DEPARTAMENT DE SISTEMES INFORMÀTICS Y COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Internet de las Cosas

Máster MUCNAP

Seminario 0
Comunicaciones M2M en la IoT y comunicaciones asíncronas con MQTT

Version 1.0

Actualización: 16/11/2020

Profesor: Joan Fons i Cors

Control de versiones

Fecha	Autor	Descripción
16/11/2020	Joan Fons	Versión incial completa del documento

Departamento de Sistemas Informáticos y Computación
Camino de Vera, s/n 46022 Valencia
Joan Fons i Cors, jjfons@dsic.upv.es, PROS/VRAIN UPV

Departamento de Sistemas Informáticos y Computación
Camino de Vera, s/n 46022 Valencia
Joan Fons i Cors, jjfons@dsic.upv.es, PROS/VRAIN UPV

Tabla de Contenido

Estrategias de comunicación en IoT

La IoT propone una plataforma de computación para publicar recursos físicos (*'things'*) en Internet. Sin embargo, no establece o 'limita' el modo en que estos recursos se comunican.

Los dos modelos de comunicación más habituales son la comunicación directa y la comunicación indirecta. Cada una tiene unas ventajas y unos inconvenientes que, en función del problema que queremos resolver, será más adecuado aplicar un modelo u otro. Las siguientes secciones describen estos modelos.

Modelo de comunicación directa o síncrona

En este paradigma, cada recurso posee un identificador (típicamente una URL) con el que se puede acceder al objeto para interactuar con él (consultar su estado, solicitar alguna acción, etc.). Este modelo de computación es directa o síncrona, en el sentido que podemos comunicarnos directamente con el recurso (físico).

Por ejemplo, imaginemos que queremos entrar en un aula a la que se accede a través de una puerta inteligente[1]. Supongamos que esta puerta está habilitada para interactuar vía NFC (ver Fig. 1), y acercamos a la puerta nuestro terminal móvil con capacidades de comunicación NFC (o dispositivo *wearable* personal). En un sistema IoT inteligente, se llevaría a cabo un descubrimiento de ambos dispositivos, una negociación de credenciales, intercambio de identificadores de los recursos (URLs), y finalmente, el terminal móvil podría solicitar a la puerta la apertura. En este modelo de comunicación existe una comunicación directa (síncrona) entre los objetos.

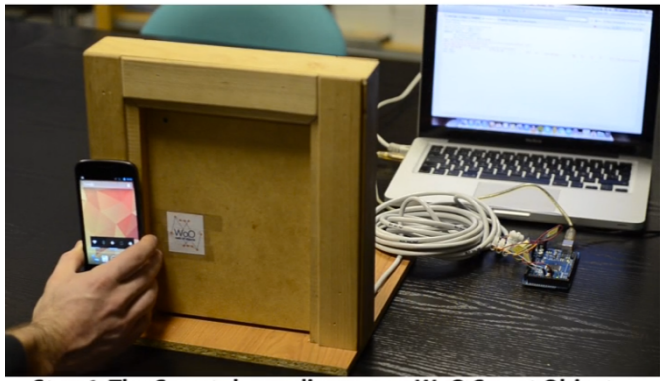

Figura 1: Ejemplo de interacción directa con una puerta inteligente

[1] https://www.youtube.com/watch?v=LF88goTPGhM&index=2&list=PLCIq4vTT0bB7Z3jOeaw2JoFbIbg14IxOk

En este modelo de comunicación directa, es necesario que todos los objetos participantes se conozcan entre ellos (mediante sus identificadores), conozcan su interfaz de uso (que puede estar estandarizada o no) y que todos estén accesibles (aunque no necesariamente deben estar cerca, como en el ejemplo anterior). Dado este escenario, se pueden definir reglas que ejecuten de manera *síncrona* consultas y acciones sobre los diferentes objetos.

Como ventajas de este modelo de comunicación tenemos:
- es posible conocer el resultado de la ejecución de alguna acción sobre un recurso, debido a la llamada síncrona
- podemos saber si una acción ha tenido efecto o ha sido procesada por el recurso
- es más adecuado para desarrollar sistemas en los que se requiere respuestas inmediatas a una solicitud
- no se necesitan intermediarios/middleware para la comunicación

Como desventajas, tenemos:
- es necesario conocer a todos los objetos (sus identificadores) y que éstos estén accesibles en el momento de la petición
- se requiere conocer la interfaz específica de cada objeto para interactuar con él
- en sistemas dinámicos como estos, en los que los objetos pueden aparecer y desaparecer constantemente, este modelo de comunicación impone cierta rigidez

Modelo de comunicación indirecta o asíncrona

Como se ha comentado en la sección anterior, las plataformas IoT publican un conjunto de recursos que atienden peticiones vía Internet, y que pueden participar en procesos de negocio IoT. Visto desde esta perspectiva, cada recurso puede equipararse a un servicio que es invocado, junto a otros, para intercambiar información o actualizar el estado del sistema (en este caso, el de objetos físicos).

