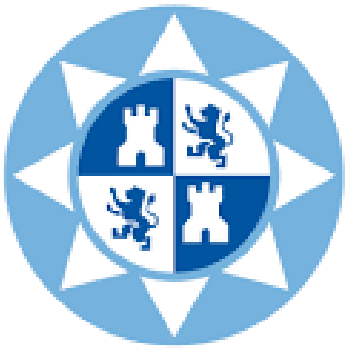
UNIVERSIDAD POLITÉCNICA DE CARTAGENA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN



Universidad Politécnica de Cartagena

Programa Oficial de Postgrado (Plan 3052):
Tecnologías de la Información y Comunicaciones

TESIS DOCTORAL

Retos Tecnológicos en la IoT en el ámbito de las Redes de Sensores.

Doctorando:
Ramón Martínez Carreras.

Directores:
Juan Ángel Pastor Franco.
Bárbara Álvarez Torres.

División de Sistemas e Ingeniería Electrónica (DSIE).

Diciembre, 2016

Universidad Politécnica de Cartagena

DT-16

CONFORMIDAD DE SOLICITUD DE AUTORIZACIÓN DE DEPÓSITO DE TESIS DOCTORAL POR EL/LA DIRECTOR/A DE LA TESIS

D.Juan Ángel Pastor Franco y Dª. Bárbara Álvarez Torres Directores de la Tesis doctoral "Retos Tecnológicos en la IoT en el ámbito de las Redes de Sensores"

INFORMAN:

Que la referida Tesis Doctoral, ha sido realizada por D. Ramón Martínez Carreras, dentro del Programa de Doctorado Tecnologías de la Información y las Comunicaciones, dando nuestra conformidad para que sea presentada ante el Comité de Dirección de la Escuela Internacional de Doctorado para ser autorizado su depósito.

La rama de conocimiento en la que esta tesis ha sido desarrollada es:

- ☐ Ciencias
- ☐ Ciencias Sociales y Jurídicas
- X Ingeniería y Arquitectura

En Cartagena, a 14 de diciembre de 2016

LOS DIRECTORES DE LA TESIS

Fdo.: Juan Ángel Pastor Franco

Fdo: Bárbara Álvarez Torres

COMITÉ DE DIRECCIÓN ESCUELA INTERNACIONAL DE DOCTORADO

Universidad Politécnica de Cartagena

DT-17

CONFORMIDAD DE DEPÓSITO DE TESIS DOCTORAL POR LA COMISIÓN ACADÉMICA DEL PROGRAMA

D. Fernando Quesada Pereira, Presidente/a de la Comisión Académica del Programa Tecnologías de la Información y Comunicaciones

INFORMA:

Que la Tesis Doctoral titulada, "Retos tecnológicos en la IoT en el ámbito de las Redes de Sensores", ha sido realizada, dentro del mencionado Programa de Doctorado, por D. Ramón Martínez Carreras, bajo la dirección y supervisión del Dr. Juan Ángel Pastor Franco y la Dra. Bárbara Álvarez Torres.

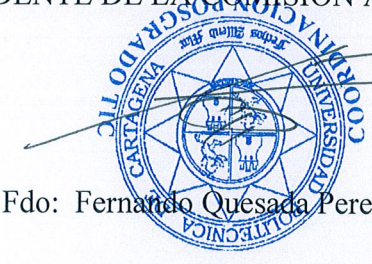
En reunión de la Comisión Académica de fecha 14/12/16, visto que en la misma se acreditan los indicios de calidad correspondientes y la autorización del Director/a de la misma, se acordó dar la conformidad, con la finalidad de que sea autorizado su depósito por el Comité de Dirección de la Escuela Internacional de Doctorado.

La Rama de conocimiento por la que esta tesis ha sido desarrollada es:

- [ ] Ciencias
- [ ] Ciencias Sociales y Jurídicas
- [X] Ingeniería y Arquitectura

En Cartagena, a 14 de diciembre de 2016

EL PRESIDENTE DE LA COMISIÓN ACADÉMICA



Fdo: Fernando Quesada Pereira

COMITÉ DE DIRECCIÓN ESCUELA INTERNACIONAL DE DOCTORADO

Agradecimientos

Inicié mi etapa investigadora tras mantener largas conversaciones con Juan Ángel Pastor en las que me explicó detalladamente el difícil camino que hay que recorrer para llegar a ser doctor, especialmente en el campo de la ingeniería de *software*. Yo estaba convencido de que mi objetivo a largo plazo era trabajar como investigador o conseguir un puesto de trabajo acorde a mi profesión pero necesitaba al menos una persona de confianza para emprender este viaje.

