SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

TESIS DE MAESTRÍA EN CIENCIAS

Desarrollo de componentes genéricos para la conexión de dispositivos móviles a una plataforma IoT utilizando la especificación NGSI de FIWARE

Presentada por
Ing. Fernando Ramírez Cipriano

Como requisito para la obtención del grado de
Maestro en Ciencias de la Computación

Director de tesis
Dra. Alicia Martínez Rebollar

Codirector de tesis
Dr. Hugo Estrada Esquivel

Cuernavaca, Morelos, México. Enero de 2019.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Morelos, 10/Diciembre/2018
OFICIO No. DCC/251/2018

Asunto: **Aceptación de documento de tesis**

DR. GERARDO V. GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del **Ing. Fernando Ramírez Cipriano**, con número de control M16CE086, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado **"Desarrollo de componentes genéricos para la conexión de dispositivos móviles a una plataforma IoT utilizando la especificación NGSI de FIWARE"** y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dra. Alicia Martínez Rebollar
Doctora en Informática
7399055

CO-DIRECTOR DE TESIS

Dr. Hugo Estrada Esquivel
Doctor en Informática
7399054

REVISOR 1

Dr. Miguel González Mendoza
Doctor en Sistemas Automáticos

REVISOR 2

Dr. Noé Alejandro Castro Sánchez
Doctor en Ciencias de la Computación
08701806

C.p. M.T.I. María Elena Gómez Torres - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente

NACS/lmz

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MEXICO

Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 14 de diciembre de 2018
OFICIO No. SAC/581/2018

Asunto: Autorización de impresión de tesis

ING. FERNANDO RAMÍREZ CIPRIANO
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS
DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Desarrollo de Componentes Genéricos para la Conexión de Dispositivos Móviles a una Plataforma IoT utilizando la especificación NGSI de FIWARE"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

SEP TecNM
CENTRO NACIONAL
DE INVESTIGACIÓN
Y DESARROLLO
TECNOLÓGICO
SUBDIRECCIÓN
ACADÉMICA

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

C.p. M.T.I. María Elena Gómez Torres .- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Dedicatoria

A mis padres Fernando Ramírez García y Santa Cipriano Mora por confiar en mí y brindarme su apoyo incondicional.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico que me brindó para realizar mis estudios de maestría.

Al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET) por brindarme la oportunidad de superarme académicamente en el programa de Maestría en Ciencias de la Computación.

Un especial agradecimiento a mi directora de tesis, la Dra. Alicia Martínez Rebollar y a mi codirector, el Dr. Hugo Estrada Esquivel, por sus gratas enseñanzas, además, por su amistad, sentido humano y por haberme enseñado el valor de la humildad.

A mis revisores, el Dr. Miguel González Mendoza y el Dr. Noé Castro Sánchez que fortalecieron mi trabajo con sus aportaciones.

A mis amigos de la maestría que me apoyaron y fortalecieron mis conocimientos.

Resumen

Actualmente, el internet de las cosas (IoT, por sus siglas en inglés) es un nuevo paradigma que está ganando terreno rápidamente en el escenario de las modernas telecomunicaciones inalámbricas. La idea básica de este concepto es la presencia activa alrededor de nosotros de una variedad de cosas u objetos tales como etiquetas de identificación por radiofrecuencia (RFID, por sus siglas en inglés), sensores, actuadores, teléfonos móviles, etc. Para interactuar entre sí y cooperar con sus vecinos para alcanzar objetivos comunes [1].

La adopción de soluciones IoT posibilitará nuevas oportunidades de negocio, mejora la eficiencia en la gestión de recursos, reducción de costes operacionales para las empresas, mejora la calidad de vida y transformación de las ciudades.

La información que llega a Internet a través de IoT puede ser combinada en sistemas bidireccionales que integran datos, personas, procesos y sistemas para tomar las mejores decisiones. Es decir, la interacción entre Sensores + Conectividad + Personas + Procesos están generando nuevos tipos de aplicaciones inteligentes y nuevos servicios [3].

Debido a los desafíos asociados a la creciente urbanización en las grandes ciudades, la Unión Europea está financiando una iniciativa denominada *FIWARE* [5] cuyo objetivo es facilitar el desarrollo de aplicaciones inteligentes mediante una infraestructura abierta basada en la nube que ofrece un catálogo de componentes genéricos (GEs por sus siglas en inglés, *Generic Enablers*). Cada componente genérico ofrece una serie de funciones de propósito general a través de interfaces de programación de aplicaciones (APIs por sus siglas en inglés, *Aplication Programming Interface*) de licencias públicas y libres.

En este contexto, en el trabajo de investigación se desarrollaron componentes genéricos que permiten la conexión de aplicaciones de dispositivos móviles a una plataforma IoT que implementen la especificación FIWARE-NGSI v2 y que tenga como propósito obtener información de los sensores del dispositivo móvil y envié la información en tiempo real al *Orion Context Broker* de *FIWARE* bajo la especificación FIWARE-NGSI v2 [13].

Por tal motivo, en este trabajo de investigación se planteó como objetivo principal desarrollar componentes genéricos para la conexión de dispositivos móviles a una plataforma IoT utilizando la especificación NGSI de *FIWARE*.

Abstract

Currently, the Internet of Things (IoT) is a new paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. The basic idea of this concept is the active presence around us of a variety of things or objects such as radio frequency identification (RFID) tags, sensors, actuators, mobile phones, etc. To interact with each other and cooperate with their neighbors to achieve common goals [1].

The adoption of IoT solutions will enable new business opportunities, improve efficiency in resource management, reduce operational costs for companies, improve the quality of life and transformation of the cities.

The information that reaches the Internet through the IoT can be combined into bidirectional systems that integrate data, people, processes and systems to make the best decisions. That is, the interaction between Sensors + Connectivity + People + Processes is generating new types of intelligent applications and new services [3].

Due to the challenges associated with the growing urbanization in large cities, the European Union is financing an initiative called FIWARE [5] whose objective is to facilitate the development of smart applications through an open infrastructure based on the cloud that offers a catalog of generic enablers (GEs). Each generic enabler offers a series of general-purpose functions through application programming interfaces (APIs), of public and free licenses.

In this context, in this research work, generic enablers were developed. They allow us the connection of mobile device applications to an IoT platform that implements the FIWARE-NGSI v2 specification and that has the purpose of obtaining information from the mobile device's sensors and sending the real-time information to the FIWARE Orion Context Broker under the FIWARE-NGSI v2 specification [13].

Therefore, in this research work the main objective was to develop generic components for the connection of mobile devices to an IoT platform using the NGSI specification of FIWARE.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se presenta una visión general del trabajo de investigación y la forma en que está estructurado el documento de esta tesis.

1.1. Motivación

Actualmente, el internet de las cosas (*Internet of Things* (IoT), por sus siglas en inglés) es un nuevo paradigma que está ganando terreno rápidamente en el escenario de las modernas telecomunicaciones inalámbricas. La idea básica de este concepto es la presencia activa alrededor de nosotros de una variedad de cosas u objetos tales como etiquetas de identificación por radiofrecuencia (RFID, por sus siglas en inglés), sensores, actuadores, teléfonos móviles, etc. Para interactuar entre sí y cooperar con sus vecinos para alcanzar objetivos comunes [1].

Un estudio realizado por Cisco IBSG (*Internet Business Solutions Group,* por su nombre en inglés) prevé que para el 2015 habrá 25 mil millones de dispositivos conectados a internet y 50 mil millones para el 2020. Además, Cisco IBSG estima que IoT nació en algún momento entre 2008 y 2009 cuando la cantidad de dispositivos conectados superó la cantidad de personas [2]. La Figura 1 muestra la estimación de dispositivos conectados a internet desde el año 2003 hasta el 2020.

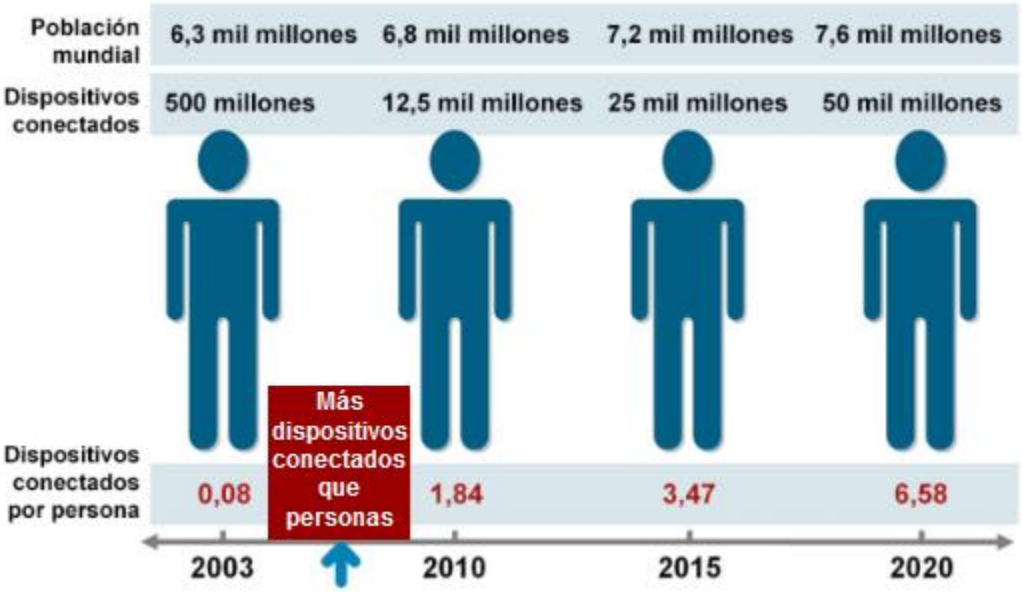

Figura 1 Estimación de dispositivos conectados a internet [2]

La adopción de soluciones IoT posibilitará nuevas oportunidades de negocio, mejora la eficiencia en la gestión de recursos, reducción de costes operacionales para las empresas, mejora la calidad de vida y transformación de las ciudades.

La información que llega a Internet a través de IoT puede ser combinada en sistemas bidireccionales que integran datos, personas, procesos y sistemas para tomar las mejores decisiones. Es decir, la interacción entre Sensores + Conectividad + Personas + Procesos están generando nuevos tipos de aplicaciones inteligentes y nuevos servicios [3].

De manera descriptiva, una ciudad inteligente es un espacio urbano con infraestructuras, redes y plataformas inteligentes, con millones de sensores y actuadores, que incluye a las personas y a sus teléfonos móviles. Éste espacio es capaz de escuchar y comprender lo que está pasando en la ciudad por lo que permite tomar mejores decisiones y proporcionar la información y los servicios adecuados a sus habitantes [4]. Además, para que los objetivos de una ciudad inteligente se cumplan ésta debe contar con una infraestructura tecnológica que la soporte.

Debido a los desafíos asociados a la creciente urbanización en las grandes ciudades, la Unión Europea está financiando una iniciativa denominada *FIWARE* [5] cuyo objetivo es facilitar el desarrollo de aplicaciones inteligentes mediante una infraestructura abierta basada en la nube que ofrece un catálogo de componentes genéricos (GEs por sus siglas en inglés, *Generic Enablers*). Cada componente genérico ofrece una serie de funciones de propósito general a través de interfaces de programación de aplicaciones (APIs por sus siglas en inglés, *Aplication Programming Interface*) de licencias públicas y libres.

1.2. Planteamiento del problema

El rápido crecimiento de la urbanización en todo el mundo está cambiando la forma de utilizar los dispositivos y las cosas presentes en el entorno tecnológico. El IoT incluye los dispositivos, servicios y soluciones que tienen conectividad con Internet. Los dispositivos están integrados con sensores, chips y otros para mejorar la experiencia del usuario. Estos dispositivos y servicios son eficientes para operar varias tareas con métodos que consumen menos tiempo. La tendencia creciente de los dispositivos habilitados para Internet y la creciente necesidad de conectividad a Internet son algunos de los principales factores que se anticipan al crecimiento positivo del mercado de IoT a nivel mundial [6].

Actualmente, las aplicaciones enfocadas al IoT implementan las tecnologías del Internet del Futuro [7] utilizan plataformas públicas o privadas tales como: *Google Cloud Platform* [8], *Amazon Web Services IoT* [9], *Watson IoT Platform* [10], *FIWARE* [5], etc. Estas plataformas cuentan con un conjunto de componentes que permiten generar aplicaciones inteligentes.

El IoT es un área nueva que plantea nuevos retos que los desarrolladores deben asumir, estos necesitan disponer de conocimientos transversales que incluyan la obtención de datos mediante sensores, comunicación de dispositivos y protocolos, así como *Big data* [11] y computo en la nube [12]. Lo más adecuado para asumir estos retos es requerir de componentes que cumplan con funciones específicas y sean reutilizables por los desarrolladores para facilitar la creación de aplicaciones inteligentes y robustas.

En este trabajo de investigación se desarrollaron componentes genéricos que permiten la conexión de aplicaciones de dispositivos móviles a una plataforma IoT que implementen la especificación FIWARE-NGSI v2 y que tenga como propósito obtener información de los sensores del dispositivo móvil y envié la información en tiempo real al *Orion Context Broker* de *FIWARE* bajo la especificación FIWARE-NGSI v2 [13].

1.3. Justificación

Actualmente, *Big data* y computo en la nube son dos términos que se utilizan en las industrias de tecnología de la información (TI, más conocida como IT por su significado en inglés, *Information Technology*).

Los recientes avances tecnológicos en dominios típicos (por ejemplo, Internet, compañías financieras, atención médica, datos generados por los usuarios, sistemas de cadena de suministro, etc.) han generado una gran cantidad de datos. Si lo comparamos con los datos tradicionales, *Big data* exhibe algunas características únicas, por lo general, es un tipo enorme y no estructurado de datos que no se puede manejar con las bases de datos tradicionales. Por lo tanto, se requieren nuevos diseños de sistema para los siguientes procesos, es decir, la recopilación de datos, la transmisión de datos, el almacenamiento y los mecanismos de procesamiento de datos a gran escala [14]. Además, el camino convencional de estos procesos está sucediendo de forma manuales y está consumiendo muchos recursos. Es decir, se necesita un enfoque avanzado para superar las barreras y las preocupaciones que enfrenta actualmente la industria del *Big data* [15].

Por otra parte, se están desarrollando una gran cantidad de aplicaciones para ciudades inteligentes en diferentes dominios de soluciones, por ejemplo: transporte y movilidad, seguridad pública, salud pública, medio ambiente y calidad de vida [16]. A continuación, se mencionan algunas aplicaciones en diferentes dominios:

- Señalización inteligente del tráfico (*Miovision TrafficLink*): monitoriza y gestiona las señales de tráfico de forma remota para priorizar los recursos y resolver posibles problemas en el momento oportuno. La arquitectura abierta permite una integración fácil y segura con el *hardware* y el *software* de tráfico que ya existe y no requiere ninguna actualización. Utilizando las cámaras *SmartView*, *TrafficLink* proporciona detección e información procesable, utilizando la inteligencia artificial más avanzada en el tráfico [17].
- Aplicaciones de transporte público (*App Moovit*): convierte, en tiempo real, los pasajeros de autobús y tren en sensores y facilita así al gobierno de la ciudad. Se integra con los sistemas urbanos de '*back end*' para mejorar,

tanto la experiencia de servicio, como de los pasajeros, y está impulsada en todo el mundo mediante *Amazon Web Services Cloud* [18].

- Iluminación inteligente de las calles (*Philips CityTouch*): es un sistema de gestión de la iluminación para el alumbrado público. Ofrece sencillas aplicaciones web para analizar, planificar y mantener la gestión de los flujos de trabajo. *CityTouch* permite a las ciudades la monitorización de los recursos energéticos que se utilizan en el alumbrado y qué elementos requieren reparación. Con *CityTouch* se decide de forma más inteligente la optimización del uso de recursos mediante la monitorización, gestión y medición de un sistema de gestión de alumbrado [19].
- Aplicación para una movilidad inteligente (*Green Route*): es una aplicación desarrollada en el ámbito de Ciudad Inteligente y centrada en apoyar la movilidad en ciudades de alta contaminación, como la Ciudad de México, con el objetivo de mejorar la calidad de vida de los ciudadanos y fomentar comportamientos respetuosos hacia el medio ambiente. Además, *Green Route* contribuye a mejorar la movilidad de una ciudad, en particular la Ciudad de México. Reúne información de unidades de monitoreo de la calidad del aire del Gobierno de la Ciudad de México y de componentes de IoT desarrollados en el proyecto de *SmartSDK*, como el *Smart Spot*. Es la primera vez que en la Ciudad de México la información sobre la calidad del aire se combina con datos de rutas de transporte público [20].

Recientemente Conacyt ha sacado convocatorias para el desarrollo de aplicaciones inteligentes, por ejemplo: Conacyt-Horizon2020 [21], Conacyt-ESRC 2018 Ciudades Inteligentes [22]. Respecto a la primera convocatoria, la Unión Europea ha impulsado el programa horizonte 2020 [23] el cual promueve la innovación. Esto ha favorecido el desarrollo de prototipos, pruebas, demostraciones, experimentos piloto, procesos de validación de productos a gran escala y operaciones de replicación comercial. Dentro de este programa la Unión Europea ha apostado por la creación de una plataforma denominada *FIWARE* la cual es una iniciativa de código abierto que pretende impulsar la creación de especificaciones y estándares necesarios para desarrollar aplicaciones inteligentes en diferentes dominios, tales como: ciudades inteligentes, puertos inteligentes, logística inteligente, fábricas inteligentes, entre otros [24].

CENIDET está participando con el proyecto *SmartSDK* [25], éste es parte del programa Horizonte 2020, que tiene como objetivo proporcionar modelos de datos y componentes genéricos reutilizables para el desarrollo de aplicaciones inteligentes.

1.4. Objetivos

El objetivo de este trabajo de investigación es desarrollar componentes genéricos para la conexión de dispositivos móviles a una plataforma IoT utilizando la especificación NGSI de *FIWARE*.

Para lograr el objetivo principal, se identificaron los siguientes objetivos específicos:

- Diseñar y desarrollar los componentes genéricos.
- Desarrollar una aplicación web y móvil que implementen los componentes genéricos.
- Probar el funcionamiento de los componentes genéricos en la aplicación web y móvil.

1.5. Metodología de solución

La metodología de solución está compuesta por tres fases: definición de los componentes genéricos, desarrollo de los componentes e implementación de los componentes en las aplicaciones. Estas fases se muestran en la Figura 2.

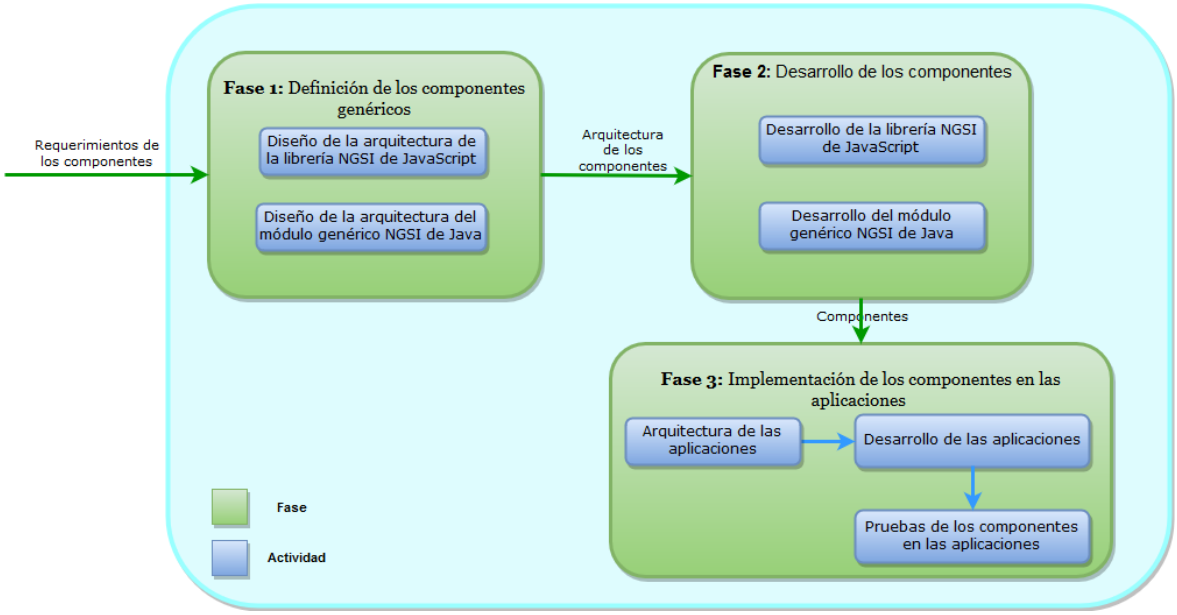

Figura 2 Metodología de solución

A continuación, se describen las tres fases de la metodología de solución:

- **Fase 1: Definición de los componentes genéricos**

  El objetivo de esta primera fase es definir el diseño de las arquitecturas de los componentes genéricos, tomando en cuenta como información de entrada los requerimientos de los componentes. A continuación, se describe cada una de las actividades.

- Diseño de la arquitectura de la librería NGSI de JavaScript. En esta actividad se diseñó la arquitectura de la librería NGSI de JavaScript, definiendo cada uno de sus módulos.
- Diseño de la arquitectura del módulo genérico NGSI de Java. En esta actividad se diseñó la arquitectura del módulo genérico NGSI de Java, definiendo cada uno de sus módulos.

- **Fase 2: Desarrollo de los componentes**

El objetivo de esta segunda fase es desarrollar cada uno de los componentes genéricos. Esta fase cuenta con dos actividades: desarrollo de la librería NGSI de JavaScript y el desarrollo del módulo genérico NGSI de Java. A continuación, se describe cada una de las actividades.

- Desarrollo de la librería NGSI de JavaScript. En esta actividad se desarrolló la librería NGSI de JavaScript. La librería NGSI de JavaScript es una herramienta de software que tiene como objetivo el tratamiento de entidades o modelos JSON para convertirlos a modelos de datos NGSI, que puedan ser manipulables y operables por el *Orion ContextBroker* de *FIWARE*. La librería se puede usar en el desarrollo de aplicaciones móviles con *frameworks* que usan JavaScript como lenguaje para desarrollar aplicaciones nativas de Android o IOS, como *React Native* o *Native Script*. Esta librería también se puede implementar en aplicaciones web a través de los servicios web *RESTFul*, con el entorno de ejecución NodeJS.
- Desarrollo del módulo genérico NGSI de Java. En esta actividad se desarrolló el módulo genérico NGSI de Java. El módulo genérico NGSI de Java tiene como objetivo convertir entidades a modelos de datos NGSI, que puedan ser manipulables y operables por el *Orion Context Broker* de *FIWARE*. Este módulo se utiliza en el desarrollo de aplicaciones móviles para Android.

- **Fase 3: Implementación de los componentes en las aplicaciones**

El objetivo de esta tercera fase es la implementación de los componentes genéricos en una aplicación web y móvil para probar su funcionamiento. Esta fase cuenta con tres actividades: arquitectura de las aplicaciones, desarrollo de las aplicaciones y pruebas de los componentes en las aplicaciones. A continuación, se describe cada una de las actividades.

- Arquitectura de las aplicaciones. En esta actividad se diseñó la arquitectura general de la aplicación web y móvil para definir en qué parte de las aplicaciones se implementa cada uno de los componentes genéricos.
- Desarrollo de las aplicaciones. En esta actividad se desarrolló la aplicación web y móvil implementando los componentes genéricos.

- Pruebas de los componentes en las aplicaciones. En esta actividad se probó el correcto funcionamiento de los componentes genéricos en la aplicación web y móvil.

1.6. Estructura de la tesis

La estructura en la que está organizado este trabajo de investigación es la siguiente:

- **Capítulo 2 Marco conceptual.** Este capítulo define los conceptos esenciales que se utilizan a lo largo de este trabajo de investigación.
- **Capítulo 3 Estado del arte.** Este capítulo describe los trabajos de investigación relacionados con validaciones de estructuras de datos en formato JSON.
- **Capítulo 4 Librería NGSI de JavaScript.** Este capítulo detalla la librería NGSI de JavaScript que permite el tratamiento de entidades o modelos JSON para convertirlos a modelos de datos NGSI, que puedan ser manipulables y operables por el *Orion Context Broker* de *FIWARE*.
- **Capítulo 5 Módulo genérico NGSI de Java.** Este capítulo detalla el módulo genérico NGSI de Java que permite convertir entidades a modelos de datos NGSI, que puedan ser manipulables y operables por el *Orion Context Broker* de *FIWARE*.
- **Capítulo 6 Pruebas y resultados.** Este capítulo detalla los criterios utilizados para evaluar los componentes genéricos desarrollados.
- **Capítulo 7 Conclusiones y trabajos futuros.** Este capítulo presenta las contribuciones de este trabajo de investigación y los trabajos futuros.

Capítulo 2

Marco conceptual

En este capítulo se definen los conceptos más relevantes utilizados en el desarrollo del presente trabajo de investigación.

2.1 El internet de las cosas

El internet de las cosas (IoT por sus siglas en inglés, *Internet of Things*) es un término acuñado por Kevin Ashton, un pionero de la tecnología británica que trabajaba en la identificación por radiofrecuencias (por sus siglas en inglés, RFID). Él concibió un sistema de sensores universales que conectaban el mundo físico a Internet. Además, los tres componentes principales del IoT son: las cosas, Internet y la conectividad, el valor se logra al cerrar la distancia entre el mundo físico y el mundo digital en sistemas que se refuerzan y mejoran automáticamente [26].

Por otra parte, la Corporación Internacional de Datos (IDC por sus siglas en inglés, *International Data Corporation*) define el IoT como una red de redes de puntos finales (o "cosas") que se comunican sin interacción humana mediante conectividad IP. El ecosistema que soporta el IoT incluye una mezcla compleja de tecnologías no limitadas a módulos/dispositivos, conectividad, plataformas IoT, almacenamiento, servidores, seguridad, software analítico y servicios de tecnologías de la información [27].

El objetivo del IoT es lograr que todo artefacto, mediante el uso de sensores y red de datos, pueda conectarse en cualquier momento y lugar con otro dispositivo o persona, todo ello para mantener un monitoreo y control total de los procesos que cada uno de estos artefactos realice [28].

El IoT es una nueva revolución de Internet. Los objetos se hacen reconocibles y obtienen inteligencia haciendo o permitiendo decisiones relacionadas con el contexto gracias al hecho de que pueden comunicar información sobre sí mismos. Pueden acceder a información que ha sido agregada por otras cosas, o pueden ser componentes de servicios complejos [28]. En términos generales, IoT se refiere a la interconexión en red de objetos cotidianos, que a menudo están equipados con inteligencia ubicua [29].

