ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Integración de una plataforma para la gestión de contexto utilizando habilitadores genéricos FIWARE para la Internet de las Cosas

Integration of a platform for context management through the use of FIWARE Generic Enablers for the Internet Of Things

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Andoni Abascal Armentia

Octubre - 2015

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Andoni Abascal Armentia
Director del TFG: Luis Sánchez González
Título: "Integración de una plataforma para la gestión de contexto utilizando habilitadores genéricos FIWARE para la Internet de las Cosas"
Title: "Platform integration for context management through the use of FIWARE Generic Enablers for the Internet of Things"

Presentado a examen el día: 28 de Octubre de 2015

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Cobo García, Adolfo
Secretario (Apellidos, Nombre): Sánchez González, Luis
Vocal (Apellidos, Nombre): García Gutiérrez, Alberto Eloy

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

En primer lugar, agradecer a mis padres Felipe y Mila y a mi hermana Alazne todo el apoyo mostrado a lo largo de estos años. Pero sobre todo gracias por la educación y valores que me han inculcado.

Mención especial merece Patri, siempre tiene palabras de ánimo, una sonrisa y actitud positiva para afrontar cualquier situación. Gracias por apoyarme, por confiar en mí, y sobre todo por cuidarme.

No quiero olvidarme de los compañeros y la gente más cercana con la que he compartido momentos a lo largo de esta etapa. Gracias por hacer los duros días de estudio más amenos.

Por último, agradecer a todos aquellos profesores de la Universidad de Cantabria que me han dado clase por haber contribuido en mayor o menor medida a formarme como Ingeniero. Especial mención merece el grupo de Ingeniería de Telemática, y sobre todo Luis Sánchez, que sin su ayuda y conocimientos no hubiese sido posible realizar este Trabajo de Fin de Grado.

Muchas Gracias a todos.

Resumen

La comunicación y conexión de los diferentes objetos que nos rodean es uno de los principales objetivos que actualmente ocupan a la industria tecnológica. Esto es lo que hoy se conoce como Internet de las Cosas (IoT por su siglas en inglés). Una consecuencia directa de esta tendencia es que una gran cantidad de datos deberán ser gestionados y procesados con el objetivo de poder sintetizar y seleccionar la información requerida por un usuario o sistema que quiera hacer uso de esa información.

Este escenario plantea la necesidad de poder acceder a los datos de una forma sencilla sin necesidad de conocer el protocolo nativo que se emplea para la recolección de esos datos o donde se encuentran. Esta necesidad se ve agravada por la heterogeneidad que se da en este tipo de entornos así como por la simplicidad de los objetos que generan estos datos, incapaces de soportar complejos protocolos. Sería deseable, por tanto, ser capaz de obtener de manera homogénea y agnóstica a la complejidad subyacente la información necesaria en el momento que se necesite o solicite.

Es por ello que resulta obligatorio en el desarrollo de cualquier escenario de estas características contar con un intermediario entre los productores de contexto (objetos, sensores, etc.) y las aplicaciones consumidoras de contexto.

En este proyecto se ha abordado esta problemática integrando un sistema de gestión de recursos e información de contexto para la IoT desarrollado a partir de un conjunto de componentes disponibles dentro de la plataforma FIWARE.

FIWARE es una plataforma desarrollada en el marco de una iniciativa "público-privada", co-financiada por la Comisión Europea y las principales compañías europeas en el campo de las tecnologías de información y las comunicaciones cuyo objetivo es promover la innovación en la industria IT y crear los nuevos estándares de las tecnologías venideras.

La integración del gestor de contexto y las herramientas afines descritas en este documento permitirá poder organizar y almacenar de manera eficaz toda la información que puedan generar los productores de contexto en cualquier dominio de aplicación. Igualmente importante, este gestor permitirá a los usuarios del mismo obtener la información de contexto que les sea relevante en el momento que la deseen o necesiten. Todo ello incorporando herramientas autenticación y control de acceso para poder elegir a quién y para qué se le da acceso a un determinado cliente.

Por último, durante el desarrollo del trabajo se ha realizado una exhaustiva campaña de verificación de funcionalidades, con el objetivo de garantizar el correcto comportamiento del sistema integrado, que también ha sido incorporada a este documento.

Abstract

The communication and connectivity of the different objects that surround us is one of the main objectives that are being addressed by the technological industry nowadays. Today, this is known as the Internet Of Things (IoT). A direct consequence of this concept is that a huge quantity of data must be managed and processed so as to synthesize and select the information required by a user or system that intends to make use of it.

This scenario poses the need of being able to access the data in an easy way, without need of knowing the native protocol being used for collecting this data, or its whereabouts. This need is even more challenging taking into account the heterogeneity given in this kind of environments, as well as the simplicity of the objects that generate that data, unable to cope with complex protocols. Therefore, it would be desirable being able to obtain the necessary information, in a homogeneous and way agnostic to the underlying complexity, in the moment that it is needed or requested.

Consequently, in the development of any scenario of these characteristics, it is mandatory to count with an intermediary between the context producers (objects, sensors, etc) and the applications that consume this context information.

These challenges have been tackled in this project by integrating a resource and information context management system for the IoT. This system has been developed by integrating a collection of generic enablers within the FIWARE platform.

FIWARE is a platform developed within the framework of a "public-private partnership" co-financed by the European Commission and the leading european companies in the Information, Technology and Communications field, whose objective is to promote innovation in the IT industry, and create the new standards for the upcoming technologies.

The integration of the context broker and the related tools described in this document, will allow its user to efficiently organize and store all the information that might be generated by the system's context producers. Equally important, this manager will allow its users to obtain the relevant context information as soon as it is requested or needed. All of it incorporating the authentication tools and access control mechanisms necessary to discriminate which content a given user can be granted access to.

Last but not least, in addition to the integration of the aforementioned generic enablers, a thorough verification campaign has been carried out. The main objetive of this campaign has been the validation of the integrated context management platform expected behavior. The procedure and results of this validation has been also presented in this document.

Índice de Figuras

Acrónimos

API	Interfaz de Programación de Aplicaciones
CB	Context Broker
CEP	Procesamientos de Eventos Complejos
CPU	Unidad Central de Procesamiento
DB	Base de Datos
DNS	Sistema de Nombres de Dominio
GE	Habilitadores Genéricos
HDFS	Sistemas de Archivos Distribuidos en Aro
HTTP	Protocolo de Transferencia de Hipertexto
IdM	Gestor de Identidad
IMEI	Sistema Internacional para la Identidad de Equipos Móviles
IoT	Internet de las Cosas
IP	Protocolo de Internet
JSON	Notaciones de Objetos JavaScript
OMA	Alianza Móvil Abierta
PAP	Punto de Administración de Políticas
PDP	Punto de Decisión de Políticas
PEP	Punto de Aplicación de Políticas
PPP	Programa de Asociación Privada
RFC	Petición de Comentario
SaaS	Software como Servicio
SAML	Lenguaje de Marcado para Confirmaciones de Seguridad
SSO	Inicio de Sesión Única
TCP	Protocolo de Control de Transmisión
TIC	Tecnologías de la Información y la Comunicación
URI	Identificador de Recursos Uniforme
URL	Localizadores Uniformes de Recursos
UX	Experiencias Avanzadas de Usuario
XACML	Control de Acceso del lenguaje de marcas Extensibles
XML	Lenguaje de Marcas Extensible

Índice

1. Introducción

1.1. Motivación

La comunicación y conexión de los diferentes objetos que nos rodean es uno de los principales objetivos que hoy en día ocupan a la industria tecnológica. Esto es lo que hoy se conoce como Internet de las cosas (IoT)(ver Figura 1). Parece indudable que cada vez será mayor la penetración que tendrán en nuestro entorno y por tanto en nuestra vida las tecnologías surgidas de este concepto.

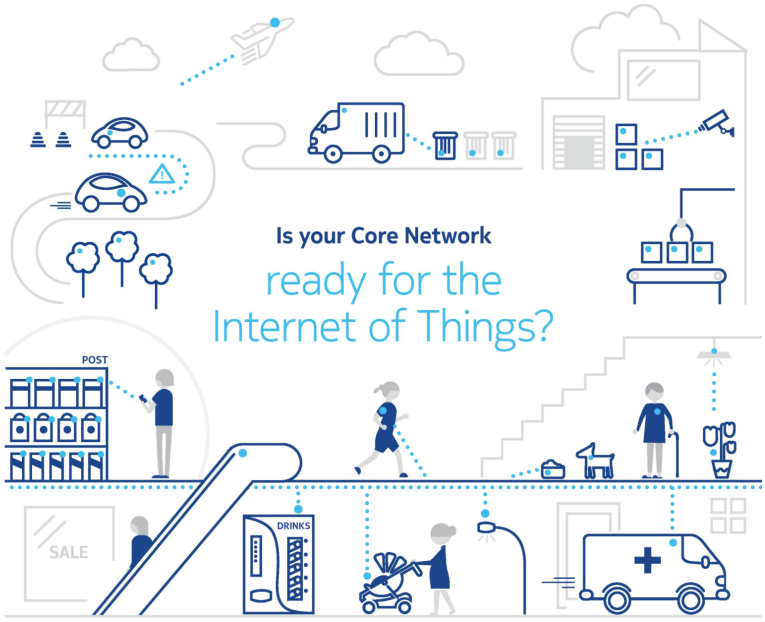

Figura 1 - Representación IoT

Una consecuencia directa de esta tendencia es que una gran cantidad de datos deberán ser gestionados y procesados con el objetivo de poder sintetizar y seleccionar la información requerida por un usuario o sistema que quiera hacer uso de esa información. Una gestión adecuada de esos datos pretende que se puedan desarrollar servicios y aplicaciones capaces de generar contenidos de gran valía que utilicen esta información de manera inteligente. Por ejemplo, son cada vez mas comunes proyectos sobre lugares inteligentes como oficinas, casas o incluso ciudades, en los cuales una gran cantidad de objetos están interconectados.

Este escenario plantea la necesidad de poder acceder a los datos de una forma sencilla sin necesidad de conocer el protocolo nativo que se emplea para la recolección de esos datos o donde se encuentran. Esta necesidad se ve agravada por la heterogeneidad que se da en este tipo de entornos así como por la simplicidad de los objetos que generan estos datos, incapaces de soportar complejos protocolos. Sería deseable, por tanto, ser capaz de obtener de manera homogénea y agnóstica a la complejidad subyacente la información necesaria en el momento que se necesite o solicite.

Es por ello que resulta obligatorio en el desarrollo de cualquier escenario de estas características contar con un intermediario entre los productores de contexto (objetos, sensores, etc.) y las aplicaciones consumidoras de contexto.

En este proyecto se ha abordado esta problemática integrando un sistema de gestión de recursos e información de contexto para la IoT desarrollado a partir de un conjunto de componentes dentro de la plataforma FIWARE.

FIWARE es una plataforma desarrollada en el marco de una iniciativa "público-privada" y co-financiada por la Comisión Europea y los principales actores europeos en el campo de las tecnologías de información y las comunicaciones cuyo objetivo es promover la innovación en la industria IT y crear los nuevos estándares de las tecnologías venideras. En este sentido, si se necesita un intermediario en un escenario IoT, que mejor gestor que aquel desarrollado con los futuros estándares.

La integración del gestor de contexto y las herramientas afines descritas en este documento permitirá poder organizar y almacenar de manera eficaz a toda la información que puedan generar los productores de contexto su sistema. Igualmente importante, este gestor permitirá a los usuarios del mismo obtener la información de contexto que les sea relevante en el momento que la deseen o necesiten. Todo ello incorporando herramientas autenticación y control de acceso para poder elegir a quien y para que se le da acceso a un determinado cliente.

Este proyecto está dirigido a todo aquel que desee implementar de una forma guiada y descriptiva un gestor de contexto y sus respectivas herramientas de seguridad a través de los componentes ofrecidos por la plataforma FIWARE. Además persigue facilitar la compresión de una forma mas clara lo que se puede llegar a conseguir con alguno de los habilitadores genéricos desarrollados por FIWARE.

1.2. Objetivos

El principal objetivo de este trabajo es la integración de una plataforma con la que gestionar la interacción entre los productores y los consumidores de contexto en el marco de la Internet de las cosas. Además, esta plataforma será capaz de soportar el control y la gestión de usuarios (creación de credenciales, verificación, etc.) con acceso a la herramienta para diferentes operaciones. La consecución de este objetivo implica el desarrollo de una serie de tareas que se han llevado acabo durante la realización de este trabajo de Fin de Grado.

La primera de estas tareas ha sido la de conocer la plataforma FIWARE y la estructura documental que facilita. La plataforma se divide en una serie de pilares básicos donde encuadrar cada uno de sus proyectos. Principalmente se ha trabajado con el Catálogo FIWARE. Este catálogo aglutina los "Generic Enablers" (Habilitadores Genéricos - GE) ya creados. Como se describirá en el capítulo 2, los GE son los componentes funcionales definidos por FIWARE para la realización de los escenarios de la Internet del Futuro.

En este proyecto se hará uso de tres habilitadores genéricos que son: Context Broker, Punto de Aplicación de Políticas Proxy (PEP Proxy) y Gestor de Identidad (IdM). Para cada uno de estos GE existirá, dentro del Catálogo FIWARE, una implementación software de referencia que son las que se han integrado.

Las diferentes etapas para la integración de estos habilitadores se debe llevar acabo sobre un sistema operativo de código abierto ya que así lo requieren los habilitadores genéricos que se instalan. Es por ello que, antes de poder abordar la integración de los GE, ha sido necesario familiarizarse con este tipo de sistemas operativos para poder comprender los diferentes pasos que se han completado. Ej. las instalaciones, el control de dependencias o la modificación de los archivos de configuración, para adaptar estos habilitadores a las necesidades de nuestro sistema.

Al mismo tiempo se han estudiado y comprendiendo las diferentes tecnologías aplicadas a nuestro trabajo. En este sentido y por nombrar las mas relevantes, se ha trabajado con tecnologías como: NGSI9, NGSI10, RESTful web servicies, XML, JSON, OAuth2, etc.

Ha sido necesaria una buena comprensión de lo descrito anteriormente para poder desarrollar una guía clara y práctica tanto de la instalación como del uso de estos habilitadores. Este objetivo hubiese resultado muy complicado de no haberse entendido bien todos aquellos aspectos implicados en el proyecto.

Además, se han testado las funcionalidades que ofrecen los habilitadores instalados, para así comprobar su correcto funcionamiento y poder crear una guía con indicaciones claras para que el usuario pueda hacer un buen uso de la herramienta.

Otro objetivo que se ha alcanzado con el proyecto es el desarrollo de un sistema de seguridad capaz de proteger el acceso a la plataforma de gestión del contexto. Se han adquirido los conocimientos necesarios en la gestión e implementación de las herramientas de seguridad y validación de usuarios como son IdM y PEP Proxy. Estas herramientas desarrolladas por FIWARE llevan acabo un papel fundamental para hacer más atractivo el gestor de contexto integrado en este trabajo.

En este documento se hará énfasis en aquellas partes de nuestro desarrollo que no coinciden completamente con lo descrito acerca de ellas en la documentación que facilita FIWARE o como se realizaron diferentes pasos a los indicados para completar la instalación, ya que nuestro sistema así lo necesitó. Esto se debe a que FIWARE da las indicaciones globales para poder implantar sus herramientas en diferentes sistemas con diferentes versiones. En este proyecto se ha sido más específico ya que sólo se trata de una única versión de cada habilitador y de un solo sistema operativo por lo que las instrucciones que podremos dar serán más claras.

Por último, para así poder facilitar el acceso a las funcionalidad de la plataforma integrada, se ha generado una máquina virtual con todos los GE integrados.

Como valor añadido genérico, durante el transcurso del proyecto se ha tenido que recurrir al empleo de foros técnicos, por lo que uno de los objetivos intrínsecos en este trabajo ha sido también el de aprender a hacer un buen uso de este tipo de recurso.

1.3. Resumen Ejecutivo

La siguiente memoria esta dividida en seis capítulos que se describen a continuación.

En el segundo capítulo se presenta el análisis del entorno tecnológico sobre el que se ha desarrollado este Trabajo de Fin de Grado. Comenzando por una breve introducción del entorno creado por el proyecto FIWARE, este capítulo describe las funcionalidades genéricas de los tres GE integrados a lo largo de este Trabajo de Fin de Grado. Además, se introducen brevemente las implementaciones de referencia para cada uno de estos GE que están disponibles en el catálogo FIWARE.

Tanto la descripción de la arquitectura del sistema integrado en este trabajo como el resumen detallado de los pasos dados para instalar y configurar las herramientas Context Broker, PEP Proxy e IdM de FIWARE, se encuentran en el tercer capítulo.

A continuación, en el cuarto capítulo, se presentan las pruebas que se han llevado a cabo para demostrar el correcto funcionamiento del sistema integrado y permitir el análisis de las distintas funcionalidad que proveen cada uno de los GEs por separado y también de manera integrada.

En el quinto capitulo se describe como se ha generado un clon de la máquina virtual que alberga el sistema para así poder instalar la herramienta en un sistema operativo libre sin necesidad de instalar cada herramienta de forma independiente.

Para concluir, en el sexto y último capítulo se realiza un análisis tanto de los aspectos más interesantes abordados durante la realización del trabajo así como de las posibles líneas futuras que se abren tras su conclusión.

2. Marco teórico del trabajo

2.1. FIWARE

2.1.1 ¿Qué es FIWARE?

La Comisión Europea y un conjunto de empresas tecnológicas emprendieron en 2011 una ambiciosa colaboración para situar al viejo continente en una posición de liderazgo en el ámbito de la Internet del Futuro. Así surge la Future Internet PPP.

Esta colaboración público-privada persigue el objetivo último de que Europa se encuentre en la mejor posición para rentabilizar los efectos de la revolución digital. Dentro de este programa, en su primera fase, nace FIWARE. Una infraestructura abierta e innovadora, que se utiliza para crear y desplegar de manera rentable, aplicaciones y servicios a una escala nunca vista anteriormente.

Con ello el gobierno comunitario quería tener una alternativa europea a las plataformas que hasta el momento dominaban el mercado de contenidos digitales: Amazon y Google. Facilitando así, alternativas para la aparición de múltiples proveedores y evitando que los emprendedores queden atados a un proveedor concreto. Para ello, este proyecto contó con un presupuesto inicial de 66 M€ durante un periodo de 3 años de los cuales 41 millones fueron financiados por la Unión Europea, participaron empresas europeas de telecomunicaciones (Telefónica, coordinadora del proyecto, Orange, Deutsche Telecom, British Telecom, Telecom Italia) otras empresas relevantes del sector TIC (SAP, IBM, Nokia Siemens, Thales, Atos, Ericsson, Alcatel-Lucent) y universidades y centros de investigación.

Una de sus virtudes es la separación que llevaron acabo en la cadena de valor de las aplicaciones, de forma que la plataforma, el desarrollo de los servicios, su despliegue, etc. puedan ser proporcionados por entidades diferentes, ayudando de esta forma a su expansión y mejorando la competitividad.

La conexión con la Internet de las Cosas, el almacenamiento, acceso, procesado, publicación y análisis tanto de contenidos multimedia como de datos a gran escala, la co-creación de aplicaciones y contenidos, el desarrollo de interfaces de usuario avanzadas con capacidades 3D y de realidad aumentada, son ejemplos de problemáticas que resultan mas sencillas de abordar utilizando la plataforma FIWARE [20].

2.1.2. ¿Como se estructura FIWARE?

La iniciativa se basa en los siguientes pilares:

FIWARE: La plataforma FIWARE, donde se encuentra el Catálogo FIWARE, proporciona un conjunto bastante simple pero potente de interfaces de programación de aplicaciones (APIs) que facilitan el desarrollo de aplicaciones inteligentes en varios sectores verticales. Las especificaciones de estas APIs son públicas y libres. Además, una implementación de referencia de código abierto de cada una de las APIs FIWARE está disponible de forma pública para que puedan surgir proveedores en el mercado con una propuesta de bajo coste.

Está basado en un conjunto de elementos denominados “Generic Enablers”, que son esencialmente programas de código abierto con funcionalidades de la Internet del Futuro. Los “Enablers” se pueden separar en los apartados: Elementos de Cloud Hosting, que proporcionan los recursos básicos de computación, de red y de almacenamiento. Elementos de manejo de

datos, que proporcionan herramientas para análisis tipo “Big Data”. Módulos para integración de aplicaciones, que proporciona elementos para integrar aplicaciones, permitir su publicación, etc… Elementos para el manejo de la Internet de las Cosas. Módulos para el acceso a la red de comunicaciones y control de terminales y Elementos para dotar de seguridad y privacidad a las aplicaciones.

FIWARE Lab: FI-LAB es una instancia gratuita de la plataforma FIWARE que está disponible para la experimentación y la prueba de las tecnologías FIWARE. Los desarrolladores pueden realizar experimentos y comenzar a ampliar el ecosistema de negocio basado en el uso de los distintos módulos de FIWARE. Explotando datos abiertos publicados por ciudades y otras organizaciones. A muy pequeña escala y con un ámbito de aplicación mucho más reducido, este trabajo tiene como objetivo la integración de un FI-LAB personalizado.

FIWARE Ops: Es una colección de herramientas que facilita el despliegue, configuración y operación de instancias por los proveedores de la plataforma. Está diseñado para ayudar a expandir la infraestructura asociada a una instancia y permite la cooperación de múltiples proveedores de la plataforma. FIWARE Ops es la herramienta que se utiliza para construir, operar y ampliar FI-LAB.

FIWARE Accelerate: Este Programa tiene como objetivo promover la asimilación de tecnologías FIWARE entre los integradores de soluciones y desarrolladores de aplicaciones, con especial atención a las PYMEs y empresas de nueva creación. Vinculado a este programa, la UE puso en marcha una ambiciosa campaña en septiembre de 2014 con una inversión de más de 80 millones de euros para apoyar a las PYMEs y los empresarios que desarrollan aplicaciones innovadoras basadas en FIWARE.

Para poder optar a esa financiación el proyecto debía de desarrollarse en uno de los siguientes ámbitos: ciudades inteligentes, salud electrónica, transporte, energía y medio ambiente, agroalimentaria, medios de comunicación y conexión, aprendizaje social, manufactura y logística.

FIWARE Mundus: A pesar de que FIWARE nació en Europa, ha sido diseñada con una ambición global, por lo que los beneficios pueden extenderse a otras regiones. El programa Fiware Mundus está diseñado para dar cobertura a este esfuerzo.

2.1.3. El Catálogo FIWARE

El Catálogo FIWARE es la componente principal de FIWARE, contiene una amplia biblioteca de Generic Enablers. Mediante la combinación de ellos, se puede comenzar a desarrollar una aplicación inmediata. Basta con echar un vistazo a las guías y aprender que hace cada habilitador y la forma en que se pueden asociar a distintas aplicaciones.

El Catálogo FIWARE es el repositorio de los GE que forman la plataforma. Aparte de los GE, también se pueden encontrar herramientas y guías prácticas que ayuden a desarrollar aplicaciones de la Internet del Futuro.

Los GE se estructuran en los siguientes capítulos:

Capítulo 1. **Desarrollo de aplicaciones sensibles al contexto:** Si se quiere que una aplicación sea "inteligente", primero se debe hacer a la aplicación "consciente". FIWARE proporciona medios para producir, recopilar, publicar y consumir información de contexto a gran escala y explotarla para transformar la aplicación en una aplicación inteligente.

La información de contexto se representa a través de valores asignados a los atributos que caracterizan a las entidades pertinentes a la aplicación. El Context Broker GE (ver Figura 2)

permite el manejo de información de contexto a gran escala para lo cual ofrece interfaces estándar basados en tecnología de RESTful web services.

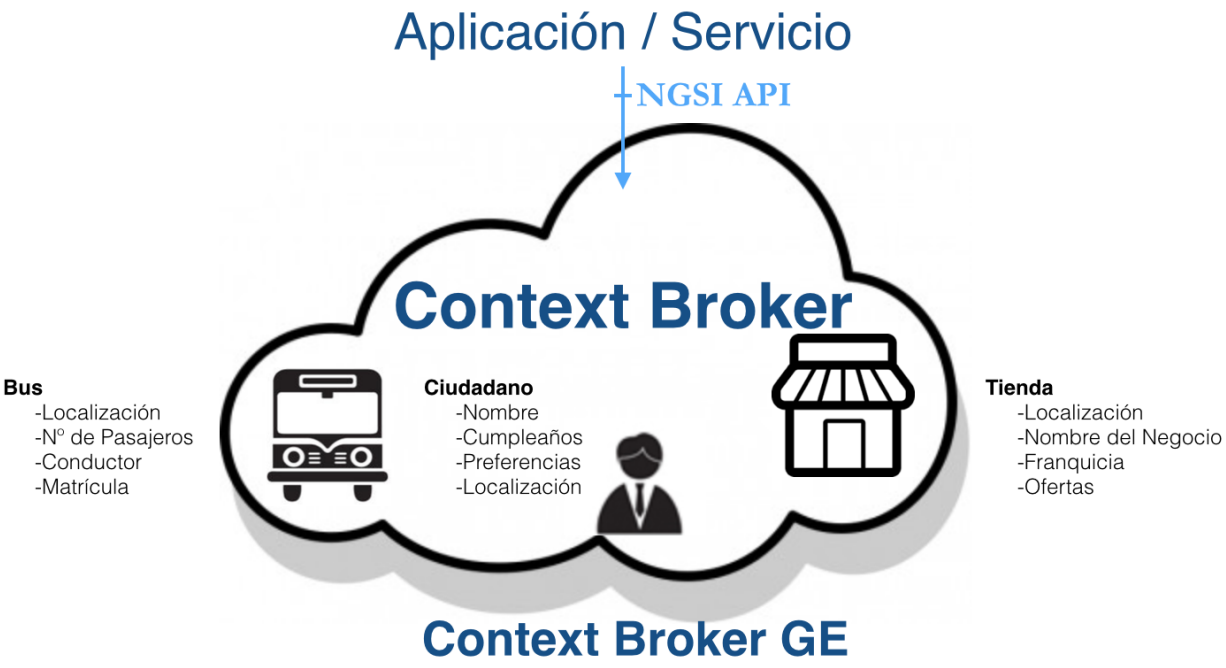

Figura 2 - Esquema Context Broker

Capítulo 2. **Conexión a la Internet de las Cosas (IoT):** La conexión de "objetos" o "cosas" implica la necesidad de superar una serie de problemas que surgen en las diferentes capas del modelo de comunicación. El uso de información de contexto o actuar sobre los "objetos" requiere la interacción con un entorno heterogéneo de dispositivos que ejecutan diferentes protocolos (debido a la falta de estándares aceptados a nivel mundial), se encuentran dispersos y son accesibles a través de múltiples tecnologías inalámbricas.

Estos dispositivos presentan un sin fin de particularidades por lo que no es posible proporcionar una solución única. Sus recursos son limitados y no pueden usar pilas de protocolos estándar completas; no pueden transmitir información con demasiada frecuencia debido al consumo de la batería; no siempre son accesibles ya que están conectados a través de redes inalámbricas heterogéneas; sus protocolos de comunicación son demasiado específicos o por otra gran cantidad de motivos es difícil de encontrar un despliegue global. Para ello FIWARE proporciona GE´s con la intención de hacer más sencillo este proceso.

Capítulo 3. **Procesamiento de eventos de contexto en tiempo real:** Es posible que se desee realizar algún procesamiento de la información de contexto disponible. A modo de ejemplo, es posible que se desee detectar automáticamente patrones que requieren activación de alguna acción o activado de cierta alarma. Se puede utilizar el procesamiento de eventos complejos (CEP) FIWARE como parte de la arquitectura de las aplicaciones para este propósito. El GE CEP permite detectar patrones anteriores de contexto. De esta manera, en vez de reaccionar a una única información de contexto se puede identificar y reaccionar a los patrones de contexto de varias entidades. El CEP recibe información de contexto como los eventos de entrada y genera observaciones que a veces se convierten en actuaciones de salida. El CEP GE analiza los datos de eventos de entrada en tiempo real, genera una visión inmediata y permite una respuesta instantánea a las condiciones cambiantes. La tecnología y las implementaciones del CEP proporcionan medios para definir y mantener la lógica de procesamiento de eventos de la aplicación de una forma flexible.

Capítulo 4. **Manejo de autorización y control de acceso a APIs:** En FIWARE se ofrecen algunos servicios y herramientas que permiten gestionar la autenticación y autorización en las aplicaciones y servicios de backend. Si se desea administrar la identidad en una aplicación sin desarrollar sus propios mecanismos, se puede ofrecer a los usuarios la posibilidad de acceder a la aplicación utilizando cuentas FIWARE.