Al poco tiempo, Juan Ángel y yo concertamos una reunión con Bárbara Álvarez para intentar que formara parte de la dirección de esta tesis. Cuando esta terminó, disponía de los mismos directores que orquestaron mi proyecto fin de carrera. En ese momento, supe que finalizaría este trabajo ya que Juan Ángel y Bárbara tienen virtudes que se complementan a la perfección. Ha llegado la hora de daros las gracias porque sois unos investigadores excepcionales, pero sobre todo porque sois unas personas ejemplares que me habéis guiado y ayudado en todo momento. Muchas gracias a los dos de todo corazón.

Cuando se decidió como tema de la investigación "Internet de las Cosas" apareció en escena Andrés Iborra (director del grupo de investigación DSIE) con el fin de ayudarme a encontrar financiación para la realización de mi tesis. Gracias a él, empezamos a investigar acerca de la plataforma FIWARE. Al principio, realicé trabajos junto a José Luis Romero fruto de una beca de colaboración. En aquel periodo, Andrés y yo viajamos a Madrid para comprobar si existía alguna aceleradora europea que destinara ayudas a proyectos FIWARE cuyo ámbito de aplicación coincidiera con el de alguna spin-off de la UPCT. Llegamos a participar en un concurso con la spin-off Widhoc en el que recibí adiestramiento de los técnicos de FIWARE. Desafortunadamente, esta propuesta quedó a un solo puesto de ser premiada. Además, gracias a Andrés, Pedro Sánchez y Andrés Carrillo he adquirido conocimientos acerca del funcionamiento de la aceleradora *Cloud Incubator* HUB así como de conceptos y herramientas muy utilizadas en el emprendimiento. Por todo esto, he de agradecer a Andrés Iborra que siempre me haya considerado un miembro más del grupo DSIE. También quisiera mandar un fuerte abrazo a Pedro Sánchez porque ha seguido muy de cerca tanto la evolución de mis trabajos como de mi estado de ánimo siendo una referencia en todo momento.

Siempre recordaré las reuniones en la cantina de la universidad. No hay nada mejor como el humor de Diego Alonso, Pedro Navarro, Carlos Fernández y Pedro Alcover para desconectar por unos minutos antes de afrontar el resto de la jornada. Aunque no haya tenido la suerte de compartir tanto tiempo o de no haberlo podido aprovechar mejor, también recordaré con afecto al resto de integrantes del grupo así como a los integrantes del servicio técnico de la universidad.

Finalmente, quisiera agradecer su apoyo incondicional a mis amigos pero, sobre todo, a mi familia: a mis primos, tíos, hermanos, cuñados, padres, suegros y abuelos porque han sido los mejores espectadores de este trabajo. Y para terminar, quisiera hacer mención especial de María, mi mujer, por su ayuda, compresión y cariño y de Mar, mi hija, que ha llenado mi vida de felicidad.

A mi familia.

A mis padres y a mis suegros.
A Gregorio, por su atención y consejo.
A María, fuente de mi inspiración.
A Mar, símbolo de mi felicidad.

Índice general

Resumen

Internet de las Cosas es la visión de un conjunto de tecnologías, sistemas y principios de diseño para conectar cosas, basadas en el entorno físico, a través de Internet. Un componente esencial de Internet de las Cosas son las redes de sensores inalámbricas. Este tipo de redes son utilizadas en muchos ámbitos, siendo la agricultura de precisión uno de ellos. El grupo de investigación División de Sistemas e Ingeniería Electrónica (DSIE), en el que el autor ha desarrollado este trabajo de tesis, tiene una experiencia considerable en la aplicación de redes de sensores inalámbricas experimentales a explotaciones hortícolas del sureste de España.

