Protocolos de comunicación de nueva generación

PID_00249442

Francisco Vázquez Gallego
Xavier Vilajosana Guillén
Pere Tuset Peiró

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya

Índice

Introducción

El uso de los sistemas informáticos en el ámbito de la automatización industrial es un campo que empezó a expandirse a principios de los años noventa con la popularización de los ordenadores personales y las redes de comunicación. En ese momento, la integración de estos sistemas permitió la monitorización de los parámetros físicos de un proceso industrial, así como el control en tiempo real de su funcionamiento. Además, gracias a la capacidad de almacenamiento, se hizo posible analizar los datos históricos y tomar decisiones para optimizar los procesos.

Año tras año, la adopción de estas tecnologías en el ámbito industrial se ha ido ampliando gracias al incremento de las capacidades de computación y almacenamiento de datos, así como el desarrollo de nuevas tecnologías de comunicación. Pero a pesar de las ventajas que ofrecen estas tecnologías en el ámbito industrial, su uso todavía está lejos de considerarse óptimo debido a la falta de estandarización de estas tecnologías, que conlleva dificultades a la hora de llevar a cabo su implantación y mantenimiento.

Por otro lado, durante los últimos tiempos, hemos presenciado cómo el desarrollo de las tecnologías y los protocolos del ámbito de internet se han realizado de manera abierta y colaborativa, y se han convertido en una auténtica referencia a la hora de conectar dispositivos heterogéneos de manera ubicua. Gracias a estos protocolos, hoy en día es posible acceder a internet desde un dispositivo móvil conectado a la red celular del mismo modo que lo hacemos desde casa utilizando una conexión de fibra óptica desde un ordenador personal.

Actualmente, en el ámbito de la industria 4.0, la interoperabilidad de las tecnologías y los protocolos de comunicación, así como los modelos estandarizados para la representación de la información, son considerados como uno de los aspectos clave para conseguir la flexibilidad de los sistemas de monitorización y control en el ámbito de la industria. Esta interoperabilidad permite desplegar sistemas de heterogéneos, garantizando su correcto funcionamiento y permitiendo la integración de la información para desarrollar conceptos como, por ejemplo, la logística avanzada o el mantenimiento predictivo.

Teniendo en cuenta esta problemática, en este módulo explicamos los nuevos protocolos de comunicación de la capa de aplicación que se utilizan para el intercambio de datos en tiempo real. En particular, centramos la explicación en los protocolos COAP (Constrained Application Protocol) y MQTT (Message Queue Telemetry Transport), procedentes del ámbito de las tecnologías de la

información y la comunicación, así como en el protocolo OPC/UA (Open Platform Communications/Unified Architecture), diseñado específicamente para la monitorización y el control en el ámbito industrial.

1. COAP

1.1. Introducción

CoAP (Constrained Application Protocol) es un protocolo de la capa de aplicación diseñado específicamente para el transporte de información en aplicaciones M2M (Machine-to-Machine), por ejemplo, en el ámbito de la automatización de edificios (*building automation*) y de red eléctrica inteligente (*smart grid*). En estos entornos, los dispositivos que se despliegan para la monitorización y el control suelen alimentarse con baterías, o incluso recogiendo la energía del ambiente mediante dispositivos de captura de energía (en inglés, *energy harvesters*), lo que comporta dos restricciones principales, tal como se describe en el RFC 7228 de la IETF (Internet Engineering Task Force).

IETF

La Internet Engineering Task Force (IETF) es un comité de estandarización abierto que desarrolla estándares de comunicaciones para las arquitecturas de internet. Son el comité que estandarizó el Internet Protocol (IP) así como los protocolos TCP y UDP, que son la base de la internet actual.

RFC

Los RFC (Request For Comments) son los documentos publicados por los diferentes grupos de trabajo de la IETF (Internet Engineering Task Force) donde se describe el funcionamiento de los protocolos de internet, así como otros aspectos relacionados. Por ejemplo, el RFC 791 describe el funcionamiento básico del protocolo IP (Internet Protocol), mientras que el RFC 2616 describe el funcionamiento básico del protocolo HTTP (HyperText Transfer Protocol).

En primer lugar, en el ámbito computacional, los dispositivos de comunicaciones tienen una capacidad de cómputo y almacenamiento de datos limitada. Por ejemplo, la CPU de estos dispositivos suele funcionar a menos de 100 MHz y suele tener entre 10 y 100 kbytes de espacio para variables (memoria RAM) y entre 100 y 1.000 kbytes de espacio para el programa (memoria FLASH).

En cuanto a las comunicaciones, suelen ser mediante tecnologías inalámbricas de bajo consumo, lo que limita el ancho de banda disponible y está sujeto a pérdida de paquetes debido a colisiones e interferencias. Por ejemplo, utilizando la tecnología inalámbrica basada en el estándar IEEE 802.15.4, las comunicaciones ofrecen una velocidad de transmisión de 250 kbps con una distancia de comunicación del orden de 100 metros en exterior y 50 metros en interior.

Teniendo en cuenta estas restricciones, Zach Shelby, fundador de la empresa Sensinode, y Carsten Borman, profesor de la Universidad de Bremen, empezaron a trabajar en el diseño del protocolo CoAP en el año 2010, pero no fue hasta el año 2013 cuanto CoAP fue aprobado por la IETF (Internet Engineering Task Force) a través de su grupo de trabajo CORE (Constrained RESTful Environments) y el protocolo pasó a formar parte de los estándares de esta organización bajo el número RFC 7252.

El principio de diseño del protocolo COAP es el uso de la misma arquitectura REST (Representational State Transfer) utilizada en el protocolo HTTP (HyperText Transfer Protocol), que tiene las siguientes características:

- **Protocolo cliente/servidor sin estado**: Los mensajes son autocontenidos y se transmiten de manera asíncrona, de modo que ni el cliente ni el servidor tienen que mantener el estado de las comunicaciones.

- **Operaciones bien definidas aplicadas a todos los recursos**: El protocolo solo define un pequeño conjunto de operaciones entre las que destacan POST, GET, PUT y DELETE.

- **Identificación de recursos universal**: El protocolo define un mecanismo para el descubrimiento de los servicios y recursos ofrecidos por un servidor basado en URI (Universal Resource Identifier).

A continuación, se describe el funcionamiento general del protocolo CoAP.

1.2. Modelo de comunicación

Como se ha introducido anteriormente, el protocolo CoAP define una arquitectura cliente/servidor y un modelo de interacción basado en petición/respuesta de manera similar a HTTP. Pero, a diferencia de HTTP, que utiliza TCP (Transmission Control Protocol) como protocolo de la capa de transporte, CoAP utiliza el protocolo UDP (User Datagram Protocol) para el intercambio de mensajes entre cliente y servidor.

Si bien esto supone una ventaja en cuanto al consumo de recursos, el uso de UDP presenta algunos problemas al tratarse de un protocolo no orientado a conexión, es decir, que no garantiza la entrega de los mensajes. Para suplir esta limitación, el protocolo CoAP implementa la confirmación de recepción de los mensajes en la capa de aplicación.

CoAP también incorpora un mecanismo de notificación (conocido también como mecanismo de publicación/suscripción) de mensajes a los que es necesario suscribirse. Este mecanismo es conocido como CoAP Observe y está estandarizado en el RFC 7641 de la IETF. El mecanismo CoAP Observe consiste, básicamente, en la implementación de un patrón observador donde el cliente se registra mediante un mensaje Observe indicando su voluntad de observar un recurso del servidor. A su vez, el servidor mantiene una lista de clientes que están observando cada recurso.

Cuando el valor de un recurso cambia, el servidor envía un mensaje a cada cliente suscrito con el nuevo valor del recurso. Además, el servidor incluye un valor incremental en cada mensaje para permitir a los clientes ordenar los datos recibidos. Para cancelar la suscripción a un recurso del servidor, un cliente puede enviar un mensaje reset, o bien hacer una nueva solicitud para el mismo recurso sin el parámetro Observe.

1.3. Tipo y formato de los mensajes

Para el intercambio de datos entre servidor y cliente, el protocolo CoAP define cuatro tipos de mensajes:

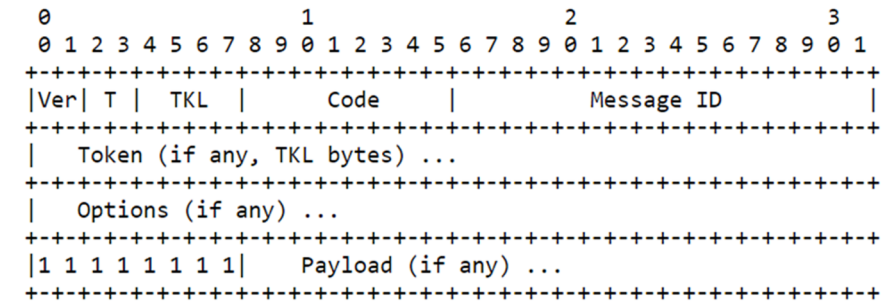
- **Confirmable**: Los mensajes de tipo confirmable son aquellos que requieren de confirmación de recepción por parte del receptor.

- **No-confirmable**: Los mensajes de tipo no-confirmable son aquellos que no requieren de confirmación por parte del receptor.

- **Confirmación (*acknowledgement*)**: Los mensajes de confirmación son usados para notificar la correcta recepción de un mensaje.

- **Reset**: Los mensajes reset se usan para indicar que el contenido de un mensaje recibido no se ha podido procesar de forma correcta.

Figura 1. Cabecera CoAP seguida del Token y las opciones.



Fuente: extraído del RFC7252

En el protocolo CoAP se define una cabecera de 4 bytes que incluye los siguientes campos:

- **Versión (Ver, 2 bits)**: Indica la versión del protocolo.

- **Tipo de mensaje (T, 2 bits)**: Indica el tipo de mensaje (confirmable, no-confirmable, confirmación, reset).

- **Token Length (TKL, 4 bits)**: Indica la longitud del campo opcional Token, que sirve para mapear una solicitud con una respuesta posterior.

- **Código (8 bits)**: Código de solicitud o de respuesta. Por ejemplo, 2.05 Content que se codifica con 1 Byte.

- **Identificador del mensaje (Message ID, 16 bits)**: Identifica un mensaje. Sirve para mapear el mensaje con su ACK. A diferencia del Token, solo mapea el mensaje con su confirmación (ACK).

- **Token (opcional)**: Es un número de secuencia generado por el origen de la petición que sirve para mapear una petición con una respuesta asíncrona. La respuesta usa el mismo Token para que puedan ser reconocidas.