Esto es posible gracias al protocolo OAuth2 y Keyrock, el componente de Gestión de Identidades de FIWARE. De la misma manera que es posible conectarse a algunos servicios utilizando cuentas de Twitter o Facebook, los usuarios van a poder utilizar sus cuentas FIWARE para acceder a los servicios ofrecidos por terceras partes.

Capítulo 5. **Información y publicación de contexto como datos públicos:** Los usuarios (tanto finales como desarrolladores) pueden publicar datos en CKAN, este GE se emplea para la publicación de datos gratuitos. Si por el contrario, se desea publicar datos para su compra se deberá emplear Ws2tore con el fin de establecer políticas de control de acceso y mantener un control sobre los beneficios. FIWARE ofrece dos tipos diferenciados de datos: información de contexto y conjuntos de datos. Cada tipo de contenido debe ser publicado de forma diferente y por ello se puede encontrar dos secciones diferenciadas.

Capítulo 6. **Análisis Big Data sobre históricos de información de contexto:** De manera similar a lo que se ha descrito en la sección publicación de información de contexto como Open Data, el software Cygnus permite almacenar todos los datos seleccionados publicados en el Context Broker en un almacenamiento basado en HDFS. Esto permite tener una base de datos histórica a largo plazo con información de contexto que puede ser utilizada para un análisis posterior. Una vez que el contexto de datos se ha almacenado, es posible utilizar los datos del GE para procesarlos. Por supuesto, también es posible procesar en los Big Data GEs otros grandes conjuntos de datos, ya sea por sí solos o en combinación con información de contexto.

Capítulo 7. **Creación de Dashboard para aplicaciones:** Wirecloud es una plataforma de mashup web dirigida a capacitar a sus usuarios sin conocimientos de programación, en crear de manera sencilla paneles de control para aplicaciones con Generic Enablers.

Capítulo 8. **Procesamiento en tiempo real:** Una aplicación también puede necesitar integrar información multimedia. La información multimedia comprende datos de audio y vídeo, que se utilizan normalmente para el intercambio de información compleja (es decir, videoconferencia, intercambio de vídeo-clip, multimedia de mensajería instantánea, etc.) Además, en los últimos años, cámaras y micrófonos se utilizan a menudo como sensores avanzados que, combinado con la visión por ordenador y otras técnicas de análisis, puede generar una rica información útil en diferentes áreas, incluyendo la sanidad electrónica, ciudades inteligentes, la seguridad, entretenimiento, etc.

Capítulo 9. **Proporcionar experiencias de usuario avanzada** (**UX**)**:** Es posible que se desee incorporar características avanzadas en una interfaz de usuario basada en web, por ejemplo, la realidad aumentada o las características de visualización 3D. Se recomienda entonces que se eche un vistazo a los GE FIWARE que se han definido en el capítulo de interfaces de usuario basados en web avanzada. Estos componentes han sido recientemente incorporados en FIWARE así que también se esta trabajando en su integración con otros GE´s FIWARE.

Capítulo 10. **Alojamiento de la aplicación en el Cloud FIWARE:** Cualquier parte de una aplicación que se ejecute en un centro de datos centralizado se pueden aprovisionar y gestionar utilizando las capacidades FIWARE Cloud. FIWARE Cloud es una plataforma de servicios basada en OpenStack.

2.2. Context Broker GE

2.2.1. Definiciones Previas

A continuación se presentan una serie de definiciones vinculadas a este Generic Enabler. Con ello se pretende establecer un vocabulario que ayude a entender este documento [3]. Para un glosario más extenso de los términos y definiciones consulte el Glosario FIWARE de Términos y Definiciones[1].

Cosa(Thing): Se refiere a la instancia de un objeto físico, organismo vivo, persona o concepto interesante para una persona o aplicación, desde la perspectiva de una instancia FIWARE, ya sea para un área de uso particular o para varias áreas de uso.
Ejemplos de objetos físicos son:
- Edificio, mesa, puente (***clases***)
- El Museo del Prado, la mesa del salón de Pedro, el puente del pueblo de Juan (***instancias***).

Ejemplos de los organismos vivos son :
- Rana, árbol, persona (***clases***)
- La rana verde del jardín, el árbol de roble en frente del Museo del Prado, el Dr. García (**instancias**).

Clase de Cosas (Class of thing): Define el tipo de Cosa en el estilo de programación orientada a objetos: se trata de una construcción que se utiliza como modelo para crear instancias, define los miembros constituyentes que permitan a instancias de clase tener estado y comportamiento. Un ejemplo de la clase de cosa es "persona" , con una instancia que es " Dr. García".

Virtual thing: Es la representación digital de una cosa dentro del componente de habilitación de servicios IoT. Consiste en un conjunto de propiedades que son interesantes para una persona o una aplicación desde la perspectiva de una instancia de FIWARE. Todas las cosas de la misma clase tienen el mismo conjunto de propiedades.

Evento: algo que ha ocurrido o se contempla que sucedió. Cambios en la información de contexto se consideran como eventos.

Dispositivo (Device): Entidad hardware, componentes o sistemas que pueden estar en relación con una cosa o un grupo de cosas. Un dispositivo tiene los medios para medir las propiedades de una cosa y convertirlo en una señal analógica o digital que puede ser leída por un programa, usuario o influir potencialmente en las propiedades de otra cosa o grupo de cosas. En caso de que sólo se puedan medir las propiedades, entonces lo llamamos sensor. En caso de que pueda influir potencialmente en las propiedades de una cosa o grupo de cosas, lo llamamos actuador. Los dispositivos más sofisticados pueden tener un identificador único, capacidades de computación empotradas, así como capacidad de comunicación.

Recurso IoT (IoT Resource): Software que proporciona los medios técnicos para llevar a cabo la detección y/o actuación en el dispositivo. Puede que cada dispositivo tenga su propio recurso o que este sea compartido entre un número de ellos. Las capacidades de actuación de un recurso IoT pueden comprender la configuración de las funciones de gestión y/o aplicaciones del dispositivo, la conectividad, control de acceso, etc. Mientras la detección puede comprender la recolección de fallos, métricas de rendimiento, etc. El recurso por lo general se encuentra alojado en el dispositivo.

[1] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.Glossary.Global

IoT Backend: Proporciona funcionalidades de gestión de dispositivos IoT de dominio específico para las aplicaciones. Componente integrado en las instancias FIWARE.

Pasarela IoT (IoT Gateway): Un dispositivo que, además o en lugar de detectar o accionar movimiento proporciona funcionalidades de conexión de red y conversión de protocolo entre los dispositivos y la IoT Backend. Por lo general se encuentran en la proximidad de los dispositivos que conecta.

Además de estos conceptos, es necesario introducir los elementos y actores básicos en el modelo funcional de un Context Broker. Estos actores son:

Context Broker: es el principal componente de la arquitectura. Funciona como un controlador de datos de contexto y como interfaz entre los actores de la arquitectura. Principalmente el Context Broker tiene que controlar el flujo de contexto entre todos los actores; con el fin de hacer eso, el Context Broker tiene que conocer todos los productores de contexto de la arquitectura; esta función se realiza a través de un proceso de anuncio que se detalla en las siguientes secciones.

Productor de Contexto: Un productor de contexto es un actor que es capaz de generar contexto. El productor de contexto básico, es aquel que actualiza de forma espontánea la información de contexto, sobre uno o mas atributos de acuerdo con su lógica interna.

Proveedor de Contexto: es un tipo especializado de productor de contexto, que proporciona información de contexto bajo demanda, en modo sincrónico; eso significa que el Context Broker o incluso el consumidor de contexto puede invocar al proveedor de contexto con el fin de adquirir información de contexto. Un proveedor de contexto proporciona datos de contexto sólo cuando contribuyen a una ejecución específica. Cada proveedor de contexto registra su disponibilidad y capacidades mediante el envío de los anuncios correspondientes y expone sus interfaces para proporcionar información de contexto al Context Broker y los consumidores de contexto.

Consumidor de Contexto: Un consumidor de contexto es una entidad (por ejemplo, una aplicación) que explota la información de contexto. El consumidor de contexto puede recuperar información de contexto enviando una solicitud al Context Broker o invocando directamente al productor de contexto a través de una interfaz específica. Otra forma para que el consumidor de contexto pueda obtener información es, mediante la suscripción a las actualizaciones de información de contexto que responden a ciertas condiciones (por ejemplo, estar relacionados con cierto conjunto de entidades). El consumidor de contexto se registra a la devolución de información sobre una operación con la suscripción, por lo que el Context Broker notifica al consumidor de contexto en las actualizaciones pertinentes invocando la función de devolución de llamada.

Entidad: Cada intercambio de datos de contexto hace referencia a una entidad especifica, la cual puede ser un grupo de orden complejo de más de una entidad. Una entidad es el objeto al que se hace referencia con los datos de contexto. La entidad está compuesta de dos partes: un tipo y un identificador. Cada proveedor de contexto soporta uno o más tipos de entidad y esta información se publica en el Context Broker durante el proceso de anuncio [18].

Tipo: Es un objeto donde clasificar un conjunto de entidades; por ejemplo, tipos de entidades son usuarios humanos (identificados por nombre de usuario), dispositivos móviles (identificados por IMEI), usuarios móviles (identificados por el número de teléfono) o grupos de otras entidades (identificados por groupID).

Disociación: Es el proceso a través del cual la información se distribuye y se comparte entre consumidores y productores de contexto, para una gestión eficiente es imprescindible tener en cuenta en los sistemas de comunicación la forma de desvinculación que proporcionan. Se admiten varias formas de disociación:

- Disociación Espacial: Las partes que interactúan no necesitan conocerse. Los proveedores publican información a través de un servicio de información del evento y los abonados (consumidores) reciben información de forma indirecta a través de ese servicio. Los proveedores y consumidores por lo general no tienen referencias de los demás y tampoco saben cuántos proveedores o consumidores participan en la interacción.

- Disociación Temporal: Las partes que interactúan no necesitan participar activamente en la interacción al mismo tiempo, es decir, el editor (proveedor) puede publicar información, mientras que el abonado se desconecta y el abonado podría obtener notificación acerca de la disponibilidad de información, mientras que el editor original esta desconectado.

- Disociación por sincronización: Los editores no están bloqueados cuando producen información, y los suscriptores pueden conseguir notificaciones asíncronas de la disponibilidad de información mientras desempeñan alguna otra actividad.

El concepto de disociación es importante para entender como la separación entre la producción y el consumo de información aumenta la escalabilidad, debido a la eliminación de todas las dependencias explícitas entre los participantes que interactúan. La eliminación de estas dependencias reduce fuertemente los requisitos de coordinación entre las diferentes entidades y hace que la infraestructura de comunicación resulte mas adaptada a entornos distribuidos. Esta ventaja se vuelve aun mas beneficiosa cuando existen entidades móviles en el sistema distribuido.

2.2.2. Descripción Funcional del Context Broker

Un Context Broker (CB) permite gestionar todo el ciclo de vida de la información de contexto incluyendo actualizaciones, consultas, registros y suscripciones. Utilizando el Context Broker se pueden registrar los elementos de contexto y gestionarlos a través de actualizaciones y consultas. Además, el usuario de un CB puede suscribirse a información de contexto por lo que cuando se produce alguna condición (por ejemplo, un intervalo de tiempo ha pasado o los elementos de contexto han cambiado) recibe una notificación.

Un gestor como este es imprescindible cuando se quiere desarrollar un escenario de gestión de contexto. En este tipo de escenarios se necesita un componente entre los consumidores de contexto (ej. smartphone) y los productores (ej. sensor). El Context Broker cumple esa funcionalidad intermedia en la arquitectura.

El Context Broker permite la publicación de información de contexto por entidades denominadas productores de contexto. La información publicada por los productores de contexto está disponible para otras entidades denominadas consumidores de contexto. Aplicaciones, dispositivos u otros "habilitadores genéricos" de la plataforma FIWARE pueden desempeñar tanto el rol de productores de contexto como de consumidores.

El CB admite dos formas de publicación/subscripción, Push y Pull. En el caso de un productor de contexto, se denomina Push cuando la información se envía desde el productor de contexto cuando está disponible, sin necesidad de que esa información haya sido requerida por un consumidor. El método Pull es aquel en el que se extrae la información al ser solicitada por el broker. De manera similar se dice emplear el método Push desde el consumidor de contexto cuando el broker es quien decide enviar la información hacia el consumidor (modo subscripción). Cuando el consumidor utiliza el método Pull, es el consumidor quien decide cuando pedir la información (modo de petición activa).

Un principio fundamental en el que se basa un Context Broker es la disociación total entre productores y consumidores de contexto. Esto significa que los productores de contexto publican datos sin saber que, donde y cuando los consumidores de contexto consumirán dicha información, por tanto no necesitan estar conectados entre ellos. Por otro lado, los consumidores

de contexto consumen información sin que esto signifique que saben que un productor de contexto publica un evento en particular. El consumidor se interesa por el evento en si pero no por quien lo generó. Como resultado el Context Broker GE es un excelente puente que permite a aplicaciones externas gestionar eventos del Internet de las Cosas (IoT) de una manera simple, ya que oculta la complejidad de recolección de medidas desde los recursos IoT (sensores).

La Figura 3 muestra la arquitectura lógica del Context Broker, con sus principales componentes y las interacciones con otros actores.

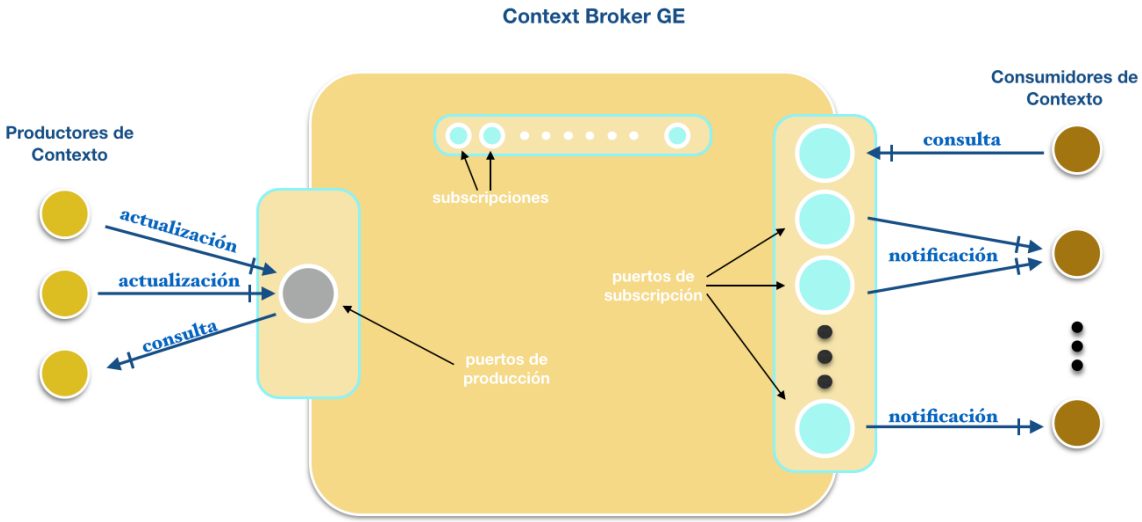

Figura 3 - Arquitectura Lógica del Context Broker GE

2.2.3. Estándares empleados por el Context Broker

El Administrador de configuración GE está basado en los principios de diseño REST. Las tecnologías y las especificaciones utilizadas en este GE son [2]:

1.**OMA NGSI 9 y NGSI10**

El Context Broker basa sus interfaces en los estándares NGSI[2] definidos en el marco de la Open Mobile Alliance (OMA). Estos estándares tienen como objetivo la gestión del contexto. Sin embargo, las especificaciones FIWARE sobre NGSI no implementan algunos aspectos definidos por OMA los cuales se consideran poco útiles o innecesariamente complejos.

OMA NGSI define dos interfaces para el intercambio de datos basado en un modelo de información común. La interfaz NGSI 10 se emplea para el intercambio de información sobre entidades y sus atributos. La interfaz NGSI 9 se emplea para el intercambio acerca de la disponibilidad de información en entidades y atributos.

Sobre el modelo de información para la gestión de contexto NGSI se definen algunos elementos básicos que es necesario conocer.

La información de contexto en FIWARE se representa pensando en estructuras de datos genéricas denominadas elementos de contexto. Un elemento de contexto se refiere a la información que se produce, se recopila y se observa que pueda ser relevante procesar. FIWARE

[2] http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf

soporta tanto estructuras de datos básicas ya integradas, como la posibilidad de definir estructuras de datos propias.

Un elemento de contexto normalmente proporciona información relevante de una entidad en particular. A modo de ejemplo, un elemento de contexto puede contener valores de la "última temperatura medida", "metros cuadrados" y "color de la pared", atributos asociados a una habitación en un edificio. Es por eso que por lo general contienen una "identidad de entidad" y un "tipo de entidad" que son una identificación exclusiva de la entidad. Por último, puede haber meta-datos vinculados a los atributos en un elemento de contexto. Sin embargo, la existencia de meta-datos en un atributo es opcional.

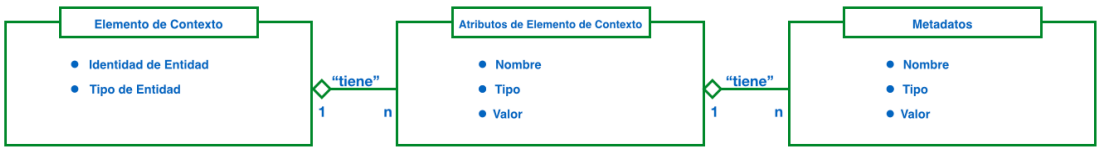

Figura 4 - Estructura del elemento de contexto

a. Estructura de los recursos NGSI-9 y NGSI-10 (Árbol de recursos)

La Figura 5 muestra la interacción básica de la publicación o suscripción de contexto desde un gestor de recursos y las entidades habitualmente relacionadas.

Figura 5 - Interacción básica

Los productores de contexto publican elementos de contexto invocando la operación updateContext en una publicación o suscripción al Context Broker. Los consumidores de contexto pueden conseguir elementos de contexto invocando la operación queryContext en una publicación o suscripción al Context Broker. Los datos de contexto se mantienen persistentes por el Context Broker y listos para ser consultados.

b. Interacción relacionada con los proveedores de contexto

Los productores de contexto publican los elementos de contexto invocando la operación updateContext en una publicación o suscripción sobre el Context Broker. Algunos productores de contexto (denominados proveedores de contexto) también pueden exportar una operación queryContext de publicación o suscripción al Context Broker cuando se les solicita.

Los consumidores de contexto pueden recuperar elementos de contexto invocando a la operación queryContext en una publicación o suscripción al Context Broker. Al contrario de la Interacción básica, como se puede ver en la Figura 6, el Context Broker reenvía la consulta al proveedor de contexto apropiado y devuelve el resultado al consumidor de contexto originario.

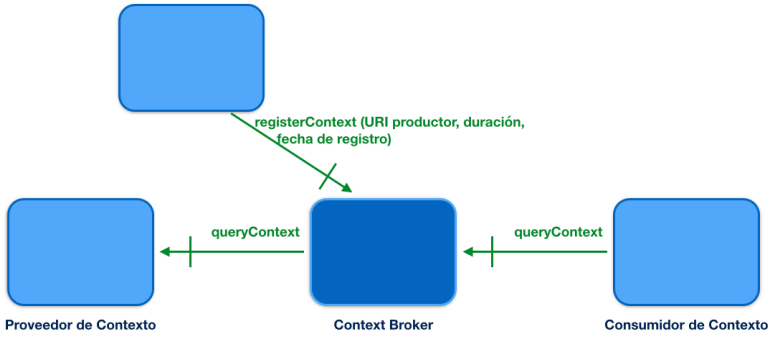

Figura 6 - Interacción proveedor de contexto

c. Interacción relacionada con las actualizaciones a los consumidores de contexto

El Context Broker es capaz de actualizar la información remitida a los consumidores de contexto capaces de procesar esa operación. Este es el caso de funcionalidad de actuación basado en la actualización de efectos secundarios como consumidor de contexto.

Como se puede ver en la Figura 7, el productor de contexto puede enviar una operación updateContext al Context Broker. Este reenvía el updateContext al consumidor de contexto apropiado y devolverá el resultado (éxito o error) al proveedor de contexto originario [2].

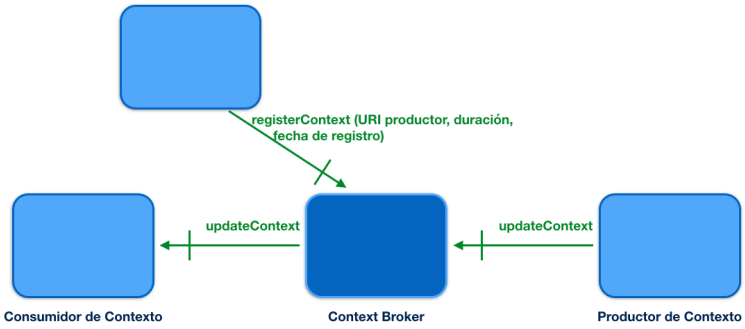

Figura 7 - Interacción de actualización

2. **HTTP**

El Protocolo de Transferencia de Hipertexto (HTTP por su siglas en inglés), es un protocolo de capa de aplicación para la comunicación entre sistemas distribuidos, y es el protocolo básico detrás de World Wide Web. A continuación se presenta un resumen sobre dicho protocolo para cubrir los aspectos mas importantes del mismo. Si se desea una mayor información lo mejor es consultar su RFC [24].

a. Conceptos básicos de HTTP

HTTP es un protocolo sin estado basado en el intercambio de mensajes entre un cliente y un servidor. Una característica importante de este intercambio es la capacidad que ofrece HTTP de negociar el tipo de contenido de estos mensajes por lo que se puede emplear como transporte de información entre sistemas de diferente naturaleza. La comunicación por lo general se lleva a cabo a través de TCP/IP, pero puede ser empleado cualquier protocolo de transporte fiable [15].

La comunicación entre cliente y servidor se produce, a través de un intercambio petición/respuesta. El cliente inicia el intercambio mediante un mensaje de petición HTTP, que es atendido a través de un mensaje de respuesta HTTP.

La versión actual del protocolo es HTTP es la 1.1, que añade algunas características adicionales a la anterior versión. La más interesante de esas características es la inclusión de conexiones persistentes.

b. Uniform Resource Locator

La forma de identificar servicios o documentos en la web es a través de cadenas de caracteres denominados localizadores uniformes de recursos (URL por sus siglas en inglés). Las URLs tienen una estructura simple, como se ve en la Figura 8, que consta de los siguientes componentes:

Figura 8 - Estructura de una URL

- Protocolo: Protocolo de comunicación.
- Servidor: Servidor con el que se comunica en la petición.
- Puerto: Puerto de red del servidor en donde conectarse.
- Identificador del recurso: ruta al recurso en el servidor.
- Parámetros de la petición: cadena de búsqueda (opcional)

c. Métodos HTTP

Las URL revelan la identidad del servicio al que queremos acceder, pero la acción que se debe realizar se especifica a través de los denominados verbos de HTTP. Por supuesto, hay unas acciones que un cliente le gustaría llevar a cabo sobre el servidor. HTTP ha formalizado en pocas operaciones los elementos esenciales necesarios para la correcta comunicación. Además son de aplicación universal.

Los métodos que se pueden emplear en las peticiones HTTP son:

GET: recoger un recurso existente. La URL contiene toda la información necesaria que el servidor necesita para localizar y devolver el recurso.

POST: crear un nuevo recurso. Las peticiones POST suelen llevar una carga útil que especifica los datos para el nuevo recurso.

PUT: actualizar un recurso existente. La carga útil puede contener los datos actualizados para el recurso.

DELETE: Eliminar un recurso existente.

Los cuatro verbos anteriores son los más populares, y la mayoría de herramientas y marcos exponen explícitamente estos verbos de petición. En ocasiones se consideran los verbos PUT y DELETE versiones especializadas del verbo POST, que pueden ser empaquetadas como peticiones POST con una carga útil que contiene la acción exacta: crear, actualizar o borrar.

Hay algunos métodos menos empleados que HTTP también soporta:

HEAD: similar a GET, pero se utiliza para recuperar únicamente las cabeceras de un servidor para un recurso determinado, generalmente para comprobar si el recurso ha cambiado, a través de marcas de tiempo.

TRACE: se utiliza para conocer los saltos que una solicitud emplear para ir y venir al servidor. Cada proxy intermedio o gateway inyectarían su IP o su nombre DNS en un campo de encabezado. Esto puede ser usado para fines de diagnóstico.

OPTIONS: se utilizan para conocer las capacidades del servidor.

d. Códigos de estado

Con las direcciones URL y los verbos, el cliente puede iniciar peticiones al servidor. A cambio, el servidor responde con códigos de estado y carga útil del mensaje. El código de estado es importante y le dice al cliente la forma de interpretar la respuesta del servidor. La especificación HTTP define ciertos rangos de números para determinados tipos de respuestas:

- 2xx: Mensaje de éxito

Esto le dice al cliente que la solicitud se ha procesado correctamente. El código más común es 200 OK. Así por ejemplo, para una petición GET, el servidor enviara el recurso en el cuerpo del mensaje. Hay otros códigos que se utilizan con menor frecuencia:

- 202 Aceptado: se aceptó la petición, pero puede no incluir el recurso en la respuesta. Esto es útil para el procesamiento asíncrono en el lado del servidor. El servidor puede optar por enviar la información para el monitoreo.
- 204 Sin contenido: no hay cuerpo del mensaje en la respuesta.
- 205 Reset del contenido: indica al cliente que restablezca la vista del documento.
- 206 Contenido parcial: indica que la respuesta sólo contiene contenido parcial. Cabeceras adicionales indican el rango exacto.

- 3xx: Re-dirección

Esto requiere que el cliente realice una acción adicional. El caso de uso más común es para saltar a una URL diferente con el fin de buscar el recurso.

- 301 Movido permanentemente: el recurso ahora está situado en una nueva dirección URL.
- 303 Ver Otros: el recurso se encuentra temporalmente en una nueva dirección URL.
- 304 Not Modified: el servidor ha determinado que el recurso no ha cambiado y el cliente debe utilizar su copia en caché.

- 4xx: Error del Cliente

Estos códigos se usan cuando el servidor piensa que el cliente tiene la culpa, ya sea mediante la solicitud de un recurso no válido o debido a una mala petición. El código más popular en esta clase es 404 Not Found, indica que el recurso no es válido y no existe en el servidor. Los otros códigos en esta clase incluyen:

- 400 Bad Request: petición incorrecta.
- 401 No autorizado: solicitud requiere autenticación. El cliente puede repetir la petición con el encabezado de autorización. Si el cliente ya incluía la cabecera Authorization, entonces las credenciales no son válidas.
- 403 Forbidden: servidor ha denegado el acceso al recurso.
- 405 Método no permitido: verbo HTTP no válido utilizado en la línea de petición, o el servidor no soporta ese verbo.

- 5xx: Error del servidor

Esta clase de códigos se utiliza para indicar una fallo en el servidor al procesar la solicitud. El código de error más común es 500 Internal Server Error.

e. Formato de Peticiones y Respuestas

Hasta ahora, se ha visto que las URLs, los métodos y los códigos de estado constituyen las piezas fundamentales de una petición / respuesta HTTP.

Figura 9 - Tramas de petición y respuesta HTTP

Ahora se va a ver el contenido de estos mensajes. La especificación HTTP indica que un mensaje de petición o respuesta tiene la siguiente estructura genérica:

```
mensaje = <linea-comienzo>
        * (<cabecera-mensaje>)

        [<cuerpo-mensaje>]

<linea-comienzo> = Petición | Estado
<cabecera-mensaje> = Nombre del campo ':' Valor del campo
```

Figura 10 - Estructura de una petición o respuesta HTTP

En la Figura 10, se puede observar como el mensaje debe estar constituido por tres partes, linea de comienzo, cabecera de mensaje y cuerpo del mensaje. El mensaje puede contener una o más cabeceras, las cuales se clasifica en:

- Cabeceras generales
- Cabeceras específicas de petición
- Cabeceras específicas de respuesta
- Cabeceras de entidad

Formato del mensaje de petición:

El mensaje de petición tiene la misma estructura genérica anteriormente descrita, a excepción de la línea de petición donde se define el método que se emplea:

```
Line-Peticion = Metodo SP URI SP Version-HTTP CRLF
Metodo = "OPTIONS"
       | "HEAD"
       | "GET"
       | "POST"
       | "PUT"
       | "DELETE"
       | "TRACE"
```

Figura 11 - Parte de la estructura del mensaje de petición HTTP

El encabezado del servidor es obligatorio para los clientes en HTTP 1.1. Las peticiones GET no tienen cuerpo de mensaje. Los encabezados de la solicitud actúan como modificadores del mensaje.

