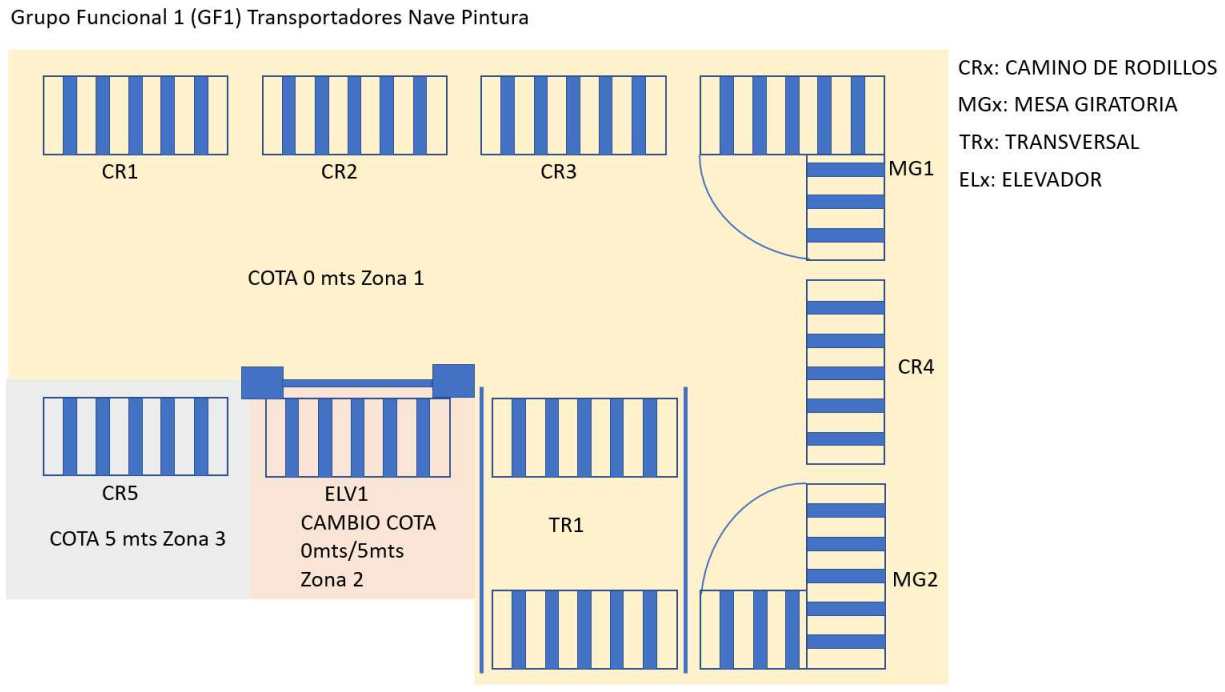

Fig. 28 Grupo Funcional 1.

2.2.4.1. AVERÍA PRODUCIDA

En *GF1* se produce una avería de *Tiempo de Transito B1* en su elemento *Camino de Rodillos 1* (*CR1*) que este situado en la *Zona 1* de este grupo funcional.

Como consecuencia de esta avería el controlador de *GF1* publica, de manera automática, un mensaje "*1"* sobre el correspondiente tópico "*Averias/Nave Pintura/Transportadores/CR1/Tiempo Transito B1"* (Tabla 1). Este mensaje es recibido por el *broker* que controla la propagación de averías y enviado por dicho *broker* a todos los suscriptores que estén jerárquicamente suscritos a él.

SE PRODUCE LA AVERIA DE TIEMPO DE TRANSITO A B1 EN EL ELEMENTO CR1 DEL GRUPO FUNCIONAL 1 DE TRANSPORTADORES DE LA NAVE DE PINTURA			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Sistema automatización "GF1" que controla "CR1"	Averias/Nave Pintura/Transportadores/GF1/Zona 1/CR1/Tiempo Transito B1	1	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....

Tabla 1 Envío avería producida.

En esta solución propuesta a este nodo estarán suscritos el *Software Propagador de Averías* y todos los *Receptores de Averías* que estén asignados a la zona donde se produzca esta avería.

En un entorno real, los equipos de mantenimiento que estén encargados del mantenimiento correctivo de los transportadores de la nave de pintura estarían suscritos a todos los nodos por debajo jerárquicamente de los transportadores de la nave de pintura, suscribiéndose al nodo:

"Averias/Nave Pintura/Transportadores/#"

El *Software Propagador de Averías*, al recibir este mensaje realiza un apunte en la base de datos del sistema con la aparición de la avería y el sello de tiempo con el instante en el que la avería ha aparecido y los equipos *Receptores de Averías* suscritos a este nodo muestran al técnico de mantenimiento la avería producida.

2.2.4.2. ACUSE DE RECEPCIÓN DE AVERÍA

El técnico de mantenimiento asignado al equipo *Receptor de Averías Smartphone 1* realiza el acuse de recibo, con lo que el equipo *Smartphone 1* envía al mismo tópico por el que ha recibido la avería un mensaje "*2-Smartphone 1"* que es recibido por el *broker* y es enviado por dicho *broker* a todos los suscriptores de este nodo, que al ser el mismo nodo serán los mismos que han recibido la avería (Tabla 2).

EL USUARIO DEL EQUIPO DE RECEPCION DE AVERIAS SMARTPHONE 1 ACUSA LA AVERIA			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Smartphone 1 (equipo de recepción de averías)	Averias/Nave Pintura/Transportadores/GF1/Zona 1/CR1/Tiempo Transito B1	2-Smartphone 1	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....

Tabla 2 Acuse de avería.

El *Software Propagador de Averías* incluye en el registro de esta avería abierta el sello de tiempo del acuse de la avería y el técnico asociado al *Receptor de Averías Smartphone* 1, que ha acusado la avería.

Por otro lado, todos los *Receptores de Averías* reciben el acuse de recibo de la avería y cambian el estado de la avería a "*Acusada"*.

2.2.4.3. PETICIÓN INFORMACIÓN SOBRE LA AVERÍA

El usuario del *Receptor de Averías Smartphone 2* requiere información extra sobre la actuación sugerida ante esta avería, para ello el *Receptor de Averías* envía un mensaje "*3-actuacion"* sobre el mismo tópico que ha recibido la avería, este mensaje es recibido por el *broker* y enviado por dicho *broker* a todos los suscriptores del nodo de la avería. En esta ocasión al ser un mensaje de tipo "*3"* es desechado por los *Receptores de Averías*.

El *Software Propagador de Averías* busca en base de datos la información requerida y envía esta información como el contenido de un mensaje que envía al tópico "*Informacion/Smartphone 2/actuacion"*. Este mensaje es recibido por el *broker* y lo envía a los participantes suscritos a este tópico, que en esta ocasión es únicamente el equipo *Receptor de Averías Smartphone 2*, que lo muestra para que el técnico que ha pedido la información pueda consultarlo (Tabla 3).

EL USUARIO DEL EQUIPO DE RECEPCION DE AVERIAS SMARTPHONE 2 PIDE INFORMACION SOBRE LA ACTUACION DE LA AVERIA			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Smartphone 2 (equipo de recepción de averías)	Averias/Nave Pintura/Transportadores/GF1/Zona 1/CR1/Tiempo Transito B1	3-actuacion	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....
EL SERVIDOR DE PROPAGADOR DE AVERIAS ENVIA LA INFORMACION REQUERIDA AL EQUIPO QUE LO HA PEDIDO			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Software Propagador de Averías	Informacion/Smartphone 2/actuacion	Descripción de la actuación	Smartphone 2 (equipo de recepción de averías)

Tabla 3 Petición de información.

2.2.4.4. RESOLUCIÓN DE LA AVERÍA

La avería *Tiempo Transito B1* en el *GF1* desaparece, automáticamente el controlador de automatización de *GF1* publica un mensaje al mismo tópico en el que publicó el mensaje de la aparición de la avería que es recibido por el *broker* que lo envía a todos los equipos suscritos a este nodo (Tabla 4), el *Software Propagador de Averías* y los *Receptores de Averías* correspondientes.

SE SOLUCIONA Y RESETEA LA AVERIA DE TIEMPO DE TRANSITO A B1 EN EL ELEMENTO CR1 DEL GRUPO FUNCIONAL 1 DE TRANSPORTADORES DE LA NAVE DE PINTURA			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Sistema automatización "GF1" que controla "CR1"	Averias/Nave Pintura/Transportadores/GF1/Zona 1/CR1/Tiempo Transito B1	0	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....

Tabla 4 Resolución Avería.

El S*oftware Propagador de Averías* borra la avería de la tabla de averías abiertas y añade el registro de esta avería en la tabla de averías cerradas, añadiendo el sello de tiempo del instante en el que la avería se resolvió.

En la siguiente tabla (Tabla 5) tenemos otro posible intercambio de mensajes completo sobre la aparición, petición de información, acuse y desaparición de la avería *Térmico* del *Motor 1* de la *Maquina de llenado 1* de *Montaje Final.*

SE PRODUCE LA AVERIA DE TERMICO EN EL ELEMENTO MOTOR 1 DE LA MAQUINA DE LLENADO 1 DE MONTAJE FINAL			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Maquina de llenado 1	Averias/Montaje Final/Maquina de llenado 1/Zona 1/Motor 1/Termico	1	Software Propagador de Averías
			Smartphone 1 (equipo de recepcion de averias)
			Smartphone 2 (equipo de recepcion de averias)
			.....
EL USUARIO DEL EQUIPO DE RECEPCION DE AVERIAS SMARTPHONE 2 PIDE TODA LA INFORMACION DE LA AVERIA			
Publicador	**Topico**	**Mensaje**	**Suscriptores**
Smartphone 2 (equipo de recepcion de averías)	Averias/Montaje Final/Maquina de llenado 1/Zona 1/Motor 1/Termico	3-todo	Software Propagador de Averías
			Smartphone 1 (equipo de recepcion de averias)
			Smartphone 2 (equipo de recepcion de averias)
			.....
EL SERVIDOR DE PROPAGADOR DE AVERIAS ENVIA LA INFORMACION REQUERIDA AL EQUIPO QUE LO HA PEDIDO			
Publicador	**Topico**	**Mensaje**	**Suscriptores**
Software Propagador de Averías	Informacion/Smartphone 2/mensaje	Mensaje avería	Smartphone 2 (equipo de recepcion de averias)
Software Propagador de Averías	Informacion/Smartphone 2/descripcion	Descripcion avería	Smartphone 2 (equipo de recepcion de averias)
Software Propagador de Averías	Informacion/Smartphone 2/actuacion	Actuación avería	Smartphone 2 (equipo de recepción de averías)
Software Propagador de Averías	Información/Smartphone 2/prioridad	0	Smartphone 2 (equipo de recepción de averías)
EL USUARIO DEL EQUIPO DE RECEPCION DE AVERIAS SMARTPHONE 1 ACUSA LA AVERIA			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Smartphone 1 (equipo de recepción de averías)	Averias/Montaje Final/Maquina de llenado 1/Zona 1/Motor 1/Termico	2	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....
SE SOLUCIONA Y RESETEA LA AVERIA DE DE TERMICO EN EL ELEMENTO MOTOR 1 DE LA MAQUINA DE LLENADO 1 DE MONTAJE FINAL			
Publicador	**Tópico**	**Mensaje**	**Suscriptores**
Maquina de llenado 1	Averias/Montaje Final/Maquina de llenado 1/Zona 1/Motor 1/Termico	0	Software Propagador de Averías
			Smartphone 1 (equipo de recepción de averías)
			Smartphone 2 (equipo de recepción de averías)
			.....

Tabla 5 Intercambio de mensajes ejemplo 2.

3. VALIDACIÓN EXPERIMENTAL

Para la validación experimental de la solución propuesta en este punto se desarrollan los diferentes componentes del sistema para su testeo. Así se desarrollará un servidor, que contendrá un SGBD con la base de datos del sistema, el *broker* MQTT y el software específico de la instalación.

Se desarrollarán tres generadores de averías, uno basado en un PLC, otro basado en un microPC embebido y otro basado en un microcontrolador, pretendiendo demostrar de ese modo la versatilidad del sistema, pudiéndose implementar en una amplia gama de dispositivos controladores de automatización actuales y futuros.

Por último, se desarrollará el software necesario para un receptor de averías, en este caso se desarrollará este software para su ejecución en un equipo con sistema operativo Android ®, ya que existe una gran cantidad de dispositivos móviles con este sistema operativo tanto de propósito general como dispositivos portátiles industriales.

3.1. SERVIDOR DE PROPAGACIÓN DE AVERÍAS

Para el desarrollo de este prototipo se utilizará una maquina con un sistema operativo Ubuntu Server 16.0.4, nuestro sistema prototipo no estará ni redundado, ni balanceado, no siendo necesario para este prototipo en el ámbito académico. En un entorno real, y teniendo en cuenta la importancia de la disponibilidad 24x7 de este tipo de sistemas se debería plantear la utilización de varios servidores redundantes para la ejecución de este tipo de sistemas.

Este sistema es el que nos permitirá implementar los componentes descritos en el punto 2.2.1.

3.1.1. BROKER MQTT

En la actualidad existen diversos *brokers* de libre distribución, Mosquitto (https://mosquitto.org/), RabbitMQ (https://www.rabbitmq.com/), EMQ (http://emqtt.io/), etc.

Para seleccionar qué *broker* vamos a utilizar para nuestro sistema de prueba, debemos definir cómo será nuestro sistema de prueba. Nuestro sistema de prueba no estará redundado ni balanceado, la autentificación de nuestros publicadores / suscriptores será por usuario y contraseña no por certificado con TLS, utilizaremos QoS 2 y tendremos menos de 10.000 clientes.

Estas especificaciones eliminan la posibilidad de utilizar RabbitMQ ya que no implementa nivel 2 en la calidad de servicio.

De entre los dos *brokers* restantes elegiremos Mosquitto ya que cumple con los requisitos que necesitamos para la realización de este sistema de prueba y existe una más amplia comunidad activa sobre este *broker*, que en el caso del *broker* EMQ, donde se podrá encontrar apoyo para su instalación, configuración, etc…

Para realizar la instalación del *broker Mosquitto* ejecutaremos los comandos siguientes:

```
# wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
# sudo apt-key add mosquitto-repo.gpg.key
# cd /etc/apt/sources.list.d/
# sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.list
# sudo apt-get update
# sudo apt-get install mosquitto
```

Configuraremos la seguridad en el servidor para tener que atacar al servidor con un usuario definido, para ello lo primero que debemos hacer es parar el servicio:

```
# sudo stop mosquitto
```

Una vez hecho esto añadiremos un usuario en archivo de user/pasword del mosquitto, para ello ejecutaremos la instrucción

```
# sudo mosquitto_passwd -c /etc/mosquitto/password_mosquitto eduardo
```

El nombre de nuestro usuario será “eduardo” y nos pedirá que introduzcamos el password donde introduciremos para nuestro test “asdf1234”.

Modificaremos el fichero de configuración del mosquitto para que no admita conexiones anónimas y busque las contraseñas y password en el fichero que acabamos de modificar. Para ello modificaremos el fichero *mosquitto.conf* añadiéndoles las siguientes dos líneas:

```
password_file /etc/mosquitto/password_mosquitto
allow_anonymous false
```

Para el destino académico de este prototipo se considera suficiente la utilización de un usuario y contraseña para securizar el acceso al *broker* MQTT, pero en un entorno productivo real, se debería considerar la utilización de seguridad mediante certificados digitales y protocolo SSL para el acceso a este *broker*. A pesar de poderse considerar las redes internas de una empresa como un entorno seguro, siguiendo la filosofía de *defensa en profundidad* deberemos añadir capas de seguridad para proteger nuestro sistema.

También configuraremos la persistencia de los mensajes del siguiente modo:

```
persistence true
persistence_location /var/lib/mosquitto
persistence_client_expiration 5h
```

Para que nuestro *broker* almacene los mensajes que no han podido ser entregados para reintentar su entrega cuando el cliente se conecte al menos de todos los mensajes no entregados durante 5 horas.

Una vez hecho esto volveremos a arrancar el servicio mosquitto con la siguiente instrucción:

```
# sudo service mosquitto start
```

Como no hemos configurado ningún puerto de conexión el servicio se ejecutará por el puerto por defecto que es 1883.

3.1.2. SGBD

En este prototipo, el sistema de gestión de bases de datos (en adelante SGBD) contendrá la lista de averías, con todos sus datos, y almacenará las incidencias relacionadas con cada avería, aparición y desaparición de cada una de ellas, acuse por parte del personal de mantenimiento correctivo, etc...

Estos datos se deben almacenar para la generación de informes de análisis de las averías que ayudan a planificar el mantenimiento preventivo, modificar los protocolos de actuación del personal de mantenimiento correctivo, etc.. Por otro lado, estos datos almacenados también se utilizarán para un análisis forense de una avería (o cadena de ellas) concreta.

Existen diversos SGBD de libre distribución, los dos más utilizados y con una mayor comunidad son MySQL y PostgreSQL. Ambos cumplirían perfectamente las funciones necesarias para la generación del sistema prototipo que se desarrolla en el presente documento.

El autor de este documento se decanta por la utilización de PostgreSQL por su conocimiento personal, aunque con cualquiera de los dos se podría realizar dicho prototipo.

Para la instalación de este SGBD sobre Ubuntu únicamente tenemos que instalarlo desde el repositorio:

```
# sudo apt-get install postgresql postgresql-contrib
```

Una vez tengamos instalado el PostgreSQL crearemos un nuevo usuario para la gestión de la BBDD del prototipo.

Para entrar en el servidor lo haremos con permisos de superusuario y entraremos con el usuario postgres (administrador por defecto del SGBD).

```
# sudo -u postgres psql
```

Una vez dentro del PostgreSQL crearemos un nuevo usuario y le daremos rol de superusuario para poder crear la BBDD con este usuario, y cambiaremos el usuario activo:

```
postgres=# CREATE USER ebuetas PASSWORD ‘EBuetas78’;
postgres=# ALTER ROLE ebuetas WITH SUPERUSER;
postgres=# SET SESSION AUTHORIZATION ebuetas;
```

El siguiente paso será crear la BBDD “averias_tfm”:

```
postgres=# CREATE DATABASE "averias_tfm"
postgres=# WITH OWNER = eduardo
postgres=# ENCODING = 'UTF8'
postgres=# TABLESPACE = pg_default;
```

Una vez tengamos creada la base de datos deberemos crear las tablas requeridas para el funcionamiento de esta prueba (Fig. 27), se adjunta a esta entrega el fichero *“averias_tfm.sql”*

que contiene el script completo para la creación de esta base de datos, para ejecutar este script en el SGBD, deberemos ejecutarlo dentro de la base de datos generada:

```
postgres=# \c averias_tfm
postgres=# \i /home/ebuetas/averias_tfm.sql
```

Este script además de crear la estructura de la base de datos creará los registros en las tablas necesarios para la realización de las pruebas de funcionamiento del sistema, en la tabla *m_averias* incluimos las averías mostradas en la Tabla 6.

area	subarea	sistema	zona	elemento	nombre	mensaje	descripcion	actuacion	id_tipo_averia	prioridad
Pintura	Transportadores	GF1	Cota 0	CR1	Tiempo Tansito B1	Se ha superado el tiempo de transito para la llegada al B1	El tiempo de transito del detector mide el tiempo entre que el skid sale del detector anterior y llega al detector actual, si supera un tiempo dado se produce este error	Comprobar si el skid patina sobre los rodillos o bien si esta enganchado mecánicamente o bien si el motor de arrastre tiene algún problema	1	0
Pintura	Transportadores	GF1	Cota 1	MG2	Termico Motor 1	Se ha producido el disparo de la protección térmica del motor 1	La protección térmica del motor se ha disparado por una sobretensión producida en el motor.	Comprobar que el motor no esta bloqueado mecánicamente.	1	0
Pintura	Transportadores	GF1	Mqtt	Mqtt	Conexión	Se ha producido un error de conexión en el equipo	El cliente MQTT del equipo se ha desconectado del broker, el equipo no enviara averias hasta que se vuelva a conectar	Comprobar que el equipo esta activo y la conexión de red con el sistema es correcta	1	0
Pintura	Maquinas	Laca 1	Zona 1	ESTATICA1	Fallo Boquilla 5	El controlador de la boquilla 5 ha dada un fallo	El controlador de la boquilla, controla el nivel de alta tensión de la boquilla así como las revoluciones a las que gira, el fallo puede deberse a un problema en la tensión o bien en el motor de giro de la misma.	Comprobar que el sistema de alta tensión esta funcionando correctamente y comprobar que el giro de la boquilla no esta bloqueada	1	0
Pintura	Maquinas	Laca 1	Mqtt	Mqtt	Conexión	Se ha producido un error de conexión en el equipo	El cliente MQTT del equipo se ha desconectado del broker, el equipo no enviara averias hasta que se vuelva a conectar	Comprobar que el equipo esta activo y la conexión de red con el sistema es correcta	1	0
Chapa	Transportadores	GF1	Cota 0	CR1	Fallo Ciclo B1	Se ha detectado el detector B1 fuera del cido correspondiente	El detector debe ser activado únicamente en las etapas del ciclo designadas, si se detecta en un momento diferentes se activa este error.	Comprobar que el camino de rodillos no esta fuera de ciclo, vaciar la mesa y ciclarla en estado vacio.	1	0
Chapa	Transportadores	GF1	Mqtt	Mqtt	Conexión	Se ha producido un error de conexión en el equipo	El cliente MQTT del equipo se ha desconectado del broker, el equipo no enviara averias hasta que se vuelva a conectar	Comprobar que el equipo esta activo y la conexión de red con el sistema es correcta	1	0

Tabla 6 Averías incluidas para las pruebas del prototipo.

También rellenará las tablas auxiliares para el correcto funcionamiento del prototipo, *m_tipos_averia* en Tabla 7.

nombre	descripcion
Digital	Las averías digitales se lanzan por el cambio de un valor digital (1 o 0)
Analogica	Las averías analogicas se lanzan cuando un valor entero supera un valor o pasa por debajo de el

Tabla 7 Tipos de avería incluidas para las pruebas del prototipo

Rellenará la tabla *m_tecnicos* (Tabla 8) con dos técnicos para realizar las pruebas:

nombre	apellido1	apellido2	id_number
Edaurdo	Buetas	Sanjuan	18046256L
Juan	Garcia	Garces	98345212Z

Tabla 8 Técnicos incluidos para las pruebas del prototipo.

Por último, rellenará la tabla *m_receptores_aveiras* (Tabla 9) con dos dispositivos de control para las pruebas:

id_tecnico	topic_receptor
1	Smartphone 1
2	Smartphone 2

Tabla 9 Receptores de averías configurados para las pruebas del prototipo.

Con estos datos se realizarán las pruebas de funcionamiento del prototipo descrito en este punto.

3.1.3. SOFTWARE DE PROPAGACIÓN DE AVERÍAS

Para este prototipo se ha desarrollado este software en java, utilizando para su desarrollo el IDE de libre distribución NetBeans 8.1. Podemos descargar este software de su sitio web oficial (https://netbeans.org/downloads/index.html).

A continuación, se describirá el software generado con este propósito, además el proyecto de este software debidamente comentado se entregará conjuntamente con este documento, estando el código debidamente comentado para su fácil comprensión, además de entregarse también la documentación autogenerada por medio del generador de documentación JavaDoc.

El código de este programa está organizado en tres paquetes diferentes:

EBuetas.TFM: con la función de ejecución del hilo principal del programa y una clase auxiliar para almacenar los errores en un log de texto del programa.

