

Desarrollo de proyectos IoT utilizando Raspberry Pi como plataforma

Miguel Ángel Martínez Sánchez



Desarrollo de proyectos IoT utilizando Raspberry Pi como plataforma

Miguel Ángel Martínez Sánchez

Tutorizada por

Prof. Tutor Primero Apellido Apellido

Prof. Tutor Segundo Apellido Apellido

Prof. Tutor Tercero Apellido Apellido

Índice general

English Abstract	1
1. Los enunciados	5
1.1. Teoremas y demostraciones	5
1.1.1. Otros enunciados	5
MQTT	7
1.2. MQTT	7
1.3. Requisitos	8
1.3.1. Escalabilidad	8
1.3.2. Seguridad	9
1.3.3. Disponibilidad	9
1.3.4. Confiabilidad	10
1.3.5. Tiempo real	11
1.3.6. Control de datos	11
1.3.7. Descubrimiento de recursos	12
1.3.8. Documentación e interoperabilidad	12

II DESARROLLO DE PROYECTOS IOT UTILIZANDO RASPBERRY PI COMO PLATAFORMA

MQTT-SN	13
1.4. Redes de sensores inalámbricas	13
1.5. MQTT-SN	14
1.6. Arquitectura MQTT-SN	14
1.7. Funcionamiento	15
1.8. MQTT y MQTT-SN	15
OpenIoT	17
1.9. Introducción	17
1.10. Plataforma	17
1.10.1. Arquitectura	17
1.10.2. Funcionalidades	21
1.11. Proyectos	25
OpenIoT	27
1.12. Introducción	27
1.13. Plataforma	28
1.13.1. Arquitectura	28
1.13.2. Flujo de datos	35
1.13.3. Uso de la plataforma	37
1.13.4. Requisitos	39
1.13.5. Caso práctico	42
1.13.6. Casos de éxito con la plataforma	42

KAA	45
1.14. Arquitectura	45
1.15. Funcionamiento	48
1.15.1. Qué ofrece Kaa	50
1.16. Uso de la plataforma	56
1.16.1. Instalación	56
1.16.2. Partes de la plataforma	56
1.17. Casos prácticos	60
1.17.1. Encender un LED en la raspberry usando Kaa	60
1.18. Problemas encontrados	60

English Abstract

According to the guidelines, every dissertation should include a short english abstract at the beginning. In the abstract, you describe in general terms what is your dissertation about, the main points you want to make, and any important consequences that may arise.

Agradecimientos

1 | Los enunciados

1.1 Teoremas y demostraciones

| **Teorema 1.1 (Euclides).** *Esto es un Teorema. Se numeran a partir del 1 en cada capítulo. Como son importantes, tienen un cuadrado rojo al principio. Llevan letra cursiva.*

Demostración. Esto es la demostración. Al final de la demostración se puede ver un cuadrado rojo similar al de los teoremas. Las demostraciones no llevan letra cursiva.

| **Definición 1.1.** *Esto es una definición. Las definiciones son importantes; también llevan un cuadradito rojo.*

1.1.1 Otros enunciados

Observación 1.1. Esto es una observación, que dice que $e = mc^2$. Como las observaciones no son importantes, no llevan cuadrado rojo, y el tipo de letra no es cursiva.

Demostración. Si la demostración acaba en una fórmula, para poner el cuadrado rojo a la altura de la última formula, hay que usar la orden \qedhere, como en este caso:

$$e = mc^2.$$

Corolario 1.1. Esto es un corolario.

Proposición 1.1. Esto es una proposición.

Lema 1.1 (Gauss). Esto es un lema.

MQTT

1.2 MQTT

MQTT(Message Queue Telemetry Transport) es un protocolo usado para la comunicación M2M (Machine to machine) en el internet de las cosas. Fue creado por IBM en 1999. Está orientado a las redes de dispositivos pequeños, como sensores, debido a que consume muy poco ancho de banda. Se ha convertido en un protocolo muy usado en IoT, aunque también es ideal para aplicaciones móviles por su envío eficiente. Un ejemplo de uso es Facebook Messenger para iPhone y Android.

Se basa en el estandar publish/subscribe usado sobre TCP/IP. Esta comunicación publish/subscribe necesita de un "broker" o servidor para transmitir los paquetes.

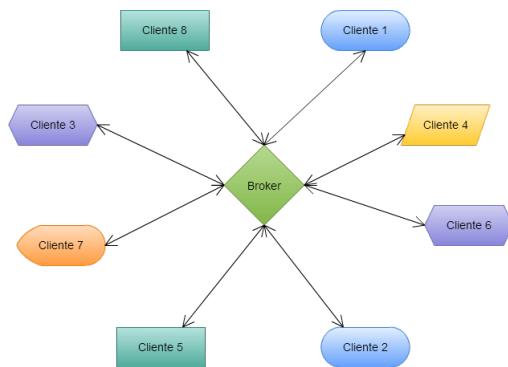


Figura 1.1: Topología MQTT.

8 DESARROLLO DE PROYECTOS IOT UTILIZANDO RASPBERRY PI COMO PLATAFORMA

La comunicación se basa en "topics" (temas) que el cliente que publica el mensaje envía y los nodos que desean recibirlo deben suscribirse a él. Un topic se representa mediante una cadena de caracteres separada por "/". De esta forma se pueden crear jerarquías como se puede ver en la figura 1.22

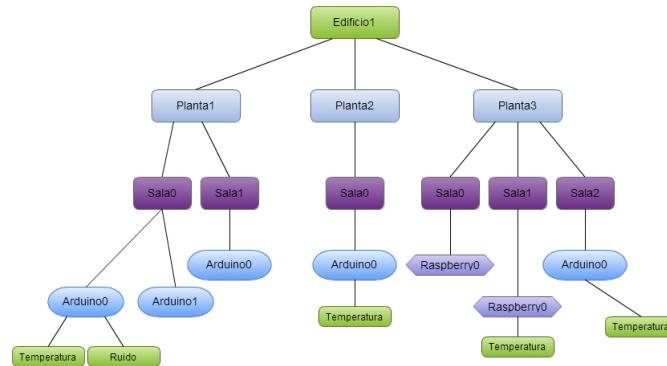


Figura 1.2: Jerarquía Topics.

1.3 Requisitos

MQTT como middleware IoT, necesita cumplir una serie de requisitos. En el escenario que se plantea hay algunos que no son indispensables y otro que sí lo son, como pueda ser la seguridad. Aquí se definen algunos de los requisitos.

1.3.1 Escalabilidad

MQTT tiene, en general, una alta escalabilidad, que va a depender en gran medida de la calidad de los nodos y de la topología que se use. La existencia de uno o más brokers dota al sistema de gran escalabilidad ya que la solución puede crecer solo con aumentar recursos en un único elemento. Para añadir un nuevo cliente, tan solo es necesario suscribirse a un topic.

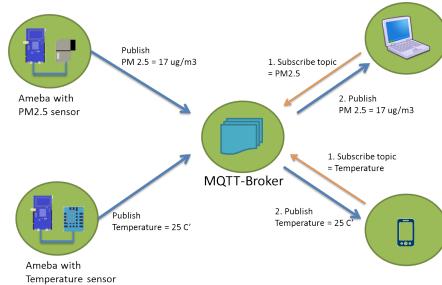


Figura 1.3: Escalabilidad MQTT

1.3.2 Seguridad

La seguridad es vital en un entorno IoT y para ello MQTT usa TLS junto con una autenticación de usuario/contraseña. Usa una autenticación usuario/contraseña a nivel de aplicación para establecer la conexión entre cliente y broker. Si TLS se resuelve correctamente, entra en juego dicha autenticación. Además, también existen políticas de autorización para denegar un determinado topic o para permitir una operación

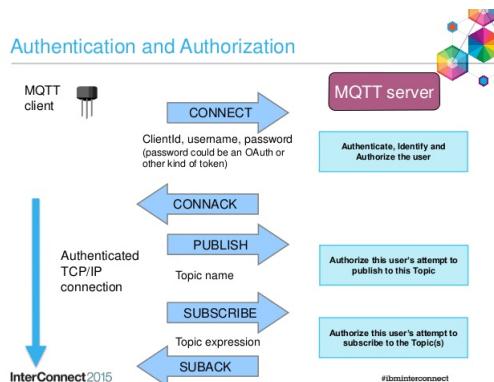


Figura 1.4: Autenticación MQTT

1.3.3 Disponibilidad

MQTT no tiene una gran disponibilidad, aunque con herramientas como平衡adores de carga, o el uso de los brokers en modo “bridge” se puede conseguir una buena disponibilidad. La disponibilidad permitirá poder seguir usando la red en caso de fallo en alguno de sus nodos.

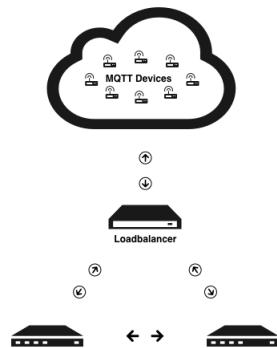


Figura 1.5: Balanceador de carga en MQTT

1.3.4 Confiabilidad

MQTT proporciona 3 grados de QoS para la entrega de los paquetes:

- **QoS 0: Como mucho una vez**

El mensaje PUBLISH se envía y el broker no manda ningún reconocimiento. El mensaje llega al broker una vez o ninguna y si llega, llegará a los subscriptores una vez o ninguna.



Figura 1.6: QoS 0

- **QoS 1: Al menos una vez**

Este nivel de calidad de servicio asegura que llega al broker al menos una vez. El broker tiene que reconocer la recepción con un paquete PUBACK. Si no recibe un acuse de recibo se vuelve a enviar de nuevo el mensaje PUBLISH con otro identificador distinto hasta que se reciba el reconocimiento. En este nivel no se garantiza que los paquetes no lleguen duplicados.

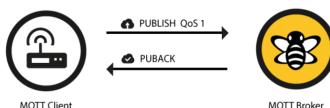


Figura 1.7: QoS 1

- **QoS 2: Exactamente una vez**

Este nivel de calidad asegura que el paquete llega una y solo una vez. Para ello se deben enviar además los paquetes PUBREC y PUBREL.

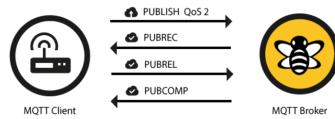


Figura 1.8: QoS 2

1.3.5 Tiempo real

Si tiempo real lo definimos en microsegundos, MQTT no cumple este requisito ya que una de las principales características de MQTT es que funciona sobre TCP. En el escenario que se plantea, estos microsegundos no son necesarios por lo que no es un requisito indispensable.

1.3.6 Control de datos

MQTT proporciona un control de datos como son el almacenamiento o el filtrado. De todo ello se encarga el broker. El broker puede almacenar los paquetes de un tópico mientras el cliente, suscrito a dicho tópico, esté offline. El broker también puede aplicar un filtrado en los tópicos, para que se aplique a un nivel o a todos los niveles de ese tópico:

- **Single level : +**

Para que el mensaje llegue a un cliente, éste debe de estar suscrito a un tópico.



Figura 1.9: Single level

co en el que solamente cambie el nivel indicado por el signo +. En este ejemplo, el mensaje destinado a **myhome/groundfloor/livingroom/temperature** o **myhome/groundfloor/kitchen/temperature** llegaría al cliente. Sin embargo, un mensaje destinado a **myhome/groundfloor/kitchen/brightness** o **myhome/firstfloor/kitchen/temperature** no llegaría al cliente.

- **Multi level: #**

En este caso, llegaría a cualquier cliente que esté suscrito a partir de ese nivel.



Figura 1.10: Multi level

Los tópicos que empiecen por el símbolo \$ están reservados para las estadísticas internas del broker MQTT. Un ejemplo puede ser:

- \$SYS/broker/clients/connected
- \$SYS/broker/clients/disconnected
- \$SYS/broker/clients/total
- \$SYS/broker/messages/sent

1.3.7 Descubrimiento de recursos

En MQTT no hay un mecanismo para el descubrimiento de recursos. El cliente se tiene que conectar al broker para su comunicación. Con MQTT-SN, una extensión de MQTT para redes de sensores, sí que hay un mecanismo de descubrimiento de nuevos dispositivos.