2.2 Ciudad inteligente

Una ciudad inteligente puede definirse como aquella ciudad que usa las tecnologías de la información y las comunicaciones para hacer que tanto su infraestructura crítica, como sus componentes y servicios públicos ofrecidos sean más interactivos, eficientes y los ciudadanos puedan ser más conscientes de ellos. De manera descriptiva, una ciudad inteligente es un espacio urbano con infraestructuras, redes y plataformas inteligentes, con millones de sensores y actuadores, que incluye a las personas y a sus teléfonos móviles. Éste espacio es capaz de escuchar y comprender lo que está pasando en la ciudad por lo que permite tomar mejores decisiones y proporcionar la información y los servicios adecuados a sus habitantes [4].

2.3 SmartSDK

SmartSDK es el *"libro de recetas"* de *FIWARE* para el desarrollo de aplicaciones inteligentes en los dominios de ciudad inteligente, salud inteligente y seguridad inteligente. Este proporciona paquetes de modelos listos para la creación de servicios inteligentes basados en la especificación NGSI v2, todos los componentes adoptados se basan en *FIWARE* y los complementos están disponibles en la comunidad *FIWARE* como código abierto. Además, *SmartSDK* proporciona un conjunto de patrones de arquitectura reutilizables y nativos de la nube para simplificar el desarrollo de los servicios inteligentes [25].

2.4 Plataformas para internet de las cosas

Actualmente se están desarrollando aplicaciones inteligentes que requieren de infraestructura que las soporten, para ello se requieren de plataformas en la nube que cuenten con estos recursos. Entre algunas de las plataformas se encuentran las siguientes: *Google Cloud Platform*, *Amazon Web Services IoT*, *IBM Watson IoT Platform* y *FIWARE*.

2.4.1 Google Cloud Platform

Google Cloud Platform es una plataforma de IoT inteligente que ayuda a obtener información útil para una empresa a partir de la red mundial de dispositivos [8]. Con *Google Cloud Platform*, se puede construir, probar e implementar aplicaciones en la infraestructura altamente escalable y confiable de Google para sus soluciones web, móviles y de *back-end* [30]. Además, *Google Cloud Platform* consta de un conjunto de activos físicos, como computadoras y unidades de disco duro, y recursos virtuales, como máquinas virtuales, que están contenidos en los centros de datos de Google en todo el mundo [31].

2.4.2 Amazon Web Services IoT

Amazon Web Services IoT es una plataforma de la nube administrada que permite a los dispositivos conectados interactuar con facilidad y seguridad con las aplicaciones en la nube y otros dispositivos. *Amazon Web Services IoT* admite miles de millones de dispositivos y billones de mensajes, y es capaz de procesar y enrutar dichos mensajes a puntos de enlace de *Amazon Web Services* y a otros dispositivos de manera fiable y segura. Con *Amazon Web Services IoT*, las aplicaciones pueden realizar un seguimiento de todos los dispositivos y comunicarse con ellos en todo momento, incluso cuando no están conectados [9].

Amazon Web Services IoT facilita la utilización de servicios como *Amazon Web Services Lambda*, *Amazon Kinesis*, *Amazon S3*, *Amazon Machine Learning*, *Amazon DynamoDB*, *Amazon CloudWatch*, *Amazon Web Services CloudTrail* y *Amazon Elasticsearch* con la integración incorporada de *Kibana* para crear aplicaciones de IoT que recopilen, procesen, analicen y utilicen datos generados

por dispositivos conectados sin necesidad de tener que administrar ninguna infraestructura [9].

2.4.3 Watson IoT Platform

Watson IoT Platform proporciona acceso potente a aplicaciones, dispositivos y datos para construir rápidamente aplicaciones analíticas y aplicaciones móviles de IoT [10].

Watson IoT Platform permite realizar operaciones potentes de administración de dispositivos, almacenar y acceder a datos de dispositivos. Además, conecta una amplia variedad de dispositivos. *Watson IoT Platform* proporciona comunicación segura desde y hacia sus dispositivos mediante el uso del protocolo MQTT (por sus siglas en inglés *Message Queue Telemetry Transport*) y seguridad de la capa de transporte (TLS por sus siglas en inglés, *Transport Layer Security*) [10].

Watson IoT Platform se comunica con sus aplicaciones y dispositivos mediante la API y el protocolo de mensajería. El panel de *Watson IoT Platform* se conecta como una interfaz de usuario frontal para simplificar las operaciones dentro de la plataforma. Los datos del dispositivo se pueden almacenar o usar con soluciones de análisis [10].

2.4.4 FIWARE

FIWARE es una plataforma que tiene como objetivo facilitar el desarrollo de aplicaciones inteligentes en diversos sectores verticales, tales como, agricultura, cuidado de la salud, transporte, casas inteligentes, industria, etc., mediante una infraestructura abierta basada en la nube que ofrece un catálogo de componentes genéricos (GEs por sus siglas en inglés, *Generic Enablers*). Cada componente genérico ofrece una serie de funciones de propósito general a través de interfaces de programación de aplicaciones (APIs por sus siglas en inglés, *Aplication Programming Interface*) de licencias públicas y libres [5].

Además, el catálogo *FIWARE* contiene una rica biblioteca de componentes genéricos con implementaciones de referencia que permiten a los desarrolladores poner en práctica funcionalidades como la conexión a IoT o el análisis de *Big Data* [32]. Al combinar cada uno de los componentes genéricos se puede comenzar a desarrollar aplicaciones inteligentes de inmediato.

La Figura 3 muestra cómo los diferentes componentes genéricos de *FIWARE* se combinan para formar una arquitectura que resuelve buena parte de las necesidades de una ciudad inteligente. Puede observarse cómo el *Orion Context Broker* es un elemento esencial en la arquitectura, puesto que recoge datos de servicios externos, IoT, análisis de flujos de media, etc., y lo pone a disposición para su análisis en tiempo real, consulta de históricos o análisis de históricos (*Big Data*) [33].

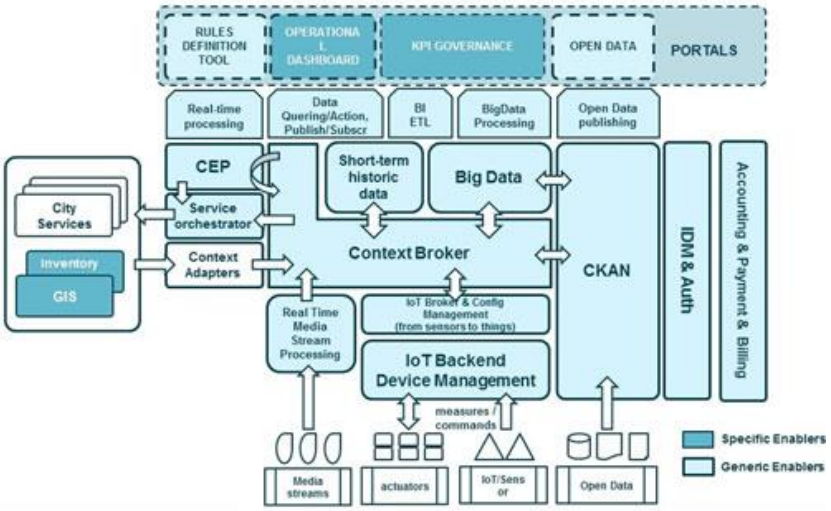

Figura 3 Arquitectura para ciudades inteligentes FIWARE [33]

En cualquier solución inteligente existe la necesidad de recopilar y gestionar información de contexto, procesar esa información e informar a los actores externos, lo que les permite actuar y, por lo tanto, alterar o enriquecer el contexto actual. El componente de *FIWARE* llamado *Orion Context Broker* es el componente central. Permite al sistema realizar actualizaciones y acceder al estado actual del contexto [32].

2.4.4.1 Orion Context Broker

Orion Context Broker es uno de los componentes más importantes de *FIWARE*. Este componente permite administrar todo el ciclo de vida de la información de contexto, incluidas actualizaciones, consultas, registros y suscripciones [34]. Además, la información de contexto se representa a través de los valores asignados a los atributos que caracterizan a las entidades relevantes para su aplicación. El *Orion Context Broker* puede manejar información de contexto a gran escala mediante la implementación de una *API REST* [35], como se muestra en la Figura 4.

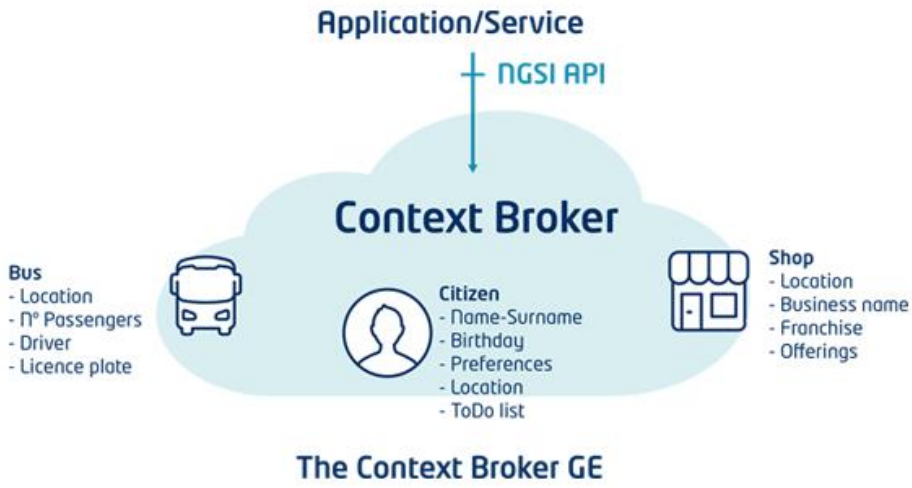

Figura 4 Esquema del Context Broker [35]

Por otra parte, el *Orion Context Broker* recibe la información de los productores de contexto (por ejemplo, sensores) y notifica a los consumidores (por ejemplo, aplicaciones móviles o web), mediante una suscripción a la información de contexto para que cuando ocurra alguna condición (por ejemplo, la información de contexto haya cambiado) el consumidor reciba una notificación, como se muestra en la Figura 5. Además, el *Orion Context Broker* implementa la especificación FIWARE-NGSI v2.

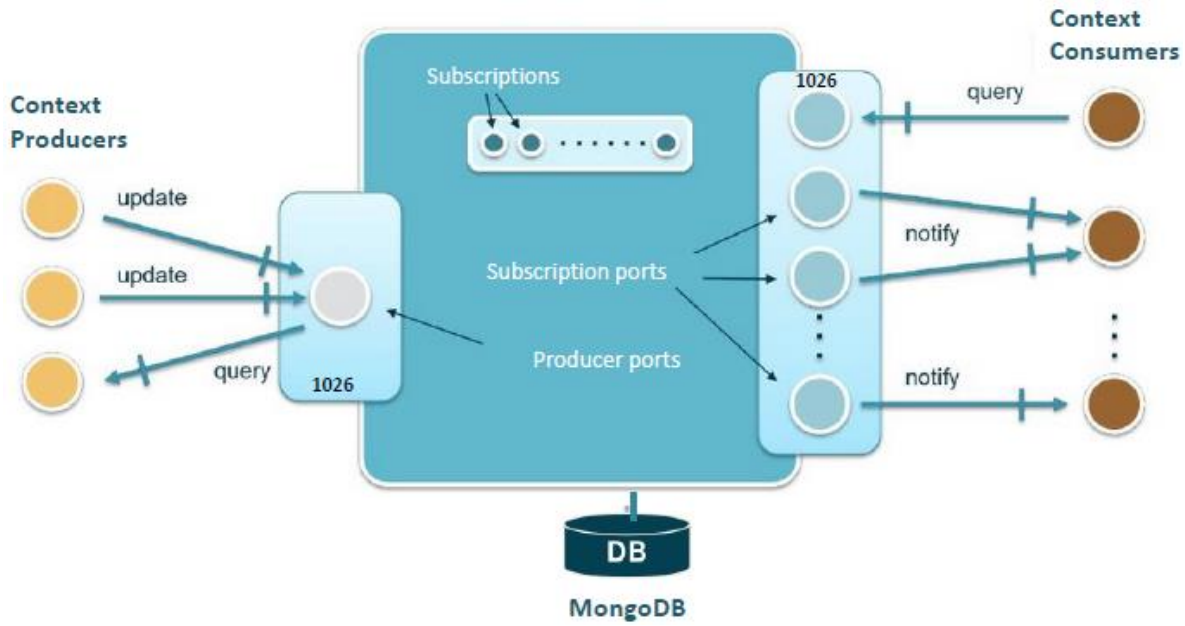

Figura 5 Arquitectura lógica del Orion Context Broker [36]

2.4.4.2 Especificación FIWARE-NGSI v2

La especificación FIWARE-NGSI (*Next Generation Service Interface* por su nombre en inglés, Interfaz de servicio de próxima generación) v2 está diseñada

para describir la administración de todo el ciclo de vida de la información de contexto, incluidas actualizaciones, consultas, registros y suscripciones [13]. La especificación FIWARE-NGSI v2 describe tres conceptos principales de los modelos de datos NGSI: entidades de contexto, atributos y metadatos, los cuales se muestran en la Figura 6.

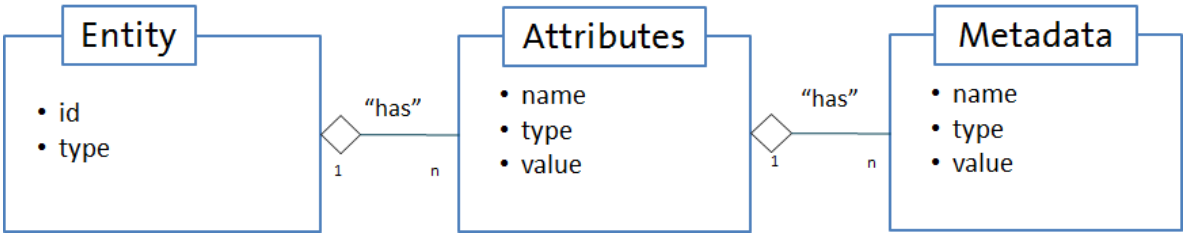

Figura 6 Elementos del modelo de datos FIWARE NSGI [13]

Las entidades de contexto: representan objetos o lugares del mundo real (por ejemplo, un sensor, una persona, una habitación, etc.) y mediante la especificación del estándar FIWARE NGSI estos son encapsulados en modelos de datos NGSI. Además, cada entidad tiene un atributo *id* que lo identifica y, mediante la combinación de los atributos *id* y *type,* es posible identificar de manera única a una entidad entre un grupo de entidades de contexto [13].
Los atributos de contexto: son propiedades que describen las características de las entidades de contexto. En el modelo de datos NGSI, los atributos tienen un nombre de atributo, un tipo de atributo, un valor de atributo y metadatos [13].
Los metadatos de contexto: son piezas de información que se encargan de describir características adicionales al atributo de contexto de una entidad. Estos metadatos son agregados como datos opcionales del valor de un atributo de una entidad, y al igual que los atributos, los metadatos poseen un nombre, un tipo y valor [13].

2.4.4.3 Gestión de entidades

Gestión de entidades (IdM por su nombre en inglés, *Identity Management*) cubre una serie de aspectos relacionados con el acceso de los usuarios a redes, servicios y aplicaciones, incluida la autenticación segura y privada de usuarios a dispositivos, redes y servicios. Además, el gestor de entidades se utiliza para autorizar servicios externos para acceder a datos personales almacenados en un entorno seguro. Por lo general, el propietario de los datos debe dar su consentimiento para acceder a los datos; el procedimiento de otorgamiento de consentimiento también implica cierta autenticación de usuario [37].

2.4.4.4 *QuantumLeap*

QuantumLeap es una API que admite el almacenamiento de datos de *FIWARE NGSI-v2* en una base de datos de series temporales [38]. En la Figura 7 se muestra la manera en que se utiliza *QuantumLeap*.

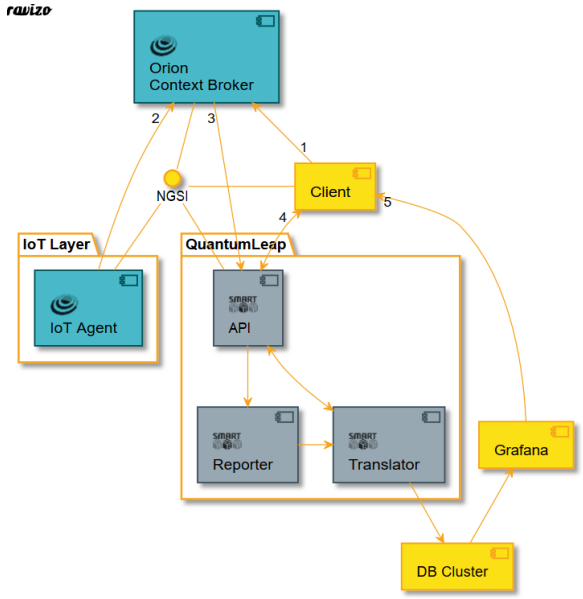

Figura 7 Ejemplo del uso de QuantumLeap [38]

La idea de *QuantumLeap* es bastante simple se utiliza el mecanismo de las notificaciones es decir se implementan las suscripciones, los clientes le dan instrucciones al *Orion Context Broker* para que notifique a *QuantumLeap* los cambios en las entidades que les interesan [38].

2.4.4.5 *Suscripciones*

Es la capacidad de suscribirse a la información de contexto para que cuando ocurra algo la aplicación reciba una notificación asincrónica. De esta forma, no es necesario que repita continuamente las solicitudes de consulta (es decir, el sondeo), el *Orion Context Broker* le informará la información cuando se presente algo [39].

Para crear una suscripción, se utiliza el método *POST* haciendo referencia a la dirección donde se encuentra el *Orion Context Broker* por ejemplo *localhost:1026/v2/subscriptions*. En la Tabla 2.1 se muestra un ejemplo de una suscripción.

Tabla 2.1 Ejemplo de suscripción

```
{
        "description": "A subscription to get info about Room1",
        "subject": {
                "entities": [{
                        "id": "Room1",
                        "type": "Room"
                }],
                "condition": {
                        "attrs": [
                                "pressure"
                        ]
                }
        },
        "notification": {
                "http": {
                        "url": "http://localhost:1028/accumulate"
                },
                "attrs": [
                        "temperature"
                ]
        },
        "expires": "2040-01-01T14:00:00.00Z",
        "throttling": 5
}
```

2.5 Desarrollo basado en componentes

La reutilización de componentes de software es un proceso inspirado en la manera en que se producen y ensamblan componentes en la ingeniería de sistemas físicos.

¿Qué es un componente de software?

Se han propuesto numerosas definiciones a este término, entre las cuales se destacan las siguientes:

Según Philippe Krutchen [40], un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas.

Según Clemens Szyperski [41], un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros.

Según Herzum y Sims [42], un componente es un artefacto de software autocontenido y claramente identificable que describe y ejecuta funciones específicas; que tiene, además, una interfaz claramente establecida, una documentación apropiada y un estatus de uso recurrente bien definido.

2.6 Librería

Una librería es una colección de recursos no volátiles utilizados por programas de computadora, a menudo para el desarrollo de software. Estos pueden incluir datos de configuración, documentación, datos de ayuda, plantillas de mensaje, código escrito previamente y subrutinas, clases, valores o especificaciones de tipo [43].

Una librería también es una colección de implementaciones de comportamiento, escrita en términos de un lenguaje, que tiene una interfaz bien definida mediante la cual se invoca el comportamiento. Por ejemplo, las personas que desean escribir un programa de nivel superior pueden usar una librería para realizar llamadas al sistema en lugar de implementar esas llamadas al sistema una y otra vez. Además, el comportamiento se proporciona para su reutilización por múltiples programas independientes. Un programa invoca el comportamiento proporcionado por la librería a través de un mecanismo del lenguaje [43].

El código de la biblioteca está organizado de tal manera que puede ser utilizado por múltiples programas que no tienen conexión entre sí, mientras que el código que es parte de un programa está organizado para ser utilizado solo dentro de ese programa. Esta distinción puede ganar una noción jerárquica cuando un programa crece en grande, como un programa de varios millones de líneas. En ese caso, puede haber bibliotecas internas que sean reutilizadas por sub porciones independientes del programa grande. La característica distintiva es que una biblioteca está organizada con el propósito de ser reutilizada por programas o subprogramas independientes, y el usuario solo necesita conocer la interfaz, y no los detalles internos de la biblioteca [43].

El valor de una biblioteca radica en la reutilización del comportamiento. Cuando un programa invoca una biblioteca, gana el comportamiento implementado dentro de esa biblioteca sin tener que implementar ese comportamiento en sí mismo. Las bibliotecas fomentan el intercambio de código de forma modular y facilitan la distribución del código [43].

2.7 JavaScript

JavaScript es el lenguaje interpretado orientado a objetos desarrollado por Netscape que se utiliza en millones de páginas web y puede ser ejecutado en un servidor (tales como, *Node.js*, *Apache CouchDB* y *Adobe Acrobat*) [44]. El núcleo de JavaScript puede extenderse para varios propósitos, por ejemplo:

- JavaScript a lado del cliente, extiende el núcleo del lenguaje proporcionando objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales de *Document Object Model*). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales

como clic del ratón, ingreso de datos al formulario y navegación de páginas [45].

- JavaScript a lado del servidor, extiende el núcleo del lenguaje proporcionando objetos relevantes a la ejecución de JavaScript en un servidor. Por ejemplo, las extensiones del lado del servidor permiten que una aplicación se comunique con una base de datos o efectuar manipulación de archivos en un servidor [45].

Por otra parte, JavaScript es un lenguaje de programación dinámico que soporta construcción de objetos basado en prototipos. La sintaxis básica es similar a Java y C++ con la intención de reducir el número de nuevos conceptos necesarios para aprender el lenguaje. Además, JavaScript puede funcionar como lenguaje procedimental y como lenguaje orientado a objetos. Los objetos se crean programáticamente añadiendo métodos y propiedades a lo que de otra forma serían objetos vacíos en tiempo de ejecución, en contraposición a las definiciones sintácticas de clases comunes en los lenguajes compilados como C++ y Java. Una vez se ha construido un objeto, puede usarse como modelo (o prototipo) para crear objetos similares [46].

2.8 Node.js

Node.js es un entorno de ejecución para JavaScript del lado del servidor (pero no limitándose a ello). Se basa en el motor de Chrome denominado V8, es el motor de código abierto de JavaScript de alto rendimiento de Google, escrito en C ++ [47]. Además, node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente.

Node.js utiliza el ecosistema de gestión de paquetes de librerías de código abierto más grande del mundo llamado npm.

- npm es el administrador de paquetes para JavaScript y el registro de software más grande del mundo [48], con aproximadamente 3 mil millones de descargas por semana. El registro contiene más de 600,000 paquetes (bloques de construcción de código) [49]. Además, npm consta de tres componentes distintos:
    - El sitio web: para descubrir paquetes, configurar perfiles y administrar otros aspectos de su experiencia npm. Por ejemplo, puede configurar Orgs (organizaciones) para gestionar el acceso a paquetes públicos o privados.
    - La interfaz de línea de comando: se ejecuta desde una terminal. Así es como la mayoría de los desarrolladores interactúan con npm.
    - El registro: es una gran base de datos pública de software JavaScript y la metainformación que lo rodea.

2.9 Big data

Big data se puede describir en torno a desafíos de administración de datos que, debido al incremento en el volumen, la velocidad y la variedad de los datos, no se puede resolver con las bases de datos tradicionales [50]. Si bien hay muchas definiciones para *Big data*, la mayoría incluye el concepto de lo que comúnmente se conoce como las "tres V" de *Big data*:

- Volumen: oscila entre *terabytes* y *petabytes* de datos.
- Variedad: Incluye datos de una amplia variedad de fuentes y formatos (por ejemplo, registros web, interacciones en las redes sociales, transacciones online y de comercio electrónico, transacciones financieras, etc.).
- Velocidad: Cada vez más, las empresas tienen requisitos exigentes desde el momento en que se generan los datos al momento en que se entrega información accionable a los usuarios. Por lo tanto, es necesario recopilar, almacenar, procesar y analizar los datos en periodos relativamente cortos de tiempo, que van desde una vez al día o en tiempo real.

2.10 Cómputo en la nube

La computación en la nube es una red de servidores remotos compartidos y recursos informáticos alojados en Internet. Es una forma de usar Internet para sus aplicaciones en lugar de su disco duro o servidor local, lo que libera su computadora para hacer otras cosas, particularmente si la capacidad del disco duro es limitada [51].

Hay tres tipos básicos de servicios en la nube:

1. Infraestructura como servicio
   Cuando un proveedor brinda a los clientes acceso a almacenamiento, redes, servidores y a otros servicios de computación que están en la nube, por los que el cliente pagará según su consumo [52].
2. Plataforma como servicio
   Cuando un proveedor de servicios ofrece acceso a un entorno basado en la nube, donde los usuarios pueden construir y entregar aplicaciones. El proveedor suministra la infraestructura subyacente [52].
3. Software como servicio
   Cuando un proveedor de servicios entrega software y aplicaciones a través de Internet. Los usuarios se suscriben al software y acceden al mismo a través de la web o de las API del proveedor [52].

Capítulo 3

Estado del arte

Este capítulo tiene como objetivo presentar un panorama general del estado del arte en las áreas de investigación que se consideraron importantes para el desarrollo de este trabajo de investigación.

Los componentes genéricos que se desarrollaron en este trabajo de investigación permiten la conexión de aplicaciones de dispositivos móviles a una plataforma IoT. Estos componentes genéricos utilizan el formato JSON como intercambio de datos entre la plataforma IoT.

A continuación, se presenta una colección de trabajos relacionados a librerías y algoritmos que gestionan información en formato JSON.

3.1 Un enfoque para la extracción de esquemas JSON y colecciones de documentos JSON extendidos

Los documentos JSON están surgiendo como un formato común para la representación de datos debido a la creciente popularidad de las bases de datos orientadas a documentos NoSQL. Una de las razones de tal popularidad es su capacidad para manejar grandes volúmenes de datos ante la ausencia de un esquema de datos explícito. Sin embargo, la información de esquema a veces es esencial para las aplicaciones durante la recuperación de datos, las tareas de integración y análisis.