EBuetas.TFM.BBDD: que contiene las clases necesarias para trabajar con la base de datos del sistema.

EBuetas.TFM.MQTT: en este paquete se crean las clases para el manejo de las comunicaciones con el protocolo MQTT.

3.1.3.1. CLASE EBuetas.TFM.PropagadorAverias

Esta clase (Fig. 29) definida en el fichero *PropagadorAverias.java* contiene únicamente la función *main* que será ejecutada como hilo principal del programa al ser ejecutado.

En esta función se creará el objeto de conexión al *broker* MQTT, activará el suscriptor y el publicador que son necesarios para el cumplimiento de los objetivos de este programa. Una vez activados el suscriptor y el publicador esta función se quedará abierta en un bucle comprobando cada minuto que la conexión sigue siendo correcta, en el momento que la conexión caiga se volverá a intentar la conexión cada cierto tiempo.

El usuario y password de conexión para el *broker* MQTT se codifican de manera directa en el código de este programa. Para los fines académicos de este trabajo se considera suficiente este modo de codificación, para una aplicación industrial de este software y a pesar de poder considerarse un entorno seguro en el que este software se va a ejecutar, se debería

cambiar el tratamiento de la seguridad, para acceder al servidor MQTT a través de un certificado SSL y almacenando el usuario y la contraseña en base de datos debidamente encriptada, aplicando los conceptos de seguridad necesarios para la creación de un software seguro y resiliente.

El servidor tendrá un programa *watchdog* (analizado en el punto 3.1.4) que se ejecutará cada cinco minutos, de este modo si el programa se cerrara por cualquier motivo, se volverá a ejecutar volviendo a intentar realizar la conexión con el *broker* MQTT.

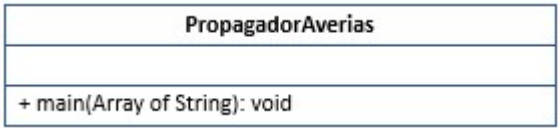

Fig. 29 Clase "PropagadorAverias".

3.1.3.2. CLASE EBuetas.TFM.EscribeLog

Esta clase (Fig. 30), definida en el fichero *EscribeLog.java,* únicamente tendrá una función estática que se encargará de escribir en el log de la aplicación las diferentes incidencias que vayan ocurriendo en el transcurso de la ejecución del programa.

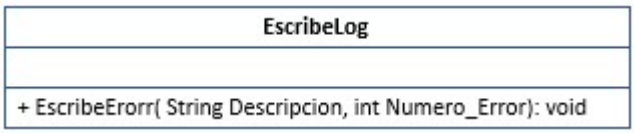

Fig. 30 Clase "EscribeLog".

Al intentar escribir un apunte en el log, si el fichero no ha sido creado esta clase lo creará y añadirá el primer apunte del log, en caso contrario añadirá los nuevos apuntes al final del fichero.

El nombre del fichero del log no es configurable y siempre será *"/var/log/PropagacionAverias.log"*.

Cada apunte además del texto del mensaje incluirá un número identificativo que identifica el error de manera unívoca, lo cual facilita encontrar en el código la zona donde se generó dicho error y el sello de tiempo del instante en el que se produjo la incidencia que provocó la generación del apunte en el log (Fig. 31).

```
Servicio de propagacion de averias activado--Nº Error:0--Hora:10/05/2018 01:32:55
Se ha perdido la conexión--Se ha perdido la conexión--Nº Error:310--Hora:10/05/2018 01:33:35
Servicio de propagacion de averias detenido por fallo de conexion con broker MQTT--Nº Error:1--Hora:10/05/2018 01:34:10
Servicio de propagacion de averias activado--Nº Error:0--Hora:10/05/2018 10:48:59
ERROR: insert or update on table "averias_abiertas" violates foreign key constraint "id_averia_a_abierta_fk"
  Detail: Key (id_averia)=(0) is not present in table "m_averias".--Nº Error:131--Hora:10/05/2018 10:49:32
Insercion de averia no valida:
Averias/Nave Pintura/Transportadores/GF1/Zona 1/CR1/B5 1--Nº Error:323--Hora:10/05/2018 10:49:33
```

Fig. 31 Ejemplo fichero log PropagadorAverias.

3.1.3.3. CLASE EBuetas.TFM.BBDD.Datos_m_averias

Como podemos observar en Fig. 32 esta clase tiene una variable privada y su correspondiente función *get* y *set* para cada uno de los campos de la tabla *m_averias* definida en la Fig. 27. Esta clase está codificada en el fichero *Datos_m_averias.java*.

Datos_m_averias
- id_averia: long - area: String - subarea: String - sistema: String - zona: String - elemento: String - prioridad: int - nombre: String - mensaje: String - descripcion: String - actuacion: String - id_tipoaveria: long
+ Datos_m_averias() + Datos_m_averias(String _area, String _subarea, String _sistema, String _zona, String _elemento, String _nombre) + getId_averia (): long + getArea (): String + getSubarea (): String + getSistema (): String + getZona (): String + getElemnto (): String + getNombre (): String + getMensaje (): String + getDescripcion (): String + getActuacion (): String + getId_tipoaveria (): long + setId_averia (long _averia): void + setArea (String _area): void + setSubarea (String _subarea): void + setSistema (String _sistema): void + setZona (String _zona): void + setElemnto (String _elemento): void + setNombre (String _nombre): void + setMensaje (String _mensaje): void + setDescripcion (String _descripcion): void + setActuacion (String _actuacion): void + setId_tipoaveria (long _id_tipo_averia): void + LeeDatos(): boolean

Fig. 32 Clase "Datos_m_averias".

Además, incluye una función llamada *LeeDatos*, esta función tiene por objeto que, una vez rellenados los datos que se obtienen a partir del tópico de recepción de la avería y mediante una consulta a la base de datos realizada con la clase *BBDD_Averias* (punto 3.1.3.4) se rellenen el resto de datos relativos a esta avería en el objeto de la clase.

Esta función devuelve un booleano que nos indicará si la función ha terminado con éxito, en caso contrario además de devolver false, se realizará un apunte en el log del programa.

3.1.3.4. CLASE EBuetas.TFM.BBDD.BBDD_Averias

Esta clase, codificada en el fichero BBDD_Averias.java, se encargará de la conexión, desconexión y generación de consultas contra la base de datos "*averias_tfm"* de control del sistema.

Para la conexión con la base de datos se utilizará el driver *jdbc* "*org.postgresql.Driver"* que está incluido en el fichero *"postgresql-9.4.1208.jre6.jar"* que podemos descargar de la página oficial de PostgreSQL (https://jdbc.postgresql.org/download.html).

Todas las consultas están realizadas mediante consultas preparadas y no mediante consultas dinámicas para minimizar el riesgo de vulnerabilidades de inyección SQL [14]. Para ello se utilizan funciones *PreparedStatement* para la ejecución de las consultas (Fig. 33).

```
/**
 * Funcion que rellena el momento del acuse de recibo por parte de un tecnico y el id que
 * identifica a dicho tecnico en la tabla de averias abiertas
 * @param dato objeto de la clase Datos_m_averias con los datos de la averia a acusar
 * @param equipo nombre del equipo que envia el acuse, a partir de este equipo se conocera el tecnico
 *               que acusa la averia
 * @return Valor Booleano que nos devuelve true si el metodo ha tenido exito
 *         y false si no lo ha tenido
 */
public boolean AcuseAveria (Datos_m_averias dato,String equipo){
    boolean respuesta=false;
    PreparedStatement  Update_Averias_Abiertas=null;
    try{
        String cadenasql="";

        cadenasql="UPDATE averias_abiertas SET ts_acuse=?,id_tecnico_acusa=(select id_tecnico from m_receptores_averias where topic_receptor=?) where id_averia=?";

        Update_Averias_Abiertas=con.prepareStatement(cadenasql);
        Update_Averias_Abiertas.setTimestamp(1, new Timestamp(System.currentTimeMillis()));
        Update_Averias_Abiertas.setString(2, equipo);
        Update_Averias_Abiertas.setLong(3, dato.getId_averia());
        Update_Averias_Abiertas.execute();
        respuesta=true;
    }catch(SQLException ex){
        EscribeLog.EscribeError(ex.getMessage(), 141);
    }catch(Exception ex){
        EscribeLog.EscribeError(ex.getMessage(), 142);

    }finally{
        if (Update_Averias_Abiertas!=null){
            try{
                if (!Update_Averias_Abiertas.isClosed()){
                    Update_Averias_Abiertas.close();
                }
            }catch(SQLException ex){
                EscribeLog.EscribeError(ex.getMessage(), 143);
            }
        }
    }

    return respuesta;
}
```

Fig. 33 Ejemplo consulta con PreparedStatement.

En esta clase se realizan todas las consultas necesarias para el funcionamiento del programa.

Todas las funciones trabajan a partir de un objeto parámetro de entrada de la clase *Datos_m_averias*, tomando los datos necesarios para realizar la consulta de este objeto y rellenando el resultado, si es que lo hubiera, en este objeto.

Como respuesta a estas funciones tenemos un booleano que nos devolverá true si la función se ha ejecutado con éxito y false en caso contrario.

Si la función ha terminado con un error además de la devolución errónea se realizará un apunte en el log del programa que nos indicará cual ha sido el error generado.

Podemos ver la estructura de la clase en la Fig. 34.

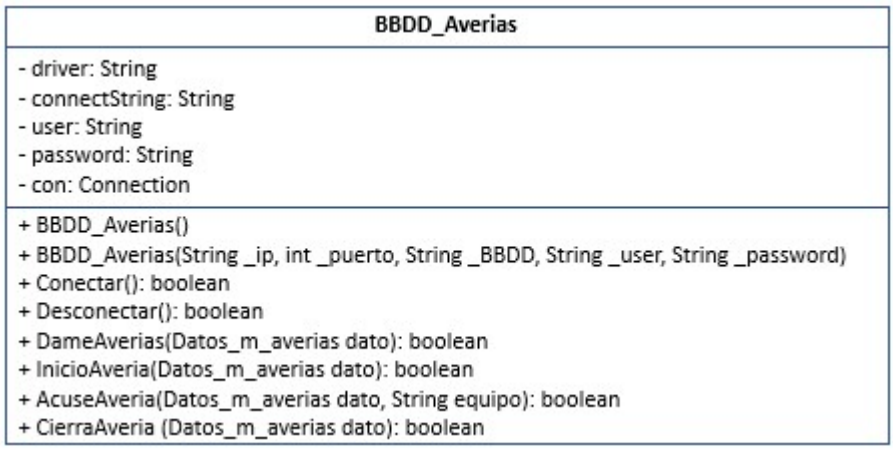

Fig. 34 Clase "BBDD_Averias".

Además, esta clase implementa la función *Conectar* que crea el objeto de conexión a la base de datos y lo guarda en la variable privada de la clase *con* que después es utilizada para la ejecución de las consultas en las funciones de consulta.

También implementa la función *Desconectar*, que desconecta el objeto de conexión (*con)*.

3.1.3.5. CLASE EBuetas.TFM.MQTTSus_Pub

En esta clase se implementa el publicador y el suscriptor MQTT, pieza fundamental del funcionamiento deseado de nuestro programa, se codifica en el fichero MQTTSus_Pub.java.

Para la implementación de estas funciones se utiliza las clases proporcionadas por la fundación *Eclipse (https://www.eclipse.org/)* en su proyecto *Paho (https://www.eclipse.org/paho/)* , este es un proyecto de la fundación Eclipse que intenta facilitar

la implementación del protocolo MQTT en aplicaciones nuevas, preexistentes o emergentes en el ámbito de la IoT.

Proporciona librerías open-source para la implementación de MQTT en diferentes plataformas y existe una amplia y activa comunidad trabajando en este proyecto.

Para la utilización de este proyecto en nuestra aplicación debemos descargarnos sus librerías que están contenidas en el archivo *"mqtt-client-0.4.0.jar"* que lo podremos descargar del repositorio oficial del proyecto (https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/mqtt-client/0.4.0/) .

Una vez descargado este archivo jar e incluido como librería de nuestro proyecto podemos utilizar los objetos de esta librería para crear nuestra clase.

Esta clase contendrá dos objetos cliente MQTT, uno suscriptor, del cual se implementarán las funciones *callback* en la misma clase y un objeto publicador, podemos ver la estructura de la clase en la Fig. 35.

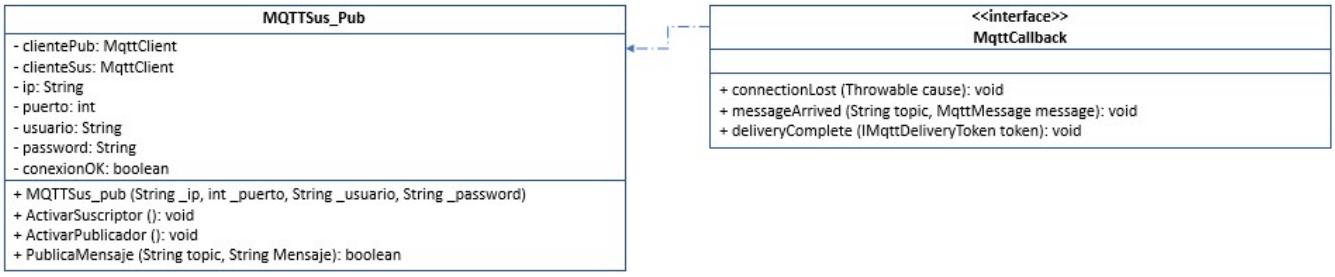

Fig. 35 Clase "MQTTSus_Pub".

Una vez creado el objeto de la clase con el constructor deberemos activar los dos clientes MQTT, para ello ejecutaremos las funciones *ActivarSuscriptor* en la cual se creará el cliente suscriptor de la aplicación, que se suscribirá al tópico "*Averias/#*" para recibir todos los mensajes relativos a las averías del sistema y se asociarán las funciones *callback* del objeto al propio objeto de la clase, para recibir las notificaciones de mensajes y de conexión y la función *ActivarPublicador* que creará el cliente publicador que se utilizará para el envío de mensajes bajo demanda a los diferentes receptores de averías del sistema.

Si las dos funciones tienen por resultado una conexión correcta, la variable *conexionOK* del objeto será true, de lo contrario será false.

Desde este momento y hasta que no tengamos un evento *connectionLost* de nuestro suscriptor el programa estará a la espera de la recepción de mensajes por parte del suscriptor.

Tenemos también en esta clase la función *PublicaMensanje* que publica un mensaje por el publicador creado en la función *ActivarPublicador* con el tópico y el mensaje que se le pasa como parámetros. Si hay un error en la publicación del mensaje, la función intenta reiniciar la conexión del publicador volviendo a ejecutar la función *ActivaPublicador.*

La función sobrescrita de la interface *MqttCallback messageArrived* es la función que lleva el peso del desarrollo normal de la aplicación, esta función recibe todos los mensajes que tienen que ver con las averías del sistema.

Lo primero que se realiza cuando esta función recibe un mensaje es examinar su tópico para comprobar que este tiene los siete componentes obligatorios que debe tener todo tópico recibido por nuestro programa:

"Averias / Area / Subarea / Sistema / Zona / Elemento / Nombre Averia"

Si no tiene estas siete partes el sistema desecha el mensaje y añade una línea a log advirtiendo de la llegada de un mensaje mal formateado.

Una vez hecho esto, y sabiendo que el tópico tiene las siete partes requeridas, el programa las descompone en sus siete componentes, crea un objeto de la clase *Datos_m_averias* y lee de la base de datos todos los datos de la avería a través de la función *LeeDatos* del objeto creado.

Con estos datos el programa actúa en consecuencia dependiendo del mensaje recibido:

"0": El programa cierra la avería, borrándola de la tabla *averias_abiertas* pasándola a la tabla *averias_cerradas*, añadiendo al registro el sello de tiempo del instante de finalización de la avería, para ello se ejecutará la función *CierraAveria* de un objeto de la clase *BBDD_Averias*.

"1": El programa abre la avería, si la avería no está en la tabla *averias_abiertas*, para ello crea un registro con el id de la avería recibida y el sello de tiempo correspondiente al inicio de la avería. Par realizar esta acción se ejecutará la función *InicioAveria* de un objeto de la clase *BBDD_Averias*. Si la avería ya está en la tabla no hará nada y realizará un apunte en el log de la aplicación.

"2": Se comprobará que el mensaje también contenga una segunda parte que debe contener el origen del acuse de recibo, si no lo tiene se realizará un apunte en el log de la aplicación y no se realizará ninguna otra acción, de lo contrario se realiza el acuse de recibo de la avería, colocando en la tabla *averias_abiertas* el sello de tiempo del instante del acuse, así como la clave identificadora del técnico que ha acusado la avería. Si la avería no existiera en

la tabla *averias_abiertas* no se realizará ninguna acción sobre la base de datos y se realizará un apunte en el log de la aplicación. Para realizar estas acciones se ejecuta la función *AcuseAveria* de un objeto de la clase *BBDD_Averias*.

"3": Se comprobará que, seguido de este indicador de mensaje, tenemos dos parámetros separados por un "-", el equipo peticionario de la información y el tipo de información pedida, si no existiera no se realizará ninguna acción sobre la base de datos y se realizará un apunte en el log de la aplicación. De lo contrario se analiza la petición de información, si es desconocida no se realizará ninguna acción en la base de datos y se realizará un apunte en el log de la aplicación, de lo contrario y con la ayuda de la función *PublicaMensaje* de esta misma clase se realiza la publicación de la/s información/es requeridas al tópico "*Informacion/peticionario/tipo informacion"*.

Si se produce un evento de perdida de comunicación del suscriptor (*connectionLost)* la variable *conexionOK* se convertirá en false lo que provocará un intento de reinicio de la conexión cada cierto tiempo.

3.1.4. WATCHDOG PROPAGADOR DE AVERÍAS

Como programa auxiliar se crea un programa que al ejecutarse se encarga de comprobar si el programa *PropagadorAverias* se encuentra entre los procesos ejecutados por la máquina y si no lo está lo lanza. Este programa tiene por objeto ejecutarse cada cierto tiempo (cinco minutos, por ejemplo) en el servidor que debe tener corriendo el propagador de averías y si por algún motivo este programa no se está ejecutando volver a lanzarlo, con el objetivo de asegurarnos que el programa *PropagadorAverias* siempre se está ejecutando.

Este programa se denominará *ControlServerTFM*, su código se entrega debidamente comentado junto con este documento y también los ficheros de documentación generados de manera automática con *javaDoc*.

Para la ejecución cíclica de este programa se creará una nueva entrada en las tareas cron del servidor.

```
# sudo crontab -e
```

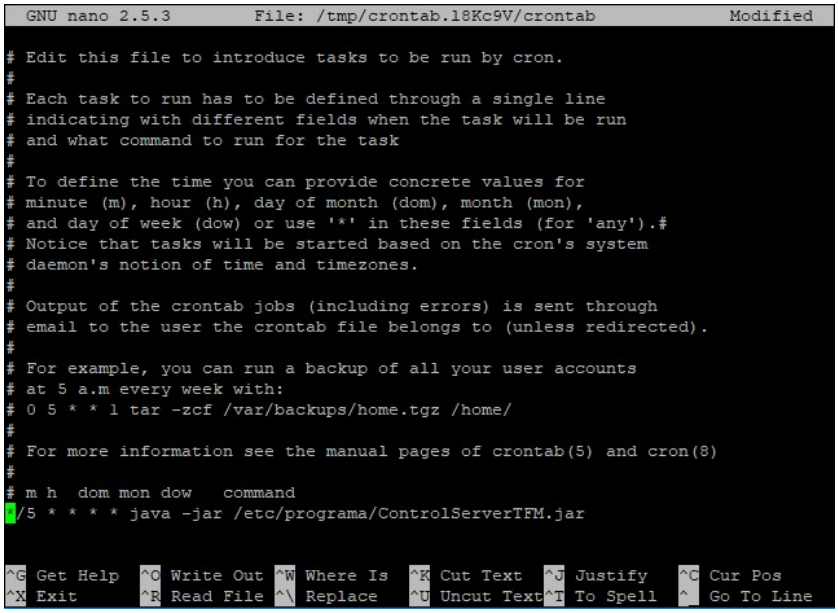

Fig. 36 Creación Cron para ControlPropagacionAverias.

3.2. CONTROLADORES DE AUTOMATIZACIÓN

En nuestro sistema los controladores de automatización, descritos en el punto 2.2.2, serán los publicadores de las averías, cuando una avería aparezca en el sistema este enviará un mensaje MQTT que identificará de manera unívoca la avería dentro de todo el sistema.

Para ello el tópico del mensaje estará compuesto de la siguiente forma:

"Averias/Area/Subarea/Sistema/Zona/Elemento/Nombre Averia"

Donde:

- *Averías:* denominador fijo para todas las averías.
- *Área:* Área general donde se encuentra el sistema generador de la avería.
- *Subárea*: Subárea dentro del área general donde se encuentra el sistema generador de la avería.
- *Sistema:* Sistema generador de la avería.
- *Zona:* Zona dentro del sistema donde se encuentra el elemento del sistema que genera la avería.
- *Elemento:* Elemento dentro del sistema que genera la avería.
- *Nombre Avería:* Nombre de la avería producida.

Cuando la avería aparezca el dato enviado en este tópico será un "*1*" y cuando la avería desaparezca el dato enviado será un "*0*".

Además de las averías propias de la instalación que controle cada controlador de automatización, tendrá una avería para controlar la correcta conexión del controlador con el *broker* MQTT. Para ello se creará una avería con la zona *"Mqtt"* y con el elemento *"Mqtt"* y el nombre de la avería será *"Conexion"*. Cuando el controlador realice una conexión exitosa al *broker* se enviará un mensaje inicial con el tópico: "*Averias/Area/ Subarea/Sistema/Mqtt/Mqtt/Conexion"* y con el contenido del mensaje igual a *"0"*.

En esta conexión se configurará el "*willTopic"* de la conexión con el tópico "*Averias/Area/Subarea/Sistema/Mqtt/Mqtt/Conexion"* y el *"willMessage"* igual a *"1"*. Con esta configuración conseguimos que el *broker* de la instalación cuando detecte una desconexión de este publicador, enviará el mensaje de conexión fallida a los diferentes suscriptores.

Todos los mensajes enviados por estos controladores de automatización se realizarán con la calidad de servicio (QoS) igualada a 2, para asegurarnos de que los mensajes lleguen una y solo una vez a los suscriptores.

Estos publicadores podrán estar basados en múltiples plataformas, siendo el protocolo MQTT un protocolo muy ligero, se podrá integrar un publicador de este protocolo en plataformas con pocos recursos.

Como ejemplos para el prototipo de nuestro sistema se realizarán tres casos de controladores de automatización:

1. Basado en PLC, en concreto en un PLC Siemens S7-1211C (Fig. 37).

Fig. 37 PLC S7-1211C.

2. Basado en MicroPC, utilizando una Raspberry Pi 3 Model B+.(Fig. 38)

Fig. 38 Raspberry Pi 3 Model B+.

3. Basado en Microprocesador, utilizando una placa Arduino MKR1000 (Fig. 39) con conectividad wifi incorporada.

Fig. 39 MKR1000.

Esta selección cubre un gran espectro de posibles sistemas de automatización, y es, una buena prueba de la flexibilidad de utilización en diferentes plataformas para este protocolo.