1.3.8 Documentación e interoperabilidad

MQTT es un middleware de código abierto perfectamente documentado en <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>

MQTT es un protocolo heterogéneo en cuanto a dispositivos y tecnologías se refiere. Puede funcionar prácticamente en cualquier dispositivo ya que es un protocolo muy sencillo que requiere poco procesamiento. Puede funcionar en un dispositivo móvil, PC, Raspberry...

MQTT-SN

1.4 Redes de sensores inalámbricas

Las redes de sensores o Wireless Sensor Networks (WSN) en inglés, se han incrementado en los últimos años. Estas redes tienen diferentes aplicaciones, como puede ser la vigilancia y seguridad, medicina, domótica o aplicaciones militares. Una red de sensores está formada por sensores y gateway para conectar con una red de datos. En estas redes es importante la comunicación de forma inalámbrica entre sensores, puesto que el número de nodos (sensores, actuadores...) es muy grande, y una infraestructura cableada tendría un coste muy elevado. Las características de estas redes son:

- Tolerancia a fallos
- Coste
- Ausencia de infraestructura de red
- Bajo consumo

A diferencia de las redes convencionales, los nodos no tienen conocimiento de la topología de la red, por lo que el enrutamiento cambia con respecto a éstas. Aquí es el nodo el que se informa de sus nuevos nodos a su alcance y de la manera de encaminarse hacia ellos. En la figura 1.36 se puede ver la estructura de una red de este tipo.

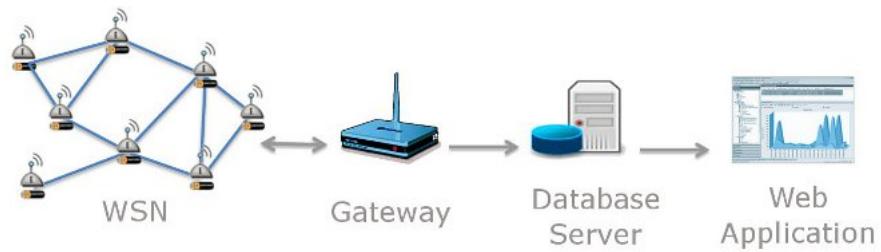


Figura 1.11: Multi level

1.5 MQTT-SN

MQTT es un protocolo usado sobre redes donde el ancho de banda sea limitado, sin embargo MQTT requiere del protocolo TCP/IP, útil para redes con dispositivos de aceptable procesamiento, puesto que ofrece una entrega correcta de los paquetes pero es demasiado complejo como para ser usado en redes de sensores, en las que los dispositivos son de poca memoria y de bajo procesamiento.

Es por ello donde surge MQTT-SN, un protocolo pub/sub específico para redes de sensores.

1.6 Arquitectura MQTT-SN

La arquitectura MQTT-SN se muestra en la figura 1.12

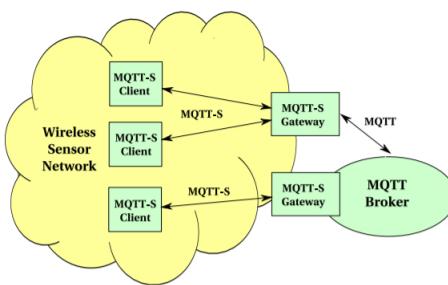


Figura 1.12: Arquitectura MQTT-SN

Hay 2 componentes: MQTT-SN clientes y MQTT-SN gateways. Los MQTT-SN clientes son los nodos en la red WSN. La comunicación entre clientes se hace a través del Gateway mediante el protocolo MQTT-SN. Los clientes realizan la comunicación pub/sub con un broker, localizado en una red tradicional, a través del Gateway. La comunicación entre el broker y el MQTT-Gateway se hace mediante el protocolo MQTT.

1.7 Funcionamiento

1.8 MQTT y MQTT-SN

OpenIoT

1.9 Introducción

Aunque existen muchas aplicaciones IoT, cada una hace uso de servicios diferentes debido a que no hay un modelo a seguir para integrar todos los servicios.

En IoT no existe un estándar en cuanto a servicios se refiere, como sí existe en redes IP. OpenIoT surge como un proyecto europeo que busca la unificación de esas soluciones. OpenIoT es una plataforma de código abierto que proporciona una interoperabilidad entre los servicios IoT en la nube. OpenIoT ofrece una infraestructura versátil para colecciónar datos de cualquier sensor disponible. Se hace uso también del concepto "datos enlazados", Linked Data en inglés. En el que los datos de los sensores se vinculan entre sí, para que puedan ser compartidos y entenderse entre ellos, ampliando así la información. Siempre buscando la principal característica en un entorno IoT, el Big Data.

1.10 Plataforma

1.10.1 Arquitectura

La arquitectura de OpenIoT está formada en 7 elementos principales divididos en 3 planos lógicos, tal y como se puede observar en la figura [1.13](#).

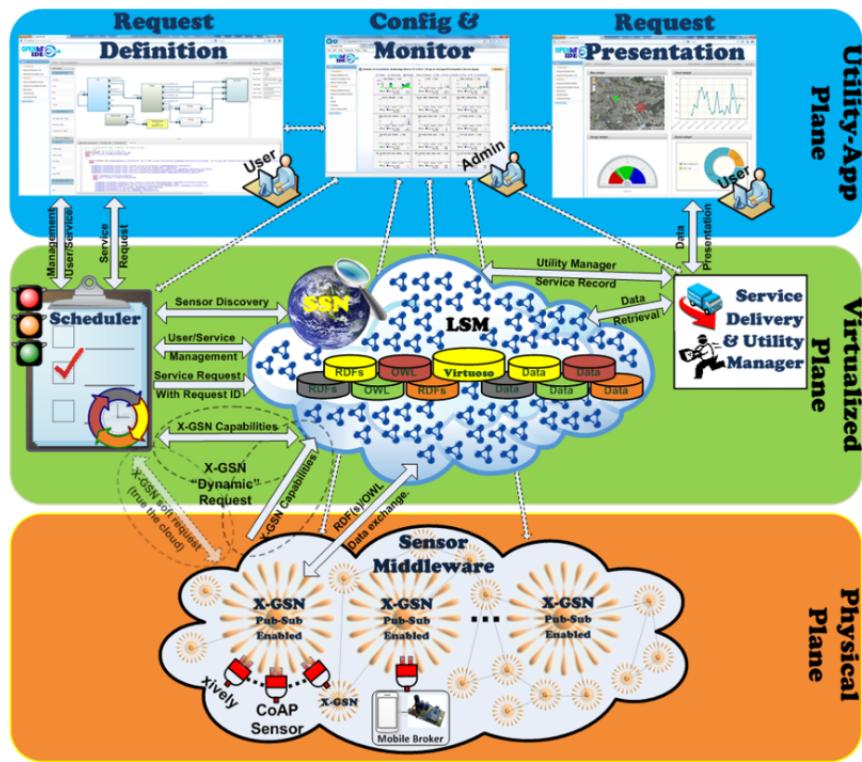


Figura 1.13: Arquitectura OpenIoT

El plano físico, virtualizado y un plano de utilidad.

Plano físico

- **The sensor middleware**

Se encarga de filtrar, combinar y colecciónar los datos de los sensores. OpenIoT usa X-GSN como middleware, una versión extendida del middleware GSN. Además, se usa un middleware publish/subscribe para la integración de sensores móviles.

Plano virtual

- **The scheduler**

El scheduler o planificador procesa peticiones de los servicios y asegura su correcto acceso a los recursos que solicitan. Entre sus principales tareas está la de descubrir sensores.

- **The cloud data storage**

Actúa como una base de datos en la nube, almacenando los datos enviados por los sensores. En OpenIoT se almacena también los metadatos asociados a esos datos. OpenIoT usa Linked Stream Middleware Light (LSM-Light)

- **The Service Delivery & Utility Manager**

Actúa, por un lado, combinando los flujos de datos para entregar el servicio solicitado. Hace uso de la descripción del servicio y de los recursos identificados por el Scheduler. Por otro lado, actúa como un servicio para realizar un seguimiento de las métricas de cada servicio (sensor). Como métricas se entiende la calidad, consumo, ancho de banda, volumen de datos...

Plano de utilidad

- **The request definition**

Este componente proporciona una interfaz web para la especificación de las peticiones de un servicio.

- **The request presentation**

Se encarga de la visualización de los servicios de salida (mostrar gráficas, flujos de datos...)

- **The configuration and monitoring**

Se encarga del control y configuración de los sensores y servicios incluidos en la plataforma OpenIoT.

La figura 1.14 muestra la arquitectura de OpenIoT en bloques. Se pueden ver los diferentes servicios situados en cada plano. En el plano físico se muestra el middleware X-GSN y el middleware publish/subscribe para sensores móviles. Se muestra también, entre otros, el servicio de seguridad (CAS) o la plataforma de almacenamiento en la nube (LSM-Light), ambos pertenecientes al plano virtual.

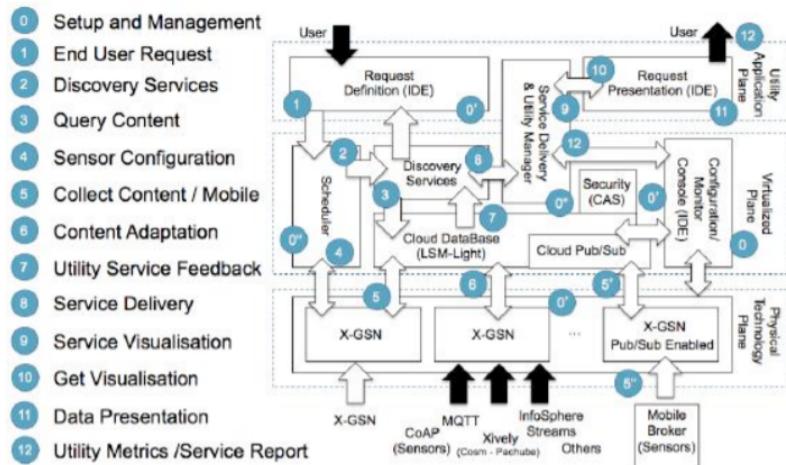


Figura 1.14: Bloques funcionales OpenIoT.

Scheduler

El Scheduler analiza cada petición de servicio y, de acuerdo con esto, interactúa con el resto de la plataforma OpenIoT, a través de la base de datos en la nube (Cloud DB). El scheduler tiene las siguientes funcionalidades:

- **Descubridor de recursos**

Es capaz de descubrir los sensores virtuales disponibles a partir de la entidad "availableSensors" de su base de datos. Esta lista está basada en la sintaxis RDF cuando se envía a través de X-GSN.

- **Control de servicios**

El usuario puede llevar un control de los servicios. Para ello se usan comandos como "register", "suspend", "unregister" o "update".

- **Actualizar recursos**

Un servicio que permite actualizar recursos

- **Obtener servicios** Se usa para obtener la descripción de un determinado servicio ya registrado.

Aunque el usuario pueda invocar a los distintos servicios, primero debe estar logueado correctamente dentro de la plataforma.

Service delivery & Utility Manager

Las principales funciones de este servicio son:

- Ejecutar y entregar los servicios pedidos.
 - Procesar los flujos de datos de la nube.
 - Seguimiento de los parámetros asociados al servicio. Por ejemplo, volumen de datos transmitidos o número de sensores usados.
 -

1.10.2 Funcionalidades

Integración de datos enlazados

Los datos enlazados es la forma que tiene la Web Semántica de vincular los distintos datos que están distribuidos en la web, así se puede llegar a la información relacionada que se hace referencia desde otros datos. Es decir, se le da una cierta capacidad de "razonar" a la web.