- **Opciones (opcional)**: Es una lista de opciones que permiten aumentar el significado de la petición. Por ejemplo, se puede especificar el tiempo de

expiración del resultado de una petición que se guarda en la cache (Max-Age). Otra opción como If-Match nos permite restringir la operación PUT, etc.

- **Marca de final de cabecera**: La marca 0xFF (11111111b) se usa para identificar el final de la cabecera. Seguido a esta marca el paquete puede contener datos de aplicación.

- **Payload (opcional)**: Los datos de aplicación que forman el contenido del paquete.

Es aconsejable que los mensajes CoAP que se envían entre servidor y cliente quepan en un único datagrama IP para evitar su fragmentación en origen. A su vez, la longitud de un datagrama IP depende de la tecnología de transporte. Por ejemplo, en el caso de una red Ethernet, el tamaño estándar de un datagrama IP son 1.280 bytes. Sin embargo, en el caso de una red 6LoWPAN, el tamaño estándar de un datagrama IP son 127 bytes.

1.4. Semántica de los mensajes

Los métodos del protocolo CoAP heredan la semántica de códigos del protocolo HTTP, los cuales son de tipo textual (no binario) para que resulte fácil de interpretar por humanos.

- **0.00 EMPTY**: Se utiliza para mandar un mensaje vacío. Básicamente, se trata de una cabecera de 4 bytes. Es útil para responder a peticiones confirmables cuando aún no se ha podido obtener la respuesta.

- **0.01 GET**: Petición GET con el mismo significado que en HTTP. La petición GET se usa para pedir datos al servidor.

- **0.02 POST**: Permite ejecutar una acción en el servidor. Opcionalmente, el resultado puede devolver datos al cliente.

- **0.03 PUT**: Permite enviar datos al servidor.

- **0.04 DELETE**: Tiene el mismo significado que DELETE en HTTP. Se utiliza para eliminar un recurso del servidor.

Las respuestas a los mensajes del protocolo CoAP también heredan la semántica de los códigos de estado del protocolo HTTP. Los mensajes que empiezan por 1 indican información, los mensajes que empiezan por 2 indican éxito, los mensajes que empiezan por 3 indican redirección, los que empiezan por 4 indican errores del cliente y, finalmente, los que empiezan por 5 indican errores del servidor. Por ejemplo, en el protocolo CoAP encontramos los siguientes tipos de respuestas:

- **2.01 CREATED**
- **2.02 DELETED**
- **2.03 VALID**
- **2.04 CHANGED**
- **4.00 BAD REQUEST**
- **4.01 UNAUTHORIZED**
- **4.02 BAD OPTION**
- **4.03 FORBIDDEN**
- **4.04 NOT FOUND**
- **5.00 INTERNAL SERVER ERROR**
- **5.01 NOT IMPLEMENTED**
- **5.02 BAD GATEWAY**

A diferencia del protocolo HTTP, donde los mensajes son siempre asíncronos, es decir, petición y respuesta por separado como mensajes HTTP, en el protocolo CoAP los mensajes de respuesta se pueden enviar por separado, igual que en HTTP, o bien incrustados en un mensaje de confirmación (ACK) en caso de que el mensaje de petición sea confirmable. Esto último permite reducir los requerimientos de ancho de banda y de consumo energético de los dispositivos.

1.5. Identificación de recursos

Los recursos en el protocolo CoAP, de la misma forma que en HTTP, se identifican con un identificador de recursos uniforme (en inglés, URI, *uniform resource identifier*). Una URI es una cadena de caracteres que identifica de forma unívoca un recurso y está formada por los siguientes elementos:

- **Esquema**: Es un nombre que identifica la especificación o protocolo utilizado por el recurso (por ejemplo, urn:, tag:, cid:, http:, mailto:, ftp:, coap, etc.).

- **Autoridad**: Es el nombre o URL del host o autoridad jerárquica donde se encuentra el recurso (por ejemplo, //www.uoc.edu).

- **Ruta**: Es la información jerárquica que identifica al recurso particular dentro del ámbito del esquema URI (por ejemplo, /sensors/light).

- **Consulta**: Es una información con estructura no jerárquica (normalmente parejas "clave=valor") que permiten parametrizar al recurso en el ámbito del esquema URI. El inicio de este elemento se indica con el carácter '?'.

En el protocolo CoAP el esquema URI se define siguiendo la siguiente expresión:

coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]

1.6. Descubrimiento de recursos

El protocolo CoAP ofrece un mecanismo para que un cliente pueda descubrir los servicios que implementa un servidor. El mecanismo se basa en que el cliente descubre las URI que referencian a todos los recursos disponibles en el espacio de nombres del servidor.

Todo servidor CoAP que quiera facilitar el descubrimiento de servicios ofrece una URI "/.well-known/core" a la que cualquier cliente puede acceder. Esta URI es accesible desde el puerto UDP por defecto asignado al protocolo CoAP; según lo define IANA (Internet Assigned Numbers Authority) es el puerto 5683 (5684 para DTLS).

Cuando un cliente accede mediante GET a dicha URI, la respuesta del servidor es un mensaje con la opción Content-Format "application/link-format" definida en el RFC 6690 de la IETF. Este formato de respuesta es una lista de URI que identifican los servicios que se ejecutan en el servidor.

Como el protocolo CoAP está pensado para ser usado en sistemas empotrados o en tecnologías de comunicación de baja capacidad, con MTU (Maximum Transmission Unit) pequeñas o limitaciones en el número de bytes que se puede transmitir, todos los tipos y opciones no se transmiten en formato textual, sino codificados. Por ejemplo, los Content-Format se codifican de la siguiente manera:

Media type/Content type	ID usado en CoAP
text/plain (charset=utf-8)	0
application/link-format	40
application/xml	41
application/octet-stream	42
application/exi	47
application/json	50

Ejemplo de descubrimiento de servicios

REQ: GET /.well-known/core

RES: 2.05 Content

</sensors/temp>;if="sensor",

</sensors/light>;if="sensor"

En el ejemplo anterior, se observa cómo una petición GET sobre la URI "/.well-known/core" devuelve una respuesta 2.05 Content, indicando que la respuesta tiene contenido. Dicho contenido son las URI de dos servicios que ofrece el servidor. Ambos servicios son accesibles a través de la interfaz if="sensor", y cada uno de ellos representa un recurso concreto de la interfaz "sensors". En este caso, son un servicio que nos devuelve información sobre temperatura y un servicio que nos devuelve información sobre luminosidad.

1.7. Caching y proxy

El protocolo CoAP incorpora un mecanismo de caching por el cual un servidor puede almacenar respuestas ya dadas a otros nodos. El objetivo es reducir el tiempo de respuesta y ancho de banda usado por la red en peticiones futuras equivalentes. Por tanto, se pretende reutilizar una respuesta ya dada para satisfacer un acceso actual. En el protocolo CoAP, la decisión de mantener una respuesta en la caché depende de si el contenido se puede guardar en la caché (*cacheability*) y si su valor es válido (*freshness*).

- ***Cacheability***: Lo determina el código de respuesta a una petición. Es decir, solo algunos de los códigos de respuesta son cacheables. Por ejemplo, el código 2.02 Valid y el código 2.05 Content son cacheables.

- ***Freshness***: Lo indica el parámetro opcional Max-Age, que define el tiempo de vida del elemento guardado en la caché. El valor por defecto es 60 segundos, después de los cuales el valor se considera obsoleto. Un servidor por defecto guardará un recurso en caché a no ser que se lo pidan con una opción Max-Age con valor 0.

El protocolo CoAP también ofrece la posibilidad de designar a un nodo como intermediario (*proxy*). Un nodo proxy puede hacer peticiones a otros en nombre del nodo que le ha delegado esa función. Esta funcionalidad le permite a un nodo CoAP delegar el acceso a un recurso y así posiblemente minimizar el consumo energético. Además, un proxy puede a su vez actuar de caché manteniendo en su memoria información que es relevante para otros.

En el protocolo CoAP se definen dos tipos de nodos proxy, forward-proxy y reverse-proxy, que se describen a continuación:

- **Forward-proxy**: Un nodo puede actuar de forward-proxy, es decir, dada una petición con unos parámetros determinados, el proxy realiza la petición a un tercer nodo en nombre del primero.

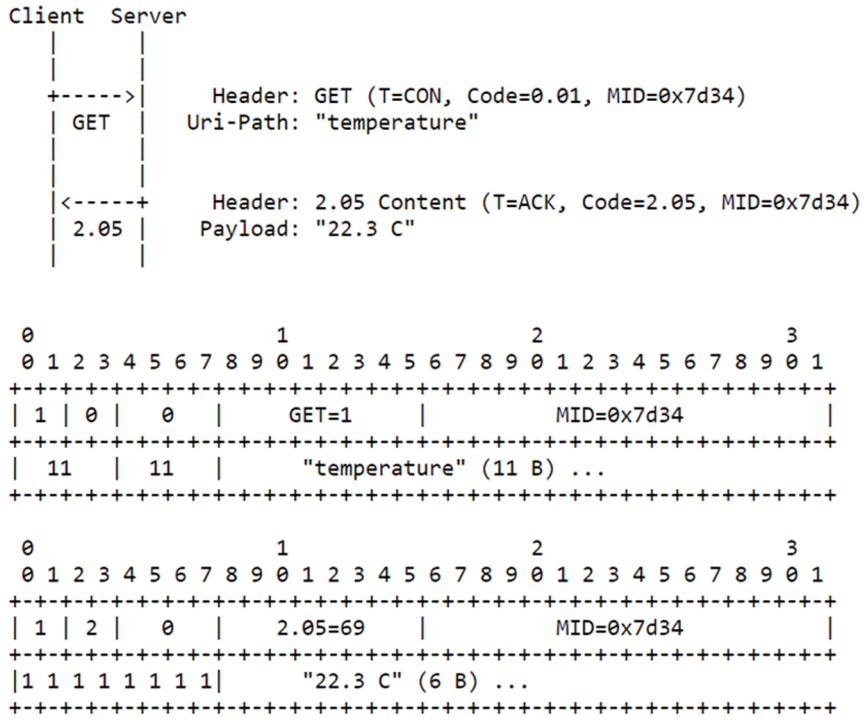
- **Reverse-proxy**: Un nodo también puede actuar como reverse-proxy, con lo que representará a un nodo (o responderá en nombre de ese nodo) ante las peticiones de otros nodos.

Cuando un proxy no implementa una caché, simplemente reenvía la solicitud directamente al destino. Por el contrario, si el nodo implementa una caché (es habitual que sea así), y esta contiene una respuesta actualizada y no caducada de la petición, entonces responde de forma inmediata. Si el proxy no tiene esa respuesta o está caducada, entonces accede al servidor que contiene la información para actualizar su valor.