3.2.1. PLC SIEMENS S7-1211C

El controlador de automatización basado en PLC, simulará ser el controlador del sistema GF1 de la nave de chapistería (Fig. 40), para las pruebas realizadas, configuraremos en este equipo dos de las averías definidas en la base de datos (Tabla 6), con los siguientes tópicos:

"Averias/Chapa/Transportadores/GF1/Cota 0/CR1/Fallo Ciclo B1"

"Averias/Chapa/Transportadores/GF1/Mqtt/Mqtt/Conexion"

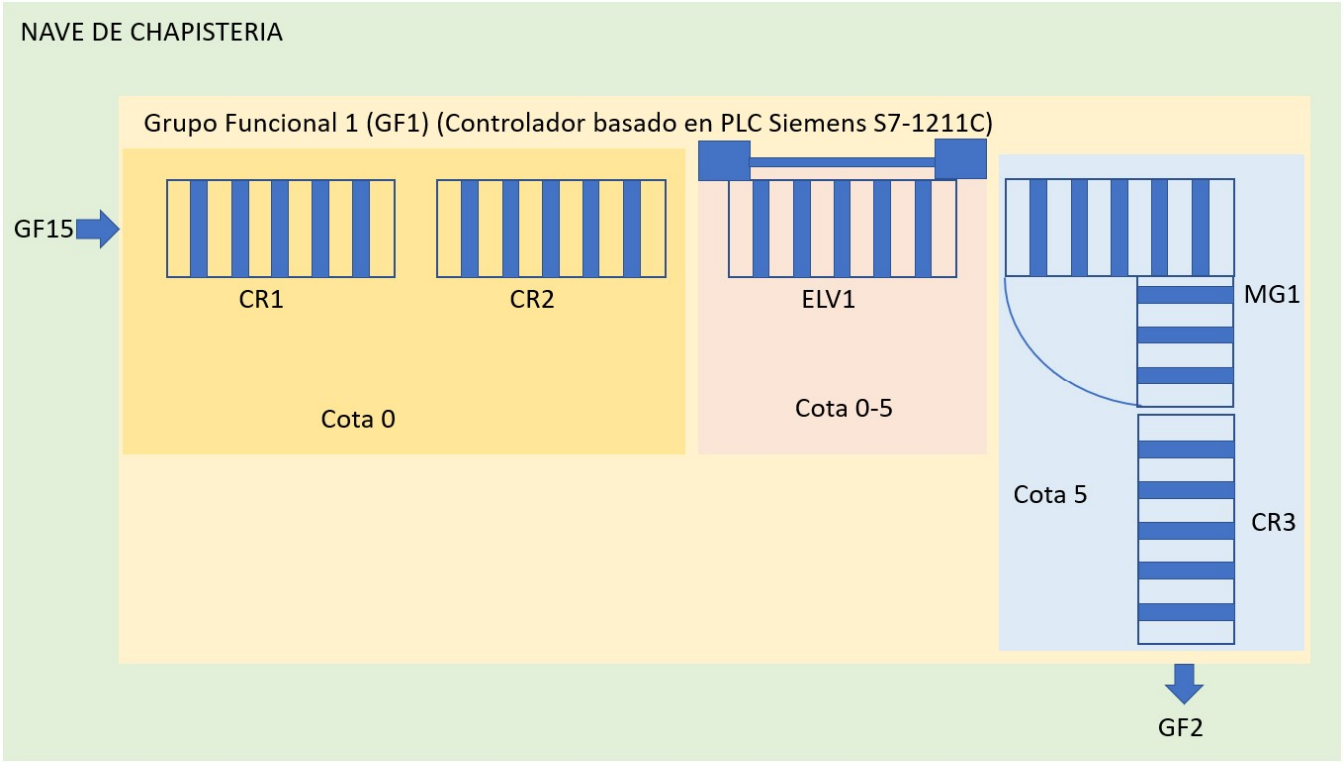

Fig. 40 Sistema automatizado basado en PLC S7-1211C.

La primera de estas averías se simulará con un pulsador y un led, cuando se pulse el pulsador una vez la avería será lanzada y el led se activará, cuando se pulse el pulsador una segunda vez la avería desaparecerá y el led se desactivará.

Para realizar la programación de este dispositivo deberemos tener instalado en nuestro equipo el software para la programación de PLCs S7-1200. En sus últimas versiones todos los software relacionados con la automatización suministrados por Siemens para la programación de sus equipos están integrados en un IDE al que se le denomina *"Totally Integration Automation"* y el software específico para la programación de sus PLCs se denomina *"Step 7 Professional"*. Para la utilización de estos software de forma continuada es necesario la adquisición de una licencia.

Sin embargo, Siemens permite la descarga de este software para su utilización libremente durante 21 días, a partir del siguiente link: https://support.industry.siemens.com/cs/document/109752566/descarga-del-simatic-step-7-y-wincc-v15-de-prueba-(trial)?dti=0&lc=es-ES. (Fig. 41)

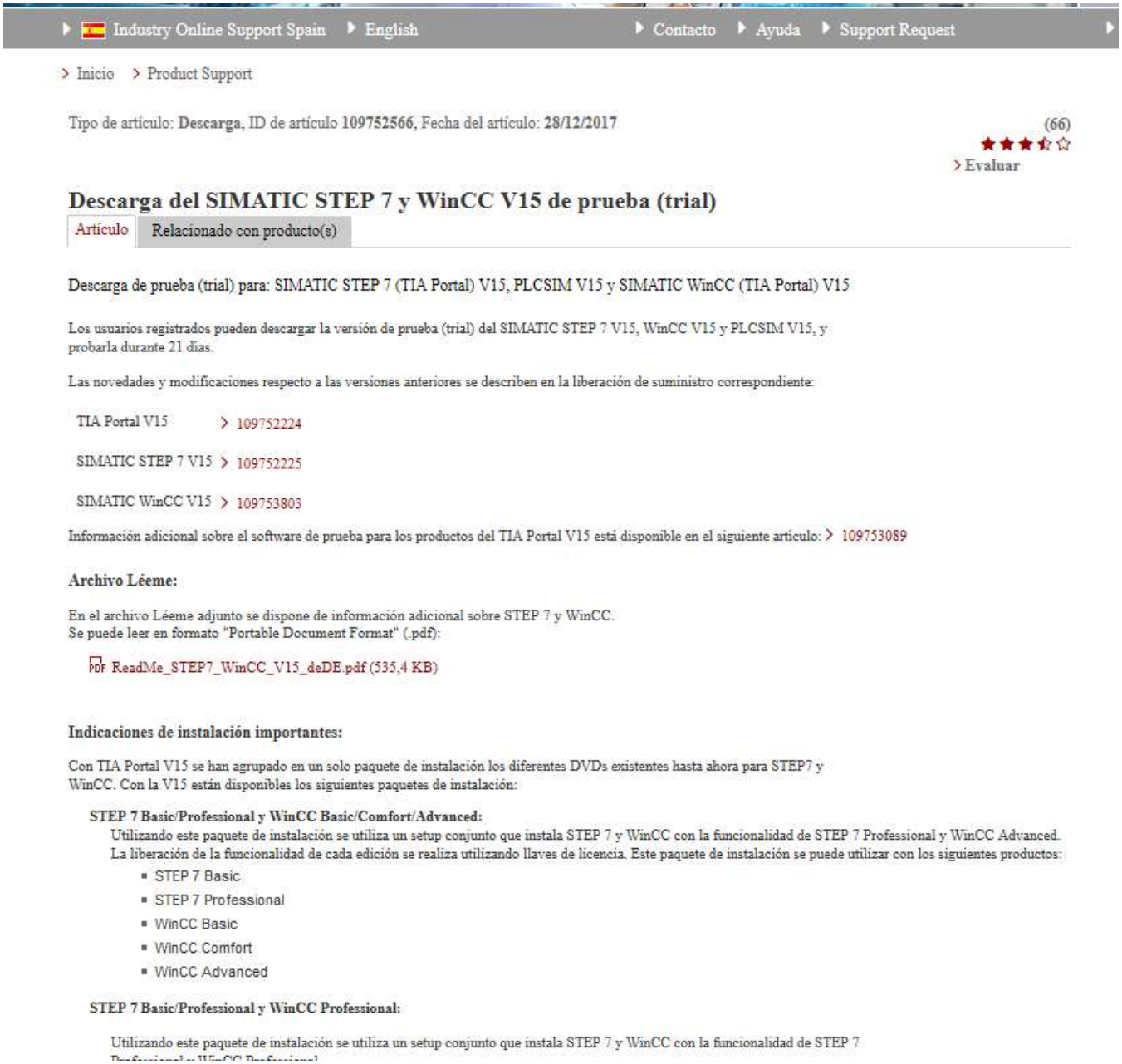

Fig. 41 Descarga Trial Tia Portal Siemens.

Deberemos registrarnos en la página y una vez registrados nos permitirá descargar este software para las pruebas realizadas a lo largo de esta práctica.

Para su instalación seguiremos las instrucciones de instalación dadas por el programa instalador.

Siemens, en uno de sus casos de prueba, proporciona a los usuarios de sus PLCs una librería para la utilización de sus PLCs como clientes MQTT.

Este ejemplo de aplicación es proporcionado como una librería externa *"LMqtt Publisher"* que puede ser incluida en los proyectos. Para ello debemos descargarnos esta

librería del sitio oficial de Siemens (https://support.industry.siemens.com/cs/document/109748872/mqtt-publisher-for-simatic-cpu?dti=0&lc=en-WW) y después incluiremos esta librería en nuestro proyecto de PLC para poder utilizar sus funcionalidades.

Con la inclusión de esta librería tendremos los siguientes módulos a nuestra disposición:

- **typeMqttConnectFlags**: tipo de datos propio del ejemplo de aplicación proporcionado por Siemens, que contendrá los *flags* para la realización de la conexión con el *broker* MQTT (Fig. 42).

typeMqttConnectFlags

	Nombre	Tipo de datos	Valor predet.	Accesible d...	Escrib...	Visible en ...	Valor de a...
1	cleanSession	Bool	false	☑	☑	☑	☐
2	will	Bool	false	☑	☑	☑	☐
3	willQoS_1	Bool	false	☑	☑	☑	☐
4	willQoS_2	Bool	false	☑	☑	☑	☐
5	willRetain	Bool	false	☑	☑	☑	☐
6	password	Bool	false	☑	☑	☑	☐
7	userName	Bool	false	☑	☑	☑	☐

Fig. 42 typeMqttConnectFlags.

- **typeMqttPublishFlags:** tipo de datos propio del ejemplo de aplicación proporcionado por Siemens, que contendrá los *flags* para la publicación de mensajes MQTT (Fig. 43).

typeMqttPublishFlags

	Nombre	Tipo de datos	Valor predet.	Accesible d...	Escrib...	Visible en ...	Valor de a...
1	qualityOfService	Int	0	☑	☑	☑	☐
2	retain	Bool	false	☑	☑	☑	☐

Fig. 43 typeMqttPublishFlags.

- **typeTcpConnParam**: tipo de datos propio del ejemplo de aplicación proporcionado por Siemens, que contendrá los parámetros de la conexión TCP requerida para la conexión con el cliente MQTT (Fig. 44).

typeTcpConnParam

	Nombre	Tipo de datos	Valor predet.	Accesible d...	Escrib...	Visible en ...	Valor de a...	Com
1	hwIdentifier	HW_ANY	0	☑	☑	☑	☐	
2	connectionID	CONN_OUC	16#0	☑	☑	☑	☐	
3	ipAdressBroker	Array[0..3] of Byte		☑	☑	☑	☐	
4	localPort	UInt	0	☑	☑	☑	☐	
5	mqttPort	UInt	0	☑	☑	☑	☐	

Fig. 44 typeTcpConnParam.

- **typeMqttParam**: tipo de datos propio del ejemplo de aplicación proporcionado por Siemens, que contendrá los parámetros para la conexión con el *broker* MQTT (Fig. 45).

typeMqttParam

	Nombre	Tipo de datos	Valor predet.	Accesible d...	Escrib...	Visible en ...	Valor de a...	Come
1	connectFlag	"typeMqttConnectFlags"		☑	☑	☑	☐	
2	publishFlag	"typeMqttPublishFlags"		☑	☑	☑	☐	
3	keepAlive	Word	16#0A	☑	☑	☑	☐	
4	packetIdentifier	Word	16#10	☑	☑	☑	☐	
5	clientIdentifier	String[23]	''	☑	☑	☑	☐	
6	willTopic	String[100]	''	☑	☑	☑	☐	
7	willMessage	String[100]	''	☑	☑	☑	☐	
8	userName	String[20]	''	☑	☑	☑	☐	
9	password	String[20]	''	☑	☑	☑	☐	
10	topic	String[100]	''	☑	☑	☑	☐	
11	message	String	''	☑	☑	☑	☐	

Fig. 45 typeMqttParam.

- **LMqtt_Publisher**: Para el funcionamiento de la conexión esta librería nos proporciona un bloque de función (FB) escrito en lenguaje SCL que se encarga de gestionar toda la conexión, publicación y suscripción de datos con el *broker* MQTT, este bloque de función está acompañado de su correspondiente bloque de datos de instancia, que en esta librería se denomina ***LMqtt_Publisher_DB*.**
- **LMqtt_Data:** Todos los datos de configuración tanto de la conexión TCP como de la conexión al *broker* MQTT se deben escribir en un bloque de datos (*DB)* debidamente formateado que nos proporciona también la librería (Fig. 46). Para la creación de este bloque de datos, la librería se vale de los tipos de datos definidos por el usuario que lo acompañan.

LMqtt_Data (instantánea generada: 09/05/2018 18:39:51)

	Nombre	Tipo de datos	Valor de arranque	Remanen...	Accesible d...	Escrib...	Visible en ...	Valor de a...
1	Static							
2	dataTcp	"typeTcpConnParam"						
3	hwIdentifier	HW_ANY	64					
4	connectionID	CONN_OUC	16#0002					
5	ipAdressBroker	Array[0..3] of Byte						
6	localPort	UInt	2000					
7	mqttPort	UInt	1883					
8	dataMqtt	"typeMqttParam"						
9	connectFlag	"typeMqttConnectF...						
10	cleanSession	Bool	TRUE					
11	will	Bool	TRUE					
12	willQoS_1	Bool	FALSE					
13	willQoS_2	Bool	TRUE					
14	willRetain	Bool	TRUE					
15	password	Bool	TRUE					
16	userName	Bool	TRUE					
17	publishFlag	"typeMqttPublishFl...						
18	qualityOfService	Int	2					
19	retain	Bool	TRUE					
20	keepAlive	Word	16#003C					
21	packetIdentifier	Word	16#0010					
22	clientIdentifier	String[23]	'publicadorPLC'					
23	willTopic	String[100]	'Averias/Nave Pintura/Transportadores/GF15//Conexion'					
24	willMessage	String[100]	'1'					
25	userName	String[20]	'eduardo'					
26	password	String[20]	'asdf1234'					
27	topic	String[100]	'Averias/Nave Pintura/Transportadores/GF15/Zona 1/CR1/B2'					
28	message	String	'0'					

Fig. 46 LMqtt_Data.

El programa realizado para este prototipo está dividido en tres zonas básicas:

- El programa principal, donde se llama a la función de gestión de envíos de las averías y donde se gestiona la aparición y desaparición de la avería.
- Una zona (carpeta de bloques) llamada *EnvioAveriasFijo*, donde se realizan las funciones básicas de tratamiento de las averías, esta zona no se modifica sean cuales sean las averías a enviar.
- Una zona (carpeta de bloques) llamada *EnvioAveriasVariable,* donde se realizan las funciones dependientes de las averías configuradas (Fig. 47).

Cuando surge una incidencia (aparición o desaparición de una avería), la zona de funciones dependientes introduce esas averías con su mensaje correspondiente en una pila (DB de datos) y la zona de funciones básicas de tratamiento de averías envía los mensajes correspondientes a las averías insertadas en la pila de forma ordenada.

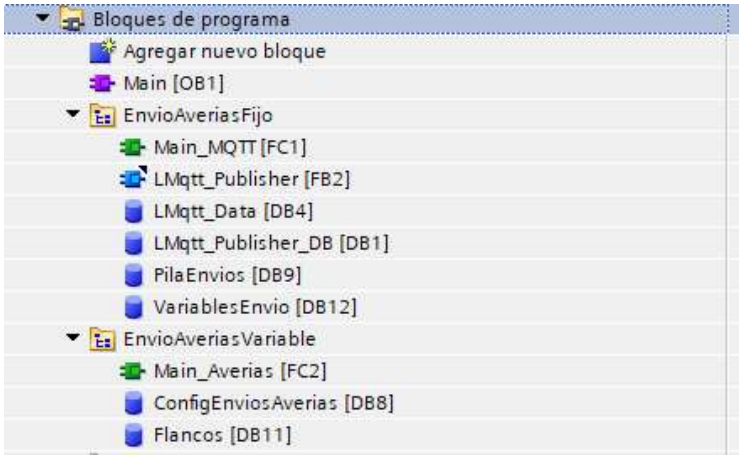

Fig. 47 Zonas de programa PLC.

3.2.1.1. BLOQUE PRINCIPAL

El bloque principal de este programa está escrito en lenguaje *KOP* por su sencillez, únicamente se encarga de activar y desactivar la avería dependiendo del pulsador externo, controlar el LED y llamar al bloque principal de gestión de averías, podemos ver todo su código en Fig. 48.

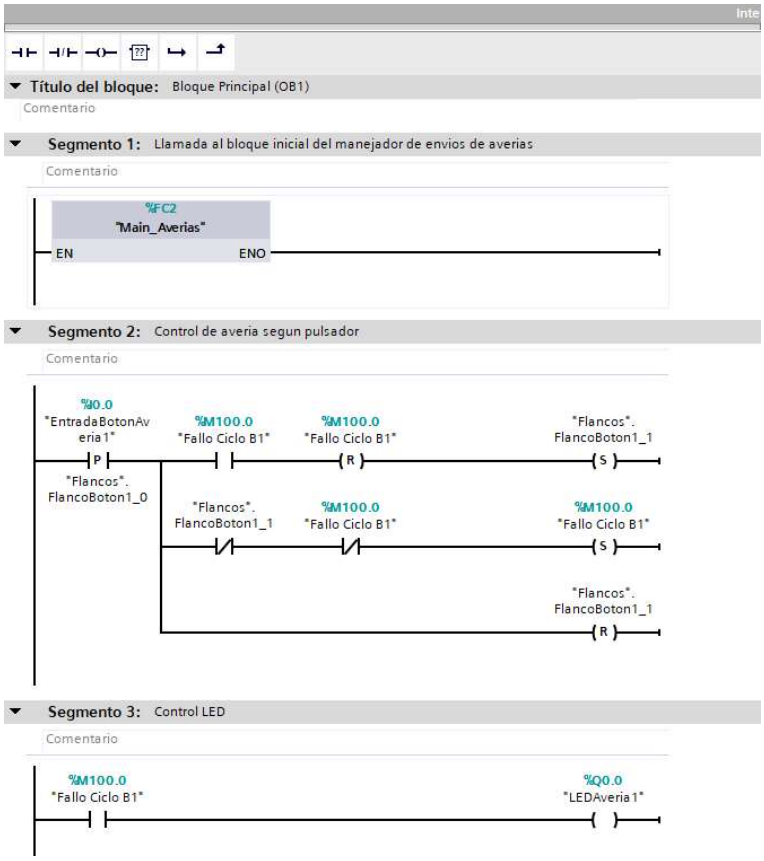

Fig. 48 OB1 programa PLC.

3.2.1.2. ENVÍO AVERÍAS ZONA VARIABLE

En esta zona únicamente tenemos una función, *Main_Averias* escrita en lenguaje *SCL* y dos bloques de datos, el primero *ConfigEnviosAverias* donde se recogen las configuraciones de conexión al *broker* MQTT (Fig. 49) y un bloque de datos auxiliar para generar los flancos necesarios para el correcto funcionamiento de la instalación (*Flancos)*.

ConfigEnviosAverias

	Nombre	Tipo de datos	Valor de arranque	Remanen...	Accesible d...	Escrib...	Visible en ...	Valor de a...	Com
1	Static								
2	Area	String	'Chapa'		☑	☑	☑		
3	Subarea	String	'Transportadores'		☑	☑	☑		
4	Sistema	String	'GF1'		☑	☑	☑		
5	ip_server	Array[0..3] of Byte			☑	☑	☑		
6	ip_server[0]	Byte	192		☑	☑	☑		
7	ip_server[1]	Byte	168		☑	☑	☑		
8	ip_server[2]	Byte	1		☑	☑	☑		
9	ip_server[3]	Byte	30		☑	☑	☑		
10	Usuario	String	'eduardo'		☑	☑	☑		
11	password	String	'asdf1234'		☑	☑	☑		

Fig. 49 ConfigEnviosAverias.

En la función *Main_Averias* se codifica el envío a través de una pila de envíos de las averías en el momento de su aparición y desaparición.

La avería "*Conexion"* solo es tratada en su desaparición (cuando se conecta al *broker* MQTT) ya que su aparición cuando se desconecta es gestionada por el *broker* MQTT.

Podemos ver el código de esta función en la siguiente figura (Fig. 50).

```
(*
 * Destinatario: Trabajo Fin de Master
 *               Master Investigacion en Ingenieria de Software y
 *               Sistemas Informaticos
 * Proyecto: Sistema de propagacion de averias en la Industria 4.0
 * Fecha Creacion: 11/05/2018
 * Fecha Revision: 11/05/2018
 * @author: Eduardo Buetas
 * @version: 1.0
 *)
REGION Cota 0
    REGION Fallo Ciclo B1
        IF "Fallo Ciclo B1" AND NOT "Plancos"."CR1 Cota 0 Fallo Ciclo B1 1" THEN
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Zona := 'Cota 0';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Elemento := 'CR1';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Nombre_Averia := 'Fallo Ciclo B1';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Valor_Averia := '1';
            "PilaEnvios".Puntero := "PilaEnvios".Puntero + 1;
            "Plancos"."CR1 Cota 0 Fallo Ciclo B1 1" := true;
        END_IF;
        IF NOT "Fallo Ciclo B1" THEN
            "Plancos"."CR1 Cota 0 Fallo Ciclo B1 1" := false;
        END_IF;
        IF NOT "Fallo Ciclo B1" AND NOT "Plancos"."CR1 Cota 0 Fallo Ciclo B1 0" THEN
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Zona := 'Cota 0';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Elemento := 'CR1';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Nombre_Averia := 'Fallo Ciclo B1';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Valor_Averia := '0';
            "PilaEnvios".Puntero := "PilaEnvios".Puntero + 1;
            "Plancos"."CR1 Cota 0 Fallo Ciclo B1 0" := true;
        END_IF;
        IF "Fallo Ciclo B1" THEN
            "Plancos"."CR1 Cota 0 Fallo Ciclo B1 0" := false;
        END_IF;
    END_REGION
    REGION conexion
        IF "VariablesEnvio".MQTTOk AND NOT "Plancos"."CR1 Mqtt Mqtt Conexion 0" THEN
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Zona := 'Mqtt';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Elemento := 'Mqtt';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Nombre_Averia := 'Conexion';
            "PilaEnvios".Averias_Enviar["PilaEnvios".Puntero].Valor_Averia := '0';
            "PilaEnvios".Puntero := "PilaEnvios".Puntero + 1;
            "Plancos"."CR1 Mqtt Mqtt Conexion 0" := true;
        END_IF;
        IF NOT "VariablesEnvio".MQTTOk THEN
            "Plancos"."CR1 Mqtt Mqtt Conexion 0" := false;
        END_IF;
    END_REGION
END_REGION

"Main_MQTT"(); //lamada a la funcion que envia las averias que mete este fichero en la pila de envio de averias.
```

Fig. 50 Función Main_Averias.