Formato del mensaje de respuesta:

El formato de respuesta es similar al mensaje de solicitud, a excepción de la línea de estado y las cabeceras. La línea de estado tiene la siguiente estructura:

```
Linea-Estado = Version-HTTP SP Code-Estado SP Frase-Razon
```

Figura 12 - Linea de estado de una respuesta HTTP

HTTP-versión se envía como HTTP 1.1. El código de estado es uno de los muchos estados comentados anteriormente.La Frase Razón es una versión legible del código de estado. Una línea de estado típica para una respuesta exitosa podría ser así:[15]

```
HTTP/1.1 200 OK
```

Figura 13 - Linea de estado de respuesta correcta

3. **Formatos de Serialización de Datos**

La serialización es el proceso de convertir un objeto (Ej: una foto, un número, etc.) en una forma que pueda ser transportada a través de un protocolo de intercambio de mensajes. Por ejemplo, se puede serializar un objeto y transportarlo a través de Internet mediante HTTP entre un cliente y un servidor. En el otro extremo, se aplica la conversión inversa para reconstruir el objeto.

El Context Broker GE de FIWARE soporta tanto la serialización de datos en formato "eXtensible Markup Language" (XML) como en formato "JavaScript Object Notación" (JSON).

XML define un conjunto de reglas para la codificación de documentos en un formato que sea legible y de lectura mecánica. Se define por las especificaciones del W3C[3] y por varias otras especificaciones relacionadas, todos las cuales son estándares abiertos y libres.

Los objetivos de diseño de XML enfatizan en la simplicidad, generalidad y usabilidad a través de Internet. Se trata de un formato de datos textual con un fuerte apoyo en Unicode. Aunque el diseño de XML se centra en los documentos, es ampliamente utilizado para la representación de estructuras de datos arbitrarias, tales como los utilizados en los servicios web.

Existen varios sistemas de esquema para la ayuda en la definición de lenguajes basados en XML, mientras que muchas APIs se han desarrollado para ayudar al tratamiento de los datos XML.

Por otro lado, JSON también es un formato estándar abierto. En este caso JSON se basa en la dupla atributo-valor. Originalmente derivada del lenguaje de programación JavaScript, JSON es un formato de datos independiente del lenguaje. El análisis y la generación de datos en formato JSON está disponible en muchos lenguajes de programación.

Esta descrito por estándares que son, RFC 7159 [27] y ECMA-404 [1]. El estándar ECMA es mínimo, describiendo solamente la sintaxis gramatical permitida, mientras que el RFC también ofrece algunas consideraciones semánticas y de seguridad.

[3] http://www.w3.org/XML/

2.2.4. Flujos de entrada y salida

Visto todo lo anterior, los interfaces de entrada y salida del Context Broker, esquematizados en la Figura 14 son:

- NGSI 9 o NGSI 10 entre Consumidores y Productores de contexto con el Broker.

- Desde el Broker hacia las aplicaciones subscritas, utilizando el puerto especificado por el Broker en la devolución de llamada en el momento de la suscripción.

- El Broker sólo soporta las interacciones con los Consumidores y Productores de contexto, pero la persistencia de los datos se hace en una base de datos que normalmente estará en la misma máquina pero que puede ser remota.

- Desde el Broker a algún otro Broker, para permitir el establecimiento de jerarquías que permitan la escalabilidad del sistema.

El rendimiento de estos flujos no se puede estimar con antelación, ya que depende totalmente de la cantidad de conexiones externas que tengan los consumidores y productores de contexto y la naturaleza de las peticiones formuladas por los consumidores y productores.

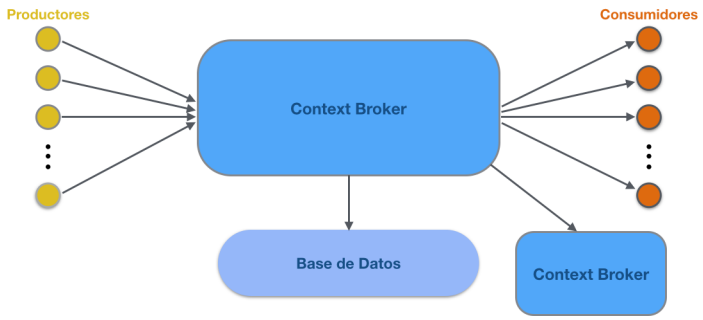

Figura 14 -Esquema de interfaces del Context Broker

2.3. PEP Proxy

2.3.1. ¿Qué es un PEP Proxy?

Un punto de aplicación de políticas (PEP) Proxy es un componente que controla el acceso a los recursos de servicios web. Es parte de un conjunto de componentes que garantizan la seguridad, autenticación y autorización en los servidores de recursos.

Cuando un tercer componente (el cliente del servicio web) quiere acceder a un recurso debe de incluir en las solicitudes una muestra, comúnmente denominado token, que sirva para identificar al usuario que realiza la acción. Este distintivo se obtiene generalmente de un componente de Gestión de Identidad (IdM) utilizando un protocolo de autenticación.

Cuando el componente PEP recibe las solicitudes, comprobará el token frente al IdM con el fin de validarlo. También se puede agregar una autorización con el fin de comprobar no sólo el acceso global al servidor de recursos, sino también los permisos específicos de acceso a recursos específicos. Para ello es necesario el componente PDP. El PDP se basa en las políticas de lenguaje extensible de marcas de acceso de control (XACML por su siglas en inglés) para llevar

a cabo esta gestión autorización. Con los componentes PEP, IdM y PDP se consigue que sólo los usuarios autorizados puedan acceder a los GEs protegidos por ellos.

2.3.2. Descripción funcional del PEP Proxy

El intercambio de identidad se basa en el protocolo OAuth [25]. Es un protocolo de autorización que permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza o servidores de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario. Es decir, aplicaciones de terceros pueden acceder a contenidos propiedad del usuario sin necesidad de que el proveedor de contenidos mantenga un registro de las credenciales de los consumidores de esos servicios. Este registro se delega a un servicio de gestión de identidades en el que se confía.

Como se ve en la Figura 15, en un escenario OAuth2 hay tres partes claramente identificadas; el cliente, que desea acceder al recurso. El propietario de dicho recurso y el proveedor en el que delega la autorización[16].

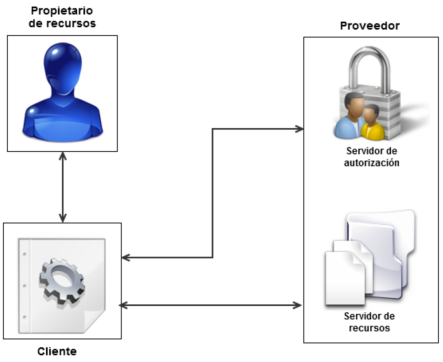

Figura 15 - Actores de un escenario OAuth2

La Figura 16 muestra el intercambio de tramas que se lleva acabo dentro del protocolo. Como se puede observar, en el intercambio de tramas, actúan las tres entidades descritas anteriormente.

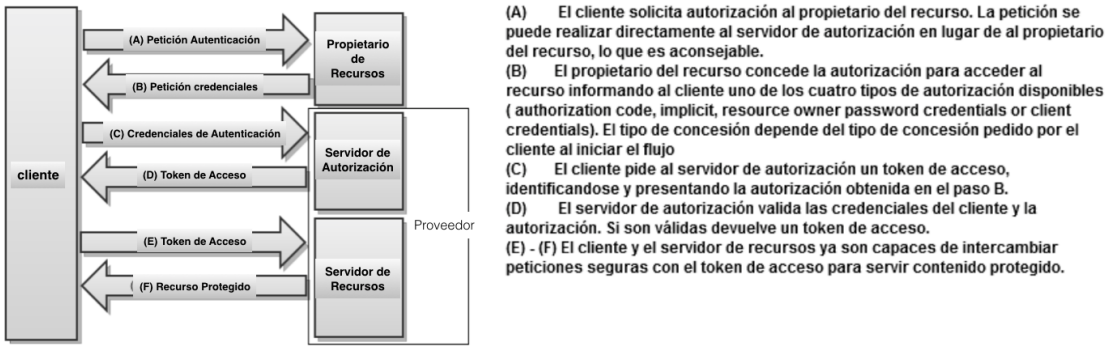

Figura 16 - Intercambio de tramas OAuth2

OAuth2 tiene dos grandes ventajas, soluciona el problema de la confianza entre un usuario y aplicaciones de terceros. Y a su vez, permite a un proveedor de servicios facilitar a aplicaciones de terceros que amplíen sus servicios con aplicaciones que hacen uso de los datos de sus usuarios de manera segura y dejando al usuario la decisión de cuando y a quien, revocar o facilitar acceso a sus datos. Creando así un ecosistema de aplicaciones alrededor del proveedor de servicios.

Para el establecimiento de políticas que se emplean entre el PDP y el PEP se utiliza la especificación XACML [26]. Se trata de un estándar que define un lenguaje declarativo de políticas de control de acceso implementado en XML y un modelo de procesamiento que describe cómo evaluar peticiones de acceso según las reglas definidas por las políticas declaradas con anterioridas [16].

2.3.3. Niveles de Seguridad

Nivel 1: Autenticación

Esta es la configuración de seguridad que se ha implementado para la protección del Orion Context Broker en este proyecto[13].

El PEP Proxy comprueba si el elemento incluido en la solicitud corresponde a un usuario autenticado. Cuando la aplicación consumidora de contexto (aplicación web) envía una petición al CB, ésta debe incluir el token OAuth2 del usuario.

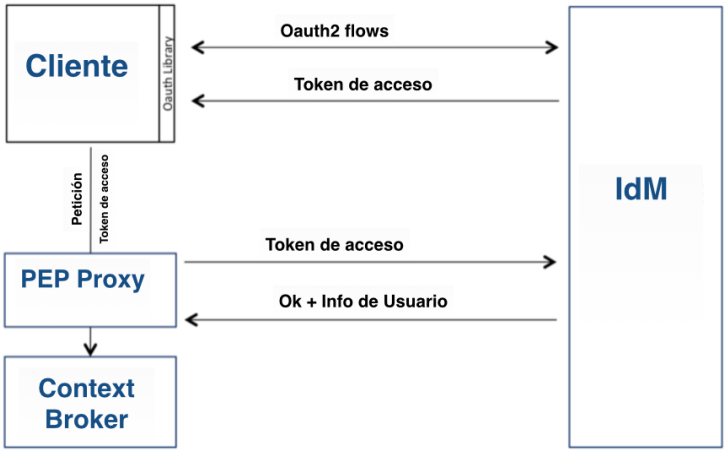

Figura 17 - Autenticación con FIWARE

La Figura 17 muestra la arquitectura de esta configuración. Cuando el PEP Proxy recibe la solicitud, se extrae el token OAuth2 de la cabecera HTTP (X-Auth-Token) y se envía una solicitud al Gestor de Identidades para que valide la autenticidad del token. Si se logra el éxito en la validación, el PEP Proxy redirige la solicitud a la dirección y puerto del servicio de backend previamente configurado (en el caso del sistema integrado en este TFG, el Context Broker).

Nivel 2: Autorización básica

Se comprueba si el elemento incluido en la solicitud corresponde a un usuario autenticado pero también si las funciones que el usuario tiene permitidas para acceder al recurso son las especificadas en la solicitud. Esto se basa en el método HTTP y la URI del recurso al que se accede.

Como se ve en la Figura 18, además de consultar al IdM, el PEP consulta al PDP por lo que, en este caso hay que configurar los roles y permisos para ese usuario en esa aplicación.

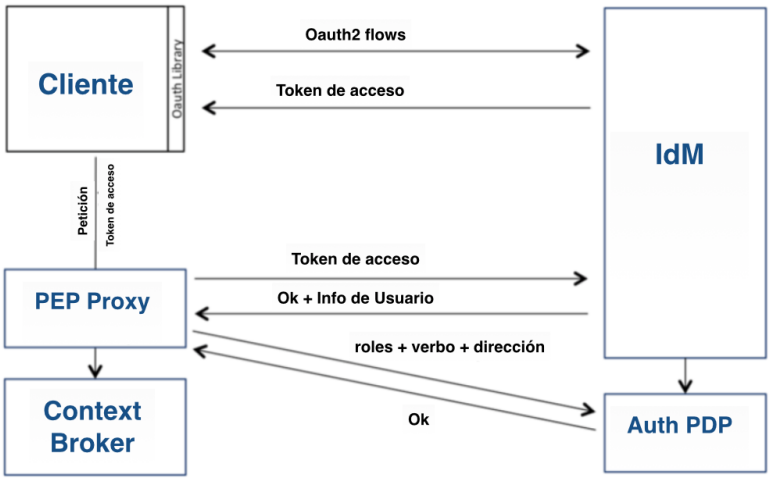

Figura 18 - Autorización básica FIWARE

Nivel 3: Autorización avanzada

No sólo se comprueba si el elemento incluido en la solicitud corresponde a un usuario autenticado sino también otros parámetros avanzados tales como el cuerpo o los encabezados de la solicitud. Se deben configurar las funciones y políticas XACML para ese usuario en esa aplicación.

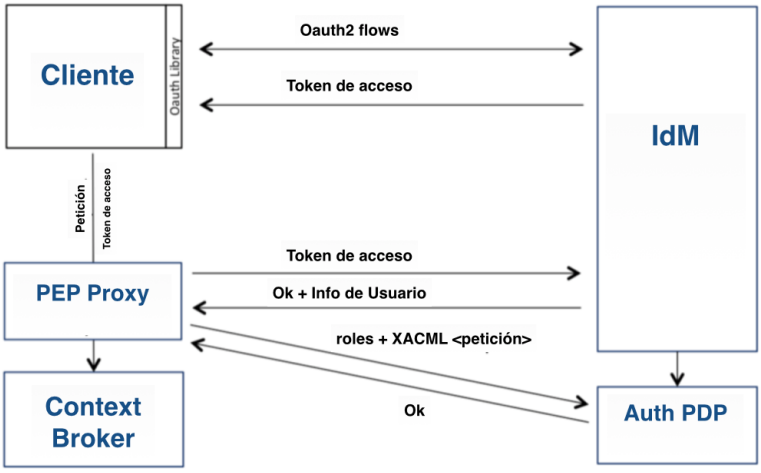

Figura 19 - Autorización avanzada FIWARE

La Figura 19 muestra la arquitectura de esta configuración. En este caso se pretende comprobar los parámetros avanzados de la solicitud. Así que, el programador debe modificar el código fuente PEP Proxy con el fin de incluir los requisitos específicos.

Con los parámetros deseados se tiene que crear una petición XACML y enviar una solicitud de autorización al PDP GE con el fin de validarla. La solicitud podría basarse en la usada en la configuración anterior [13].

2.4. Identity Manager (IdM)

2.4.1. Definiciones Previas

Autenticación: Verificación de que un usuario es quien dice ser. Se emplea el uso de una contraseña, la biometría como una huella digital, o el comportamiento distintivo para su reconocimiento.

Autorización: Define las operaciones que un usuario puede llevar a cabo en el contexto de una aplicación específica. Por ejemplo, un usuario podría ser autorizado a entrar a ver una información, mientras que otro está autorizado para la modificación de esa información.

Roles: Se les concede un papel a los usuarios, se establecen grupos de usuarios. Por ejemplo, el rol de administrador de usuarios puede ser autorizado para restablecer la contraseña de un usuario, mientras que el rol de administrador del sistema puede tener la posibilidad de asignar un usuario a un servidor específico.

Acceso de usuarios: Permite a los usuarios asumir una identidad digital específica a través de las aplicaciones, lo que permite a los controles de acceso asignar y evaluar en contra de esta identidad. El uso de una única identidad para un determinado usuario en múltiples sistemas facilita las tareas a los administradores y usuarios. Simplifica la monitorización de acceso y verificación.

Cuando se implementa un proceso de gestión de la identidad, su motivación normalmente no es gestionar un conjunto de identidades, sino más bien el conceder derechos de acceso adecuados a esas entidades a través de sus identidades. En otras palabras, la gestión de acceso es normalmente la motivación para la gestión de la identidad.

2.4.2. ¿Qué es un IdM?

En las ciencias de la computación, la gestión de identidad describe el cometido de organizar usuarios, su autenticación, autorización y sus privilegios dentro de un sistema con el objetivo de aumentar la seguridad. Cubre temas tales como, cómo los usuarios adquieren una identidad, la protección de la identidad y las tecnologías de apoyo para la protección (Ej: protocolos de red, certificados digitales, contraseñas, etc.).

El IdM se encarga de la tarea de controlar la información sobre los usuarios. Dicha información incluye información que autentica la identidad de un usuario y las acciones que está autorizado a llevar a cabo. También incluye la gestión de la información descriptiva del usuario y cómo y por quién se puede acceder y modificar esa información [23].

Ofrece herramientas a los administradores para apoyar el manejo de funciones del ciclo de vida de los usuarios. Reduce el esfuerzo para la creación y administración de cuentas, ya que soporta la aplicación de políticas y procedimientos para el registro de usuarios, administración de perfiles y modificación de cuentas.

Para los usuarios finales, el IdM proporciona una solución conveniente para el registro de solicitudes, ya que les da una forma de reutilizar atributos como dirección, correo electrónico u otros, lo que permite un manejo fácil y conveniente de la información del perfil. Los usuarios y administradores pueden confiar en soluciones estandarizadas para permitir funciones de autoservicio de usuario. Además de proporcionar un nombre de usuario nativo, el IdM apoya la integración de varios proveedores de autenticación de terceros.

Como es posible configurar varias aplicaciones vinculadas a un IdM, el principal beneficio para los usuarios es un inicio de sesión único (SSO) para todas estas aplicaciones.

Las aplicaciones no tienen que ejecutar y administrar sus propios almacenes de datos de usuario persistentes, utilizando en su lugar el almacenamiento de perfil de usuario IdM como software como servicio (SaaS) [23].

2.4.3. Módulos e Interfaces comunes en un IdM

Un sistema de gestión de identidad suele constar de los siguientes componentes básicos[21]:

- Portal IdM

Proporcionar la interfaz de aplicación para el usuario. Su funcionalidad incluye hacer más amigable la gestión de perfiles de usuario y modificación de cuentas de usuario (p.e. configuración de la contraseña y almacenamiento seguro de los datos del usuario).

- API IdM

Soporta protocolos de seguridad estandarizados (Ej: SAML 2.0, OAuth 2.0 y 2.0 SCIM).

- Sistema de IdM

El componente central del sistema maneja las solicitudes de autenticación de los usuarios. Los métodos de autenticación compatibles más empleados son:

1. Estándar SAML: Estándar basado en XML para el intercambio de datos de autenticación y autorización entre dominios de seguridad.

2. Estándar OAuth2: Estándar abierto para la autorización.

3. Nombre de usuario y Contraseña: Además se proporcionan los mecanismos de autenticación básica como es nombre de usuario y contraseña.

- Base de Datos

Repositorio central que almacena los datos de usuario, perfiles y preferencias, así como las preferencias del proveedor de servicios. Podría ser implementado como un sistema de almacenamiento distribuido, dependiendo del escenario de uso.

3. Integración de la Plataforma

3.1. Arquitectura del Sistema Integrado

3.1.1. Arquitectura de alto nivel

La Figura 20 presenta de forma esquemática como se ha llevado a cabo la integración de los GEs presentados en las secciones anteriores. Asimismo, presenta de manera explícita las implementaciones de referencia empleadas para cada uno de los GE. En este sentido, los componente utilizados, dentro de los que estaban disponibles en el catálogo FIWARE han sido:

Orion: Implementación de referencia Context Broker en FIWARE, está completamente integrado con el ecosistema y el resto de los habilitadores genéricos de FIWARE. Trabaja sobre estándares como HTTP, REST y NGSI.

Wilma: Implementación de referencia PEP Proxy en FIWARE, ya que está completamente integrado con el ecosistema y cuentas FIWARE. Esta pensado para trabajar con protocolos OAuth2 y XACML, los estándares para la autenticación y autorización elegidos por FIWARE.

Este esquema será una buena referencia donde acudir en los momentos que se pierda la perspectiva de cualquiera de las futuras secciones.

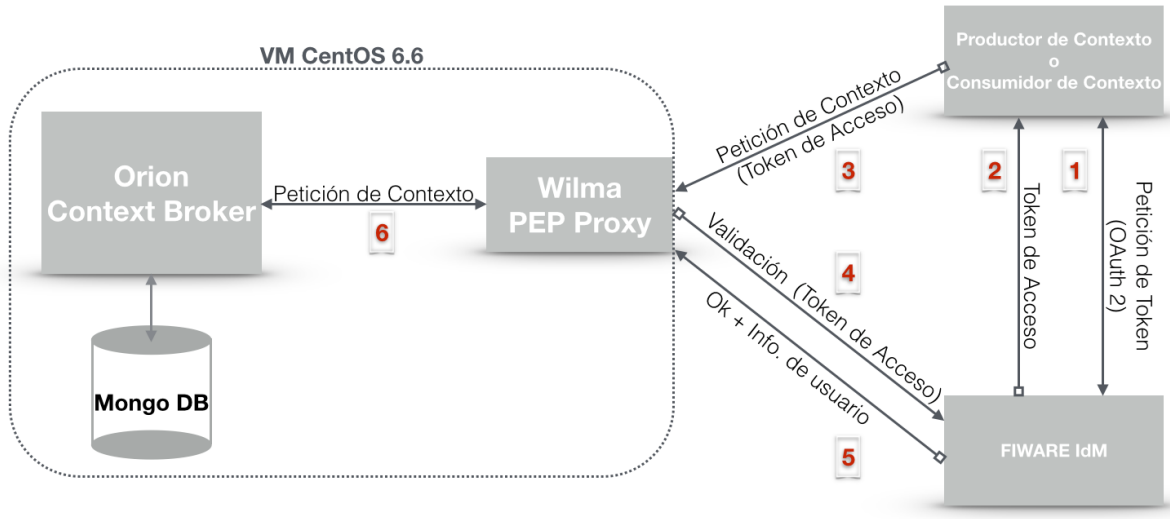

Figura 20 - Arquitectura del Sistema Integrado

Como se observa en la Figura 20, la instalación de las dos herramientas seleccionadas se ha hecho bajo una máquina virtual CentOS 6.6. El Context Broker Orion debe ser integrado obligatoriamente junto a la base de datos Mongo DB para la persistencia de la información de contexto. Si se quisiera emplear otra base de datos, esta debería ir enlazada a Mongo DB. La instanciación del tercer GE, en lugar de hacerla a través de una implementación exclusiva, se ha hecho utilizando el portal de FIWARE que incluye tanto el portal IdM como todo el sistema IdM.

Las peticiones de contexto ya sean de un productor o un consumidor se reciben en el PEP Proxy Wilma, para así tener controlado cualquier tipo de acceso al Context Broker. El PEP Proxy tras

validar el token de acceso frente al portal FIWARE donde se encuentra el IdM, reenviará las peticiones a Orion. Si el token no es validado rechazará esa petición.

El productor o consumidor de contexto debe de solicitar el token de acceso al IdM, será el administrador del sistema el que deba de dar acceso al usuario par poder conseguir tokens de acceso validos para esta aplicación.

3.1.2. Casos de uso del sistema

Antes de abordar la descripción de como se ha llevado a cabo la integración de cada uno de los GEs conviene comentar los usos más comunes en nuestro sistema.

El primero, por el que se debe comenzar es el registro de usuario. Es la primera interacción que se debe de ejecutar en el sistema para poder tener acceso al resto de los servicios. Sin este proceso no se podrá acceder a los demás recursos, a no ser que sea el administrador de la plataforma.

Una vez establecido como se realiza este proceso necesario tanto para productores como para consumidores de contexto, se presentarán las operaciones básicas asociadas a uno de estos roles, esto es, actualización de contexto, para los productores y consulta de contexto para los consumidores.

- Registro de usuario:

Para registrarse como usuario en el sistema, hacemos uso del portal FIWARE. Este portal tiene implementado un IdM. La gestión de usuarios para nuestra aplicación la dejamos en manos de FIWARE por lo que la administración que se lleva acabo del IdM es mínima ya que recae la mayor parte del peso sobre FIWARE.

En la sección de instalación del Portal KeyRock IdM se detalla de una forma más detallada como interactuar con este portal. Aquí daremos los primeros detalles para conocer quien esta actuando en este procedimiento y las nociones básicas de registro [10].

Lo primero que se debe hacer es acceder al portal en cuestión y más concretamente a la parte de registro de usuario[4]. Una vez ahí, se completa la información que solicita la página. Como es lógico sólo se admitirá una dirección de correo por usuario. Una vez creado el usuario recibiremos un correo para la confirmación de la cuenta. Realizado este proceso ya podremos acceder a la página donde administrar nuestra cuenta y hacer las operaciones necesarias para la finalización del registro.

Figura 21 - Registro de usuario FIWARE

[4] https://account.lab.fiware.org/sign_up/

Una vez creado el usuario, al acceder a la página de FIWARE veremos una sección donde aparecerán las diferentes aplicaciones en las que estamos autorizados. En un primer momento no estaremos autorizados al uso de ninguna aplicación. Las autorizaciones vendrán otorgadas por el propietario de dichas aplicaciones que a través del propio portal podrá tanto declarar dichas aplicaciones como especificar que usuarios tienen permisos sobre ellas.

El propietario deberá saber nuestro nombre de usuario para la búsqueda de nuestro perfil y así autorizarnos y asignarnos diferentes roles (por defecto, proveedor y consumidor). Dependiendo del Rol que asigne el administrador se podrán realizar diferentes operaciones en el sistema.

En el momento que el administrador nos conceda la autorización en la sección de aplicaciones autorizadas aparecerá dicha aplicación.

- Generación de token de acceso:

Una vez registrado el usuario, cualquier operación que se quiera tramitar deberá de ser dirigida a la URL que aparece en la descripción de la aplicación. En el caso del sistema que hemos integrado, cuando un proveedor o consumidor de contexto quita realizar alguna de las operaciones de gestión de contexto posibles, deberá obtener el consentimiento del PEP Proxy que redirigirá las peticiones a Orion. Para obtener dicho consentimiento, cuando el PEP Proxy reciba la solicitud de alguna operación comprobará frente al IdM si ésta contiene los elementos de validación correctos. Para conseguir una correcta validación, las operaciones deben contener un token de acceso. Este token se genera a partir de las credenciales del cliente.

Para generar el token de acceso se debe formular una petición como la que se detalla en la Figura 22. La URL de envío es la que se muestra en la Figura 22, para la obtención del token el método HTTP que se debe usar es POST. En el campo de cabeceras deberán ir la cabecera Authorization-Basic, generada a partir de las credenciales de usuario codificadas en base64 y la cabecera de Content-Type que da mayor información al servidor para el correcto entendimiento de la petición de token. El campo de body que en la Figura 22 se muestra en algunos lugares con puntos suspensivos se debe rellenar con los datos de cada usuario. Siendo el username el correo electrónico dado de alta en el portal FIWARE, la contraseña aquella que se introdujo para la validación de la cuenta y el client_id y el client_password las credenciales asignadas con anterioridad [5].

```
URL: https://account.lab.fiware.org/oauth2/token
Método: POST

Headers:
        Authorization-Basic: base64(client_id:client_secret)
        Content-Type: application/x-www-form-urlencoded


Body:
        grant-type=password&username=......@....&password=.......&client_id=........&client_secret=......
```

Figura 22 - Petición HTTP para la obtención del token de acceso

Una vez realizada la petición obtendremos como resultado el token. Este token deberá de ir como cabecera en las peticiones que se formulen frente a la aplicación.

- Actualización de contexto

Como se ha introducido anteriormente la actualización de contexto siempre la va a llevar a cabo el proveedor de contexto. Esta actualización puede venir precedida de diferentes opciones. Como se describe en la sección del Context Broker del Marco Teórico del Capítulo 2, un

Context Broker como Orion admite dos formas tanto de publicación como de subscripción que son Push y Pull.

Si la información se envía desde el productor de contexto al Context Broker cuando se encuentra disponible (método Push) nos encontramos frente a una operación básica según el estándar OMA NGSI. Esta operación es de tipo updateContext y una petición de actualización de este tipo debe de tener la siguiente forma:

```
(curl host:port/v1/updateContext -s -S --header 'Content-Type: application/json' ├--header 'Accept: application/json' --header "X-Auth-Token:
$AUTH_TOKEN" --header "X-Subject-Token: $SUB_TOKEN" -d @- | python -mjson.tool) <<EOF
{
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1",
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": "23"
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": "720"
          }
      ]
      }
  ],
  "updateAction": "APPEND"
  }
  EOF
```

Figura 23 - Petición de actualización de contexto

Alguna de las cabeceras que se observan en la petición han sido descritas con anterioridad en este documento. El resto de cabeceras, así como el cuerpo del mensaje deberá ser completado en función del contenido que se quiera actualizar, la serialización de datos que se vaya a emplear, el host y puerto destino al que invocar, etc.