1.8. Ejemplos de uso

En el primer ejemplo (extraído del RFC 7252), se ilustra una petición GET confirmable para el recurso coap://server/temperature. La respuesta viene embebida (en inglés, *piggybacked*) en el mensaje de confirmación.

Figura 2. Ejemplo presentando una interacción cliente servidor con mensajes confirmables y respuestas embebidas en la confirmación (ACK)

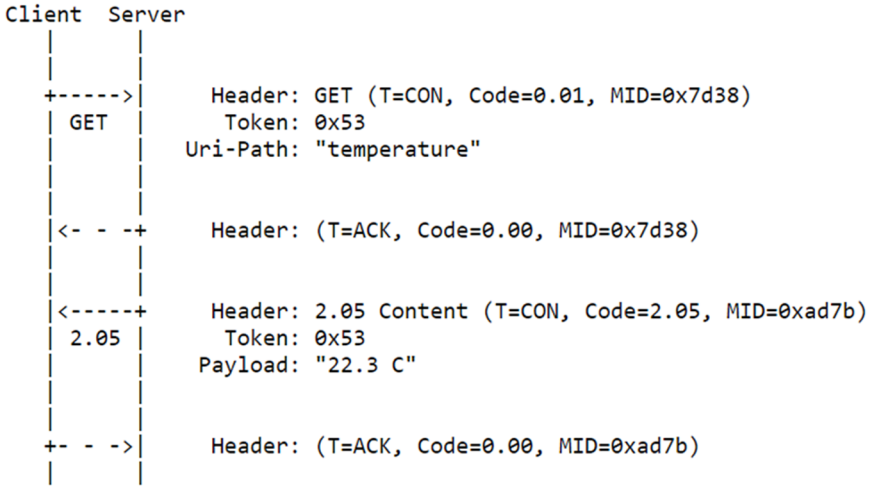


Fuente: extraído del RFC7252

El mensaje de confirmación (ACK) contiene el código de respuesta 2.05 Content, indicando que el mensaje tiene contenido (0x69), el mismo identificador de mensaje (MID) que permite mapear petición con respuesta, el marcador 0xFF que marca el final de la cabecera CoAP y delimita el contenido que lo sigue. En este caso, un valor de temperatura en formato textual ocupando 6 bytes.

En el siguiente ejemplo se detalla el uso de un mensaje confirmable, pero esta vez la respuesta no está embebida en la confirmación (ACK), sino que viene *a posteriori* en un nuevo mensaje confirmable del servidor. Por un lado, se observa que el mensaje de confirmación ACK del servidor devuelve un Code=0.00 Empty, indicando que la respuesta vendrá después en un mensaje aparte. Seguidamente, se observa la petición del servidor con tipo (T=2.05 Content) con el contenido de respuesta. Se puede observar que los Token coinciden para que la respuesta pueda ser identificada por el cliente. Finalmente, vemos la confirmación ACK del cliente con un mensaje de tipo ACK y código 0.00 Empty.

Figura 3. Ejemplo presentando una interacción cliente servidor con mensajes confirmables y respuestas no embebidas.



Fuente: extraído del RFC7252

2. MQTT

2.1. Introducción

MQTT (Message Queue Telemetry Transport) es un protocolo de la capa de aplicación dentro del modelo OSI (Open Systems Interconnection). El protocolo MQTT ha sido diseñado para el transporte de mensajes en aplicaciones en las que la capacidad de comunicación de los nodos está muy restringida, ya sea por limitaciones en su capacidad de computación (por ejemplo, debido a una baja velocidad de la CPU o un espacio de memoria reducido) o en su capacidad de transmisión de datos (por ejemplo, conectividad intermitente, ancho de banda limitado, velocidad de transmisión baja, etc.).

OSI

El modelo de referencia OSI (Open Systems Interconnection) está compuesto por siete niveles o capas, y en cada nivel se agrupan una serie de funciones o protocolos necesarios para comunicar sistemas. Las capas son: Aplicación, Presentación, Sesión, Transporte, Red, Enlace de datos y Física.

El origen del protocolo MQTT se remonta al año 1999, cuando Andy Stanford, de IBM, y Arlen Nipper, de Cirrus Link, lo diseñaron e implementaron para su aplicación en la monitorización de tuberías de petróleo utilizando nodos sensores alimentados por baterías y con conexión vía satélite. Más adelante, IBM incorporó el protocolo MQTT como parte de Websphere MQ, una suite de mensajería que facilitaba la integración de aplicaciones y datos de negocio en entornos multiplataforma. Finalmente, en el año 2013, a la vista de la aplicabilidad del protocolo MQTT en el ámbito de internet de las cosas, IBM propuso la incorporación de la versión 3.1 del protocolo MQTT a OASIS (Organization for the Advancement of Structured Information Standards) para garantizar su aceptación.

OASIS

OASIS (Organization for the Advancement of Structured Information Standards) es un consorcio internacional sin ánimo de lucro orientado al desarrollo, la convergencia y la adopción de estándares abiertos para la sociedad de la información global.

Gracias a su escalabilidad y estandarización, el protocolo MQTT goza de gran aceptación por parte de la comunidad y se utiliza en diversos ámbitos, como por ejemplo en el servicio de mensajería de Facebook. Además, existen implementaciones libres y gratuitas del broker MQTT, y también librerías de cliente MQTT para diferentes lenguajes de programación (C/C++, Java, Python, Node, etc.), lo cual facilita su adopción por parte de los desarrolladores de aplicaciones.

2.2. Descripción funcional

El protocolo MQTT se implementa sobre el protocolo de transporte TCP/IP, utiliza una arquitectura cliente/servidor y un modelo de comunicación de tipo publicación/suscripción para facilitar el intercambio de mensajes.

La arquitectura cliente/servidor del protocolo MQTT define dos entidades: brokers y clientes. Los brokers o servidores MQTT se encargan de transferir los mensajes entre los clientes MQTT. Por su parte, los clientes MQTT son los dispositivos que se conectan a un broker MQTT para publicar mensajes, o sus-

cribirse a los mensajes que publican otros clientes. Es decir, los clientes no se comunican directamente entre ellos, sino que se conectan a un broker a través del cual se realiza el intercambio de mensajes. El protocolo MQTT ofrece una capacidad de hasta diez mil clientes conectados a un broker.

El modelo de publicación/suscripción del protocolo MQTT funciona de la forma siguiente. En MQTT los mensajes que envían los clientes se organizan en tópicos. Para conectarse a un broker, los clientes eligen un identificador único, y cuando publican un nuevo mensaje eligen un tópico para clasificar el contenido de la información que van a enviar en el mensaje. Por otro lado, para recibir mensajes, los clientes deben suscribirse a los tópicos en los que estén interesados. Cuando el broker recibe un nuevo mensaje publicado por un cliente, el broker envía el mensaje a todos aquellos clientes que se hayan suscrito al tópico de dicho mensaje. Por tanto, la comunicación puede ser de uno-a-uno, o de uno-a-muchos.

Los tópicos de los mensajes MQTT se definen como una cadena de caracteres organizada de manera jerárquica en forma de árbol. Para separar los diferentes niveles de la jerarquía se utiliza el carácter '/'. A continuación, se muestran varios ejemplos de tópicos de mensajes que pueden ser publicados por un nodo cliente ubicado en uno de los apartamentos de un edificio,

<edificio>/<planta>/<apartamento>/temperatura

<edificio>/<planta>/<apartamento>/consumoElectrico

<edificio>/<planta>/<apartamento>/estadoAlarmas

y un ejemplo de tópico de mensajes al cual se puede suscribir dicho cliente para recibir comandos de control,

<edificio>/<planta>/<apartamento>/termostato/ajusteTemp

Un cliente puede suscribirse a un tópico específico, y también puede suscribirse simultáneamente a diversos tópicos mediante el uso de los caracteres '+' y '#' dentro de la jerarquía. El carácter '+' se puede colocar en cualquier nivel de la jerarquía de tópicos, excepto en el último nivel. El carácter '#' solo se puede colocar en el último nivel de la jerarquía. A continuación, se muestran algunos ejemplos de selección de tópicos utilizando los caracteres '+' y '#'.

hilton/planta1/+/consumoElectrico

Para suscribirse al tópico de consumo eléctrico de todos los apartamentos de la planta 1 del edificio Hilton.

hilton/+/+/temperatura

Para suscribirse al tópico de temperatura de todos los apartamentos de todas las plantas del edificio Hilton.

hilton/planta24/puerta1/#

Para suscribirse al tópico de temperatura, consumo eléctrico y estado alarmas del apartamento ubicado en la planta 24, puerta 1, del edificio Hilton.

Es importante destacar que los caracteres '+' y '#' no pueden utilizarse para seleccionar un tópico cuando un cliente publica un mensaje.

Un cliente puede suscribirse a un tópico de forma «persistente» o «no persistente». Cuando la suscripción es persistente y el cliente está conectado, entonces el broker envía inmediatamente los mensajes al cliente. Si el cliente no está conectado, entonces el broker almacena los mensajes hasta que el cliente se vuelva a conectar al broker. Por el contrario, cuando la suscripción no es persistente, entonces el tiempo de suscripción es el mismo que el tiempo en que el cliente permanece conectado al broker, es decir, el broker no guarda los mensajes si el cliente se desconecta.

En lo que concierne la seguridad de las comunicaciones, el protocolo MQTT ofrece mecanismos de autenticación y de encriptación. En cuanto a la autenticación, el protocolo MQTT soporta la autenticación de los clientes mediante el uso de un usuario y contraseña durante el proceso de conexión. Por su parte, el broker MQTT puede utilizar una base de datos de usuarios interna o bien utilizar otros sistemas de autenticación como LDAP (Lightweight Directory Access Protocol) o también OAuth, definidos en los RFC 4511 y RFC 6749, respectivamente. En cuanto a la encriptación, el protocolo MQTT puede utilizar cifrado sobre SSL (Secure Sockets Layer), de modo que las comunicaciones entre el cliente y el broker viajen de manera protegida. El puerto TCP/IP asignado por defecto al protocolo MQTT es el puerto 1883, y para usar MQTT sobre SSL se utiliza el puerto TCP/IP 8883.

SSL

SSL (Secure Sockets Layer) es un protocolo que proporciona comunicaciones seguras entre los extremos de una red, permitiendo la autenticación y privacidad de la información mediante el uso de la criptografía. El protocolo SSL fue desarrollado originalmente por Netscape. La primera especificación de SSL 3.0 fue publicada en el año 1996 por la IETF en el RFC 6101.