Este trabajo de investigación [53], presenta un enfoque llamado *JSON Schema Discovery*. Su propósito es extraer esquemas de JSON o colecciones de documentos JSON extendidos. La motivación principal de este trabajo es contribuir con tareas complejas, como la integración y recuperación de datos a partir de fuentes de datos sin esquema, en particular, las colecciones JSON que normalmente se mantienen mediante la base de datos orientada a documentos NoSQL u otro repositorio de documentos. Una vez que se descubre un esquema de estas fuentes de datos, es posible saber qué atributos, tipos de datos y restricciones están asociados a los datos brutos.

JSON Schema Discovery contribuye al descubrimiento de esquemas para datos JSON al permitir, al mismo tiempo, la extracción automática de esquema de documentos JSON y JSON extendidos en formato de esquema JSON, y la extracción del nombre de atributo, tipo de datos y restricción de participación (atributo obligatorio / opcional). Además, este proceso de extracción considera operaciones de resumen de datos basadas en una estructura de datos que mantiene varios metadatos sobre los datos del documento para fines de extracción de esquemas, así como operaciones de agregación que mejoran el rendimiento de la tarea de extracción de esquemas. Sin embargo, este enfoque está disponible como una herramienta de software libre que puede generar, almacenar y visualizar esquemas JSON a partir de colecciones de datos almacenadas en la base de datos NoSQL de MongoDB.

3.2 Extracción de esquemas y detección de valores atípicos estructurales para almacenes de datos NoSQL basados en JSON

La mayoría de los almacenamientos de datos NoSQL carecen de un esquema, la información sobre las propiedades estructurales de los datos persistentes es, sin embargo, esencial durante el desarrollo de la aplicación. De lo contrario, acceder a los datos simplemente no es práctico.

En este trabajo de investigación [54], se presenta un algoritmo para la extracción de esquemas que está operando fuera del almacén de datos NoSQL. Este método está específicamente dirigido a datos semi-estructurados que persisten en almacenamientos NoSQL, por ejemplo, en formato JSON. En lugar de diseñar el esquema desde el principio, extraer un esquema en retrospectiva puede verse como un paso de ingeniería inversa. Con base en la información del esquema extraído, se propone un conjunto de medidas de similitud que capturan el grado de heterogeneidad de los datos JSON y que revelan valores atípicos estructurales en los datos.

Se probó este enfoque en diferentes colecciones JSON administradas en MongoDB. En las pruebas de detección de valores atípicos se encontraron varios documentos atípicos que no conocían previamente los propietarios de estos datos. Conocer los valores atípicos es un prerrequisito importante para mejorar la calidad de los datos.

3.3 Investigación sobre la traducción de XSD a esquema JSON

La eficiencia de intercambio de datos del XML tradicional es baja y su versatilidad es baja, mientras que JSON tiene una mayor eficiencia de transmisión en comparación con XML. Como un formato liviano de intercambio de datos, JSON se está volviendo cada vez más popular, pero no tan común como XML para su última aparición en la aplicación de servicios web.

XSD describe la estructura de un documento XML y, por lo general, se usa para verificar si un documento XML cumple con sus requisitos. En otras palabras, el diseñador de documentos puede usar el esquema XML para especificar la estructura y el contenido de un documento XML, y para verificar si un documento XML es válido.

En este trabajo de investigación [55], se propuso un método para traducir los datos del formato XSD al formato de esquema JSON y los cuales fueron probados con algunos documentos de prueba. Además, se pueden convertir los datos del formato XSD al esquema JSON atravesando el nodo del documento XSD y configurando el diccionario de atributos de datos para XSD en la conversión al esquema JSON.

3.4 Mison: Un analizador rápido de JSON para análisis de datos

En este trabajo de investigación [56], se presenta un analizador JSON llamado Mison que está especialmente adaptado a aplicaciones donde el análisis de JSON domina el rendimiento y el costo, presionando hacia abajo ambas proyecciones y operadores de filtros para consultas analíticas en el analizador.

Para lograr estas características, se descartó el enfoque tradicional de máquinas de estados finitos para la construcción del analizador. En su lugar, se utilizó un enfoque de dos niveles que permite al analizador saltar directamente a la posición correcta de un campo consultado sin tener que realizar costosos pasos de tokenización para encontrar el campo. En el nivel superior, Mison predice especulativamente las ubicaciones lógicas de campos consultados basándose en patrones vistos previamente en un conjunto de datos. En el nivel inferior, Mison crea índices estructurales en datos JSON para asignar ubicaciones lógicas a ubicaciones físicas.

Experimentalmente Mison fue evaluado utilizando conjuntos de datos JSON representativos del mundo real demostrando que Mison produce importantes beneficios de rendimiento sobre los mejores analizadores JSON existentes; en algunos casos, la mejora del rendimiento es de más de un orden de magnitud.

3.5 SJSON: Una representación concisa de los documentos de notación de objetos de JavaScript

Este trabajo de investigación [57], se centra en describir y analizar SJSON; la cual es una librería que explora representaciones concisas de documentos JSON como un medio para reducir el uso de memoria de los archivos en la memoria principal, y para permitir la compresión de los archivos JSON almacenados en el disco. En la librería SJSON se representa la estructura del documento con árboles concisos, a diferencia de la implementación habitual basada en punteros.

Como resultado, se desarrollaron algoritmos capaces de representar de manera consistente documentos JSON a través de conjuntos de datos del mundo real utilizando hasta 82% menos de memoria RAM en comparación con bibliotecas populares, la mayoría de las veces utilizando menos espacio que el tamaño del archivo original. Además, el análisis empírico muestra que SJSON generó archivos comprimidos hasta un 41% más pequeño que el documento original.

3.6 Jsongen: una librería basada en QuickCheck para probar JSON en servicios web

Este trabajo de investigación [58], describe un enfoque sistemático para probar los aspectos de comportamiento de los servicios web que se comunican utilizando el formato de datos JSON. Como componente clave, la herramienta de prueba basada en propiedades Quviq QuickCheck se utiliza para generar

automáticamente una gran cantidad de casos de prueba a partir de una descripción abstracta del comportamiento del servicio en forma de una máquina de estados finitos. Para generar datos JSON aleatorios para rellenar las pruebas se desarrolló la librería jsongen, que da una caracterización a los datos JSON como un esquema JSON.

La librería jsongen, deriva automáticamente un generador QuickCheck que puede generar un número infinito de valores JSON que validan contra el esquema y proporciona una máquina de estado QuickCheck genérica que es capaz de seguir los enlaces documentados en el esquema JSON, para explorar automáticamente el servicio web.

El comportamiento predeterminado de la máquina de estado se puede personalizar fácilmente para incluir comprobaciones específicas del servicio web. Además, el artículo ilustra el enfoque al desarrollar un modelo de máquina de estado finito para la prueba de un servicio web basado en JSON.

3.7 Interoperabilidad basada en JSON aplicando el modelo de programación pull-parser

El formato JSON se ha aplicado en una variedad de aplicaciones: se establece como el estándar por defecto para la representación del almacenamiento de documentos; es ampliamente utilizado para lograr la interoperabilidad, y como el formato de intercambio en las *API web RESTful*.

En este trabajo de investigación [59], se presenta un enfoque que utiliza el modelo de datos JSON como formato de manejo para la interoperabilidad con distintos modelos de datos NoSQL. Se aprovecha su estructura textual anidada para aplicar el modelo de programación *pull-parser* para procesarlo y desarrollar traductores entre JSON y un conjunto de formatos NoSQL representativos. Se centra en la extracción de JSON y en el desarrollo y la aplicación de las transformaciones de datos. Se validó este enfoque a través de una implementación que maneja una gran cantidad de estrategias de representación de datos.

Contribuciones del trabajo [59]:

- Utiliza modelos de datos anidados JSON como base para la interoperabilidad entre diferentes formatos de datos NoSQL. La utilización de JSON ha confirmado para ser una elección efectiva, ya que tiene mucho soporte para varias API, lo que facilita la conexión a diferentes almacenes de datos de salida.
- Utilizó el modelo de programación *pull-parser*, que ya se ha utilizado en el contexto XML, para leer la entrada de un flujo de objetos. Esto permite tener grandes archivos como entrada, ya que no necesita mantener los objetos de entrada en la memoria. La traducción en sí misma está libre de contexto, si los objetos JSON son documentos anidados bien formados

3.8 Convertir conjuntos de datos a JSON

En este trabajo de investigación [60], se presenta un algoritmo para convertir un conjunto de datos a formato JSON. Este enfoque puede usarse para inferir un tipo preciso desde cero para una colección de objetos JSON, así como para validar un conjunto de datos contra un tipo diseñado por el ser humano y, si es necesario, para adaptar y mejorar este tipo.

El algoritmo propuesto en este trabajo de investigación está formado por dos etapas:

- En la primera etapa, un trabajo de asignación / reducción procesa todo el conjunto de datos e infiere, para cada objeto JSON, un tipo. En particular, durante la fase de mapeo, los mapeadores examinan todos los objetos y, para cada objeto, devuelven un registro que contiene un tipo inferido como la clave.

  Antes de que comience la fase para reducir, los registros emitidos por los mapeadores se agrupan comparando sus claves por medio de un algoritmo de equivalencia de tipo estructural.

  En la fase reducir, cada reductor cuenta el número de elementos para cada clase de equivalencia.

  El resultado de la primera etapa es entonces un conjunto de pares $<T_i; m_i>$, de modo que el tipo T_i describe con precisión el conjunto de datos de entrada, y cada m_i cuenta los valores de entrada descritos por T_i.
- El objetivo de la segunda etapa es colapsar tipos similares mediante el uso de un algoritmo de tipo de fusión, sin perder demasiada precisión. Esta segunda fase producirá una nueva colección más refinada y de menor tamaño.

3.9 Una biblioteca de Python para acceso y almacén de FAIRer para el Metabolomics Workbench Data Repository

El *Metabolomics Workbench Data Repository* es un repositorio público de espectrometría de masas, datos y metadatos de resonancia magnética nuclear derivados de una amplia variedad de estudios de metabólica. Los datos y metadatos de cada estudio se depositan, almacenan y acceden a través de archivos del domino específico 'mwTab', que son archivos en formato plano.

En este trabajo de investigación [61], se implementó una biblioteca y un paquete en Python para mejorar la accesibilidad, reutilización e interoperabilidad de los datos y metadatos almacenados en archivos con formato plano. Este paquete de Python, llamado 'mwtab', es un analizador sintáctico para el formato de archivo 'mwTab' de dominio específico, que proporciona facilidades para leer, acceder y escribir archivos formateados 'mwTab'. Además, el paquete proporciona recursos para convertir archivos 'mwTab' en un formato JSON equivalente, lo que permite la reutilización fácil de los datos por todos los lenguajes

de programación modernos que implementan los analizadores JSON. El paquete 'mwtab' implementa su funcionalidad de validación de metadatos basada en un esquema JSON predefinido que puede ser fácilmente especializado para tipos específicos de estudios de metabolómica. La biblioteca también proporciona una interfaz de línea de comandos para la interconversión entre formatos 'mwTab' y JSONized en texto sin procesar y una variedad de formatos de archivos binarios comprimidos.

Capítulo 4

Librería NGSI de JavaScript

En este capítulo se presenta la librería NGSI (por sus siglas en inglés, *Next Generation Service Interface*) de JavaScript desarrollada en esta tesis.

La librería NGSI de JavaScript se encuentra enmarcada por la especificación FIWARE-NGSI v2 y se desarrolló en el lenguaje de programación JavaScript.

Respecto a la especificación FIWARE-NGSI v2 describe la administración de todo el ciclo de vida de la información de contexto, incluidas actualizaciones, consultas, registros y suscripciones. Además, esta especificación cuenta con tres elementos principales de los modelos de datos NGSI: entidades de contexto, atributos y metadatos, como se muestra en la Figura 8.

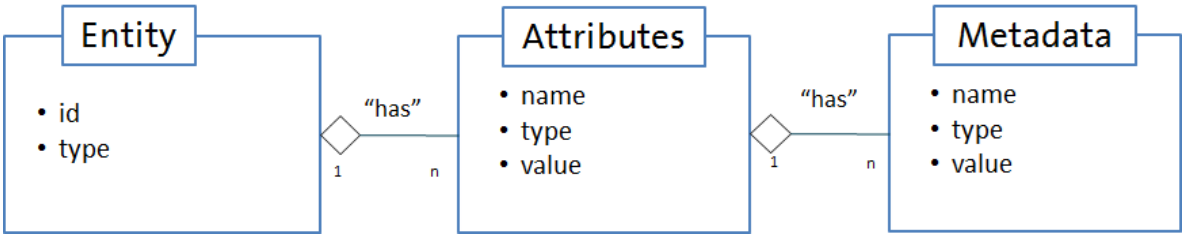

Figura 8 Elementos del modelo de datos FIWARE NSGI [13]

A continuación, se describe cada uno de los elementos de los modelos de datos NGSI:

- **Las entidades de contexto**: representan objetos o lugares del mundo real (por ejemplo, un sensor, una persona, una habitación, etc.) y mediante la especificación FIWARE NGSI v2 estos son encapsulados en modelos de datos NGSI. Además, cada entidad tiene un atributo *id* que la identifica y, mediante la combinación de los atributos *id* y *type,* es posible identificar de manera única a una entidad entre un grupo de entidades de contexto.
- **Los atributos de contexto**: son propiedades que describen las características de las entidades de contexto. En el modelo de datos NGSI, los atributos tienen un nombre de atributo, un tipo de atributo, un valor de atributo y metadatos.
- **Los metadatos de contexto**: son piezas de información que se encargan de describir características adicionales al atributo de contexto de una entidad. Estos metadatos son agregados como datos opcionales del valor de un atributo de una entidad, y al igual que los atributos, los metadatos poseen un nombre, un tipo y valor.

En cuanto a, la librería NGSI de JavaScript es una herramienta de software que tiene como objetivo el tratamiento de entidades o modelos JSON para convertirlos a modelos de datos NGSI, que puedan ser manipulables y operables por el *Orion Context Broker* de FIWARE. La librería se puede usar en el desarrollo de aplicaciones móviles con *frameworks* que usan JavaScript como lenguaje para desarrollar aplicaciones nativas de Android o IOS, como *React Native* o *Native Script*. Esta librería también se puede implementar en aplicaciones web a través de los servicios web *RESTFul*, con el entorno de ejecución NodeJS.

Actualmente, se han desarrollado varias bibliotecas con una funcionalidad similar. La principal diferencia es la división de esta librería en dos módulos npm. En este enfoque, los desarrolladores pueden implementar uno o ambos módulos de la librería, debido al bajo acoplamiento entre estos módulos. Esta librería sirve como base a todos los desarrolladores que deseen desarrollar aplicaciones móviles o Web dentro del ecosistema de FIWARE utilizando el *Orion Context Broker* o en plataformas que implemente el *Orion Context Broker*.

Entre las principales funcionalidades que ofrece esta librería se encuentran: conexión y comunicación con una instancia del *Orion Context Broker* de FIWARE, a través de solicitudes CRUD (*Create, Read, Update y Delete*), análisis de objetos JSON, para verificar si estos cumplen con las especificaciones de un modelo de datos NGSI, transformar objetos JSON a entidades de contexto NGSI, evaluación de entidades de contexto NGSI, para validar si la entidad de contexto sigue la especificación de un esquema de modelo de datos de FIWARE y manejo de errores.

4.1 Arquitectura de la librería NGSI de JavaScript

La arquitectura de la librería NGSI de JavaScript está compuesta por dos módulos npm: ngsi-parser y ocb-sender. La Figura 9 muestra la arquitectura de la librería NGSI de JavaScript.

El primer módulo tiene como objetivo, analizar y convertir objetos o atributos JSON no estructurados o de un valor de atributo a la sintaxis de una entidad NGSI que cumpla con la especificación FIWARE-NGSI v2. Además, este módulo proporciona la funcionalidad de verificar si la entidad NGSI cumple con la especificación oficial del esquema JSON de un modelo de datos de FIWARE, con la finalidad de comprobar que la estructura JSON de la entidad de contexto sea considerada como un modelo de datos FIWARE. Este módulo contiene tres elementos principales, los cuales son: JSON Parser, Queries Parser y Data JSON Schema Analizer. A continuación, se describe cada uno de estos elementos.

1. **JSON Parser:** incluye las funciones que son utilizadas para el análisis y conversión de un Objeto JSON no estructurado a uno que cumpla con la especificación FIWARE-NGSI v2.
2. **Queries Parser:** es el encargado de interpretar objetos JSON para generar consultas en formato de cadena de contexto personalizadas para obtener datos específicos del *Orion Context Broker* de FIWARE.
3. **Data JSON Schema Analizer:** es el encargado de determinar si un objeto JSON cumple o no con un modelo de datos y genera una lista de errores con respecto al esquema JSON del modelo de datos.

El segundo módulo tiene como objetivo gestionar (enviar, actualizar, eliminar, etc.) la información de contexto como es el caso de entidades de

contexto NGSI y/o modelos de datos FIWARE a una instancia del Orion Context Broker. Este módulo contiene cuatro elementos principales, los cuales son: Entities Functions, Queries Functions, Subscripcions Functions y HTTP-Client. Los tres primeros elementos contienen las funcionalidades de cliente del *Orion Context Broker*. A continuación, se describe cada uno de estos elementos.

1. **Entities Functions:** contiene las funciones de manipulación de entidades del *Orion Context Broker* de FIWARE.
2. **Queries Functions:** contiene las funciones de consultas personalizadas del *Orion Context Broker* de FIWARE.
3. **Subscripcions Functions:** contiene las funciones de manipulación de suscripciones del *Orion Context Broker* de FIWARE.
4. **HTTP-Client:** es el encargado de la conexión con el *Orion Context Broker* de FIWARE, además es utilizado por el módulo de ngsi-parser para obtener esquemas JSON desde un repositorio remoto para ser utilizados.

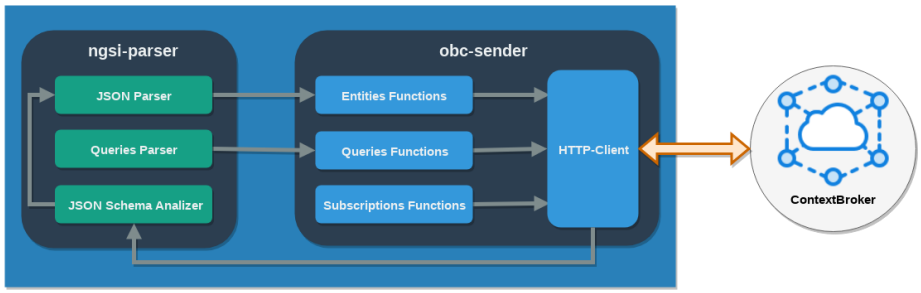

Figura 9 Arquitectura de la librería NGSI de JavaScript

En la siguiente sección se detalla a profundidad cada uno de los módulos.

4.2 Módulos de la librería NGSI de JavaScript

En esta sección se detalla a profundidad cada una de las funcionalidades que constituyen a cada uno de los módulos de la librería NGSI de JavaScript y se muestra un ejemplo de cada una de sus funcionalidades.

4.2.1 Módulo ngsi-parser

La verificación de una entidad NGSI se lleva a cabo en ciertos tipos de datos que soporta el módulo ngsi-parser, estos tipos de datos que soporta son los siguientes:

- Text: se utiliza cuando el valor es una cadena (*string*).
- Number: se utiliza cuando el valor es un número.
- Boolean: se utiliza cuando el valor es un booleano (*true* o *false*).
- DateTime: se utiliza cuando el valor es una fecha.
- StructuredValue: se utiliza cuando el valor es un objeto o un arreglo.
- None: se utiliza cuando el valor es nulo.

Las siguientes subsecciones: funciones con entidades, generar consultas de contexto y analizador de esquema JSON de modelos de datos, detallan cada uno de las funciones que soporta el módulo ngsi-parser.

4.2.1.1 Funciones con entidades

Esta subsección describe cada una de las funciones que analizan gramaticalmente la sintaxis y convierte las entidades a la especificación de FIWARE-NGSI v2, estas funciones son la siguientes: analiza una entidad, analiza atributos de una entidad y analiza el valor de un atributo.

- Analiza una entidad

  La función *parseEntity(jsonEntity)*, recibe como parámetro un objeto JSON de una entidad no estructurada (*jsonEntity*). Esta función analiza gramaticalmente la sintaxis y convierte la entidad para que esta cumpla con la especificación FIWARE-NGSI v2. Cabe decir, que el objeto JSON de una entidad no estructurada obligatoriamente debe llevar el identificador único de la entidad (*id*) y el tipo de entidad (*type*).

  Por ejemplo, la función *parseEntity(jsonEntity)* debe recibir el parámetro *jsonEntity* que es un objeto JSON de una entidad no estructurada (ver Tabla 4.1 columna entidad a analizar), se observa que la entidad cuenta con un identificador único (*id*), un tipo de entidad (*type*) y los atributos: temperatura (*temperature*) y fecha (*dateStamp*).

  El atributo temperatura cuenta con un valor (*value*) pero no indica su tipo de dato (*type*). Además, este atributo cuenta con la propiedad de metadatos (*metadata*) que contiene dos atributos: frecuencia (*frecuency*) y escala (*scale*), a estos dos atributos se le asignan directamente sus valores (40 y '*Celsious*' respectivamente); omitiendo las propiedades valor del atributo (*value*) y tipo de dato (*type*) para cada uno de ellos.

  Al atributo fecha se le asigna directamente la fecha actual (*new Date()*) omitiendo las propiedades valor del atributo (*value*) y tipo de dato (*type*) para este atributo.

  La función *parseEntity(jsonEntity)* corrige estos tipos de inconsistencias en la entidad para que cumpla con la especificación FIWARE-NGSI v2, teniendo como resultado el JSON de la Tabla 4.1 columna salida de la entidad.

Tabla 4.1 Ejemplo de la función que analiza una entidad

Entidad a analizar	Salida de la entidad
`var entity = ngsi.parseEntity({ id: 'Room1', type: 'Room', temperature: { value: 50, metadata: { frecuency: 40, scale: 'Celsious' } }, dateStamp: new Date() })`	`{ "id": "Room1", "type": "Room", "temperature": { "value": 50, "type": "Number", "metadata": { "frecuency": { "value": 40, "type": "Number" }, "scale": { "value": "Celsious", "type": "Text" } } }, "dateStamp": { "value": "2017-10-08T04:01:19.560Z", "type": "DateTime", "metadata": {} } }`

- Analiza atributos de una entidad

  La función *parseAttrs(jsonAttrs)*, recibe como parámetro un objeto JSON con atributos de una entidad no estructurada (*jsonAttrs*). Esta función analiza gramaticalmente la sintaxis y convierte atributos de una entidad a la especificación de FIWARE-NGSI v2.

  Por ejemplo, la función *parseAttrs(jsonAttrs)* debe recibir el parámetro *jsonAttrs* que es un objeto JSON de atributos de una entidad no estructurada (ver Tabla 4.2 columna atributos a analizar), se observa que el JSON que se pasa como parámetro cuenta con el atributo temperatura (*temperature*) con un valor (*value*) pero no indica su tipo de dato (*type*). Además, este atributo cuenta con la propiedad de metadatos (*metadata*) que contiene dos atributos: frecuencia (*frecuency*) y escala (*scale*), a estos dos atributos se le asignan directamente sus valores (50 y ' *Fahrenheit*' respectivamente); omitiendo las propiedades valor del atributo (*value*) y tipo de dato (*type*) para cada uno de ellos.

  La función *parseAttrs(jsonAttrs)* corrige estos tipos de inconsistencias en los atributos de entidades para que cumplan con la especificación FIWARE-NGSI v2, teniendo como resultado el JSON de la Tabla 4.2 columna salida de atributos.

Tabla 4.2 Ejemplo de la función que analiza atributos de una entidad

Atributos a analizar	Salida de atributos

`var attribute = ngsi.parseAttrs({`
`  temperature: {`
`  value: 50,`
`  metadata: {`
`  frecuency: 50,`
`  scale: 'Fahrenheit'`
`  }`
`  }`
`})`
	`{`
	`  "temperature": {`
	`  "value": 50,`
	`  "type": "Number",`
	`  "metadata": {`
	`  "frecuency": {`
	`  "value": 50,`
	`  "type": "Number"`
	`  },`
	`  "scale": {`
	`  "value": "Fahrenheit",`
	`  "type": "Text"`
	`  }`
	`  }`
	`  }`
	`}`

- Analiza el valor de un atributo
  La función *parseValue(valueAttr)*, recibe como parámetro un valor de un atributo (*valueAttr*). Esta función analiza gramaticalmente la sintaxis y convierte el valor de un atributo a la especificación de FIWARE-NGSI v2.
  Por ejemplo, la función *parseValue(valueAttr)* debe recibir el parámetro *valueAttr* que es el valor de un atributo (ver Tabla 4.3 columna valor del atributo a analizar), esta función convierte al atributo a la especificación FIWARE-NGSI v2 añadiéndole las siguientes propiedades: valor del atributo (*value*), tipo de datos del atributo (*type*) y el metadato (*metadata*), el resultado se puede ver en la Tabla 4.3 columna salida del valor del atributo.

Tabla 4.3 Ejemplo de la función que analiza el valor de un atributo

Valor del atributo a analizar	Salida del valor del atributo

`var value = ngsi.parseValue(50)`	`{`
	`  "value": 50,`
	`  "type": "Number",`
	`  "metadata": {}`
	`}`

4.2.1.2 Generar consultas de contexto

Esta subsección describe la manera en que se generan consultas dinámicas convirtiendo un JSON a una consulta. La función que realiza esta tarea es la siguiente: *createQuery(jsonQuery)* recibe como parámetro un JSON con los atributos y sus respectivos valores (*jsonQuery*) para formar una consulta en formato de cadena. Esta consulta posteriormente se utiliza en el módulo ocb-sender para interactuar directamente con el *Orion Context Broker* de FIWARE.

Por ejemplo, la función *createQuery(jsonQuery)* recibe el parámetro *jsonQuery* que es un objeto JSON con los atributos de la consulta (ver Tabla 4.4

columna JSON a convertir). Esta devolverá una consulta en formato de cadena como se muestra en la Tabla 4.4 columna salida de la consulta.