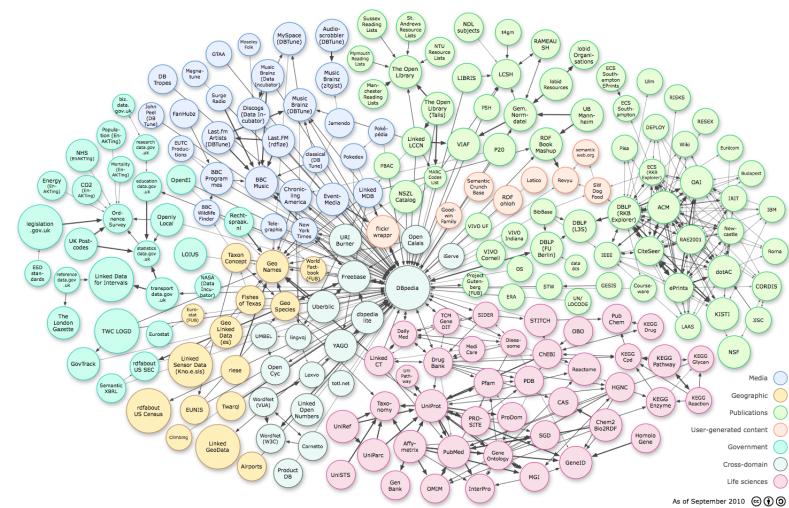


Figura 1.15: Conjunto de datos vinculados entre sí en la web.

La principal funcionalidad de OpenIoT es el almacenamiento y procesado de datos, recopilados por sensores, en la nube. Y que estos datos puedan ser compartidos

entre sí para poder ampliar la información. Es decir, hace uso del concepto de la web semántica. Para conseguir este concepto de datos enlazados ("Linked Data"), OpenIoT hace uso del middleware LSM.

Para la integración y compartición de datos en la nube, es necesario enviar los datos de forma correcta a la nube. Es aquí donde entra en juego X-GSN, un middleware usado en el plano físico para la distribución de datos. La idea principal de X-GSN es el uso de sensores virtuales. Estos sensores virtuales pueden ser abstracciones lógicas de uno o más sensores reales, objetos o cualquier entidad que capture datos.

Cada sensor virtual tiene una instancia almacenada en la nube en formato RDF (Resource description format), una sintaxis basada en SSN ontology. En la figura 1.16 se puede ver esta sintaxis. Estos metadatos de los sensores virtuales, se exponen como una representación "Linked Data" mediante el middleware LSM, así pueden ser descubiertos o adquiridos por componentes de las capas superiores en la plataforma OpenIoT. Esta representación en la que se exponen los datos es RDF.

```

@base <http://openiot.eu/test/id/> .

<sensor/5010> rdf:type aws:CapacitiveBead,ssn:Sensor;
    rdfs:label "Sensor 5010";
    ssn:observes aws:air-temperature ;
    phenonet:hasSerialNumber <sensor/5010/serial/serial2> ;
    ssn:onPlatform <site/narrabri/Pweather> ;
    ssn:offFeature <site/narrabri/sf/sf_narrabri> ;
    ssn:hasMeasurementProperty <sensor/5010/accuracy/acc_1> ;
    prov:wasGeneratedBy "AuthorName";
    DUL:hasLocation <place/location1>;
    lsm:hasSensorType <sensorType1>;

<sensor/5010/serial/serial2> rdf:type phenonet:SerialNumber;
    phenonet:hasId "5010" .

<site/narrabri/Pweather> rdf:type ssn:Platform ;
    ssn:inDeployment <site/narrabri/deployment/2013> .
<site/narrabri/deployment/2013> rdf:type ssn:Deployment.

<sensor/5010/accuracy/acc_1> rdf:type ssn:Accuracy ;
    qu:nationalValue "0.3"^^xsd:double ;
    DUL:hasParameter phenonet:degreeCelsius .

```

Figura 1.16: Sintaxis usada por XGSN.

La figura 1.17 muestra un registro de un sensor virtual, en el que se almacena una instancia del sensor (metadatos) en la nube. Mediante XGSN se envían esos metadatos y ,a través de LSM, se exponen como datos enlazados mediante la representación RDF, para que otros servicios puedan descubrirlo y obtener información.

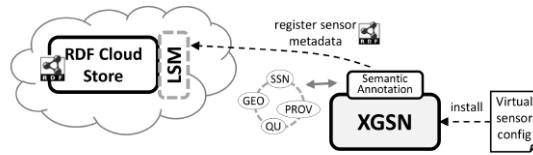


Figura 1.17: Registro de un sensor virtual.

Mobile broker

OpenIoT ofrece soporte para el descubrimiento y recolección de datos por parte de sensores móviles tales como pulseras, gafas, relojes, en definitiva, sensores incluidos en dispositivos móviles. Todo esto se realiza a través de un middleware publish/subscribe para IoT llamado CUPUS (Cloud-based Publish/Subscribe middleware). CUPUS tiene dos componentes principales: 1) un agente (mobile broker, de ahora en adelante) ejecutándose en un dispositivo móvil y 2) un motor de procesado en la nube basado en publish/subscribe (cloud broker, de ahora en adelante), que se encarga del procesamiento de los datos recopilados por los sensores. CUPUS soporta el contenido basado en publish/subscribe tanto en la nube como en los dispositivos móviles. En la arquitectura OpenIoT, los datos recopilados por los dispositivos móviles se anotan y almacenan en la nube, a través de X-GSN, de la misma forma que con los sensores estacionarios.

El mobile broker puede controlar los sensores conectados localmente y realizar un preprocesamiento de los datos adquiridos por los sensores y enviarlos a la nube. Además, también puede recibir publicaciones de la nube y notificar a los clientes que estén suscritos. Como se puede observar en la siguiente figura, el mobile broker recibe los datos de los sensores a través de un mensaje publish y los envía al cloud broker. Pudiendo también conectarse o desconectarse del mismo.

El cloud broker puede enviar una notificación al mobile broker. Por otro lado, el cloud broker envía una instancia de los datos del sensor ya procesados al almacenamiento RDF, a través de X-GSN, de la misma forma que en los sensores estacionarios, tal y como se ha visto anteriormente.

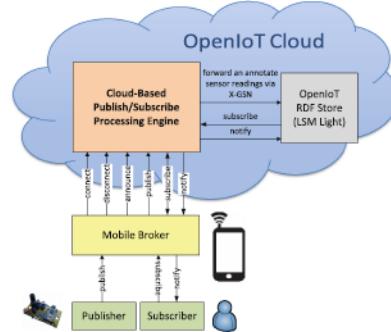


Figura 1.18: Arquitectura publish/subscribe

Descubrimiento y registro de recursos

OpenIoT controla el registro y descubrimiento de sensores a través de X-GSN. Cada sensor se tiene que registrar en el LSM y así, otras aplicaciones y usuarios pueden descubrirlo y obtener acceso a sus datos. Los sensores se registran enviando la representación de sus metadatos a través de XGSN. Como se ha visto, XGSN se encarga de crear la sintaxis RDF y enviarla al repositorio. LSM se encarga de exponer esos datos como datos enlazados para que puedan ser compartidos entre sí.

Autenticación y acceso a los recursos

La diversidad de aplicaciones interactuando en un entorno IoT hace que la seguridad sea un punto clave para poder proteger los datos. OpenIoT deja la seguridad en manos del servicio CAS (Central authorization service), encargado de la seguridad en la web.

La primera vez se redirecciona a los usuarios a la página de login para que se lleve a cabo una autenticación. Si esta autenticación es correcta, el CAS redirecciona al usuario a la pagina web original enviando un token. Este token se envía de un servicio a otro en cada petición y cada servicio se encarga de comprobar la validez del token.

1.11 Proyectos

Hay varios casos de éxito desarrollados mediante la plataforma OpenIoT. Algunos de ellos son Urban Crowdsensing, Phenonet o Silver Angel.

- **Urban Crowdsensing**

Este proyecto se aplica al medio ambiente. Realiza una monitorización del aire de entornos urbanos. En la figura 1.19 se muestra la plataforma OpenIoT sobre la que está construida esta aplicación

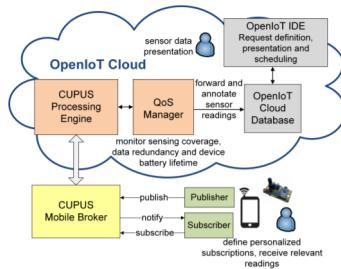


Figura 1.19: Arquitectura Crowd Sensing

Cualquier dispositivo recoge datos del aire en el ambiente urbano, como por ejemplo, O₃, O₂, CO₂. Estos datos son enviados a la nube OpenIoT para que sean procesados y filtrados y, posteriormente, reenviados a los smartphones, posibilitando así una monitización en tiempo real del aire de una ciudad. Pudiendo así, por ejemplo, evitar salir a practicar un deporte en situaciones de alta polución.

- **Phenonet**

Se aplica en la agricultura. Permite procesar y visualizar datos del terreno en tiempo real. Esto ayuda a los científicos a identificar las plagas de cultivos y así, incrementar la eficiencia y rendimiento en la agricultura.

- **Silver Angel**

Tiene una aplicación en la vida diaria de las personas. Está pensada para ayudar a la vida independiente de las personas mayores y, además, para facilitar el encuentro con amigos. Silver Angel permite reunir información de la cantidad de gente que hay en un determinado lugar, que hagan uso de Silver Angel, también reúne información del ruido ambiente o de los niveles de polen del medio ambiente. Así, permite a las personas reunirse dependiendo de sus preferencias;

muchas o pocas personas, un alto o bajo ruido...

Silver Angel emite alarmas para ayudar en la vida diaria de personas mayores. Cualquier objeto alrededor de la casa podría avisar de cualquier problema que se haya detectado. Por ejemplo, puede avisar cuando la puerta esté abierta o cerrada o cuando la temperatura de la casa esté por encima o por debajo de un umbral. Si detecta el problema, se envía una alarma a los usuarios que se hayan predefinido.

OpenIoT

1.12 Introducción

A pesar de la expansión de las aplicaciones IoT en la nube, la ausencia de una semántica que proporcione una interoperabilidad entre las diferentes aplicaciones es una de las principales limitaciones de IoT. Esta ausencia de una unificación se refleja en los diferentes vocabularios y formas para describir las cosas/objetos físicos. No existe un modelo a seguir para integrar todos los servicios, como sí existe en redes IP.

OpenIoT surge como un proyecto europeo que busca la unificación de esas soluciones. Se trata de una plataforma de código abierto que proporciona una convergencia entre los diversos sistemas IoT. Mezcla el concepto Cloud-Computing con el concepto de redes de sensores de IoT.

OpenIoT se basa en SSN como el modelo para la unificación de los diferentes sistemas IoT y flujo de datos. OpenIoT ofrece una infraestructura versátil para coleccionar datos de cualquier sensor disponible. Se hace uso del concepto "datos enlazados", Linked Data en inglés. En el que los datos de los sensores se vinculan entre sí, para que puedan ser compartidos y entenderse entre ellos, ampliando así la información. OpenIoT incluye también un middleware que facilita la recolección de datos de cualquier sensor disponible.

1.13 Plataforma

1.13.1 Arquitectura

La arquitectura de la plataforma se puede ver en la figura 1.20

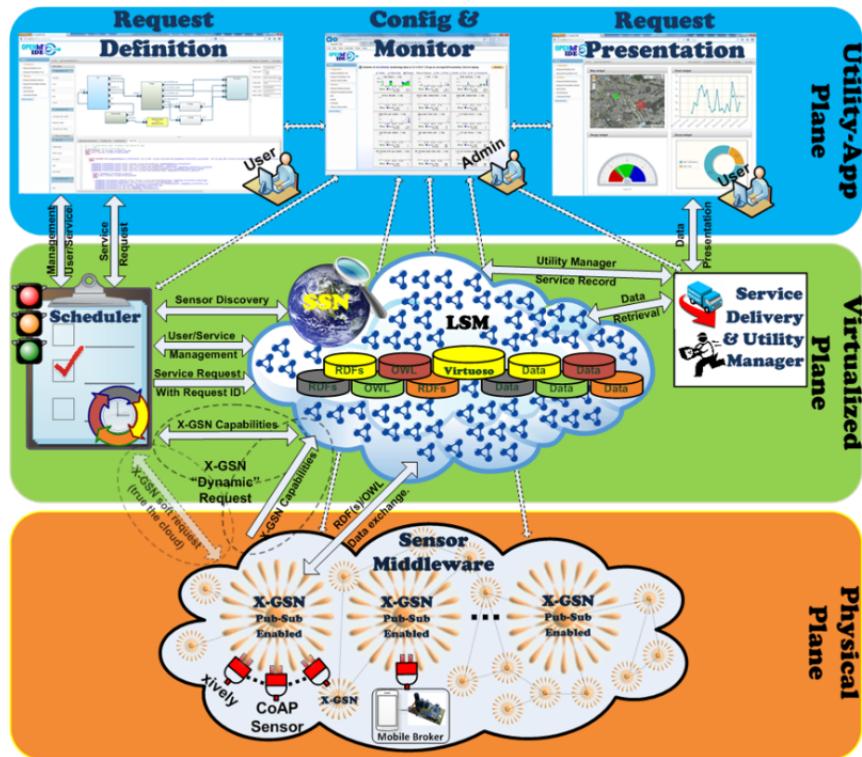


Figura 1.20: Arquitectura OpenIoT

Plano físico

En el plano físico se encuentran los sensores. En OpenIoT se usa X-GSN como middleware para el intercambio de información entre los sensores y la nube. Este middleware se encarga de filtrar, combinar y colecciónar los datos recopilados por los sensores. X-GSN es una versión extendida de GSN, cuya principal característica es el uso de sensores virtuales. Los datos de los sensores están escritos basándose en SSN (Semantic Sensor Network), una semántica para las redes de sensores. Esto proporciona una representación que hace más fácil compartir, descubrir, integrar e interpretar

los datos. Un ejemplo de esta semántica se puede ver en la figura 1.21

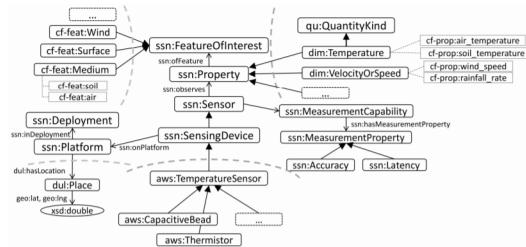


Figura 1.21: Semántica de SSN.