Este escenario encaja perfectamente en la problemática de la Integración de Aplicaciones En este dominio, se busca la interoperabilidad de sistemas heterogéneos a través del *intercambio de mensajes* utilizando middleware capaz de gestionar y comunicar a los diferentes participantes.

El uso de mensajes como mecanismo de comunicación introduce una ventaja adicional: estos mensajes actúan como contrato entre clientes y servidores, de manera que cada uno puede estar desarrollado con un enfoque o tecnología diferente, y facilita el 'desacoplamiento tecnológico' entre los participantes. El punto clave es acordar tanto el formato y codificación de datos (texto, XML, JSON,

binario, ANSI X12, EDIFACT, ...), como en el significado del mensaje (esquema del documento).

En este modelo de comunicación se usa el concepto de ESB (*Enterprise Software Bus*), un *bus software para en el envío y recepción de mensajes (messaging backbone)*. Es habitual que en este tipo de enfoques se usen sistemas intermediarios (middleware) que gestionen este bus (ver Fig. 2).

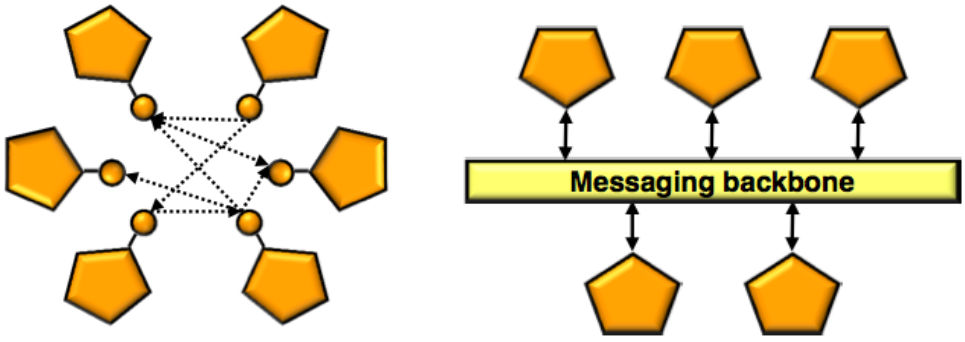

Figura 2: A la izquierda, modelo de comunicación directa.
A la derecha, modelo de comunicación con bus de mensajería

Este modelo de comunicación se configura como un modelo indirecto, ya que la interacción entre los participantes se realiza a través de un middleware responsable del bus de comunicación. En un uso habitual de estas plataformas, los mensajes suelen enviarse de manera asíncrona, de manera que es responsabilidad del middleware atender la petición y procesarla o redirigirla convenientemente. Las respuestas recibidas tras procesar las peticiones suelen realizarse también vía el envío de un mensaje, también asíncrono.

Muchas de las **plataformas** actuales para desarrollar soluciones basadas en la **IoT** (AWS IoT, Eclipse Ditto/Hono, FiWare, SOFIA2, ...) siguen este enfoque de desarrollo. Todos ellos proporcionan APIs para el envío de mensajes a estas plataformas, cuyo formato está previamente definido en los llamados '*gestores de contexto*' (*Context Brokers*). Estas plataformas permiten tanto la suscripción a mensajes, como publicación de nuevos mensajes, además de ofrecer sistemas de persistencia (SQL y no-SQL) para posteriormente permitir tanto la consulta de datos, como un análisis de datos masivo (minería de datos o Big Data).

Otro de los middleware de integración más utilizados para la integración de aplicaciones, son las **colas de mensajería**. Una cola de mensajes (ver Fig. 3) es un contenedor de mensajes, que puede usarse básicamente para dos propósitos:

- *publicación de mensajes*: el creador del mensaje (productor) envía un mensaje a una cola
- *suscripción a una cola*: pueden haber diferentes suscritores (consumidor) a una cola, que recibirán automáticamente (si la cola está configurada como 'push',

modo habitual) los mensajes, o bajo demanda (si la cola se configura como 'pull').