Tabla 4.4 Ejemplo de la función que convierte un JSON a una consulta

JSON a convertir	Salida de la consulta
let query = ngsi.createQuery({ "id": "Device.*", "type": "Device", "options": "keyValues", "dateObserved": ">=2018-02- 20T16:54:03.931-06:00" }) console.log(query)	?id=Device.*&Device&type=Devi ce&options=keyValues&q=dateOb served>=2018-02- 20T16:54:03.931-06:00

En el siguiente ejemplo, se muestra la implementación de la consulta que retorna la función *createQuery(jsonQuery)*. Esta consulta se utiliza en el parámetro de la función *getWithQuery(query)* del módulo ocb-sender, ver la Tabla 4.5. La función *getWithQuery(query)* retorna a todas aquellas entidades que cumplen con la condición de la consulta que se realiza al *Orion Context Broker* de FIWARE.

Tabla 4.5 Ejemplo de la función que consulta al Orion Context Broker

```
cb.getWithQuery(query)
  .then((result) => console.log(result))
  .catch((err) => console.log(err))
```

4.2.1.3 Analizador de esquema JSON de modelos de datos

En esta subsección se describe cada una de las funciones que permiten analizar e identificar errores en entidades generadas por desarrolladores y contrastar con algún esquema de modelo de datos oficial de FIWARE o algún esquema de modelo de dato propio, para que estas entidades cumplan con la especificación FIWARE-NGSI v2 y puedan ser utilizadas por el *Orion Context Broker* de FIWARE. Esta subsección cuenta con dos funcionalidades y estas son: obtener esquemas de modelos de datos y verificar entidades con esquemas de modelos de datos, a continuación, se describe cada una de las subsecciones.

- **Obtener esquemas de modelos de datos**
  Esta funcionalidad describe cada forma en que obtienen esquemas de modelos de datos ya sea remotamente o localmente mediante un archivo JSON, estas funciones son las siguientes: importar un esquema desde un archivo JSON, importar un esquema desde un repositorio remoto e importar varios esquemas desde varias fuentes de origen.

- Importar un esquema desde un archivo JSON
  Esta función permite obtener un esquema de modelo de datos desde un archivo JSON, posteriormente el esquema de modelo de datos se utiliza en el análisis de una determinada entidad.
  Por ejemplo, en la Tabla 4.6 se muestra la forma en que se utiliza la función *setModel(schema)*. En primer lugar, en la variable (*ngsi*) se importa el módulo ngsi-parser, en segundo lugar, en la variable (*myLocalSchema*) se obtiene el esquema de modelo de datos desde un archivo JSON que se encuentra localmente en nuestro proyecto, por último, mediante la variable (*ngsi*) se obtiene la función *setModel(schema)* que recibe como parámetro un JSON con el atributo (*mySchema*) al cual se le asigna el valor de la variable (*myLocalSchema*).

Tabla 4.6 Ejemplo de la función que importa un esquema desde un archivo JSON

```
var ngsi = require('ngsi-parser');
var myLocalSchema = require('./mySchema.json');
ngsi.setModel({
        mySchema: myLocalSchema
});
```

- Importar un esquema desde un repositorio remoto
  Esta función permite obtener un esquema de modelo de datos desde un repositorio remoto, posteriormente el esquema de modelo de datos se utiliza en el análisis de una determinada entidad.
  Por ejemplo, en la Tabla 4.7 se muestra la forma en que se utiliza la función (*setModel(schemas)).* En primer lugar, en la variable (*ngsi*) se importa el módulo ngsi-parser y, en segundo lugar, mediante la variable (*ngsi*) se obtiene la función *setModel(schemas)* que recibe como parámetro un JSON con el atributo (*myRemoteSchema*) al cual se le asigna la dirección del repositorio remoto donde se encuentra el esquema de modelo de datos.

Tabla 4.7 Ejemplo de la función que importa un esquema desde un repositorio remoto

```
var ngsi = require('ngsi-parser');
ngsi.setModel({
        myRemoteSchema: 'https://yourdatamodels.com/myRemote'
});
```

- Importar varios esquemas desde varias fuentes de origen
  Esta función permite obtener uno o más esquemas de modelos de datos ya sea de un repositorio remoto o de un archivo JSON que se encuentra localmente en el proyecto, posteriormente los esquemas de modelos de datos se utilizan en el análisis de una determinada entidad.
  Por ejemplo, en la Tabla 4.8 se muestra la forma en que se utiliza la función (*setModel(schemas)*). En primer lugar, en la variable (*ngsi*) se importa el

módulo ngsi-parser, en segundo lugar, en la variable (*myLocalSchema*) se obtiene el esquema de modelo de datos desde un archivo JSON que se encuentra localmente en nuestro proyecto, por último, mediante la variable (*ngsi*) se obtiene la función *setModel(schemas)* que recibe el parámetro *schemas* que es un JSON con los atributos (*mySchema, myRemoteSchema, AlertModel, anotherModel*) para obtener en cada atributo un esquemas de modelos de datos ya sea remoto o localmente mediante un archivo JSON.

Tabla 4.8 Ejemplo de la función que importa varios esquemas desde varias fuentes de origen

```
var ngsi = require('ngsi-parser');
var myLocalSchema = require('mySchema.json');
ngsi.setModel({
        mySchema: myLocalSchema,
        myRemoteSchema: 'https://yourdatamodels.com/myRemote',
        AlertModel:
'https://raw.githubusercontent.com/smartsdk/dataModels/master/Alert/schem
a.json',
        anotherModel: require('anotherSchema.json')
});
```

Después de describir las diferentes formas en que se obtienen los esquemas de los modelos de datos. A continuación, se presenta la funcionalidad de verificar entidades con esquemas de modelos de datos.

- **Verificar entidades con esquemas de modelos de datos**
  Esta funcionalidad permite describir cada una de las formas en que se verifica una entidad y cumpla con un esquema de modelo de datos bajo la especificación FIWARE-NGSI v2. Estas formas que verifican una entidad son las siguientes: usando esquemas de modelos de datos desde un archivo JSON y usando esquemas de modelos de datos desde un repositorio remoto.

  - Usando esquemas de modelos de datos desde un archivo JSON
    Esta forma verifica una entidad a partir de un esquema de modelo de datos desde un archivo JSON que se encuentra almacenado localmente. Para realizar esta verificación se utiliza la función *verifyModel(Schema, entity)* que recibe como parámetros un esquema (*Schema*) que se obtiene desde un archivo JSON y la entidad (*entity*) que se desea contrastar con el esquema de modelos de datos.
    Por ejemplo, en la Tabla 4.9 se muestra la implementación de la función *verifyModel(Schema, entity)*. En primer lugar, se declara la variable (*ngsi*) donde se importa el módulo ngsi-parser, esta variable sirve para acceder a cada una de las funciones del módulo, en segundo lugar, se declara la variable (*mySchema*) esta variable sirve para obtener el esquema de modelo de datos que se encuentra en el archivo *AlertSchema.json,* para

más detalle de la estructura del esquema de modelo de datos para este ejemplo consultar el Anexo 1, en tercer lugar, mediante la variable (*ngsi*) se obtiene la función *setModel(schemas)* que recibe como parámetro un JSON con el atributo (*mySchema*) al cual se le asigna el valor de la variable (*mySchema*) que contiene el esquema del modelo de datos que se encuentra en el archivo *AlertSchema.json*, en cuarto lugar, se declara la variable (*entity*) que contiene la entidad que se contrasta con el esquema de modelo de datos, en este ejemplo se está utilizando el modelo de datos *Alert*, por último, se declara una variable (*errors*) que se le asigna el valor que retorna la función *verifyModel('mySchema', entity),* esta retorna un arreglo de todas las inconsistencias que tiene la entidad en caso de que no coincida la sintaxis con el esquema de modelo de datos.

Tabla 4.9 Ejemplo de la función para usar esquemas de modelos de datos desde un archivo JSON

```
var ngsi = require('ngsi-parser');
var mySchema = require('./AlertSchema.json');
ngsi.setModel({
        mySchema: mySchema
});
var entity = {
        id: "Alert:Device_Smartphone_7a85d9df7209b8bc:1519086635021",
        type: "Alert",
        alertSource: "Device_Smartphone_7a85d9df7209b8bc",
        category: "traffic",
        dateObserved: new Date(),
        description: "Car Accident on Cenidet",
        location: {
                type: "geo:point",
                value: "18.81186166666667 ,-98.96342000000001"
        },
        severity: "medium",
        subCategory: "carAccident",
        validFrom: new Date(),
        validTo: new Date(),
        dateCreated: new Date()
};
let errors = ngsi.verifyModel('mySchema', entity);
if (errors.length === 0) {
        console.log("The entity it's OK")
} else {
        errors.map(console.log)
}
```

- Usando esquemas de modelos de datos desde un repositorio remoto
  Esta forma verifica una entidad a partir de un esquema de modelo de datos desde un repositorio remoto (para este caso se está utilizando el repositorio de los modelos de datos de FIWARE). Para realizar esta verificación se utiliza la función *verifyModel(Schema, entity, ocb)* que recibe como parámetros un esquema (*Schema*) que se obtiene desde un repositorio remoto, la entidad (*entity*) que se desea contrastar con el esquema de modelo de datos y el

ocb que nos permite hacer uso de las funciones del módulo ocb-sender para descargar el esquema de modelo de datos desde el repositorio remoto.
Por ejemplo, en la Tabla 4.10 se muestra la implementación de la función *verifyModel(Schema, entity, ocb)*. En primer lugar, se declara las variables (*ngsi y ocb*) donde se importan los módulos ngsi-parser y ocb-sender respectivamente, estas variables sirven para acceder a cada una de las funciones de los módulos, en segundo lugar, mediante la variable (*ngsi*) se obtiene la función *setModel(schemas)* que recibe como parámetro un JSON con el atributo (*Alert*) al cual se le asigna la dirección remota donde se encuentra el esquema del modelo de datos, en tercer lugar, se declara la variable (*alertEntity*) que contiene la entidad que se contrasta con el esquema del modelo de datos, en este ejemplo se está utilizando el modelo de datos *Alert*, por último, con la variable (*ngsi*) se obtiene la función *verifyModel(Schema, entity, ocb)*, esta retorna un arreglo de todas las inconsistencias que tiene la entidad en caso de que no coincida la sintaxis con el esquema de modelo de datos.

Tabla 4.10 Ejemplo de la función para usar esquemas de modelos de datos desde un repositorio remoto

```
var ngsi = require('ngsi-parser');
var ocb = require('ocb-sender');

ngsi.setModel({
      Alert:
'https://raw.githubusercontent.com/smartsdk/dataModels/master/Alert/schema.json'
})

var alertEntity = {
   id: "Alert:Device_Smartphone_7a85d9df7209b8bc:1519086635022",
   type: "Alert",
   alertSource: "Device_Smartphone_7a85d9df7209b8bc",
   category: "traffic",
   dateObserved: new Date(),
   description: "Car Accident on Cenidet",
   location: {
       type: "geo:point",
       value: "18.81186166666667 ,-98.96342000000001"
   },
   severity: "medium",
   subCategory: "carAccident",
   validFrom: new Date(),
   validTo: new Date(),
   dateCreated: new Date()
}

ngsi.verifyModel('Alert', alertEntity, ocb)
  .then((errors) => {
     if (errors.length === 0) {
          console.log("The entity it's OK");
          var ngsiEntity = ngsi.parseEntity(alertEntity);
          ocb.createEntity(ngsiEntity)
           .then((result) => console.log(result))
                .catch((err) => console.log(err))
          } else {
               errors.map(console.log)
             }
          })
```

4.2.2 Módulo ocb-sender

Este módulo tiene como objetivo gestionar (enviar, actualizar, eliminar, etc.) la información de contexto de entidades NGSI y/o modelos de datos FIWARE en una instancia del *Orion Context Broker* de FIWARE. Además, el módulo cuenta con tres grupos de funciones. Estos grupos son los siguientes:

4.2.2.1 Funciones de gestión de entidades

- Función para crear entidad
  createEntity(entity)
- Funciones para obtener información de entidades
  - Obtiene el valor de un atributo de una entidad

    ```
    getEntityAttributeValue(idEntity, attribute)
    ```
  - Obtiene un atributo de una entidad

    ```
    getEntityAttribute(idEntity,attribute)
    ```
  - Obtiene los atributos de una entidad

    ```
    getEntityAttrs(idEntity)
    ```
  - Obtiene una entidad

    ```
    getEntity(idEntity)
    ```
  - Obtiene una lista de entidades por su tipo

    ```
    getEntityListType(typeEntity)
    ```
  - Obtiene todas las entidades

    ```
    listEntities()
    ```
- Funciones para actualizar entidades
  - Actualiza el valor de un atributo de una entidad

    ```
    updateEntityAttributeValue(idEntity, nameObjectAttribute, val)
    ```
  - Actualiza los datos de atributos de una entidad

    ```
    updateJSONAttrEntity(idEntity, nameAttribute,jsonAttr)
    ```
  - Reemplaza todos los atributos de una entidad

    ```
    replaceAllEntityAttributes(idEntity, jsonObjectAttrs)
    ```
  - Actualiza los atributos existentes de una entidad

    ```
    updateEntityAttrs(idEntity,jsonObjectAttrs)
    ```
  - Actualiza o adjunta atributos a una entidad

    ```
    addJSONAttributeToEntity(idEntity, JSONAttribute)
    ```
- Funciones para eliminar
  - Elimina una entidad

    ```
    deleteEntity(idEntity)
    ```
  - Elimina un atributo de una entidad

    ```
    deleteEntityAttribute(idEntity, attribute)
    ```

4.2.2.2 Funciones de consultas dinámicas

- Consultas dinámicas de contexto

```
getWithQuery(query)
```

- Consultas de entidades dentro de un área

```
queryEntitiesOnArea(coordsPolygon, idEntity, entityType,
optionskeyValues)
```

4.2.2.3 Funciones de gestión de suscripciones

- Función para crear una suscripción

```
createSubscription(subscription)
```

- Funciones para obtener información de suscripciones
  - Obtiene todas las suscripciones

```
listSubscriptions()
```

  - Obtiene una suscripción

```
getSubscription(idSubscription)
```

- Funciones para actualizar suscripciones
  - Actualiza una suscripción

```
updateSubscription(idSubscription, jsonSubscription)
```

  - Actualiza el estado de una suscripción

```
updateSubscriptionStatus(idSubscription, status)
```

- Función para eliminar suscripción

```
deleteSubscription(idSubscription)
```

A continuación, se describe detalladamente cada uno de los tipos de funciones.

4.2.2.1 Funciones de gestión de entidades

Este grupo de funciones describe cada una de las funciones que gestionan entidades en el *Orion Context Broker* de FIWARE. Además, este grupo cuenta con cuatro subgrupos de funciones: funciones para crear entidades, funciones para obtener información de entidades, funciones para actualizar entidades y funciones para eliminar. A continuación, se describe cada uno de estos subgrupos de funciones.

- Función para crear entidad
  La función *createEntity(entity)*, permite crear una entidad en el *Orion Context Broker* de FIWARE y recibe como parámetro una entidad (*entity*) en formato JSON que cumpla con la especificación FIWARE-NGSI v2.

En la Tabla 4.11 se muestra un ejemplo de la implementación de la función *createEntity* que recibe una entidad de tipo ***Device*** con sus respectivos atributos para generarla en el *Orion Context Broker* de FIWARE.

Tabla 4.11 Ejemplo de la función para crear entidad

```
cb.createEntity({
        "id": "Device_Smartphone_40bbcd9c6cae67c0",
        "type": "Device",
        "batteryLevel": {
                "type": "Number",
                "value": 69,
                "metadata": {}
        },
        "category": {
                "type": "Text",
                "value": "smartphone",
                "metadata": {}
        },
        "dateCreated": {
                "type": "DateTime",
                "value": "2018-04-12T02:06:10.00Z",
                "metadata": {}
        },
        "dateModified": {
                "type": "Text",
                "value": "2018-05-08T01:24:16Z",
                "metadata": {}
        },
        "ipAddress": {
                "type": "Text",
                "value": "192.168.1.71",
                "metadata": {}
        },
        "location": {
                "type": "geo:point",
                "value": "18.8699251, -99.2119878",
                "metadata": {}
        },
        "osVersion": {
                "type": "Text",
                "value": "7.0",
                "metadata": {}
        },
        "owner": {
                "type": "Text",
                "value": "User_1524067170397",
                "metadata": {}
        },
        "refDeviceModel": {
                "type": "Text",
                "value": "DeviceModel_motorola_MotoG5Plus",
                "metadata": {}
        },
        "serialNumber": {
                "type": "Text",
                "value": "ZY22488CGH",
                "metadata": {}
        }
}).then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Funciones para obtener información de entidades
  Estas funciones permiten obtener información de entidades del *Orion Context Broker* de FIWARE. Estas funciones son las siguientes:

  - Obtiene el valor de un atributo de una entidad
    La función *getEntityAttributeValue(idEntity, attribute)*, permite obtener el valor de un atributo de una entidad y recibe como parámetros el identificador de la entidad (*idEntity*) y el nombre del atributo (*attribute*).
    En la Tabla 4.12 se muestra un ejemplo de la implementación de la función *getEntityAttributeValue* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, para obtener como respuesta el valor (*value*) del atributo ***batteryLevel***.

Tabla 4.12 Ejemplo de la función que obtiene el valor de un atributo de una entidad

Implementación:
```
cb.getEntityAttributeValue("Device_Smartphone_40bbcd9c6cae67c0",
"batteryLevel")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

Respuesta de la función:
```
{ value: 73, attribute: 'batteryLevel' }
```

  - Obtiene un atributo de una entidad
    La función *getEntityAttribute(idEntity,attribute)*, permite obtener un atributo de una entidad y recibe como parámetros el identificador de la entidad *(idEntity)* y el nombre del atributo (*attribute*).
    En la Tabla 4.13 se muestra un ejemplo de la implementación de la función *getEntityAttribute* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, para obtener como respuesta el atributo ***batteryLevel***.

Tabla 4.13 Ejemplo de la función que obtiene un atributo de una entidad

Implementación:
```
cb.getEntityAttribute("Device_Smartphone_40bbcd9c6cae67c0",
"batteryLevel")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

Respuesta de la función:
```
{ type: 'Number', value: 69, metadata: {} }
```

- Obtiene los atributos de una entidad
  La función *getEntityAttrs(idEntity)*, permite obtener todos los atributos de una entidad y recibe como parámetro el identificador de la entidad (*idEntity*).
  En la Tabla 4.14 se muestra un ejemplo de la implementación de la función *getEntityAttrs* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, para obtener como respuesta todos los atributos de dicha entidad.

Tabla 4.14 Ejemplo de la función que obtiene los atributos de una entidad

Implementación:

```
cb.getEntityAttrs("Device_Smartphone_40bbcd9c6cae67c0")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

Respuesta de la función:

```
{
          batteryLevel: {
                    type: 'Number',
                    value: 69,
                    metadata: {}
          },
          category: {
                    type: 'Text',
                    value: 'smartphone',
                    metadata: {}
          },
          dateCreated: {
                    type: 'DateTime',
                    value: '2018-04-12T02:06:10.00Z',
                    metadata: {}
          },
          dateModified: {
                    type: 'Text',
                    value: '2018-05-08T01:24:16Z',
                    metadata: {}
          },
          ipAddress: {
                    type: 'Text',
                    value: '192.168.1.71',
                    metadata: {}
          },
          location: {
                    type: 'geo:point',
                    value: '18.8699251, -99.2119878',
                    metadata: {}
          },
          osVersion: {
                    type: 'Text',
                    value: '7.0',
                    metadata: {}
          },
          owner: {
                    type: 'Text',
                    value: 'User_1524067170397',
                    metadata: {}
          },
          refDeviceModel: {
                    type: 'Text',
                    value: 'DeviceModel_motorola_MotoG5Plus',
                    metadata: {}
          }
}
```

- Obtiene una entidad

  La función *getEntity(idEntity)*, permite obtener una entidad específica y recibe como parámetro el identificador de la entidad (*idEntity*) a buscar.

  En la Tabla 4.15 se muestra un ejemplo de la implementación de la función *getEntity* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, para obtener como respuesta la entidad completa.

Tabla 4.15 Ejemplo de la función que obtiene una entidad

Implementación:

```
cb.getEntity('Device_Smartphone_40bbcd9c6cae67c0')
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

Respuesta de la función:

```
{
          id: 'Device_Smartphone_40bbcd9c6cae67c0',
          type: 'Device',
          batteryLevel: {
                    type: 'Number',
                    value: 69,
                    metadata: {}
          },
          category: {
                    type: 'Text',
                    value: 'smartphone',
                    metadata: {}
          },
          dateCreated: {
                    type: 'DateTime',
                    value: '2018-04-12T02:06:10.00Z',
                    metadata: {}
          },
          dateModified: {
                    type: 'Text',
                    value: '2018-05-08T01:24:16Z',
                    metadata: {}
          },
          location: {
                    type: 'geo:point',
                    value: '18.8699251, -99.2119878',
                    metadata: {}
          },
          osVersion: {
                    type: 'Text',
                    value: '7.0',
                    metadata: {}
          },
          owner: {
                    type: 'Text',
                    value: 'User_1524067170397',
                    metadata: {}
          },
          refDeviceModel: {
                    type: 'Text',
                    value: 'DeviceModel_motorola_MotoG5Plus',
                    metadata: {}
          }
}
```

- Obtiene una lista de entidades por su tipo
  La función *getEntityListType(typeEntity)*, permite obtener una lista de entidades por el tipo de entidad a la que pertenecen y recibe como parámetro el tipo de la entidad (*typeEntity*).
  En la Tabla 4.16 se muestra un ejemplo de la implementación de la función *getEntityListType* que recibe como entrada el tipo (*type*) ***Device***, para obtener como respuesta a todas aquellas entidades que pertenezcan al tipo *Device*.

Tabla 4.16 Ejemplo de la función que obtiene una lista de entidades por su tipo

Consulta:

```
cb.getEntityListType('Device')
.then((entities) => {console.log(entities)})
.catch((err) => console.log(err))
```

Respuesta de la consulta:

```
[{
                id: 'MyNuevoDevice',
                type: 'Device',
                position: {
                        type: 'geo:point',
                        value: '18.811747, -98.963506',
                        metadata: {}
                },
                pressure: {
                        type: 'Integer',
                        value: 90,
                        metadata: {}
                }
        },
…
…
…,
{
                id: 'Device_Smartphone_40bbcd9c6cae67c0',
                type: 'Device',
                batteryLevel: {
                        type: 'Number',
                        value: 69,
                        metadata: {}
                },
                category: {
                        type: 'Text',
                        value: 'smartphone',
                        metadata: {}
                },
                dateCreated: {
                        type: 'DateTime',
                        value: '2018-04-12T02:06:10.00Z',
                        metadata: {}
                },
                dateModified: {
                        type: 'Text',
                        value: '2018-05-08T01:24:16Z',
                        metadata: {}
                },
                location: {
                        type: 'geo:point',
                        value: '18.8699251, -99.2119878',
                        metadata: {}
                }
        }
]
```

- Obtiene todas las entidades

  La función *listEntities()*, permite obtener todas las entidades que se encuentran en el *Orion Context Broker* de FIWARE.

  En la Tabla 4.17 se muestra un ejemplo de la implementación de la función *listEntities* que obtiene como respuesta a todas las entidades que se encuentren en el *Orion Context Broker* de FIWARE.

Tabla 4.17 Ejemplo de la función que obtiene una lista con todas las entidades

Consulta:

```
cb.listEntities()
.then((entities) => {console.log(entities)})
.catch((err) => console.log(err))
```

Respuesta de la consulta:

```
[{
            id: 'Device_Smartphone_40bbcd9c6cae67c0',
            type: 'Device',
            batteryLevel: {
                      type: 'Number',
                      value: 69,
                      metadata: {}
            },
            category: {
                      type: 'Text',
                      value: 'smartphone',
                      metadata: {}
            },
            dateCreated: {
                      type: 'DateTime',
                      value: '2018-04-12T02:06:10.00Z',
                      metadata: {}
            },
            dateModified: {
                      type: 'Text',
                      value: '2018-05-08T01:24:16Z',
                      metadata: {}
            },
            location: {
                      type: 'geo:point',
                      value: '18.8699251, -99.2119878',
                      metadata: {}
            },
            osVersion: {
                      type: 'Text',
                      value: '7.0',
                      metadata: {}
            },
            owner: {
                      type: 'Text',
                      value: 'User_1524067170397',
                      metadata: {}
            },
            refDeviceModel: {
                      type: 'Text',
                      value: 'DeviceModel_motorola_MotoG5Plus',
                      metadata: {}
            }
          }
…,
…n
]
```

- Funciones para actualizar entidades
  Estas funciones permiten actualizar información de contexto de entidades en el *Orion Context Broker* de FIWARE. Estas funciones son las siguientes:

  - Actualiza el valor de un atributo de una entidad
    La función *updateEntityAttributeValue(idEntity, nameObjectAttribute, val)*, permite actualizar el valor de un atributo de una entidad y recibe como parámetros el identificador de la entidad (*idEntity*), el nombre del atributo (*nameObjectAttribute*) que se actualizará y el valor a actualizar (*val*).
    En la Tabla 4.18 se muestra un ejemplo de la implementación de la función *updateEntityAttributeValue* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, el atributo a actualizar ***batteryLevel*** y el nuevo valor del atributo **80**.

Tabla 4.18 Ejemplo de la función que actualiza el valor de un atributo de una entidad

```
cb.updateEntityAttributeValue('Device_Smartphone_40bbcd9c6cae67c0'
, 'batteryLevel', 80)
.then((result) => {console.log(result)})
.catch((err) => console.log(err))
```

  - Actualiza los datos de un atributo de una entidad
    La función *updateJSONAttrEntity(idEntity, nameAttribute,jsonAttr)*, permite actualizar los datos de un atributo de una entidad cambiando el tipo de dato (*type*) y valor (*value*) del atributo. La función recibe como parámetros el identificador de la entidad (*idEntity*), el nombre del atributo (*nameAttribute*) y un JSON con los datos del atributo a actualizar (*jsonAttr*).
    En la Tabla 4.19 se muestra un ejemplo de la implementación de la función *updateJSONAttrEntity* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, el atributo a actualizar ***batteryLevel*** y el JSON con los datos a actualizar.

Tabla 4.19 Ejemplo de la función que actualiza los datos de un atributo de una entidad

```
cb.updateJSONAttrEntity('Device_Smartphone_40bbcd9c6cae67c0',
'batteryLevel', {
    "type": "Text",
    "value": "90"
})
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Reemplaza todos los atributos de una entidad
  La función *replaceAllEntityAttributes(idEntity, jsonObjectAttrs)*, permite reemplazar todos los atributos de una determinada entidad por el JSON de los nuevos atributos que se envían. La función recibe como parámetros el identificador de la entidad (*idEntity*) y el JSON con los nuevos atributos (*jsonObjectAttrs*).
  En la Tabla 4.20 se muestra un ejemplo de la implementación de la función *replaceAllEntityAttributes* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0*** y el JSON con los datos a reemplazar.