En X-GSN se configuran los sensores usando una descripción en XML, en la que se definen los campos de los sensores. Estos campos están asociados a la semántica SSN. En la figura 1.24 se puede ver un ejemplo de esta descripción.

Para realizar el registro de sensores en la plataforma OpenIoT, es necesario que cada sensor tenga una instancia del mismo almacenada en la nube. Este registro se realiza mediante el envío a la plataforma LSM de un archivo de los metadatos del sensor, en el que se definen las propiedades del mismo. Un ejemplo de este archivo se puede ver en la figura 1.22.

```

sensorName=opensense_1
source="http://planetdata.epfl.ch:22002/gsn?REQUEST=113&name=opensense_1"
author=opensense
sensorType=lausanne
sourceType=lausanne
information=Air Quality Sensors from Lausanne station 1
sensorId="http://lsm.deri.ie/resource/115080594572850"
feature="http://lsm.deri.ie/OpenIoT/opensensefeature"
fields="humidity"
field.humidity.propertyName="http://lsm.deri.ie/OpenIoT/Humidity"
field.humidity.unit=Percent
field.temperature.propertyName="http://lsm.deri.ie/OpenIoT/Temperature"
field.temperature.unit=C
  
```

Figura 1.22: Metadatos de un sensor.

mediante un archivo XML enviado a la plataforma LSM. En este XML se usa una descripción en formato RDF. Los sensores, situados en el plano físico, envían una instancia de sus datos (situación geográfica, memoria...) a la nube. Estos metadatos se envían en formato RDF, una semántica usada en la Web enlazada ("Linked Web", en inglés) y así pueden conectar con LSM.

En la figura 1.23 se puede ver el proceso para el registro de un sensor en OpenIoT. Inicialmente se crea y se configura el sensor, mediante la descripción XML. Un ejemplo de esta descripción se puede ver en la figura 1.24. Posteriormente se envía un archivo a la plataforma LSM en la nube conteniendo las propiedades del sensor. Éste se transforma en una descripción RDF, una descripción usada en la Web Semántica, que permite una representación de datos enlazados, lo que facilita el descubrimiento de recursos.

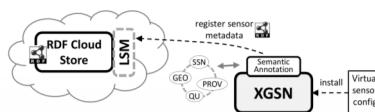


Figura 1.23: Registro de un sensor virtual

```

<?xml version="1.0" encoding="UTF-8"?>
<virtual-sensor name="demo_weatherstation" priority="10" >
<processing-class>
<class-name>org.openiot.gsn.vsensor.LSMExporter</class-name>
<init-params>
<param name="allow-nulls">false</param>
<param name="publish-to-lsm">true</param>
</init-params>
<output-structure>
<field name="temp" type="double"/>
<field name="humidity" type="double" />
</output-structure>
<processing-class>
<description>CSIRO demo station</description>
<life-cycle pool-size="10" />
<addressing>
<streams>
<stream name="input1">
<source alias="source1" sampling-rate="1" storage-size="1">
<address wrapper="csv">
<predicate key="file">data/station_1057.csv</predicate>
<predicate key="fields">timed, temp, humid, co_2, co2_4, no2</predicate>
<predicate key="formats">timestamp(d/M/y H:m), numeric, numeric, numeric, numeric, numeric</predicate>
<predicate key="bad-values">NaN,6999,-6999,null</predicate>
<predicate key="timezone">Etc/GMT-2</predicate>
<predicate key="sampling">4000</predicate>
<predicate key="check-point-directory">csv-check-points</predicate>
</address>
<query>select * from wrapper</query>
</source>
<query>select temp as temp,humid as humidity, timed from source1</query>
</stream>
</streams>
</virtual-sensor>

```

Figura 1.24: Configuración de un sensor.

OpenIoT ofrece soporte para el descubrimiento y recolección de datos por parte de sensores móviles tales como pulseras, gafas, relojes, en definitiva, sensores incluidos en dispositivos móviles. Todo esto se realiza a través de un middleware publish/subscribe para IoT llamado CUPUS (Cloud-based Publish/Subscribe middleware). CUPUS tiene dos componentes principales: 1) un agente (mobile broker, de ahora en adelante) ejecutándose en un dispositivo móvil y 2) un motor de procesamiento en la nube basado en publish/subscribe (cloud broker, de ahora en adelante), que se encarga del procesamiento de los datos recopilados por los sensores. CUPUS soporta el contenido

basado en publish/subscribe.

En la arquitectura OpenIoT, los datos recopilados por los dispositivos móviles se anotan y almacenan en la nube, a través de X-GSN, de la misma forma que con los sensores estacionarios.

El mobile broker puede controlar los sensores conectados localmente y realizar un preprocesamiento de los datos adquiridos por los sensores y enviarlos a la nube. Además, también puede recibir publicaciones de la nube y notificar a los clientes que estén suscritos. Como se puede observar en la figura 1.25, el mobile broker recibe los datos de los sensores a través de un mensaje publish y los envía al cloud broker. Pudiendo también conectarse o desconectarse del mismo.

El cloud broker puede enviar una notificación al mobile broker. Por otro lado, el cloud broker envía una instancia de los datos del sensor ya procesados al almacenamiento RDF, a través de X-GSN, de la misma forma que en los sensores estacionarios, tal y como se ha visto anteriormente.

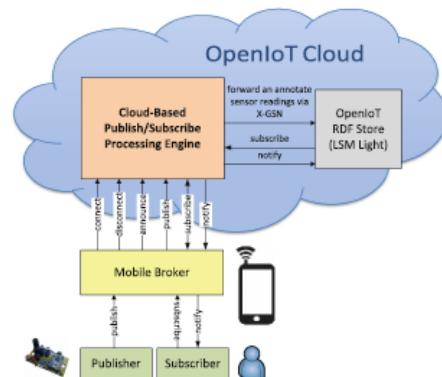


Figura 1.25: Arquitectura publish/subscribe

Plano Virtual

El plano virtual está compuesto por el almacenamiento en la nube (LSM-Light), el Scheduler y el servicio de entrega.

LSM-Light (Linked Sensor Middleware Light) es el componente principal de OpenIoT. Es la infraestructura de almacenamiento en la nube. Esta infraestructura, además de almacenar datos y metadatos, también es capaz de ofrecer computación en la nube (software) como por ejemplo Scheduler y SD&UM.

LSM considera los sensores como fuentes de entrada de datos. Los datos provenientes de los sensores se transforman en una representación de datos enlazados, como por ejemplo, RDF. Existen dos formas de importar datos en LSM, pull y push. Una es la fuente de datos (X-GSN, CoAP...) la que se encarga de enviar los datos y la otra es el propio LSM quien obtiene los datos periódicamente. LSM está compuesto por dos módulos, LSM-Client y LSM-Server

OpenIoT usa Virtuoso, un middleware y motor de almacenamiento, que combina RDF, XML y base de datos virtuales en un solo sistema. Es el corazón de LSM-Light.

El scheduler se encarga de formular las peticiones realizadas por los usuarios y de acuerdo con ellas interactúa con la plataforma OpenIoT a través de la base de datos en la nube. El Scheduler tiene dos funciones principales: descubrir sensores y controlar los servicios. Todo esto se realiza mediante peticiones a la BBDD. Las peticiones se realizan en lenguaje SPARQL, un lenguaje usado en la web semántica para consulta de sentencias RDF. En la figura 1.26 se observa la Cloud DB junto con cada uno de los servicios que realiza peticiones. Request presentation y Request Definition realizan las peticiones a través de otros servicios (Scheduler y SD&UM).

El servicio Request Definition es quien realiza las peticiones al Scheduler para que éste las procese. La forma que tiene de enviar las peticiones es mediante una API, especificada en la documentación.

El SD&UM (Service Delivery & Utility Manager), al igual que el Scheduler, realiza peticiones a la Cloud DB mediante una API.

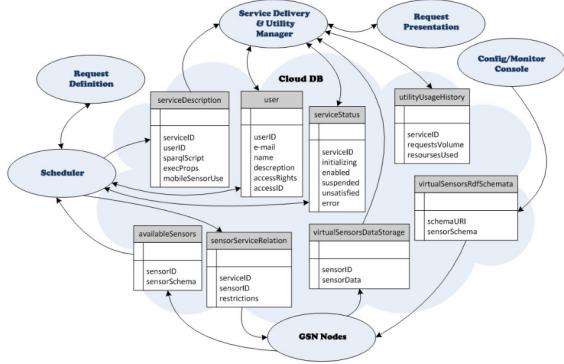


Figura 1.26: Relación de cada servicio con la Cloud DB

Plano de utilidad

En este plano se sitúan las interfaces de usuario. Request Definition es una aplicación que permite a los usuarios visualizar sus servicios de OpenIoT usando una interfaz basada en nodos. Cada modelo de grafos se divide en aplicaciones, siendo cada una de estas aplicaciones un conjunto de servicios que describen a la aplicación. Esto le permite al usuario controlar diferentes aplicaciones desde un solo punto. Todos estos servicios se almacenan en el Scheduler y se cargan automáticamente cuando el usuario accede a la web. Un ejemplo de esta interfaz se muestra en la figura 1.27

34 DESARROLLO DE PROYECTOS IOT UTILIZANDO RASPBERRY PI COMO PLATAFORMA

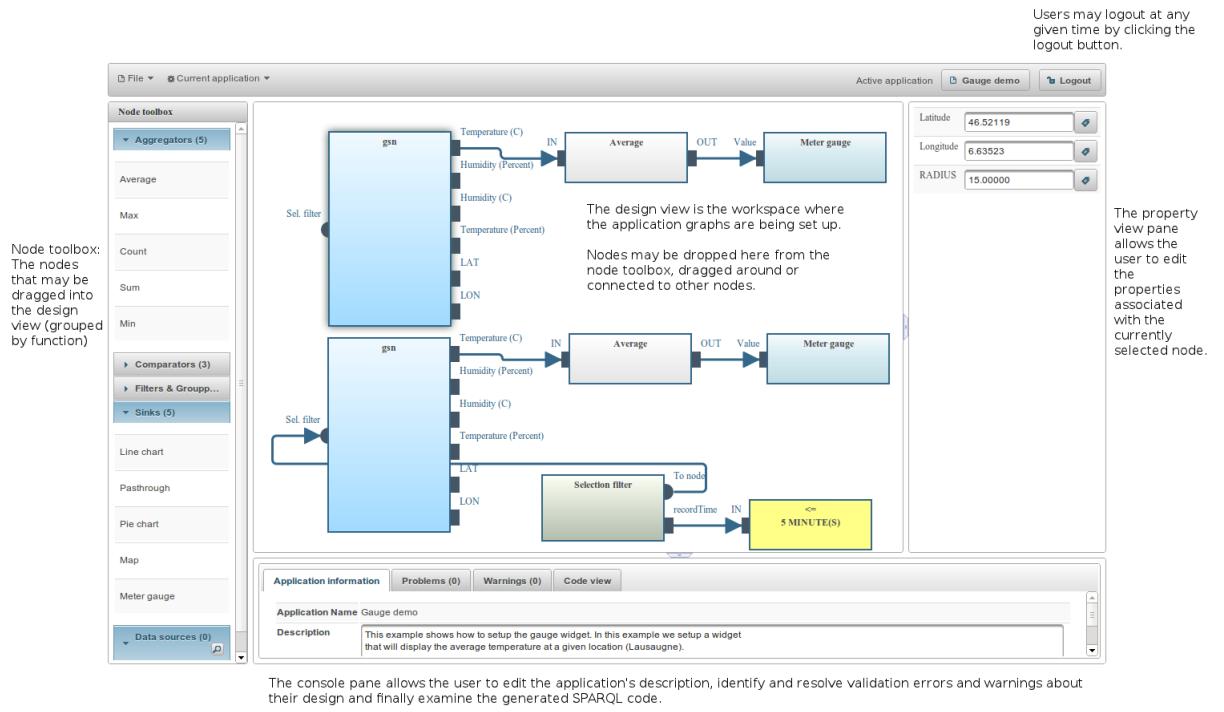


Figura 1.27: Request Definition

Request Presentation es una aplicación web que proporciona al usuario una interfaz visual de los servicios que previamente ha creado en Request Definition. Obtiene la información de los nodos de Request Definition y muestra una interfaz con los datos.