Como podemos ver en el código esta función es la encargada de llamar a la función principal de la zona fija de envío de averías (*Main_MQTT)*.

3.2.1.3. ENVÍO AVERÍAS ZONA FIJA

En esta zona tenemos todos los elementos de la librería que hemos importado para la publicación MQTT así como las funciones auxiliares para construir los tópicos y mensajes adecuados según las averías insertadas en la pila por la zona variable de envío de averías.

Para ello, en el desarrollo de este prototipo se han creado dos bloques de datos auxiliares, el primero de ellos es un bloque de datos auxiliar para almacenar las diferentes variables que son requeridas para la llamada a la función (*VariablesEnvio* Fig. 51*)* y por otro el segundo bloque de datos que actuará como pila de envíos de averías (*PilaEnvios* Fig. 52*)*.

VariablesEnvio

	Nombre	Tipo de datos	Valor de arranq...	Remanen...	Accesible d...	Escrib...	Visible en ..	Valor de a..	Con
1	Static								
2	HabilitarMQTT	Bool	false						
3	PublicarMQTT	Bool	false						
4	TCPOk	Bool	false						
5	MQTTOk	Bool	false						
6	MQTTTrabajando	Bool	false						
7	MQTTError	Bool	false						
8	MQTTStatus	DWord	16#0						
9	MQTTStatusID	Int	0						
10	FP_HabilitarMQTT	Bool	false						

Fig. 51 VariablesEnvio.

PilaEnvios

	Nombre	Tipo de datos	Valor de arranq...	Remanen...	Accesible d...	Escrib...	Visible en ..	Valor de a..	Com
1	Static								
2	Puntero	Int	1						
3	N_Max_Registros	Int	10						
4	Averias_Enviar	Array[1..10] of "tip...							
5	Averias_Enviar[1]	"tipoPilaEnvio"							
6	Zona	String	''						
7	Elemento	String	''						
8	Nombre_Averia	String	''						
9	Valor_Averia	String	''						
10	Averias_Enviar[2]	"tipoPilaEnvio"							
11	Zona	String	''						
12	Elemento	String	''						
13	Nombre_Averia	String	''						
14	Valor_Averia	String	''						
15	Averias_Enviar[3]	"tipoPilaEnvio"							
16	Averias_Enviar[4]	"tipoPilaEnvio"							
17	Averias_Enviar[5]	"tipoPilaEnvio"							
18	Averias_Enviar[6]	"tipoPilaEnvio"							
19	Averias_Enviar[7]	"tipoPilaEnvio"							
20	Averias_Enviar[8]	"tipoPilaEnvio"							
21	Averias_Enviar[9]	"tipoPilaEnvio"							
22	Averias_Enviar[10]	"tipoPilaEnvio"							

Fig. 52 PilaEnvios.

Para la creación del bloque de datos de la pila de envíos se ha generado un tipo de datos de usuario específico *tipoPilaEnvio* que se muestra en Fig. 53.

tipoPilaEnvio

	Nombre	Tipo de datos	Valor predet.	Accesible d...	Escrib...	Visible en ...	Valor de a...	Com
1	Zona	String	''	☑	☑	☑	☐	
2	Elemento	String	''	☑	☑	☑	☐	
3	Nombre_Averia	String	''	☑	☑	☑	☐	
4	Valor_Averia	String	''	☑	☑	☑	☐	

Fig. 53 tipoPilaEnvio.

La función *Main_MQTT* realiza tres tareas básicas:

- Controlar la llamada a la función importada de la librería *LMqtt_Publisher*, reintentado la conexión en el caso de que esta caiga por alguna razón.
- Configurar los datos básicos tanto de la conexión como de los mensajes, dependiendo de los valores almacenados en los diferentes bloques de datos de configuración Fig. 54
- Realizar el envío de los mensajes que están esperando para ser lanzados en la pila (bloque de datos *PilaEnvios*) (Fig. 55).

```
70 REGION "Datos de conexion MQTT"
71     //Datos fijos de conexion MQTT
72     #Topic := 'Averias/';
73     #Topic := CONCAT(IN1:=#Topic, IN2:="ConfigEnviosAverias".Area);
74     #Topic := CONCAT(IN1 := #Topic, IN2 := '/');
75     #Topic := CONCAT(IN1 := #Topic, IN2 := "ConfigEnviosAverias".Subarea);
76     #Topic := CONCAT(IN1 := #Topic, IN2 := '/');
77     #Topic := CONCAT(IN1 := #Topic, IN2 := "ConfigEnviosAverias".Sistema);
78     #Topic := CONCAT(IN1 := #Topic, IN2 := '/');
79     "LMqtt_Data".dataMqtt.userName := "ConfigEnviosAverias".Usuario; //usuario de conexion
80     "LMqtt_Data".dataMqtt.password := "ConfigEnviosAverias".password; //password de conexion
81     //Will topic se refiere al mensaje que se le enviara a los suscriptores en el caso de la perdida de conexion
82     //con el publicador, este mensaje se almacena en el broker hasta que el publicador pierde la conexion con el
83     //broker en este momento el mensaje es enviado a los suscriptores
84     "LMqtt_Data".dataMqtt.willTopic := CONCAT(IN1 := #Topic, IN2 := 'Mqtt/Mqtt/Conexion');
85     "LMqtt_Data".dataMqtt.willMessage := '1';
86     "LMqtt_Data".dataMqtt.connectFlag.will := TRUE;
87     "LMqtt_Data".dataMqtt.connectFlag.willQoS_1 := FALSE;
88     "LMqtt_Data".dataMqtt.connectFlag.willQoS_2 := TRUE;
89     "LMqtt_Data".dataMqtt.connectFlag.willRetain := true;
90     //Los mensajes que envie este publicador se almacenaran en el broker hasta que todos los
91     //suscripotores los reciban
92     "LMqtt_Data".dataMqtt.publishFlag.retain := true;
93     //Los mensajes intercamebiados por est Bool scriptor con el broker se enviaran con una
94     //calidad deservicio QoS=2
95     "LMqtt_Data".dataMqtt.publishFlag.qualityOfService := 2;
96     //El timepo de KeepAlive sera de 60 segundos entre este publicador y el broker
97     "LMqtt_Data".dataMqtt.keepAlive := 60;
98 END_REGION
```

Fig. 54 Configuraciones básicas de la conexión MQTT. Función Main_MQTT.

```
111 REGION Envio de mensajes de averia desde la pila
112     //Estos datos son fijos para todos los envios por MQTT de las averias
113     IF "PilaEnvios".Puntero > 1 AND #listo_para_envio THEN //si la pila es <> 1
114                                                            //es que hay algo que enviar, si no se esta enviando se envia
115         #Topic := CONCAT(IN1 := #Topic, IN2 := "PilaEnvios".Averias_Enviar[1].Zona);
116         #Topic := CONCAT(IN1 := #Topic, IN2 := '/');
117         #Topic:= CONCAT(IN1 := #Topic, IN2 :="PilaEnvios".Averias_Enviar[1].Elemento);
118         #Topic := CONCAT(IN1 := #Topic, IN2 := '/');
119         "LMqtt_Data".dataMqtt.topic := CONCAT(IN1 := #Topic, IN2 := "PilaEnvios".Averias_Enviar[1].Nombre_Averia);
120         "LMqtt_Data".dataMqtt.message := "PilaEnvios".Averias_Enviar[1].Valor_Averia;
121         "VariablesEnvio".PublicarMQTT:= true;
122     END_IF;
123 END_REGION
```

Fig. 55 Envío de mensajes MQTT. Función Main_MQTT.

Para las pruebas relativas a este prototipo se ha conectado el pulsador de control de la avería a la entrada I0.0 y el led a la salida Q0.0. (Fig. 56).

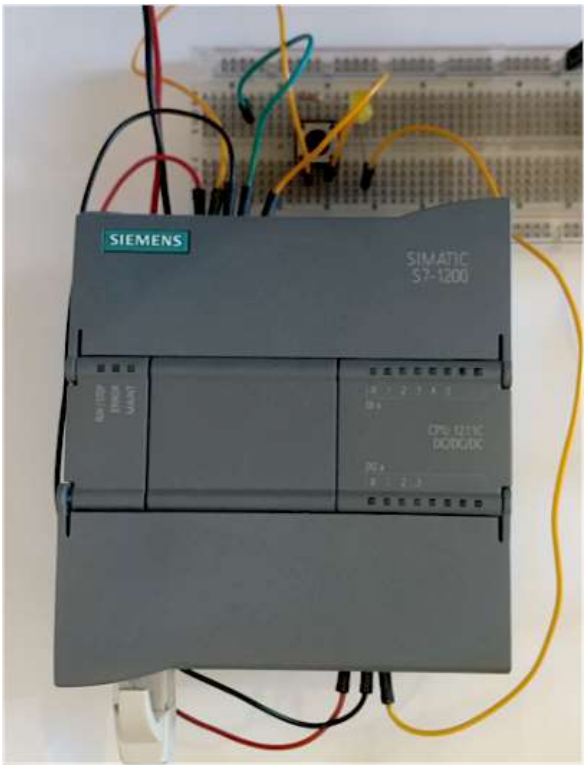

Fig. 56 Imagen Prototipo PLC.

3.2.2. MicroPC RASPBERRY Pi 3 MODEL B+

El controlador de automatización basado en Raspberry Pi, simulará ser el controlador del sistema GF1 de la nave de pintura (Fig. 57), para las pruebas realizadas configuraremos en este equipo tres de las averías definidas en la base de datos (Tabla 6), con los siguientes tópicos:

"Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1"

"Averias/Pintura/Transportadores/GF1/Cota 1/MG2/Termico Motor 1"

"Averias/Pintura/Transportadores/GF1/Mqtt/Mqtt/Conexion"

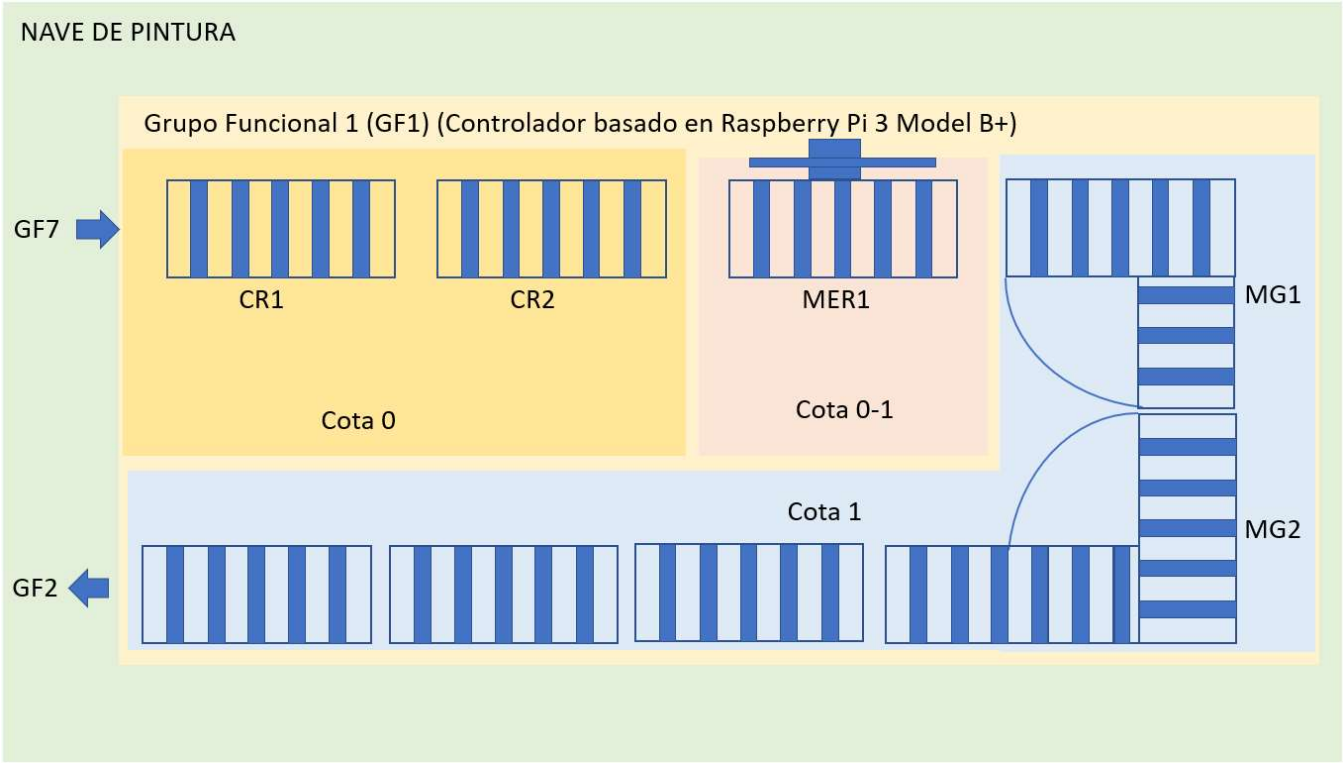

Fig. 57 Sistema automatizado basado en microPC Raspberry Pi 3 Model B+.

En este caso la conexión con el *broker* MQTT se realizará a través de conexión Wifi.

Las dos primeras averías se simularán con un pulsador y un led, cuando se pulse el pulsador una vez la avería será lanzada y el led se activará, cuando se pulse el pulsador una segunda vez la avería desaparecerá y el led se desactivará.

La tercera avería se configurará con mensaje "*1*" como mensaje que el *broker* enviará cuando detecte una desconexión de este controlador y cada vez que haya una conexión con el *broker* MQTT el controlador enviará la avería con mensaje "0".

Para este prototipo se ha desarrollado este software en java, utilizando para su desarrollo el IDE de libre distribución NetBeans 8.1, podemos descargar este software de su sitio web oficial (https://netbeans.org/downloads/index.html).

A continuación, se describirá el software generado con este propósito, además el proyecto de este software debidamente comentado se entregará conjuntamente con este documento, estando el código debidamente comentado para su fácil comprensión, además de entregarse también la documentación autogenerada por medio del generador de documentación *JavaDoc*.

Este prototipo tiene que gestionar los eventos producidos por las entradas y salidas de la Raspberry Pi, para controlar los eventos de los dos pulsadores controladores de las averías y los leds de señalización de estas, para ello se utilizará la librería Pi4J licenciada bajo GNU LGPL v3.0. Este proyecto pretende abstraer de los elementos de bajo nivel del control de las entradas salidas de las Raspberry Pi a los desarrolladores que implementen proyectos en java para estos equipos. Este proyecto está desarrollado por Robert Savage y por Daniel Sendula.

Podemos descargar estas librerías de su página oficial (http://pi4j.com/download.html), es importante que descarguemos la última versión liberada 1.2, ya que las versiones anteriores presentan problemas de incompatibilidades con las últimas versiones de hardware de los equipos (Fig. 58).

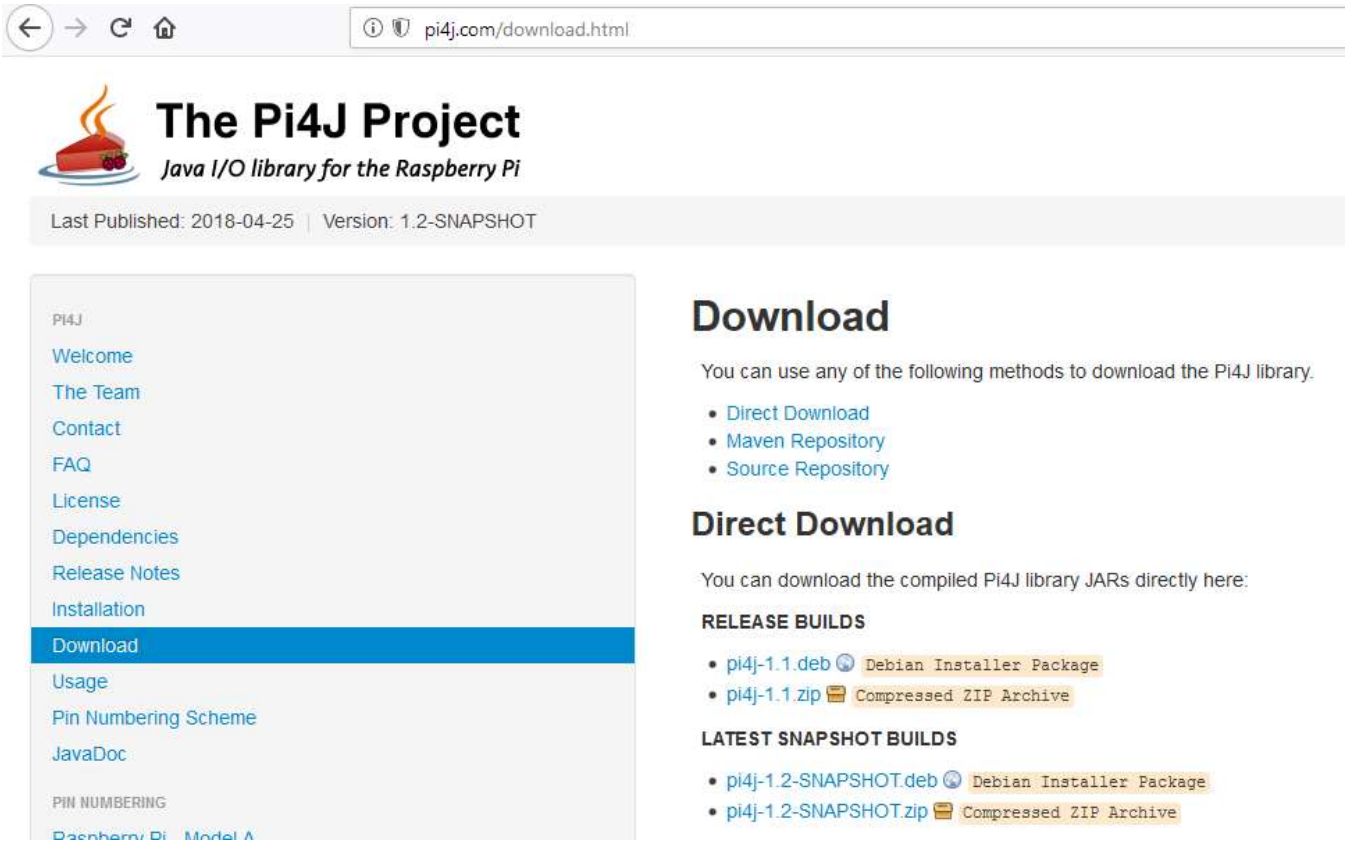

Fig. 58 Descarga Pi4J.

Una vez tengamos descargada esta librería deberemos añadir los archivos jar descargados a nuestro proyecto.

Por otro lado, para la implementación de las funciones de conexión al *broker* MQTT se utiliza las clases proporcionadas por la fundación *Eclipse (https://www.eclipse.org/)* en su proyecto *Paho (https://www.eclipse.org/paho/)* , este es un proyecto de la fundación Eclipse que intenta facilitar la implementación del protocolo MQTT en aplicaciones nuevas, preexistentes o emergentes en el ámbito de la IoT.

Proporciona librerías open-source para la implementación de MQTT en diferentes plataformas y existe una amplia y activa comunidad trabajando en este proyecto.

Para la utilización de este proyecto en nuestra aplicación debemos descargarnos sus librerías que están contenidas en el archivo *"mqtt-client-0.4.0.jar"* que lo podremos descargar del repositorio oficial del proyecto (https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/mqtt-client/0.4.0/) .

Una vez descargado este archivo jar e incluido como librería de nuestro proyecto podemos utilizar los objetos de esta librería para crear nuestra clase.

El código de este programa está organizado en tres paquetes diferentes:

EBuetas.TFM.RBPi: con la función de ejecución del hilo principal del programa y una clase para manejar los eventos de los botones y su interface correspondiente para exportar los eventos de dichos botones.

EBuetas.TFM.RBPi.Auxiliares: que contiene una clase para la creación de apuntes en el log de la aplicación ante posibles incidencias de la misma.

EBuetas.TFM.RBPi.MQTT: en este paquete se crea la clase para el manejo de las comunicaciones con el protocolo MQTT.

3.2.2.1. CLASE EBuetas.TFM.RBPi.Averias40RBPi

Esta clase definida en el fichero *Averias40RBPi.java* contiene la función *main* que será ejecutada como hilo principal del programa al ser ejecutado.

Además, contiene los objetos de control de los botones que controlan las averías y sus correspondientes *listeners* para manejar sus eventos.

En la función *main* primero se configurarán las entradas y salidas utilizadas en esta aplicación. Configurando los pines 31 y 33 como salidas (GPIO 22 y GPIO 23 según la denominación de Pi4J) y los pines 35 y 37 como entradas (GPIO 24 y GPIO 25 según la denominación de Pi4J), los primeros para comandar los LEDs de señalización de las averías y los segundos como entrada de los pulsadores que activan y desactivan las averías.

A los pines de entrada se les asignará el *listener* que controlará sus eventos.

En estos eventos cuando el pulsador sea pulsado, si la avería esta activa (LED encendido) se desactivará y enviaremos un mensaje *"0"* al tópico correspondiente. Si por el contrario la avería esta desactivada (LED apagado) se activará y enviaremos un mensaje *"1"* al tópico correspondiente. Podemos ver este código en la siguiente figura (Fig. 59).

```
/**
 * @see EBuetas.TFM.RBPi.BotonListener#onPulsado(java.util.EventObject)
 * @param ev
 */
@Override
public void onPulsado(EventObject ev){
    if (GPIO22.isHigh()){
        GPIO22.setState(PinState.LOW);
        publicador.Publicar(1, "0");
    }else{
        GPIO22.setState(PinState.HIGH);
        publicador.Publicar(1, "1");
    }
}
```

Fig. 59 Control Avería con pulsador. Raspberry Pi.

Después de la asignación de los *listeners* se realizará la conexión al *broker* MQTT creando un objeto de la clase *PublicadorMQTT* (punto 3.2.2.3).

Después la función *main* entrará en un bucle que comprobará cada cierto tiempo si la conexión MQTT sigue activa, si no siguiera activa se destruirá y volverá a crear el objeto de la clase *PublicadorMQTT* para realizar de nuevo un intento de conexión.

Podemos ver la estructura de esta clase en Fig. 60.

Averias40RPi
- gpio: GpioController - BotonAveria1: ControlBotones - BotonAveria2: ControlBotones - GPIO22: GpioPinDigitalOutput - GPIO23: GpioPinDigitalOutput - GPIO24: GpioPinDigitalInput - GPIO25: GpioPinDigitalInput - Publicador: PublicadorMQTT - ListenerBotAveria1: BotonListener - ListenerBotAveria2: BotonListener
+ main(Array of String): void

Fig. 60 Clase "Averias40RPi".

3.2.2.2. CLASE EBuetas.TFM.RBPi.ControlBotones

Esta clase, codificada en el fichero *ControlBotones.java,* se crea para controlar las pulsaciones de los botones. A partir del evento estándar de la librería Pi4J *handleGpioPinDigitalStateChangeEvent* se generan tres eventos *onPulsado, onSoltado* y *onAlarma.*

Para ello además de esta clase se genera una *interface* con estos tres eventos *BotonListener*, codificado en el archivo *BotonListener.java*.

Podemos ver esta estructura en Fig. 61.