Tabla 4.20 Ejemplo de la función que reemplaza todos los atributos de una entidad

```
cb.replaceAllEntityAttributes("Device_Smartphone_40bbcd9c6cae67c0"
, {
    "batteryLevel": {
                "type": "Number",
                "value": 90,
                "metadata": {}
        },
        "category": {
                "type": "Text",
                "value": "smartphone",
                "metadata": {}
        },
        "refDeviceModel": {
                "type": "Text",
                "value": "DeviceModel_motorola_MotoG5Plus",
                "metadata": {}
        }
})
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Actualiza los atributos existentes de una entidad
  La función *updateEntityAttrs(idEntity,jsonObjectAttrs)*, permite actualizar los datos de los atributos existentes de una determinada entidad. La función recibe como parámetros el identificador de la entidad (*idEntity*) y el JSON con los datos de los atributos a actualizar (*jsonObjectAttrs*).
  En la Tabla 4.21 se muestra un ejemplo de la implementación de la función *updateEntityAttrs* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0*** y el JSON con los datos de los atributos a actualizar.

Tabla 4.21 Ejemplo de la función que actualiza los atributos existentes de una entidad

```
cb.updateEntityAttrs('Device_Smartphone_40bbcd9c6cae67c0', {
    "batteryLevel": {
                "type": "Text",
                "value": "98",
                "metadata": {}
        },
        "category": {
                "type": "Text",
                "value": "Smartphone",
                "metadata": {}
        },
        "refDeviceModel": {
                "type": "Text",
                "value": "DeviceModel_samsung_s9",
                "metadata": {}
        }
})
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Actualiza o adjunta atributos a una entidad
  La función *addJSONAttributeToEntity(idEntity, JSONAttribute)*, permite actualizar y añadir atributos al mismo tiempo a una entidad específica. La función recibe como parámetros el identificador de la entidad (*idEntity*) y el JSON con los atributos que se actualizarán y se añadirán (*JSONAttribute*).
  En la Tabla 4.22 se muestra un ejemplo de la implementación de la función *addJSONAttributeToEntity* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0*** y el JSON con los datos de los atributos.

Tabla 4.22 Ejemplo de la función que actualiza o adjunta atributos a una entidad

```
cb.addJSONAttributeToEntity("Device_Smartphone_40bbcd9c6cae67c0",{
        "location": {
                "type": "geo:point",
                "value": "18.8699251, -99.2119878",
                "metadata": {}
        },
        "refDeviceModel": {
                "type": "Text",
                "value": "DeviceModel_samsung_s9",
                "metadata": {}
        },
        "batteryLevel": {
                "type": "Number",
                "value": 69,
                "metadata": {}
        }
})
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Funciones para eliminar
  Estas funciones permiten eliminar una entidad o un atributo de una entidad en el *Orion Context Broker* de FIWARE. Estas funciones son las siguientes:

  - Elimina una entidad
    La función *deleteEntity(idEntity)*, permite eliminar una entidad que se encuentre en el *Orion Context Broker* de FIWARE y recibe como parámetro el identificador de la entidad (*idEntity*) a eliminar.
    En la Tabla 4.23 se muestra un ejemplo de la implementación de la función *deleteEntity* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***.

Tabla 4.23 Ejemplo de la función para eliminar una entidad

```
cb.deleteEntity("Device_Smartphone_40bbcd9c6cae67c0")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

  - Elimina un atributo de una entidad
    La función *deleteEntityAttribute(idEntity, attribute)*, permite eliminar un atributo de una determinada entidad que se encuentre en el *Orion Context Broker* de FIWARE y recibe como parámetro el identificador de la entidad (*idEntity*) y el nombre del atributo (*attribute*) a eliminar.
    En la Tabla 4.24 se muestra un ejemplo de la implementación de la función *deleteEntityAttribute* que recibe como entrada el identificador ***Device_Smartphone_40bbcd9c6cae67c0***, el atributo a eliminar ***batteryLevel***.

Tabla 4.24 Ejemplo de la función para eliminar un atributo de una entidad

```
cb.deleteEntityAttribute("Device_Smartphone_40bbcd9c6cae67c0", "
batteryLevel")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

4.2.2.2 Funciones de consultas dinámicas

Este grupo de funciones describe cada una de las funciones que permiten realizar consultas personalizadas al Orion *Context Broker* de FIWARE de entidades.

- Consultas dinámicas de contexto
  La función *getWithQuery(query)*, permite realizar consultas dinámicas a través de una cadena que funciona como una condicional para obtener

todas las entidades que cumplan dicha condición. Esta función recibe como parámetro la cadena condicional (*query*).
En la Tabla 4.25 se muestra un ejemplo de la implementación de la función *getWithQuery* donde se declara la cadena condicional ***query*** y recibe como entrada dicha consulta ***query*.**

Tabla 4.25 Ejemplo de la función que permite realizar consultas dinámicas de contexto

Cadena condicional:

```
let query =
"?id=.*&type=Device&georel=coveredBy&q=owner==Idowner&geometry=polygon&co
ords=18.879751306118546,-99.22197723761204;18.87991373199594,-
99.22199869528413;18.87996449005033,-
99.22190750017762;18.879984793267777,-
99.2218270339072;18.879939111025056,-
99.22174656763676;18.879893428769883,-
99.22165537253022;18.87973100287282,-99.22145152464509;18.8795888800837,-
99.22130132094026;18.879390923140832,-99.221076015383;18.87928940666914,-
99.22097945585847;18.87893917436966,-
99.22117793932557;18.87856356210443,-
99.22121012583375;18.878675230703656,-
99.22134960070255;18.878776747547473,-
99.22145152464509;18.87888841600463,-
99.22154808416965;18.87903053938793,-
99.22144079580903;18.879203117619838,-
99.22140860930085;18.87936554402868,-
99.22153199091554;18.87948228791276,-
99.22165537253022;18.879614259162025,-
99.22181630507114;18.879751306118546,-
99.22197723761204&options=keyValues"

cb.getWithQuery(query)
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Consultas de entidades dentro de un área
  La función *queryEntitiesOnArea(coordsPolygon, idEntity, entityType, optionskeyValues)*, permite determinar si un determinado tipo de entidad se encuentra dentro de un área determinada. Esta función recibe los siguientes parámetros: las coordenadas del polígono (*coordsPolygon*), el identificador de la entidad (*idEntity*) o expresión de las entidades, el tipo de entidad (*entityType*) y un valor booleano (*optionskeyValues*), si el valor booleano es true la función retorna la información en formato keyValues y en caso contrario retorna la información en la especificación FIWARE-NGSI v2. A continuación, en la Tabla 4.26 se muestra la función con los parámetros que recibe.

Tabla 4.26 Ejemplo de la función que consulta entidades dentro de un área

```
cb.queryEntitiesOnArea(coordsPolygon, idEntity, entityType, optionskeyValues)
.then((result) => console.log(JSON.stringify(result)))
.catch((err) => console.log(err))
```

En la Tabla 4.27 se muestra un ejemplo de la implementación de la función *queryEntitiesOnArea* que recibe como entradas las coordenadas del polígono **[[18.879751306118546,-99.22197723761204],…,[n]]**, la expresión **".*"** y el tipo de entidad "***Device***" estos dos parámetros indican que se obtendrán todas las entidades de tipo ***Device*** y por último el valor booleano ***true*** indica que la información que nos retorna la función es en formato keyValues.

Tabla 4.27 Ejemplo de implementación de la función que consulta entidades dentro de un área

```
cb.queryEntitiesOnArea([
    [18.879751306118546,-99.22197723761204],
    [18.87991373199594,-99.22199869528413],
    [18.87996449005033,-99.22190750017762],
    [18.879984793267777,-99.22182703390 72],
    [18.879939111025056,-99.22174656763676],
    [18.879893428769883,-99.22165537253022],
    [18.87973100287282,-99.22145152464509],
    [18.8795888800837,-99.22130132094026],
    [18.879390923140832,-99.221076015383],
    [18.87928940666914,-99.22097945585847],
    [18.87893917436966,-99.22117793932557],
    [18.87856356210443,-99.22121012583375],
    [18.878675230703656,-99.22134960070255],
    [18.878776747547473,-99.22145152464509],
    [18.87888841600463,-99.22154808416965],
    [18.87903053938793,-99.22144079580903],
    [18.879203117619838,-99.22140860930085],
    [18.87936554402868,-99.22153199091554],
    [18.87948228791276,-99.22165537253022],
    [18.879614259162025,-99.22181630507114],
    [18.879751306118546,-99.22197723761204]
],".*","Device",true)
.then((result) => console.log(JSON.stringify(result)))
.catch((err) => console.log(err))
```

4.2.2.3 Funciones de gestión de suscripciones

Una de las características importantes del *Orion Context Broker* es la implementación de suscripciones, que es la capacidad de acceder a la información de contexto para que cuando ocurra algo, su aplicación reciba una notificación asincrónica. Es decir, el *Orion Context Broker* se encarga de notificar cuando se presenta algún cambio en la información de contexto.

Este grupo de funciones describe cada una de las funciones que gestionan suscripciones en el *Orion Context Broker* de FIWARE. Además, este grupo cuenta con cuatro subgrupos de funciones: función para crear una suscripción, funciones para obtener información de suscripciones, funciones para actualizar suscripciones y función para eliminar suscripción. A continuación, se describe cada uno de estos subgrupos de funciones.

- Función para crear una suscripción
  La función *createSubscription(subscription)*, permite crear una suscripción en el *Orion Context Broker* de FIWARE y recibe como parámetro una suscripción (*subscription*) en formato JSON.

En la Tabla 4.28 se muestra un ejemplo de la implementación de la función *createSubscription* que recibe como entrada una suscripción para generarla en el *Orion Context Broker* de FIWARE. Esta suscripción monitorea a todas las entidades de tipo ***Alert*** que cuenten con los atributos en la ***condition*** para enviar la notificación (***notification***) de la información a un determinado servicio (***url***).

Tabla 4.28 Ejemplo de la función para crear una suscripción

```
cb.createSubscription({
      "description": "Alert subscription TEST",
      "subject": {
            "entities": [{
               "idPattern": ".*",
               "type": "Alert"
            }],
            "condition": {
               "attrs": [
                      "id",
                      "type",
                      "category",
                      "subCategory",
                      "location",
                      "address",
                      "dateObserved",
                      "validFrom",
                      "validTo",
                      "description",
                      "alertSource",
                      "data",
                      "severity"
               ]
            }
      },
      "notification": {
            "attrs": [
               "id",
               "type",
               "category",
               "subCategory",
               "location",
               "address",
               "dateObserved",
               "validFrom",
               "validTo",
               "description",
               "alertSource",
               "data",
               "severity"
            ],
            "attrsFormat": "normalized",
            "http": {
               "url": "http://service.mx"
            }
      },
      "throttling": 5
  })
  .then((result) => console.log(result))
  .catch((err) => console.log(err))
```

- Función para obtener información de suscripciones
  Estas funciones permiten obtener información de suscripciones del *Orion Context Broker* de FIWARE. Estas funciones son las siguientes:

- Obtiene todas las suscripciones
  La función *listSubscriptions()*, permite obtener todas las suscripciones que se encuentran en el *Orion Context Broker* de FIWARE.
  En la Tabla 4.29 se muestra un ejemplo de la implementación de la función *listSubscriptions* que obtiene como respuesta a todas las suscripciones que se encuentren en el *Orion Context Broker* de FIWARE.

  Tabla 4.29 Ejemplo de la función que obtiene todas las suscripciones

```
cb.listSubscriptions()
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Obtiene una suscripción
  La función *getSubscription(idSubscription)*, permite obtener una suscripción específica y recibe como parámetro el identificador de la suscripción (*idSubscription*) a buscar.
  En la Tabla 4.30 se muestra un ejemplo de la implementación de la función *getSubscription* que recibe como entrada el identificador **5a83c5463fc4dec59e4ef8e2**, para obtener como respuesta la suscripción.

  Tabla 4.30 Ejemplo de la función que obtiene una suscripción

```
cb.getSubscription("5a83c5463fc4dec59e4ef8e2")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Funciones para actualizar suscripciones
  Estas funciones permiten actualizar información de suscripciones en el *Orion Context Broker* de FIWARE. Estas funciones son las siguientes:

  - Actualiza una suscripción
    La function *updateSubscription(idSubscription, jsonSubscription)*, permite actualizar los datos de una determinada suscripción. La función recibe como parámetros el identificador de la suscripción (*idSubscription*) y el JSON con los datos de los atributos a actualizar (*jsonSubscription*) de la suscripción.
    En la Tabla 4.31 se muestra un ejemplo de la implementación de la función *updateSubscription* que recibe como entrada el identificador **5a93a9063fc4dec59e4ef8eb** y el JSON con los datos de los de la suscripción.

Tabla 4.31 Ejemplo de la función que actualiza una suscripción

```
cb.updateSubscription("5a93a9063fc4dec59e4ef8eb", {
    "description": "Alert subscription TEST",
    "subject": {
    "entities": [
        {
            "idPattern": ".*",
            "type": "Alert"
        }
    ],
    "condition": {
        "attrs": [
            "id",
            "type",
            "location",
            "address",
            "dateObserved",
            "validFrom",
            "validTo",
            "description",
        ]
        }
    },
    "notification": {
        "attrs": [
            "id",
            "type",
            "category",
            "subCategory",
            "location",
            "address",
            "dateObserved",
            "validFrom",
            "validTo",
            "description",
            "alertSource",
            "data",
            "severity"
        ],
        "attrsFormat": "normalized",
        "http": {
            "url": "http://crateservice.com"
        }
    },
    "throttling": 5
})
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Actualiza el estado de una suscripción

  La función *updateSubscriptionStatus(idSubscription, status)*, permite actualizar el estado de una determinada suscripción. La función recibe como parámetros el identificador de la suscripción (*idSubscription*) y el estado (*status*) de la suscripción. El estado de la

suscripción puede tomar uno de los siguientes valores: *active, inactive, expired o failed*.
En la Tabla 4.32 se muestra un ejemplo de la implementación de la función *updateSubscriptionStatus* que recibe como entrada el identificador **5a81e50a3fc4dec59e4ef8dc** y el estado ***active*** de la suscripción.

Tabla 4.32 Ejemplo de la función que actualiza el estado de una suscripción

```
cb.updateSubscriptionStatus("5a81e50a3fc4dec59e4ef8dc",
"active")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

- Function para eliminar suscripción
  La función *deleteSubscription(idSubscription)*, permite eliminar una suscripción que se encuentre en el *Orion Context Broker* de FIWARE y recibe como parámetro el identificador de la suscripción (*idSubscription*) a eliminar.
  En la Tabla 4.33 se muestra un ejemplo de la implementación de la función *deleteSubscription* que recibe como entrada el identificador **5a93a9103fc4dec59e4ef8ec**.

Tabla 4.33 Ejemplo de la function para eliminar suscripción

```
cb.deleteSubscription("5a93a9103fc4dec59e4ef8ec")
.then((result) => console.log(result))
.catch((err) => console.log(err))
```

4.3 Implementación de los módulos

El objetivo principal de esta sección es detallar la implementación de cada uno de los módulos de la librería NGSI de JavaScript, hay que tener en cuenta ciertos pre-requisitos y ciertas actividades de instalación e importación de los módulos para que se implementen de una manera correcta.

En cuanto a, la librería NGSI de JavaScript se compone de dos módulos npm: NGSI-parser y OCB-sender, el código de cada uno de los módulos se encuentra en la cuenta de GitHub(https://github.com/cenidetiot/ngsi-js-library).

- El módulo NGSI-parser se encuentra en la siguiente dirección: https://github.com/cenidetiot/NGSI.jsLibrary
- El módulo OCB-sender se encuentra en la siguiente dirección: https://github.com/cenidetiot/OCB.jsLibrary

Por otra parte, la documentación de los módulos se encuentra en la siguiente dirección: https://ngsi-js-library.readthedocs.io/en/latest/ o https://cenidetiot.github.io/ngsi-js-library/

4.3.2 Pre-requisitos

Para utilizar de manera correcta cada uno de los módulos es necesario tener en cuenta los siguientes requisitos: Node.js y npm.

La instalación de Node.js se realiza de varias formas mediante un instalador ejecutable o descargando el código fuente, esto depende del sistema operativo en el que se está instalando. En la página oficial de Node.js se encuentran los archivos de descarga y los pasos a seguir para su instalación, ver el siguiente enlace https://nodejs.org/en/download/. Asimismo, al realizar la instalación de Node.js este incluye el administrador de paquetes npm.

Por lo que se refiere a, la librería NGSI de JavaScript se desarrolló en la versión 8.10.0 de Node.js, esta versión incluye el administrador de paquetes npm en su versión 5.6.0. Entonces, para implementar cada uno de los módulos de la librería NGSI de JavaScript se requiere la instalación previa de Node.js en su versión 8.10.0 o superior y npm en su versión 5.6.0 o superior.

Finalmente, para comprobar la correcta instalación de Node.js y npm abrimos una terminal de línea de comandos y ejecutamos los siguientes comandos: node -v y npm -v, ver Figura 10.

```
PS C:\WINDOWS\system32> node -v
v8.9.4
PS C:\WINDOWS\system32> npm -v
5.6.0
```

Figura 10 Comprobando instalación de Node.js y npm

4.3.3 Instalar los módulos en el proyecto

Una vez teniendo instalados los pre-requisitos que se mencionaron en el apartado anterior, primeramente, se debe generar un proyecto con node.js y npm, posteriormente abrir la terminal de línea de comandos y acceder al directorio donde se encuentra el proyecto para instalar cada uno de los módulos de la librería NGSI de JavaScript y ejecutar los siguientes comandos:

- Instalación del módulo OCB-sender
  npm install ocb-sender o yarn add ocb-sender
- Instalación del módulo NGSI-parser
  npm install ngsi-parser o yarn add ngsi-parsers

Finalmente, para comprobar si cada uno de los módulos de la librería NGSI de JavaScript se instalaron correctamente abrimos el archivo *package.json* del proyecto. Este archivo debe mostrar en el apartado de *dependencies* cada una de los módulos instalados, como se muestra la siguiente Figura 11.

```
{
  "name": "usage",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.3",
    "ngsi-parser": "^2.2.1",
    "node-fetch": "^2.1.2",
    "ocb-sender": "^2.1.1"
  },
  "devDependencies": {
    "nodemon": "^1.17.4"
  }
}
```

Figura 11 Archivo package.json dependencias instaladas

4.3.4 Importar cada uno de los módulos

En este apartado se mencionan las formas en que los módulos de la librería NGSI de JavaScript son implementados en un proyecto de node. Cada uno de los módulos de la librería pueden ser implementados en las versiones de ECMAScript 5 o 6 (ES5 o ES6). A continuación, se menciona cada una de las formas de implementación:

- Implementación del módulo OCB-sender:
  - Utilizando ES5
    var cb = require('ocb-sender');
  - Utilizando ES6
    import OCB as cb from ocb-sender;

  Una vez implementado el módulo OCB-sender mediante la variable cb se puede acceder a cada una de las funciones del módulo. Por ejemplo en la Tabla 4.34 se muestra la función *config(urlContextBroker, port, version)* que permite configurar la conexión del *Orion Context Broker* de FIWARE. Esta función recibe como parámetros la dirección del *Orion Context Broker* (***urlContextBroker***), el puerto de conexión (***port***) y la versión de la API a la que se está conectando (***version***).

Tabla 4.34 Configuración del módulo OCB-sender

```
cb.config(urlContextBroker, port, version)
 .then((result) => console.log(result))
 .catch((err) => console.log(err))
```

- Implementación del módulo NGSI-parser
    - Utilizando ES5
      var ngsi = require('ngsi-parser')
    - Utilizando ES6
      import NGSI as ngsi from 'ngsi-parser'

Una vez implementado el módulo NGSI-parser mediante la variable ngsi se puede acceder a cada una de las funciones del módulo.

Capítulo 5

Módulo genérico NGSI de Java

En este capítulo se presenta el módulo genérico NGSI (por sus siglas en inglés, *Next Generation Service Interface*) de Java desarrollado en esta tesis.

Este módulo genérico NGSI de Java tiene como objetivo convertir entidades o modelos de datos a entidades NGSI que cumpla con la especificación FIWARE-NGSI v2, para que puedan ser manipulables y operables por el *Orion Context Broker* de FIWARE. Este módulo se puede usar en el desarrollo de aplicaciones móviles para Android.

En la siguiente sección se describe la arquitectura del módulo genérico NGSI de Java.

5.1 Arquitectura del módulo genérico NGSI de Java

La arquitectura del módulo genérico NGSI de Java está compuesto por cinco elementos: Modelos de datos, utilerías, ngs-parser, funciones con entidades y cliente-http. La Figura 12 muestra la arquitectura del módulo genérico NGSI de Java. A continuación, se describen cada uno de sus elementos.

1. **Modelos de datos:** contiene entidades y modelos de datos que son convertidos a entidades NGSI que cumplan con la especificación FIWARE-NGSI v2. Además, incluye los tipos de datos que soporta el *Orion Context Broker*.
2. **Utilería:** contiene funciones que son generalmente utilizadas por aplicaciones móviles, por ejemplo: funciones para obtener las características del dispositivo móvil, funciones para obtener fechas del dispositivo móvil en diferentes formatos y funciones para gestionar las preferencias de aplicaciones móviles.
3. **ngsi-parser:** es el encargado de convertir una entidad o un modelo de datos tal cual se obtienen desde la aplicación móvil a una entidad NGSI que cumpla con la especificación FIWARE-NGSI v2.
4. **Funciones con entidades:** contiene las funciones de manipulación de entidades del *Orion Context Broker* de FIWARE.
5. **Cliente-http:** es el encargado de la conexión con el *Orion Context Broker* de FIWARE.

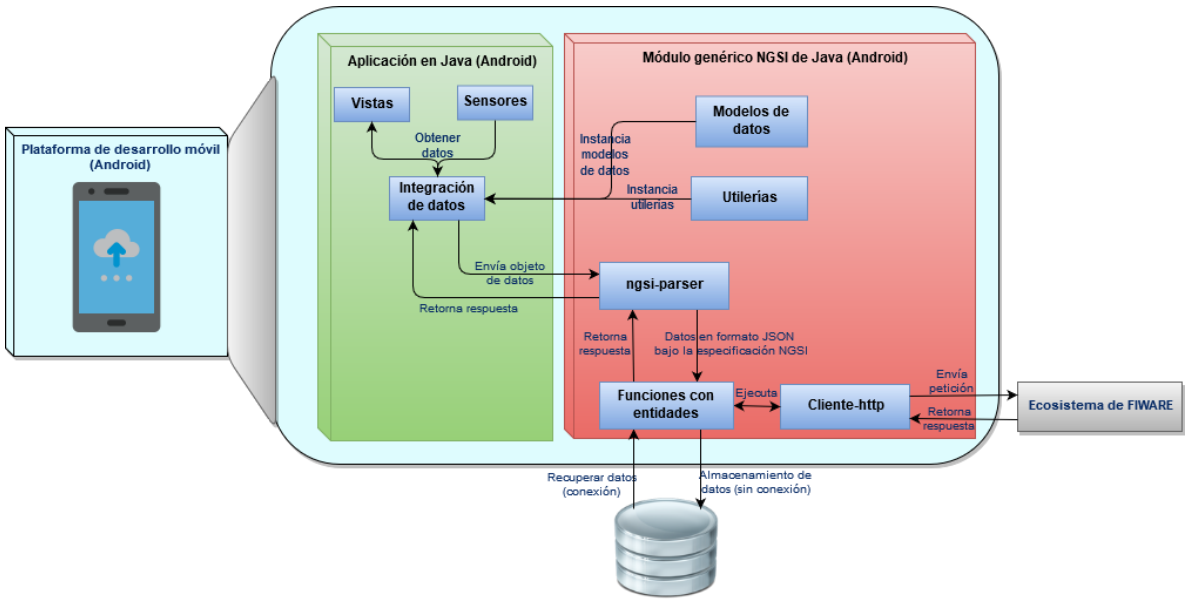

Figura 12 Arquitectura del módulo genérico NGSI de Java

A continuación, en la siguiente sección se describen los tipos de datos que soporta el módulo genérico NGSI de Java.

5.2 Tipos de datos

La verificación de una entidad NGSI se lleva a cabo en ciertos tipos de datos que soporta el módulo genérico NGSI de Java, estos tipos de datos que soporta son los siguientes:

- Text: se utiliza cuando el valor es una cadena (*string*)
- Number: se utiliza cuando el valor es un número.
- Boolean: se utiliza cuando el valor es un booleano (*true* o *false*).
- DateTime: se utiliza cuando el valor es una fecha.
- geo:point: se utiliza cuando la cadena contiene un par de latitud-longitud, separados por comas.
- geo:json: se utiliza cuando se representa una ubicación codificada.

Por otra parte, se generó una clase por cada tipo de dato que soporta el módulo genérico NGSI de Java. La Figura 13, muestra cada una de las clases por cada tipo de datos. Además, en el Anexo 2, se muestra un ejemplo de una clase de un determinado tipo de dato.

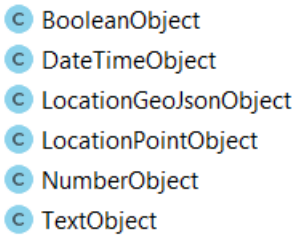

Figura 13 Clases por cada tipo de datos

Por otro lado, en el Anexo 3 se muestra un ejemplo para generar una entidad con los tipos de datos que soporta el módulo genérico NGSI de Java.

A continuación, en la siguiente sección se detalla a profundidad cada una de las funciones que soporta el módulo genérico NGSI de Java.

5.3 Funciones del módulo genérico NGSI de Java

El módulo genérico NGSI de Java tiene como objetivo gestionar (enviar, actualizar, etc.) la información de contexto de entidades NGSI en una instancia del *Orion Context Broker* de FIWARE. Las funciones que soporta el módulo genérico son las siguientes:

5.3.1 Función para crear una entidad

```
createEntity(Context context, String id, Object arg)
```

5.3.2 Función para actualizar una entidad

```
updateEntity(Context context, String id, Object arg)
```

5.3.3 Función para consultar entidades

```
getEntity(Map<String, String> params)
```

5.3.4 Función para eliminar una entidad

```
deleteEntityById(String id)
```

5.3.5 Función que recupera entidades almacenadas en el dispositivo móvil para crearlas en el Orion Contex Broker

```
createEntitySaveOffline(String id)
```

5.3.6 Función que recupera entidades almacenadas en el dispositivo móvil para actualizarlas en el Orion Context Broker

```
updateEntitySaveOffline(String id)
```

A continuación, se describen detalladamente cada una las funciones mencionadas anteriormente.