En la figura 1.28 se muestra un ejemplo con la definición del servicio en Request Definition, y en la figura 1.29 la interfaz gráfica de los datos en Request Presentation, una vez recopilados.

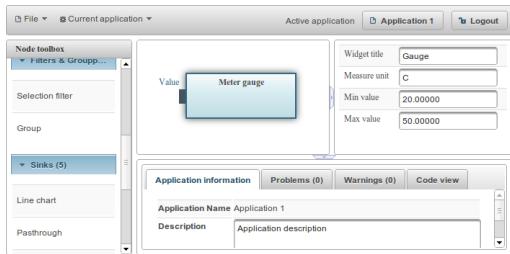


Figura 1.28: Definition

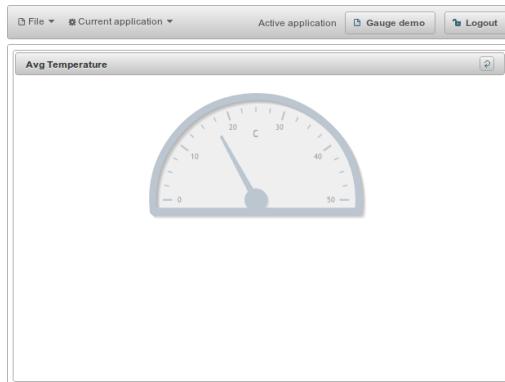


Figura 1.29: Presentation

El IDE es el otro de los servicios situados en el plano de utilidad. Proporciona accesibilidad a los otros módulos o servicios de OpenIoT. Otra de las funcionalidades que soporta IDE es la monitorización. Para implementar dicha funcionalidad se usa JavaMelody. Entre las funciones de monitorización se encuentra proporcionar datos sobre el tiempo medio de respuesta o el número de ejecuciones, la toma de decisiones ante problemas o, mostrar gráficos sobre número de sesiones, consumo de java o número de ejecuciones.

1.13.2 Flujo de datos

En base a la arquitectura que se muestra en la figura 1.13, la figura 1.30 representa un ejemplo del flujo que siguen los datos en la plataforma.

X-GSN publica los datos de los sensores virtuales basados en la configuración local de cada nodo (sensor). Paso 0.

Los usuarios realizan peticiones al Scheduler (paso 1) de los sensores disponibles con determinados atributos usando la interfaz Request Definition.

El Scheduler ejecuta (paso 2) estas peticiones (en lenguaje SPARQL) enviadas por los usuarios.

Una vez que tenga la respuesta (los sensores disponibles) se envía de vuelta al Scheduler (paso 3) y éste la reenvía al módulo Request Definition (paso 4), mostrándose la información al usuario.

El usuario, con ayuda de Request Definition, define peticiones para realizar determinadas reglas sobre los sensores analizados. Esta información se guarda en un objeto OSDSpec (Figura 1.34). Este objeto se envía entonces al Scheduler con la ayuda de

'registerService (paso 5).

El Scheduler analiza la información recibida y envía la petición al servicio necesario (paso 6).

Una vez que se haya configurado, el usuario puede usar el módulo Request Presentation para visualizar los datos del servicio registrado.

Con ayuda del SD&UM 'getAvailableAppIDs' el Request Presentation (pasos 7,8,9 y 10) recupera todos los servicios/aplicaciones registrados de acuerdo a un usuario específico

El usuario realiza una petición para recuperar los resultados relacionados con el servicio en concreto. Esto se hace enviando una petición ("pollForReport") desde Request Presentation al SD&UM pasándole el ID de la aplicación ('application ID') (paso 11). El SD&UM realiza una petición ("getService") (paso 12) para solicitar toda la información relacionada al servicio.

El servicio proporciona la información al SD&UM (paso 13).

El SD&UM analiza la información, disponible en un objeto OSMO, y reenvía el script SPARQL incluido, el cual ha sido creado por Request Definition (paso 5) y almacenado por el Scheduler (paso 6), a la interfaz SPARQL del servicio (paso 14).

El resultado se envía al SD&UM (paso 15) en formato SparqlResultsDoc

El SD&UM lo reenvía al Request Presentation (paso 16) en un objeto que incluye información de cómo esos datos se deben presentar (Figura 1.32).

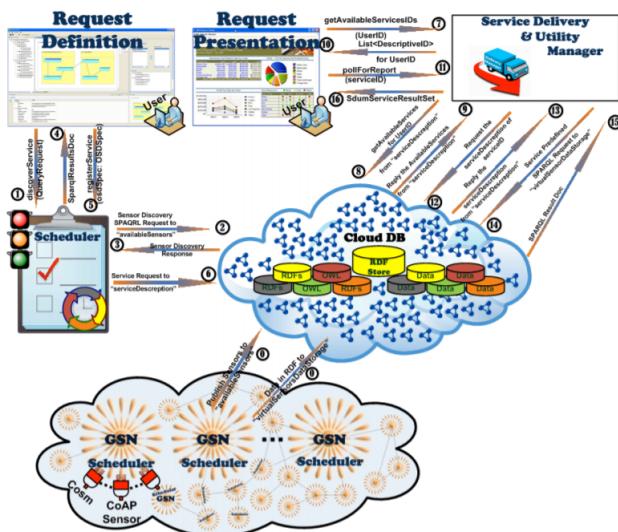
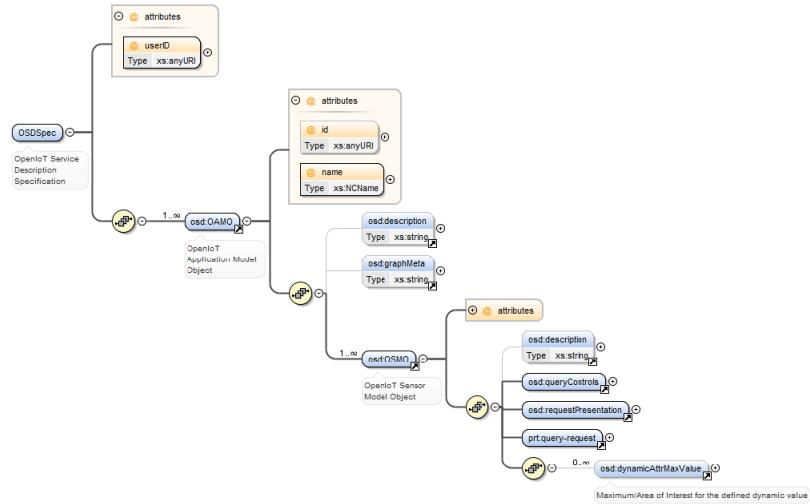
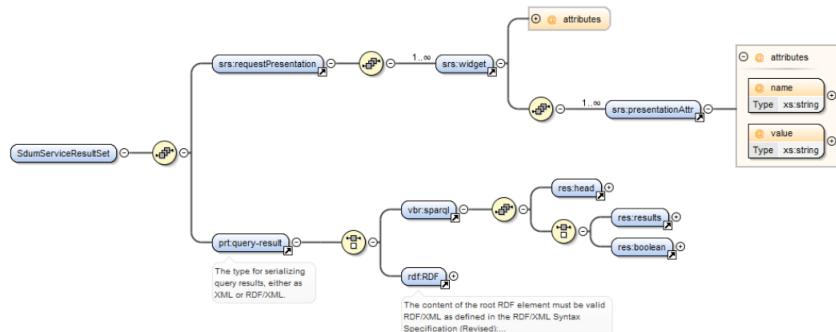


Figura 1.30: Flujo de datos en OpenIoT

Figura 1.31: Grafo de un objeto `OSDSpec`.Figura 1.32: Grafo de un objeto `SdumServiceResultSet`.

1.13.3 Uso de la plataforma

...

Ventajas

En IoT cualquier cosa u objeto (TV, Smartphone...) genera una gran cantidad de eventos. Cada uno de estos objetos usa tecnologías diferentes y, desarrollar aplicaciones para manejarlos es un desafío.

ciones y servicios para controlar todo esto puede llegar a ser una tarea muy complicada. Por ello, se usan middlewares con el fin facilitar el desarrollo proporcionando una integración entre la comunicación y la computación de los dispositivos. Entre los middleware se encuentra MAPS, TinyDB o MQTT, mencionado anteriormente.

A pesar de que con los middleware se facilita el desarrollo de aplicaciones en IoT, cada middleware cumple una serie de requisitos que puede que no sirvan para una determinada aplicación. Por ejemplo, en este caso, MQTT no cumplía alguno de los requisitos exigidos. No hay un middleware que reúna todas necesidades exigidas. Es aquí donde aparece OpenIoT, que permite reunir los servicios en una única plataforma.

La principal ventaja de OpenIoT es la unificación de los diferentes sistemas o servicios IoT, así como el envío de datos.

A través de la herramienta IDE (Integrated Development Environment) Core (figura 1.34) que ofrece OpenIoT, se puede controlar las aplicaciones IoT. Esta herramienta integra muchas de las herramientas de OpenIoT en una. Permite configurar los sensores para su integración en X-GSN (Schema Editor), monitorización del estado de los servicios IoT (SDUM) o definir los servicios IoT (Request Definition). Esto permite el desarrollo de aplicaciones IoT de una manera más rápida y sencilla.