Finalmente, el protocolo MQTT incorpora un mecanismo denominado «última voluntad y testamento» que funciona de la forma siguiente. Cuando un cliente se conecta a un broker MQTT, el cliente puede definir un tópico y un mensaje de aplicación para que este se publique en caso de que el cliente se desconecte del broker de manera inesperada. De este modo, el broker MQTT notifica a los clientes suscritos a un determinado tópico que un cliente se ha desconectado, de modo que puedan tomar las acciones apropiadas.

2.3. Tipos y formato de los paquetes MQTT

En el protocolo MQTT se intercambian una serie de paquetes entre el broker y los clientes MQTT para implementar las funcionalidades del modelo de publicación/suscripción descritas en el apartado anterior.

Un paquete MQTT consta de tres partes:

1) Una **cabecera fija**, que está presente en todos los paquetes;

2) Una **cabecera variable**, que está presente en algunos paquetes; y

3) Una **payload**, que está presente en algunos paquetes.

La **cabecera fija** tiene una longitud de 2 o más bytes e incorpora los campos siguientes:

- **Packet type (4 bits)**: Indica el tipo de paquete que se está enviando. Los tipos de paquetes MQTT son los siguientes: CONNECT, DISCONNECT, CONNACK, PINGREQ, PINGRESP, PUBACK, PUBCOMP, PUBLISH, PUBREC, PUBREL, SUBACK, SUBSCRIBE, UNSUBSCRIBE y UNSUBACK. La funcionalidad asociada a cada tipo de paquete se describe más adelante.

- **Flags for packet type (4 bits)**: Indican diversos atributos que dependen del tipo de paquete. Por ejemplo, en el caso de los paquetes de tipo PUBLISH, los flags indican si el mensaje está duplicado (flag DUP en bit 3), la calidad de servicio del mensaje (bits 2 y 1) y, finalmente, si el broker tiene que retener el mensaje de manera persistente (bit 0).

- **Remaining length (1 byte o más)**: Indica el número de bytes que hay en el resto del paquete, incluyendo los datos de la cabecera variable y la payload.

La **cabecera variable** de los paquetes MQTT solo está incluida en algunos tipos de paquetes y se encuentra entre la cabecera fija y la payload. El contenido de la cabecera variable depende del tipo de paquete.

Por ejemplo, en diversos tipos de paquetes, la cabecera variable está formada por un identificador de paquete de 2 bytes. Estos tipos de paquetes son PUBLISH (con QoS mayor que 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE y UNSUBACK. Cada vez que un cliente envía un nuevo paquete de uno de estos tipos, el cliente debe asignarle un nuevo identificador de paquete. Si el cliente reenvía un paquete, entonces debe asignar el mismo identificador en envíos sucesivos del mismo paquete. Este mecanismo se utiliza para controlar la correcta recepción de los paquetes, permite identificar paquetes perdidos o recibidos de manera no consecutiva, tanto a los clientes como al broker, garantizando el funcionamiento del protocolo MQTT.

Algunos paquetes MQTT contienen una payload variable al final del paquete. Los paquetes que contienen payload son del tipo CONNECT, PUBLISH, SUBSCRIBE, SUBACK y UNSUBSCRIBE. En el caso de los paquetes de tipo PUBLISH,

la payload es el mensaje enviado por la aplicación del cliente. En el resto de tipos de paquetes de la lista anterior, el contenido de la payload depende de la funcionalidad asociada a cada tipo.

Por ejemplo, en el caso de los paquetes de tipo CONNECT, la payload contiene uno o más campos de longitud fija, cuya presencia viene determinada por el valor de los flags incluidos en la cabecera variable. Dichos campos, si están presentes, aparecen en el orden siguiente: el identificador del cliente; el tópico del mensaje que se envía; el mensaje generado por la aplicación del cliente; el nombre de usuario y la contraseña, utilizados para la autenticación y autorización del cliente por parte del broker. Todos estos campos de la payload del paquete CONNECT son opcionales, excepto el identificador del cliente, que debe estar siempre presente.

Como se ha descrito en los párrafos anteriores, el protocolo MQTT define un conjunto de paquetes que son intercambiados entre el cliente y el broker para llevar a cabo las comunicaciones. La siguiente tabla enumera los diferentes tipos de paquetes definidos en el protocolo MQTT, y describe brevemente la funcionalidad asociada a cada tipo de paquete. Los niveles de calidad de servicio (QoS, *quality of service*) de MQTT que aparecen en la tabla se describen más adelante.

Tipo	Funcionalidad
CONNECT	Es el primer paquete que debe ser enviado por el cliente al broker para solicitar una conexión. Si el broker recibe un segundo paquete CONNECT del mismo cliente, este último se considera un error y el cliente debe desconectarse.
CONNACK	El broker reconoce (ACK) o acepta la solicitud de conexión del cliente. Es el primer paquete enviado por el broker al cliente en respuesta a un paquete CONNECT. Si el tiempo de recepción del paquete CONNACK excede un tiempo predefinido, el cliente debe cerrar la conexión de red.
PUBLISH	Es un paquete enviado desde un cliente a un broker, o desde un broker a un cliente, para transferir un mensaje de aplicación.
PUBACK	Es un paquete de respuesta de reconocimiento (ACK) a un mensaje recibido con QoS 1. Se envía desde un cliente a un broker, o desde un broker a un cliente.
PUBREC	Es un paquete de respuesta a un mensaje recibido con QoS 2. Es el segundo paquete que se intercambian para asegurar QoS 2. Se envía desde un cliente a un broker, o desde un broker a un cliente.
PUBREL	Es un paquete de respuesta a un paquete de tipo PUBREC. Es el tercer paquete que se intercambian para asegurar QoS 2. Se envía desde un cliente a un broker, o desde un broker a un cliente.
PUBCOMP	Es un paquete de respuesta a un paquete de tipo PUBREL. Es el cuarto y último paquete que se intercambian para asegurar QoS 2. Se envía desde un cliente a un broker, o desde un broker a un cliente.
SUBSCRIBE	El paquete SUBSCRIBE lo envía un cliente a un broker para crear una o más suscripciones a uno o más tópicos. Para cada suscripción, se especifica el nivel máximo de QoS con que el broker puede enviar mensajes de aplicación al cliente.

Tipo	Funcionalidad
SUBACK	Es un paquete de respuesta (enviado por el broker) a un paquete de tipo SUBSCRIBE para confirmar la recepción y procesado del paquete SUBSCRIBE.
UNSUBSCRIBE	El cliente envía el paquete UNSUBSCRIBE para darse de baja de uno o más tópicos.
UNSUBACK	Es un paquete de respuesta (enviado por el broker) a un paquete de tipo UNSUBSCRIBE para confirmar la recepción y procesado del paquete UNSUBSCRIBE.
PINGREQ	El cliente envía un PING al broker cuando el cliente no tiene ningún otro paquete para enviar al broker. Se utiliza para indicar el broker que el cliente está vivo y para comprobar que la conexión permanece activa.
PINGRESP	El broker responde al PING enviado por el cliente para indicarle que el broker está activo.
DISCONNECT	El paquete DISCONNECT lo envía el cliente al broker para indicarle una desconexión del cliente.

2.4. Identificador del cliente

Como se ha descrito anteriormente, el protocolo MQTT define un identificador de cliente que se utiliza para distinguir los clientes conectados a un broker. Dicho identificador de cliente es una cadena de texto de hasta 23 caracteres, que debe ser única, y no puede ser utilizado por ningún otro cliente mientras exista un cliente conectado al broker con ese identificador.

A la hora de generar un identificador único para cada cliente, se pueden utilizar diversas estrategias:

- Utilizar un identificador único vinculado a la ubicación del dispositivo en el mundo físico. Por ejemplo, se puede utilizar el número de bastidor o de matrícula de un coche en el caso de que el dispositivo esté abordo.

- Utilizar un identificador único vinculado al propio hardware del dispositivo. Por ejemplo, se puede utilizar la dirección EUI-48 (48 bits) o EUI-64 (64 bits) grabado en la tarjeta de red del dispositivo.

- Utilizar un identificador único generado de manera aleatoria a la hora de realizar la conexión al broker MQTT. Por ejemplo, se puede utilizar una cadena de caracteres y números generados a partir de una función random.

EUI

Los EUI (Extended Unique Identifier) son utilizados por los protocolos de la capa 2 del modelo OSI para identificar de forma unívoca a cada dispositivo conectado a una red. El EUI-48 es un EUI de 48 bits que se corresponde con la dirección MAC del dispositivo o tarjeta de red, en la que los primeros 24 bits identifican a la organización o fabricante (OUI, Organizationally Unique Identifier), y los últimos 24 bits los asigna el fabricante a cada dispositivo. El EUI-64 se obtiene a partir del EUI-48, añadiéndole un campo de 16 bits reservado para usos futuros. Por el momento, el valor de dicho campo es un valor fijo definido por el IEEE.

Como es lógico, cada estrategia, a la hora de escoger un identificador de cliente, presenta sus ventajas y desventajas según la aplicación que se vaya a implementar. Por ejemplo, la ventaja de utilizar la ubicación física es que resulta fácil de recordar por las personas y resulta transparente en el caso de reemplazar el elemento. Por contra, la ventaja de utilizar un identificador único vinculado al propio hardware es que es automático y reduce posibles problemas de du-

plicidad en el identificador de un nodo. Finalmente, la ventaja de utilizar un identificador único generado de manera aleatoria es que la posibilidad de duplicidad es nula, pero hace que conocer el dispositivo físico resulte imposible.

Para finalizar, es importante remarcar que, en caso de que dos nodos tengan el mismo identificador, el broker MQTT puede actuar de manera errática, por ejemplo, desconectando el cliente que ya estaba conectado. A su vez, este cliente volvería a intentar conectarse al broker MQTT, creando así un bucle infinito de conexiones y desconexiones de los clientes. Esta situación no resulta sencilla de detectar y podría terminar agotando los recursos de los nodos y del propio broker MQTT, por lo que se recomienda ser precavido a la hora de escoger la estrategia para asignar identificadores a los dispositivos.

2.5. Calidad de servicio

El protocolo MQTT define tres niveles de calidad de servicio. Cuando un cliente publica mensajes en un broker, puede definir el nivel de QoS para cada mensaje, lo cual determinará el comportamiento del broker a la hora de entregar cada mensaje publicado por los clientes en un tópico.

El protocolo de entrega de paquetes es simétrico, es decir, en las siguientes descripciones de los niveles de QoS el cliente y el broker pueden tomar el rol de Emisor o Receptor, indistintamente. Además, el protocolo de entrega se refiere únicamente a la entrega de un mensaje de aplicación de un único Emisor a un único Receptor. Cuando el broker está entregando un mensaje de aplicación a más de un cliente, cada cliente se trata de forma independiente.