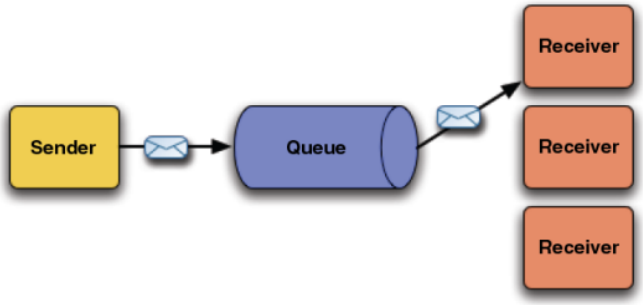

Figura 3: Comunicación vía Colas de Mensajería

Un gestor de colas de mensajes o *broker* de mensajería es una aplicación (middleware) en el que se pueden crear diferentes colas, y son responsables de notificar a los diferentes suscriptores cuando un mensaje es enviado a la cola en la que están suscritos. En función del gestor de colas (existen muchos, como ActiveMQ, MSMQ, OpenJMS, RabbitMQ, zeroMQ, ...), se puede configurar diferente visibilidad y accesibilidad a colas (públicas/privadas, lectura/escritura, roles y permisos, etc.).

Las ventajas de este modelo de comunicación son:

- no es necesario conocer a los diferentes participantes en una comunicación
- facilita la escalabilidad, ya que es sencillo extender las comunicaciones entre productores y consumidores, debido a que éstos se pueden configurar dinámicamente
- no es necesario conocer las interfaces de uso de los diferentes sistemas, ni realizar vínculos tecnológicos con éstos para usar sus interfaces
- no es necesario que los participantes estén en ejecución en el mismo intervalo de tiempo

Las desventajas son:

- no es adecuado para necesidades de comunicación directa o síncrona (no tenemos garantías de que, al dejar un mensaje en una cola, este sea procesado inmediatamente, ni que sea redirigido al suscriptor (si está configurado como '*pull*'), por ejemplo)
- se requiere de infraestructura intermediaria para posibilitar las comunicaciones

Colas de Mensajes para la IoT: MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo para la conectividad "máquina a máquina" (M2M) para la IoT sobre TCP/IP (ver Fig. 4) que proporciona conexiones ordenadas, 'sin pérdida' y bidireccional. Fue diseñado para ofrecer transporte de mensajes extremadamente ligero con un enfoque publicación/suscripción. Es adecuado para conexiones entre conexiones remotas donde se requieren (tienen) pocas capacidades de cómputo, o para minimizar el ancho de banda de la red. Fue diseñado en IBM en 1999, y estandarizado por OASIS (2014), y estándar ISO/IEC JTC1 en Julio de 2016 .