Figura 1.33: OpenIoT IDE

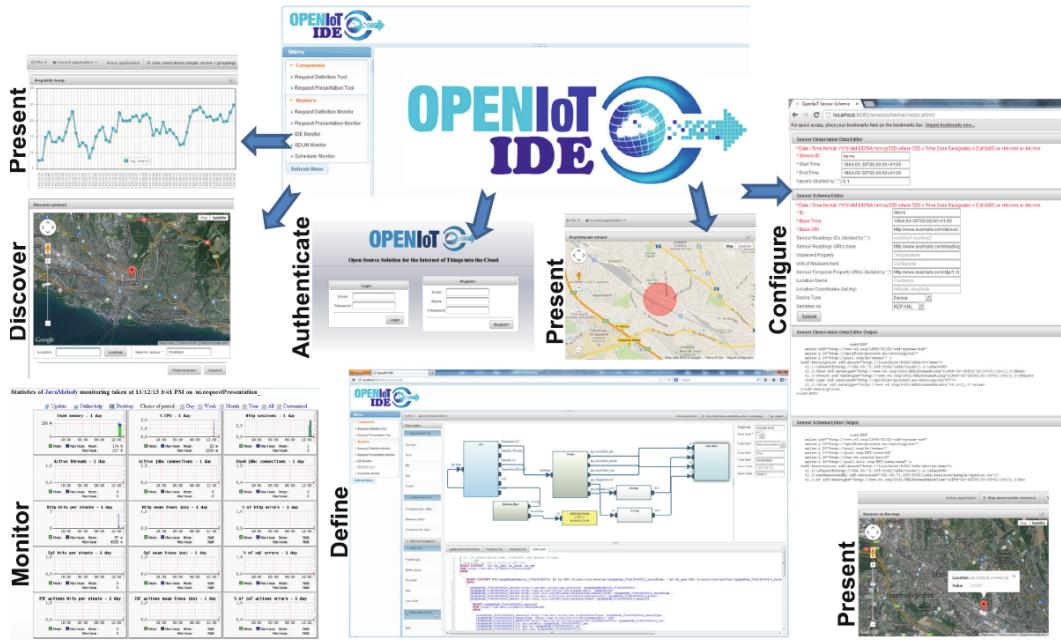


Figura 1.34: Servicios integrados en IDE

1.13.4 Requisitos

Las aplicaciones IoT deben cumplir una serie de requisitos, algunos de ellos necesarios en un entorno IoT. Algunos de estos requisitos son:

Escalabilidad

OpenIoT es una plataforma totalmente escalable debido a la característica de descubrimiento de recursos que posee. Se pueden añadir nuevos nodos tan solo configurando el sensor y enviando la información sobre sus datos (metadatos) al almacenamiento en la nube. Una vez registrado el sensor, el usuario puede acceder al nuevo nodo (sensor).

Disponibilidad

OpenIoT está formado por varios módulos/servicios, cuyo funcionamiento es responsable del funcionamiento global de la plataforma. Por ello, un fallo en alguno de los servicios (i.e el Scheduler), implica un fallo en la plataforma. Debido a esta dependencia, se puede decir que OpenIoT no dispone de una gran disponibilidad.

Seguridad

La diversidad de aplicaciones interactuando en un entorno IoT hace que la seguridad sea un punto clave para poder proteger los datos. OpenIoT deja la seguridad en manos del servicio CAS (Central authorization service), encargado de la seguridad en la web.

La primera vez se redirecciona a los usuarios a la página de login para que se lleve a cabo una autenticación. Si esta autenticación es correcta, el CAS redirecciona al usuario a la pagina web original enviando un token. Este token se envía de un servicio a otro en cada petición y cada servicio se encarga de comprobar la validez del token.

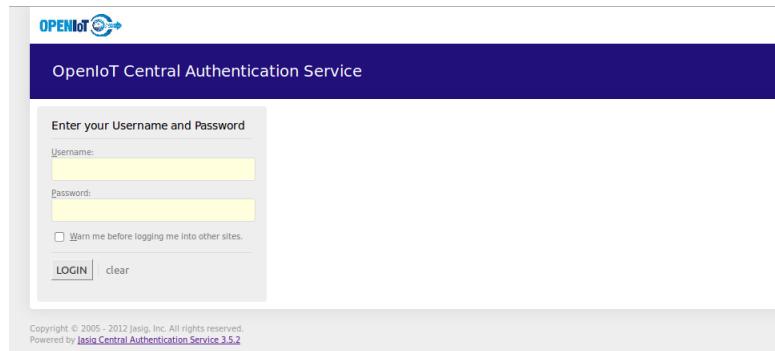


Figura 1.35: Login en CAS

La seguridad en OpenIoT se divide en 3 módulos:

- **Security Server:** En este módulo se usa CAS para la autenticación y autorización. Sus datos se almacenan en LSM-server. Cada cliente se tiene que registrar en CAS.

- **Security Client:** Este módulo proporciona control de acceso y autenticación. Se puede usar en aplicaciones web que interactúan con los usuarios.
- **Security Management:** Se trata de un módulo que se usa para llevar un control sobre servicios, usuarios y permisos.

Descubrimiento de recursos

Una de las características principales de OpenIoT es el descubrimiento de recursos. Una vez que los sensores envían su configuración a la nube a través de X-GSN, OpenIoT mediante el Scheduler, haciendo uso de la semántica SSN, es capaz de descubrir nuevos sensores o nodos.

Control de recursos

OpenIoT, a través de la herramienta SDUM, permite llevar una constante monitoreo sobre los recursos. Esto permite, además de la recopilación y visualización de los datos, llevar un control de los recursos.

Tiempo real

OpenIoT proporciona un servicio en tiempo real. El usuario a través de las interfaces de usuario puede realizar un seguimiento en tiempo real de los datos recopilados por los sensores.

Interoperabilidad

La interoperabilidad de los servicios IoT es la idea fundamental por la que surgió OpenIoT. Se buscaba unificar distintos servicios en una única plataforma. En OpenIoT se unifica todo lo relacionado con redes de sensores (Middleware...) y lo relacionado con el Cloud Computing (base de datos, herramientas de procesamiento...).

Distribuido

1.13.5 Caso práctico

1.13.6 Casos de éxito con la plataforma

Entre quién ha desplegado la plataforma con éxito se encuentra:

- Universidad nacional de Irlanda - DERI
- Universidad de Zagreb - FER
- SENSAP S.A
- CSIRO
- Universidad de Estambul

OpenIoT en la industria

Cada vez más, en la industria se instalan un gran número de sensores para controlar el proceso de producción de una planta. Estos sensores generan un gran volumen de datos por lo que es necesario una solución para capturar, almacenar y procesar esos datos. SENSAP S.A (www.sensap.eu) ha desarrollado una solución basada en OpenIoT para la monitorización y seguimiento del flujo de materiales en un proceso de producción. Le permite definir y visualizar dinámicamente los KPIs (indicadores de rendimiento en la industria). En este entorno, los KPIs son el flujo de datos necesarios, los cuales son: A)recopilados por sensores físicos situados en la planta, B)transformados en el modelo de datos EPC-IS, un estándar usado en la industria, C) transmitidos al middleware X-GSN que asegura una anotación semántica de estos datos y su posterior publicación a la nube OpenIoT. Los sensores virtuales son capaces de calcular entre otros:

- Tasa de operación para un proceso específico
- Métricas de utilización de las maquinas
- Tasa de producción por tipo de producto
- Porcentaje de tiempo una operación que ha sido completada

Una vez que se publica la información a la nube, los fabricantes son capaces de obtener y sintetizar la información de los sensores virtuales, con el fin de calcular los

KPIs para los determinados procesos.

Para llevar a cabo este proyecto, se usó la siguiente implementación:

- Sensores: Sensores físicos con el fin de calcular KPIs (tasas de operación, información de calidad...). Sensores ópticos, escáneres de códigos de barras, son algunos de los sensores utilizados en este proyecto.
- S-BOX: Productos propios desarrollados por SENSAPE, para colecciónar datos de los sensores y transformarlos en eventos EPC-IS, asociados a los procesos de fabricación.
- X-GSN: Ese flujo de datos EPC-IS se envía al middleware X-GSN. Siguiendo el proceso de OpenIoT, el middleware X-GSN convierte ese flujo de datos en la anotación SSN.
- LSM: La información KPIs se envía a la nube (LSM Cloud) a través de X-GSN, tal y como marca el proceso de OpenIoT
- Visualización: Usando la herramienta Request Definition de OpenIoT, se definen servicios que calculan los KPIs asociados al proceso de fabricación. Este cálculo lo realiza a partir de los datos disponibles en el LSM Cloud.

OpenIoT en la agricultura

Otros de los proyectos en los que se ha usado OpenIoT ha sido Phenonet, desarrollado por CSIRO. Se trata de un proyecto usado en la agricultura que permite procesar y visualizar datos del terreno en tiempo real. Esto ayuda a la toma de decisiones sobre el cultivo como, por ejemplo, planificar los recursos de agua o de nitrógeno en el cultivo y así, incrementar la eficiencia y rendimiento. La arquitectura de Phenonet se puede ver en la figura 1.36

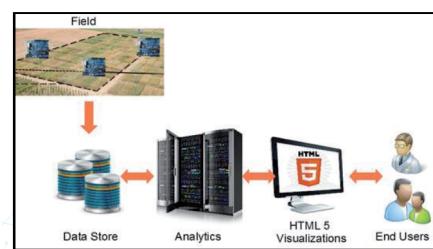


Figura 1.36: Arquitectura de Phenonet

En el campo ('field') se sitúan los sensores para medir temperatura, humedad o velocidad del viento entre otros. Los datos y metadatos se almacenan y posteriormente se procesan y analizan mediante el componente 'Data analysis'. A este componente se accede a través de una API mediante HTML. Phenonet está basado en los siguientes módulos de OpenIoT:

- X-GSN: Se encarga de los dato provenientes del Data Store/Field.
- Scheduler y SD&UM: Usado para construir un experimento Phenonet en OpenIoT.
- LSM-Light: Se almacenan los datos existentes en Data Store para poder permitir el descubrimiento de sensores.
- Request Definition y Presentation: Estas herramientas se usan para el diseño.

KAA

1.14 Arquitectura

Kaa es un middleware de código abierto que, al igual que OpenIoT, busca un control e integración de las aplicaciones IoT. Kaa tiene un poderoso back-end que facilita el desarrollo de aplicaciones en un entorno IoT. Kaa soporta múltiples plataformas en el lado del cliente, mediante puntos finales (SDKs), en diferentes lenguajes de programación. El SDK es una librería 'empotrada' en el dispositivo conectado. Esto, junto con un lenguaje de definición de datos ("data schema"), hace que Kaa sea una plataforma muy rápida y flexible. Kaa ha sido diseñada como una plataforma robusta y fácil de usar.

En la figura 1.38 se puede ver su arquitectura.

Un cluster Kaa representa un número de nodos (Kaa server) interconectados. La figura 1.39 muestra un flujo de datos común, donde se puede ver un cluster Kaa, formado por un kaa server.

Nota: La versión de la plataforma analizada en este documento ha sido la 0.10, la última hasta la fecha.

//////////Palabras clave: CLUSTER,SDK,NODE,SCHEMA,LOG/////////

46 DESARROLLO DE PROYECTOS IOT UTILIZANDO RASPBERRY PI COMO PLATAFORMA

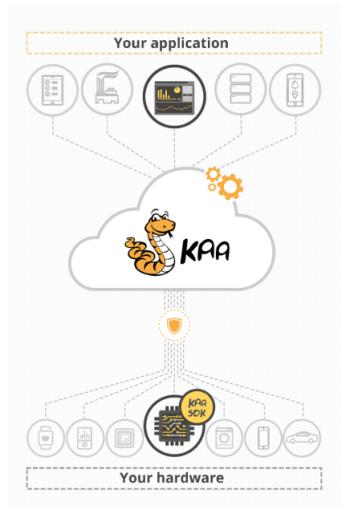


Figura 1.37: Arquitectura Kaa

Un conjunto de nodos interconectados representa un Cluster Kaa. Un Cluster requiere bases de datos SQL y NoSQL para almacenar los datos y metadatos de los endpoints. El servidor Kaa usa Apache Zookeeper para coordinar los servicios.

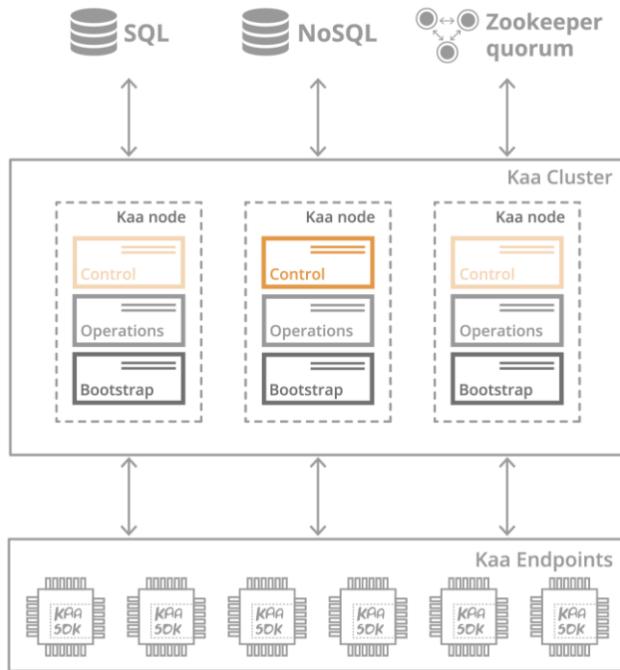


Figura 1.38: Cluster Kaa

Un nodo en un cluster está ejecutando una combinación de servicios de control, operaciones y bootstrap.