De igual manera si el productor de contexto emplease el método Pull (se extrae la información tras su solicitud) la petición que generaría el proveedor de contexto tendría un formato similar.

- Consumo de contexto:

Desde la perspectiva del consumidor de contexto, el tipo de solicitud que se genera es siempre de petición de información. Al igual que en la actualización de contexto, el consumo puede ser mediante el método push (el broker decide mandar la información) o el método pull (el consumidor decide cuando solicitar esa información).

Si el consumo es mediante el método push, también denominado modo subscripción el tipo de operación que se llevará a cabo será un updateContext desde el Broker al consumidor. Este updateContext se genera dependiendo de las métricas que tenga asociadas esa subscripción (ONTIME y ONCHANGE).

Si por el contrario, el método de consumo es de tipo Pull (modo de petición activa) será el consumidor el que con una operación queryContext inicie la recogida de datos. A la cual el Broker responderá con la información de contexto solicitada.

```
(curl host:port/v1/queryContext  -s -S --header 'Content-Type: application/json' --header 'Accept: application/json' --header "X-Auth-Token:
$AUTH_TOKEN" --header "X-Subject-Token: $SUB_TOKEN" -d @- | python -mjson.tool) <<EOF
{
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ]
  }
  EOF
```

Figura 24 - Petición de consumo de contexto

3.2. Instalación de Orion

3.2.1. ¿Qué es Orion?

Es la implementación de referencia Context Broker en FIWARE, está completamente integrado con el ecosistema y el resto de los habilitadores genéricos de FIWARE. Trabaja sobre estándares como HTTP, REST y NGSI. Proporciona interfaces NGSI9 y NGSI10. Empleando estos interfaces se puede realizar diferentes operaciones típicas de un Context Broker, como el registro de contexto, la actualización de información, notificaciones de contexto, etc [22].

3.2.2. Introducción

La guía de instalación que se describe a continuación se ha desarrollado sobre una máquina virtual CentOS 6.6 por lo que si se usa una versión diferente u otro sistema operativo libre supondrá alguna variación sobre lo descrito a continuación.

Además, la versión que se ha instalado de Orion Context Broker es la 0.21.0, al igual que ocurre con la distribución del sistema operativo si se instala cualquier otra versión, es posible que se encuentre alguna variación a lo descrito en esta guía. La información relativa a otras versiones se puede consultar en el manual de instalación FIWARE[5].

Los requerimientos para el correcto funcionamiento de Orion y su base de datos MongoDB son al menos un host con dos CPU cores y 4 GB de RAM. Se trata de una estimación conservadora, aunque la posibilidad de que funcione con menos recursos depende de la cantidad de proveedores y consumidores de contexto que se pretenda soportar. El recurso crítico es la memoria RAM, ya que la base de datos hace un gran uso de ella [7].

Como base de datos se emplea MongoDB, ya sea en la misma máquina o en otro servidor diferente accesible a través de la red. Es posible enlazar cualquier otra base de datos con MongoDB. La versión recomendada de MongoDB para la versión 0.21.0 de Orion Context Broker es la 2.6.9.

Será necesaria la instalación previa de algunos paquetes RPM para solucionar conflictos de dependencias. Orion Context Broker depende de los siguientes paquetes: boost-filesystem, boost-thread, libmicrohttpd, logrotate y libcurl.

El paquete contextBroker-test (opcional pero muy recomendable) depende de los siguientes paquetes: python, python-flask , python-jinja2, python-werkzeug, curl , libxml2 , libxslt , nc, mongo-10gen y contextBroker. La dependencia mongo-10gen necesita configurar el repositorio MongoDB.

[5] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Broker_-_Orion_Context_Broker_-_Installation_and_Administration_Guide

La descarga de los paquetes correspondientes al Orion Context Broker se pueden encontrar en la zona de archivos FIWARE[6] dentro de la sección "DATA-OrionContextBroker".

3.2.3. Instalación

La instalación de Orion Context Broker se puede realizar con cualquiera de las siguientes opciones:

1. Instalación empleando el comando rpm. Una vez descargado el paquete desde el enlace anterior y instaladas todas las dependencias, ejecutar el siguiente comando:

```
sudo rpm -i contextBroker-0.21.0-1.x86_64.rpm
```

Durante este proceso de instalación el sistema puede encontrar errores de dependencias aún habiendo instalado los paquetes sugeridos en FIWARE. Las librerías que pueden presentar estos errores son:

libboost_filesystem-mt.so.5(), libboost-system-mt.so.5() y libboost_thread.so.5()

El error se debe a que Orion Context Broker depende de antiguas librerías de CentOS 6. Para resolverlo, primero se deberán eliminar las librerías actuales [14]:

```
sudo yum remove boost boost-devel boost-system boost-filesystem boost-thread
```

A continuación descargar las antiguas librerías[7]. Por último, proceder a la instalación de estos tres paquetes y volver a ejecutar el comando de instalación de Orion.

2. Opcionalmente, si tenemos acceso al RPM desde el repositorio yum. Podemos instalarlo de la siguiente forma:

```
sudo yum install contextBroker
```

Si queremos emplear esta opción debemos de tener en cuenta que el paquete que se instalará será la ultima versión y no la que se ha usado durante este Trabajo de Fin de Grado. Además para poder instalar el paquete desde el repositorio deberemos añadir el fichero testbed-fi-ware.repo con el siguiente contenido:

```
[testbed-fi-ware]
name=Fiware Repository
baseurl=http://repositories.testbed.fiware.org/repo/rpm/x86_64/
gpgcheck=0
enabled=1
```

en el repositorio /etc/yum.repos.d a fin de que yum pueda localizar el paquete de instalación de Orion.

[6] https://forge.fiware.org/frs/?group_id=7.

[7] http://rpmfind.net/linux/RPM/centos/6.6/x86_64/Packages/boost-thread-1.41.0-25.el6.x86_64.html
http://rpmfind.net/linux/RPM/centos/6.6/x86_64/Packages/boost-system-1.41.0-25.el6.x86_64.html
http://rpmfind.net/linux/RPM/centos/6.6/x86_64/Packages/boost-filesystem-1.41.0-25.el6.x86_64.html

3.2.4. Instalación Orion Context Broker Test y MongoDB

Una vez instalado Orion, se recomienda instalar el contextBroker-test el cual será útil para las primeras pruebas de funcionamiento. Para ello, lo descargamos del mismo enlace del cual obtuvimos el contextBroker. Al intentar instalarlo surge un error de dependencias y se solicita mongoDB-10gen. Por lo que debemos proceder a su instalación [11].

Mongo ofrece diferentes tipos de paquetes que pueden ser instalados individualmente, en este caso se han instalado todas las librerías.

Como hemos hecho anteriormente vamos a configurar un archivo para poder instalar desde yum el repositorio MongoDB. Generamos un archivo de nombre mongodb-org-2.6.repo con el siguiente contenido:

```
[mongodb-org-2.6]
name=MongoDB 2.6 Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
```

Este archivo deberá ir almacenado en el directorio /etc/yum.repos.d/. Ya podemos pasar a la instalación del paquete y de sus herramientas asociadas:

```
sudo yum install -y mongodb-org-2.6.1 mongodb-org-server-2.6.1 mongodb-org-
shell-2.6.1 mongodb-org-mongos-2.6.1 mongodb-org-tools-2.6.1
```

Se debe tener cuidado ya que yum actualiza los paquetes cuando una nueva versión está disponible. Para evitar actualizaciones no deseadas, se agrega la siguiente directiva al archivo / etc/yum.conf:

```
exclude=mongodb-org,mongodb-org-server,mongodb-org-shell,mongodb-org-
mongos,mongodb-org-tools
```

Puede encontrar documentación más extensa sobre MongoDB a través de visite su manual[8].

Una vez descargado e instalado MongoDB se nos permite instalar contextBroker-test que debe ser la misma versión que el paquete principal Orion Context Broker, en nuestro caso la versión 0.21.0. Para su instalación:

```
sudo rpm -i contextBroker-test-0.21.0-1.x86_64.rpm
```

No se recomienda la actualización ni de Orion Context Broker ni de MongoDB. Estas versiones son lo bastante estables y suficientemente completas como para requerir nuevas funcionalidades. Ademas así se evitan posibles problemas.

3.2.5. Ejecución de Orion Context Broker

Una vez instalado hay dos formas de ejecutar Orion Context Broker, manualmente desde linea de comandos o como un servicio del sistema. No se recomienda mezclar ambas opciones porque puede conllevar problemas [4].

1. Desde linea de comandos:

Podemos ejecutar el broker con el siguiente comando:

[8] http://docs.mongodb.org/manual/tutorial/install-mongodb-on-red-hat/

```
$ contextBroker
```

Orion arrancará en “background” por defecto, por lo que deberemos pararlo. Podemos emplear argumentos en la linea de comando, por ejemplo para especificar el puerto de escucha del Context Broker empleando la opción -port:

```
$ contextBroker -port 5057
```

En la sección 3.2.6. se detallan el resto de opciones que se pueden emplear.

2. Como servicio de sistema:

Normalmente serán necesarios privilegios de superusuario para ejecutar Orion Context Broker como un servicio de sistema. Por lo que los siguientes comandos deberán ser ejecutados como root o empleando comandos sudo.

Para iniciar el Broker como servicio, ejecute:

```
# /etc/init.d/contextBroker start
```

Luego, para parar el Context Broker:

```
# /etc/init.d/contextBroker stop
```

Y para reiniciarlo:

```
# /etc/init.d/contextBroker restart
```

Se puede usar el comando `chkconfig` para iniciar y parar automáticamente el context Broker cuando el sistema se inicie y cierre. Para un mayor control sobre el gestor, se recomienda eliminar como servicio cualquier arranque automático de los paquetes descritos con anterioridad. Para ello se observa que servicios tiene automáticos el sistema[17]:

```
chkconfig  -- list
```

Este comando nos mostrará un listado de los servicios del sistema, y nos indicará, para distintos niveles de ejecución si dicho servicio está activado o desactivado, ejemplo:

```
[aabascal@localhost ~]$ chkconfig --list
NetworkManager  0:off   1:off   2:on    3:on    4:on    5:on    6:off
abrt-ccpp       0:off   1:off   2:off   3:on    4:off   5:on    6:off
abrtd           0:off   1:off   2:off   3:on    4:off   5:on    6:off
acpid           0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
auditd          0:off   1:off   2:on    3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
blk-availability        0:off   1:on    2:on    3:on    4:on    5:on    6:off
bluetooth       0:off   1:off   2:off   3:on    4:on    5:on    6:off
certmonger      0:off   1:off   2:off   3:on    4:on    5:on    6:off
contextBroker   0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

Figura 25 - Servicios del sistema y niveles de ejecución

Añadiremos o quitaremos servicios del siguiente modo:

```
chkconfig --level <runlevel> <servicio> on
chkconfig --level <runlevel> <servicio> off
```

3.2.6. Opciones de linea de comando:

Las opciones de línea de comandos son de gran utilidad ya que permiten configurar parámetros en el arranque desde la terminal. Para obtener una lista de opciones disponibles, use:

```
$ contextBroker -u
```

Para obtener más información (incluyendo valores por defecto), utilice:

```
$ contextBroker -U
```

Estos son algunas de las opciones disponibles:

--help Muestra la ayuda del contextBroker.
--version Muestra la versión instalada.
-port <puerto> Para especificar el puerto de escucha. El puerto por defecto es 1026.
-ipv4 Ejecuta el Broker sólo en modo IPv4 (por defecto, el Broker ejecuta en IPv4 e IPv6).
-ipv6 Ejecuta el Broker sólo en modo IPv6.
-db <db> La base de datos MongoDB a utilizar o el prefijo a bases de datos por servicio. Desde la versión 0.17.0 este campo se limita a un máximo de 10 caracteres.
-dbhost <host> Para especificar el host y el puerto de escucha de MongoDB.
-dbuser <usuario> El usuario MongoDB a utilizar. Si su MongoDB no utiliza autorización, evite esta opción.
-dbPwd <pass> La contraseña MongoDB a utilizar. Si su MongoDB no utiliza autorización, evite esta opción.
-https . Trabajar en modo HTTP seguro.
-logDir <dir> Especifica el directorio que se utilizará para el archivo de registro contextBroker.
-fg Ejecuta el Broker en primer plano (útil para depurar)
-pidpath <pid_file> Especifica donde almacenar el archivo PID del proceso del Broker.

3.2.7. Procedimientos de Administración

1. Copia de Seguridad

Utilice el siguiente comando para obtener una copia de seguridad de la base de datos de Orion. Es altamente recomendable detener el Broker antes de hacer una copia de seguridad.

```
# mongodump --host <dbhost> --db <db>
```

Esto creará la copia de seguridad en el directorio dump/

2. Restaurar la base de datos

Utilice el siguiente comando para restaurar a una copia de seguridad anterior de la base de datos de Orion. Es altamente recomendable detener el Broker antes de restaurar la base de datos Orion.

Vamos a suponer que la copia de seguridad se encuentra en el directorio dump/<db>. Para restaurarlo:

```
# mongorestore --host <dbhost> --db <db> dump/<db>
```

3. Comprobación del Estado

Sólo está disponible si se está ejecutando el Context Broker como un servicio de sistema. Para comprobar el estado del Broker, utilice el siguiente los siguientes comandos con privilegios de superusuario (con el usuario root o el comando sudo):

```
/etc/init.d/contextBroker status
```

Si Broker está ejecutándose se obtendrá:

```
Checking contextBroker...                          Running
```

Si broker no se está ejecutando se obtendrá:

```
Checking contextBroker...                          Service not running
```

4.Estadísticas

Orion tiene un conjunto de contadores para los diferentes registros. Se accede a través de REST:

```
curl <host>:<port>/statistics
```

Para restablecer las estadísticas, se utiliza DELETE:

```
curl -X DELETE <host>:<port>/statistics
```

Restableciendo los contadores de estadísticas se obtiene la siguiente respuesta:

```
<orion>
  <message>All statistics counter reset</message>
</orion>
```

Hay una gran cantidad de contadores, pero sólo aquellos que se han utilizado desde la última puesta a cero se incluyen en la respuesta a la solicitud.

5. Procedimientos de Comprobación

La comprobación de procedimientos son los pasos que un administrador del sistema tomará para verificar la instalación y puesta en marcha de la herramienta. Se trata pues de un conjunto preliminar de pruebas para asegurar el funcionamiento antes de proceder a las pruebas unitarias, pruebas de integración y validación de usuarios [7].

Prueba de extremo a extremo

1. Arrancar Orion en el puerto por defecto (1026)
2. Ejecutar el siguiente comando:

```
curl localhost:1026/version
```

3. Comprobar que se recibe el número de versión como respuesta así como información de tiempo de actividad y del entorno de compilación:

```
curl localhost:1026/version
INFO@08:55:59  rest.cpp[768]: Starting transaction from 0.0.0.0:54851/version
<orion>
  <version>0.21.0</version>
  <uptime>0 d, 0 h, 1 m, 17 s</uptime>
  <git_hash>7565069869c839ab7c4f90d5ec752d5d0d57986e</git_hash>
INFO@08:55:59  rest.cpp[382]: Transaction ended
  <compile_time>Sun May 10 19:38:32 CEST 2015</compile_time>
  <compiled_by>fermin</compiled_by>
  <compiled_in>centollo</compiled_in>
</orion>
```

Figura 26 - Respuesta de prueba extremo a extremo

Lista de procesos en ejecución

Se debe de poder observar en algún proceso el nombre “contextBroker” en funcionamiento:

```
# ps ax | grep contextBroker
```

3.3. Instalación de Wilma

3.3.1. ¿Qué es Wilma?

Implementación de referencia PEP Proxy en FIWARE, está completamente integrada con el ecosistema y cuentas FIWARE. Esta pensado para trabajar con los protocolos OAuth2 y XACML. Se emplea para la autorización y autenticación de los usuarios para el acceso a aplicaciones en backend.

3.3.2. Introducción

A continuación se describe como instalar y administrar PEP Proxy Wilma. Con el fin de ejecutar el PEP Proxy Wilma, es necesario tener instalado previamente el siguiente software [9]:

- Servidor Node.js V0.8.17 o superior.
- Node Packaged Modules (por lo general, se incluye dentro de Node.js)

3.3.3. Instalación

Para la instalación del Wilma PEP Proxy se deben efectuar los siguientes pasos:

Primero, descargar el software utilizando GitHub con el siguiente comando:

```
git clone https://github.com/ging/fi-ware-pep-proxy
```

Instalar todas las bibliotecas necesarias empleando NPM:

```
cd fi-ware-pep-proxy
npm install
```

Editar el archivo de configuración, para ello se puede copiar el archivo config.js.template y editarlo con la información correspondiente. A continuación se puede ver un ejemplo:

```
var config = {};

config.account_host = ‘https://account.lab.fi-ware.org';

config.keystone_host = 'cloud.lab.fi-ware.org';
config.keystone_port = 4731;

config.app_host = 'www.google.es';
config.app_port = ’80';

config.username = 'pepProxy';
config.password = ‘pepProxy';

config.check_permissions = false;

module.exports = config;
```

En el Capítulo 4 de este documento se detallará como se han completado los diferentes campos de este archivo de configuración en la integración que se ha realizado. Pero para dar una breve descripción de cada uno de estos campos saber que "config.account_host" debe de completarse con la Url donde se encuentra el registro de la aplicación que contiene las credenciales del PEP Proxy. El "config.keystone_host" y "config.keystone_port" deben de direccionar al servidor donde esta alojado el IdM, en uno de los campos contendrá la Url y en el otro el puerto. De igual manera "config.app_host" y "config.app_port" contienen la información de redirección al servidor y puerto donde se ubica la aplicación en backend. En "config.username" y "config.password" se deben introducir las credenciales con las que autentica Wilma frente al IdM.

A continuación, ejecutar el siguiente comando, son necesarios permisos administrativos ya que se ejecuta sobre el puerto TCP 80:

```
node server.js
```

Otra opción es instalar forever.js para así ejecutarlo en un entorno de producción:

```
sudo npm install forever -g
```

Y a continuación, ejecutar el servidor utilizando forever:

```
forever start server.js
```

3.3.4. Sistema de Administración

1. Procedimientos de Comprobación

La comprobación de procedimientos al igual que en la instalación de Orion son los pasos que se deben de llevar acabo por el administrador del sistema para verificar la instalación y puesta en marcha de la herramienta [8].

Prueba de extremo a extremo

Las solicitudes al Proxy deben realizarse incluyendo la cabecera HTTP: X-Auth-Token. Esta cabecera contiene el token de acceso de OAuth obtenido del IdM tal y como se detallará en la siguiente sección.

Ejemplo de petición:

```
GET / HTTP/1.1
Host: proxy_host
X-Auth-Token:z2zXk...ANOXvZrmvxvSg
```

Para comprobar que el proxy puede generar esta solicitud ejecute el siguiente comando:

```
curl --header "X-Auth-Token:z2zXk...ANOXvZrmvxvSg" http://proxy_host
```

Una vez autenticado, la respuesta remitida incluirá cabeceras adicionales con información de usuario HTTP.

3.3.5. Alta del Wilma en el portal FIWARE

Si se desea emplear el PEP Proxy Wilma junto con el IdM del portal FIWARE, se debe de dar de alta el PEP Proxy en el portal. Para ello el administrador del PEP Proxy deberá solicitar

desde el portal del IdM en la sección relativa a la sección PEP Proxy, las credenciales de este. Se generará automáticamente un nombre de usuario y una contraseña para ese proxy.

Para que el IdM reconozca al PEP cuando establezcan comunicación se deben de introducir estas credenciales en el archivo de configuración comentado en la parte de instalación dentro de los campos "config.username" y "config.password".

Con ello, al arrancar Wilma se autenticará frente al IdM y obtendrá el token necesario para la validación de los tokens contenidos en las operaciones de gestión de contexto que Wilma debe supervisar.

El siguiente archivo de configuración es el que se ha tenido que editar para llevar acabo este trabajo:

```
var config = {};

config.pep_port =80 ; // Puerto de escucha de la aplicacion Wilma. Por defecto el 80 porque es el de salida a internet

config.account_host = 'http://account.lab.fiware.org';


config.keystone_host = 'cloud.lab.fiware.org';
config.keystone_port = 4731;


// Donde esta la app. contextBroker. Puerto del ContextBroker
config.app_host = '0.0.0.0';
config.app_port = '9090';

// Este es el usuario que he creado en account.lab.fiware.org, lo que venía en el correo.


// Credenciales WILMA

config.username = 'pep_proxy_4f9f304c0aa24e588185da2e822221e7';
config.password = '0cd25a4be96c47baa8b9cc785fc367ba';
```

Figura 27 - Archivo de Configuración Wilma

3.4. Configuración KeyRock sobre el Portal Web

3.4.1 Introducción

Para la integración de un IdM en el sistema se plantea mas de una posibilidad.

La opción que se ha desarrollado en este trabajo es el empleo del portal FIWARE como IdM. Esta opción confía la gestión del IdM a la plataforma FIWARE, siendo totalmente dependiente de ella el correcto funcionamiento del sistema alojado. Como administrador de un sistema alojado, se tendrán mayores permisos que los de usuario, pero siempre estos permisos serán muy limitados en comparación al desarrollo de un IdM propio. Todo aquel que desee tener acceso a un servicio dependiente del IdM del portal FIWARE deberá crearse una cuenta en dicho portal [19].

Si por el contrario, se prefiere integrar en el sistema un IdM totalmente controlado, se recomienda hacerlo a través de las herramientas mediante las cuales se ha desarrollado el portal web de FIWARE. Estas herramientas son Keystone[9] (el servicio de identidad de OpenStack) y Horizon[10] (el servicio de interfaz web de OpenStack).

[9] http://docs.openstack.org/developer/keystone/

[10] http://docs.openstack.org/developer/horizon/

Para acceder al portal web de FIWARE se hará desde el siguiente enlace[11].

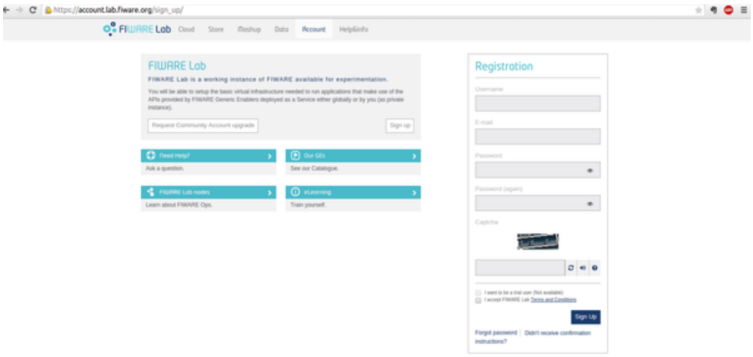

Figura 28 - Página principal del portal FIWARE

3.4.2. Instrucciones al Administrador

Lo primero que se debe hacer es crear una cuenta en el portal. Para ello pulsar "Sign up" en la página principal. Una vez ahí se deben de rellenar una serie de campos para crear la cuenta de registro. Una vez registrado se observa una página con dos secciones principales, Applications y Organizations. Como administrador de la aplicación Orion Context Broker se deberá ir a registrar la aplicación en el portal.

El siguiente paso será darle un nombre, una descripción, una URL y la URL donde se encuentra ubicada la aplicación Orion.

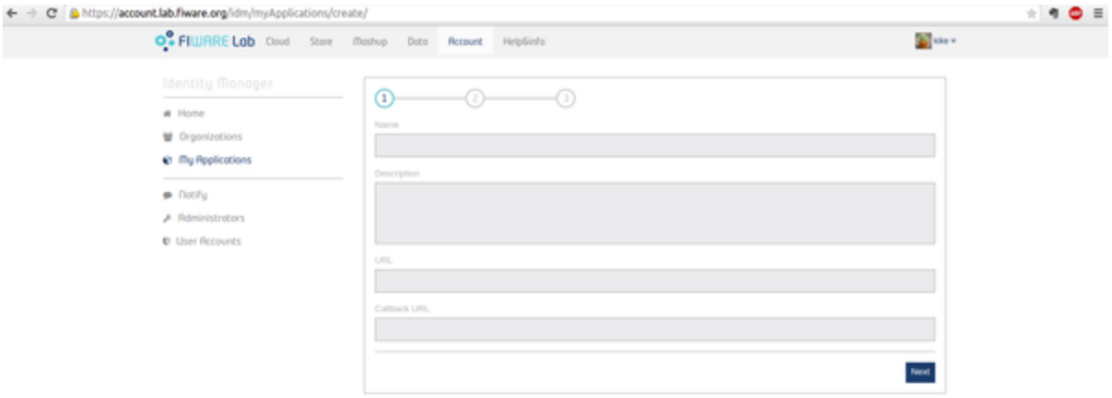

Figura 29 - Registro de una aplicación en el portal FIWARE

Se puede seleccionar una imagen para la aplicación. Por último, el tercer paso sirve para crear roles y permisos de aplicación. Los roles preconfigurados son los de proveedor y consumidor. Una vez finalizado el proceso, se podrá observar la aplicación ya creada, con los parámetros configurados anteriormente. Como administradores de la aplicación tendremos acceso a las credenciales PEP Proxy. Estas credenciales deberán ir almacenadas en el archivo de configuración de Wilma. Hablaremos mas adelante de ellas en la sección de Testing.

Además como usuarios de la aplicación tendremos una sección de credenciales OAuth2 donde encontramos un Client ID y un Client Secret los cuales se deben emplear para la generación de un Token de acceso. Por último, veremos dos campos de gestión, que son el de Authorized Users y Authorized Organizations. Ahí se añaden a aquellos usuarios u organizaciones que tengan cuenta en FIWARE y queramos dar acceso a Orion, previo paso por el PEP Wilma que validará sus cuentas frente al KeyRock IdM.

[11] https://account.lab.fiware.org/

4.Validación Funcional del Sistema

4.1. Arranque del sistema integrado

Como se describió en la guía de instalación, no se ha permitido que ninguna de las herramientas instaladas arranquen como servicio automático del sistema. Por lo que, antes de comenzar, debemos activar nuestro sistema. Primero se inicia la base de datos Mongo que posteriormente utilizará el gestor de contexto Orion para la persistencia de los datos. [30]

```
sudo service mongod start
```

```
[aabascal@localhost ~]$ sudo service mongod start
[sudo] password for aabascal:
Starting mongod:                                           [  OK  ]
```

Figura 30 - Respuesta en el arranque de Mongo DB

Nos devolverá un OK si el arranque ha sido correcto. Seguidamente se arranca Orion. Para poder tener ubicados de forma ordenada los archivos que se van a generar en el arranque, entramos en la carpeta donde se creó la instalación. En el ejemplo descrito, se le indica al gestor de contexto que utilice el puerto 9090 y que la base de datos es accesible en la máquina cuya dirección IP es 127.0.0.1 (localhost). Tanto el fichero de log como el PID del proceso se crearán en el mismo directorio en el que se ejecuta el comando [12].

```
cd contextBroker
/usr/bin/contextBroker -port 9090 -logDir ./ -pidpath ./ -dbhost 127.0.0.1 -db orion
```

```
[aabascal@localhost ~]$ cd contextBroker
[aabascal@localhost contextBroker]$ /usr/bin/contextBroker -port 9090 -logDir ./ -pidpath ./ -dbhost 127.0.0.1 -db orion
log directory: './'
INFO@08:03:32  contextBroker.cpp[1354]: Orion Context Broker is running
ERROR@08:03:32  contextBroker.cpp[958]: PID File (open './': Is a directory
[aabascal@localhost contextBroker]$ INFO@08:03:32  MongoGlobal.cpp[184]: Successful connection to database
INFO@08:03:32  contextBroker.cpp[1165]: Connected to mongo at 127.0.0.1:orion
INFO@08:03:32  MongoGlobal.cpp[502]: Database Operation Successful ({ conditions.type: "ONTIMEINTERVAL" })
INFO@08:03:32  contextBroker.cpp[1441]: Startup completed
```

Figura 31 - Respuesta en el arranque de Orion Context Broker

Ya sólo nos queda por arrancar el PEP Proxy Wilma, para tener todos los componentes de nuestro sistema activos. Antes de arrancarlo, se muestra el archivo de configuración que acompaña a este caso [28].