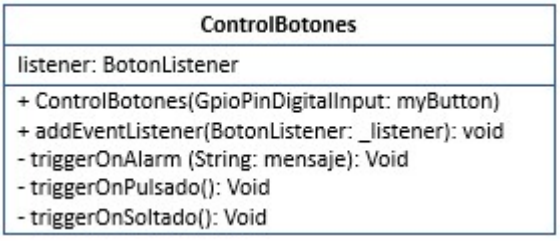

Fig. 61 Clase "ControlBotones".

3.2.2.3. CLASE EBuetas.TFM.RBPi.PublicadorMQTT

En esta clase codificada en el fichero *PublicadorMQTT.java* se implementa el publicador de los mensajes por MQTT, esto se realizará con las clases obtenidas del proyecto *Paho* de la fundación *Eclipse.*

Esta clase implementará los eventos del cliente MQTT, aunque este cliente no va a recibir ningún mensaje se implementan los eventos de esta clase para tratar el evento *connectionLost.*

En el constructor de esta clase se realiza la conexión con el *broker* MQTT, este cliente no se suscribe a ningún tópico, simplemente se mantiene conectado y cuando se produce alguna incidencia en las averías se encarga de publicar el mensaje correspondiente a la incidencia.

En esta clase se definen como constantes los tópicos de las averías gestionadas por el controlador de automatización, y se le pasan como parámetros en el constructor de esta los parámetros de conexión.

En la función de publicación, se le pasará como parámetro un entero que nos indicará el tópico a utilizar (0-Conexión, 1-Alarma 1, 2-Alarma 2) y el mensaje a enviar.

Tenemos una función *conectado* que será consultada en el bucle principal para en el caso de ocurrir una desconexión volver a intentar realizar la conexión cada cierto tiempo hasta conseguirlo.

Podemos ver la estructura de esta clase en Fig. 62.

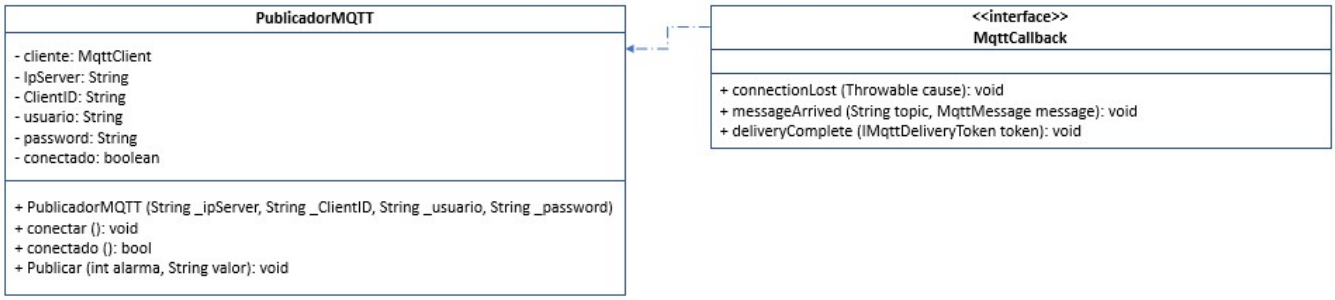

Fig. 62 Clase "PublicadorMQTT".

3.2.2.4. CLASE EBuetas.TFM.RBPi.Auxiliares.EscribeLog

Esta clase (Fig. 63), definida en el fichero *EscribeLog.java*, únicamente tendrá una función estática que se encargará de escribir en el log de la aplicación las diferentes incidencias que vayan ocurriendo en el transcurso de la ejecución del programa.

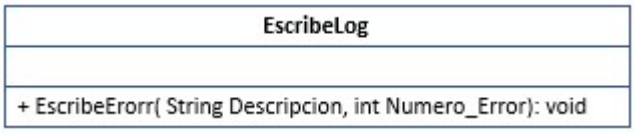

Fig. 63 Clase "EscribeLog".

Al intentar escribir un apunte en el log, si el fichero no ha sido creado esta clase lo creará y añadirá el primer apunte del log, en caso contrario añadirá los nuevos apuntes al final del fichero.

El nombre del fichero del log no es configurable y siempre será "/var/log/Averias40RBPi.log".

Cada apunte además del texto del mensaje incluirá un número identificativo que identifica el error de manera unívoca, lo cual facilita encontrar en el código la zona donde se generó dicho error y el sello de tiempo del instante en el que se produjo la incidencia que provocó la generación del apunte en el log.

Para las pruebas relativas a este controlador se ha realizado un prototipo con una Raspberry Pi y elementos discretos montados en una *protoboard* (Fig. 64).

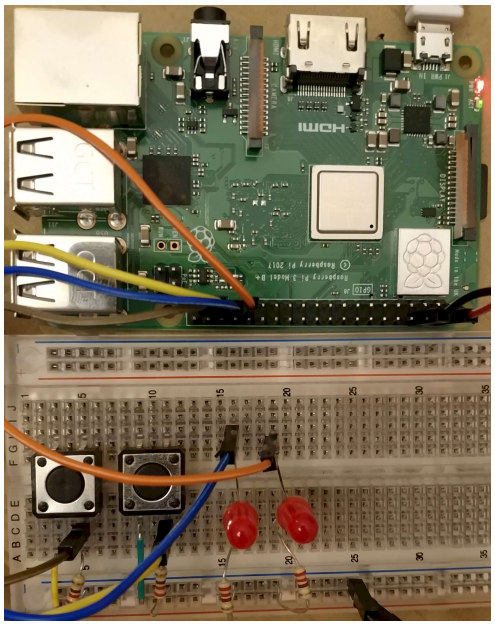

Fig. 64 Imagen Prototipo Raspberry Pi.

3.2.3. MICROPROCESADOR ARDUINO

El prototipo de automatización basado en microcontrolador se realizará con una placa Arduino MKR1000 que nos proporciona un entorno de desarrollo sencillo, basado en el microprocesador Atmel ATSAMW25 SoC, con conexión USB para su programación y conectividad WiFi integrada.

Para realizar el desarrollo de este prototipo utilizaremos el IDE proporcionado de forma gratuita por Arduino *Arduino Software (IDE)*, que podemos descargar de su página web oficial (https://www.arduino.cc/en/Main/Software).

En nuestro programa tendremos que añadir dos librerías para poder conseguir los objetivos de nuestra aplicación, por un lado, necesitaremos una librería para poder utilizar el hardware Wifi incluido en la placa de desarrollo y una librería para poder realizar una conexión con el *broker* MQTT.

Para la gestión del hardware Wifi de nuestra placa utilizaremos la librería *Wifi101* creada por la propia organización Arduino y para la conexión y publicación con el *broker* MQTT utilizaremos la librería *PubSubClient* desarrollada por Nick O'Leary, ambas librerías son de libre distribución y pueden ser descargadas directamente desde el *Arduino Software (IDE)*.

Para descargar estas librerías debemos ir en el software de Arduino en su menú a la opción *Programa → Incluir Librería → Gestionar Librería*, en el formulario que nos aparece debemos buscar la librería que queremos e instalarla, como podemos ver en las figuras Fig. 65 y Fig. 66.

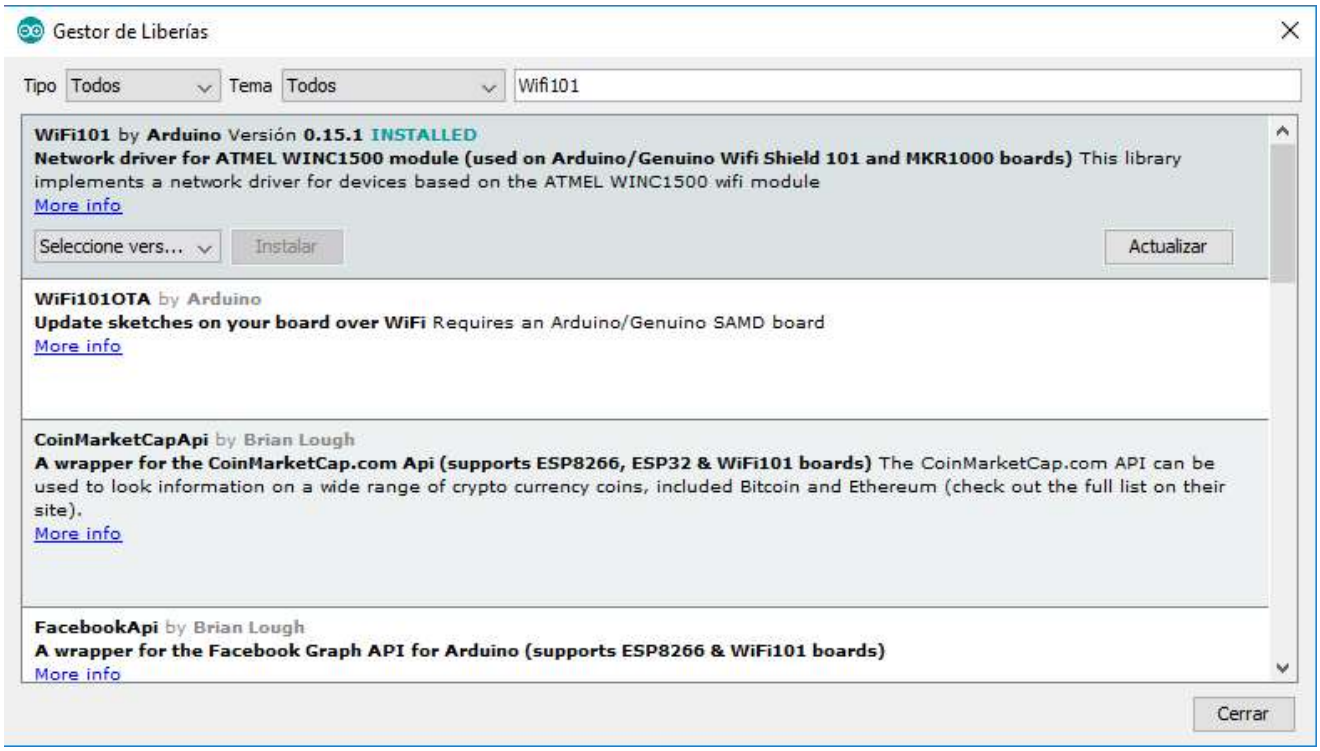

Fig. 65 Instalación librería Wifi101. Android IDE.

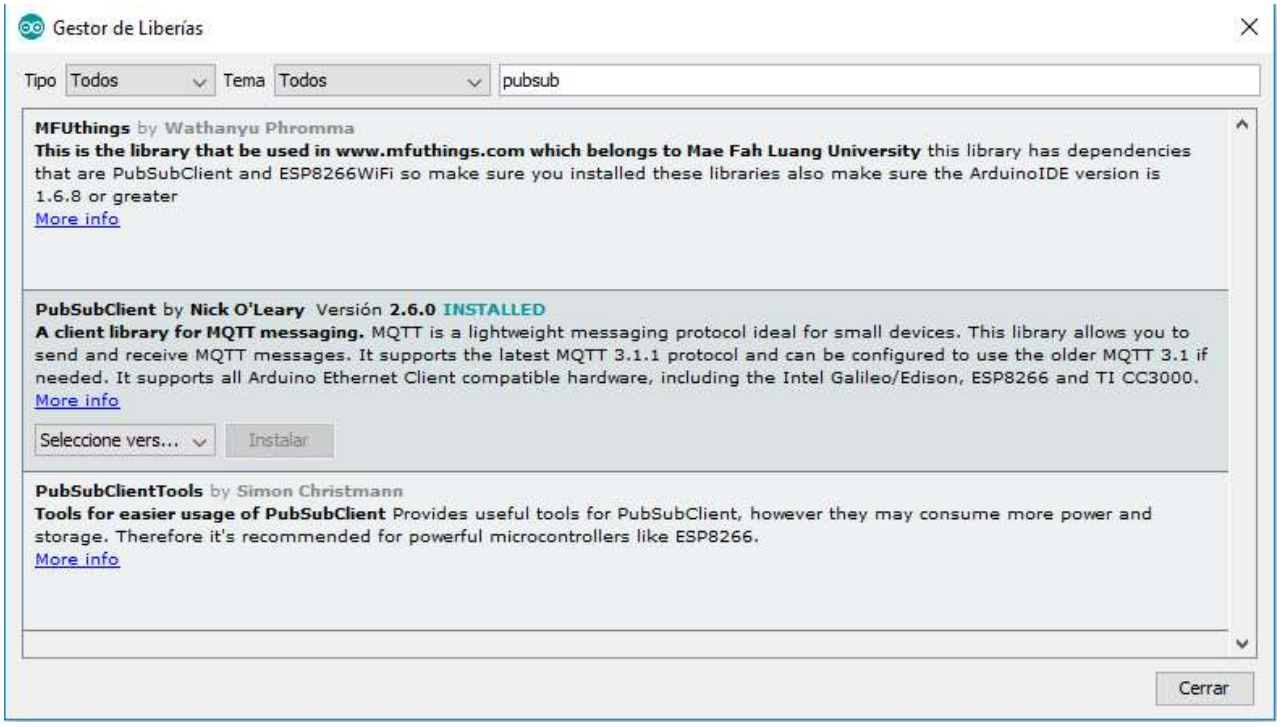

Fig. 66 Instalación librería PubSubClient. Arduino IDE.

El controlador simulado con este microprocesador simulara el controlador del equipo *Laca 1* del área de *Pintura* y la subárea *Maquinas (*Fig. 67*).*

Para las pruebas realizadas configuraremos en este equipo dos de las averías definidas en la base de datos (Tabla 6), con los siguientes tópicos:

"Averias/Pintura/Maquinas/Laca 1/Zona 1/ESTATICA1/Fallo Boquilla 5"

"Averias/Pintura/Maquinas/Laca 1/Mqtt/Mqtt/Conexion"

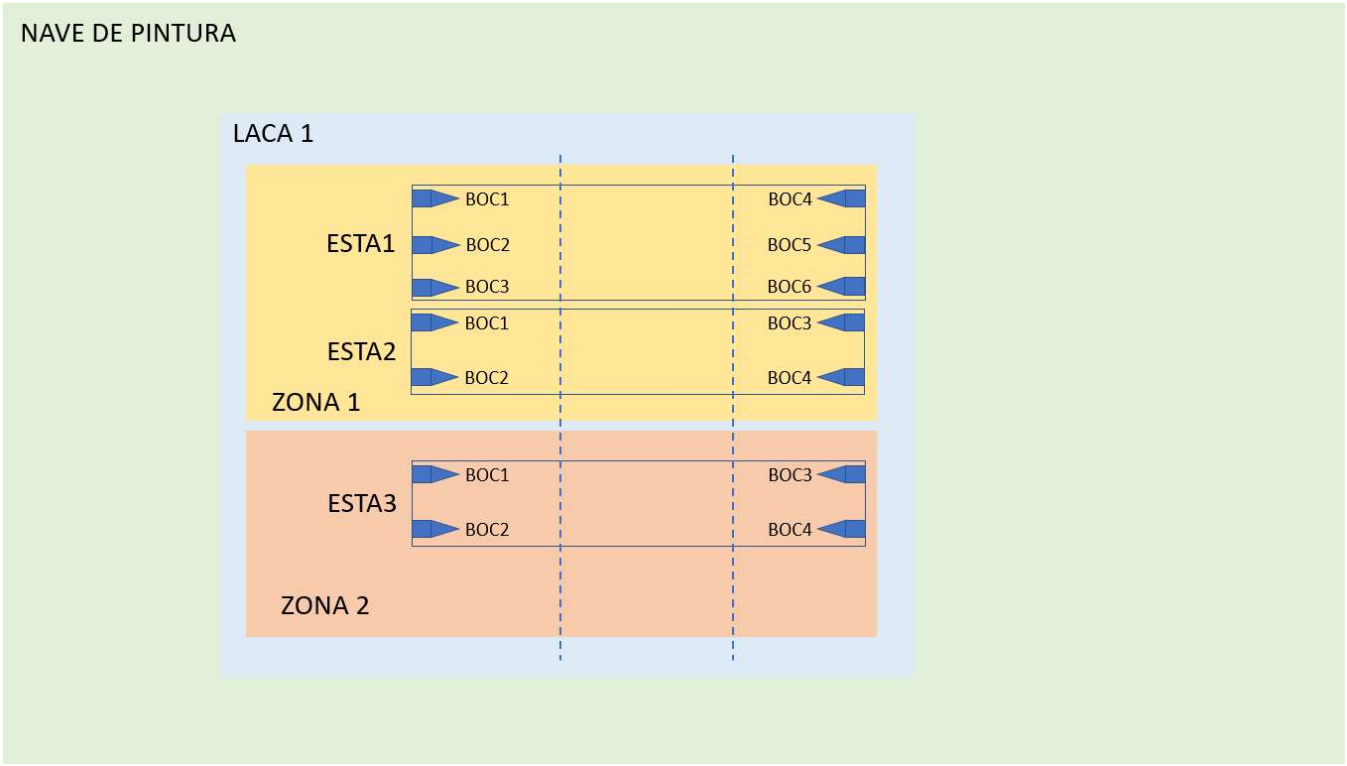

Fig. 67 Sistema automatizado basado en Arduino.

La primera avería será controlada con un pulsador y un LED, una pulsación del pulsador activará la avería y la siguiente pulsación la desactivará, el LED señalizará el estado de esta avería.

La segunda avería será enviada con mensaje *"0"* cada conexión correcta con el *broker* MQTT y será configurada en el *willTopic* con un mensaje *willMessage "1"* para ser enviado ante una desconexión del equipo al *broker*.

Los *Sketch* (programas) que se cargan en los procesadores Arduino tienen dos funciones principales gestionadas por el *bootloader* cargado en estos microprocesadores. La primera función se ejecuta una única vez cuando el microprocesador arranca *void setup()* y una

segunda función que se ejecuta de forma cíclica e infinita mientras el procesador esté en funcionamiento *void loop()*.

Además de estas dos funciones nuestro Sketch tiene una función llamada *void conectarMQTT().*

En la función *setup* de nuestro programa se arrancarán los hardware que vamos a utilizar en el desarrollo del prototipo.

Primero se configura y arranca el puerto serie del microprocesador, únicamente con propósitos de depuración y comprobación de funcionamiento. A continuación, se arranca la Wifi del sistema, conectándose al *ssid* que hemos configurado como constante en el programa.

Una vez realizado esto configuramos el pin para controlar el botón (pin 2 en nuestro caso) como entrada y el pin de control del led como salida (pin 5 en nuestro caso) y por último crearemos el objeto de conexión al *broker* MQTT. Podemos ver esta función en la Fig. 68.

```
/**
 * Funcion de setup se lanza cuando el micro arranca
 */
void setup() {
  Serial.begin(9600); //Arrancamos el puerto serie para ver los mensajes de debug
  while (!Serial) { //Esperamos hasta que el puerto esta abierto
    delay(1000);
  }
  WiFi.begin(ssid, pass); //Conectamos con la wifi
  delay(10000);           //Esperamos para darle tiempo al wifi a conectarse
  // Inicializamos el pin del led como salida
  pinMode(ledPin, OUTPUT);
  // Inicializamos el pin del boton como entrada
  pinMode(buttonPin, INPUT);
 // Metemos la configuracion del servidor en el objeto del clinte Mqtt
 client.setServer(server, 1883);
}
```

Fig. 68 Función setup Sketch Arduino.

En la función *loop*, que se ejecuta cíclicamente primero comprobamos si la conectividad wifi esta ok, si no es así volveremos a intentar la conexión. A continuación, si la conexión es correcta se comprueba que el cliente MQTT está debidamente conectado, si todo es correcto pasamos al control de la avería por medio del pulsador, si la conexión con el *broker* MQTT no es correcta volvemos a intentar la reconexión con dicho *broker*.

Si la gestión del pulsador nos da como resultado que se activa la alarma se publicará un mensaje *"1"* y si se desactiva la alarma se enviará un mensaje *"0"*, ambos por el tópico de la alarma gestionada (Fig. 69).

```
/**
 * Funcion que se ejecuta en bucle una vez terminado el setup y de manera infinita mientras el micro
 * este activo
 */
void loop() {
  //lo primero en el bucle es mirar si esta conectada la wifi, si no lo esta volvemos a intentar que se conecte
   switch (WiFi.status()) {
    case WL_CONNECT_FAILED:
    case WL_CONNECTION_LOST:
    case WL_DISCONNECTED: WiFi.begin(ssid, pass);
  }
  // Si la wifi no esta conectada no hacemos nada, de lo contrario entramos en el if
  if (WiFi.status()==WL_CONNECTED){
    //Comprobamos si estamos conectados al Mqtt, si no lo estamos reconectamos
    if (!client.connected()) {
      reconnect();
    }
    buttonState = digitalRead(buttonPin);
    //En el flanco positivo de la pulsacion del boton cambiamos el estado de la averia
    if (buttonState == HIGH && !flanco_senal) {
        estado_averia=!estado_averia;
        flanco_senal=true;
        Serial.println("Senal 1");
    }
    if (buttonState == LOW) {
       flanco_senal=false;
       Serial.println("Senal 0");
    }
    //en el flanco positivo de la averia enviamos un mensaje 1 al topico de la averia
    if (estado_averia && !flanco_positivo){
       flanco_positivo=true;
       client.publish(topicname,"1");
       digitalWrite(ledPin, HIGH);
       Serial.println("mando averia 1");
    }
    //en el flanco negativo de la averia enviamos un mensaje 1 al topico de la averia
    if (!estado_averia && flanco_positivo){
       flanco_positivo=false;
       client.publish(topicname,"0");
       digitalWrite(ledPin, LOW);
       Serial.println("mando averia 0");
    }
    //Escribimos el estado de las variables por el puerto para depuracion
    char mensaje[1024];
    sprintf(mensaje,"Flanco Positivo: %d Flanco Senal: %d  Estado Averia: %d",flanco_positivo,flanco_senal,estado_averia);
    Serial.println(mensaje);
  }else{
    //Escribimos el estado de la wifi para depuracion por el puerto serie
    Serial.println("Wifi no conectada");
  }
  //llamamos a la funcion loop del objeto mqtt, es necesario llamarla en el loop del sketch
  client.loop();
  delay(250);
}
```

Fig. 69 Función loop Sketch Arduino.

Tanto para la primera conexión como para las conexiones siguientes ante la detección de una desconexión al *broker* MQTT se utilizará la función *conectarMQTT*, esta función intenta

conectar con el *broker* MQTT, en el caso de conseguirlo envía el mensaje *"0"* al tópico de la avería "*Conexion"*. Podemos ver la función en Fig. 70.

```
/**
 * Funcion para conectar con el broker MQTT
 */
void conectarMQTT() {

  Serial.print("Esperando para reconexion Mqtt...");
  // Intneto conexion
  if (client.connect(clienteid,user,password,willtopic,2,1,"1")) {
     //Conectamos con el mqtt y configuramos el mensaje will (para cuando se desconecte)
    Serial.println("Conectado"); //Sacamos la conexion por el puerto serie para debug
     delay(5000);           //Esperamos para darle tiempo al mqtt a conectarse
    // Enviamos el mensaje de conexion ok
    Serial.println("Enviando willtopic...");
    client.publish(willtopic,"0");
    Serial.println("WillTopic enviado");
  } else {
    //Sacamos el fallo por el puerto serie para debug
    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println("Probaremos en 5 segundos...");
    delay(5000);
  }
}
```

Fig. 70 Función conectarMQTT Sketch Arduino.

Podemos ver el prototipo desarrollado para este punto en la siguiente figura (Fig. 71).