A continuación, se describen los tres niveles de calidad de servicio definidos en el protocolo MQTT: QoS 0, QoS 1 y QoS 2.

1) QoS 0: Entrega de un mensaje como máximo

El nivel QoS 0 no garantiza ninguna calidad de servicio, ya que el broker MQTT no espera ninguna respuesta por parte del Receptor al cual se intenta entregar el mensaje y tampoco se define ningún mecanismo de retransmisión de los mensajes más allá de los propios definidos por el protocolo de transporte (TCP). Por tanto, un mensaje con QoS 0 se puede perder en caso de que el cliente se desconecte de manera inesperada o si el broker falla. Por contra, el nivel de QoS 0 es el que menos recursos requiere por parte de los clientes y del broker MQTT y, por tanto, es ampliamente utilizado en casos donde la información no resulte crítica.

2) QoS 1: Entrega de un mensaje como mínimo

El nivel QoS 1 garantiza que el broker va a entregar como mínimo un mensaje a cada cliente que se encuentre suscrito a un tópico, pero puede haber duplicados.

El funcionamiento del protocolo de entrega utilizando QoS 1 es el siguiente. Cuando un cliente publica un mensaje, el cliente espera la confirmación por parte del broker. Cuando el broker recibe un mensaje correctamente, lo almacena en una base de datos, envía un paquete PUBACK de respuesta al cliente confirmando la recepción del mensaje y, finalmente, publica el mensaje a todos los clientes que se encuentren suscritos al tópico del mensaje.

En caso de que el cliente no reciba la confirmación PUBACK por parte del broker dentro de un intervalo de tiempo determinado, el cliente vuelve a enviar el mensaje con el flag DUP activado (DUP=1) para indicar que es un mensaje duplicado. A su vez, cuando el broker recibe un mensaje con el flag DUP activado, el broker envía otro paquete PUBACK al cliente y reenvía el mensaje a todos los clientes que se estén suscritos al tópico.

A modo de ejemplo, los paquetes de tipo SUBSCRIBE y UNSUSCRIBE que un cliente envía al broker para suscribirse o darse de baja de un tópico se envían con QoS 1 para garantizar su recepción.

Resumiendo, en el protocolo de entrega con QoS 1, el Emisor debe: (1) enviar un paquete PUBLISH con el identificador del paquete y los flags QoS=1, y DUP=0; (2) esperar la recepción de un paquete PUBACK procedente del Receptor; y (3) si no recibe el paquete PUBACK, entonces debe reenviar el paquete PUBLISH con los flags QoS=1, y DUP=1 (paquete duplicado). Por su parte, el Receptor debe responder con un paquete PUBACK con el identificador del paquete PUBLISH recibido.

3) QoS 2: Entrega de un mensaje exactamente

El nivel QoS 2 garantiza que el broker entrega exactamente un mensaje a cada cliente. En el protocolo de entrega con nivel QoS 2, el Emisor envía el paquete PUBLISH con los flags QoS=2 y DUP=0. Cuando el Receptor recibe el paquete PUBLISH, responde con un paquete PUBREC al Emisor, el cual responde con un paquete PUBREL, y finalmente el Receptor responde con un paquete PUBCOMP. Es decir, una vez enviado del paquete PUBLISH con el mensaje de aplicación se produce un intercambio de tres paquetes entre Emisor y Receptor: (1) PUBLISH, (2) PUBREC, (3) PUBREL y (4) PUBCOMP. Este mecanismo garantiza que no lleguen mensajes de aplicación duplicados al Receptor.

2.6. Ventajas de MQTT

El protocolo MQTT puede utilizarse en cualquier tipo de aplicación de monitorización y control en la que se realice un intercambio de información, por ejemplo, datos, eventos, alarmas, comandos. Entre las ventajas del protocolo MQTT destacan las siguientes:

1) La especificación del protocolo MQTT es abierta y libre de licencias, lo cual facilita su adopción.

2) El modelo de comunicación de tipo publicación/suscripción del protocolo MQTT facilita y agiliza el envío y la recepción de mensajes. Por ejemplo, la principal diferencia con respecto al protocolo HTTP es que un cliente MQTT no tiene que solicitar información al broker, sino que es el broker quien la envía al cliente cuando tiene información nueva. Esta característica es relevante en aquellas aplicaciones en las que es necesario reducir la latencia en la transferencia de datos, eventos y alarmas.

3) Una de las ventajas principales del protocolo MQTT es su bajo nivel de consumo de recursos, tanto de capacidad de computación, memoria, comunicación y consumo energético, lo cual permite integrarlo en todo tipo de sistemas embebidos.

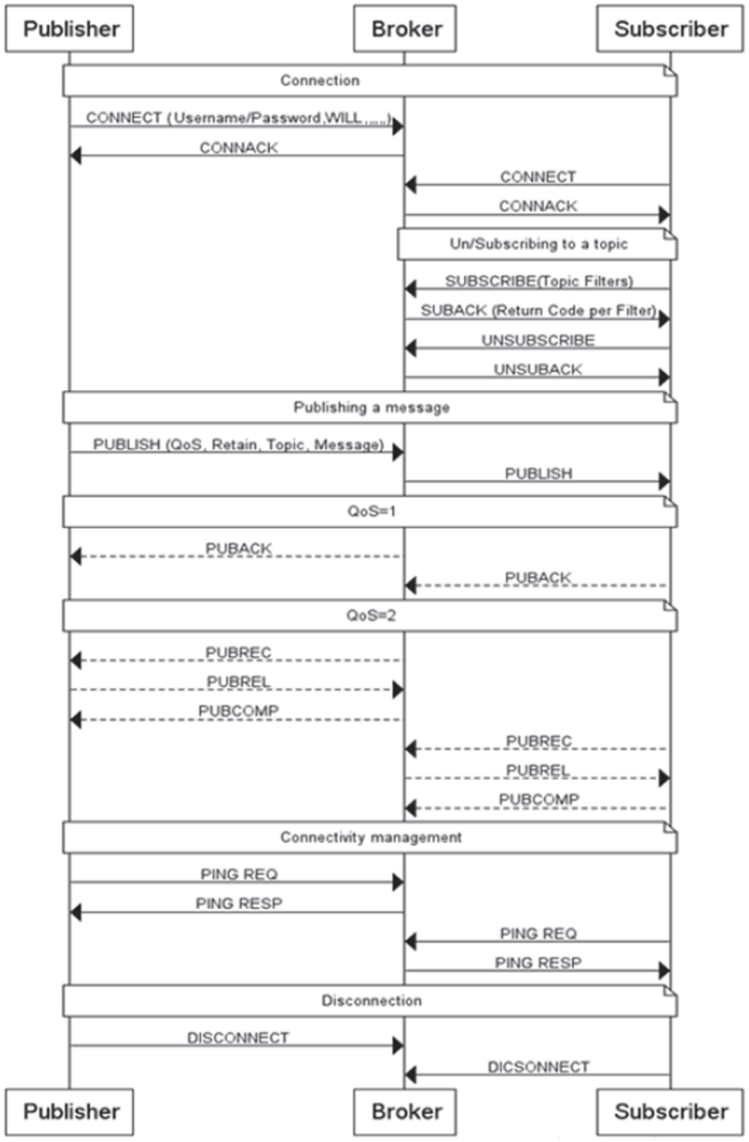
2.7. Ejemplos de uso

A continuación, se muestra un ejemplo de uso del protocolo MQTT donde se describe el intercambio de paquetes entre dos clientes y un broker MQTT usando los tres niveles de calidad de servicio. La figura YYY muestra un diagrama de secuencia del protocolo MQTT que resume la interacción entre un cliente que publica mensajes (publisher), el broker y un cliente que está suscrito al tópico de los mensajes (subscriber).

En primer lugar, como puede observarse en la figura YYY, el cliente publisher y el subscriber envían una solicitud de conexión (paquete CONNECT) al broker, que acepta ambas solicitudes respondiendo con un paquete CONNACK a cada cliente. A continuación, el subscriber envía un paquete SUBSCRIBE al broker para suscribirse a uno o más tópicos con un nivel máximo de QoS, y el broker le responde con un paquete SUBACK que confirma la recepción y procesado del paquete SUBSCRIBE. Hecho esto, el subscriber puede decidir darse de baja de algún tópico enviando un paquete UNSUBSCRIBE al broker.

Cuando el publisher tiene que enviar o publicar un mensaje de aplicación asociado a un cierto tópico, el publisher envía un paquete PUBLISH al broker incluyendo el mensaje, e indicando el nivel de QoS: 0, 1 o 2. En el diagrama de secuencia de la figura YYY se muestra la transferencia de un único mensaje de aplicación desde el cliente publisher con los tres niveles de QoS.

Figura 4. Diagrama de secuencia del protocolo MQTT



Fuente: extraído de Houimli2017

En primer lugar, el mensaje publicado por el publisher es de nivel QoS 0. En el diagrama de secuencia está indicado con el texto "Publishing a message". Como puede observarse, el broker no responde al publisher con ningún paquete de ACK, sino que lo único que hace es comprobar si hay algún cliente suscrito al tópico del mensaje, y se lo envía a dicho cliente (subscriber) en un paquete de tipo PUBLISH.

En segundo lugar, el mensaje publicado por el publisher es de nivel QoS 1. En el diagrama de secuencia está indicado con el texto "QoS=1". En este caso, el broker comprueba si hay algún cliente suscrito al tópico del mensaje, y se lo envía a dicho cliente (subscriber) en un paquete de tipo PUBLISH. Además, para garantizar la QoS=1, el broker responde al publisher con un paquete PUBACK, y el subscriber responde al broker con otro paquete PUBACK.

En tercer lugar, el mensaje publicado por el publisher es de nivel QoS 2. En el diagrama de secuencia está indicado con el texto "QoS=2". En este caso, el broker comprueba si hay algún cliente suscrito al tópico del mensaje, y se lo envía a dicho cliente (subscriber) en un paquete de tipo PUBLISH. A continuación, para garantizar la QoS=2, el broker y el publisher se intercambian la siguiente secuencia de paquetes: (1) PUBREC de broker a pusblisher; (2) PUBREL de pusblisher a broker; y (3) PUBCOMP de broker a pusblisher. Por otro lado, el subscriber y el broker también se intercambian los paquetes PUBREC, PUBREL y PUBCOMP.

En la parte inferior del diagrama de secuencia de la figura YYY puede observarse el envío de paquetes de tipo PING desde los dos clientes al broker, para comunicarle que tanto el subscriber como el publisher permanecen activos. Finalmente, ambos clientes solicitan la desconexión del broker mediante el envío de un paquete de tipo DISCONNECT.