```
var config = {};

config.pep_port =80 ; // Puerto de escucha de la aplicacion Wilma. Puerto 80, de salida a internet

config.account_host = 'http://account.lab.fiware.org';
config.keystone_host = 'cloud.lab.fiware.org';

config.keystone_port = 4731;

config.app_host = '0.0.0.0';
config.app_port = '9090';

config.username = 'pep_proxy_4f9f304c0aa24e588185da2e822221e7';
config.password = '0cd25a4be96c47baa8b9cc785fc367ba';
```

```
// En segundos

config.chache_time = 300;

// if enabled PEP checks permissions with AuthZForce GE.
// only compatible with oauth2 tokens engine
//config.azf = {
//    enabled: false,
//    host: '130.206.84.5',
//    port: 6019,
//    path: '/authzforce/domains/d698df7f-ffd4-11e4-a09d-ed06f24e1e78/pdp'
//};

// list of paths that will not check authentication/authorization
// example: ['/public/*', '/static/css/']

config.public_paths = [];

// options: oauth2/keystone

config.tokens_engine = 'oauth2';

config.magic_key = undefined;

module.exports = config;
```

Figura 32 - Archivo de Configuración Wilma

En el archivo de configuración de la Figura 32 se puede observar que Wilma realiza la escucha por el puerto 80, este parámetro se establece en la variable "config.pep_port". También se debe configurar la ubicación del IdM, en este caso, que se hace uso del alojado en el portal FIWARE, sus dirección y puerto son los que se observan en "config.keystone_host" y "config.keystone_port". En el campo "config.account_host" se debe ubicar donde se aloja la cuenta de registro de la aplicación que contiene el PEP Proxy.

Los campos de "config.app_host" y "config.app_port" deben contener la dirección y puerto donde se ubica la aplicación de backend. En este caso, Orion Context Broker. Es donde Wilma reenviará las peticiones validadas. La dirección que se le ha dado es la "0.0.0.0" pero bien podría ser "127.0.0.1" ya que se encuentra en la misma dirección en la que esta Wilma, el puerto será aquel en el se decidió arrancar Orion, en esta sección de validación del sistema fue el 9090.

También dentro del archivo de configuración se debe introducir tanto el identificador del PEP Proxy como su contraseña ("config.username" y "config.password"). Estas credenciales han sido generadas por el IdM de FIWARE cuando hemos dado de alta un PEP Proxy dentro de la aplicación Context Broker también creada en ese portal. Estas credenciales serán las que debe emplear el PEP Proxy para su propia autenticación frente al IdM. Además, también se puede observar que el método de autorización que emplea es OAuth2. Para arrancar el PEP Proxy introducimos el siguiente comando:

```
cd fi-ware-pep-proxy
sudo node server.js
```

```
[aabascal@localhost fi-ware-pep-proxy]$ sudo node server.js
express deprecated app.configure: Check app.get('env') in an if statement server.js:35:5
Starting PEP proxy in port 80. Keystone authentication ...
Success authenticating PEP proxy. Proxy Auth-token:  5b770b4fc734409b897f67e29bcff60a
```

Figura 33 - Respuesta de arranque PEP Proxy Wilma

Como se aprecia en la Figura 33, lo primero que debe hacer el PEP Proxy al arrancar es autenticarse frente al IdM. Al igual que un cliente que desee tener acceso al servicio y deba generar un token de acceso, Wilma para remitir los token de acceso que le envíen terceros debe también generar su propio token de acceso y estar validado frente el IdM. Sus credenciales de validación son las que se han introducido en el archivo de configuración Wilma de la Figura 32.

Las Figuras 34 muestra en detalle la captura realizada a través de un analizador de protocolos, correspondiente al paso 1 de la Figura 35.

```
No.     Time        Source          Destination     Protocol Length Info
      8 0.127305863 10.211.55.12    130.206.84.8    HTTP      466 POST /v3/auth/tokens HTTP/1.1  (application/json)

▷ Frame 8: 466 bytes on wire (3728 bits), 466 bytes captured (3728 bits) on interface 0
▷ Ethernet II, Src: Parallel_74:a7:08 (00:1c:42:74:a7:08), Dst: Parallel_00:00:18 (00:1c:42:00:00:18)
▷ Internet Protocol Version 4, Src: 10.211.55.12 (10.211.55.12), Dst: 130.206.84.8 (130.206.84.8)
▷ Transmission Control Protocol, Src Port: 34301 (34301), Dst Port: remcap (4731), Seq: 1, Ack: 1, Len: 412
▽ Hypertext Transfer Protocol
  ▽ POST /v3/auth/tokens HTTP/1.1\r\n
    ▷ [Expert Info (Chat/Sequence): POST /v3/auth/tokens HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /v3/auth/tokens
      Request Version: HTTP/1.1
    User-Agent: node-XMLHttpRequest\r\n
    Accept: */*\r\n
    Content-Type: application/json\r\n
    Host: cloud.lab.fiware.org:4731\r\n
  ▷ Content-Length: 228\r\n
    Connection: close\r\n
    \r\n
    [Full request URI: http://cloud.lab.fiware.org:4731/v3/auth/tokens]
▽ JavaScript Object Notation: application/json
  ▽ Object
    ▽ Member Key: "auth"
      ▽ Object
        ▽ Member Key: "identity"
          ▽ Object
            ▽ Member Key: "methods"
              ▷ Array
            ▽ Member Key: "password"
              ▽ Object
                ▽ Member Key: "user"
                  ▽ Object
                    ▽ Member Key: "name"
                        String value: pep_proxy_4f9f304c0aa24e588185da2e822221e7
                    ▽ Member Key: "password"
                        String value: 0cd25a4be96c47baa8b9cc785fc367ba
                    ▽ Member Key: "domain"
                      ▽ Object
                        ▷ Member Key: "id"
```

Figura 34 - Captura de petición de Token del PEP Proxy Wilma

Podemos apreciar en la captura de la Figura 34, que el envío de petición de token se esta realizando a través del protocolo de transporte HTTP, como ya se describió de forma teórica en el Capítulo 3. Vemos como la petición que envía Wilma al ejecutar el arranque, es un POST al Host donde se ubica el IdM FIWARE para solicitar un token de acceso. Si observamos el contenido del campo JSON, se observan las credenciales del PEP Proxy que se introdujeron en el archivo de configuración para la autenticación y que estas se envían en claro.

Si observaremos la captura del paso 2 según la Figura 35, podríamos ver como el IdM de FIWARE le asigna el token de acceso al PEP Proxy Wilma. Este token es quien le va a permitir a Wilma estar autenticado durante un periodo de tiempo en el IdM FIWARE para poder validar tokens de terceros.

La Figura 35 resume el estado del sistema una vez completados todos estos pasos. En las siguientes secciones se procederá a describir las diferentes pruebas funcionales realizadas sobre él.

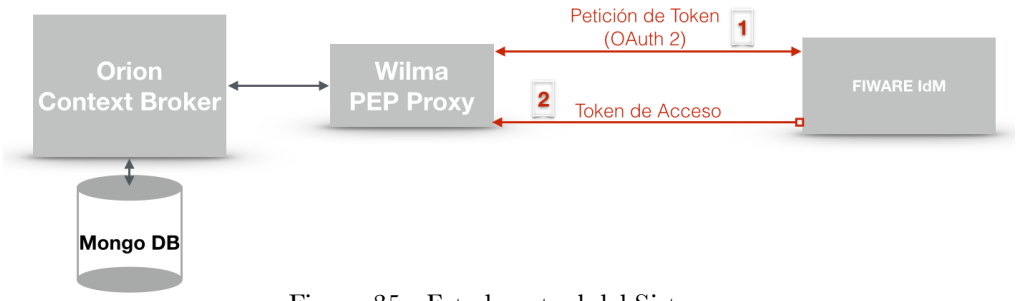

Figura 35 - Estado actual del Sistema

4.2. Acceso de terceros al sistema integrado

El proceso que se describe en esta sección corresponde a como un usuario accede al gestor de contexto Orion, ya sea para producir o para consumir contexto. El cliente que va a solicitar acceso al Context Broker deberá haber creado una cuenta en el portal FIWARE como se explicó en el Capítulo 3. También debe generar una aplicación de "ClienteWeb" donde poder lograr unas credenciales OAuth2 propias. Además el administrador de la aplicación Context Broker dentro de FIWARE y donde se alojan las credenciales del PEP Proxy Wilma deberá dar autorización a este usuario (y con ello, implícitamente, a sus aplicaciones) para el acceso a la aplicación Contexto Broker.

La Figura 36 muestra el esquema funcional del proceso completo de acceso a los servicios del gestor de contexto. El primer paso, que se corresponde con las interacciones 1 y 2 de la Figura 36, es la obtención, por parte del productor o consumidor de contexto, de un token de acceso. Este token se empleará en las siguientes peticiones que realice hacia el Context Broker.

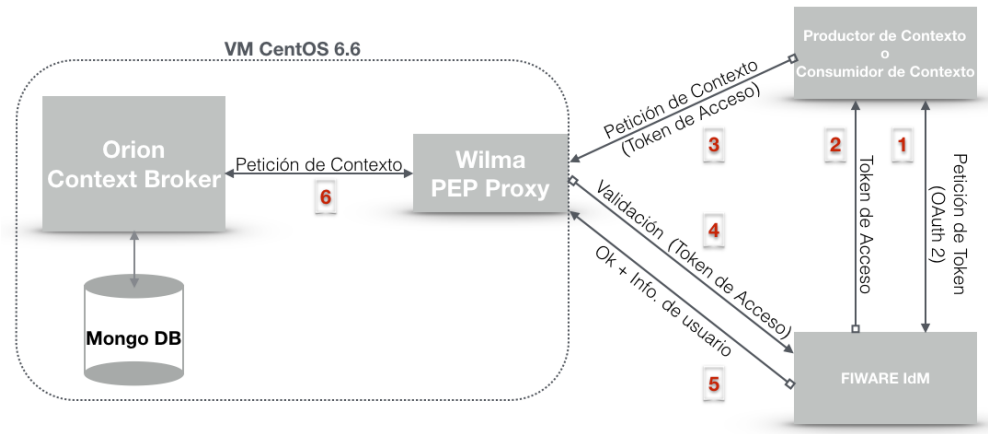

Figura 36 - Interacción de terceros con el sistema integrado

La petición se puede realizar desde cualquier aplicación que permita hacer peticiones REST. En este caso, se utilizó la aplicación de Google Chrome, Postman. La Figura 37 muestra en detalle la petición realizada.

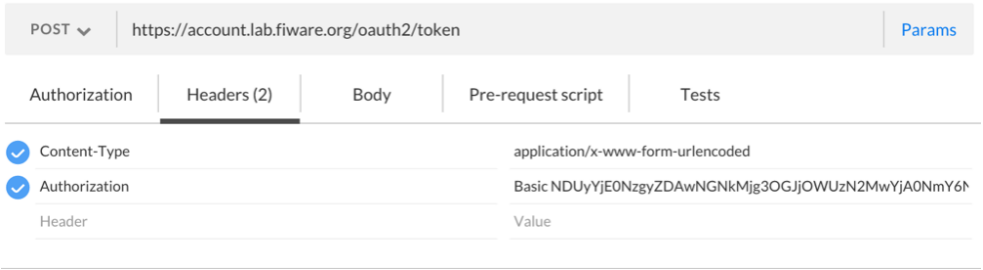

Figura 37 - Petición de token al IdM de FIWARE

La cabecera Authorization se crea a partir de las credenciales obtenidas para la aplicación en el portal IdM utilizadas como Autenticación básica HTTP. El body de la petición de la Figura 37 es el siguiente:

```
grant_type=password&username=aaa86@alumnos.unican.es&password=telematica&client_id=452b14782d004cd2878bc9e37c0b046f&client_secret=4a742be028a04105845cb8bc35fd527f
```

Donde, el usuario y password son los de acceso a la cuenta de usuario FIWARE y el client_id y el client_secret las credenciales OAuth2 obtenidas en la aplicación "ClienteWeb" para este usuario.

La respuesta que se genera es el access_token que se muestra en la Figura 38. No se muestra la captura de tráfico Wireshark ya que se vería un tráfico muy similar al mostrado anteriormente en las Figura 34.

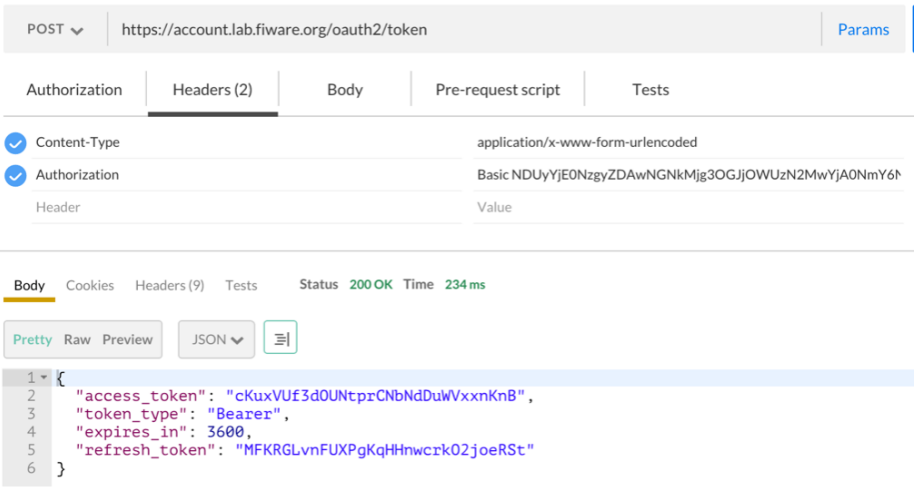

Figura 38 - Respuesta de token al IdM de FIWARE

Una vez obtenido un token de acceso para el cliente del gestor de contexto, se procede a realizar la petición de contexto a Orion utilizando este token de acceso (paso 3 de la Figura 36). El cliente, cree que la solicitud que envía la realiza directamente a Orion. Sin embargo, esa solicitud de contexto la recibe el PEP Proxy Wilma que, tras validar el token recibido en el IdM (pasos 4 y 5 de la Figura 36), redirige la petición hacia Orion (paso 6 de la Figura 36).

El cliente sólo realiza una petición, el resto de redireccianamientos y validaciones son gestionadas de forma automática por Wilma.

El ejemplo que se detalla en la Figura 39 se corresponde con la creación de una entidad de contexto, pero bien podría ser cualquier petición NGSI9 o NGSI10 de las soportadas por el Orion Context Broker. Para más información acerca de las operaciones soportadas se puede consultar el Apéndice 1. Esta vez, la petición se realizó mediante el complemento de Firefox llamado RESTClient, pero cualquier cliente capaz de realizar peticiones REST sería válido.

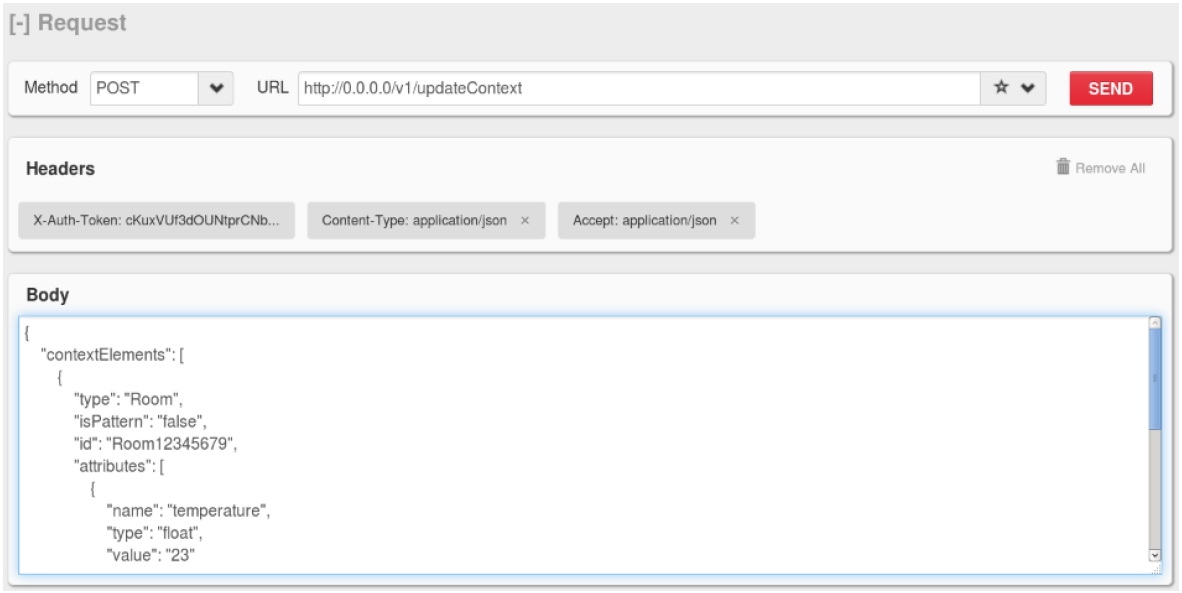

Figura 39 - Petición de Contexto a Orion Context Broker

Como cabeceras debemos introducir las cabeceras necesarias para que Orion comprenda que se trata de datos de contexto en formato JSON y la X-Auth-Token con el token de acceso generado anteriormente para que Wilma autentique al cliente.

```
[TOKEN] Checking token with IDM...
[ROOT] Access-token OK. Redirecting to app...
Refused to set unsafe header "accept-encoding"
Refused to set unsafe header "content-length"
[TOKEN] Checking token with IDM...
User access-token not authorized
```

Figura 40 - Redirección de Wilma a Orion tras validar el Token

Consultando los mensajes de log de Wilma y que se ven en la Figura 40, se puede observar como redirecciona esa petición hacia Orion si la autenticación ha sido correcta. La Figura 40 también recoge el caso en el que la autenticación del cliente no ha sido exitosa (el token de acceso no esta autorizado).

La respuesta a la petición anterior en el primer caso será la que se muestra en la Figura 41 y la Figura 42.

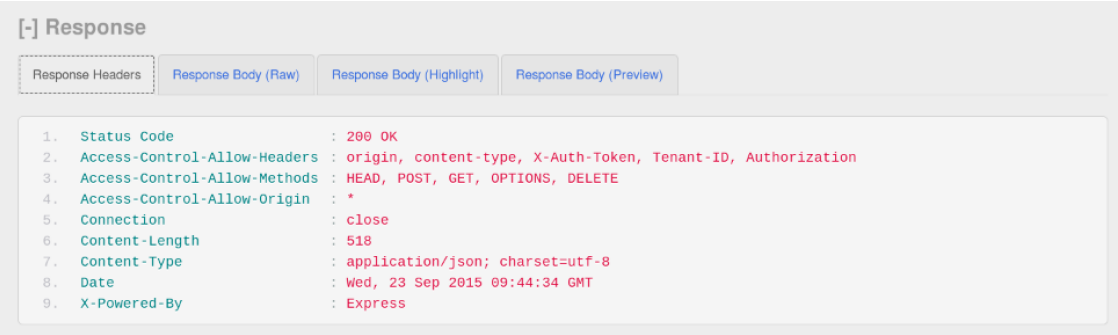

Figura 41 - Cabecera de respuesta en el REST Client a la petición de contexto

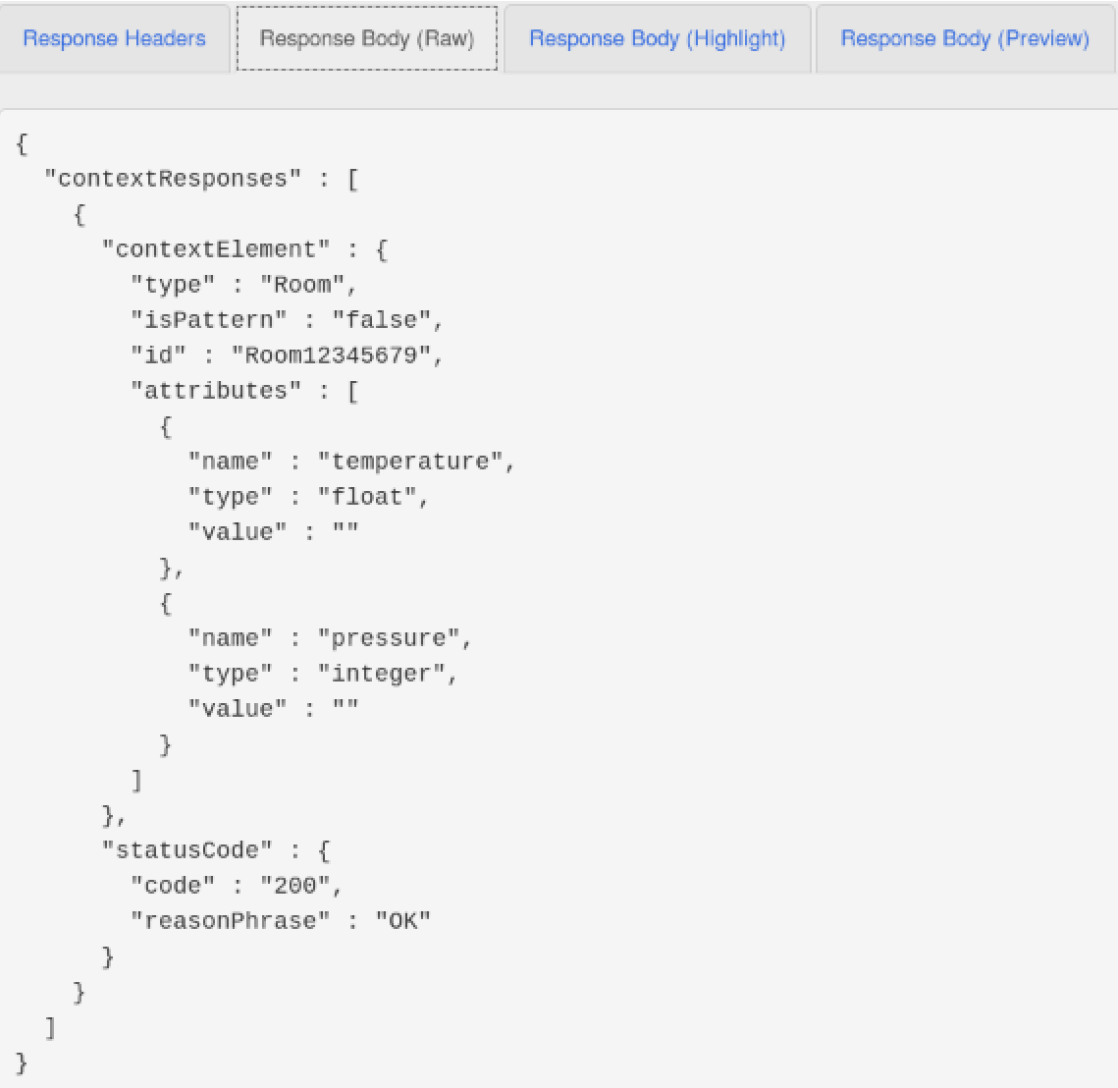

Figura 42 - Cuerpo de la respuesta en el REST Client a la petición de contexto

El tráfico HTTP que se acaba de generar para el acceso a terceros en el sistema integrado, ha sido capturado en el enlace de conexión representado como paso 6 en la Figura 36. Las Figuras 43, 44 y 45 muestra información acerca del contenido de este tráfico.

Si observamos el POST (paso 3 en la Figura 36):

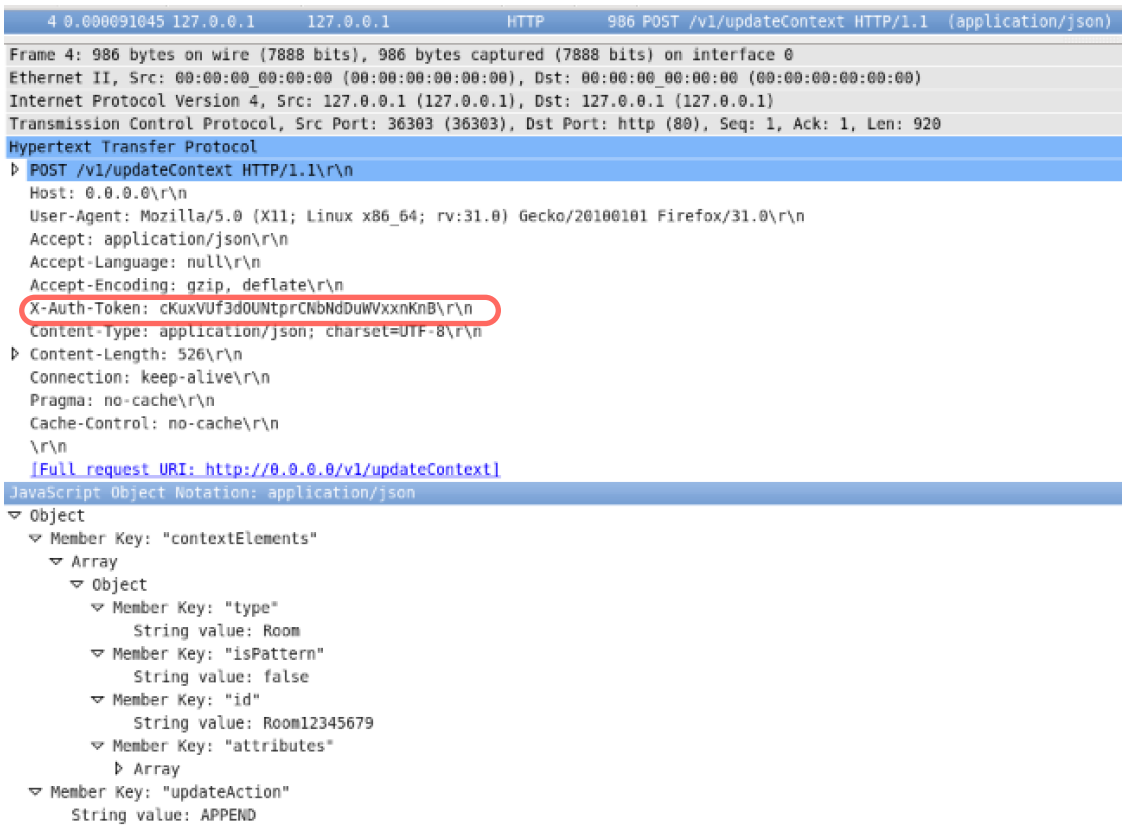

Figura 43 - Captura Wireshark de petición de contexto con token de acceso en claro

Podemos ver en la Figura 43, como también en esta ocasión son visibles los campos de la petición. Ya que no se está usando HTTPS. Se puede observar tanto los campos que se han introducido en el body para Orion Context Broker como lo que es mas interesante aún, el X-Auth-Token. Si alguien interceptase ese token podríamos tener problemas de suplantación de identidad.

En la misma captura, una vez realizados todos los pasos de forma satisfactoria podemos observar el tráfico que hay entre Orion y Mongo.

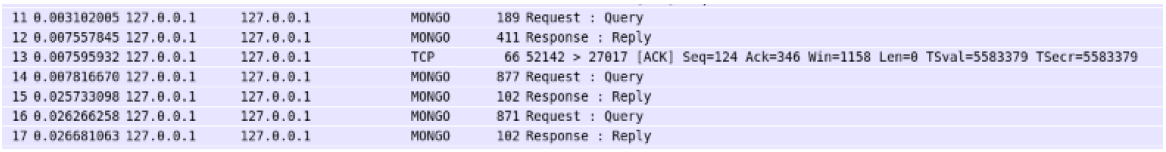

Figura 44 - Captura Wireshark intercambio de información Orion y Mongo

Una vez procesado por todos los componentes la petición, se procede a la respuesta de la misma al REST Client como se puede observar en la Figura 45.

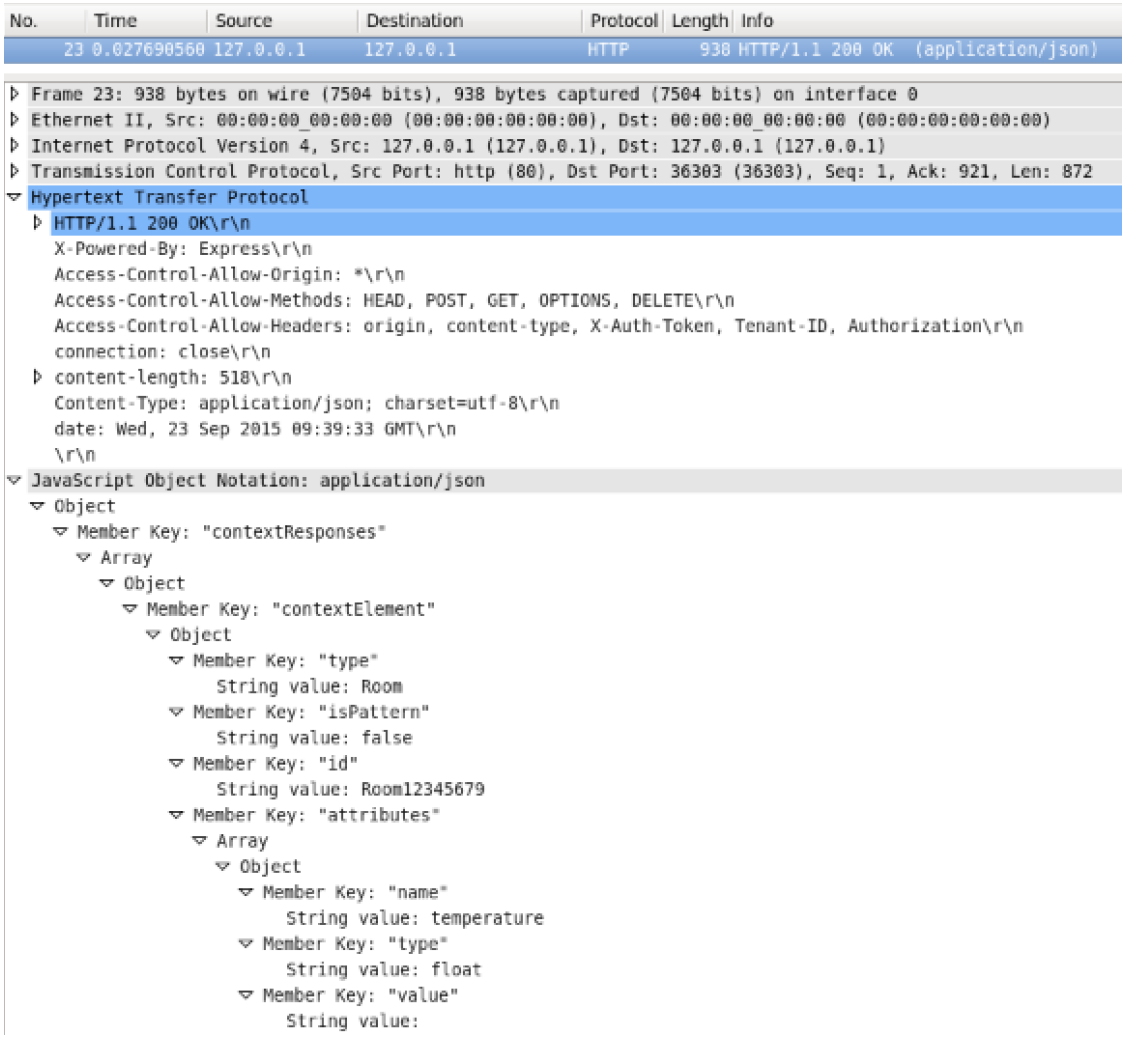

Figura 45 - Captura Wireshark respuesta de petición de contexto

Al igual que en la petición se observa que la respuesta devuelta por Wilma hacia el cliente es claro. En esta ocasión ya no contiene el X-Auth-Token al no ser necesario, este ya cumplió su función de validación.

Por último, podemos entrar a observar como se ha almacenado la nueva entidad dentro de la base de datos Mongo DB. Para el acceso a la misma se introducen los siguientes comandos:

```
mongo
show dbs
use orion
db.entities.find()
```

Figura 46 - Contenido de la base de datos Mongo DB

De esta forma queda testado el correcto funcionamiento del sistema completo. Se ha mostrado una interacción básica, pero cualquier otro tipo de petición de contexto de las soportadas por Orion Context Broker seguirá el mismo proceso.

5. Generación de una Máquina Virtual

5.1. Motivación

Tanto en el marco de este trabajo como en el que pueda ser desarrollado por el lector del mismo, puede surgir el interés de poder clonar la máquina virtual que alberga el core del sistema implementado, para así poder ser instalado en diferentes dispositivos sin tener la necesidad de repetir la instalación de cada herramienta de forma independiente.

Es por ello, que se cree conveniente explicar al menos de forma breve como se ha generado una una imagen del sistema para poder ser instalado en en el software de virtualiazación VirtualBox.

5.2. Convertir Parallels a VirtualBox

El sistema que se va a clonar es CentOS 6.6 albergado de forma virtual en Paralles Desktop. Para la clonación de la máquina es necesario que ésta no se encuentre en ejecución por lo que deberá de encontrarse el sistema CentOS 6.6. apagado en todo momento.

Una vez cerrado, se debe descargar una versión del software VMware Fusion[12]. Se instala y arrancamos dicho software. Seleccionamos Archivo -> Importar. Habrá que ir a la carpeta donde se encuentra ubicada la máquina virtual y escoger el archivo de extensión .pvm. Una vez escogido, pulsamos sobre el botón de Importar, escogemos el nombre, la localización de la nueva máquina virtual y pulsamos en Aceptar.

Una vez generada, debemos de ir a la localización donde fue creada. Pulsar botón derecho sobre el archivo y escoger la opción “mostrar contenido del paquete”. Debemos de crear una nueva carpeta en otro lugar, asignarla un nombre e introducir en ella todo el contenido que se nos acaba de mostrar.

Apagamos el software VMware Fusion y abrimos VirtualBox para crear la nueva máquina. Pulsaremos en “Nueva”, le asignaremos nombre y la memoria RAM que va a usar la máquina. En el siguiente paso, deberemos escoger la opción de “Usar un archivo de disco duro virtual existente”. Debemos ir entonces a la carpeta que creamos anteriormente, donde ubicamos todo el contenido y seleccionar aquel archivo de extensión .vmdk que no este enumerado o sino aquel que sea de valor cero. Pulsamos “siguiente” y por último “finalizar” [29].

Con esto ya hemos creado una copia exacta con la plataforma integrada de este proyecto perfectamente operativa.

[12] http://www.vmware.com/products/fusion/fusion-evaluation

6. Conclusiones y Líneas Futuras

6.1. Conclusiones

El trabajo "Integración de una plataforma para la gestión de contexto utilizando habilitadores genéricos FIWARE para la Internet de las Cosas" corresponde al trabajo de Fin de Grado en Ingeniería de Tecnologías de Telecomunicación en la Universidad de Cantabria durante el curso 2014-2015.

En este trabajo se han integrado diferentes habilitadores genéricos desarrollados por FIWARE para la provisión de un servicio de gestión de contexto con capacidades de autenticación. El resultado final es una guía de cómo se ha integrado esta herramienta paso a paso, así como las capacidades que se pueden esperar de ella.

Para extraer las conclusiones de este trabajo, es necesario regresar a los objetivos del mismo. Así, se podrá observar el nivel de satisfacción logrado con cada uno de estos objetivos y proporcionar así un análisis de los aspectos más destacables del mismo.

Como se definió al comienzo de este documento los objetivos que se han perseguido durante la realización de este trabajo de Fin de Grado se pueden resumir en los siguientes:

1. Integración de una plataforma de gestión de contexto.
2. Soporte de control y gestión de usuarios en la plataforma.
3. Comprensión de la plataforma FIWARE.
4. Testeo del sistema integrado.

Además, no tan relacionados con el producto final resultado del trabajo realizado sino mas bien como competencias adquiridas durante el proceso, es conveniente destacar los siguientes aspectos que también se plantearon como metas a completar las siguientes:

1. Familiarización con el uso de sistemas operativos de código abierto.
2. Adquisición de conocimientos relativos a los estándares implicados

El primer objetivo del proyecto se ha logrado de forma satisfactoria, ya que se ha conseguido integrar una plataforma de gestión de contexto operativa, a partir de habilitadores genéricos disponibles en el marco de FIWARE. El segundo objetivo va muy ligado al primero. Así como la plataforma de gestión de contexto ha sido correctamente integrada y es plenamente funcional, el segundo de los objetivos no se ha logrado en su totalidad. Esto se debe a que no se ha logrado integrar de forma completa en el sistema un IdM.

Si bien se ha logrado la correcta configuración y sincronismo de la plataforma con el IdM ofrecido por el FI-Lab estando este totalmente accesible y disponible para su uso. El inconveniente para no dar por cumplido de forma rotunda este objetivo es que la gestión del IdM cae sobre la plataforma FIWARE y hubiera sido mucho más recomendable haber integrado a través de las herramientas disponibles en OpenStack un IdM no vinculado a otras entidades y totalmente gestionado desde la propia plataforma integrada.

La compresión tanto de la filosofía de la plataforma FIWARE como de sus diferentes secciones es un objetivo que se ha logrado cumplir. Es notorio en el trabajo el tiempo que se ha tenido que invertir en la lectura y comprensión de diferentes documentos albergados en la plataforma para conseguir llevar a cabo el trabajo. Además, en este trabajo sea descrito de una forma clara aspectos de esta plataforma para así facilitar, en la medida de lo posible, la comprensión de la

plataforma FIWARE a los lectores de este documento. Uno de los inconvenientes que se puede relacionar con este objetivo es la constante actualización de documentos técnicos que ha sufrido la plataforma a lo largo de este trabajo. Puede que esto haya sido un inconveniente a la hora del desarrollo, pero sin duda esto otorga mayor empaque al uso de esta plataforma, demostrando que este trabajo se enmarca dentro de una plataforma en continua expansión y cuenta con el apoyo de gigantes de las telecomunicaciones.

Por último, en lo que se refiere a los objetivos directos del trabajo, se ha llevado acabo un testeo funcional exhaustivo de las herramientas integradas. Este testeo a permitido observar con detalle los procesos que siguen los diferentes habilitadores genéricos durante el acceso a sus servicios así como las interacciones entre los GE integrados en la plataforma.

En lo que se refiere a la adquisición de competencias necesarias para la consecución de los anteriores objetivos, el conocimiento de uso que se ha desarrollado a lo largo de este trabajo en el empleo de sistemas de código abierto ha sido extenso. Es verdad que son sistemas complejos que requieren un gran conocimiento, pero viendo el punto de partida se puede considerar más que cumplido el objetivo.

En relación con las bases teóricas de los estándares implicados para el desarrollo de este trabajo, se han aprendido los principales aspectos de aquellos estándares que no se conocían así como el repaso de aquellos que han formado parte de los estudios de Grado en Ingeniería de Tecnologías de Telecomunicación.

Como conclusión final referente a los objetivos, podemos decir que prácticamente se ha logrado cumplir todos ellos en su totalidad.

Un aspecto destacable del trabajo es que se trata de una aplicación final y siguiendo los pasos dados en este trabajo poder lograr desarrollar una herramienta propia. Es por ello que los resultados obtenidos tienen validez tanto en el mundo académico como empresarial. Si lo que se desea es conocimiento de las diferentes tecnologías albergadas en este trabajo se podrá encontrar unos buenos pilares. Si por el contrario se desea implementar dicha herramienta para darle un uso comercial, en este trabajo también se podrán encontrar los pasos necesarios para ello.

A nivel personal, mis conclusiones al respecto son favorables. Creo que en general se han logrado los objetivos con los que comenzó el trabajo. Además de haber desarrollado un trabajo útil e interesante en diferentes aspectos.

6.2. Líneas Futuras

A la vista de las conclusiones anteriores se puede hacer un resumen de las necesidades futuras que se abren tras la finalización de este trabajo.

La integración de un IdM gestionado en su totalidad por el administrador de la plataforma se considera una mejora del sistema muy recomendable. Para poder implementar esta mejora, se recomienda integrar dentro del sistema la API Keystone la cual es la componente principal del IdM KeyRock desarrollado por FIWARE. Si además se quisiera otorgar una interfaz gráfica a este IdM se recomienda acudir a otra herramienta de OpenStack llamada Horizon.

También, resultaría interesante poder integrar dentro del gestor de contexto una herramienta que fuese capaz de almacenar históricos. Como se describió en la introducción del Context Broker Orion, este gestor únicamente almacena la última actualización proveniente del productor de contexto. Así que, en estos momentos si se quisiera almacenar información de actualizaciones anteriores esa función debería de recaer en cada uno de los consumidores, generando ellos mismos su propia base de datos como crean conveniente. Si se quisiera agrupar

esa base de datos en una única, para la gestión de históricos de todo el sistema en común se podría recurrir a otro habilitador genérico que también proporciona FIWARE como es Cygnus[13]. Este habilitador genérico proporciona un sistema de persistencia de contexto estableciendo un enlace desde Orion a una de las base de datos con las que puede trabajar este habilitador y almacenando en ella de forma estructurada para su futura recolección la información proveniente de todos los productores de contexto [6].

[13] https://github.com/telefonicaid/fiware-cygnus/tree/master

7. Bibliografía

[1] ECMA-404. ECMAScript® Language Specification; “The JSON Data Interchange Format”. Octubre 2013

[2] Especificación abierta FIWARE sobre Gestor de Contexto https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.PubSub

[3] Especificación abierta FIWARE sobre IoT https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.IoT.Backend.ConfMan

[4] Github de Orion Context Broker https://github.com/telefonicaid/fiware-orion/blob/master/README.md

[5] Github de empleo de la instancia FIWARE-Lab https://github.com/ging/fiware-idm/wiki/Using-the-FIWARE-LAB-instance#authorization-code-grant

[6] GitHub de Cygnus https://github.com/telefonicaid/fiware-cygnus

[7] Guía de Administración y Instalación de Orion Context Broker https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Broker_-_Orion_Context_Broker_-_Installation_and_Administration_Guide

[8] Guía de Administración y Instalación de KeyRock http://fiware-idm.readthedocs.org/en/latest/admin_guide.html

[9] Guía de Administración y Instalación del PEP Proxy Wilma http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/PEP_Proxy_-_Wilma_-_Installation_and_Administration_Guide

[10] Guía de Creación de cuenta en FIWARE-Lab http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_Lab:_Upgrade_to_Community_Account

[11] Guía de Instalación de Mongo DB sobre CentOS http://docs.mongodb.org/manual/tutorial/install-mongodb-on-red-hat/

[12] Guía de Programador y Usuario de Orion Context Broker https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Broker_-_Orion_Context_Broker_-_User_and_Programmers_Guide

[13] Guía de Programador y Usuario del PEP Proxy Wilma http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/PEP_Proxy_-_Wilma_-_User_and_Programmers_Guide

[14] Información de comandos yum http://www.alcancelibre.org/staticpages/index.php/como-yum

[15] Información de HTTP http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177

[16] Información de OAuth2 http://www.thegameofcode.com/2012/07/conceptos-basicos-de-oauth2.html

[17] Información sobre servicios al Inicio del Sistema http://rm-rf.es/anadir-quitar-servicios-al-inicio-del-sistema-red-hat-centos/

[18] Modelo de información NGSI9 y NGSI10 https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model

[19] Página oficial de cursos FIWARE https://edu.fiware.org

[20] Página oficial de Future-Internet PPP https://www.fi-ppp.eu

[21] Página oficial de la documentación de KeyRock http://fiware-idm.readthedocs.org/en/latest/

[22] Página oficial de Orion Context Broker http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker

[23] Página oficial del IdM KeyRock http://catalogue.fiware.org/enablers/identity-management-keyrock

[24] RFC 2616. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee; "Hypertext Transfer Protocol -- HTTP/1.1". Junio 1999.

[25] RFC 6749. D. Hardt, Ed.; " The OAuth 2.0 Authorization Framework". Octubre 2012.

[26] RFC 7061. R. Sinnema y E. Wilde; "eXtensible Access Control Markup Language (XACML) XML Media Type". Noviembre 2013.

[27] RFC 7159. T. Bray, Ed.; " The JavaScript Object Notation (JSON) Data Interchange Format". Marzo 2014.

[28] Stackoverflow sobre PEP Wilma y KeyRock http://stackoverflow.com/questions/29628531/how-to-configure-the-fiware-pep-wilma-proxy-to-use-a-keyrock-and-orion-instance

[29] Tutorial de conversión Parallels a VirtualBox http://benfrain.com/osx-converting-parallels-or-vmware-to/

[30] Tutorial FIWARE NGSI APIv1 https://fiware-orion.readthedocs.org/en/develop/user/walkthrough_apiv1/index.html

8. Apéndice 1: Tutorial Orion

8.1. Introducción

Este tutorial adopta un enfoque práctico con el que se espera ayudar a los lectores a que se familiaricen con el Orion Context Broker y las diferentes operaciones que soporta. Este tutorial ha sido creado por la plataforma FIWARE. El documento original se puede encontrar en el enlace siguiente[14].

En este proyecto, se ha comprobado el correcto funcionamiento de las diferentes operaciones que en el documento FIWARE se describen para el uso en el sistema integrado de este trabajo.

Las dos primeras secciones tratan sobre la gestión de contexto utilizando NGSI10 y gestión de la disponibilidad contexto usando NGSI9. En ellas se describe la funcionalidad básica de Context Broker, tanto para la gestión de contexto (información sobre entidades, tales como la temperatura de un coche) y gestión de la disponibilidad de contexto (información no sobre las propias entidades, pero si acerca de los proveedores de esa información). Algunas observaciones a tener en cuenta para utilizar este material:

1) Gestión de contexto y gestión de disponibilidad contexto son funcionalidades independientes correspondientes a diferentes partes de la interfaz NGSI, NGS10 y NGSI9 respectivamente.

2) Tenga en cuenta que cada sección principal se divide en dos sub-secciones: la primera es acerca de las operaciones estándar, mientras que la segunda parte trata de operaciones de conveniencia. De hecho, cada sub-sección es un tutorial independiente que se pueden hacer paso a paso, simplemente copiando y pegando los comandos de este documento.

3) Antes de iniciar el tutorial, reinicie Orion Context Broker.

4) Se recomienda comenzar con la parte de las operaciones estándar y luego hacer la parte de las operaciones de conveniencia ya que son necesarias algunas explicaciones y conceptos descritos en la primera parte para comprender la segunda.

Se recomienda familiarizarse con algunos conceptos teóricos del modelo NGSI en el que se basa antes de comenzar. Por ejemplo entidades, atributos, etc.

8.2. Antes de Empezar

Antes de empezar, vamos a introducir el caso del ejemplo que se utiliza durante los tutoriales y cómo ejecutar e interactuar con el Orion Context Broker.

8.2.1 Ejemplo Práctico:

Supongamos que tenemos un edificio con varias habitaciones y que queremos utilizar Orion Context Broker para gestionar su información de contexto. Las habitaciones son Room1, Room2, Room3 y Room4 y cada habitación tiene dos sensores: la temperatura y la presión (atmosférica) (excepto Room4 que sólo cuenta con un sensor de presión). Además, vamos a considerar que tenemos dos coches (Car1 y Car2) el primero de ellos con un sensor de velocidad y el segundo con un sensor de ubicación.

[14] https://fiware-orion.readthedocs.org/en/develop/user/walkthrough_apiv1/index.html

La mayoría de las veces vamos se va a utilizar únicamente Room1 y Room2 en el tutorial. Room3, Room4, Car1 y Car2 solo se utilizarán en la sección relativa a las suscripciones de disponibilidad de contexto.

Orion Context Broker interactúa con aplicaciones productoras contexto (que proporcionan información de los sensores) y una aplicación de consumidor de contexto (que procesa esa información, por ejemplo, para mostrar en una interfaz gráfica de usuario). Vamos a jugar el papel de ambos tipos de aplicaciones en este tutorial.

Una vez iniciado el Broker y borrado su base de datos como se describe en este trabajo en el capitulo de instalación de Orion Context Broker. Se debe iniciar el servidor de acumulación incluido en el contextBroker-test. Es un servidor de acumulación muy sencillo, tan sólo escucha una URL determinada (vamos a emplear localhost:1028/accumulate) e imprime lo que se interpone en la ventana de terminal en el que se ejecuta.Se ejecuta mediante el siguiente comando:

```
# cd /dir/where/accumulator-server/is/downloaded
# chmod a+x accumulator-server.py
# ./accumulator-server.py 1028 /accumulate ::1 on
```

8.2.2. Comandos del Broker:

Para emitir peticiones al Broker se utiliza la herramienta de línea de comandos CURL. Se eligió CURL porque es casi omnipresente en cualquier sistema GNU/Linux y simplifica la inclusión de ejemplos en este documento ya que se puede copiar y pegar con sencillez. Por supuesto, no es obligatorio el uso de CURL, se puede utilizar cualquier otra herramienta de cliente REST en su lugar (por ejemplo RESTClient). De hecho, en un caso real, es probable que se interactúe con el Orion Context Broker usando una biblioteca de lenguaje de programación que implemente REST en la aplicación del cliente.

Los patrones básicos para todos los ejemplos CURL en este documento son los siguientes:

Para POST:

```
curl localhost:1026/<operation_url> -s -S [headers]' -d @- <<EOF
[payload]
EOF
```

Para PUT:

```
(curl localhost:1026/<operation_url> -s -S [headers] -X PUT -d @- <<EOF
[payload]
EOF
```

Para GET:

```
curl localhost:1026/<operation_url> -s -S [headers]'
```

Para DELETE:

```
curl localhost:1026/<operation_url> -s -S [headers] -X DELETE
```

En cuanto a las [headers] que se tienen que incluir son las siguientes:

La cabecera Accept para especificar el formato de la carga útil que desea recibir en la respuesta. Se debe especificar explícitamente JSON.

```
curl ... --header 'Accept: application/json' …
```

Sólo en el caso de uso de carga útil en la solicitud (es decir, POST o PUT), tiene que utilizar la cabecera Context-Type para especificar el formato (JSON).

```
curl ... --header 'Content-Type: application/json' …
```

8.2.3. Operaciones contenidas en el tutorial

En la siguiente tabla se pueden ver todas las operaciones que se realizan en este tutorial. El propósito no es otro que resumir de forma gráfica lo que se va a encontrar en las siguientes hojas de este documento y hacer una pequeña introducción a las mismas.

Operación	Descripción	NGSI	Testada
updateContext (APPEND)	Creación de una entidad	NGSI10	✔
updateContext (UPDATE)	Actualización de una entidad	NGSI10	✔
queryContext	Consulta de contexto	NGSI10	✔
subscribeContext (ONTIMEINTERVAL)	Subscripción de contexto, cada cierto tiempo se recibe información	NGSI10	✔
subscribeContext (ONCHANGE)	Subscripción de contexto, se recibe información cuando varía el valor	NGSI10	✔
updateContextSubscription	Actualización de la subscripción	NGSI10	✔
unsubscribeContext	Eliminar una subscripción	NGSI10	✔
registerContext	Registro de disponibilidad de contexto	NGSI9	✔
discoverContextAvailability	Descubrimiento de disponibilidad de contexto	NGSI9	✔
subscribeContextAvailability	Subscripción de disponibilidad de contexto	NGSI9	✔
updateContextAvailabilitySubscription	Actualización de disponibilidad de subscripción de contexto	NGSI9	✔
unsubscribeContextAvailability	Eliminar la subscripción a la disponibilidad de contexto	NGSI9	✔

8.3. Gestión de Contexto usando NGSI10

8.3.1.Operaciones Estándar de NGSI10:

Esta sección muestra las diferentes operaciones estándar NGSI10 que el Orion Context Broker soporta, mostrando ejemplos de solicitudes y respuestas. Utilizamos el término "estándar" a las operaciones que se derivan directamente de la especificación OMA NGSI, para distinguirlas de las operaciones de conveniencia que han sido definidas por el proyecto FIWARE para facilitar el uso de la implementación NGSI.

Al final de esta sección, se tendrá los conocimientos básicos para crear aplicaciones (tanto en los productores como en los consumidores de contexto) utilizando Orion Context Broker con NGSI10 y sus operaciones estándar:

- updateContext
- queryContext

- subscribeContext
- updateContextSubscription
- unsubscribeContext

a) Creación de una Entidad:

Orion Context Broker comenzará en un estado vacío, por lo que en primer lugar tenemos que hacer que sea consciente de la existencia de ciertas entidades. En particular, vamos a "crear" Room1 y Room2 , cada una de ellas contendrán dos atributos (temperatura y presión). Hacemos esto mediante la operación updateContext con el tipo de acción APPEND (el otro tipo de acción principal, UPDATE, se comentara en una próxima sección).

En primer lugar, vamos a crear Room1. Supongamos sus atributos de temperatura y presión son 23 ºC y 720 mmHg, respectivamente.

```
 (curl localhost:1026/v1/updateContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1",
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": "23"
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": "720"
          }
      ]
      }
  ],
  "updateAction": "APPEND"
  }
  EOF
```

La petición updateContext contiene en su carga útil una lista de elementos contextElement. Cada contextElement está asociado a una entidad (cuya identificación se proporciona en el elemento entityId, en este caso su identificación es Room1) y contiene una lista de elementos contextAttribute. Cada contextAttribute proporciona el valor para un atributo dado (identificado por nombre y tipo) de la entidad. Además de la lista de elementos contextElement, la carga útil incluye también un elemento updateAction. Utilizamos APPEND, lo que significa que queremos añadir nueva información.

Orion Context Broker no realiza ninguna comprobación del tipo de variable (por ejemplo, no comprueba que cuando una aplicación de producción de contexto actualiza el valor de la temperatura, este valor tiene el formato de un float como "25.5" o "-40.23" y no algo como “caliente").Tras la recepción de esta solicitud, el Broker creará la entidad en su base de datos interna, estableciendo los valores de sus atributos y la respuesta a las siguientes peticiones:

```
{
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
```

```
            {
                "name": "temperature",
                "type": "float",
                "value": ""
            },
            {
                "name": "pressure",
                "type": "integer",
                "value": ""
            }
            ],
            "id": "Room1",
            "isPattern": "false",
            "type": "Room"
        },
        "statusCode": {
            "code": "200",
            "reasonPhrase": "OK"
        }
        }
    ]
}
```

Como se puede ver, se sigue la misma estructura que en la solicitud, aunque sólo al reconocer que la solicitud fue procesada correctamente para estos elementos de contexto. Como se puede observar los elementos contextValue están vacíos, esto se debe a que no es necesario mostrar los valores en la respuesta ya que es la única persona que los proporcionó en la solicitud. A continuación, vamos a crear Room2 de manera similar, en este caso estableciendo la temperatura y la presión a 21 ºC y 711 mmHg respectivamente.

```
(curl localhost:1026/v1/updateContext -s -S --header 'Content-Type: application/json'
--header 'Accept: application/json' -d @- | python -mjson.tool ) <<EOF
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room2",
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": "21"
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": "711"
          }
      ]
      }
  ],
  "updateAction": "APPEND"
  }
  EOF
```

La respuesta a esta solicitud es:

```
{
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
```

```
                "name": "temperature",
                "type": "float",
                "value": ""
            },
            {
                "name": "pressure",
                "type": "integer",
                "value": ""
            }
            ],
            "id": "Room2",
            "isPattern": "false",
            "type": "Room"
        },
        "statusCode": {
            "code": "200",
            "reasonPhrase": "OK"
        }
    }
    ]
}
```

Además de los valores simples (como strings) para valores de atributos, también puede utilizar estructuras complejas o metadatos personalizados. Estos son temas avanzados que se describen en esta sección[15] y esta otra[16], respectivamente.

b) Operaciones QueryContext:

Ahora vamos a realizar el papel de una aplicación de consumo, con ganas de tener acceso a la información de contexto almacenada por Orion Contexto Broker para hacer algo interesante con ella (por ejemplo, mostrar una gráfica con la temperatura ambiente en una interfaz gráfica). La solicitud NGSI10 queryContext se utiliza en este caso, para obtener información de contexto de Room1:

```
 (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json'
--header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ]
  }
  EOF
```

La respuesta incluye todos los atributos pertenecientes a Room1 y podemos comprobar que la temperatura y la presión tienen los valores que nos fijamos en la creación entidad con updateContext (23ºC y 720 mmHg).