5.3.1 Función para crear una entidad

La función *createEntity(Context context, String id, Object arg)*, permite crear una entidad en el *Orion Context Broker* de FIWARE. Además, la función valida que el dispositivo móvil se encuentre conectado a internet para generar la entidad. Si el dispositivo móvil se encuentra conectado a internet y crea la entidad correctamente en el *Orion context Broker*, genera un registro en la base de datos

del dispositivo móvil con *status* igual a uno para indicar que la entidad se ha generado correctamente, en caso que el dispositivo móvil no se encuentre conectado a internet automáticamente la información se registra en la base de datos con *status* igual a cero para indicar que la entidad no se ha generado en el *Orion Context Broker*.

Por otra parte, la función recibe como primer parámetro el contexto (*context)* de la aplicación en el que se ejecuta la función, como segundo parámetro el identificador de la entidad (*id*) y como tercer parámetro el objeto de la entidad (*arg*) que es transformado a una entidad NGSI de Fiware.

5.3.2 Función para actualizar una entidad

La función *updateEntity(Context context, String id, Object arg)*, permite actualizar una entidad que se encuentra creada en el *Orion Context Broker* de Fiware. Además, la función valida que el dispositivo móvil se encuentre conectado a internet para actualizar la entidad. Si el dispositivo móvil no se encuentra conectado a internet automáticamente la información se almacena en la base de datos con *status* igual a cero para indicar que la entidad no se ha actualizado en el *Orion Context Broker*.

Por otra parte, la función recibe como primer parámetro el contexto (*context)* de la aplicación en el que se ejecuta la función, como segundo parámetro el identificador de la entidad (*id*) y como tercer parámetro el objeto de la entidad (*arg*) que es transformado a una entidad NGSI de Fiware.

5.3.3 Función para consultar entidades

La función *getEntity(Map<String, String> params*), permite consultar entidades que se encuentran en el *Orion Context Broker*. Esta función recibe como parámetro una lista (clave, valor) de tipo *String*, donde la clave es el nombre del atributo y el valor es el dato del atributo por el que se desea buscar. Entonces, si la función recibe como parámetro una lista vacía esta obtiene todas las entidades del *Orion context Broker*, en caso contrario la función obtiene las entidades del *Orion context Broker* que cumplan con el criterio de búsqueda.

5.3.4 Función para eliminar una entidad

La función *deleteEntityById(String id)*, permite eliminar una entidad que se encuentra en el *Orion Context Broker* de Fiware. Esta función recibe como parámetro el identificador de la entidad (*id*) a eliminar.

5.3.5 Función que recupera entidades almacenadas en el dispositivo móvil para crearlas en el Orion Contex Broker

La función *createEntitySaveOffline(String id)*, permite recuperar los registros de las entidades que se encuentran almacenadas en la base de datos del

dispositivo móvil que no han sido creadas en el *Orion Context Broker* de Fiware. Además, la función valida que el dispositivo móvil se encuentre conectado a internet para crear la entidad. Asimismo, la función recibe como parámetro el identificador de la entidad (*id*) que permite buscar en la base de datos del dispositivo móvil los registros que no se ha enviado.

5.3.6 Función que recupera entidades almacenadas en el dispositivo móvil para actualizarlas en el Orion Context Broker

La función *updateEntitySaveOffline(String id)*, permite recuperar los registros de las entidades que se encuentran almacenadas en la base de datos del dispositivo móvil que no se han actualizado en el *Orion Context Broker* de Fiware. Además, la función valida que el dispositivo móvil se encuentre conectado a internet para actualizar la entidad. Asimismo, la función recibe como parámetro el identificador de la entidad (*id*) que permite buscar en la base de datos del dispositivo móvil los registros que no se ha enviado.

A continuación, se describe la implementación del módulo genérico NGSI de Java en un proyecto de Android.

5.4 Implementación del módulo genérico NGSI de Java

El objetivo principal de esta sección es detallar la implementación del módulo genérico NGSI de Java en un proyecto de Android, para ello se debe tener en cuenta ciertos pre-requisitos y ciertas actividades.

5.4.1 Pre-requisitos

Para implementar el módulo genérico NGSI de Java es necesario tener en cuenta los siguientes requisitos: en primer lugar, se debe tener instalado *Android Studio*, ver el siguiente enlace para su descarga https://developer.android.com/studio/ y en segundo lugar se debe generar un proyecto en *Android Studio* en la versión 16 del SDK o superior.

5.4.2 Importación del módulo genérico NGSI de Java

En este apartado se describen cada uno de los pasos que permiten implementar de una manera correcta el módulo genérico NGSI de Java en un proyecto de Android.

En primer lugar, una vez generado el proyecto en *Android Studio* dirigirse al menú archivo (*file*), nuevo (*new*) e importar módulo (*Import Module*), ver Figura 14.

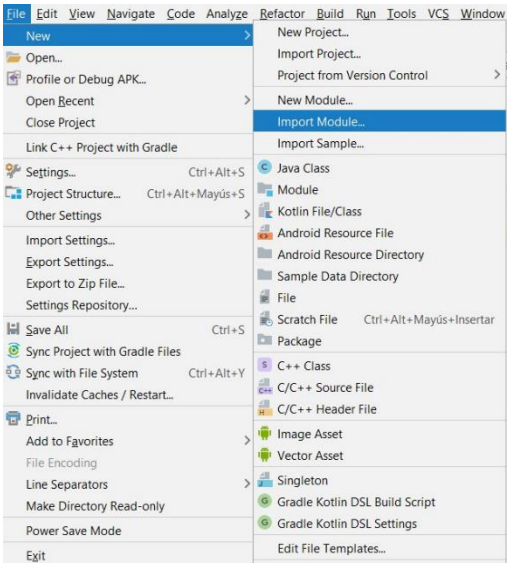

Figura 14 Importar módulo genérico NGSI de Java

En segundo lugar, se muestra una ventana para seleccionar la ruta donde se encuentra el módulo genérico NGSI de Java y finalmente dar clic en el botón finalizar (*Finish*), ver Figura 15.

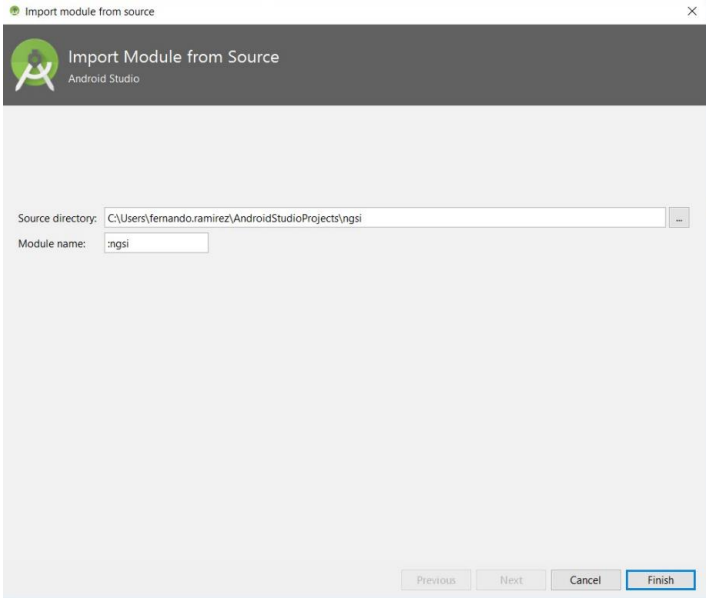

Figura 15 Seleccionar directorio del módulo genérico NGSI de Java

Por otro lado, una vez realizado los dos pasos anteriores el módulo se debe mostrar importado dentro del proyecto de Android, ver Figura 16.

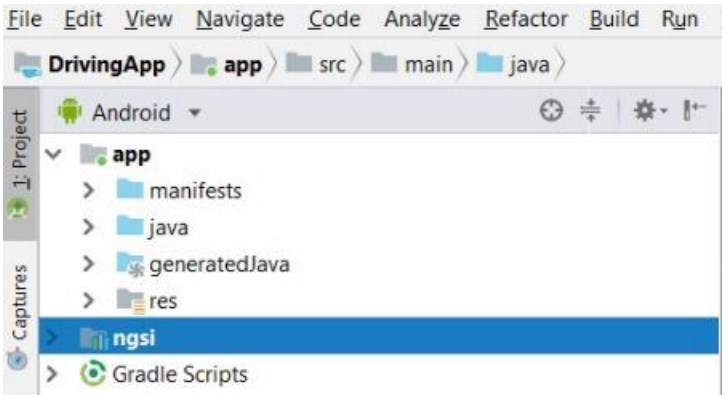

Figura 16 Comprobar la importación del módulo genérico NGSI de Java

5.4.3 Añadir la dependencia del módulo genérico NGSI de Java

En este apartado se describen cada uno de los pasos para añadir la dependencia del módulo genérico NGSI de Java en la aplicación de Android.

En primer lugar, ir al menú archivo (*file*), estructura del proyecto (*Project Structure*), ver Figura 17.

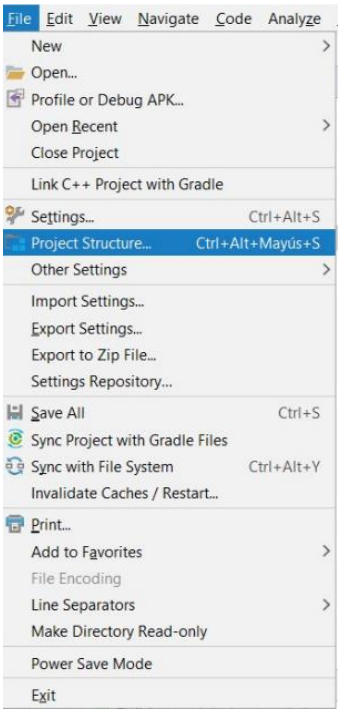

Figura 17 Añadir dependencia del módulo genérico NGSI de Java

En segundo lugar, se muestra una ventana para seleccionar la dependencia del módulo, dirigirse a la pestaña de dependencias (*Dependencies*), pulsar sobre el botón más y seleccionar la opción dependencia del módulo (*Module dependency*), ver Figura 18.

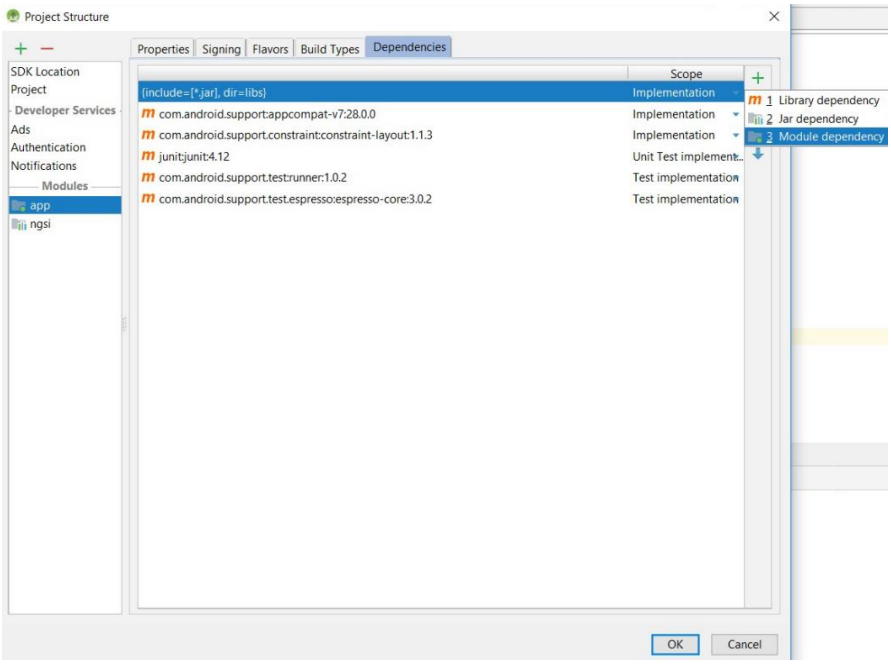

Figura 18 Estructura del proyecto para añadir dependencia

Posteriormente, se muestra una ventana donde se selecciona el módulo y se pulsa el botón *Ok*, ver Figura 19.

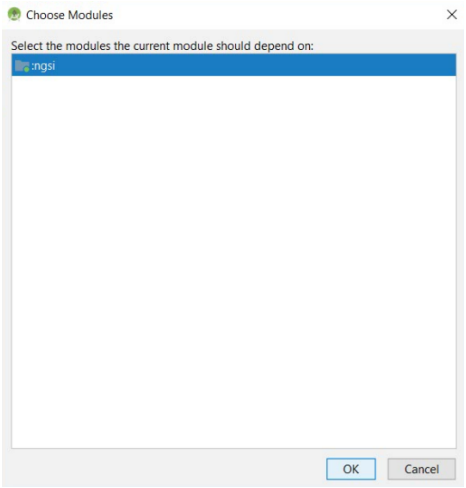

Figura 19 Buscar la dependencia del módulo

Finalmente, para comprobar si la dependencia del módulo genérico NGSI de Java se añadió correctamente, abrir el archivo (*build.gradle*) de la aplicación y verificar que el nombre del módulo se encuentre implementado, ver Figura 20.

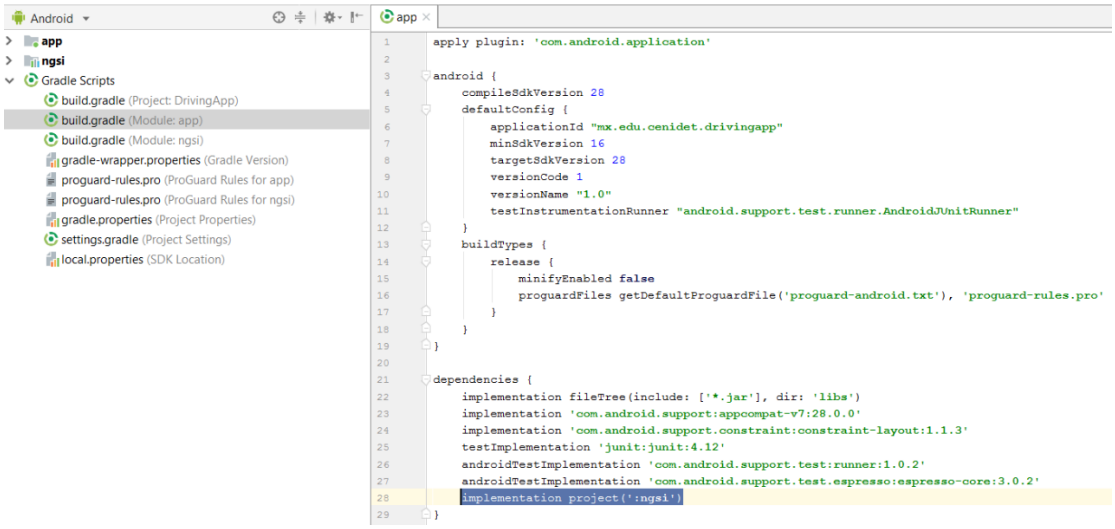

Figura 20 Verificar la dependencia del módulo genérico NGSI de Java

5.4.4 Configuración e inicialización de la conexión del módulo genérico NGSI de Java

En este apartado se describe la configuración e inicialización de la conexión del módulo genérico NGSI de Java en una aplicación de Android.

Respecto a, la configuración de conexión del *Orion Context Broker* de Fiware. Se deben editar los parámetros del archivo que se encuentra en la dirección "*ngsi/src/main/assets/config.properties*", estos parámetros son los siguientes:

1. Dirección del host del *Orion Context Broker*
   http.host=http://200.23.5.142
2. Puerto por el que escucha el *Orion Context Broker*
   http.port=1026
3. La versión de la API del *Orion Context Broker*
   http.apiversion=v2

En cuanto a, la inicialización de la conexión esta se debe realizar solo una vez en la activada de la aplicación donde se comiencen a utilizar las funciones del módulo genérico NGSI de Java. En la Figura 21, se muestra un ejemplo de la inicialización de los datos de conexión en una actividad. La función *initialize(file, context)*, recibe como parámetros el nombre del archivo (*config.properties*) donde se encuentran los parámetros de configuración y el contexto (*context)* de la aplicación en el que se ejecuta la función.

```java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Context context= getApplicationContext();

        //Inicio de la inicialización de los datos de conexión
        try {
            Tools.initialize( file: "config.properties", context);
        } catch (Exception e) {
            e.printStackTrace();
        }
        //Fin de la inicialización de los datos de conexión
    }
}
```

Figura 21 Inicialización de datos de configuración del módulo genérico NGSI de Java

Finalmente, en el Anexo 4 se encuentra un ejemplo de la implementación del módulo genérico para generar una entidad en el *Orion Context Broker* de Fiware.

Capítulo 6

Pruebas y resultados

El objetivo principal de este capítulo es evaluar la librería NGSI de JavaScript y el módulo genérico NGSI de Java.

El objetivo principal de este capítulo es evaluar la librería NGSI de JavaScript y el módulo genérico. Para lograr este objetivo, se desarrollaron dos aplicaciones: la primera web (*FIWARE Driving Monitor*) y la segunda móvil (*DrivingApp*) en las que se implementó la librería NGSI de JavaScript y el módulo genérico NGSI de Java para probar su funcionamiento. A continuación, la Figura 22 muestra la manera en que se implementó la librería NGSI de JavaScript y el módulo genérico NGSI de Java. En primer lugar, la librería NGSI de JavaScript se implementó en los servicios web desarrollados en node.js, estos servicios son utilizados por la aplicación web y la aplicación móvil. En segundo lugar, el módulo genérico se implementó en la aplicación móvil.

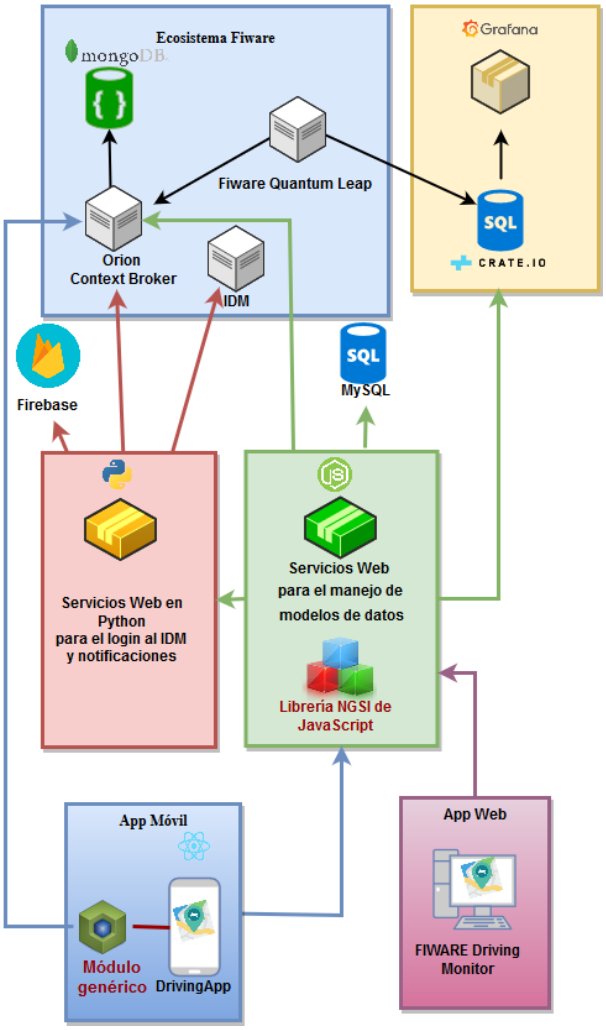

Figura 22 Implementación de la librería NGSI de JavaScript y el módulo genérico

Respecto a, *FIWARE Driving Monitor* es una aplicación web que tiene como objetivo delimitar zonas para monitorear eventos generados por los dispositivos móviles de los usuarios que se encuentran dentro de una determinada zona. Además, permite buscar a usuarios que se encuentran o se encontraban dentro

de una determinada zona. A continuación, se presentan las principales funcionalidades de *FIWARE Driving Monitor*:

- Delimitación de zonas
- Mostrar todos los usuarios que están actualmente en determinada zona
- Buscar un usuario que se encuentra actualmente en determinada zona
- Buscar un usuario dentro de una zona en determinada fecha
- Mostrar las últimas diez alertas actuales en determinada zona
- Mostrar el historial de las últimas diez alertas en determinada zona

Por otra parte, *DrivingApp* es una aplicación móvil que tiene como objetivo monitorear constantemente a los dispositivos móviles que se encuentran dentro de un área determinada. Además, la aplicación móvil envía alertas manuales y automáticas de accidentes o situaciones inusuales para alertar a otros usuarios que se encuentran dentro de un área determinada. A continuación, se presentan las principales funcionalidades de la aplicación móvil:

- Notificar alertas generadas dentro de una zona
- Mostrar el historial de la últimas diez alertas de una determinada zona
- Obtener los modelos de datos zonas
- Obtener los modelos de datos de estacionamientos
- Obtener los modelos de datos de segmentos de calle/carretera
- Envío de alertas
- Envío del modelo de datos DeviceModel
- Envío del modelo de datos Device

En la sección 6.1, 6.2 y 6.3 se describe la implementación de las funcionalidades de la librería NGSI de JavaScript y el módulo genérico NGSI de Java en cada una de las aplicaciones.

6.1 Implementación de la librería NGSI de JavaScript en la aplicación web

El objetivo principal de esta sección es detallar la implementación de las funciones de la librería NGSI de JavaScript en la aplicación web. Estas funcionalidades de la aplicación web utilizan los servicios web de node.js junto con la librería NGSI de JavaScript. A continuación, se describen las funcionalidades de la aplicación web.

6.1.1 Delimitación de zonas

Esta función permite registrar una zona, recibe como datos el nombre de la zona (*Zone name*), dirección de la zona (*Address´s zone*), descripción de la zona (*Zone description*) y la delimitación de la zona en el mapa mediante un polígono, ver Figura 23.

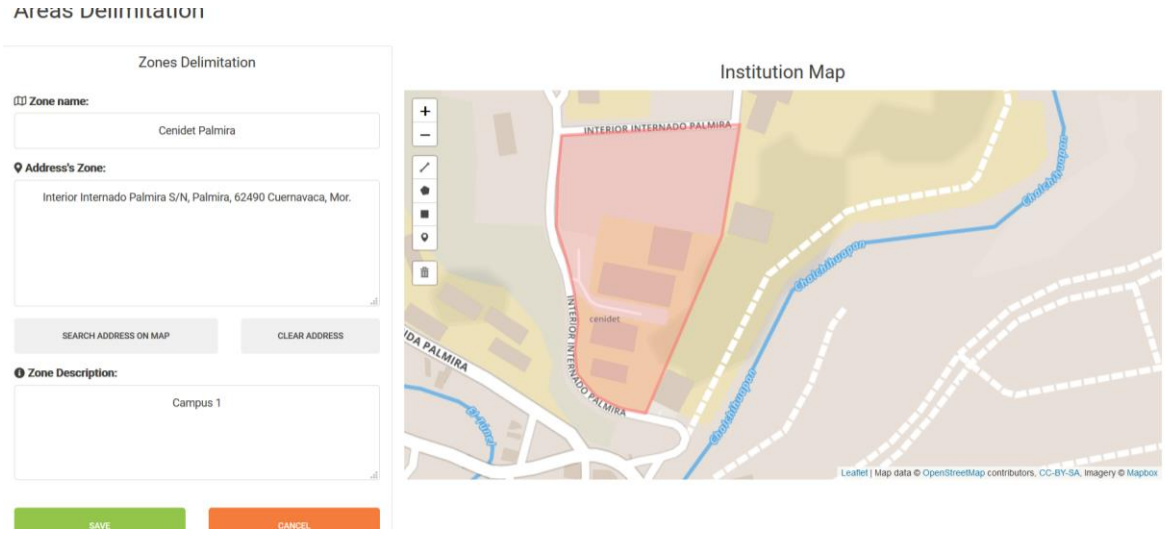

Figura 23 Delimitación de una zona

Por otra parte, al realizar el registro de la zona se genera un JSON en la especificación FIWARE-NGSI v2 del modelo de datos *Building*, para más detalle sobre este modelo de datos, ver el Anexo 5.

6.1.2 Mostrar todos los usuarios que están actualmente en determinada zona

Esta función permite mostrar a todos aquellos usuarios que se encuentran actualmente dentro de una zona determinada.

La Figura 24 muestra la funcionalidad de esta función, de entrada, se selecciona la opción de búsqueda (*Select a searching options*) y la zona (*Select the zone*) y finalmente se pulsa el botón consultar (*Consult*) que muestra en el mapa a todos aquellos usuarios que se encuentren dentro de la zona.

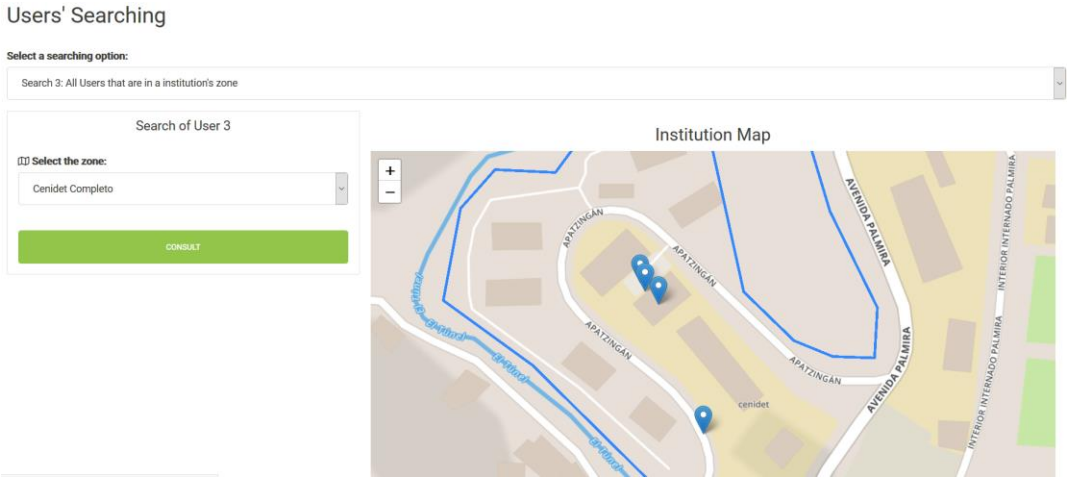

Figura 24 Mostrar todos los usuarios que están actualmente en determinada zona

6.1.3 Buscar un usuario que se encuentra actualmente en determinada zona

Esta función permite buscar a un usuario que se encuentra actualmente dentro de una zona determinada.