3. OPC/UA

3.1. Introducción

En el año 1994, cinco empresas internacionales del mundo de la automatización industrial (Fisher-Rosemount, Rockwell Software, Opto 22, Intellution, and Intuitive Technology) se unieron para fundar la OPC Foundation. El objetivo de esta organización consistía en desarrollar y promocionar una tecnología para la monitorización y el control de procesos industriales de manera distribuida que resultase independiente de los fabricantes de hardware para facilitar su adopción. En aquel momento, se escogió la tecnología COM/DCOM de Microsoft para la representación y el intercambio de datos entre los diferentes dispositivos, de modo que las siglas OPC correspondían a OLE (Object Linking and Embedding) for Process Control.

COM/DCOM

COM/DCOM es un conjunto de tecnologías propietarias de Microsoft que fueron introducidas a partir del año 1993. Por un lado, COM (Component Object Model) facilita la creación dinámica de objetos y la comunicación entre procesos de manera independiente del lenguaje de programación. Por otro lado, DCOM (Distributed Component Object Model) es una extensión de COM para facilitar el desarrollo de componentes de software en entornos distribuidos.

En el año 1996, la OPC Foundation publicó la especificación de la tecnología OPC, donde se definían tres modelos de comunicación y acceso a los datos para entornos de monitorización y control industrial: DA (Data Access), A&E (Alarms and Events) y HDA (Historical Data Access). Pero a pesar de la adopción de la tecnología OPC en el ámbito industrial, el modelo basado en COM/DCOM requería del uso de ordenadores basados en el sistema operativo Windows, lo que suponía una limitación a la hora de incorporar la tecnología en entornos de tiempo real debido a las limitaciones de capacidad de los dispositivos de la época. Esto obligaba a introducir equipos intermedios, con el consiguiente aumento en los costes de despliegue y mantenimiento, lo cual limitaba su adopción por parte de la industria.

A la vista de esta situación, en el año 2003, la OPC Foundation definió el estándar OPC/XML-DA. Dicho estándar es una evolución abierta de OPC/DA, que utilizaba XML (eXtensible Markup Language) para el modelado de la información, y HTTP/SOAP para el transporte de información. En ese momento, el significado de las siglas OPC se cambió por Open Platform Communications. El objetivo de estos cambios era doble. En primer lugar, desacoplarse de la arquitectura COM/DCOM de Microsoft, consiguiendo así una mayor flexibilidad a la hora de implementar los sistemas OPC en el ámbito industrial. Y en segundo lugar, posicionarse como una organización y estándar abiertos, con el objetivo de atraer el interés y la participación de otros fabricantes y usuarios para consolidarse como tecnología para la monitorización y control en el ámbito industrial.

XML

XML (eXtensible Markup Language) es un metalenguaje que permite definir lenguajes de marcas para su aplicación a la estructuración de información textual de manera independiente a la plataforma. Igual que HTML, XML proviene del lenguaje SGML (Standard Generalized Markup Language) y fue desarrollado por el W3C (World Wide Web Consortium) para su aplicación en el transporte de información en el ámbito de internet.

Si bien el cambio a un modelo abierto resultó un acierto, el uso de XML y HTTP/SOAP consiguió su objetivo solo parcialmente. La solución presentaba problemas de rendimiento debido a las limitaciones computacionales de los

dispositivos de control en tiempo real de la época. Teniendo en cuenta esto, la OPC Foundation empezó a trabajar en la definición del estándar OPC/UA (Open Platform Communications/Unified Architecture), que consistía en una evolución de las diferentes versiones del protocolo OPC clásico (DA, A&E y HDA) para permitir el intercambio de datos de manera óptima e independiente de la plataforma.

Finalmente, el estándar OPC/UA fue presentado en el año 2006 y supone la armonización de las diferentes versiones de OPC existentes hasta la fecha, ofreciendo un único espacio de direccionamiento para la gestión de datos en tiempo real, alarmas y eventos, y datos históricos. Al mismo tiempo, OPC/UA supone el paso de una arquitectura cliente/servidor punto-a-punto a una arquitectura distribuida, donde la información puede estar disponible en diferentes servidores.

La última versión del estándar OPC/UA es la 1.03, publicada en octubre de 2015. Esta versión cuenta con un total de trece documentos independientes donde se definen, entre otros, los modelos de información para la organización y representación de los datos, así como los protocolos de comunicación utilizados para el intercambio de los mismos. Además, gracias al trabajo de estandarización promovido por la OPC Foundation, la tecnología OPC/UA se ha aceptado como estándar internacional bajo la norma IEC 62541.

IEC

La IEC (International Electrotechnical Commission) es una organización internacional fundada en el año 1906 y dedicada a la normalización de las tecnologías eléctricas y electrónicas, así como de las tecnologías relacionadas. Algunas de estas normas se desarrollan conjuntamente con la ISO (International Organization for Standardization), por lo que se conocen como normas ISO/IEC.

A día de hoy, la OPC Foundation se ha convertido en una organización sin ánimo de lucro formada por más de medio millar de empresas y universidades de todo el mundo, que se encarga de mantener, desarrollar y promover el estándar OPC/UA, así como de certificar las implementaciones de los diferentes fabricantes para garantizar la interoperabilidad entre ellos, y asegurar un correcto funcionamiento en los despliegues de la tecnología.

3.2. Revisión de OPC clásico

Como se ha mencionado en la introducción, el objetivo de la tecnología OPC clásica consiste en estandarizar la representación de los datos y el protocolo de intercambio de información para facilitar el desarrollo de aplicaciones en el ámbito de la automatización industrial. Para ello, se han desarrollado tres modelos de OPC complementarios: OPC/DA, OPC/AE y OPC/HDA.

En primer lugar, **OPC/DA** se utiliza para leer, escribir y monitorizar datos en tiempo real. En segundo lugar, **OPC/AE** se utiliza para la recepción de notificaciones y alarmas derivadas de un evento ocurrido en el proceso. Finalmente, **OPC/HDA** se centra en el almacenamiento y recuperación de datos históricos.

Por tanto, en una instalación de automatización industrial, OPC/DA se utiliza para transferir datos en tiempo real entre un servidor, típicamente un PLC (Programmable LogicController), y un cliente, típicamente sistemas HMI o SCADA, para su representación gráfica. A su vez, los sistemas HMI y SCADA incorporan un servidor OPC/AE y/o OPC/HDA que permite a otros clientes OPC (típicamente sistemas de gestión de alarmas o sistemas de supervisión) acceder a la información.

A pesar de que la funcionalidad que ofrece cada una de estas versiones de la tecnología OPC es diferente, todas ellas utilizan una arquitectura de comunicación y un modelo de datos común.

Respecto a la **arquitectura de comunicación**, la tecnología OPC clásica se basa en un modelo cliente/servidor, donde el servidor OPC es el proveedor de los datos y el cliente OPC es el encargado de leer los datos. Así pues, el cliente realiza peticiones al servidor y este responde con los datos utilizando los mecanismos de *polling* o *exception* según convenga.

El mecanismo de *polling* (del inglés, sondeo) se puede realizar de manera síncrona o asíncrona, tal como se describe a continuación:

- **Sondeo síncrono**: El cliente realiza una petición de lectura al servidor y espera que este envíe una respuesta. En caso de que el servidor no pueda responder, el cliente espera un tiempo máximo (*timeout*).

- **Sondeo asíncrono**: El cliente realiza una petición de lectura al servidor e indica un tiempo de refresco o actualización de los datos. El servidor confirma la recepción de la petición y envía de manera periódica el valor de los datos al cliente, el cual confirma la recepción.

En el mecanismo de *exception*, el cliente indica al servidor que desea suscribirse a un parámetro. Por su parte, el servidor responde con el valor actual de dicho parámetro. A partir de ese momento, el servidor enviará información al cliente

PLC

Un PLC (Controlador Lógico Programable, del inglés Programmable LogicController) es un dispositivo de computación utilizado en el ámbito de la automatización industrial para el sensado y control de procesos. A diferencia de los ordenadores convencionales, los PLC ejecutan código en tiempo real estricto y disponen de entradas y salidas analógicas y digitales para realizar la conexión con sensores y actuadores. Además, están diseñados para ser robustos en entornos industriales, donde las condiciones de funcionamiento (rangos de temperatura, vibraciones, etc.) son más extremas.

HMI y SCADA

En el ámbito industrial HMI (Interfaz Hombre-Máquina, del inglés Human-Machine Interface) y SCADA (Supervisión, Control y Adquisición de datos, del inglés Control And Data Acquisition) son conceptos que van ligados. Por un lado, un sistema HMI es un dispositivo formado por un ordenador y una pantalla diseñados para funcionar en entornos industriales donde las condiciones de trabajo son extremas. Por otro lado, un software SCADA es un programa utilizado para el diseño de interfaces de usuario para realizar la adquisición de datos y el control de un proceso industrial. Así pues, los software SCADA se suelen ejecutar en un sistema HMI y se conectan a los PLC mediante un protocolo de comunicaciones para realizar la adquisición de datos y el control a tiempo real del proceso.

solo si ha habido algún cambio en el valor físico o en el factor de calidad del parámetro a monitorizar. Al recibir las actualizaciones, el cliente también confirma la correcta recepción de la información al servidor.

Factor de calidad

El factor de calidad de una medida dentro de la tecnología OPC permite al servidor dar información contextual de la medida física que envía al cliente. El factor de calidad puede tener tres estados (erróneo, incierto, correcto) y en cada estado se definen algunos subestados. Por ejemplo, en el caso del estado erróneo, existen los subestados siguientes: error de configuración, dispositivo no conectado, dispositivo fuera de servicio, fallo de dispositivo, fallo del sensor y fallo de comunicación, entre otros.

Por tanto, un servidor OPC es un dispositivo a nivel de planta que se encarga de recoger los datos de un proceso industrial y ofrecerlos de manera estructurada a través de la red. Por su parte, el cliente OPC es cualquier sistema conectado a la red y que ejecuta una aplicación que implemente el protocolo OPC para leer o escribir dichos datos desde/a un servidor OPC. Es importante tener en cuenta que el mecanismo de *polling* es intensivo en recursos de CPU y de red por parte del servidor, de modo que es aconsejable utilizar el mecanismo de *exception*, a menos que se trate de un parámetro cuyos valores deban ser monitorizados o controlados en tiempo real.

Respecto al **modelo de datos** utilizado para la representación de la información, la tecnología OPC clásica se basa en la tecnología COM/DCOM de Microsoft. En concreto, OPC define un espacio de direccionamiento para la representación de los datos en el lado del servidor. Este espacio de direccionamiento puede ser plano o jerárquico y está organizado en grupos e ítems.