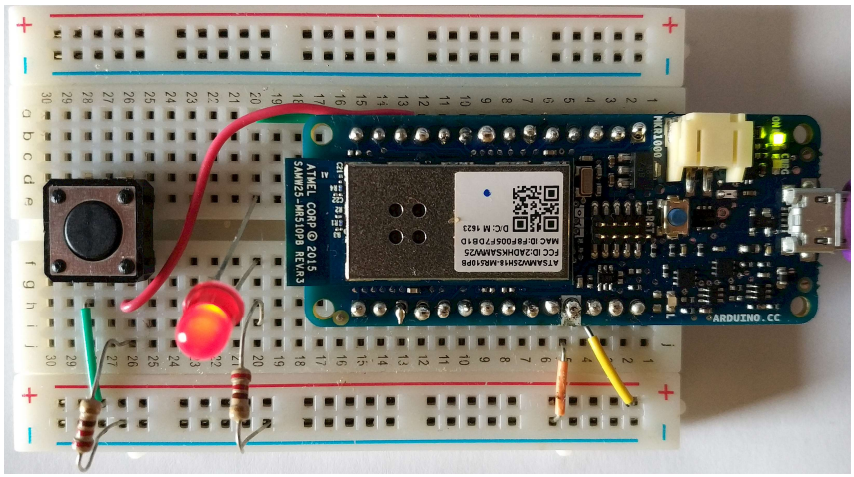

Fig. 71 Imagen Prototipo Arduino MKR1000.

3.3. EQUIPOS DE RECEPCIÓN DE AVERÍAS

Los técnicos de mantenimiento correctivo portarán estos equipos móviles que les notificarán las averías producidas en los sistemas de su competencia. Estos equipos estarán suscritos a los tópicos que estén en el área o subárea de la que sean responsables.

En el prototipo desarrollado en este TFM se realizará una aplicación Android® para poder convertir cualquier equipo que ejecute este sistema operativo en un dispositivo de recepción de averías. Este desarrollo implementa lo descrito en el punto 2.2.3.

Cuando se produzca una avería en uno de estos tópicos a los que está suscrito, de manera automática estos equipos recibirán el mensaje con la información de área, subárea, elemento, zona y nombre de avería, y se mostrara en la pantalla principal de la aplicación las averías recibidas (Fig. 72), junto con la hora a la que se recibió la incidencia, todas las averías se mostrarán ordenadas por instante de aparición, la más nueva, la primera en ser mostrada. Si existen más averías de las que se pueden mostrar en la pantalla, se habilitará una barra de scroll para poder desplazarnos por las averías existentes.

Fig. 72 Averías mostradas en la APP recepción de averías.

Si el portador de este equipo se va a encargar de solventar la avería enviará un mensaje de acuse de recibo para que quede almacenado en el sistema el instante de acuse de recibo y que equipo (técnico) ha respondido a la avería. Para ello pulsará sobre la avería que va a tratar y le aparecerá una pantalla en la que podrá acusar la avería y pedir más información sobre ella, pulsando en el botón "*Acusar*" se enviará el acuse al servidor y a todos los dispositivos de recepción de averías que estén suscritos al tópico de esta avería, al servidor para que almacene en base de datos su acuse y a los demás suscriptores para informarles de que esa avería ya ha sido atendida por un usuario del sistema (Fig. 73).

Fig. 73 Avería acusada APP recepción de averías.

En la misma pantalla que se usa para acusar las averías también existe un botón para pedir la información adicional que deseemos para la avería pulsada, esta pulsación provocará el envío de un mensaje con el tópico de la avería con el mensaje *"3-Nombre Equipo"* que será recibido por el servidor y este enviará un mensaje de respuesta con la información requerida al tópico "*Informacion/Nombre equipo/Tipo Informacion"*. Esta información será mostrada en la misma pantalla desde donde ha sido pedida Fig. 74.

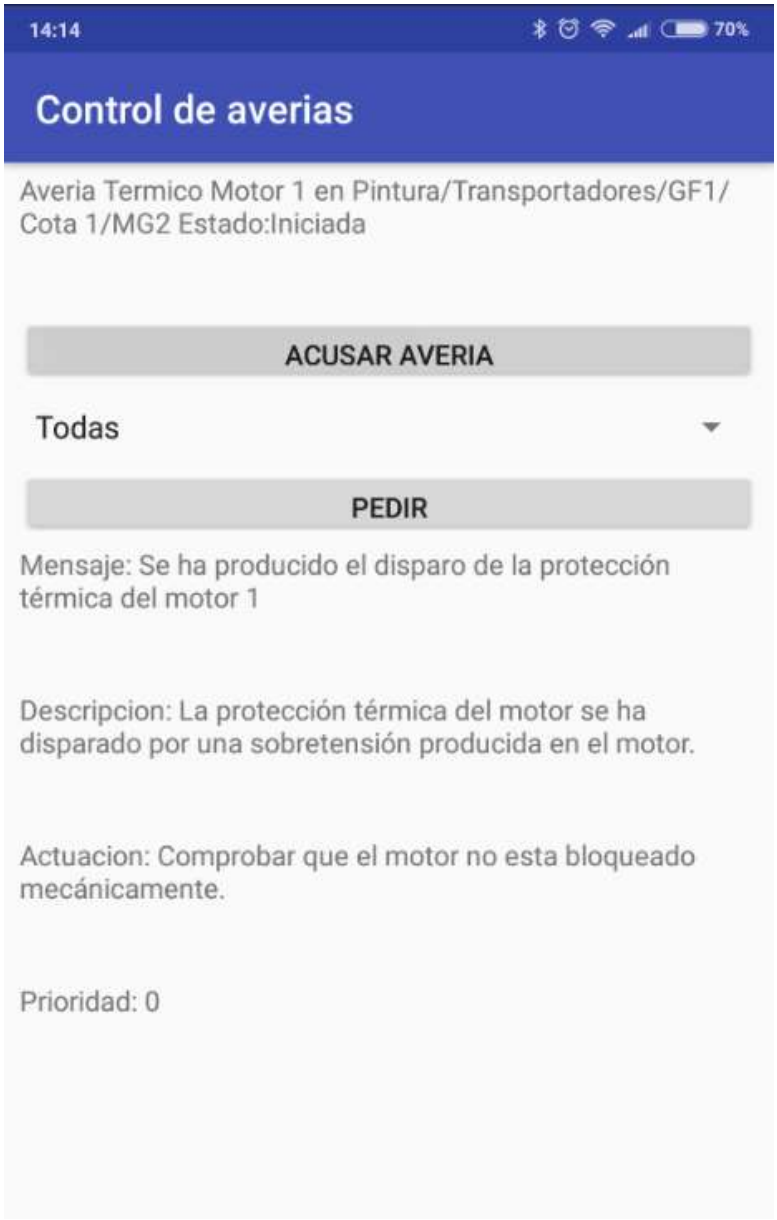

Fig. 74 Petición de información APP recepción de averías.

Cuando llegue un mensaje de avería finalizada, si la aplicación está configurada para mostrar las averías finalizadas esta se mostrará con fondo verde (Fig. 75), si está configurada para no mostrar las averías finalizadas desaparecerá de la lista.

Fig. 75 Avería finalizada APP recepción de averías.

Una vez arrancada la aplicación, cada vez que llegue una publicación a nuestro equipo, este en el estado en el que este la aplicación, en primer plano o minimizada y el dispositivo, activo o en suspenso se realizará un aviso sonoro y de vibración y si la aplicación no está en primer plano también se lanzará una notificación a la barra de notificaciones del dispositivo, para informar al usuario de las notificaciones pendientes de comprobar que existen en el dispositivo (Fig. 76).

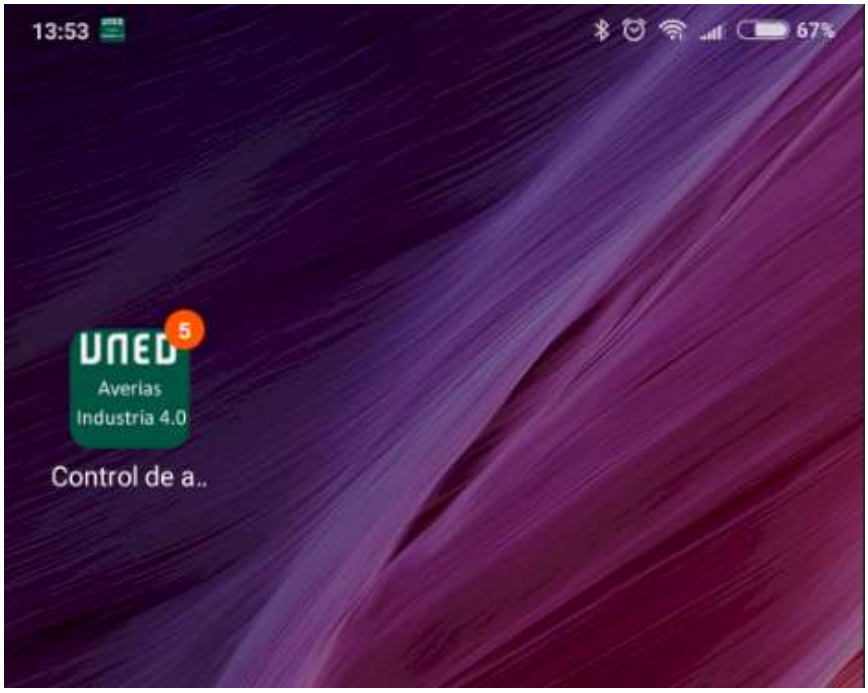

Fig. 76 Notificaciones APP recepción de averías.

Estas notificaciones contendrán la información de la publicación recibida Fig. 77

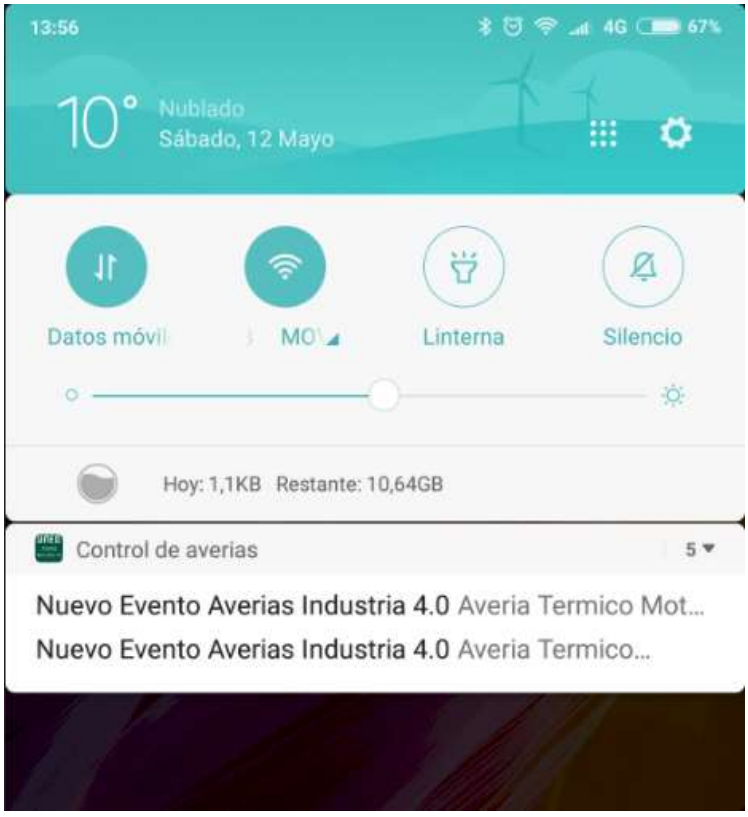

Fig. 77 Vista Notificaciones APP recepción de averías.

Para la configuración de la aplicación tendremos en el menú contextual de la pantalla principal de la aplicación un menú *Configuración* que al pulsarlo veremos la pantalla de configuración de la aplicación (Fig. 78), en la que podremos configurar los distintos parámetros de configuración de la aplicación.

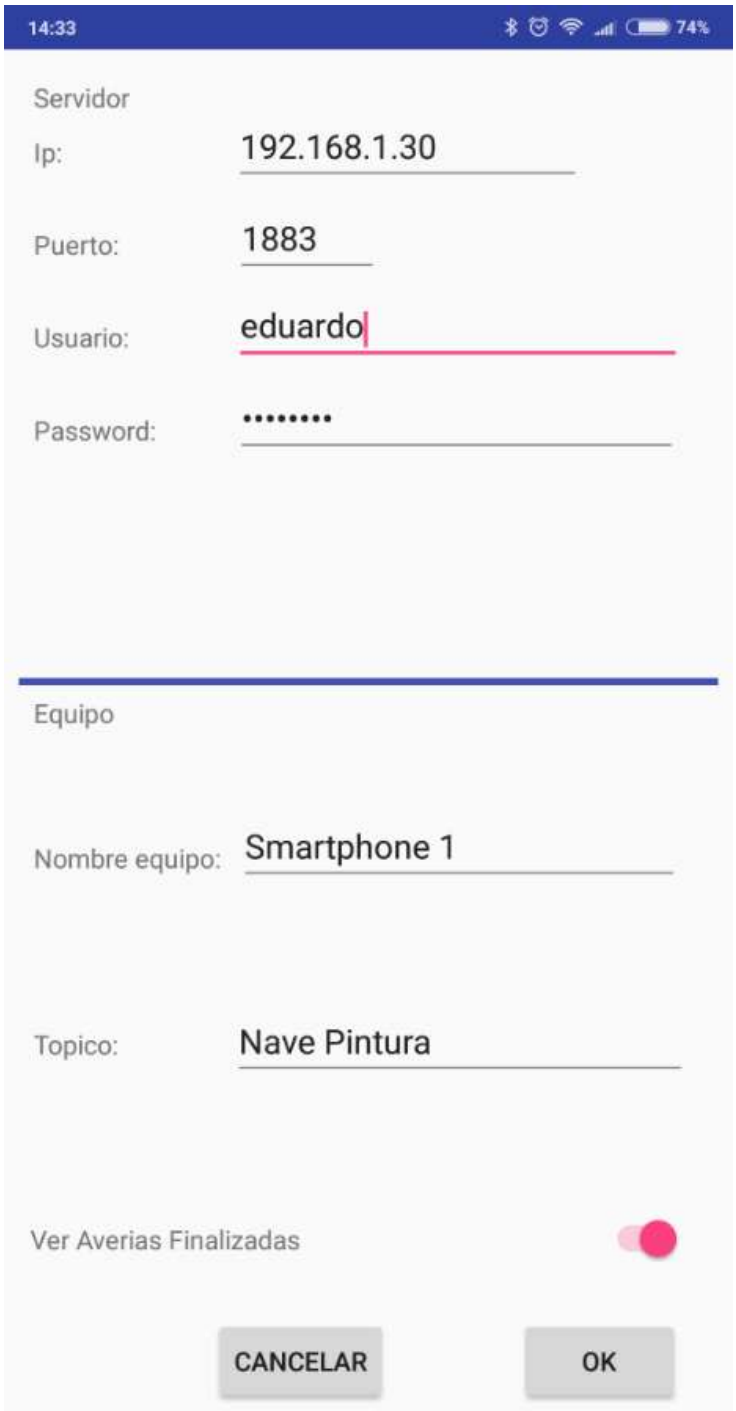

Fig. 78 Pantalla de configuración APP recepción de averías.

El desarrollo de la aplicación Android para este prototipo se realizará en el IDE de libre distribución *Android Studio* que podemos descargarlo de su página oficial (https://developer.android.com/studio/?hl=es-419).

Para la conexión con el broker MQTT desde nuestra aplicación Android utilizaremos las librerías del proyecto *Paho* de la fundación *Eclipse* diseñadas específicamente para su utilización en aplicaciones Android. Podemos descargar estas librerías así como su documentación de su repositorio oficial https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/org.eclipse.paho.android.service/ .

Esta librería proporciona las herramientas necesarias para integrar un cliente MQTT en una aplicación Android.

En los siguientes puntos se describe el desarrollo de la aplicación.

3.3.1. CLASE ebuetas.tfm.averiasandroid.Averias

Esta clase servirá para almacenar las averías tratadas en la aplicación. Además de las funciones *get / set* de los valores de las averías sobrescribe dos métodos.

El primer método sobrescrito es el método *equals* para poder realizar comparaciones entre dos objetos de esta clase, únicamente comparando los valores que definen de manera unívoca cada avería:

- Área
- Subárea
- Sistema
- Zona
- Elemento
- Nombre

Por otro lado, sobrescribe el método *toString* para devolver la información de la avería el siguiente formato:

Avería *nombre* en área / *subárea* / *sistema* / *zona* / *elemento* / Estado: *estado*

Podemos ver la estructura de la clase en Fig. 79.

Averias
- Id_averia: long - area: String - subarea: String - sistema: String - zona: String - elemento: String - nombre: String - mensaje: String - descripcion: String - actuacion: String - prioridad: int - id_tipo_averia: long - hora: String
+ Averias(String: _area, String: _subarea, String: _sistema, String _zona, String: _elemento, String _nombre, int _estado, String _hora) + getId_averia (): long + getArea (): String + getSubarea (): String + getSistema (): String + getZona (): String + getElemento (): String + getNombre (): String + getMensaje (): String + getDescripcion (): String + getActuacion (): String + getPrioridad (): int + getId_tipo_averia (): long + getHora (): String + setId_averia (_Id_averia: long):void + setArea (_area:String):void + setSubarea (_subarea: String):void + setSistema (_sistema: String):void + setZona (_zona: String):void + setElemento (_elemento: String):void + setNombre (_nombre: String):void + setMensaje (_mensaje: String):void + setDescripcion (_descripcion: String):void + setActuacion (_actuacion: String):void + setPrioridad (_prioridad: int):void + setId_tipo_averia (_id_tipo_averia: long):void + setHora (_hora: String):void + equals(obj:Object): boolean + toString (): String

Fig. 79 Clase "Averias" APP recepción averías.

3.3.2. CLASE ebuetas.tfm.averiasandroid.Configuracion

En un objeto de esta clase se almacenará los datos de configuración de la aplicación, estos datos de configuración se almacenarán como los datos de preferencias compartidas de esta aplicación, utilizando un objeto de la clase de Android *SharedPreferences*. Los objetos de esta clase tendrán las funciones para leer (*LeerConfiguracion)* y escribir (*EscribirConfiguracion)*. La estructura de la clase podemos verla en la siguiente figura Fig. 80.

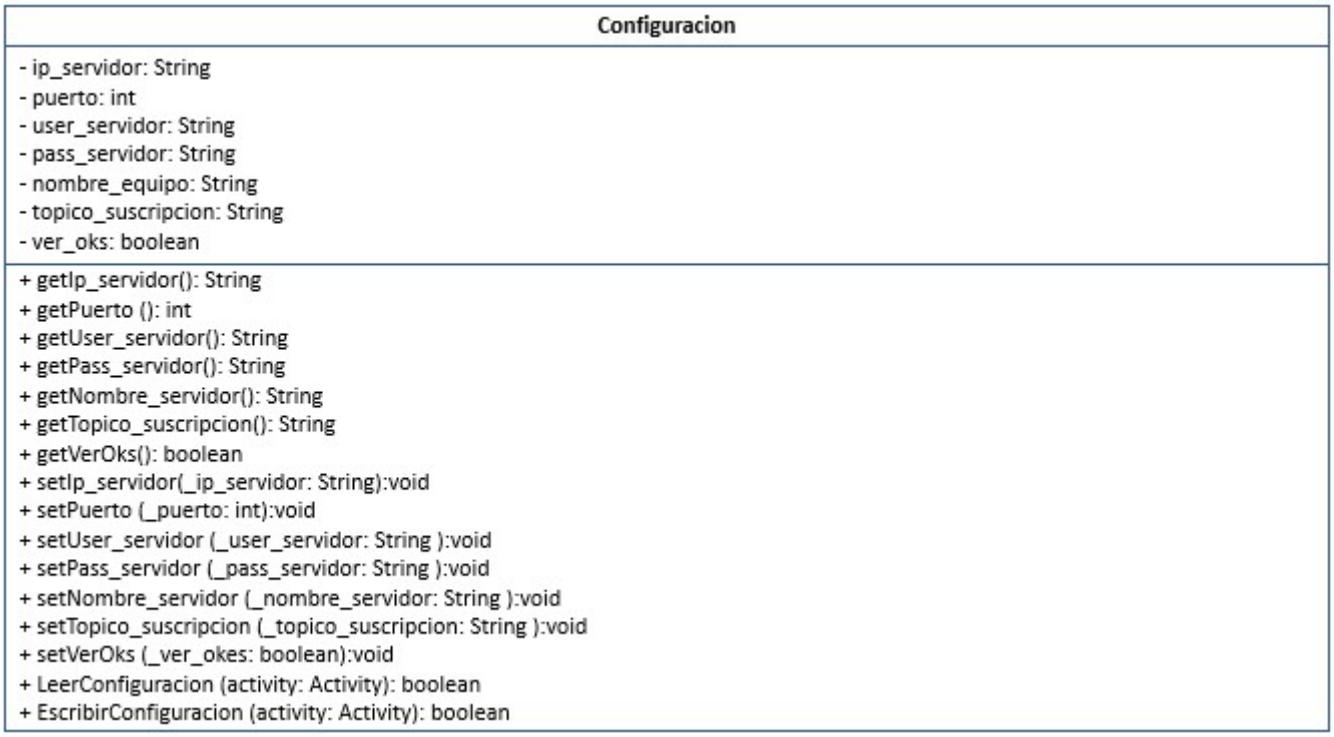

Fig. 80 Clase "Configuracion" APP recepción averías.

3.3.3. CLASE ebuetas.tfm.averiasandroid.FuncAux

En esta clase se codificarán las diferentes funciones auxiliares que se van a utilizar a lo largo de la aplicación: emisión de sonidos y vibraciones (*success_wav, fail_wav* y *beep_wav*), gestión de notificaciones (*notification, createNotificationChanel* y *BorrarNotificaciones*) y una función más para mostrar mensajes en la aplicación *MostrarAlert*.

Todas estas funciones están definidas como estáticas en la clase para poder ser utilizadas directamente sin necesidad de crear un objeto de la clase para su utilización.

3.3.4. CLASE ebuetas.tfm.averiasandroid.AveriasAdapter

Esta clase heredada de la clase BaseAdapter implementa el adaptador utilizado para mostrar cada avería en un *ListView* en la clase de la pantalla principal de esta aplicación, el layout de este adaptador este definido en el fichero xml *item_averias.xml* incluido en el directorio *res/layouts* de la aplicación.

Este adaptador en su función *getView* realiza la presentación de los datos de cada avería dentro de los adaptadores que conforman la *ListView* de la aplicación. Mostrando los datos necesarios en cada uno de los elementos y cambiando el color de fondo del layout de cada elemento dependiendo del estado de la avería. Podemos ver la estructura de la clase en la siguiente figura Fig. 81.

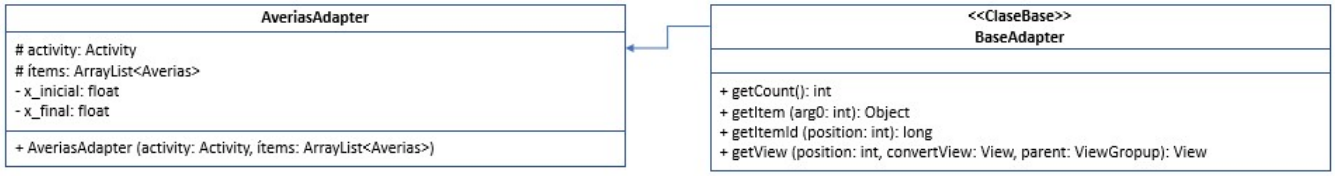

Fig. 81 Clase "AveriasAdapter" APP recepción averías.