- **Servicio de control:** Es el encargado del sistema de datos, procesa llamadas de las APIs y envía notificaciones. Recibe continuamente información de Zookeeper. Para conseguir una alta disponibilidad, un Cluster Kaa tiene que incluir al menos dos nodos con el servicio de control activado. En el modo de alta disponibilidad, uno de ellos estará activo y el otro en standby, siendo zookeeper quien se encargue de su control y activación.
- **Servicio de operaciones:** Es el encargado de procesar y enviar peticiones a los endpoints.
- **Servicio bootstrap:** Es el encargado de establecer la conexión con los endpoints. Envía información a los endpoints sobre los parámetros de conexión que pueden ser dirección IP, puerto, protocolos...

1.15 Funcionamiento

La figura 1.39 muestra un flujo de datos en Kaa. Los sensores recogen datos y, a través del SDK, los envían al servidor. El SDK dependerá de la plataforma y del lenguaje de programación. El envío de datos al servidor se realiza en un formato común, previamente definido en el Servidor mediante los "Log Schema".

Como plataforma Cloud-IoT que es, se almacenan esos datos para que puedan ser procesados y analizados posteriormente. Kaa no ofrece un análisis de los datos como sí ofrecía OpenIoT, si no que tan solo ofrece una persistencia de los datos. Kaa ofrece tanto bases de datos relacionales como no-relacionales para el almacenamiento. MariaDB y MongoDB respectivamente, son las bases de datos por defecto usadas por Kaa.

La plataforma se organiza en cluster conectados entre sí, siendo cada cluster un nodo o servidor. Por defecto, está formada por un único cluster.

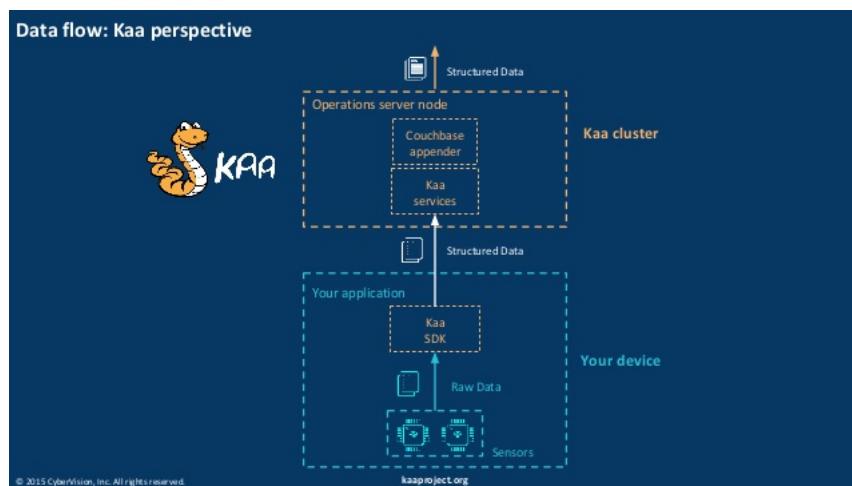


Figura 1.39: Flujo de datos en Kaa

Esquemas de datos

Los datos se recolectan desde los endpoints y se envían al servidor en el formato definido por el 'log schema', previamente creado por el desarrollador para la aplicación en concreto. En la figura 1.40 se muestra una arquitectura de los datos. Log Schema usa un formato compatible con Apache Avro. El desarrollador define cómo quiere que se envíe y almacenen los datos mediante estos esquemas, lo que hace a la

plataforma muy flexible, facilitando la integración entre los diferentes servicios.

Entre el servidor y los endpoints se pueden usar diferentes tecnologías o protocolos, como TCP, HTTP, XMPP, CoAP o MQTT.

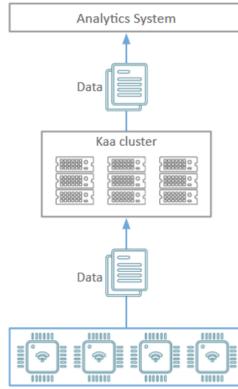


Figura 1.40: Arquitectura de los esquemas de datos

```

{
  "type": "record",
  "name": "LogData",
  "namespace": "org kaaproject kaa schema sample logging",
  "fields": [
    {
      "name": "level",
      "type": {
        "type": "enum",
        "name": "Level",
        "symbols": [
          "DEBUG",
          "ERROR",
          "FATAL",
          "INFO",
          "TRACE",
          "WARN"
        ]
      }
    },
    {
      "name": "tag",
      "type": "string"
    },
    {
      "name": "message",
      "type": "string"
    }
  ]
}
  
```

Figura 1.41: Ejemplo de Log Schema

1.15.1 Qué ofrece Kaa

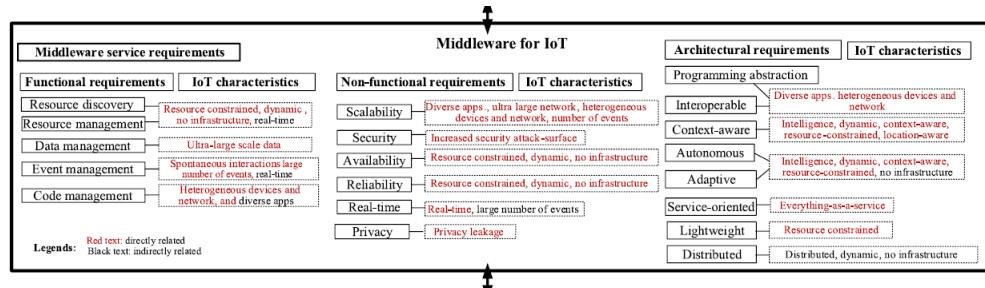


Figura 1.42: Requisitos de un middleware IoT.

Descubrimiento de recursos

Kaa ofrece un sistema de eventos que permite enviarlos tanto a los diferentes endpoints como al servidor. Al igual que el almacenamiento de datos, los eventos se definen mediante esquemas. La siguiente figura muestra un diagrama de cómo se generan y procesan los eventos. Estos eventos se pueden usar para avisar al resto de endpoints de la aparición de un nuevo recurso (endpoint).

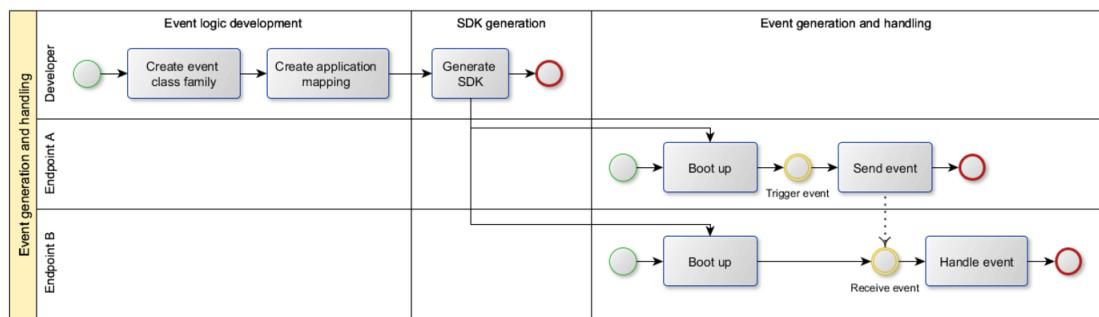


Figura 1.43: Diagrama de eventos en Kaa

cos, entregando los mensajes a los endpoints suscritos. Al igual que el almacenamiento de datos, las notificaciones se definen mediante esquemas. Para el envío de mensajes se usan grupos y perfiles endpoints.

Control de recursos

En IoT se busca que los recursos puedan ser controlados para que, por ejemplo, se pueda reaccionar ante un fallo de éstos. Kaa no ofrece un control de recursos como tal, pero haciendo uso de los eventos y notificaciones se puede conseguir tal propósito. Por ejemplo, una solución sencilla podría ser que un endpoint guarde cada 5/10 min un valor característico, como su timestamp y, mediante un sistema de análisis, analizar los datos y, en caso de fallo, enviar un evento o notificación a un endpoint en concreto, que será el encargado de reaccionar ante tal efecto. Por ejemplo, avisando a un nodo de monitorización para que intente restablecer el endpoint caído.

Otro ejemplo podría ser implementar en un SDK una función 'ping' a la que, haciendo uso del sistema de eventos de kaa, tengan que responder todos los endpoints en un determinado tiempo, si no lo hacen se podría considerar como endpoint caído y actuar ante tal efecto.

Por tanto, el sistema de eventos y notificaciones es un sistema muy completo con el que se pueden realizar muchas funciones. Toda funcionalidad tendrá que ser implementada en los endpoints, donde podrán ser conectados a servidores para realizar funciones mas complejas. El motivo por el cual se usa Kaa es, sin duda, su robustez y heterogeneidad con distintas plataformas así como su facilidad de uso.

...(en un proyecto de kaa en la salud habla de control y configuración de los dispositivos de forma remota. Se podría conseguir mediante eventos???)

Control de datos

Los datos son la principal característica de las aplicaciones IoT, cuando se habla de datos se hace referencia principalmente a los recopilados por los sensores. Un procesamiento, almacenamiento así como una monitorización o filtrado de datos es fundamental para poder desarrollar una aplicación IoT.

Kaa, a diferencia de otras plataformas comerciales, no ofrece un procesamiento o visualización de datos. En kaa tan solo se almacenan los datos provenientes de senso-

res. A pesar de que no ofrezca herramientas de análisis de datos como tal, se pueden usar herramientas externas.

Con estas herramientas se recopilan los datos almacenados en Kaa de manera que sean estas herramientas las que gestionen los datos. Para comunicarse con los EndPoints se usan las notificaciones Kaa mediante su API. Más adelante se verá un ejemplo práctico sobre esto, en el que se ha usado Apache Zeppelin como herramienta para el análisis de datos y Cassandra como base de datos de Kaa. Una vez analizados, si supera un cierto umbral, se envía una notificación a Kaa para que encienda un LED en la raspberry. En la figura -- se puede ver un esquema de uso con Apache Storm.

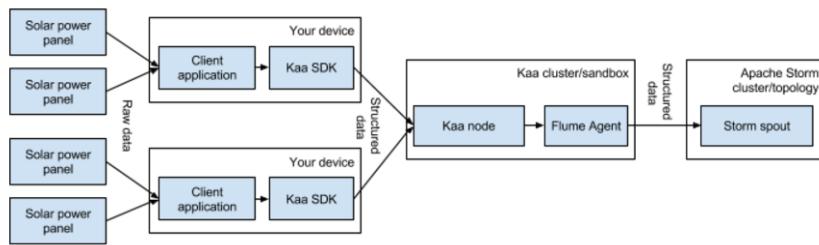


Figura 1.44: Esquema de uso para la monitorización de datos con Apache Storm

Escalabilidad

La plataforma Kaa ha sido diseñada para ser escalable horizontalmente. Se pueden añadir miles de endpoints al mismo nivel, es decir no puede haber varios niveles en la jerarquía. Para poder agrupar distintos endpoints se puede hacer uso de los grupos de endpoints (endpoint group).

Tiempo real

//////////Kaa está diseñado para que soporte virtualmente cualquier protocolo de transporte, por ello permite a los desarrolladores usar diferentes protocolos de transporte de datos para diferentes acciones, en el mismo Endpoint. Por defecto se usa HTTP sobre TCP para la comunicación SDK y cluster.//////////
Kaa es un sistema en tiempo real, entendiendo tiempo real como unos ms de retraso. Kaa usa HTTP sobre TCP para la comunicación entre SDK y Cluster, por tanto, por definición es un servicio con retardos, aunque despreciables para el escenario que se plantea.

Disponibilidad

Kaa es un sistema con una alta disponibilidad. Está formado por clusters, donde cada cluster representa un servidor. Estos clusters están interconectados mejorando así la disponibilidad en caso de fallo de alguno de ellos. Para la coordinación entre los diferentes nodos Kaa usa Apache Zookeeper. Además, los datos de los endpoints se almacenan en bases de datos tolerantes a fallos.