La Figura 25 muestra la funcionalidad de esta función, de entrada, se selecciona la opción de búsqueda (*Select a searching options*) y la zona (*Select the zone*), además, se escribe el número de teléfono del usuario a buscar (*User´s Phone Number*) y finalmente se pulsa el botón consultar (*Consult*) para mostrar en el mapa al usuario que se está buscando.

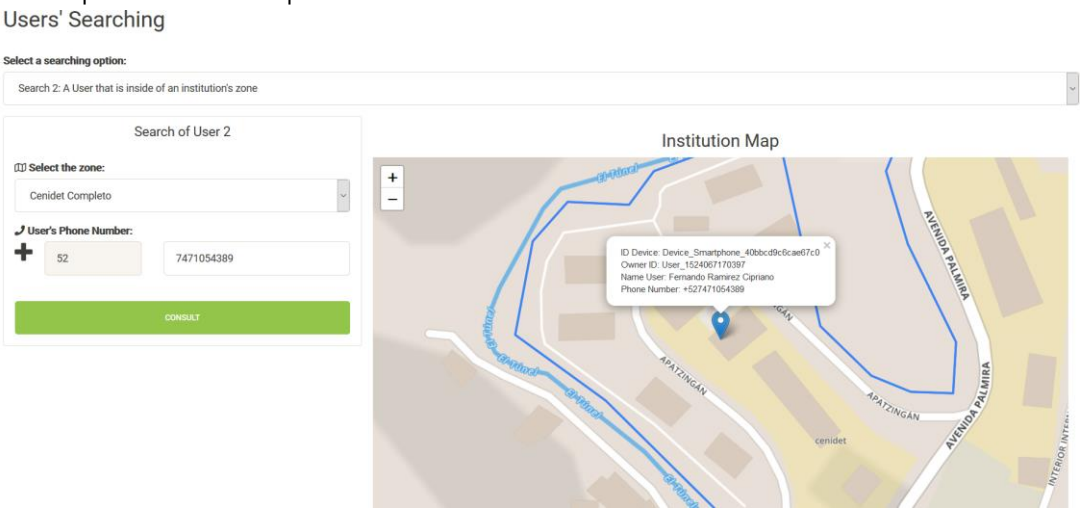

Figura 25 Buscar un usuario que se encuentra actualmente en determinada zona

6.1.4 Buscar un usuario dentro de una zona en determinada fecha

Esta función permite buscar a un usuario dentro de una zona en una fecha determinada.

La Figura 26 muestra la funcionalidad de esta función, en primer lugar, se selecciona la opción de búsqueda (*Select a searching options*) y la zona (*Select the zone*), en segundo lugar, se escribe el número de teléfono del usuario a buscar (*User´s Phone Number*), en tercer lugar, se selecciona la fecha (*Select the Date*) y la hora (*Select the hour (24 format)*) y finalmente se pulsa el botón consultar (*Consult*) para mostrar en el mapa al usuario que se está buscando.

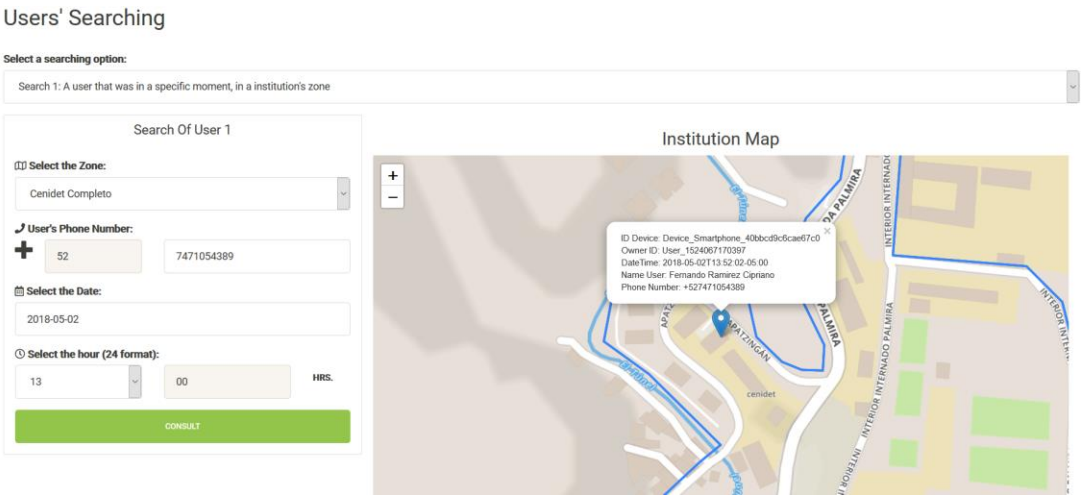

Figura 26 Buscar un usuario dentro de una zona en determinada fecha

6.1.5 Mostrar las últimas diez alertas actuales en determinada zona

Esta función permite mostrar las últimas diez alertas generadas durante el día dentro de una zona.

La Figura 27 muestra la funcionalidad de esta función, en primer lugar, se selecciona la zona (*Select the zone*) y la opción de visualización de las alertas (*Choose an alert visualization*) y finalmente se pulsa el botón consultar (*Consult*) que muestra en el mapa a todas las alertas generadas dentro de la zona.

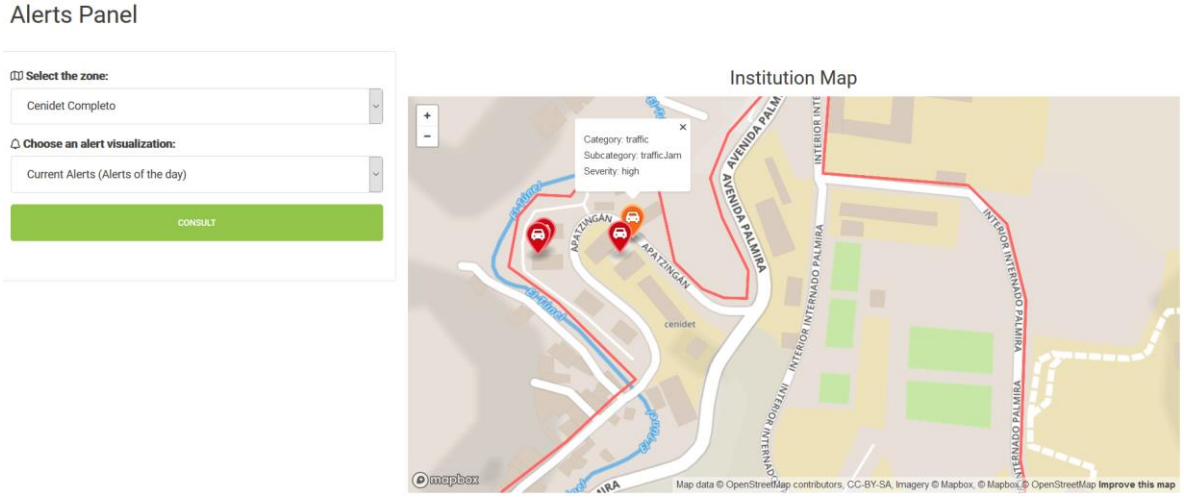

Figura 27 Mostrar las últimas diez alertas actuales de determinada zona

6.1.6 Mostrar el historial de las últimas diez alertas en determinada zona

Esta función permite obtener el historial de las últimas diez alertas generadas dentro de la zona.

La Figura 28 muestra la funcionalidad de esta función, en primer lugar, se selecciona la zona (*Select the zone*) y la opción de visualización de las alertas

(*Choose an alert visualization*) y finalmente se pulsa el botón consultar (*Consult*) que muestra en el mapa a todas las alertas generadas dentro de la zona.

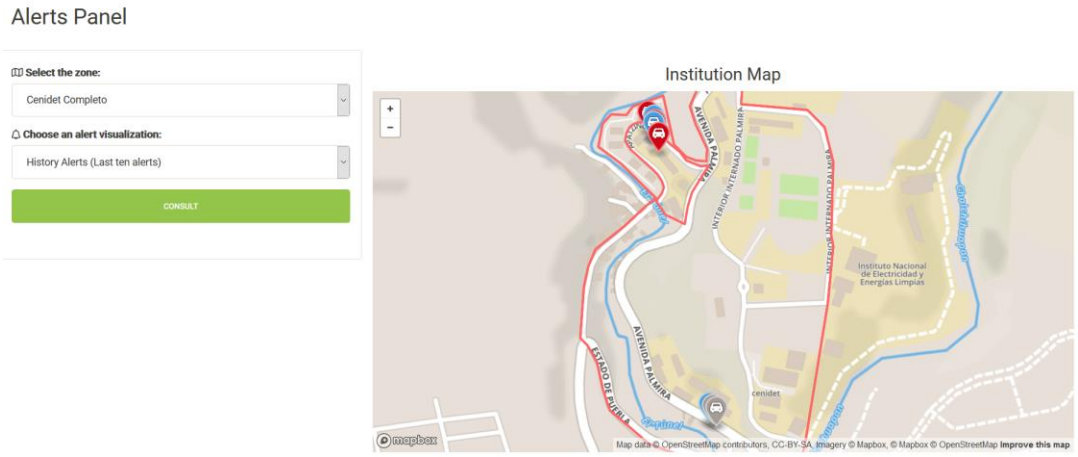

Figura 28 Mostrar el historial de las últimas diez alertas en determinada zona

6.2 Implementación de la librería NGSI de JavaScript en la aplicación móvil

El objetivo principal de esta sección es detallar la implementación de las funciones de la librería NGSI de JavaScript en la aplicación móvil. Estas funcionalidades de la aplicación móvil utilizan los servicios web de node.js junto con la librería NGSI de JavaScript. Estas funcionalidades se describen a continuación.

6.2.1 Notificar alertas generadas dentro de una zona

Esta función notifica las alertas generadas a los usuarios que se localicen dentro de la misma zona. Ver en la Figura 29.

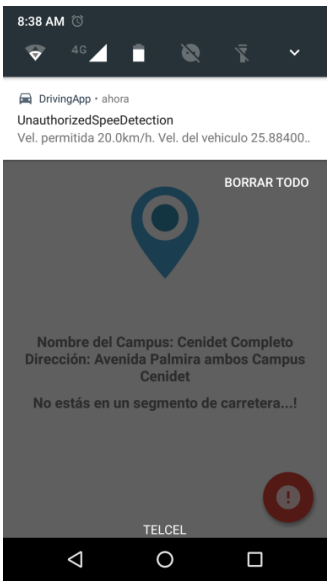

Figura 29 Notificar alertas generadas dentro de una zona

6.2.2 Mostrar el historial de la últimas diez alertas de una determinada zona

Esta función muestra el historial de las últimas diez alertas generadas dentro de una zona. Además, en el mapa se observa el lugar donde ocurrieron las alertas. Ver en la Figura 30.

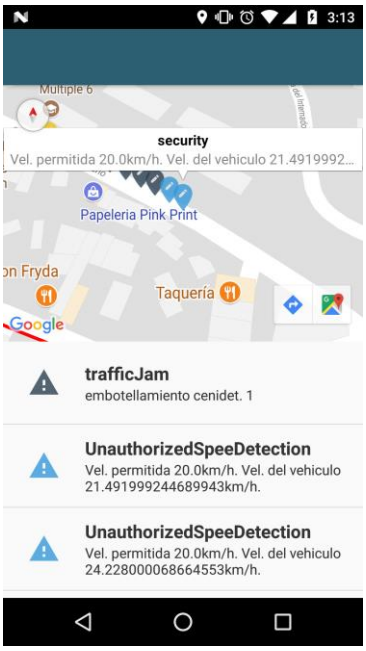

Figura 30 Mostrar el historial de la últimas diez alertas en una determinada zona

6.2.3 Obtener los modelos de datos zonas

Esta función obtiene los modelos de datos de tipo *Building* (zona). Además, en el Anexo 5 se muestra la estructura del JSON del modelo de datos *Building* en la especificación FIWARE-NGSI v2. Este modelo de datos es utilizado por la aplicación para determinar si un usuario se encuentra dentro o fuera de alguna zona. Ver la Figura 31 donde se muestra el nombre de la zona en la que se encuentra el usuario actualmente.

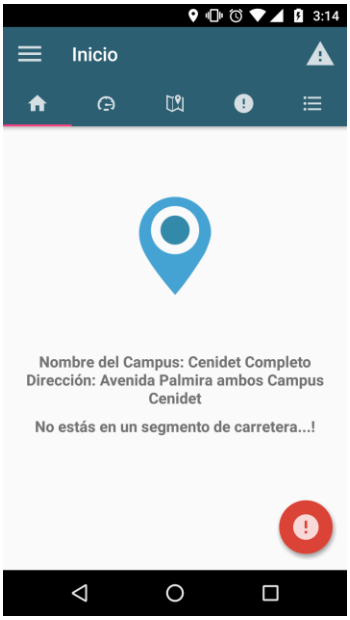

Figura 31 Determinar si se encuentra dentro o fuera de una zona

6.2.4 Obtener los modelos de datos de estacionamientos

Esta función obtiene las modelos de datos de tipo *OffStreetParking* (estacionamiento) que forman parte de una zona. Además, en el Anexo 6 se muestra la estructura del JSON del modelo de datos *OffStreetParking* en la especificación FIWARE-NGSI v2. Este modelo de datos es utilizado por la aplicación para determinar si un usuario se encuentra dentro o fuera de un estacionamiento. Ver la Figura 32 donde se muestra la zona delimitada con línea de color rojo y en su interior los estacionamientos delimitados con líneas de color azul.

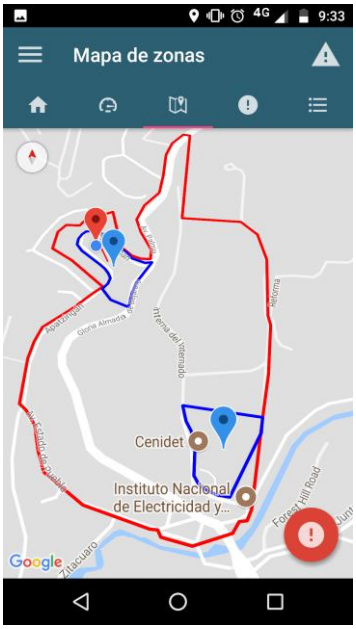

Figura 32 Obtener los modelos de datos de estacionamientos

6.2.5 Obtener los modelos de datos de segmentos de calle/carretera

Esta función obtiene los modelos de datos de tipo *RoadSegment* (segmentos de calle/carretera) que forman parte de una calle o de un estacionamiento. Además, en el Anexo 7 se muestra la estructura del JSON del modelo de datos *RoadSegment* en la especificación FIWARE-NGSI v2. Este modelo de datos es utilizado por la aplicación para determinar si el usuario se encuentra sobre un segmento de calle/carretera. Ver la Figura 33 donde se muestra la información del segmento de calle/carretera.

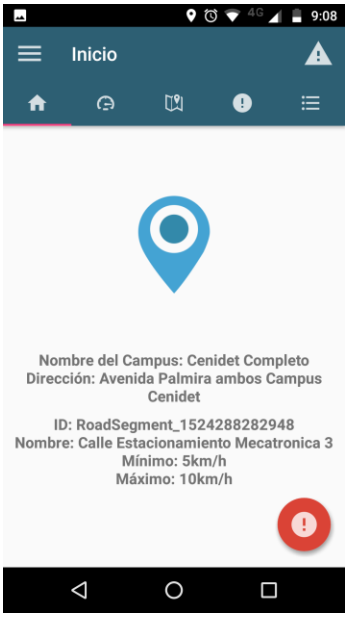

Figura 33 Muestra información del segmento de calle/carretera

6.3 **Implementación del módulo genérico en la aplicación móvil**

El objetivo principal de esta sección es detallar la implementación de las funciones del módulo genérico NGSI de Java en la aplicación móvil. A continuación, se describe detalladamente cada una de las funciones.

6.3.1 Envío de alertas

Esta subsección describe las dos formas en que la aplicación móvil envía las alertas, la primera forma es el envío de alertas manuales y la segunda forma es el envío de alertas automáticas de exceso de velocidad. Además, en el Anexo 8 se muestra la estructura del JSON del modelo de datos *Alert* en la especificación FIWARE-NGSI v2 que se utiliza para el envío de alertas. A continuación, se describe cada una de las formas de envío de alertas.

6.3.1.1 *Envío de alertas manuales*

Esta forma permite enviar tres tipos de alertas manuales: alertas de emergencia desconocida, alertas de accidente automovilístico y alertas de embotellamiento vehicular. Cada uno de estos tipos de alertas tiene un nivel de severidad (informacional, bajo, medio, alto o crítico). El primer tipo de alertas se envían con el nivel de severidad crítico y las dos últimas con alguno de los siguientes niveles de severidad informacional, bajo, medio, alto o crítico dependiendo el nivel que el usuario haya seleccionado.

Ver la Figura 34 donde se muestra el menú de las opciones de los tipos de alertas que se pueden enviar.

Figura 34 Menú de alertas manuales

6.3.1.2 *Envío de alertas automáticas de exceso de velocidad*

Esta forma permite enviar alertas automáticas de exceso de velocidad una vez la aplicación ha detectado que el usuario se encuentra sobre un segmento de calle/carretera y este ha excedido la velocidad permitida. Ver la Figura 35 donde se muestra la alerta generada por la aplicación al detectar este tipo de evento.

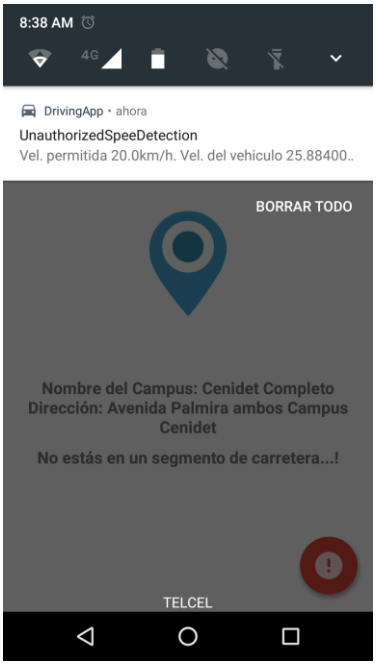

Figura 35 Envío de alertas automáticas de exceso de velocidad

6.3.2 Envío del modelo de datos DeviceModel

El modelo de datos DeviceModel obtiene todas las propiedades estáticas comunes de un dispositivo móvil. Por otra parte, esta función permite generar el modelo de datos DeviceModel en el Orion Context Broker de FIWARE.

La Tabla 6.1 muestra un ejemplo del modelo de datos DeviceModel bajo la especificación FIWARE-NGSI v2. Además, para este ejemplo el modelo representa al dispositivo móvil de la marca (*brandName*) motorola y del modelo (*modelName*) motoG5Plus.

Tabla 6.1 Modelo de datos DeviceModel

```
{
          "id": "DeviceModel_motorola_MotoG5Plus",
          "type": "DeviceModel",
          "brandName": {
                    "type": "Text",
                    "value": "motorola",
                    "metadata": {}
          },
          "category": {
                    "type": "Text",
                    "value": "smartphone",
                    "metadata": {}
          },
          "manufacturerName": {
                    "type": "Text",
                    "value": "motorola",
                    "metadata": {}
          },
          "modelName": {
                    "type": "Text",
                    "value": "MotoG5Plus",
                    "metadata": {}
          }
}
```

6.3.3 Envío del modelo de datos Device

El modelo de datos *Device* obtiene las propiedades que cambian constantemente en un dispositivo móvil. Por otra parte, esta función permite generar y actualizar el modelo de datos *Device* en el *Orion Context Broker* de FIWARE.

La Tabla 6.2 muestra un ejemplo del modelo de datos *Device* bajo la especificación FIWARE-NGSI v2. Este modelo con el atributo ubicación (*location*) permite saber el lugar donde se encuentra el dispositivo móvil y con el atributo propietario (*owner*) permite saber quién es el usuario del dispositivo móvil. Además, mediante el atributo ***refDeviceModel*** permite saber a qué instancia del modelo de datos *DeviceModel* pertenece el dispositivo móvil.

Tabla 6.2 Modelo de datos Device


```
{
          "id": "Device_Smartphone_40bbcd9c6cae67c0",
          "type": "Device",
          "accuracy": {
                    "type": "Number",
                    "value": 15.170000076,
                    "metadata": {}
          },
          "batteryLevel": {
                    "type": "Number",
                    "value": 56,
                    "metadata": {}
          },
          "category": {
                    "type": "Text",
                    "value": "smartphone",
                    "metadata": {}
          },
          "dateCreated": {
                    "type": "DateTime",
                    "value": "2018-04-12T02:06:10.00Z",
                    "metadata": {}
          },
          "location": {
                    "type": "geo:point",
                    "value": "18.87971229, -99.22168199",
                    "metadata": {}
          },
          "osVersion": {
                    "type": "Text",
                    "value": "7.0",
                    "metadata": {}
          },
          "owner": {
                    "type": "Text",
                    "value": "User_1524067170397",
                    "metadata": {}
          },
          "refDeviceModel": {
                    "type": "Text",
                    "value": "DeviceModel_motorola_MotoG5Plus",
                    "metadata": {}
          }
}
```

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones que se obtuvieron en esta tesis. Además, se describen los trabajos futuros que se pueden desarrollar a partir de este trabajo de investigación.

7.1 Conclusiones

Debido a los desafíos asociados a la creciente urbanización en las grandes ciudades, la Unión Europea está financiando una iniciativa denominada *FIWARE* [5] cuyo objetivo es facilitar el desarrollo de aplicaciones inteligentes mediante una infraestructura abierta basada en la nube que ofrece un catálogo de componentes genéricos. Cada componente genérico ofrece una serie de funciones de propósito general a través de interfaces de programación de aplicaciones de licencias públicas y libres. En este contexto, en el trabajo de investigación se desarrollaron componentes genéricos que permiten la conexión de aplicaciones de dispositivos móviles a una plataforma IoT que implementen la especificación FIWARE-NGSI v2 y que tenga como propósito obtener información de los sensores del dispositivo móvil y envié la información en tiempo real al *Orion Context Broker* de *FIWARE* bajo la especificación FIWARE-NGSI v2 [13].

Por tal motivo, en este trabajo de investigación se planteó como objetivo principal desarrollar componentes genéricos para la conexión de dispositivos móviles a una plataforma IoT utilizando la especificación NGSI de *FIWARE*.

Para lograr el objetivo principal, se identificaron los siguientes objetivos específicos:

- *Diseñar y desarrollar los componentes genéricos*

  Para lograr este objetivo se identificaron los requerimientos de los componentes para definir el tipo de componente a desarrollar. El primer compone, es una la librería NGSI de JavaScript y el segundo componente, es un módulo genérico NGSI de Java para cada uno de estos componentes se diseñó su arquitectura. Además, se desarrolló cada componente genérico siguiendo el diseño de su respectiva arquitectura.

- *Desarrollar una aplicación web y móvil que implementen los componentes genéricos*

  Se desarrollaron dos aplicaciones una web y una móvil con la finalidad de implementar cada uno de los componentes genéricos (librería NGSI de JavaScript y módulo genérico NGSI de Java) desarrollados.

- *Probar el funcionamiento de los componentes genéricos en la aplicación web y móvil*

  Las funcionalidades de los componentes genéricos (librería NGSI de JavaScript y módulo genérico NGSI de Java) fueron probadas tanto en la aplicación web como en la móvil.

7.2 Trabajos futuros

Para ampliar la investigación, se presentan algunos posibles trabajos futuros:

- Desarrollar un módulo genérico NGSI para dispositivos móviles con sistema operativo iOS.
- Agregar nuevos tipos de datos que soporte el *Orion Context Broker* de FIWARE tanto a librería NGSI de JavaScript y al módulo genérico NGSI de Java.
- Implementar los componentes genéricos en el desarrollo de otras aplicaciones.

7.3 Logros obtenidos

A continuación, se enlistan los logros obtenidos a partir del desarrollo de este trabajo de investigación:

- Martinez, A., Ramirez, F., Estrada, H., and Torres, L. A.: Generic Module for collecting data in Smart cities, International Society for Photogrammetry and Remote Sensing. Spatial Inf. Sci., XLII-4/W3, 65-72, 2nd International Conference on Smart Data and Smart Cities October, Puebla, México https://doi.org/10.5194/isprs-archives-XLII-4-W3-65-2017 , 2017.
- Presentación oral del artículo "*Generic Module For Collecting Data In Smart Cities*" en el congreso *in the 2ND INTERNATIONAL CONFERENCE ON SMART DATA AND SMART CITIES*.
- Publicación de la aplicación móvil en la *Google Play* que lleva por nombre *DrivingApp y se encuentra en la siguiente dirección https://play.google.com/store/apps/details?id=mx.edu.cenidet.app*.
- Alicia Martinez, Hugo Estrada, Fernando Ramirez, Miguel González. A New Software Library for Mobile Sensing Using FIWARE Technologies. Journal Advances in Soft Computing 17th Mexican International Conference on Artificial Intelligence, MICAI 2018, Guadalajara, Mexico, October 22-27, 2018, Proceedings, Part I © Springer Nature Switzerland AG 2018 Batyrshin et al. (eds.): MICAI 2018, Part I, LNAI 11288, 2018. URL: https://www.springer.com/la/book/9783030044909 pp. 1-10.

Referencias

[1] L. A. (Eds. . D. Giusto, A. Iera, G. Morabito, *The Internet of Things*. Springer, 2010.

[2] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything," *CISCO white Pap.*, no. April, pp. 1–11, 2011.

[3] Amazon, "AWS | Almacenamiento de datos seguro en la nube (S3)." [Online]. Available: https://aws.amazon.com/es/s3/. [Accessed: 21-Aug-2018].

[4] F. Telefónica, "Smart Cities: un primer paso hacia la internet de las cosas," *Editor. Ariel*, pp. 13–16, 2011.

[5] FIWARE-ABOUT US, "ABOUT US » FIWARE," 2016. [Online]. Available: https://www.fiware.org/about-us/. [Accessed: 03-Apr-2017].