3.3.5. CLASE ebuetas.tfm.averiasandroid.PrincipalActivity

En esta clase, heredada de *AppCompactActivity*, se crea la actividad principal de la aplicación, el layout de esta aplicación está en el fichero *activity_principal.xml* del directorio *res/layout* de la aplicación.

Esta actividad es la que muestra la *ListView* principal de la aplicación donde se muestran las alarmas y desde la que se llama a las otras actividades de la aplicación, la actividad de configuración (punto 3.3.7) y la actividad de acuse y petición de detalles de cada avería (punto 3.3.6).

Para la creación de los objetos del layout en la clase y sus eventos se crean dos funciones auxiliares que son llamadas desde la función *onCreate* de la clase: *AsignarElementos* y *AsignarEventos.*

Para la recepción de los eventos *broadcast* enviados desde el servicio que se encarga de recibir los mensajes por MQTT (punto 3.3.8) se crea una nueva clase *ServiceReceiver* que hereda de la clase *BroadCastReceiver* y que gestionará los mensajes broadcast generados por el servicio MQTT, conexión ok, conexión fallida, perdida de conexión y mensaje recibido.

La estructura de estas dos clases la podemos ver en la Fig. 82.

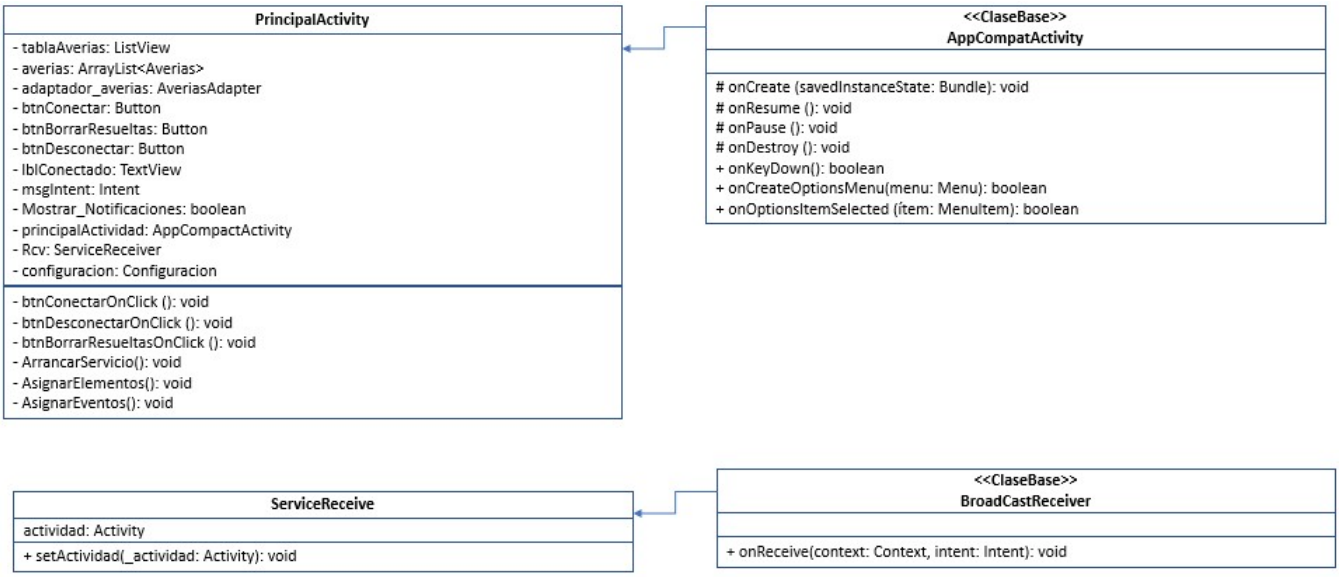

Fig. 82 Clase "PrincipalActivity" y clase auxiliar "ServiceReceiver" APP recepción averías.

3.3.6. CLASE ebuetas.tfm.averiasandroid.DetallesActivity

En esta clase, heredada de *AppCompactActivity*, se crea la actividad que permite al usuario acusar una avería o pedir las informaciones adicionales de una avería, es lanzada al pulsar sobre una avería de la actividad principal de la aplicación, el layout de esta aplicación está en el fichero *activity_detalles.xml* del directorio *res/layout* de la aplicación.

Para la creación de los objetos del layout en la clase y sus eventos se crean dos funciones auxiliares que son llamadas desde la función *onCreate* de la clase: *AsignarElementos* y *AsignarEventos.*

También en esta clase se crea un cliente MQTT suscrito al tópico "*Informacion/Nombre dispositivo/#"* para enviar los mensajes de acuse y petición de información y recibir las informaciones enviadas desde el servidor en respuesta a una petición de información generada por esta aplicación.

La estructura de la clase la podemos ver en la Fig. 83.

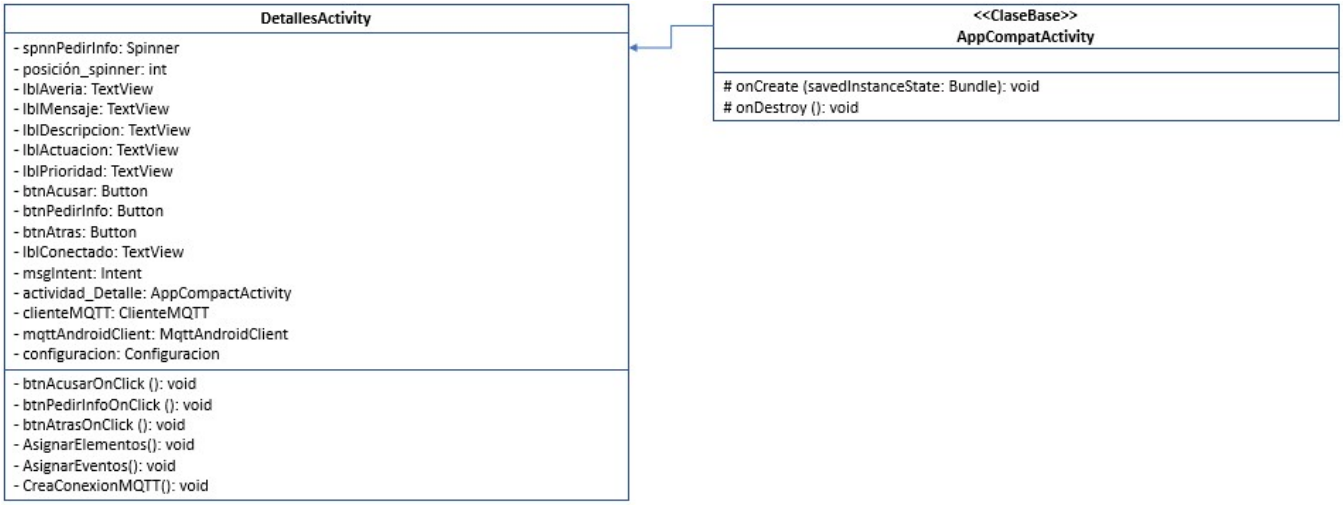

Fig. 83 Clase "DetallesActivity" APP recepción averías.

3.3.7. CLASE ebuetas.tfm.averiasandroid.ConfigActivity

En esta clase, heredada de *AppCompactActivity*, se crea la actividad que permite al usuario configurar la aplicación, es lanzada al pulsar sobre el menú configuración de la actividad principal de la aplicación, el layout de esta aplicación está en el fichero *activity_config.xml* del directorio *res/layout* de la aplicación.

Para la creación de los objetos del layout en la clase y sus eventos se crean dos funciones auxiliares que son llamadas desde la función *onCreate* de la clase: *AsignarElementos* y *AsignarEventos*, una vez creados los objetos se lanza la función *LeerConfiguracion* para rellenar los datos de configuración con los datos actuales.

Si pulsamos el botón *OK* se lanza la función *guardar_configuracion* para almacenar las configuraciones introducidas en las preferencias de la aplicación.

La estructura de la clase la podemos ver en Fig. 84.

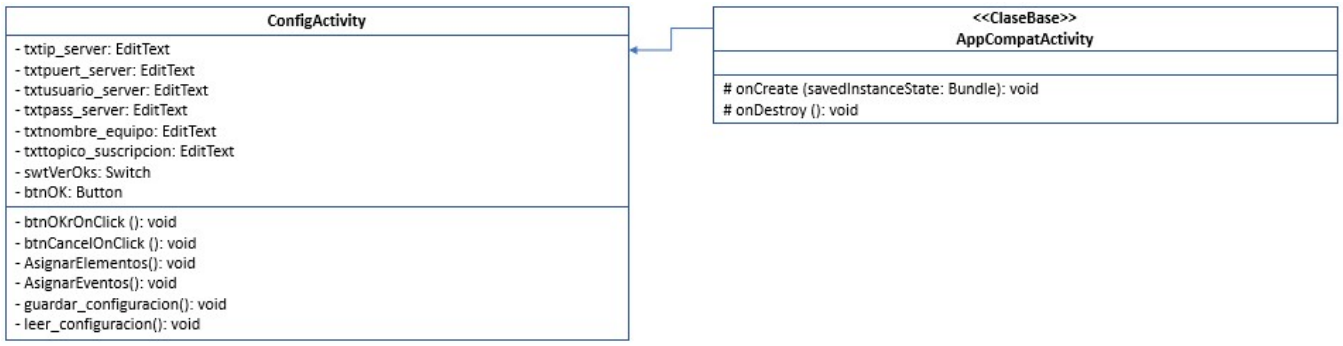

Fig. 84 Clase "ConfigActivity" APP recepción averías.

3.3.8. CLASE ebuetas.tfm.averiasandroid.mqtt.ServicioMQTT

Esta clase genera el servicio que mantendrá el cliente MQTT en segundo plano funcionando en todo momento desde que se conecte, mediante el botón creado para ello en la actividad principal, hasta que se desconecte, mediante el botón correspondiente también en la actividad principal.

Esta clase heredada de la clase *Service* y crea el cliente MQTT que recibirá las incidencias de las averías correspondientes dependiendo del tópico configurado en la aplicación.

Este servicio enviará los datos recibidos mediante su suscripción MQTT a través de eventos broadcast que serán recibidos por la actividad principal. Para la recepción de estos mensajes se creará un *listener* para recibir los eventos generados por el cliente MQTT configurado.

Podemos ver la estructura de la clase en la siguiente figura (Fig. 85).

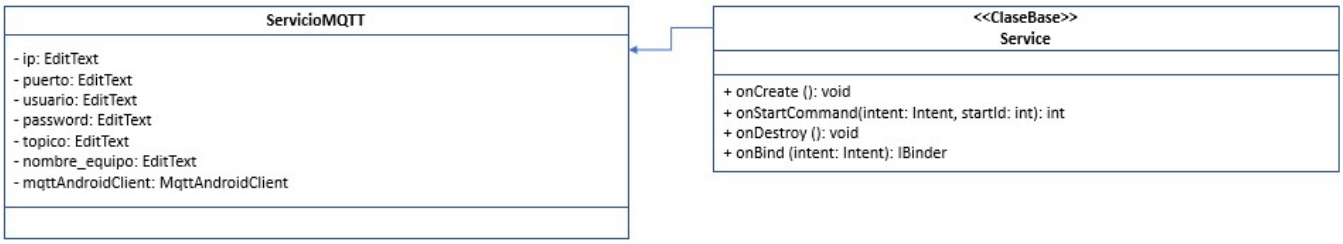

Fig. 85 Clase "ServicioMQTT" APP recepción averías.

3.4. PRUEBA DE FUNCIONAMIENTO

Para la prueba de funcionamiento utilizaremos las configuraciones de los equipos tal y como se han mostrado a lo largo del desarrollo de este prototipo.

Configurando nuestro receptor de averías para la recepción de las averías únicamente de la nave de pintura, colocando en la configuración del tópico de nuestro dispositivo *"Pintura"* por lo que recibirá las averías generadas por nuestros prototipos basados en Raspberry Pi y Arduino, no recibiendo las averías generadas por nuestro prototipo basado en PLC.

3.4.1. PUBLICACIÓN AVERÍA

Primero generaremos una avería desde nuestro prototipo creado con Raspberry Pi que simula el controlador de automatización *"GF1"*, al pulsar el primer botón de este prototipo generamos la avería *"Tiempo Transito B1"* del elemento *"CR1"*, al generar esta avería podemos ver en el log del *broker* del servidor, que el dispositivo ha publicado la avería y ha sido enviada a sus suscriptores (Fig. 86):

```
Received PUBLISH from publicadorRBPi (d0, q2, r1, m8, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Sending PUBREC to publicadorRBPi (Mid: 8)
Received PUBREL from publicadorRBPi (Mid: 8)
Sending PUBCOMP to publicadorRBPi (Mid: 8)
Sending PUBLISH to Smartphone 1 (d0, q2, r0, m10, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Sending PUBLISH to suscriptor_server (d0, q2, r0, m6, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Received PUBREC from suscriptor_server (Mid: 6)
Sending PUBREL to suscriptor_server (Mid: 6)
Received PUBCOMP from suscriptor_server (Mid: 6)
Received PUBREC from Smartphone 1 (Mid: 10)
Sending PUBREL to Smartphone 1 (Mid: 10)
Received PUBCOMP from Smartphone 1 (Mid: 10)
Received PINGREQ from publicador_server
```

Fig. 86 Log Mosquitto publicación avería.

Vemos en el log que el *broker* ha recibido la publicación de la avería enviada por el controlador creado con la Raspberry Pi y lo ha enviado a los dos suscriptores que lo deben recibir, el servidor y el receptor de averías *Smartphone 1*. Cada uno de estos mensajes se negocia con cuatro mensajes para asegurar la recepción del mensaje una y una sola vez (QoS 2) vemos en Fig. 87 que el servidor ha incluido la avería en la tabla correspondiente *averias_abiertas (*punto 2.2.1.2.5*)*.

```
averias_tfm=# select m_averias.area, m_averias.subarea, m_averias.sistema, m_averias.zona, m_averias.elemento, m_averias.nombre, averias_abiertas.* from averias_abiertas, m
_averias where averias_abiertas.id_averia=m_averias.id_averia;
  area   |     subarea     | sistema |  zona  | elemento |       nombre       | id_averia_abierta | id_averia |        ts_inicio          | ts_acuse | id_tecnico_acusa
---------+-----------------+---------+--------+----------+--------------------+-------------------+-----------+---------------------------+----------+------------------
 Pintura | Transportadores | GF1     | Cota 0 | CR1      | Tiempo Transito B1 |               121 |        16 | 2018-05-13 11:04:10.760966 |          |
(1 row)

averias_tfm=#
```

Fig. 87 Avería abierta en tabla averias_abiertas.

En la Fig. 88 podemos ver como el receptor de averías muestra la nueva avería en su lista con fondo rojo.

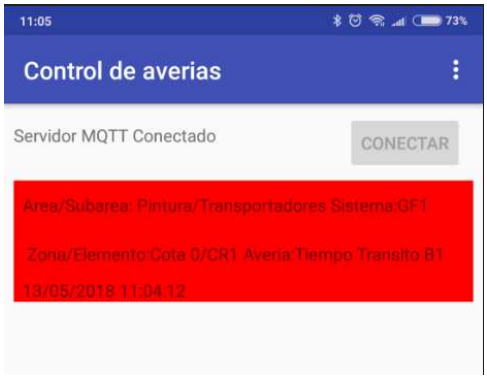

Fig. 88 Avería en lista de averías.

La siguiente figura (Fig. 89) muestra los procesos desencadenados por la generación de esta avería, en esta prueba solo disponemos de un receptor de averías, en el caso de tener más de un receptor de averías suscrito a este tópico la publicación les llegaría a todos:

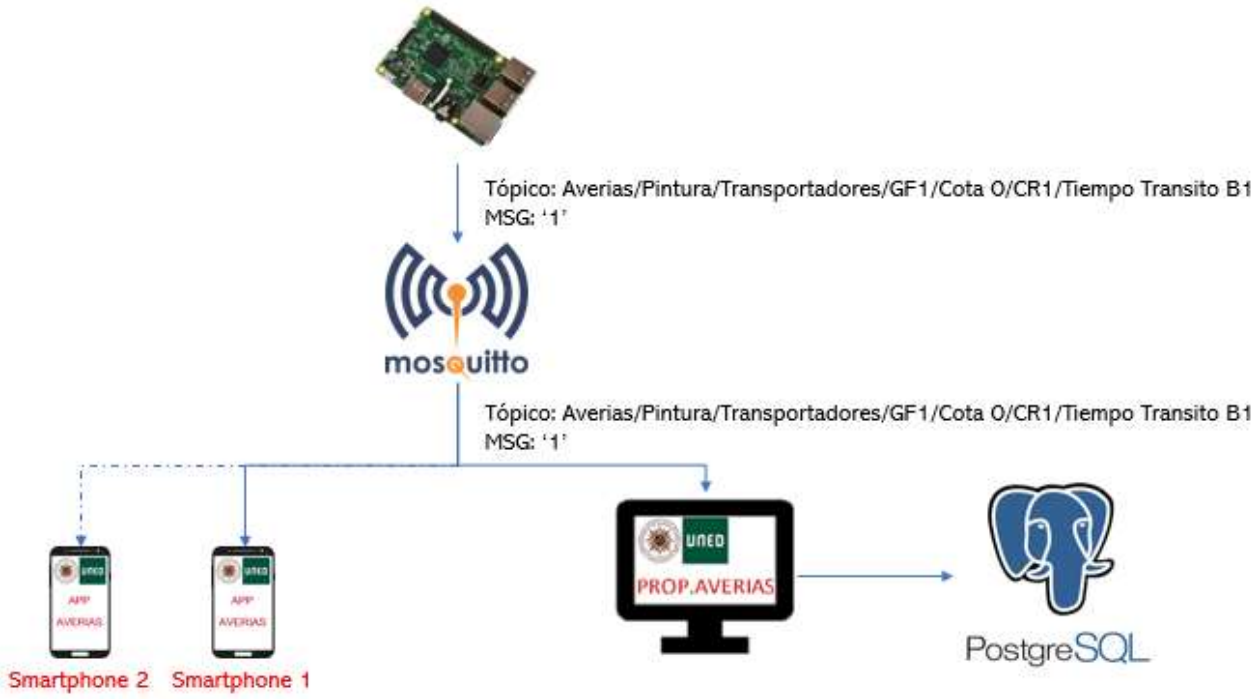

Fig. 89 Proceso al producirse una avería.

3.4.2. ACUSE AVERÍA

Acusamos la avería desde *Smartphone 1*, para ello pulsamos sobre la avería y en la pantalla que nos aparece pulsamos sobre el botón "*ACUSAR AVERIA".*

Vemos en el log del *broker* como el mensaje de acuse es recibido por el *broker* desde el receptor de averías *Smartphone 1* y enviado a él mismo y al servidor (Fig. 90).

```
Received PUBLISH from Smartphone 1_publicador (d0, q2, r0, m2, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (14 bytes))
Sending PUBREC to Smartphone 1_publicador (Mid: 2)
Received PUBREL from Smartphone 1_publicador (Mid: 2)
Sending PUBCOMP to Smartphone 1_publicador (Mid: 2)
Sending PUBLISH to suscriptor_server (d0, q2, r0, m8, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (14 bytes))
Sending PUBLISH to Smartphone 1 (d0, q2, r0, m15, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (14 bytes))
Received PUBREC from Smartphone 1 (Mid: 15)
Sending PUBREL to Smartphone 1 (Mid: 15)
Received PUBREC from suscriptor_server (Mid: 8)
Sending PUBREL to suscriptor_server (Mid: 8)
Received PUBCOMP from Smartphone 1 (Mid: 15)
Received PUBCOMP from suscriptor_server (Mid: 8)
Received DISCONNECT from Smartphone 1_publicador
```

Fig. 90 Log Mosquitto acuse avería.

El servidor añade en la base de datos el sello de tiempo del acuse de avería y su acusador Fig. 91 en el correspondiente registro de la tabla *averias_abiertas*.

```
averias_tfm=#  select m_averias.area, m_averias.subarea, m_averias.sistema, m_averias.zona, m_averias.elemento, m_averias.nombre, averias_abiertas.* from averias_abiertas, m_averias where
averias_abiertas.id_averia=m_averias.id_averia;
  area    |     subarea      | sistema |  zona   | elemento |       nombre       | id_averia_abierta | id_averia |         ts_inicio          |         ts_acuse           | id_tecnico_acusa
----------+------------------+---------+---------+----------+--------------------+-------------------+-----------+----------------------------+----------------------------+------------------
 Pintura | Transportadores | GF1     | Cota 0 | CR1      | Tiempo Transito B1 |               121 |        16 | 2018-05-13 11:04:10.760966 | 2018-05-13 11:05:47.243 |                1
(1 row)

averias_tfm=#
```

Fig. 91 Avería acusada en base de datos.

El receptor de averías al recibir el mensaje cambia su estado en la lista de averías del receptor Fig. 92, cambiando el fondo a amarillo.

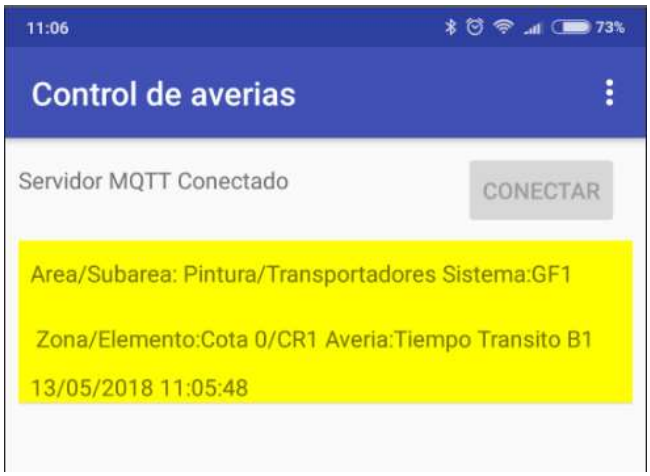

Fig. 92 Avería acusada en lista de averías.