Los **grupos** son los elementos de nivel superior en la jerarquía, se utilizan para contener ítems que, a su vez, contienen propiedades tales como el nombre, el modo de actualización, el porcentaje de cambio, el tiempo de actualización y el método de lectura y escritura.

Los **ítems** forman parte de uno o más grupos y heredan sus propiedades. A su vez, cada ítem incluye atributos, que pueden ser obligatorios u opcionales. Entre los atributos obligatorios destacan el valor del parámetro físico, el tiempo en el que se ha adquirido y el factor de calidad. Como atributos opcionales, destacan el tipo de dato y las unidades de ingeniería (máximo y mínimo) correspondientes a la muestra.

Gracias a esta arquitectura de comunicación y al modelo de datos descrito, la tecnología OPC clásica permite desacoplar los diferentes dispositivos que forman parte de la supervisión y el control de un proceso industrial, permitiendo así la integración de soluciones de diferentes fabricantes. Sin embargo, como se ha mencionado anteriormente, el uso de la tecnología OPC clásica presenta dos problemas fundamentales que limitan su despliegue. En primer lugar, la tecnología COM/DCOM de Microsoft está obsoleta debido a la falta de soporte, por lo que no es recomendable su uso en nuevas instalaciones. En segundo lugar, la sintaxis de los grupos y los ítems utilizada para el modelado de los datos es muy limitada, y la información depende de cada fabricante, por lo que la gestión de la información disponible resulta compleja.

Estos dos problemas de la tecnología OPC clásica son los que impulsaron el desarrollo de la tecnología OPC/XML-DA en primera instancia. Pero a pesar de que OPC/XML-DA resolvía ambas limitaciones, el rendimiento de la solución

no era el adecuado debido a las prestaciones reducidas de los dispositivos de la época. Además, la propuesta OPC/XML-DA no solucionaba los problemas de integración de las diferentes versiones de la tecnología OPC clásica. De este modo, se inició el desarrollo de la tecnología OPC/UA tal como se describe a continuación.

3.3. Revisión de OPC/UA

Tal como se ha introducido anteriormente, OPC/UA integra en un único estándar las funciones de acceso a datos en tiempo real (DA), la gestión de alarmas y eventos (AE), y el acceso a datos históricos (HDA), con el objetivo de simplificar la integración de la tecnología y ofrecer más funcionalidades a sus usuarios.

Para lograr este objetivo, OPC/UA plantea una evolución en el modelado de datos para describir la información y en los mecanismos de comunicación entre sistemas para llevar a cabo el intercambio de datos.

En el caso de OPC clásico, el modelado de datos se limitaba a proporcionar una etiqueta identificativa, el valor de la medida y las unidades del valor medido. Sin embargo, uno de los fundamentos de OPC/UA es el modelado de datos, que permite ofrecer información de contexto a los datos de la propia medida. Por ejemplo, se puede añadir información sobre los dispositivos de medida (fabricante, modelo, etc.) para ofrecer una visión más detallada sobre los datos del proceso y de los sistemas implicados. Así pues, para llevar a cabo el modelado de datos, el estándar OPC/UA se basa en los principios de ingeniería del software que se describen a continuación:

- **Orientación a objetos**: Los atributos y métodos del mundo físico se encapsulan en objetos digitales que los representan.

- **Información tipada**: Los tipos de datos que se utilizan para almacenar y representar la información están bien definidos y se pueden acceder de manera ordenada.

- **Jerarquía de objetos**: Las relaciones entre objetos se definen de manera jerárquica a través de referencias, de manera que es posible acceder a ellos de manera estructurada.

- **Basado en el servidor**: El modelado de la información se realiza en el lado del servidor, de modo que los clientes lo pueden obtener al conectarse para poder realizar las consultas.

Teniendo en cuenta estos principios, a continuación se describe el modelado, la codificación, el transporte y la seguridad de la información en OPC/UA.

3.4. Modelado de la información

Como hemos visto anteriormente, el modelado de la información en OPC/DA se basaba en un *address space*(del inglés, espacio de direcciones) formado por carpetas y variables que permiten organizar la información de manera jerárquica para facilitar el acceso remoto. Si bien este modelo resulta funcional, presenta una limitación de escalabilidad a medida que los diferentes fabricantes de dispositivos añaden nuevos campos de datos.

Para superar esta limitación, en OPC/UA se utiliza un modelado de los datos basado en objetos siguiendo los principios de la ingeniería del software. Este modelo tiene por finalidad disponer de un mecanismo que proporcione flexibilidad a la hora de organizar el espacio de direcciones, permitiendo separar el diseño de la arquitectura de su implementación en el servidor, facilitando así la escalabilidad del sistema y la accesibilidad de la información.

Así pues, para realizar el modelado de la información, OPC/UA define un espacio de direcciones donde residen los objetos, que pueden ser del tipo nodo (del inglés, *node*) o referencia (del inglés, *reference*), tal como se describe a continuación.

Por un lado, los objetos del tipo nodo contienen diversos atributos que dependen de la clase de nodo (del inglés, NodeClass) a la que pertenecen. Además, para cada clase de nodo existe un conjunto de atributos que son comunes, tal como se muestra a continuación:

- **NodeId (NodeId)**: Identifica de manera única un nodo dentro de un servidor para poder intercambiar datos o ejecutar funciones a través de los servicios de comunicación.

- **NodeClass (NodeClass)**: Identifica la clase de nodo a la que pertenece el nodo.

- **BrowseName (QualifiedName)**: Identifica el nombre del nodo de manera textual.

- **DisplayName (LocalizedText)**: Contiene el nombre del nodo que se mostrará en la interfaz de usuario de un cliente.

- **Description (LocalizedText)**: Contiene la descripción del nodo que se mostrará en el interfaz de usuario de un cliente.

- **WriteMask (UInt32)**: Es opcional e indica los atributos que son susceptibles de ser modificados por parte de un cliente.

- **UserWriteMask (UInt32)**: Es opcional e indica los atributos que pueden ser modificados por parte del cliente conectado en función de sus permisos.

Es importante destacar que, para cada atributo de un nodo, se indica entre paréntesis la clase de nodo a la que pertenece. Los atributos pueden ser primitivos o complejos. En el caso de atributos primitivos, tenemos los datos numéricos, entre los que existen, por ejemplo, el tipo UInt32. En el caso de atributos complejos, podemos ver como estos a su vez son nodos dentro del espacio de direcciones. Por tanto, cada atributo complejo es a su vez un nodo que pertenece a una clase de nodo. Por ejemplo, en el caso del NodeClass podemos ver cómo es un atributo de un objeto y pertenece al tipo NodeClass, que es un tipo de nodo en sí mismo.

En el caso de los nodos que pertenecen a una clase de nodo determinado, además de los atributos comunes mostrados en la lista anterior, estos también pueden contener variables y métodos. Así pues, los objetos de la clase variable (del inglés, variable) representan un espacio de memoria donde pueden almacenar un valor/estado y su tamaño dependerá del tipo de variable a almacenar. Los clientes pueden acceder a los valores de las variables mediante los servicios de lectura, escritura, consulta y suscripción. Por su parte, los objetos de la clase método (del inglés, *method*) representan una función ejecutable, es decir, un elemento que puede ser invocado de manera remota por parte de un cliente y, una vez ejecutado, devuelve un valor. Cada método define los argumentos de entrada que el cliente tiene que ofrecer al realizar la invocación y los resultados que el cliente va a obtener una vez completada su ejecución por parte del servidor.

Por su parte, los objetos del tipo referencia indican las relaciones que se establecen entre los diferentes nodos dentro del espacio de direcciones para permitir navegar entre ellos. Los objetos del tipo referencia pertenecen a la clase de nodo *ReferenceType* y contienen los siguientes atributos:

- **IsAbstract (Boolean)**: Indica si la ReferenceType se puede utilizar como referencia o si únicamente se puede utilizar.

- **Symmetric (Boolean)**: Indica si la referencia es simétrica, es decir, si el significado de la referencia es válido para ambas direcciones.

- **InverseName (LocalizedText)**: Es opcional e indica un nombre para la referencia inversa. Es solo válido para referencias no simétricas siempre que esta no sea abstracta.

Es importante destacar que las referencias solo se pueden utilizar para acceder a otro nodo y, por tanto, su contenido no es directamente accesible por parte de los clientes. Además. también es importante destacar que el nodo objetivo de la referencia puede estar en el propio espacio de direcciones del propio ser-

vidor OPC/UA o bien apuntar al espacio de direcciones de un servidor OPC/UA remoto. Finalmente, también es importante destacar que la combinación del nodo origen, el tipo de referencia y el nodo objetivo se utilizan para identificar de forma exclusiva las referencias, por lo que cada nodo puede hacer referencia a otro nodo con el mismo tipo de referencia solo una vez.

3.5. Codificación y transporte de la información

Como hemos visto en el apartado anterior, el modelado de la información permite al servidor OPC/UA definir los datos a los que pueden acceder los clientes de manera remota. Pero para poder llevar a cabo dicho acceso remoto hacen falta dos nuevos elementos:

- Un **mecanismo de codificación** que permita mapear el modelo de datos sobre los paquetes que viajarán desde el servidor al cliente a través de la red de comunicaciones.

- Un **mecanismo de transporte de información** a través de la red de comunicaciones que permita a servidor y cliente intercambiar la información previamente codificada.

El protocolo OPC/UA define dos **mecanismos de codificación** de la información, uno de tipo textual basado en XML y otro de tipo binario, y dos **mecanismos de transporte de la información**, uno basado en SOAP sobre el protocolo HTTP y otro binario que funciona directamente sobre el protocolo TCP/IP. A continuación, se resume el funcionamiento y las ventajas e inconvenientes de los mecanismos de codificación y de transporte de datos definidos en el protocolo OPC/UA.

SOAP

SOAP (del inglés, Simple Object Access Protocol) es un estándar utilizado para la implementación de servicios web que define cómo dos procesos remotos pueden comunicarse mediante el intercambio de mensajes unidireccionales codificados mediante XML. La tecnología SOAP fue creada por un consorcio formado por Microsoft, IBM y otras empresas del sector, a finales de los años noventa, y actualmente se encuentra mantenida por el W3C (World Wide Web Consortium).

El W3C es un organismo internacional que se encarga de garantizar el avance y la interoperabilidad de las tecnologías utilizadas WWW (World Wide Web) a través del desarrollo de recomendaciones y estándares.

3.6. Codificación de datos

Como se ha introducido anteriormente, la codificación de datos es el proceso que permite mapear el modelo de datos sobre los paquetes que viajarán desde el servidor al cliente a través de la red de comunicaciones. La tecnología OPC/UA define dos tipos de mecanismos de codificación, XML y binaria, que se describen a continuación.