[6] RESEARCH NESTER, "Internet Of Things Iot Market Global Demand Growth Analysis Opportunity Outlook 2023." [Online]. Available: https://www.researchnester.com/reports/internet-of-things-iot-market-global-demand-growth-analysis-opportunity-outlook-2023/216. [Accessed: 25-Aug-2018].

[7] F. I. PPP, "Led by industry, driven by users Addressing the challenge of Internet development in Europe," 2015. [Online]. Available: https://www.fi-ppp.eu/. [Accessed: 07-Apr-2017].

[8] Google Cloud, "Cloud Computing, servicios de alojamiento y APIs de Google Cloud | Google Cloud." [Online]. Available: https://cloud.google.com/. [Accessed: 05-Jun-2018].

[9] Amazon Web Services, "Plataforma AWS IoT – Amazon Web Services," 2017. [Online]. Available: https://aws.amazon.com/es/iot-platform/. [Accessed: 13-Jun-2017].

[10] IBM Cloud, "About Watson IoT Platform," *Last Updated*, 2018. [Online]. Available: https://console.bluemix.net/docs/services/IoT/iotplatform_overview.html#about_iotplatform. [Accessed: 05-Jun-2018].

[11] aws, "¿Qué son los big data? – Amazon Web Services (AWS)," 2018. [Online]. Available: https://aws.amazon.com/es/big-data/what-is-big-data/. [Accessed: 24-Aug-2018].

[12] aws, "AWS | Informática en la nube. Ventajas y Beneficios," 2018. [Online]. Available: https://aws.amazon.com/es/what-is-cloud-computing/. [Accessed: 24-Aug-2018].

[13] FIWARE-NGSI v2-Specification, "FIWARE-NGSI v2-Specification," 2017. [Online]. Available: http://fiware.github.io/specifications/ngsiv2/stable/. [Accessed: 11-Jun-2017].

[14] S. Malhotra, M. N. Doja, B. Alam, and M. Alam, "Bigdata analysis and comparison of bigdata analytic approches," *Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017*, vol. 2017–Janua, pp. 309–314, 2017.

[15] T. H. A. S. Siriweera, I. Paik, B. T. G. S. Kumara, and K. R. C. Koswatta, "Intelligent Big Data Analysis Architecture Based on Automatic Service Composition," *2015 IEEE Int. Congr. Big Data*, pp. 276–280, 2015.

[16] AWS, "Smart Cities | AWS para ciudades inteligentes, conectadas y sostenibles." [Online]. Available: https://aws.amazon.com/es/smart-cities/. [Accessed: 26-Aug-2018].

[17] Miovision, "Miovision TrafficLink - Smart cities start with smart signals." [Online]. Available: http://miovision.com/trafficlink/. [Accessed: 26-Aug-2018].
[18] moovit, "Moovit: Tu Guía de Transporte Público." [Online]. Available: https://moovit.com/. [Accessed: 26-Aug-2018].
[19] PHILIPS, "CityTouch - Smart street lighting | Philips Lighting." [Online]. Available: http://www.lighting.philips.com/main/systems/lighting-systems/citytouch. [Accessed: 26-Aug-2018].
[20] INFOTEC, "Green Route, la aplicación para una movilidad inteligente." [Online]. Available: https://www.infotec.mx/es_mx/infotec/green_route_la_aplicacion_para_una_movilidad_inteligente. [Accessed: 26-Aug-2018].
[21] Conacyt, "4 ° Convocatoria Conacyt- Horizon2020," vol. 2020, pp. 1–5, 2016.
[22] Conacyt, "Convocatoria CONACYT-ESRC 2018 Ciudades Inteligentes Este," 2018.
[23] Comisión Europea, *Horizon 2020 en breve*. Luxemburgo: Unión Europea, 2014.
[24] Telefonica, "FIWARE, el estándar que necesita el IoT," 2017. [Online]. Available: https://iot.telefonica.com/blog/2016/09/es-fiware-estandar-iot. [Accessed: 01-May-2017].
[25] FIWARE-SmartSDK, "SmartSDK," 2017. [Online]. Available: https://www.smartsdk.eu/home/. [Accessed: 03-Apr-2017].
[26] Amazon Web Services, "What is the Internet of Things (IoT)," 2017. [Online]. Available: https://aws.amazon.com/es/iot/what-is-the-internet-of-things/. [Accessed: 09-May-2017].
[27] C. Macgillivray, "The Platform of Platforms in the Internet of Things," *Idc*, no. February, 2016.
[28] P. F. Ovidiu Vermesan, *Ecosystems, Internet of Things-Converging Technologies for Smart Environments and Integrated*. River Publishess, 2013.
[29] L. and A. V. Feng Xia, LaurenceT.Yang, "Internet of Things," *Int. J. Commun. Syst.*, vol. 23, no. 5, pp. 633–652, 2010.
[30] Google Cloud, "Google Cloud Platform Documentation | Documentation | Google Cloud." [Online]. Available: https://cloud.google.com/docs/?hl=es. [Accessed: 05-Jun-2018].
[31] Google Cloud, "Overview | Overview | Google Cloud." [Online]. Available: https://cloud.google.com/docs/overview/?hl=ko. [Accessed: 05-Jun-2018].
[32] FIWARE, "Home | FIWARE Catalogue." [Online]. Available: https://catalogue.fiware.org/. [Accessed: 11-May-2018].
[33] eSMARTCITY, "Fiware: Una plataforma abierta y estándar para Ciudades Inteligentes." [Online]. Available: https://www.esmartcity.es/comunicaciones/i-congreso-ciudades-inteligentes-fiware. [Accessed: 27-Aug-2018].
[34] FIWARE, "Fiware-Orion." [Online]. Available: https://fiware-orion.readthedocs.io/en/master/index.html. [Accessed: 06-Jun-2018].
[35] FIWARE-IOT-Stack, "Context Broker (Orion)," 2017. [Online]. Available: http://fiware-iot-stack.readthedocs.io/en/latest/context_broker/. [Accessed: 09-May-2017].
[36] T. I. and F. F. Telecom Italia, "FIWARE.OpenSpecification.Data.ContextBroker - FIWARE Forge Wiki." [Online]. Available:

https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.ContextBroker. [Accessed: 28-Oct-2018].
[37] Fiware, "Identity Management - KeyRock | FIWARE Catalogue." [Online]. Available: https://catalogue-server.fiware.org/enablers/identity-management-keyrock. [Accessed: 06-Jun-2018].
[38] Fiware, "Home - QuantumLeap." [Online]. Available: https://smartsdk.github.io/ngsi-timeseries-api/. [Accessed: 06-Jun-2018].
[39] Fiware, "API Walkthrough (v2) - Fiware-Orion." [Online]. Available: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html#subscriptions. [Accessed: 06-Jun-2018].
[40] A. W. Brown and K. C. Wallnau, "An Examination of the Current State of CBSE," *IEEE Softw.*, no. October 1998, pp. 37–46, 1998.
[41] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd Editio. Addison-Wesley Professional.
[42] P. Herzum and O. Sims, *Business Component Factory : A Comprehensive Overview of Component-Based Development for the Enterprise*, 1st Editio. 2000.
[43] Wikipedia, "Library (computing)," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Library_(computing). [Accessed: 09-May-2018].
[44] Mozilla, "JavaScript | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed: 07-May-2018].
[45] Mozilla, "Introduction - JavaScript | MDN," 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. [Accessed: 09-May-2018].
[46] Mozilla, "About JavaScript - JavaScript | MDN," 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed: 07-May-2018].
[47] node js, "Node.js." [Online]. Available: https://nodejs.org/en/. [Accessed: 09-May-2018].
[48] npm, "npm." [Online]. Available: https://www.npmjs.com/. [Accessed: 11-May-2018].
[49] npm, "01 - What is npm? | npm Documentation." [Online]. Available: https://docs.npmjs.com/getting-started/what-is-npm. [Accessed: 11-May-2018].
[50] aws, "What is Big Data? – Amazon Web Services (AWS)." [Online]. Available: https://aws.amazon.com/big-data/what-is-big-data/?nc1=h_ls. [Accessed: 27-Aug-2018].
[51] R. AU, "To Cloud or Not to Cloud," *Presentation*, no. August, pp. 1–15, 2011.
[52] IBM, "IBM - modelos de servicio en nube IaaS PaaS SaaS - México." [Online]. Available: https://www.ibm.com/cloud-computing/mx-es/learn-more/iaas-paas-saas/. [Accessed: 27-Aug-2018].
[53] A. A. Frozza, R. dos S. Mello, and F. de S. da Costa, "An Approach for Schema Extraction of JSON and Extended JSON Document Collections," *2018 IEEE Int. Conf. Inf. Reuse Integr.*, pp. 356–363, 2018.
[54] M. Klettke, U. Störl, and S. Scherzinger, "Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores," *Btw*, vol. 241, pp. 425–

444, 2015.
[55] S. Guo, H. Xia, and G. Xiang, "Research on the translation from XSD to JSON schema," *2017 9th IEEE Int. Conf. Commun. Softw. Networks, ICCSN 2017*, vol. 2017–Janua, pp. 1393–1396, 2017.
[56] Y. Li, N. R. Katsipoulakis, B. Chandramouli, J. Goldstein, and D. Kossmann, "Mison: A Fast JSON Parser for Data Analytics," *Very Large DataBases*, vol. 10, no. 10, pp. 1118--1129, 2017.
[57] E. Anjos, J. Lee, and S. R. Satti, "SJSON: A succinct representation for JavaScript object notation documents," *2016 11th Int. Conf. Digit. Inf. Manag. ICDIM 2016*, pp. 173–178, 2016.
[58] C. Benac Earle, L.-Å. Fredlund, Á. Herranz, and J. Mariño, "Jsongen: A QuickCheck based library for testing JSON web services," *Erlang 2014 - Proc. 2014 ACM SIGPLAN Erlang Work.*, pp. 33–41, 2014.
[59] L. Pulgatti and M. Didonet Del Fabro, "JSON-based Interoperability Applying the Pull-parser Programming Model," *Proc. 20th Int. Conf. Enterp. Inf. Syst.*, no. March, pp. 95–102, 2018.
[60] D. Colazzo, G. Ghelli, and C. Sartiani, "Typing Massive JSON Datasets," *Int. Work. Cross-model Lang. Des. Implement.*, vol. Vol. 541, no. July, p. 12, 2012.
[61] A. Smelter and H. N. B. Moseley, "A Python library for FAIRer access and deposition to the Metabolomics Workbench Data Repository," *Metabolomics*, vol. 14, no. 5, p. 0, 2018.

Anexos

Anexo 1. El esquema de alerta importado del repositorio de modelos de datos de FIWARE

```
{
  "$schema": "http://json-schema.org/schema#",
  "id": "https://fiware.github.io/dataModels/Alert/schema.json",
  "title": "Alert data model JSON Schema",
  "description": "An alert generated by a user or device in a givel location",
  "type": "object",
  "allOf": [{
           "$ref":                 "https://fiware.github.io/dataModels/common-
schema.json#/definitions/GSMA-Commons"
      },
      {

           "$ref":                 "https://fiware.github.io/dataModels/common-
schema.json#/definitions/Location-Commons"

      },
      {
           "properties": {
                "description": {
                     "type": "string"
                },
                "dateObserved": {
                     "type": "string",
                     "format": "date-time"
                },
                "validFrom": {
                     "type": "string",
                     "format": "date-time"
                },
                "validTo": {
                     "type": "string",
                     "format": "date-time"
                },
                "severity": {
                     "type": "string",
                     "enum": [
                        "informational",
                        "low",
                        "medium",
                        "high",
                        "critical"
                     ]
                },
                "category": {
                     "type": "string",
                     "enum": [
                        "traffic",
                        "weather",
                        "environment",
                        "health",
                        "security"
                     ]
                },
                "subCategory": {
                     "type": "string",
                     "enum": [
                        "trafficJam",
```

```
                        "carAccident",
                        "carWrongDirection",
                        "carStopped",
                        "pothole",
                        "roadClosed",
                        "roadWorks",
                        "hazardOnRoad",
                        "injuredBiker",
                        "rainfall",
                        "highTemperature",
                        "lowTemperature",
                        "heatWave",
                        "ice",
                        "snow",
                        "wind",
                        "fog",
                        "flood",
                        "tsunami",
                        "tornado",
                        "tropicalCyclone",
                        "hurricane",
                        "asthmaAttack",
                        "bumpedPatient",
                        "fallenPatient",
                        "heartAttack",
                        "suspiciousAction",
                        "robbery",
                        "assault"
                    ]
                },
                "alertSource": {
                    "oneOf": [{
                            "type": "string",
                            "format": "uri"
                        },
                        {
                            "$ref":
"https://fiware.github.io/dataModels/common-
schema.json#/definitions/EntityIdentifierType"
                        }
                    ]
                },
                "data": {
                    "type": "object"
                },

                "type": {
                    "type": "string",
                    "enum": [
                        "Alert"
                    ],
                    "description": "NGSI Entity type"
                }
            }
        }
    ],
    "oneOf": [{
            "required": [
                "id",
                "type",
                "location",
                "alertSource",
                "category",
```

```
                    "dateObserved"
               ]
          },
          {
               "required": [
                    "id",
                    "type",
                    "address",
                    "alertSource",
                    "category",
                    "dateObserved"
               ]
          }
     ]
}
```

Anexo 2 Clase del tipo de dato Text

```
public class TextObject {
   private String type = "Text";
   private String value;
   private Object metadata;

    public TextObject(){
        metadata = new Object();
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public Object getMetadata() {
        return metadata;
    }

    public void setMetadata(Object metadata) {
        this.metadata = metadata;
    }

}
```

Anexo 3. Generar la entidad Alert con los tipos de datos que soporta el módulo genérico NGSI de Java

```
public class Alert {
    private String id;
    private String type = "Alert";
    private TextObject alertSource;
    private TextObject category;
    private DateTimeObject dateObserved;
    private TextObject description;
    private LocationPointObject location;
    private TextObject severity;
    private TextObject subCategory;
    private DateTimeObject validFrom;
    private DateTimeObject validTo;

    public Alert() {
        alertSource = new TextObject();
        category = new TextObject();
        dateObserved = new DateTimeObject();
        description = new TextObject();
        location = new LocationPointObject();
        severity = new TextObject();
        subCategory = new TextObject();
        validFrom = new DateTimeObject();
        validTo = new DateTimeObject();
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public TextObject getAlertSource() {
        return alertSource;
    }

    public void setAlertSource(TextObject alertSource) {
        this.alertSource = alertSource;
    }

    public TextObject getCategory() {
        return category;
    }

    public void setCategory(TextObject category) {
        this.category = category;
    }

    public DateTimeObject getDateObserved() {
        return dateObserved;
```

```
    }

    public void setDateObserved(DateTimeObject dateObserved) {
        this.dateObserved = dateObserved;
    }

    public TextObject getDescription() {
        return description;
    }

    public void setDescription(TextObject description) {
        this.description = description;
    }

    public LocationPointObject getLocation() {
        return location;
    }

    public void setLocation(LocationPointObject location) {
        this.location = location;
    }

    public TextObject getSeverity() {
        return severity;
    }

    public void setSeverity(TextObject severity) {
        this.severity = severity;
    }

    public TextObject getSubCategory() {
        return subCategory;
    }

    public void setSubCategory(TextObject subCategory) {
        this.subCategory = subCategory;
    }

    public DateTimeObject getValidFrom() {
        return validFrom;
    }

    public void setValidFrom(DateTimeObject validFrom) {
        this.validFrom = validFrom;
    }

    public DateTimeObject getValidTo() {
        return validTo;
    }

    public void setValidTo(DateTimeObject validTo) {
        this.validTo = validTo;
    }
}
```

Anexo 4 Implementación del módulo genérico para generar una entidad en el Orion Context Broker de Fiware

Añadir al archivo AndroidManifest.xml de la aplicación, los siguientes permisos

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

Archivo activity_main.xml

```
<Button
    android:id="@+id/btnCreateEntity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="106dp"
    android:text="CREAR ENTIDAD" />
```

Archivo MainActivity.java

```
import android.content.Context;
import android.content.DialogInterface;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import www.fiware.org.ngsi.controller.DeviceController;
import www.fiware.org.ngsi.datamodel.entity.Alert;
import www.fiware.org.ngsi.httpmethodstransaction.Response;
import www.fiware.org.ngsi.utilities.DevicePropertiesFunctions;
import www.fiware.org.ngsi.utilities.Functions;
import www.fiware.org.ngsi.utilities.Tools;

public class MainActivity extends AppCompatActivity implements
DeviceController.DeviceResourceMethods {
    private DeviceController deviceController;
    Button btnCreateEntity;
    Context MAIN_CONTEXT = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MAIN_CONTEXT = this;
        Context context= getApplicationContext();
        deviceController = new DeviceController(this);

        //Inicio de la inicialización de los datos de conexión
        try {
            Tools.initialize("config.properties", context);
        } catch (Exception e) {
            e.printStackTrace();
        }
        //Fin de la inicialización de los datos de conexión
        btnCreateEntity = findViewById(R.id.btnCreateEntity);
        btnCreateEntity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                confirmAlert();
            }
        });
    }

    @Override
```

```
    public void onCreateEntity(Response response) {
        if(response.getHttpCode() == 201 || response.getHttpCode() == 200){
            Toast.makeText(MAIN_CONTEXT, "Entidad creada correctamente",
Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText(MAIN_CONTEXT, "Error al crear la entidad",
Toast.LENGTH_SHORT).show();
        }
    }
    @Override
    public void onCreateEntitySaveOffline(Response response) {

    }
    @Override
    public void onUpdateEntity(Response response) {
    }
    @Override
    public void onUpdateEntitySaveOffline(Response response) {

    }
    @Override
    public void onDeleteEntity(Response response) {
    }
    @Override
    public void onGetEntities(Response response) {
    }

    private void confirmAlert(){
        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(MAIN_CONTEXT);
        alertDialogBuilder.setTitle("Crear Entidad")
                .setMessage("Confirmar para crear una entidad")
                .setIcon(R.drawable.ic_launcher_background)
                .setNegativeButton("No",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                //acciones del boton No
                            }
                        })
                .setPositiveButton("Si",
                        new DialogInterface.OnClickListener(){
                            public void onClick(DialogInterface dialog, int id){
                                    Alert alert = new Alert();
                                    alert.setId(new
DevicePropertiesFunctions().getAlertId(MAIN_CONTEXT));
                                    alert.getAlertSource().setValue(new
DevicePropertiesFunctions().getDeviceId(MAIN_CONTEXT));
                                    alert.getCategory().setValue("unknownAlert");

alert.getDateObserved().setValue(Functions.getActualDate());
                                    alert.getDescription().setValue("Unknown Alert");
                                    alert.getLocation().setValue("19.2909353, -
99.163239");
                                    alert.getSeverity().setValue("critical");
                                    alert.getSubCategory().setValue("unknown");
alert.getValidFrom().setValue(Functions.getActualDate());

alert.getValidTo().setValue(Functions.getActualDate());
                                    try {
                                        deviceController.createEntity(MAIN_CONTEXT,
alert.getId(), alert);
                                    } catch (Exception e) {
                                        e.printStackTrace();
                                    }
                            }
                        });
        AlertDialog alert = alertDialogBuilder.create();
        alert.show();
    }
}
```

Anexo 5 Modelo de datos Building

Este modelo de datos hace referencia al nombre del modelo de datos zona (zone).


```
{
   "category": [],
   "location": [
               [18.875628, -99.219688],
               [18.875658, -99.219854],
               [18.875739, -99.220031],
               [18.875886, -99.22016],
               [18.876039, -99.220176],
               [18.876227, -99.220155],
               [18.876419, -99.220181],
               [18.876638, -99.220235],
               [18.876846, -99.220305],
               [18.877161, -99.220294],
               [18.87744, -99.220267],
               [18.877511, -99.219039],
               [18.876785, -99.219162],
               [18.875628, -99.219688]
   ],
   "centerPoint": [18.876443, -99.22011499999999],
   "idZone": "Zone_1527792778837",
   "type": "Building",
   "refBuildingType": "Zone",
   "name": "Cenidet Palmira",
   "address": "Interior Internado Palmira S\/N, Palmira, 62490 Cuernavaca,
Mor.",
   "description": "Campus 1",
   "dateCreated": "2018-05-31T18:52:58.000Z",
   "dateModified": "2018-05-31T18:52:58.000Z",
   "status": "1"
}
```

Anexo 6 El modelo de datos OffStreetParking

Este modelo de datos hace referencia al nombre del modelo de datos estacionamiento.


```
{
        "category": ["private", "forEmployees", "forVisitors", "forStudents"],
        "location": [
                [18.879981, -99.221991],
                [18.880031, -99.221932],
                [18.880042, -99.221851],
                [18.880042, -99.221747],
                [18.879983, -99.221647],
                [18.879884, -99.221548],
                [18.879752, -99.221417],
                [18.879633, -99.221253],
                [18.879511, -99.221154],
                [18.879425, -99.220937],
                [18.879438, -99.220824],
                [18.879064, -99.221068],
                [18.878861, -99.221138],
                [18.878615, -99.221181],
                [18.878745, -99.221468],
                [18.878943, -99.22161],
                [18.879039, -99.221524],
                [18.879138, -99.221489],
                [18.879224, -99.221524],
                [18.879313, -99.221583],
                [18.879399, -99.221672],
                [18.879468, -99.22176],
                [18.879656, -99.221977],
                [18.879778, -99.222063],
                [18.879892, -99.222047],
                [18.879981, -99.221991]
        ],
        "idOffStreetParking": "OffStreetParking_1523933778251",
        "type": "OffStreetParking",
        "name": "CENIDET APATZINGÁN Parking",
        "description": "ZONA CENIDET PALMIRA",
        "areaServed": "Zone_1524284309191",
        "dateCreated": "2018-04-17T02:56:18.000Z",
        "dateModified": "2018-04-17T02:56:18.000Z",
        "status": "1"
}
```

Anexo 7 Modelo de datos RoadSegment

Este modelo de datos hace referencia al nombre del modelo de datos segmento de calle/carretera.


```
{
        "location": [
                [18.879885497821526, -99.22159684589134],
                [18.879776367934124, -99.22174168517815],
                [18.879776367934124, -99.22174168517815],
                [18.879718, -99.2216666],
                [18.879718, -99.2216666],
                [18.8794501, -99.22154322]
        ],
        "startPoint": [18.879885497821526, -99.22159684589134],
        "endPoint": [18.8794501, -99.22154322],
        "laneUsage": ["forward"],
        "idRoadSegment": "RoadSegment_1524288282948",
        "type": "RoadSegment",
        "name": "Calle Estacionamiento Mecatronica 3",
        "refRoad": "OffStreetParking_1523933778251",
        "totalLaneNumber": 1,
        "maximumAllowedSpeed": 10,
        "minimumAllowedSpeed": 5,
        "width": 4,
        "dateCreated": "2018-04-21T05:24:42.000Z",
        "dateModified": "2018-04-21T05:24:42.000Z",
        "status": "1"
}
```

```
{
        "location": [
                [18.8783548, -99.2215944],
                [18.8783234, -99.2217209],
                [18.8783076, -99.2217739],
                [18.8782881, -99.2218191],
                [18.8782682, -99.2218577],
                [18.8782337, -99.2219008],
                [18.8781895, -99.2219476],
                [18.8781413, -99.2219965],
                [18.8780963, -99.2220401],
                [18.8780538, -99.2220716],
                [18.8780005, -99.2221031],
                [18.8779483, -99.2221253],
                [18.8779047, -99.2221386],
                [18.8778425, -99.2221487],
                [18.8777735, -99.222153],
                [18.8777181, -99.2221514],
                [18.8776643, -99.2221421],
                [18.8776153, -99.2221293],
                [18.8775714, -99.2221141],
                [18.8774404, -99.2220505],
                [18.8772315, -99.2219288],
                [18.87716, -99.2218952],
                [18.8771204, -99.2218798],
                [18.8770231, -99.2218513],
                [18.876606, -99.2217478],
                [18.8765021, -99.221714],
                [18.8764415, -99.2216901],
                [18.8763984, -99.2216662],
                [18.8763529, -99.2216332],
                [18.8763172, -99.2216049],
                [18.8762823, -99.2215705],
                [18.8762458, -99.2215294],
```

```
                    [18.8762042, -99.2214677],
                    [18.8761155, -99.2213139],
                    [18.8758447, -99.2207862],
                    [18.8757322, -99.2206018],
                    [18.8756334, -99.2204517],
                    [18.8755456, -99.2203161],
                    [18.8754773, -99.2202223],
                    [18.8754175, -99.2201538],
                    [18.8753567, -99.2201019]
            ],
            "startPoint": [18.8783548, -99.2215944],
            "endPoint": [18.8753567, -99.2201019],
            "laneUsage": ["forward"],
            "idRoadSegment": "RoadSegment_1524289166358",
            "type": "RoadSegment",
            "name": "Avenida Palmira",
            "refRoad": "Road_1524105730856",
            "totalLaneNumber": 1,
            "maximumAllowedSpeed": 20,
            "minimumAllowedSpeed": 5,
            "width": 3,
            "dateCreated": "2018-04-21T05:39:26.000Z",
            "dateModified": "2018-04-21T05:39:26.000Z",
            "status": "1"
}
```

Anexo 8 Modelo de datos Alert


```
{
        "id": "Alert:Device_Smartphone_40bbcd9c6cae67c0:1525126778",
        "type": "Alert",
        "alertSource": {
                "type": "Text",
                "value": "Device_Smartphone_40bbcd9c6cae67c0",
                "metadata": {}
        },
        "category": {
                "type": "Text",
                "value": "security",
                "metadata": {}
        },
        "dateObserved": {
                "type": "DateTime",
                "value": "2018-04-30T17:19:38.00Z",
                "metadata": {}
        },
        "description": {
                "type": "Text",
                "value": "Vel. permitida 20.0km/h. Vel. del vehiculo
39.82km/h.",
                "metadata": {}
        },
        "location": {
                "type": "geo:point",
                "value": "18.88129038, -99.2205856",
                "metadata": {}
        },
        "severity": {
                "type": "Text",
                "value": "critical",
                "metadata": {}
        },
        "subCategory": {
                "type": "Text",
                "value": "UnauthorizedSpeeDetection",
                "metadata": {}
        },
        "validFrom": {
                "type": "DateTime",
                "value": "2018-04-30T17:19:38.00Z",
                "metadata": {}
        },
        "validTo": {
                "type": "DateTime",
                "value": "2018-04-30T17:19:38.00Z",
                "metadata": {}
        }
}
```