En el caso de tener más equipos configurados para recibir esta información también la recibirían, y también cambiarían su estado por *"acusada"*. Podemos ver el esquema de este proceso en la siguiente figura (Fig. 93):

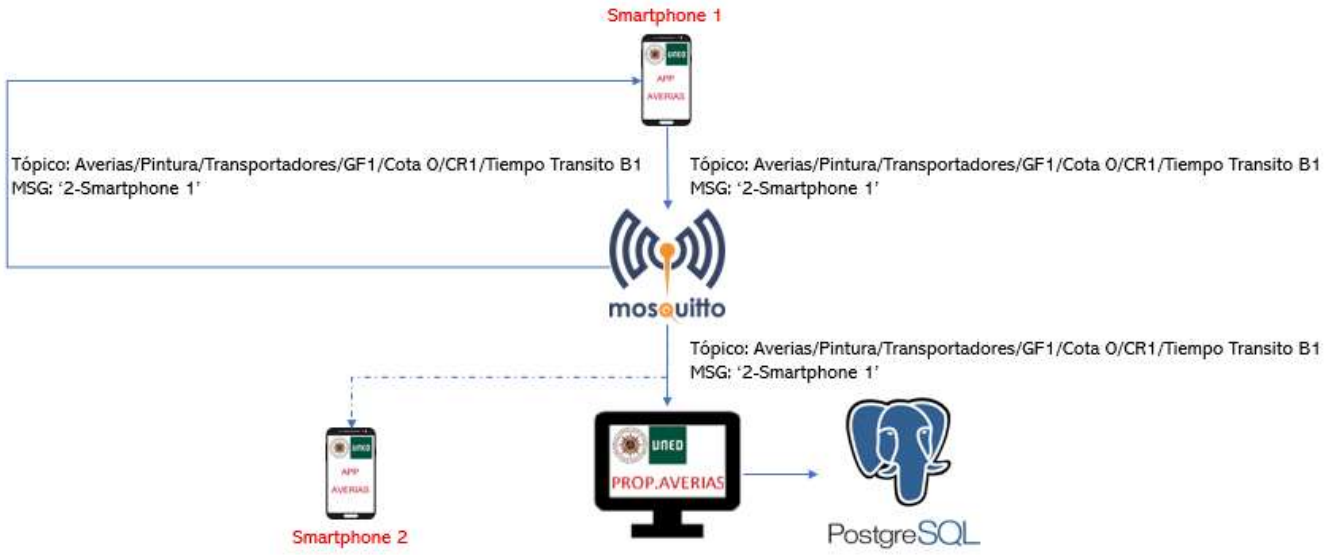

Fig. 93 Proceso al acusarse una avería.

3.4.3. PETICIÓN DE INFORMACIÓN

En nuestra prueba la información requerida será la actuación propuesta, para ello en la aplicación de recepción de averías de *Smartphone 1* pulsaremos sobre la avería y en la pantalla que nos aparece seleccionaremos *"Actuación"* en el *spinner* de selección y pulsaremos el botón "*PEDIR"*.

Vemos en el log del *broker* los mensajes que se intercambian los participantes de la comunicación Fig. 94, primero el peticionario de la información *Smartphone 1* envía el mensaje de petición con el tópico de la avería, que es recibido por el mismo (que lo descarta) y por el servidor y este publica un mensaje con la información requerida en el tópico "*Informacion/Smartphone 1/actuacion".*

```
Received PUBLISH from Smartphone 1_publicador (d0, q2, r0, m2, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (24 bytes))
Sending PUBREC to Smartphone 1_publicador (Mid: 2)
Received PUBREL from Smartphone 1_publicador (Mid: 2)
Sending PUBCOMP to Smartphone 1_publicador (Mid: 2)
Sending PUBLISH to suscriptor_server (d0, q2, r0, m7, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (24 bytes))
Sending PUBLISH to Smartphone 1 (d0, q2, r0, m34, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (24 bytes))
Received PUBREC from suscriptor_server (Mid: 7)
Sending PUBREL to suscriptor_server (Mid: 7)
Received PUBREC from Smartphone 1 (Mid: 34)
Sending PUBREL to Smartphone 1 (Mid: 34)
Received PUBLISH from publicador_server (d0, q2, r0, m1, 'Informacion/Smartphone 1/actuacion', ... (140 bytes))
Sending PUBREC to publicador_server (Mid: 1)
Received PUBCOMP from Smartphone 1 (Mid: 34)
Received PUBREL from publicador_server (Mid: 1)
Sending PUBCOMP to publicador_server (Mid: 1)
Sending PUBLISH to Smartphone 1_publicador (d0, q2, r0, m1, 'Informacion/Smartphone 1/actuacion', ... (140 bytes))
Received PUBCOMP from suscriptor_server (Mid: 7)
Received PUBREC from Smartphone 1_publicador (Mid: 1)
Sending PUBREL to Smartphone 1_publicador (Mid: 1)
Received PUBCOMP from Smartphone 1_publicador (Mid: 1)
```

Fig. 94 Log Moaquitto, petición y envío información.

En la aplicación la información aparece en la pantalla de detalles desde donde ha sido requerida (Fig. 95).

Fig. 95 Información sobre la actuación requerida, prueba 1.

El esquema del proceso se divide en dos pasos, la petición de información (Fig. 96) y la devolución de esta información (Fig. 97).

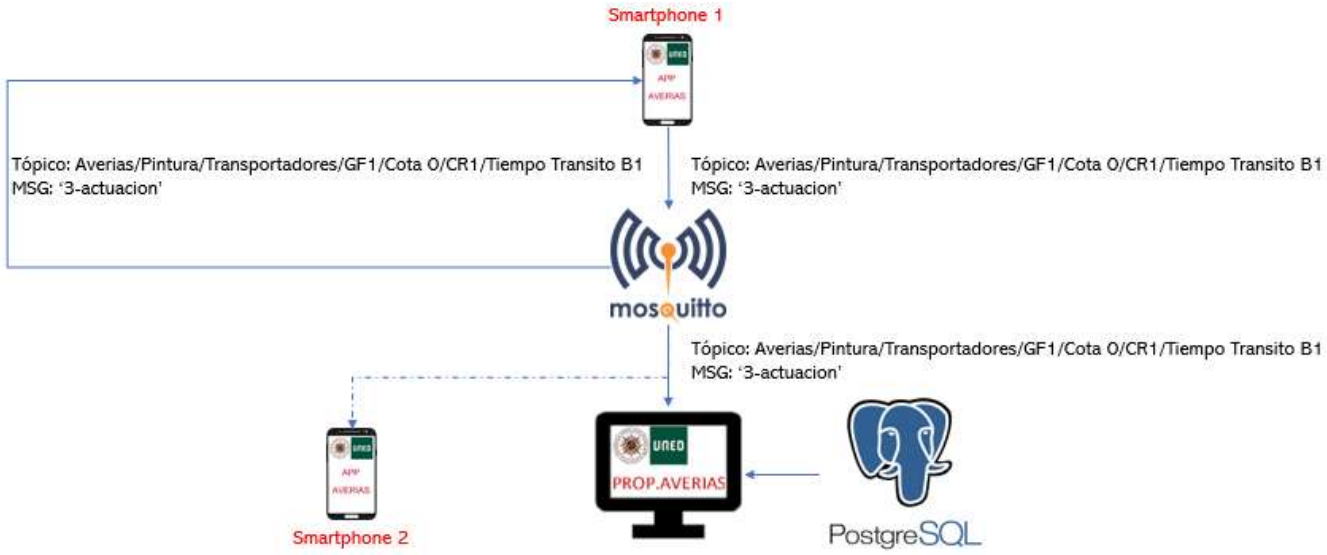

Fig. 96 Paso 1: petición de información.

Fig. 97 Paso 2: envío de la información.

3.4.4. DESAPARICIÓN DE AVERÍA

Para provocar la desaparición de la avería volvemos a pulsar el pulsador de la avería 1 de nuestro controlador desarrollado con Rasbperry Pi y la avería es finalizada, vemos los mensajes intercambiados, el *broker* recibe la publicación de la Raspberry Pi y envía el mensaje al receptor de averías *Smartphone 1* y al servidor (Fig. 98).

```
Received PUBLISH from publicadorRBPi (d0, q2, r1, m6, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Sending PUBREC to publicadorRBPi (Mid: 6)
Received PUBREL from publicadorRBPi (Mid: 6)
Sending PUBCOMP to publicadorRBPi (Mid: 6)
Sending PUBLISH to suscriptor_server (d0, q2, r0, m8, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Sending PUBLISH to Smartphone 1 (d0, q2, r0, m35, 'Averias/Pintura/Transportadores/GF1/Cota 0/CR1/Tiempo Transito B1', ... (1 bytes))
Received PUBREC from suscriptor_server (Mid: 8)
Sending PUBREL to suscriptor_server (Mid: 8)
Received PUBREC from Smartphone 1 (Mid: 35)
Sending PUBREL to Smartphone 1 (Mid: 35)
Received PUBCOMP from suscriptor_server (Mid: 8)
Received PUBCOMP from Smartphone 1 (Mid: 35)
```

Fig. 98 Log Mosquitto desaparición avería.

El servidor borra la avería de la tabla *averias_abiertas* y la coloca en *averias_cerradas* (punto 2.2.1.2.6) (Fig. 99)

```
averias_tfm=#  select m_averias.area, m_averias.subarea, m_averias.sistema, m_averias.zona, m_averias.elemento, m_averias.nombre, averias_abiertas.* from averias_abiertas, m_averias where averias_abiertas.id_averia=
m_averias.id_averia;
 area | subarea | sistema | zona | elemento | nombre | id_averia_abierta | id_averia | ts_inicio | ts_acuse | id_tecnico_acusa
------+---------+---------+------+----------+--------+-------------------+-----------+-----------+----------+-----------------
(0 rows)

averias_tfm=#  select m_averias.area, m_averias.subarea, m_averias.sistema, m_averias.zona, m_averias.elemento, m_averias.nombre, averias_cerradas.* from averias_cerradas, m_averias where averias_cerradas.id_averia=
m_averias.id_averia;
  area   |   subarea    | sistema |  zona  | elemento |       nombre       | id_averia_cerrada | id_averia |         ts_inicio          |         ts_acuse           | id_tecnico_acusa |          ts_fin
---------+--------------+---------+--------+----------+--------------------+-------------------+-----------+----------------------------+----------------------------+------------------+----------------------------
 Pintura | Transportadores | GF1     | Cota 0 | CR1      | Tiempo Transito B1 |               195 |        16 | 2018-05-13 11:04:10.760966 | 2018-05-13 11:12:15.116 |                1 | 2018-05-13 11:20:26.895244
(1 row)

averias_tfm=#
```

Fig. 99 Avería cerrada en tabla averias_cerradas.

En el receptor de averías cambia de estado y aparece con fondo verde. Si tuviéramos configurada la aplicación para que las averías terminadas desaparecieran, esta hubiera desaparecido, al tenerlo configurado para verlas hasta que pulsemos el botón "*Borrar Resueltas"* esta avería continuará en la lista de averías (Fig. 100).

Fig. 100 Avería finalizada en APP recepción averías.

El proceso se puede ver de manera esquemática en Fig. 101

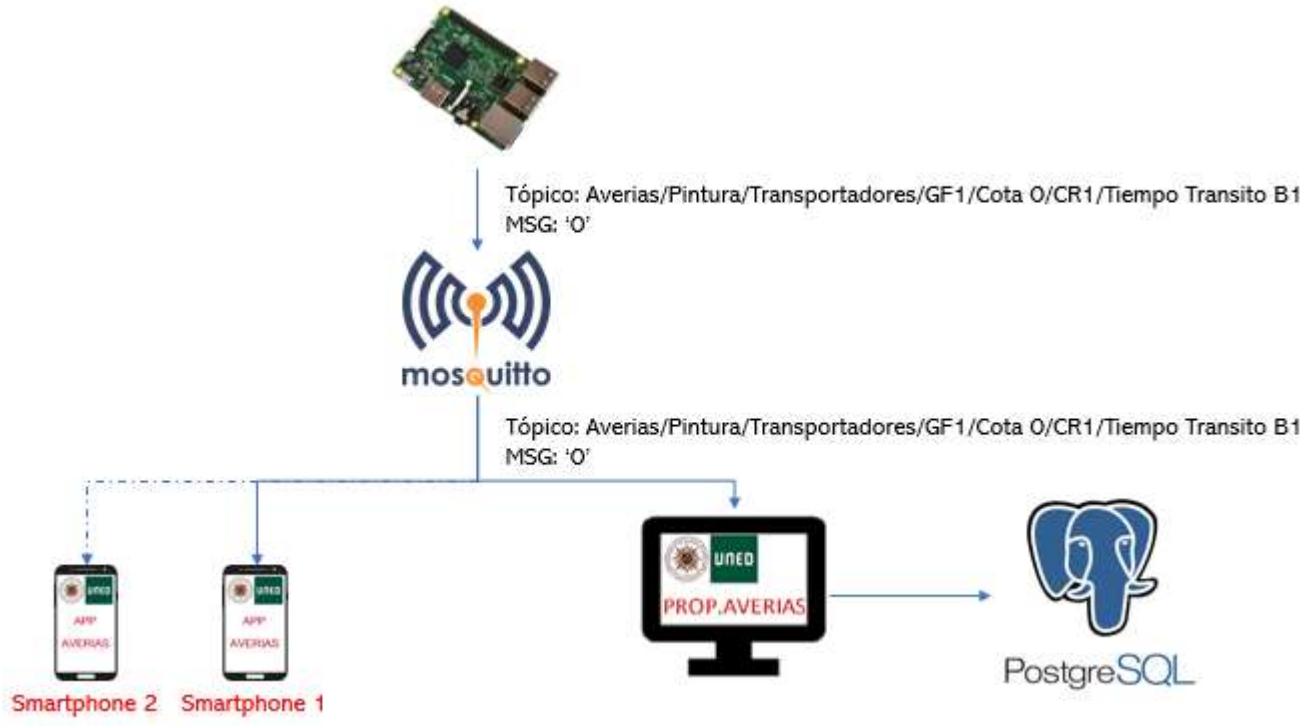

Fig. 101 Proceso resolución de avería.

4. CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS

4.1. CONCLUSIONES

A lo largo del desarrollo de este TFM se ha pretendido exponer el estado actual y futuro de las comunicaciones industriales y la problemática que existe y que se incrementará con la introducción de los nuevos sistemas de control industrial en el ámbito de la industria 4.0.

Estos nuevos elementos de control que van a tener que convivir con los sistemas de control actuales, y ambos, los sistemas de control actuales habitualmente controlados por sistemas PLC y los nuevos que serán controlados por otros tipos de sistemas de control tendrán que propagar informaciones a los sistemas superiores de control industrial, como pueden ser sistemas de control de producción, MES, ERPs, propagadores de averías, etc…

En la mayoría de los trabajos hasta la fecha se parte de la premisa que para conseguir que estas comunicaciones sean posibles, será necesaria la inclusión de *gateways* para saltar entre las diferentes capas de red existentes entre los sistemas de control industrial y los servidores superiores de gestión de fabricación. En este trabajo se pretende demostrar que para la comunicación de los sistemas de automatización industrial actuales y futuros, y utilizando los protocolos existentes en la actualidad, es posible realizar esta comunicación sin tener que utilizar estas *gateways*, esto es posible debido a que la inmensa mayoría de los sistemas de control industrial actuales (PLC) y los que se utilizan en la industria 4.0 tienen capacidades de comunicación TCP/IP, con lo que utilizando un protocolo liguero, que utilice como protocolo de transporte el TCP, es posible integrar en todos los controladores en una red para una comunicación directa entre los sistemas de automatización industrial y los servidores superiores de la industria.

Con este objetivo en este trabajo se realiza un prototipo en el que se incluyen controladores de automatización realizados con tres tipos de dispositivos y valiéndose del protocolo MQTT, muy extendido en el ámbito del IoT, se consigue realizar un propagador de averías sin la necesidad de la inclusión de ninguna *gateway* en el sistema.

4.2. LÍNEAS DE INVESTIGACIÓN FUTURAS

Este trabajo no pretende ser un estudio final de las comunicaciones industriales en la industria 4.0, sino más bien un punto de partida para una investigación futura más amplia sobre este tema.

Quedan abiertos muchos temas de investigación que pueden tener como base este trabajo. Si bien el prototipo presentado en este trabajo es funcional para una demostración académica, deja sin resolver algunas cuestiones, como son el tema de la seguridad de las comunicaciones, la configuración de las redes de comunicación, la posibilidad de redundancia de los servidores, etc…

En este trabajo se realiza la autentificación contra el *broker* MQTT únicamente con un usuario y contraseña que viajan en texto plano por la red, al igual que los mensajes, y aunque podamos suponer que las redes de las factorías son seguras, debemos plantearnos en una estrategia de seguridad en profundidad, realizar estas comunicaciones a través de protocolos seguros que impidan que un ataque *man-in-the-middle* puedan vulnerar la seguridad de nuestro sistema.

Por otro lado, uno de los objetivos que pretende la solución propuesta en este trabajo es la mínima configuración de los equipos nuevos introducidos en el sistema, y si bien esta configuración es mínima en los equipos, queda sin resolver el problema de la configuración de las redes por las que viajan los mensajes del sistema. En la actualidad para la inclusión de un elemento en la red de una factoría se deben configurar todos los puntos por los cuales sus mensajes deben circular (switch, routers, firewalls, etc…), por lo que una futura línea de investigación paralela, después de la investigación realizada en este TFM, debería ser la utilización de redes definidas por software (SDN, por sus siglas en ingles) en el ámbito industrial.

En el prototipo de este documento se trabaja con el *broker* MQTT Mosquitto y un software a parte para la gestión del servidor de propagación de averías, otra línea de investigación abierta en este documento es la realización de los servidores de la industria, como el servidor de propagación de averías, incluyendo en su código el *broker* MQTT. De este modo se podría tener el control total del funcionamiento de dicho *broker* incluyendo las medidas de seguridad y balanceo que un sistema de estas características requiere en un ámbito tan exigente como es el ámbito industrial.

Por último, para la implantación de este sistema en la industria sería clave la creación de un generador automático de código que generará tanto los módulos de código para la

implementación de las comunicaciones MQTT en los diferentes sistemas como las listas de averías de cada sistema de automatización, para poder ser importadas de manera automática en la base de datos del sistema.

Este alumno en un trabajo anterior correspondiente a la asignatura de Generación Automática de Código ha realizado un trabajo de investigación en esta dirección que bien podría ser la base de una futura investigación en esta línea.

5. BIBLIOGRAFÍA

[1] Communication Promoters Group of the Industry-Science, «Recommendations for implementing the strategic initiative Industrie 4.0,» National Academy of Science and Engineering, Frankfurt, 2013.

[2] M. S. Mahmoud y X. Yuanquing, «The Interaction between Control and Cmputing Theories: New Approaches,» *International Journal of Automation and Computing,* vol. 3, nº 14, pp. 254-274, 2017.

[3] I. Zolatová, M. Bundzel y T. Lojka, «Industry IoT Gateway for Could Connectivity,» de *IFIP International Conference on Advances in Production Managment Systems*, 2015.

[4] A. Astarloa, J. Jiménez, A. Zuloaga y J. Lázaro, «Intelligent gateway for Industry 4.0-compliant production,» de *IECON 2016-42nd Annual Conference of the IEEE Industrila Electronics Society*, Florence, 2016.

[5] Network Working Group, «The Constrained Application Protocol (CoAP),» Internet Engineering Task Force (IETF), junio 2014. [En línea]. Available: https://tools.ietf.org/html/rfc7252. [Último acceso: 14 abril 2018].

[6] J. H. Saltzer y M. S. Schroeder, «The Protection of Information in Computer Systems.,» *Proccedings of the IEEE 63,* nº 9, pp. 1278-1308, 1975.

[7] Amazon Web Service Inc, «Como funciona AWS IoT,» Amazon, 2017. [En línea]. Available: https://docs.aws.amazon.com/es_es/iot/latest/developerguide/aws-iot-how-it-works.html. [Último acceso: 07 01 2018].

[8] M. Yuan, «Explore MQTT and the Internet of Things service,» 22 Noviembre 2017. [En línea]. Available: https://www.ibm.com/developerworks/cloud/library/cl-mqtt-bluemix-iot-node-red-app/index.html. [Último acceso: 07 01 2018].

[9] Google Cloud Platform, «Google Cloud Internet of Things Core,» Google, [En línea]. Available: https://cloud.google.com/iot/docs/how-tos/mqtt-bridge. [Último acceso: 07 Enero 2018].

[10] S. e. a. Katsikeas, «Lightweight & Secure Industrial IoT Communications via the MQ Telemetry Transport Protocol,» de *IEEE Symposium on Computers and Communications (ISCC)*, 2017.

[11] P. Ferrari, E. Sisinni, D. Brandao y M. Rocha, *Evaluation of communication latency in Industrial IoT applications,* Naples, 2017, pp. 1-6.

[12] M. Iglesias, A. Orive, A. Moran, J. Bilbao y A. Urbieta, *Towards a lightweight protocol for Industry 4.0: An implementation based benchmark,* Donostia-San Sebastian, 2017, pp. 1-6.

[13] R. J. Cohn y R. J. Coppen, «MQTT Version 3.1.1,» 29 Octubre 2014. [En línea]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. [Último acceso: 09 01 2018].

[14] OWASP Foundation, «OWASP Top 10 -2017,» 2017. [En línea]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf. [Último acceso: 11 Marzo 2018].

6. SIGLAS

API	Aplication Protocol Interface
CoAP	Constrained Application Protocol
CPS	Cyber Phisical Systems
CRUD	Create Retrive Update Delete
ERP	Enterprise Resource Planning
HTML	Hyper Text Markup Languaje
IDE	Integrated Development Environment
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
OASIS	Advancing Open Standards for the Information Society
PLC	Programmable Logic Controller
QoS	Quality of Service
RAM	Random Access Memory
ROM	Read Only Memory
SGBD	Sistema de Gestion de Bases de Datos
TCP	Transmision Control Protocol
TFM	Trabajo Fin de Máster
UDP	User Datagram Protocol

7. ANEXO I: FICHEROS ENTREGADOS

Además del presente documento en esta entrega se envían una serie de documentos que son los resultados de algunas de las pruebas realizadas en el desarrollo de este TFM.

- *TFM_EduardoBuetas.pdf*: El presente documento que contiene el desarrollo de la memoria de este TFM.
- *averias_tfm.sql:* fichero que contiene el script sql para crear la base de datos utilizada para las pruebas realizadas en este TFM.
- *Averias40RBPi.zip:* fichero comprimido que contiene el proyecto NetBeans del programa de gestión de las averías sobre el prototipo realizado sobre Raspberry Pi escrito en java.
- *Averias40RBPi_JavaDoc.zip:* fichero comprimido que contiene la documentación, autogenerada con la herramienta JavaDoc, del programa *Averias40RBPi*.
- *ComprobarAverias40RBPi.zip:* fichero comprimido que contiene el proyecto NetBeans del *Watchdog* para comprobar que el funcionamiento del programa *Averias40RBPi* escrito en java.
- *ComprobarAverias40RBPi_JavaDoc.zip:* fichero comprimido que contiene la documentación, autogenerada con la herramienta JavaDoc, del programa *ComprobarAverias40RBPi*.
- *PropagadorAverias.zip:* fichero comprimido que contiene el proyecto NetBeans del programa *PropagadorAverias* del servidor de propagación de averías realizado para el prototipo realizado en este TFM, escrito en java.
- *PropagadorAverias_JavaDoc.zip:* fichero comprimido que contiene la documentación, autogenerada con la herramienta JavaDoc, del programa PropagadorAverias.
- *ControlPropagadorAverias.zip:* fichero comprimido que contiene el proyecto NetBeans del *Watchdog* para comprobar que el funcionamiento del programa *ControlPropagadorAverias*, escrito en java.
- *ControlPropagadorAverias_JavaDoc.zip:* fichero comprimido que contiene la documentación, autogenerada con la herramienta JavaDoc, del programa *ControlPropagadorAverias*.

- *AveriasI40Arduino.zip:* fichero comprimido que contiene el Sketch de Arduino generado para el prototipo realizado en Arduino para este TFM.
- *Chapa_GF1.zip:* fichero comprimido que contiene el proyecto Tia Portal del prototipo realizado sobre PLC S7-1211C para este TFM.
- *AveriasAndriod.zip:* fichero comprimido que contiene el proyecto Android Studio de la app generada como prototipo de receptor de averías para este TFM.
- *AveriasAndriod_JavaDoc.zip:* fichero comprimido que contiene la documentación, autogenerada con la herramienta JavaDoc, del programa *AveriasAndriod*.