La **codificación binaria** está pensada para ofrecer el máximo rendimiento y está especialmente indicada para escenarios donde los recursos son limitados, como por ejemplo en PLC. Esta codificación se basa en un conjunto predefinido de tipos de datos, por ejemplo *strings* (del inglés, cadenas de caracteres), que son mapeados directamente a su formato binario siguiendo unas reglas preestablecidas, lo que facilita la codificación y descodificación. Los tipos de datos más complejos se obtienen a partir de la combinación de tipos de datos básicos, permitiendo crear modelos de datos avanzados.

La **codificación XML** está pensada para facilitar el uso de la información en entornos multiplataforma y para que sea fácilmente interpretable por humanos, de modo que se utiliza principalmente en el ámbito corporativo, en sistemas MES o ERP. Así pues, los tipos de datos básicos definidos por OPC/UA son representados siguiendo las especificaciones del estándar XML, y los tipos de datos compuestos se construyen anidando tipos de datos básicos.

MES y ERP

Los MES (del inglés, Manufacturing Execution System) son sistemas de información corporativos encargados de orquestar y monitorizar los procesos de una planta incluyendo, entre otros, la generación de órdenes de trabajo y de tareas de mantenimiento.

Los ERP (del inglés, Enterprise Resource Planning) son sistemas de información corporativos encargados de gestionar las operaciones de una empresa incluyendo, entre otros, aspectos de producción, logística e inventario.

A continuación, se muestra un ejemplo de codificación binaria y XML para un tipo de datos básico del tipo *string* (en inglés, cadena de caracteres). Es fácil observar que la representación binaria no solo es mucho más compacta, sino que resulta más fácil de interpretar por parte de un ordenador. Por el contrario, la representación XML es portable y resulta más fácil de interpretar por un humano, pero requiere el transporte de más información a través de la red y mayor potencia de cálculo para su procesado.

Figura 5. Ejemplo de codificación binaria y XML en OPC/UA

Codificación binaria de la cadena de caracteres “OPC/UA”						
5	O	P	C	/	U	A
05000000	4F	50	43		55	41

```
<xs:element name="String"
type="sx:string" minOccur="0" />

<String> OPC/UA </String>
```

Fuente: elaboración propia

3.7. Transporte de datos

En cuanto al transporte de datos, OPC/UA define dos mecanismos para el intercambio entre cliente y servidor, uno basado en servicios web utilizando la tecnología SOAP sobre el protocolo de aplicación HTTP, y otro basado en un protocolo binario propio sobre el protocolo TCP de la capa de transporte.

En el primer caso, los datos OPC/UA se mapean sobre un mensaje SOAP y se envían al servidor en el cuerpo de una petición HTTP del tipo POST. Del mismo modo, las respuestas del servidor se mapean sobre un mensaje SOAP y se envían al cliente en el cuerpo de una respuesta HTTP del tipo POST. La

ventaja de utilizar esta alternativa es su seguridad e interoperabilidad. En primer lugar, HTTP puede utilizar TLS para el intercambio de mensajes seguros entre cliente y servidor. En segundo lugar, al tratarse de tecnologías web, no se requieren configuraciones especiales en los dispositivos de seguridad de la red (por ejemplo, *firewalls*).

En el segundo mecanismo de transporte de datos de OPC/UA se utiliza un protocolo binario definido específicamente para el intercambio de datos entre el cliente y el servidor. Este protocolo se encarga de negociar el tamaño de los buffers para el envío y recepción de datos en la aplicación, así como de multiplexar diferentes canales de comunicación entre servidor y cliente a la misma dirección y puerto del servidor. Para ello, el protocolo define una estructura de mensaje dividida en dos partes: la cabecera y el cuerpo del mensaje. La cabecera del mensaje contiene información sobre el tipo y la longitud del cuerpo, mientras que el cuerpo del mensaje contiene los datos OPC/UA codificados y encriptados que son reenviados a las capas superiores, o bien contienen información para la gestión de las comunicaciones.

Para realizar el establecimiento de la conexión entre el cliente y el servidor OPC/UA, el protocolo binario define tres tipos de mensajes, que se describen a continuación:

- **Hello**: El mensaje HELLO lo envía el cliente al servidor para establecer una conexión. En el mensaje HELLO se definen el tamaño de los buffers de envío y recepción de datos, así como el tamaño de los mensajes que se intercambian cliente y servidor.

- **Acknowledge**: El mensaje ACKNOWLEDGE lo envía el servidor al cliente y confirma o modifica los tamaños de los buffers de envío/recepción de datos o del tamaño de los mensajes.

- **Error**: El mensaje de ERROR sirve para que tanto cliente como servidor puedan indicar problemas durante el establecimiento de la conexión o durante el intercambio de información. Por ejemplo, en caso de que ocurra un problema de conexión o de que un mensaje no se pueda procesar correctamente.

La principal ventaja del protocolo binario respecto al protocolo basado en SOAP/HTTP es que el protocolo binario ofrece un rendimiento superior y una carga inferior, de modo que se utiliza principalmente en entornos donde los recursos son limitados o el tiempo de respuesta resulte crítico. Por el contrario, el mecanismo de transporte basado en SOAP/HTTP se utiliza en entornos donde resulte necesario garantizar la seguridad e interoperabilidad de las comunicaciones, a expensas de incurrir en un menor rendimiento.

3.8. Seguridad

Como hemos visto en el apartado anterior, los mecanismos de codificación y transporte de datos de OPC/UA permiten mapear el modelo de datos del servidor y gestionar el intercambio de información con el cliente. La versatilidad de estos mecanismos hace que OPC/UA pueda utilizarse en todos los niveles de la pirámide de automatización, desde el nivel de planta hasta el nivel de gestión de la empresa.

Por ejemplo, en la planta, un PLC puede ejecutar un servidor OPC/UA para proporcionar el acceso a variables de funcionamiento en tiempo real a un sistema HMI/SCADA que incorpore un cliente OPC/UA. Por otro lado, un sistema ERP, en el ámbito corporativo, puede establecer la conexión con diferentes servidores OPC/UA en la planta con el objetivo de recopilar información sobre las horas de funcionamiento de cada máquina y poder planificar las operaciones de mantenimiento en función de las necesidades.

Otro aspecto importante de la tecnología OPC/UA es la gestión de la seguridad de la información y las comunicaciones teniendo en cuenta que estas pueden resultar críticas, no solo para el correcto funcionamiento del proceso industrial bajo control, sino también para la integridad de las personas. En este sentido, la seguridad de la información y las comunicaciones se puede analizar desde un punto de vista organizativo y técnico.

Desde el punto de vista organizativo, la seguridad se refiere a las políticas que define la empresa para garantizar que el personal es consciente de los riesgos y amenazas, y cómo hacer frente a ellas en caso de que ocurran. Desde el punto de vista técnico, la seguridad se refiere a las medidas pasivas o activas que la empresa implementa para minimizar la probabilidad de que ocurra algún suceso y, en caso de que ocurra, reducir su impacto.

Teniendo en cuenta lo anterior, OPC/UA define una arquitectura de seguridad multinivel formada por varias capas, denominadas transporte, comunicación y aplicación, en las que cada capa tiene unas responsabilidades específicas para garantizar la seguridad del conjunto y ofrecer garantías en su despliegue en aplicaciones de automatización industrial.

En primer lugar, para el caso de la comunicación mediante el protocolo binario propio, en cuanto al transporte, se define el uso de sockets orientados a conexión (TCP) para realizar el envío y la recepción de datos entre el cliente y el servidor. Sobre estos sockets se implementan mecanismos para negociar el tamaño de los buffers y de los mensajes a intercambiar entre cliente y servidor, de modo que se puede garantizar la disponibilidad del sistema frente ataques de *buffer overflow* (desbordamiento de buffer) o *denial of service* (denegación de servicio), pues el servidor sabe qué cantidad de información puede esperar del cliente.

Buffer overflow

Buffer overflow (desbordamiento de buffer) es un ataque informático que tiene como objetivo dejar inoperativo un servicio aprovechando errores de programación en la lectura de datos del exterior, logrando sobrescribir espacios de memoria restringidos y generando un error de ejecución.

En segundo lugar, en lo que respecta a la comunicación, se define el concepto de canal seguro, que se encarga de proporcionar confidencialidad e integridad en el intercambio de información entre cliente y servidor a través de la encriptación de los mensajes y del uso de firmas digitales para verificar su contenido. Además, en la comunicación también se ofrecen mecanismos para gestionar la autorización mediante el uso de certificados digitales. Por ejemplo, para la autenticación y autorización OPC/UA permite el uso de usuarios anónimos, usuarios con contraseña o bien usuarios con un certificado, lo que permite definir niveles de seguridad en función de lo críticos que resulten los sistemas.

Finalmente, en cuanto a la aplicación, se define el concepto de sesión, que se encarga de la autenticación y autorización tanto de los usuarios como de las aplicaciones a través del uso de listas de acceso. Esto permite, entre otros, dar acceso a cierta información o ejecutar una función en un servidor OPC/UA según los permisos de los que disponga cada usuario.

Denial of service

Denial of service (denegación de servicio) es un ataque informático que tiene como objetivo saturar los recursos disponibles de un servidor a través del envío de un gran número de peticiones en un corto periodo para conseguir que deje de prestar servicio a los usuarios legítimos.

Autenticación

Autenticación es el proceso de verificación en el que una entidad es realmente quien dice ser, mientras que autorización es el proceso a través del cual se verifica que un usuario autenticado tiene permiso para realizar ciertas acciones. Para la autenticación, se suelen usar mecanismos como nombre de usuario y contraseña o certificados digitales. Para la autorización, se utiliza una base de datos que vincula cada usuario con un nivel de permisos que le permite realizar (o no) algunas acciones.

Bibliografía

Bormann, C.; Castellani, A. P.; Shelby, Z. (2012). «CoAP: An Application Protocol for Billions of Tiny Internet Nodes». *IEEE Internet Computing* (vol. 16, núm. 2, págs. 62-67).

Houimli, M.; Kahloul, L.; Benaoun, S. (2017). «Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things». En: *International Conference on Mathematics and information Technology*. Adrar, Algeria (4-5 de diciembre).

Mahnke, W.; Leitner, S. H.; Damm, M. (2009). *OPC Unified Architecture*. Berlín / Heidelberg: Springer.

OASIS (2015, 10 de diciembre). *MQTT Version 3.1.1 Plus Errata 01* [OASIS Standard Incorporating Approved Errata 01]. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>

Shelby, Z.; Hartke, K.; Bormann, C. (2014, junio). *The Constrained Application Protocol (CoAP)* [RFC 7252]. Internet Engineering Task Force (IETF). <https://tools.ietf.org/html/rfc7252>