```
{
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
```

[15] https://fiware-orion.readthedocs.org/en/develop/user/structured_attribute_valued/index.html#structured-attribute-values

[16] https://fiware-orion.readthedocs.org/en/develop/user/metadata/index.html#custom-attribute-metadata

```
            "name": "temperature",
            "type": "float",
            "value": "23"
        },
        {
            "name": "pressure",
            "type": "integer",
            "value": "720"
        }
        ],
        "id": "Room1",
        "isPattern": "false",
        "type": "Room"
    },
    "statusCode": {
        "code": "200",
        "reasonPhrase": "OK"
    }
    }
]
}
```

Si utiliza un elemento de atributos vacío en la solicitud, la respuesta incluirá todos los atributos de la entidad. Si se incluye una lista con los atributos que se solicitan (por ejemplo, temperatura) solamente se recuperan dichos atributos, como se muestra en la siguiente petición:

```
 (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json'
--header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ],
  "attributes": [
      "temperature"
  ]
  }
  EOF
```

Cuya respuesta es la siguiente:

```
 {
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
              "name": "temperature",
              "type": "float",
              "value": "23"
          }
          ],
          "id": "Room1",
          "isPattern": "false",
          "type": "Room"
      },
      "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
        }
      }
```

```
  ]
  }
```

Por otra parte, una función potente de Orión Context Broker es que se puede utilizar una expresión incompleta para el ID de la entidad. Por ejemplo, puedes consultar las entidades que su ID comienza con "Room" usando la expresión regular “Room.*". En este caso, se tiene que fijar isPattern a "true", como se muestra a continuación:

```
 (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/json'
--header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      },
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room2"
      }
  ],
  "attributes": [
      "temperature"
  ]
  }
  EOF

  (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "true",
      "id": "Room.*"
      }
  ],
  "attributes": [
      "temperature"
  ]
  }
  EOF
```

Ambas peticiones producen la misma respuesta:

```
 {
  "contextResponses": [
    {
    "contextElement": {
          "attributes": [
       {
             "name": "temperature",
             "type": "float",
             "value": "23"
       }
         ],
         "id": "Room1",
         "isPattern": "false",
         "type": "Room"
      },
      "statusCode": {
          "code": "200",
```

```
            "reasonPhrase": "OK"
        }
        },
        {
        "contextElement": {
            "attributes": [
            {
                "name": "temperature",
                "type": "float",
                "value": "21"
            }
            ],
            "id": "Room2",
            "isPattern": "false",
            "type": "Room"
        },
        "statusCode": {
            "code": "200",
            "reasonPhrase": "OK"
        }
        }
    ]
    }
```

Por último, se deber de tener en cuenta que se recibirá un error en el caso en el que se intente consultar una entidad o un atributo que no exista, como se muestra en los siguientes casos siguientes:

```
   (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
    {
    "entities": [
        {
        "type": "Room",
        "isPattern": "false",
        "id": "Room5"
        }
    ]
    }
    EOF

    (curl localhost:1026/v1/queryContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
    {
    "entities": [
        {
        "type": "Room",
        "isPattern": "false",
        "id": "Room1"
        }
    ],
    "attributes": [
        "humidity"
    ]
    }
    EOF
```

Ambas peticiones producirían la misma respuesta de error:

```
 {
    "errorCode": {
        "code": "404",
        "reasonPhrase": "No context elements found"
    }
```

```
}
```

Comentarios adicionales

Se debe de tener en cuenta que por defecto sólo se devuelven 20 entidades. Para cambiar este comportamiento, consulte la sección sobre la paginación del siguiente manual[17].

c) Elementos UpdateContext:

Se puede actualizar el valor de los atributos de las entidades mediante la operación updateContext. La regla básica a tener en cuenta en updateContext es que APPEND crea nuevos elementos de contexto, mientras que UPDATE actualiza los elementos de contexto ya existentes.

Ahora vamos imaginar una aplicación productora de contexto, es decir, una fuente de información de contexto. Vamos a suponer que esta aplicación en un momento dado quiere ajustar la temperatura y la presión del Room1 a 26,5 ºC y 763 mmHg, respectivamente, por lo que emite la petición siguiente:

```
 (curl localhost:1026/v1/updateContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1",
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": "26.5"
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": "763"
          }
      ]
      }
    ],
    "updateAction": "UPDATE"
  }
  EOF
```

Como se puede ver, la estructura de la solicitud es igual a la que utilizamos para updateContext con APPEND para la creación de entidades, salvo que utilizamos UPDATE ahora como tipo de acción.

Tras la recepción de esta solicitud, el Broker actulizará los valores de los atributos de la entidad en su base de datos y la responderá con lo siguiente:

```
{
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
              "name": "temperature",
```

[17] https://fiware-orion.readthedocs.org/en/develop/user/pagination/index.html#pagination

```
                "type": "float",
                "value": ""
            },
            {
                "name": "pressure",
                "type": "integer",
                "value": ""
            }
            ],
            "id": "Room1",
            "isPattern": "false",
            "type": "Room"
        },
        "statusCode": {
            "code": "200",
            "reasonPhrase": "OK"
        }
        }
    ]
    }
```

Una vez más, la estructura de la respuesta es igual a la que se usa en updateContext con APPEND para la creación de entidades. La operación updateContext es bastante flexible, ya que permite actualizar tantas entidades y atributos como se quiera: es solo una cuestión de que contextElements se incluyen en la lista. Se podría incluso actualizar toda la base de datos de Orion Broker Context (tal vez incluyendo a miles de entidades / atributos) en una sola operación updateContext.

Para ilustrar esta flexibilidad, se muestra cómo actualizar Room2 en dos solicitudes separadas updateContext (ajuste de la temperatura a 27,4 ºC y su presión a 755 mmHg), cada una modificando sólo un atributo. Esto también demuestra que no es necesario incluir todos los atributos de una entidad en el updateContext, sólo los que desea actualizar (el resto de atributos mantienen su valor actual).

```
 (curl localhost:1026/v1/updateContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room2",
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": "27.4"
          }
      ]
      }
    ],
    "updateAction": "UPDATE"
  }
  EOF
  (curl localhost:1026/v1/updateContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "contextElements": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room2",
      "attributes": [
          {
```

```
            "name": "pressure",
            "type": "integer",
            "value": "755"
            }
        ]
        }
  ],
  "updateAction": "UPDATE"
  }
  EOF
```

Las respuestas a estas solicitudes son, respectivamente:

```
{
  "contextResponses": [
      {
        "contextElement": {
          "attributes": [
          {
              "name": "temperature",
              "type": "float",
              "value": ""
          }
          ],
          "id": "Room2",
          "isPattern": "false",
          "type": "Room"
      },
      "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
      }
      }
  ]
  }

  {
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
              "name": "pressure",
              "type": "integer",
              "value": ""
          }
          ],
          "id": "Room2",
          "isPattern": "false",
          "type": "Room"
      },
      "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
      }
      }
  ]
  }
```

Ahora, se podría utilizar la operación queryContext como se describió anteriormente para comprobar que Room1 y Room2 han actualizado sus atributos. Además de los valores simples

(strigns) para los valores de los atributos, también se pueden utilizar estructuras complejas. De este tema se puede encontrar mas información en la siguiente sección[18].

Por último, puede utilizar REPLACE como updateAction. En este caso, los atributos de la entidad se sustituyen por los de la solicitud. Por ejemplo, si su entidad tiene los atributos de A y B y le enviáramos una solicitud updateContext REPLACE de A, entonces la entidad resultante solo contendría el atributo A (es decir, se elimina el atributo B).

d) Subscripciones de contexto:

Las operaciones NGSI10 que conoces hasta ahora (updateContext y queryContext) son los bloques básicos de construcción para aplicaciones síncronas de consumo y producción de contexto. Sin embargo, Orion Context Broker tiene otra característica importante con la que se puede suscribirse a información de contexto para que cuando "algo" suceda (se explicaran los diferentes casos de ese "algo") su solicitud recibirá una notificación asíncrona. De esa manera, no es necesario repetir continuamente peticiones queryContext, Orion Context Broker le hará saber la información solicitada sin necesidad de repetir la solicitud.

Antes de comenzar a emplear dicha función, inicie el servidor de acumulación para capturar notificaciones.

En realidad, hay dos tipos de subscribeContext: suscripciones ONTIMEINTERVAL y ONCHANGE las cuales se describen a continuación en las próximas dos subsecciones.

El estándar NGSI describe un tercer tipo de suscripción, llamado ONVALUE, pero la versión actual del Orion Context Broker no lo soporta.

ONTIMEINTERVAL

La siguiente es la solicitud correspondiente a una suscripción ONTIMEINTERVAL:

```
(curl localhost:1026/v1/subscribeContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ],
  "attributes": [
      "temperature"
  ],
  "reference": "http://localhost:1028/accumulate",
  "duration": "P1M",
  "notifyConditions": [
      {
      "type": "ONTIMEINTERVAL",
      "condValues": [
          "PT10S"
        ]
      }
  ]
  }
  EOF
```

[18] https://fiware-orion.readthedocs.org/en/develop/user/structured_attribute_valued/index.html#structured-attribute-values

Vamos a examinar en detalle los diferentes elementos incluidos en la carga útil:

entityIdList y AttributeList ('entidades' y 'atributos' para abreviar, en JSON) definen qué elementos de contexto se incluyen en el mensaje de notificación. En este ejemplo, estamos especificando que la notificación tiene que incluir el atributo temperatura en la entidad Room1.

La URL donde enviar las notificaciones se define como un elemento de referencia. Estamos utilizando la dirección URL del programa accumulator-server.py incivilizado antes. Las suscripciones tienen una duración específica utilizando el formato estándar ISO 8601. Una vez que ha caducado, la suscripción se ingora (sin embargo, seguirá almacenada en la base de datos del Broker y necesitará ser eliminada usando el procedimiento descrito en el manual de administración).

Se puede extender la duración de la suscripción actualizandola. En este caso estamos utilizando "P1M", que significa "un mes" de subscripción.

El elemento notifyCondition define el "disparador" para la suscripción. Hay un elemento de tipo (cuyo valor en este caso es ONTIMEINTERVAL) y un elemento condValueList. La estructura de elemento condValueList depende del tipo. En el caso de ONTIMEINTERVAL, incluye exactamente un elemento hijo condValue cuyo valor es un intervalo de tiempo (usando de nuevo, como es habitual en NGSI, el formato ISO 8601). Se envía una notificación con una frecuencia igual a ese intervalo. En el ejemplo anterior estamos utilizando 10 segundos como intervalo.

La respuesta correspondiente a la solicitud contiene un identificador de suscripción (un número hexadecimal utilizado para la actualización y cancelación de la suscripción) y un reconocimiento duración:

```
  {
  "subscribeResponse": {
      "duration": "P1M",
      "subscriptionId": "51c04a21d714fb3b37d7d5a7"
  }
  }
```

Si se observa la ventana de terminal accumulator-script.py, se verá un mensaje parecido al siguiente, el cual se recibe cada 10 segundos:

```
POST http://localhost:1028/accumulate
  Content-Length: 492
  User-Agent: orion/0.9.0
  Host: localhost:1028
  Accept: application/xml, application/json
  Content-Type: application/json
  {
  "subscriptionId": "51c04a21d714fb3b37d7d5a7",
  "originator": "localhost",
  "contextResponses": [
      {
      "contextElement": {
          "attributes": [
          {
              "name": "temperature",
              "type": "float",
              "value": "26.5"
          }
          ],
          "type": "Room",
          "isPattern": "false",
          "id": "Room1"
      },
      "statusCode": {
```

```
            "code": "200",
            "reasonPhrase": "OK"
        }
        }
  ]
  }
```

Orion Context Broker notifica con subscribeContext en NGSI10 utilizando el método HTTP POST (con la URL utilizada como referencia en la suscripción) con una carga útil notifyContextRequest. Aparte del elemento subscriptionId (el mismo que el de la respuesta a la solicitud subscribeContext) y el elemento iniciador, hay un elemento contextResponseList el cual es el mismo que el utilizado en las respuestas queryContext.

Las suscripciones se pueden actualizar mediante updateContextSubcription en NGSI10. La solicitud incluye un subscriptionId que identifica la suscripción a modificar y la carga útil de actualización. Por ejemplo, si queremos cambiar el intervalo de notificación a 5 segundos, utilizaremos el siguiente (sustituir el valor subscriptionId con el que se devolvió en su respuesta subscribeContext en el paso anterior) comando:

```
(curl localhost:1026/v1/updateContextSubscription -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "subscriptionId": "51c04a21d714fb3b37d7d5a7",
  "notifyConditions": [
      {
      "type": "ONTIMEINTERVAL",
      "condValues": [
          "PT5S"
      ]
      }
  ]
  }
  EOF
```

La respuesta es muy similar a la de la solicitud subscribeContext:

```
 {
  "subscribeResponse" : {
    "subscriptionId" : "51c04a21d714fb3b37d7d5a7",
  }
  }
```

Se puede comprobar en accumulator-server.py que la frecuencia de notificación ha cambiado a 5 segundos.

Por último, se puede cancelar una suscripción usando la operación NGSI10 unsubscribeContext, que sólo utiliza subscriptionId en la solicitud de carga útil:

```
 (curl localhost:1026/v1/unsubscribeContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "subscriptionId": "51c04a21d714fb3b37d7d5a7"
  }
  EOF
```

La respuesta es simplemente un reconocimiento de que la cancelación se ha realizado correctamente:

```
{
```

```
    "statusCode": {
    "code": "200",
    "reasonPhrase": "OK"
    },
    "subscriptionId": "51c04a21d714fb3b37d7d5a7"
  }
```

Se puede observar en el acumulador accumulator-server.py que el flujo de notificación se ha detenido.
ONCHANGE

Suponemos que el programa accumulator-server.py todavía se está ejecutando. De lo contrario, iniciarlo de nuevo como se describió anteriormente.

Las suscripciones ONCHANGE se utilizan cuando se desea ser notificado no cuando un determinado intervalo de tiempo ha pasado sino cuando se han producido algunos cambios de los atributos. Consideremos el siguiente ejemplo:

```
 (curl localhost:1026/v1/subscribeContext -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ],
  "attributes": [
      "temperature"
  ],
  "reference": "http://localhost:1028/accumulate",
  "duration": "P1M",
  "notifyConditions": [
      {
      "type": "ONCHANGE",
      "condValues": [
          "pressure"
      ]
      }
  ],
  "throttling": "PT5S"
  }
  EOF
```

Si se observa la carga útil podemos comprobar que es muy similar a la utilizada en ONTIMEINTERVAL, con dos excepciones:

El elemento notifyCondition utiliza el tipo ONCHANGE pero, en este caso el condValueList contiene una lista actual de elementos condValue, cada una con un nombre de atributo. Definen los "atributos de activación", es decir, los atributos que, tras la creación o cambio activan la notificación. La regla es que si alguno de esos atributos cambia de valor, a continuación, se envía una notificación. Se debe de tener en cuenta que la notificación incluye los atributos en la parte AttributeList, que no deben incluir necesariamente ningún atributo en el condValue.

El elemento “throttling” se utiliza para especificar un tiempo mínimo entre la llegada de notificación. Así, el establecimiento de limitación a 5 segundos como en el ejemplo anterior hace que la notificación no se envíe si una notificación anterior se envío hace menos de 5 segundos, no importa cuántos cambios reales tienen lugar en ese período. Esto se hace para no saturar el receptor de notificaciones en caso de tener productores de contexto que actualizan los valores de sus atributos con demasiada frecuencia. En realidad, la limitación no es un campo exclusivo para suscripciones ONCHANGE: desde un punto de vista teórico puede ser utilizado en las

suscripciones ONTIMEINTERVAL pero, dado que en ese caso se puede controlar con precisión la frecuencia de notificación no tiene ningún sentido práctico.

Al igual que en las suscripciones ONTIMEINTERVAL, la respuesta consiste en una ID de suscripción, un reconocimiento duración y ademas un reconocimiento de “throttling”:

```
{
  "subscribeResponse": {
      "duration": "P1M",
      "subscriptionId": "51c0ac9ed714fb3b37d7d5a8",
      "throttling": "PT5S"
  }
  }
```

Si se observa ahora el accumulator-server.py. veremos una (y sólo una por el momento, no importa cuánto se espere) notifyContextRequest, similar a este:

```
POST http://localhost:1028/accumulate
Content-Length: 492
User-Agent: orion/0.9.0
Host: localhost:1028
Accept: application/xml, application/json
Content-Type: application/json

{
"subscriptionId": "51c0ac9ed714fb3b37d7d5a8",
"originator": "localhost",
"contextResponses": [
    {
    "contextElement": {
        "attributes": [
        {
            "name": "temperature",
            "type": "float",
            "value": "26.5"
        }
        ],
        "type": "Room",
        "isPattern": "false",
        "id": "Room1"
    },
    "statusCode": {
        "code": "200",
        "reasonPhrase": "OK"
    }
    }
]
}
```

Se puede preguntar por qué accumulator-server.py está recibiendo este mensaje si en realidad no se ha realizado ninguna actualización. Esto es porque Orion Context Broker considera la transición de "suscripción no existente" a "suscrito" como un cambio.

e) Resumen de operaciones estándar NGSI10 sobre URL:

/v1/updateContext
/v1/queryContext
/v1/subscribeContext
/v1/updateContextSubscription
/v1/unsubscribeContext

8.3.2.Operaciones de Conveniencia de NGSI10:

En esta sección se describen las distintas operaciones de conveniencia redactadas como parte de FIWARE NGSI REST API NGSI10 que Orión Context Broker apoya, mostrando ejemplos de solicitudes y respuestas. Las operaciones de conveniencia son un conjunto de operaciones que se han definido por FIWARE para facilitar el uso de implementaciones NGSI como complemento de las operaciones estándar definidos en la especificación OMA NGSI.

Se debe de reiniciar el Broker antes de comenzar con el tutorial de esta sección como se ha descrito anteriormente en este documento. Al final de esta sección, se habrá familiarizado con el uso de las operaciones de conveniencia como una alternativa útil para algunas operaciones estándar descritas en el apartado anterior.

a) Creación de una Entidad mediante operaciones de conveniencia:

Orion Context Broker comienza en un estado vacío, por lo que en primer lugar se debe hacer que sea consciente de la existencia de ciertas entidades. Por tanto, vamos a crear una primera entidad Room1 con atributos de temperatura y presión (con sus valores iniciales):

```
 (curl localhost:1026/v1/contextEntities/Room1 -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -X POST -d @- | python -
mjson.tool) <<EOF
  {
  "attributes": [
      {
      "name": "temperature",
      "type": "float",
      "value": "23"
      },
      {
      "name": "pressure",
      "type": "integer",
      "value": "720"
      }
  ]
  }
  EOF
```

La respuesta es:

```
 {
  "contextResponses": [
      {
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": ""
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": ""
          }
      ],
      "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
      }
      }
  ],
  "id": "Room1",
```

```
  "isPattern": "false",
  "type": ""
  }
```

Ahora haremos lo mismo con Room2:

```
  (curl localhost:1026/v1/contextEntities/Room2 -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -X POST -d @- | python -
mjson.tool) <<EOF
  {
  "attributes": [
      {
      "name": "temperature",
      "type": "float",
      "value": "21"
      },
      {
      "name": "pressure",
      "type": "integer",
      "value": "711"
      }
  ]
  }
```

La respuesta es:

```
 {
  "contextResponses": [
      {
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": ""
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": ""
          }
        ],
        "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
      }
      }
  ],
  "id": "Room2",
  "isPattern": "false",
  "type": ""
  }
```

También se puede crear un atributo (y la entidad que contiene) de la siguiente manera:

```
 (curl localhost:1026/v1/contextEntities/Room3/attributes/temperature -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -X POST -d @- |
python -mjson.tool) <<EOF
  {
  "value" : "21"
  }
  EOF
```

En comparación con la creación de una entidad según la base de funcionamiento estándar observamos las siguientes diferencias:

Se esta utilizando el verbo POST en el /v1/contextEntities/{EntityID} recurso para crear nuevas entidades. No se puede crear más de una entidad a la vez con las solicitudes de operación de conveniencia. La capacidad de carga útil en las peticiones y respuestas de las operaciones de conveniencia son muy similares a las utilizadas en las operaciones estándar, ya que los elementos contextAttributeList y contextResponseList son los mismos.

b) Query Context como operación de conveniencia

Se va a describir las operaciones de conveniencia para la consulta de información de contexto. Podemos consultar todos los valores de los atributo en una entidad determinada, por ejemplo, Room1:

```
curl localhost:1026/v1/contextEntities/Room1 -s -S --header 'Accept: application/
json' | python -mjson.tool
```

Cuya respuesta es:

```
{
 "contextElement": {
     "attributes": [
     {
         "name": "temperature",
         "type": "float",
         "value": "23"
     },
     {
         "name": "pressure",
         "type": "integer",
         "value": "720"
     }
     ],
     "id": "Room1",
     "isPattern": "false",
     "type": "Room"
 },
 "statusCode": {
     "code": "200",
     "reasonPhrase": "OK"
 }
 }
```

También podemos consultar un único atributo de una entidad determinada, por ejemplo, la temperatura de Room2:

```
 curl localhost:1026/v1/contextEntities/Room2/attributes/temperature -s -S --header
'Accept: application/json' | python -mjson.tool
```

Su respuesta es:

```
{
 "attributes": [
     {
     "name": "temperature",
     "type": "float",
     "value": "21"
     }
 ],
 "statusCode": {
     "code": "200",
     "reasonPhrase": "OK"
 }
 }
```

En comparación con la operación queryContext estándar observamos las siguientes diferencias:

Las operaciones de conveniencia utilizan el método GET y no usan carga útil en la solicitud (más simple que la operación estándar). El elemento contextElementResponse respuesta usada en la respuesta de la operación de conveniencia para consultar todos los atributos de una entidad tiene la misma estructura que la que aparece en el interior de las respuestas de queryContext estándar. Sin embargo, el elemento contextAttributeResponse en la respuesta de la operación de conveniencia utiliza como respuesta a la consulta un único atributo de una entidad.

También se pueden consultar todas las entidades pertenecientes a un mismo tipo, o bien todos los atributos o uno en particular, como se muestra a continuación.

Petición para conseguir todos los atributos:

```
curl localhost:1026/v1/contextEntityTypes/Car -s -S --header 'Accept: application/
json' | python -mjson.tool
```

Petición para conseguir un único atributo (p.e. speed):

```
curl localhost:1026/v1/contextEntityTypes/Car/attributes/speed -s -S --header
'Accept: application/json' | python -mjson.tool
```

Conseguir todas las entidades:

Se pueden obtener todas las entidades mediante la siguiente operación de conveniencia:

```
curl localhost:1026/v1/contextEntities -s -S --header 'Content-Type: application/
json' --header 'Accept: application/json' | python -mjson.tool
```

En este caso se devuelve Room1 y Room2:

```
{
    "contextResponses": [
        {
            "contextElement": {
                "attributes": [
                    {
                        "name": "temperature",
                        "type": "float",
                        "value": "23"
                    },
                    {
                        "name": "pressure",
                        "type": "integer",
                        "value": "720"
                    }
                ],
                "id": "Room1",
                "isPattern": "false",
                "type": ""
            },
            "statusCode": {
                "code": "200",
                "reasonPhrase": "OK"
            }
        },
        {
            "contextElement": {
                "attributes": [
                    {
                        "name": "temperature",
                        "type": "float",
```

```
                        "value": "21"
                    },
                    {
                        "name": "pressure",
                        "type": "integer",
                        "value": "711"
                    }
                ],
                "id": "Room2",
                "isPattern": "false",
                "type": ""
            },
            "statusCode": {
                "code": "200",
                "reasonPhrase": "OK"
            }
        }
    ]
}
```


Comentarios adicionales:

Conseguir todas las entidades almacenadas en Orion no es buena idea (excepto si usted tiene un número limitado de entidades). Echa un vistazo a la sección sobre filtros[19]. Tenga en cuenta que, por defecto, sólo 20 entidades son devueltas.

c) Explorar todos los tipos y la información detallada sobre un tipo:

La siguiente operación se puede utilizar para obtener una lista de todos los tipos de entidades existentes en Orion Context Broker en un momento dado:

```
curl localhost:1026/v1/contextTypes -s -S --header 'Content-Type: application/json'
--header 'Accept: application/json' | python -mjson.tool
```


```
  {
  "statusCode": {
      "code": "200",
      "reasonPhrase": "OK"
  },
  "types": [
      {
      "attributes": [
          "speed",
          "fuel",
          "temperature"
      ],
      "name": "Car"
      },
      {
      "attributes": [
          "pressure",
          "hummidity",
          "temperature"
      ],
      "name": "Room"
      }
  ]
  }
```


Como se puede ver, la información de atributo se proporciona para cada tipo. Algunas observaciones importantes:

[19] https://fiware-orion.readthedocs.org/en/develop/user/filtering/index.html#filters

Dado que NGSI no obliga a todas las entidades de un tipo determinado a tener el mismo conjunto de atributos (es decir, las entidades del mismo tipo pueden tener un atributos diferentes establecidos) los atributos que retorna esta operación son todos los atributos establecidos en la union de ese tipo de entidad.

d) Update Context como operación de conveniencia:

Vamos a establecer los valores de temperatura y presión de Room1:

```
 (curl localhost:1026/v1/contextEntities/Room1/attributes -s -S --header 'Content-
Type: application/json' --header 'Accept: application/json' -X PUT -d @- | python -
mjson.tool) << EOF
  {
  "attributes": [
      {
      "name": "temperature",
      "type": "float",
      "value": "26.5"
      },
      {
      "name": "pressure",
      "type": "integer",
      "value": "763"
      }
  ]
  }
  EOF
```

La respuesta es:

```
 {
  "contextResponses": [
      {
      "attributes": [
          {
          "name": "temperature",
          "type": "float",
          "value": ""
          },
          {
          "name": "pressure",
          "type": "integer",
          "value": ""
          }
      ],
      "statusCode": {
          "code": "200",
          "reasonPhrase": "OK"
      }
      }
  ]
  }
```

Realizamos el mismo proceso con Room2. Se puede actualizar un único atributo de una entidad determinada de la siguiente manera:

```
(curl localhost:1026/v1/contextEntities/Room2/attributes/temperature -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -X PUT -d @- |
python -mjson.tool) <<EOF
  {
    "value" : "26.3"
  }
  EOF
```

e) Operaciones de Conveniencia para las suscripciones de contexto:

Puede utilizar las siguientes operaciones de conveniencia para gestionar las suscripciones de contexto:

POST/v1/contextSubscriptions, para crear la suscripción, usando la misma carga útil que la operación estándar susbcribeContext.

PUT/v1/contextSubscriptions /{subscriptionID} para actualizar la suscripción identificada por {susbcriptionID} utilizando la misma carga útil que la operación estándar updateContextSubscription. El ID en la carga útil debe coincidir con el ID en la URL.

DELETE/v1/contextSubscriptions/{subscriptionID}, para cancelar la suscripción identificado por {subscriptionID}. En este caso, no se utiliza la carga útil.

f) Resumen de las operaciones de conveniencia NGSI10 para URLs

Las operaciones de Conveniencia utilizan una URL para identificar el recurso y un verbo HTTP para identificar la operación en ese recurso, se hace siguiendo la convención habitual de REST: GET se utiliza para recuperar información, POST se utiliza para crear nueva información, PUT se utiliza para actualizar la información y DELETE para destruir información.

8.4. Gestión de la disponibilidad de contexto NGSI9

8.4.1.Operaciones Estándar de NGSI9:

En esta sección se describen las distintas operaciones estándar de NGSI9 que Orion Context Broker apoya, mostrando ejemplos de solicitudes y respuestas. Utilizamos el término "estándar" para aquellas operaciones que se derivan directamente de la especificación OMA NGSI, para distinguirlos de la familia de operaciones de conveniencia que han sido definidas por el proyecto FIWARE para facilitar el uso de implementaciones NGSI. No se olvide de reiniciar el Broker como se ha descrito anteriormente en este documento antes de comenzar este tutorial.

Al final de esta sección, tendrá los conocimientos básicos para crear aplicaciones (tanto a los productores y consumidores de contexto) utilizando Orion Context Broker con NGSI9 y sus operaciones estándar:

registerContext
discoverContextAvailability
subscribeContextAvailability
updateContextAvailabilitySubscription
unsubscribeContextAvailability

a) Operación de registro de contexto:

En primer lugar hay que registrar Room1y Room2. Para ello, utilizamos la operación registerContext de NGSI9:

```
(curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
```

```
    "entities": [
        {
        "type": "Room",
        "isPattern": "false",
         "id": "Room1"
        },
        {
        "type": "Room",
        "isPattern": "false",
        "id": "Room2"
        }
    ],
    "attributes": [
        {
        "name": "temperature",
        "type": "float",
        "isDomain": "false"
        },
        {
        "name": "pressure",
        "type": "integer",
        "isDomain": "false"
        }
    ],
    "providingApplication": "http://mysensors.com/Rooms"
    }
  ],
"duration": "P1M"
}
EOF
```

La carga útil incluye una lista de elementos contextRegistration, cada una con la siguiente información:

Una lista de las entidades que se van a registrar. En nuestro caso, son las entidades Room1 y Room2. Para cada entidad se especifica un tipo (en este caso, estamos usando "Room" como tipo) y un ID (que son "Room1" y "Room2", respectivamente). El campo isPattern no se utiliza en registerContext, por lo que siempre tiene un valor de "false".

Una lista de los atributos que se inscriben en las entidades. En nuestro caso, son los atributos de temperatura y presión. Para cada uno de ellos, definimos un nombre, un tipo y si se trata de un atributo de dominio o no. Orion Context Broker como ya se comento anteriormente no realiza ninguna comprobación de tipos. Además, los atributos de dominio no son compatibles, por lo que isDomain siempre se debe establecer en "false".

Se debe de tener en cuenta que en este caso se están registrando ambas habitaciones usando sólo un elemento contextRegistration, pero también se podría haber realizado con dos contextRegistrations, cada uno para una habitación diferente.

Por último, se debe de tener en cuenta que la carga útil incluye un elemento de duración. El elemento de duración fija la duración de la inscripción así que después de que haya pasado el tiempo, puede ser considerado como finalizada (sin embargo, la duración puede extenderse). Al igual que en NGSI10 se utiliza la norma ISO 8601 para el formato de duración. Estamos utilizando "P1M", que significa un mes.
Se obtiene la siguiente respuesta:

```
{
 "duration" : "P1M",
   "registrationId" : "52a744b011f5816465943d58"
 }
```

El registrationId (cuyo varía dependiendo de cuando se ejecuta la petición, ya que se genera utilizando la marca de hora de la hora actual) es un dígito hexadecimal que proporciona una referencia única para el registro. Se utiliza para actualizar el registro.

b) Operación de descubrimiento de disponibilidad de contexto:

Ahora el Broker tiene la información de registro sobre Room1 y Room2. ¿Cómo podemos acceder a esa información? Utilizando la operación NGSI9 discoverContextAvailability. Por ejemplo, podemos descubrir las inscripciones para Room1 usando:

```
(curl localhost:1026/v1/registry/discoverContextAvailability -s -S --header 'Content-
Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
    ]
  }
  EOF
```

Esto genera la siguiente respuesta:

```
{
  "contextRegistrationResponses": [
      {
    "contextRegistration": {
          "attributes": [
          {
              "isDomain": "false",
              "name": "temperature",
              "type": "float"
          },
          {
              "isDomain": "false",
              "name": "pressure",
              "type": "integer"
          }
          ],
          "entities": [
          {
              "id": "Room1",
              "isPattern": "false",
              "type": "Room"
          }
          ],
          "providingApplication": "http://mysensors.com/Rooms"
      }
      }
  ]
  }
```

Se debe de tener en cuenta que utilizamos unos atributos vacíos en la solicitud. Si queremos ser más precisos, se puede incluir el nombre de un atributo a buscar:

```
 (curl localhost:1026/v1/registry/discoverContextAvailability -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
  "entities": [
```

```
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      }
  ],
  "attributes": [
      "temperature"
  ]
  }
  EOF
```

Lo cual produce la siguiente respuesta:

```
 {
  "contextRegistrationResponses": [
      {
      "contextRegistration": {
          "attributes": [
          {
              "isDomain": "false",
              "name": "temperature",
              "type": "float"
          }
          ],
          "entities": [
          {
              "id": "Room1",
              "isPattern": "false",
              "type": "Room"
          }
          ],
          "providingApplication": "http://mysensors.com/Rooms"
        }
      }
  ]
  }
```

Si el Broker no tiene información de registro, devolverá una respuesta similar a esta. Por lo tanto, la siguiente petición:

```
 (curl localhost:1026/v1/registry/discoverContextAvailability -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
  "entities": [
      {
    "type": "Room",
    "isPattern": "false",
    "id": "Room1"
      }
  ],
  "attributes": [
    "humidity"
  ]
  }
  EOF
```

Produciría la siguiente respuesta:

```
 {
  "errorCode": {
      "code": "404",
      "reasonPhrase": "No context element registrations found"
```

```
  }
  }
```

También puede buscar una lista de las entidades, por ejemplo, para descubrir la temperatura tanto en Room1 y Room2 :

```
(curl localhost:1026/v1/registry/discoverContextAvailability -s -S --header 'Content-
Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room1"
      },
      {
      "type": "Room",
      "isPattern": "false",
      "id": "Room2"
      }
  ],
  "attributes": [
      "temperature"
]
  }
  EOF
```

La cual produce la siguiente respuesta:

```
 {
  "contextRegistrationResponses": [
      {
      "contextRegistration": {
          "attributes": [
          {
              "isDomain": "false",
              "name": "temperature",
              "type": "float"
          }
          ],
          "entities": [
          {
              "id": "Room1",
              "isPattern": "false",
              "type": "Room"
          },
          {
              "id": "Room2",
              "isPattern": "false",
              "type": "Room"
          }
          ],
          "providingApplication": "http://mysensors.com/Rooms"
      }
      }
  ]
  }
```

Por último, una potente función de Orión Context Broker es que se puede utilizar una expresión incompleta para el ID de entidad. Por ejemplo, se pueden descubrir entidades cuyos ID comienza con "Room" usando la expresión "Room.*". En este caso, usted tiene que fijar isPattern a "true", como se muestra a continuación:

```
 (curl localhost:1026/v1/registry/discoverContextAvailability -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
    "entities": [
    {
        "type": "Room",
        "isPattern": "true",
        "id": "Room.*"
    }
    ],
    "attributes": [
    "temperature"
    ]
}
  EOF
```

Esto producirá la misma respuesta que el ejemplo anterior. Como se comentó anteriormente por defecto sólo se devuelven 20 registros.

c) Subscripciones de disponibilidad de contexto:

Las operaciones NGSI9 que conoces hasta ahora (registerContext y discoverContextAvailability) son los bloques básicos de construcción para aplicaciones síncronas para consumidores y productores de contexto. Sin embargo, Orion Context Broker tiene otra característica de gran alcance de la que se puede tomar ventaja que es la capacidad de disponibilidad de contexto de la información de modo que cuando "algo" sucede su solicitud recibirá una notificación asíncrona. De esa manera, no es necesario repetir continuamente peticiones discoverContextAvailability, Orion Context Broker le hará saber la información.
Suponemos que el programa accumulator-server.py sigue en ejecución. De lo contrario, se debe iniciar como se describió en la sección anterior.

Las suscripciones de disponibilidad de contexto se utilizan cuando se quiere ser notificado no sobre la información de contexto, sino acerca de la disponibilidad del contexto en la fuente. Se va a aclarar lo que esto significa con un ejemplo.

Se considera que una solicitud de una aplicación de consumo de contexto quiere ser notificada cada vez que el Orion Context Broker pone al tanto un nuevo registro de habitaciones, por ejemplo, porque un nuevo icono de la habitación se ha de elaborar en la interfaz gráfica de usuario que la aplicación ofrece a los usuarios finales. Así, cada vez que una nueva entidad del tipo "Sala" se ha registrado en el Broker (usando la operación registerContext), el Broker debe de ser capaz de enviar notificaciones.

Para configurar este comportamiento, se utiliza la siguiente solicitud NGSI9 subscribeContextAvailability:

```
 (curl localhost:1026/v1/registry/subscribeContextAvailability -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
 {
"entities": [
    {
         "type": "Room",
         "isPattern": "true",
         "id": ".*"
    }
    ],
    "attributes": [
     "temperature"
    ],
    "reference": "http://localhost:1028/accumulate",
```

```
    "duration": "P1M"
}
EOF
```

Como se puede ver, la estructura de subscriptionContextAvailability es similar a la estructura de NGSI10 subscribeContext, aunque en este caso no utilizamos notifyConditions ni "throttling".

La respuesta a la solicitud subscribeContextAvailability es un identificador de suscripción y un reconocimiento de duración. Una vez más, bastante similar a un subscribeContext.

```
{
 "duration": "P1M",
   "subscriptionId": "52a745e011f5816465943d59"
 }
```

En el accumulator-server.py, veremos la siguiente notificación:

```
POST http://localhost:1028/accumulate
 Content-Length: 638
 User-Agent: orion/0.9.0
 Host: localhost:1028
 Accept: application/xml, application/json
 Content-Type: application/json

 {
 "subscriptionId": "52a745e011f5816465943d59",
 "contextRegistrationResponses": [
     {
     "contextRegistration": {
         "entities": [
         {
             "type": "Room",
             "isPattern": "false",
             "id": "Room1"
         },
         {
             "type": "Room",
             "isPattern": "false",
             "id": "Room2"
         }
         ],
         "attributes": [
         {
             "name": "temperature",
             "type": "float",
             "isDomain": "false"
         }
         ],
         "providingApplication": "http://mysensors.com/Rooms"
     }
     }
 ]
}
```

Orion Context Broker notifica la operación subscribeContextAvailability en NGSI9 utilizando el método HTTP POST con una carga útil notifyContextAvailabilityRequest. Aparte del elemento subscriptionId (que coincide con el de la respuesta a la solicitud subscribeContextAvailability) y el elemento originador, el elemento contextResponseList es el mismo que el utilizado en las respuestas discoverContextAvailability.

La notificación inicial incluye todas las entidades actualmente registradas que coinciden con el entityIdList / AttributeList utilizado en la solicitud subscribeContextAvailability. Es decir, el registro correspondiente a Room1 Room2 y la temperatura. Se debe de tener en cuenta que, aunque Room1 y Room2 estén registradas y sus dos atributos (temperatura y presión) sólo se muestra la temperatura, ya que el AttributeList en subscribeContextAvailability sólo incluye la temperatura.

Vamos a ver lo que sucede cuando se registra una nueva entidad (Room3) con la temperatura y la presión:

```
(curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
      "entities": [
          {
          "type": "Room",
          "isPattern": "false",
          "id": "Room3"
          }
      ],
        "attributes": [
        {
          "name": "temperature",
          "type": "float",
          "isDomain": "false"
        },
        {
          "name": "pressure",
          "type": "integer",
          "isDomain": "false"
        }
        ],
      "providingApplication": "http://mysensors.com/Rooms"
      }
    ],
    "duration": "P1M"
  }
  EOF
```

Como era de esperar, el accumulator-server.py notifica de la nueva inscripción. Una vez más, aunque el registro incluye la temperatura y la presión de Room3, sólo el primer atributo se incluye en la notificación.

```
POST http://localhost:1028/accumulate
  Content-Length: 522
  User-Agent: orion/0.9.0
  Host: localhost:1028
  Accept: application/xml, application/json
  Content-Type: application/json

  {
  "subscriptionId": "52a745e011f5816465943d59",
  "contextRegistrationResponses": [
      {
      "contextRegistration": {
          "entities": [
          {
              "type": "Room",
              "isPattern": "false",
              "id": "Room3"
```

```
          }
          ],
        "attributes": [
          {
              "name": "temperature",
              "type": "float",
              "isDomain": "false"
          }
          ],
          "providingApplication": "http://mysensors.com/Rooms"
      }
      }
  ]
  }
```

También se puede comprobar que los registros de contexto que no coincidan con la suscripción no desencadenan ninguna notificación. Por ejemplo, vamos a registrar una entidad (Room4) con sólo la presión como atributo (recordemos que la suscripción sólo incluye temperatura en AttributeList):

```
 (curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
      "entities": [
          {
          "type": "Room",
          "isPattern": "false",
          "id": "Room4"
          }
      ],
      "attributes": [
          {
          "name": "pressure",
          "type": "integer",
          "isDomain": "false"
          }
      ],
      "providingApplication": "http://mysensors.com/Rooms"
      }
  ],
  "duration": "P1M"
  }
  EOF
```

Se puede comprobar que ninguna nueva notificación llega a accumulator-server.py. Al igual que con las suscripciones de contexto, las suscripciones disponibilidad de contexto se pueden actualizar (mediante el NGSI9 updateContextAvailabilitySubscription). La solicitud incluye el subscriptionId que identifica la suscripción a modificar, y la carga útil de actualización. Por ejemplo, vamos a cambiar las entidades de suscripción a algo diferente: los coches en lugar de las habitaciones y todos los atributos se quitan (es decir, un elemento AttributeList vacío).

```
 (curl localhost:1026/v1/registry/updateContextAvailabilitySubscription -s -S --
header 'Content-Type: application/json' --header 'Accept: application/json' -d @- |
python -mjson.tool) <<EOF
  {
  "entities": [
      {
      "type": "Car",
      "isPattern": "true",
      "id": ".*"
```

```
    }
  ],
  "duration": "P1M",
  "subscriptionId": "52a745e011f5816465943d59"
  }
  EOF
```

La respuesta es similar a la de la solicitud subscribeContextAvailability:

```
  {
  "duration": "P1M",
    "subscriptionId": "52a745e011f5816465943d59"
  }
```

Dado que actualmente no existen entidades de coches registradas, no se recibirá ninguna notificación inicial. Si se registran dos coches: Car1 con un atributo velocidad y Car2 con un atributo ubicación:

```
(curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
      "entities": [
          {
          "type": "Car",
          "isPattern": "false",
          "id": "Car1"
          }
        ],
      "attributes": [
          {
          "name": "speed",
          "type": "integer",
          "isDomain": "false"
          }
        ],
          "providingApplication": "http://mysensors.com/Cars"
      }
  ],
  "duration": "P1M"
  }
  EOF

 (curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
        "entities": [
          {
          "type": "Car",
          "isPattern": "false",
          "id": "Car2"
          }
      ],
      "attributes": [
          {
          "name": "location",
          "type": "ISO6709",
          "isDomain": "false"
          }
      ],
```

```
    "providingApplication": "http://mysensors.com/Cars"
    }
],
  "duration": "P1M"
}
EOF
```

Como ambos registros coinciden en entityIdList y AttributeList utilizados en la updateContextAvailabilitySubscription, obtendremos una notificación para cada registro del coche, como se puede ver en accumulator-server.py:

```
POST http://localhost:1028/accumulate
Content-Length: 529
User-Agent: orion/0.9.0
Host: localhost:1028
Accept: application/xml, application/json
Content-Type: application/json
{
"subscriptionId": "52a745e011f5816465943d59",
"contextRegistrationResponses": [
    {
    "contextRegistration": {
        "entities": [
        {
            "type": "Car",
            "isPattern": "false",
            "id": "Car1"
        }
        ],
        "attributes": [
        {
            "name": "speed",
            "type": "integer",
            "isDomain": "false"
        }
        ],
        "providingApplication": "http://mysensors.com/Cars"
    }
    }
]
}
POST http://localhost:1028/accumulate
Content-Length: 535
User-Agent: orion/0.9.0
Host: localhost:1028
Accept: application/xml, application/json
Content-Type: application/json

{
"subscriptionId": "52a745e011f5816465943d59",
"contextRegistrationResponses": [
    {
    "contextRegistration": {
        "entities": [
        {
            "type": "Car",
            "isPattern": "false",
            "id": "Car2"
        }
        ],
        "attributes": [
        {
            "name": "location",
            "type": "ISO6709",
            "isDomain": "false"
```

```
          }
          ],
          "providingApplication": "http://mysensors.com/Cars"
      }
      }
  ]
  }
```

Por último, se puede cancelar una suscripción usando la operación NGSI9 unsubscribeContextAvailability, simplemente usando el subscriptionId de la solicitud de carga útil.

```
(curl localhost:1026/v1/registry/unsubscribeContextAvailability -s -S --header
'Content-Type: application/json' --header 'Accept: application/json' -d @- | python -
mjson.tool) <<EOF
  {
        "subscriptionId": "52a745e011f5816465943d59"
  }
  EOF
```

La respuesta es un reconocimiento de que la cancelación se ha realizado correctamente.

```
 {
"statusCode": {
  "code": "200",
  "reasonPhrase": "OK"
},
    "subscriptionId": "52a745e011f5816465943d59"
  }
```

d)Resumen de las operaciones estándar NGSI9 sobre URLs

/v1/registry/registerContext
/v1/registry/discoverContextAvailability
/v1/registry/subscribeContextAvailability
/v1/registry/updateContextAvailabilitySubscription
/v1/registry/unsubscribeContextAvailability

8.4.1.Operaciones de conveniencia de NGSI9:

En la siguiente sección se describen las distintas operaciones de conveniencia descritas como parte de NGSI REST API NGSI9 en FIWARE que Orion Context Broker soporta, mostrando ejemplos de solicitudes y respuestas.Las operaciones de conveniencia son un conjunto de operaciones que han sido definidas en el proyecto FIWARE para facilitar el uso de las implementaciones NGSI como complemento a las operaciones estándar definidas en la especificación OMA NGSI.

No se olvide de reiniciar el corredor antes de comenzar este tutorial como se ha descrito anteriormente en este documento. Al final de esta sección, se habrá aprendido a emplear las operaciones de conveniencia como una alternativa útil para algunas operaciones estándar descritas en el apartado anterior.

a) Operación de conveniencia Register Context:

En primer lugar, se registra Room1 y Room2 con sus atributos de temperatura y presión, utilizando los siguientes comandos:

```
(curl localhost:1026/v1/registry/contextEntities/Room1/attributes/temperature -s -S
--header 'Content-Type: application/xml' -d @- | xmllint --format - ) << EOF
(curl localhost:1026/v1/registry/contextEntities/Room1/attributes/temperature -s -S
--header 'Content-Type: application/json' --header 'Accept: application/json'  -d @-
| python -mjson.tool) << EOF
  <?xml version="1.0"?>
{
  <registerProviderRequest>
"duration" : "P1M",
    <duration>P1M</duration>
"providingApplication" : "http://mysensors.com/Rooms"
    <providingApplication>http://mysensors.com/Rooms</providingApplication>
}
  </registerProviderRequest>
EOF
  EOF

  (curl localhost:1026/v1/registry/contextEntities/Room1/attributes/pressure -s -S --
header 'Content-Type: application/xml' -d @- | xmllint --format - ) << EOF
(curl localhost:1026/v1/registry/contextEntities/Room1/attributes/pressure -s -S --
header 'Content-Type: application/json' --header 'Accept: application/json'  -d @- |
python -mjson.tool) << EOF
  <?xml version="1.0"?>
{
  <registerProviderRequest>
"duration" : "P1M",
    <duration>P1M</duration>
"providingApplication" : "http://mysensors.com/Rooms"
    <providingApplication>http://mysensors.com/Rooms</providingApplication>
}
  </registerProviderRequest>
EOF
  EOF

  (curl localhost:1026/v1/registry/contextEntities/Room2/attributes/temperature -s -S
--header 'Content-Type: application/xml' -d @- | xmllint --format - ) << EOF
(curl localhost:1026/v1/registry/contextEntities/Room2/attributes/temperature -s -S
--header 'Content-Type: application/json' --header 'Accept: application/json'  -d @-
| python -mjson.tool) << EOF
  <?xml version="1.0"?>
{
  <registerProviderRequest>
"duration" : "P1M",
    <duration>P1M</duration>
"providingApplication" : "http://mysensors.com/Rooms"
    <providingApplication>http://mysensors.com/Rooms</providingApplication>
}
  </registerProviderRequest>
EOF
  EOF

  (curl localhost:1026/v1/registry/contextEntities/Room2/attributes/pressure -s -S --
header 'Content-Type: application/xml' -d @- | xmllint --format - ) << EOF
(curl localhost:1026/v1/registry/contextEntities/Room2/attributes/pressure -s -S --
header 'Content-Type: application/json' --header 'Accept: application/json'  -d @- |
python -mjson.tool) << EOF
  <?xml version="1.0"?>
{
  <registerProviderRequest>
"duration" : "P1M",
    <duration>P1M</duration>
"providingApplication" : "http://mysensors.com/Rooms"
    <providingApplication>http://mysensors.com/Rooms</providingApplication>
}
  </registerProviderRequest>
EOF
```

```
EOF
```

¿Cuales son las diferencias con respecto a la operación registerContext estándar?

Necesitábamos cuatro solicitudes, en lugar de una solicitud en el funcionamiento estándar. Estamos utilizando más operaciones, pero la carga útil que se utiliza en cada operación es mucho más simple. Esta carga útil es una versión simplificada de la carga útil en registerContext, incluyendo sólo la duración y la prestación de aplicación. Desde la perspectiva del Orion Context Broker, hay 4 registros independientes (es decir, 4 diferentes IDs de registro) a todos los efectos.

La respuesta a cada una de estas solicitudes es la misma que la respuesta a una registerContext estándar (una respuesta para cada una de las cuatro solicitudes, con un ID diferente):

```
{
 "duration": "P1M",
   "registrationId": "51c1f5c31612797e4fe6b6b6"
 }
```

Entidades de un único tipo de registro que utilizan operaciones de conveniencia

Se pueden utilizar los "contextEntityTypes" de las operaciones de conveniencia en NGSI9 para registrar los tipos de cada entidad sin una identificación específica. Vamos a ilustrarlo con un ejemplo.

Se registra la entidad de tipo "Funny" (tenga en cuenta que no estamos especificando la ID de la entidad):

```
(curl localhost:1026/v1/registry/contextEntityTypes/Funny -s -S --header 'Content-
Type: application/xml' -d @- | xmllint --format - ) << EOF
<?xml version="1.0"?>
<registerProviderRequest>
   <duration>P1M</duration>
   <providingApplication>http://mysensors.com/Funny</providingApplication>
</registerProviderRequest>
EOF
```

Ahora, vamos a realizar el descubrimiento de ese tipo:

```
curl localhost:1026/v1/registry/contextEntityTypes/Funny -s -S --header 'Content-
Type: application/xml' | xmllint --format -

<discoverContextAvailabilityResponse>
  <contextRegistrationResponseList>
    <contextRegistrationResponse>
      <contextRegistration>
        <entityIdList>
          <entityId type="Funny" isPattern="false">
            <id/>
          </entityId>
        </entityIdList>
        <providingApplication>http://mysensors.com/Funny</providingApplication>
      </contextRegistration>
    </contextRegistrationResponse>
  </contextRegistrationResponseList>
</discoverContextAvailabilityResponse>
```

Como se puede ver, el elemento de identificación está vacío (lo cual tiene sentido, ya que no se especifica ninguna identificación en el registro).

Por otra parte, se puede registrar atributos en estos registros, por ejemplo:

```
(curl localhost:1026/v1/registry/contextEntityTypes/MoreFunny/attributes/ATT -s -S --
header 'Content-Type: application/xml' -d @- | xmllint --format - ) << EOF
<?xml version="1.0"?>
<registerProviderRequest>
   <duration>P1M</duration>
   <providingApplication>http://mysensors.com/Funny</providingApplication>
</registerProviderRequest>
EOF

curl localhost:1026/v1/registry/contextEntityTypes/MoreFunny -s -S --header 'Content-
Type: application/xml' | xmllint --format -
<?xml version="1.0"?>
<discoverContextAvailabilityResponse>
  <contextRegistrationResponseList>
    <contextRegistrationResponse>
      <contextRegistration>
        <entityIdList>
          <entityId type="MoreFunny" isPattern="false">
            <id/>
          </entityId>
        </entityIdList>
        <contextRegistrationAttributeList>
          <contextRegistrationAttribute>
            <name>ATT</name>
            <type/>
            <isDomain>false</isDomain>
          </contextRegistrationAttribute>
        </contextRegistrationAttributeList>
        <providingApplication>http://mysensors.com/Funny</providingApplication>
      </contextRegistration>
    </contextRegistrationResponse>
  </contextRegistrationResponseList>
</discoverContextAvailabilityResponse>
```

b) Disponibilidad de descubrimiento de contexto con operaciones de conveniencia:

Con el uso de las operaciones de conveniencia se puede descubrir la información de registro para una sola entidad o para un par de entidad-atributo. Por ejemplo, para descubrir las inscripciones para Room1 (independientemente de los atributos):

```
 curl localhost:1026/v1/registry/contextEntities/Room1 -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' | python -mjson.tool
```

Para descubrir las inscripciones a Room2 con atributo de temperatura:

```
 curl localhost:1026/v1/registry/contextEntities/Room2/attributes/temperature -s -S
--header 'Content-Type: application/json' --header 'Accept: application/json' |
python -mjson.tool
```

El descubrimiento de los elementos no registrados (por ejemplo Room5 o la humedad de Room1) producirá un error. Por ejemplo en las siguientes peticiones:

```
  curl localhost:1026/v1/registry/contextEntities/Room3 -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' | python -mjson.tool

  curl localhost:1026/v1/registry/contextEntities/Room2/attributes/humidity -s -S --
header 'Content-Type: application/json' --header 'Accept: application/json' | python
-mjson.tool
```

Ambas producen la misma respuesta de error:

```
  {
"errorCode": {
  "code": "404",
   "reasonPhrase": "No context element found"
     }
  }
```

En comparación con la operación discoverContextAvailability estándar las operaciones de conveniencia utilizan el método GET sin necesidad de carga útil en la solicitud (más simple que la operación estándar). Sin embargo, hay dos diferencias en el contenido. En primer lugar, cada atributo siempre viene en un elemento contextRegistrationResponse diferente (como cada atributo corresponde a un registro diferente, como se ha explicó antes).

También puede descubrir por todas las entidades pertenecientes a un mismo tipo, o bien todos los atributos o a una en particular, como se muestra a continuación. En primer lugar, para registrar una par de entidades de tipo de coche que utilizan operaciones registerContext estándar (dado que, como se describe en el apartado anterior, no se pueden registrar entidades con los tipos que utilizan con operaciones de conveniencia):

```
(curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
      "entities": [
          {
          "type": "Car",
          "isPattern": "false",
          "id": "Car1"
          }
      ],
      "attributes": [
          {
          "name": "speed",
          "type": "integer",
          "isDomain": "false"
          }
      ],
      "providingApplication": "http://mysensors.com/Cars"
      }
  ],
  "duration": "P1M"
  }
  EOF

  (curl localhost:1026/v1/registry/registerContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- | python -mjson.tool)
<<EOF
  {
  "contextRegistrations": [
      {
      "entities": [
          {
          "type": "Car",
          "isPattern": "false",
          "id": "Car2"
          }
      ],
      "attributes": [
          {
          "name": "fuel",
          "type": "float",
          "isDomain": "false"
```

```
        }
    ],
    "providingApplication": "http://mysensors.com/Cars"
    }
],
"duration": "P1M"
}
EOF
```

Petición sin especificar atributos:

```
curl localhost:1026/v1/registry/contextEntityTypes/Car -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' | python -mjson.tool
```

Respuesta:

```
{
 "contextRegistrationResponses": [
     {
     "contextRegistration": {
         "attributes": [
         {
             "isDomain": "false",
             "name": "speed",
             "type": "integer"
         }
         ],
         "entities": [
         {
             "id": "Car1",
             "isPattern": "false",
             "type": "Car"
         }
         ],
         "providingApplication": "http://mysensors.com/Cars"
     }
     },
     {
     "contextRegistration": {
         "attributes": [
         {
             "isDomain": "false",
             "name": "fuel",
             "type": "float"
         }
         ],
         "entities": [
         {
             "id": "Car2",
             "isPattern": "false",
             "type": "Car"
         }
         ],
         "providingApplication": "http://mysensors.com/Cars"
     }
     }
]
}
```

Solicitud especificando un atributo (p.e. velocidad):

```
curl localhost:1026/v1/registry/contextEntityTypes/Car/attributes/speed -s -S --
header 'Content-Type: application/json' --header 'Accept: application/json' | python
-mjson.tool
```

Respuesta:


```
{
 "contextRegistrationResponses": [
     {
     "contextRegistration": {
         "attributes": [
         {
             "isDomain": "false",
             "name": "speed",
             "type": "integer"
         }
         ],
         "entities": [
         {
             "id": "Car1",
             "isPattern": "false",
             "type": "Car"
         }
         ],
         "providingApplication": "http://mysensors.com/Cars"
     }
     }
 ]
 }
```


Tenga en cuenta que por defecto sólo se devuelven 20 registros.

c) Operaciones de Conveniencia para suscripciones de disponibilidad contexto:

Puede utilizar las siguientes operaciones de conveniencia para gestionar suscripciones disponibilidad contexto:

POST/v1/registry/contextAvailabilitySubscriptions, para crear la suscripción, con la misma carga útil de la operación susbcribeAvailabilityContext estándar.

PUT/v1/registry/contextAvailabilitySubscriptions/{subscriptionID}, para actualizar la suscripción identificada por {subscriptionId}, con la misma carga útil que la operación updateContextAvailabilitySubscription estándar. El ID en la carga útil debe coincidir con el ID en el URL.

DELETE/v1/registry/contextAvailabilitySubscriptions/{subscriptionID}, para cancelar la suscripción identificada por {subscriptionID} abono o suscripción. En este caso, no se utiliza la carga útil.

d) Resumen de las operaciones de conveniencia en NGSI9 sobre URLs

Las operaciones de Conveniencia utilizan una URL para identificar el recurso y un verbo HTTP para identificar la operación en ese recurso siguiendo la convención habitual REST: GET se utiliza para recuperar información, POST se utiliza para crear nueva información, PUT se utiliza para actualizar la información y DELETE se usa para destruir información. Encontrará un resumen en el siguiente documento[20].

[20] https://docs.google.com/spreadsheets/d/1f4m624nmO3jRjNalGE11lFLQixnfCENMV6dc54wUCCg/edit#gid=0