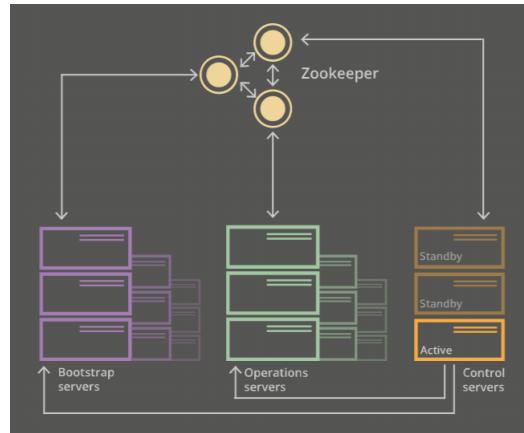


Figura 1.45: Esquema Zookeeper en Kaa

Seguridad y privacidad

Por defecto, en la comunicación entre Kaa y el SDK se usa una encriptación con RSA y AES. Kaa también asegura un almacenamiento de datos seguro en la base de datos así como la autenticación en la plataforma. Kaa usa el hash SHA-1 de la clave pública como identificador del endpoint en el sistema.

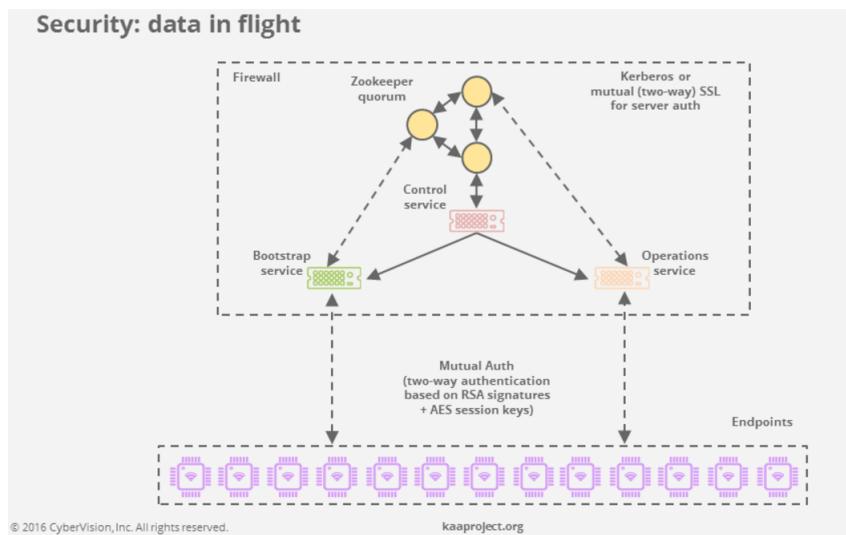


Figura 1.46: Seguridad en Kaa

Popularidad

Kaa es un middleware reciente y con mucho soporte y documentación, lo que facilita su despliegue.

Facilidad para su despliegue

Kaa es un middleware que ofrece facilidades para su desarrollo. Dispone de una versión en una máquina virtual ya preconfigurada con aplicaciones y ejemplos de uso de la plataforma. El uso de SDKs facilita la interoperabilidad de diferentes plataformas sin la intervención del usuario. Con Kaa se consigue que el desarrollador se 'olvide' de la parte del servidor.

Interoperabilidad

La interoperabilidad de distintas plataformas es una de las características de Kaa, en la que una misma aplicación puede ser desarrollada para diferentes plataformas, tan solo descargando el SDK generado por Kaa. Esto hace que sea una plataforma totalmente heterogénea en cuanto a plataformas se refiere. Un endpoint puede ser ejecutado en distintos dispositivos, desde un teléfono móvil a una raspberry o arduino.

Distribuido

Kaa es una plataforma distribuida, que se organiza en clusters donde cada uno de ellos puede ejecutar una determinada funcionalidad que será controlada como un único sistema. Por defecto, Kaa está formada por un único cluster ejecutando el servidor Kaa.

Abstracción de la programación

Kaa proporciona una interfaz al desarrollador, posibilitando una sencillez a la hora de crear las aplicaciones. El desarrollador tan solo tiene que crear los esquemas de datos, pues la plataforma se encarga de generar el código. Además de la interfaz, Kaa proporciona una API que permite realizar todas las funciones que se pueden llevar a cabo a través de la interfaz, mediante llamadas a la misma. Esto es muy útil a la hora de quien tenga que realizar las funciones sea el endpoint automáticamente, lo que hace que Kaa sea una plataforma autónoma.

Basado en servicio

--

Adaptativo

Kaa soporta virtualmente cualquier protocolo de red, endpoint o almacenamiento de datos, lo que hace a la plataforma tolerante a cambios.

Control de eventos

Control de código

Una de las principales características de Kaa es que es Open Source con todo lo que ello implica. El código es totalmente transparente al usuario.

1.16 Uso de la plataforma

1.16.1 Instalación

Para instalar la plataforma hay 2 formas, o bien usar la maquina virtual que ya viene preconfigurada y con unos ejemplos de uso (SandBox) o bien, construir un servidor Kaa propio mediante el código fuente. Todo esto está perfectamente documentado.

1.16.2 Partes de la plataforma

Las características de Kaa son:

Eventos

Kaa proporciona un mecanismo para la entrega de eventos (mensajes) a través de los endpoints. Los eventos pueden ser unicast o multicast, eligiendo así el/los destinatario/s. Los endpoints generan los eventos y los envían al Kaa server, éste se encarga de enviarlos a los endpoints acorde al esquema de eventos, previamente configurado por el desarrollador. La figura 1.44 muestra el proceso de envío y recepción de eventos.

En el siguiente ejemplo se muestra la definición de un esquema de eventos. El formato está basado en [Avro Schema](#).

```
{
  "namespace": "com.company.project",
  "type": "record",
  "classType": "event",
  "name": "SimpleEvent2",
  "fields": [
    { "name": "field1", "type": "int" },
    { "name": "field2", "type": "string" }
  ]
}
```

Cada evento se basa en una clase en concreto (EC), definida en el esquema de eventos. Una EC se identifica mediante su nombre completo, en el ejemplo de arriba sería: "com.company.project.SimpleEvent2". El identificador es único y no puede haber 2 ECs con el mismo nombre en el mismo tenat ///////////////////////////////explicar tenat//////////.

Por otra parte, las clases de eventos (ECs) se agrupan en familias de clases de eventos (ECFs). Un ECF se identifica por su nombre y/o nombre de clase, por lo que no puede haber 2 ECFs con el mismo nombre o nombre de clase en un mismo tenat. El siguiente esquema muestra un ejemplo de definición de ECF:

```
[
  {
    "namespace": "com.company.project.family1",
    "name": "SimpleEvent1",
    "type": "record",
    "classType": "event",
    "fields": []
  },
  {
    "namespace": "com.company.project.family1",
    "name": "SimpleEvent2",
    "type": "record",
    "classType": "event",
    "fields": [
      { "name": "field1", "type": "int" },
      { "name": "field2", "type": "string" }
    ]
  }
]
```

Colección de datos

Los endpoints almacenan datos recopilados ("log") siguiendo una estructura predefinida. El SDK implementa la subida de 'logs' desde los endpoints al servidor. El servidor lo almacena en bases de datos. En el siguiente ejemplo se muestra un simple esquema de almacenamiento de datos (log schemas), compatible con [Avro Schema](#):

```
{
```

```
"name": "LogData",
"namespace": "org.kaaproject.sample",
"type": "record",
"fields": [
    {
        "name": "tag",
        "type": "string"
    },
    {
        "name": "message",
        "type": "string"
    }
]
```

La especificación de la base de datos se hace mediante log appenders usando la interfaz de usuario.

Perfiles y grupos

Notificaciones

Kaa dispone de un sistema de notificaciones para la entrega de mensajes desde el clúster Kaa hasta los endpoints. La estructura de los datos se define en el esquema de notificaciones, configurado en el servidor y desplegado dentro de los endpoints. Las notificaciones a tópicos de forma que para recibir una notificación, el endpoint tiene que suscribirse a uno o varios tópicos. También, se puede asignar un tópico a todo un grupo de endpoints. El envío de notificaciones se puede hacer mediante la interfaz de usuario o mediante una llamada a la API.

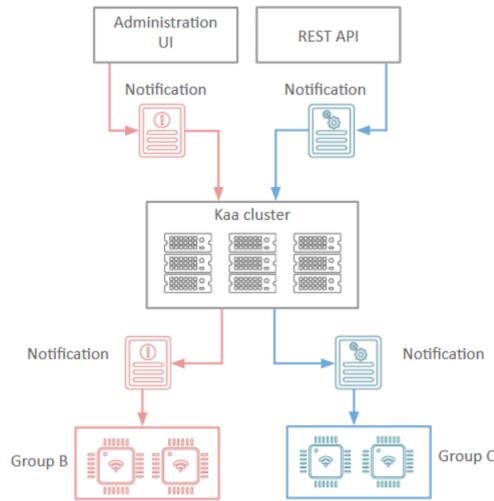


Figura 1.47: Notificaciones en Kaa

Distribución de datos

La distribución de datos es una de las principales características de Kaa ya que los desarrolladores pueden definir cualquier tipo de datos mediante los esquemas de Kaa.

Abstracción en la capa de transporte

Kaa ha sido diseñada para soportar virtualmente cualquier protocolo de transporte de datos. Además se pueden usar diferentes protocolos en un mismo endpoint, por ejemplo las notificaciones mediante SMS y la configuración y datos mediante TCP. //////////explicar virtualmente e incluirlo si consigo usar MQTT como protocolo de transporte//////////

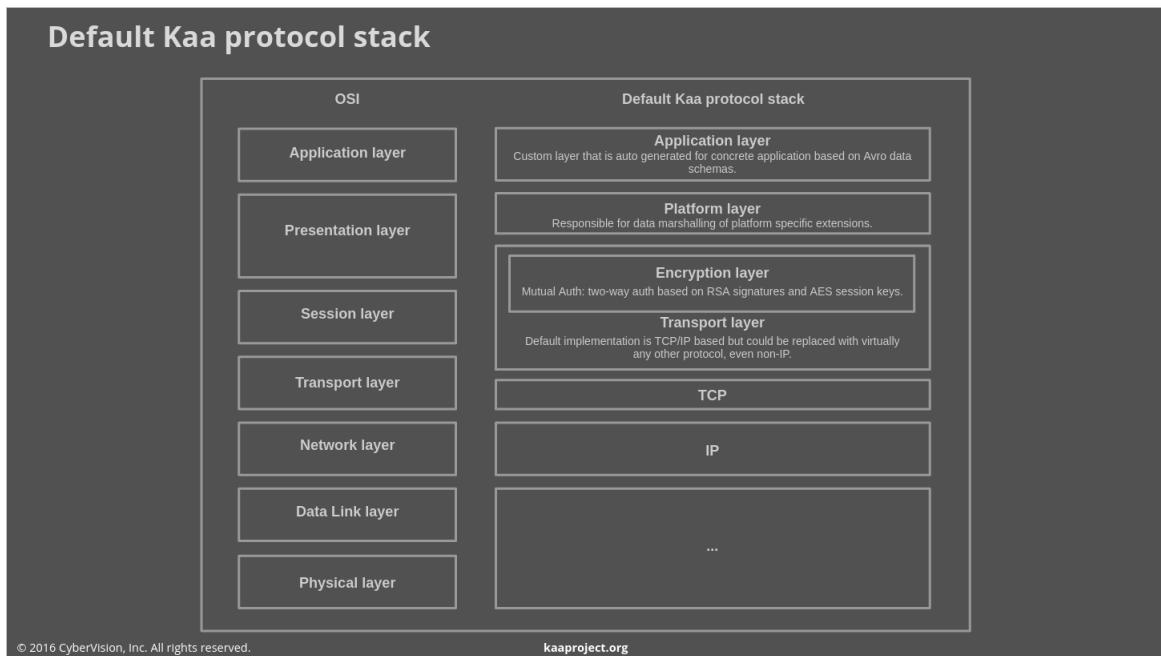


Figura 1.48: Pila de protocolos de Kaa

1.17 Casos prácticos

1.17.1 Encender un LED en la raspberry usando Kaa

Analizar datos con Apache Zeppelin

1.18 Problemas encontrados