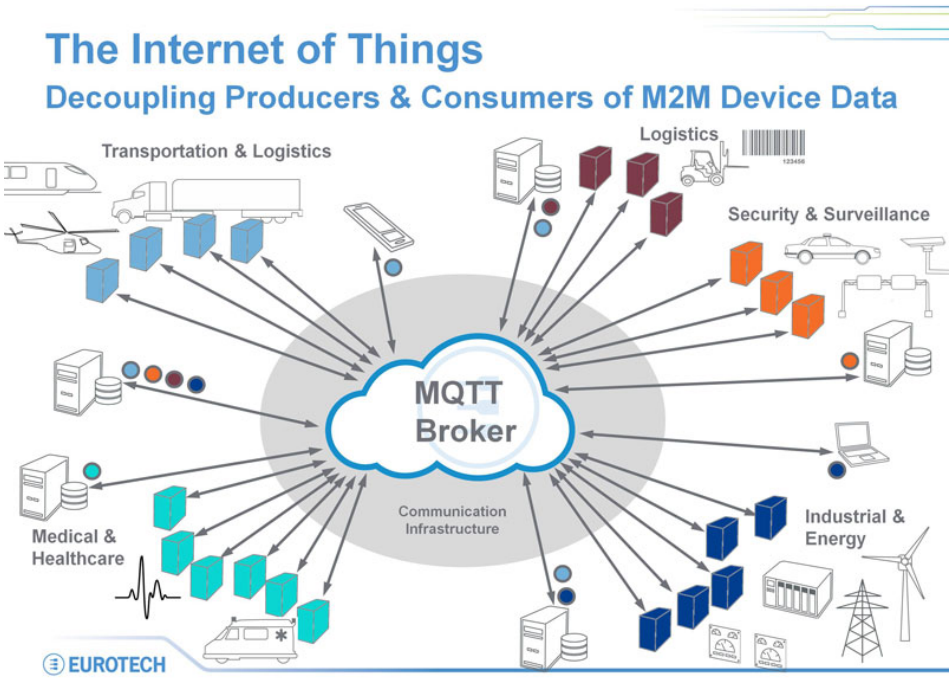

Figura 4: MQTT como mecanismo de interacción en soluciones M2M/IoT

MQTT usa el **puerto** TCP/IP 1883 para comunicaciones no seguras, y el 8883 para comunicaciones sobre SSL (ver Fig. 5). A nivel de seguridad, MQTT permite gestionar perfiles de usuario (usuario/contraseña), así como el cifrado de las comunicaciones mediante SSL. Se pueden añadir niveles adicionales de seguridad directamente por las aplicaciones que publican o se suscriben a las colas de MQTT.

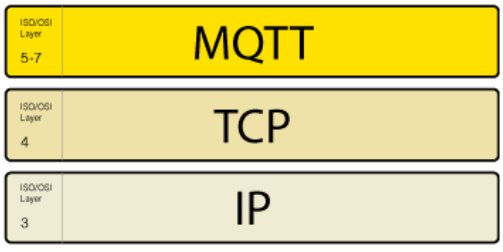

Figura 5: MQTT sobre TCP/IP

Mecanismo de Publicación/suscripción

Se usa el patrón publicación/suscripción (**pub/sub**) como alternativa al modelo clásico cliente/servidor. En este patrón, un cliente se comunica directamente con el servicio que atiende la petición (modelo de comunicación directa). En MQTT, el patrón pub/sub desacopla al cliente que envía un mensaje particular (publicador) de otro cliente (o más de uno), que recibe el mensaje (suscritor). Esto implica que el publicador y suscritor no conocen la existencia uno de otro. Es el broker MQTT el que conoce a ambos, y filtra los mensajes entrantes para distribuirlos correctamente (ver Fig. 6).

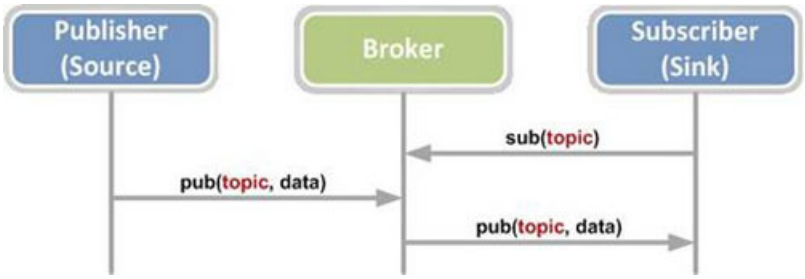

Figura 6: Modelo de comunicación pub/sub en MQTT

Topics

MQTT usa una estrategia de publicación en colas llamadas ***'topics'***. Un topic no es más que un descriptor de una temática de interés en la que se generarán mensajes, y a las que se podrán suscribir los consumidores de mensajes. Estos topics se pueden crear 'on the fly' al enviar un mensaje a un topic no existente a priori (en este caso, se creará tal topic).

Un topic es una cadena de texto en UTF-8, definida en niveles. Cada nivel se separa con el carácter "/". A continuación se muestra un ejemplo de un topic donde se publicarían las incidencias de la carretera V30.

```
carreteras/V30/incidencias
```

Un topic debe contener al menos 1 carácter, puede contener espacios en blanco (aunque no se recomienda) y es sensible a mayúsculas/minúsculas

Cuando un cliente se suscribe a un topic, tiene dos opciones: suscribirse a un topic exacto (indicando el topic entero), o suscribirse a un conjunto de topics. En el segundo caso, se puede especificar 'wildcards' para reemplazar con subtopics.

El wildcard '+' sustituye a un subnivel. Así, por ejemplo, una suscripción a:

```
carreteras/+/incidencias
```

se suscribe a cualquier topic que muestre incidencias de cualquier carretera.

El wildcard '#' sustituye a un número múltiple de subniveles (sólo se puede usar al final). Así, por ejemplo, una suscripción a:

```
carreteras/#
```

se suscribe a todos los mensajes que se publiquen dentro del topic `carreteras` o cualquiera de sus subtopics.

Por último, toda la comunicación y publicación de mensajes MQTT se realiza en el **broker de mensajería** MQTT. Existen diferentes implementaciones del estándar. En esta asignatura vamos a utilizar mosquitto (parte del proyecto eclipse). Para encontrar información sobre su instalación, uso y servicios que ofrece, ver el proyecto en http://www.eclipse.org/mosquitto/.