Debido a los enormes avances en el campo de la electrónica, durante los próximos años se espera un incremento significativo en el despliegue de redes de sensores in situ que provocará numerosos desafíos técnicos. Diferentes *middleware* de Internet de las Cosas ayudarán a superar dichos desafíos. Este trabajo comprueba si una de estas soluciones *middleware*, FIWARE, podrá escalar en la misma medida en la que lo harán las aplicaciones agrícolas. La respuesta es importante porque el uso de una plataforma determinada supone una apuesta a largo plazo. Para superar dicha meta, se ha diseñado e implementado un banco de pruebas.

El banco de pruebas desarrollado es una aplicación típica FIWARE, completa, pero lo suficientemente simple y comprensible para mostrar las principales características y componentes de FIWARE, así como la complejidad de la utilización de esta tecnología. Aunque el banco de pruebas ha sido desplegado en un entorno de laboratorio, su diseño está basado en el análisis de un escenario de caso de uso de Internet de las Cosas en el ámbito de la agricultura de precisión. Así, este banco de pruebas permite simular diferentes despliegues y condiciones de carga, obteniéndose a partir de cada simulación, un conjunto de medidas acerca del rendimiento de FIWARE. Esta información es tratada y representada gráficamente para el posterior estudio y análisis del rendimiento de la plataforma evaluada.

Los resultados y conclusiones de este trabajo de investigación son válidos para el análisis de otras aplicaciones inteligentes que integren, haciendo uso de otras plataformas, los patrones y componentes arquitectónicos del paradigma de Internet de las Cosas según la visión de la plataforma FIWARE, siempre que tengan los mismos requisitos que las aplicaciones agrícolas.

Abstract

Internet of Things is the vision of a set of technologies, systems and design principles to connect things, based on the physical environment, through Internet. An essential component of Internet of Things is wireless sensor network. This kind of network is used in many fields, being precision farming one of them. The author's research group, Electronics Engineering and Systems Division (DSIE), has considerable experience in applying such networks to horticultural farms in south-eastern Spain, most of which are experimental.

Due to the enormous advances in the electronics field, over the next few years a significant increase is expected in the deployment in situ of sensor networks, which will lead to numerous technical challenges. Different Internet of Things middleware will help overcome these challenges. This paper checks whether one of these middleware solutions, FIWARE, can be scaled to the same extent as agricultural applications. The answer is important because the selection of a particular platform implies a long-term commitment. We have created a testbed to achieve this goal.

The testbed is a typical FIWARE application, complete, yet simple and comprehensible enough to show the main features and components of FIWARE, as well as the complexity of using this technology. Although the testbed has been deployed in a laboratory environment, its design is based on the analysis of an Internet of Things use case scenario in the domain of precision agriculture. Thus, this testbed allows to simulate different deployments and load conditions. A set of measures about the performance of FIWARE is obtained from each simulation. This information is treated and graphically represented for the subsequent performance study and analysis of the platform evaluated.

The results and conclusions of this research work are valid for the analysis of other smart applications that integrate, using other platforms, the architectural patterns and components of the Internet of Things paradigm according to the vision of the FIWARE platform, provided they have the same requirements as agricultural applications.

Lista de Figuras

Lista de Tablas

Lista de Códigos

Lista de Acrónimos

CPS	**Cyber-Physical System**
ICT	**Information and Communication Technologies**
IoT	**Internet of Things**
RFID	**Radio Frequency Identification**
HIoT	**Human Internet of Things**
IIoT	**Industrial Internet of Things**
WSAN	**Wireless Sensor and Actuator Network**
PA	**Precision Agriculture**
WSN	**Wireless Sensor Network**
M2M	**Machine to Machine**
SCADA	**Supervisory Control And Data Acquisition**
SN	**Sensor Networks**
BSN	**Body Sensor Network**
VSN	**Visual or Video Sensor Network**
V2V,V2I	**Vehicular Sensor Network**
USN	**Ubiquitous Sensor Network**
DNA	**Devices-Networks-Applications**
DCM	**Device-Connect-Manage**
HVAC	**Heating, ventilation and air conditioning**
MEMS	**Microelectromechanical systems**
NEMS	**Nanoelectromechanical systems**
PAN	**Personal Area Network**
LAN	**Local Area Network**
MAN	**Metropolitan Area Network**
MN	**Mesh Network**
WAN	**Wide Area Network**
GSM	**Global System for Mobile communications**
WiMax	**Worldwide Interoperability for Microwave Access**
CDMA	**Code Division Multiple Access**
AP	**Access point**
MPU	**Microprocessor**
MCU	**Microcontroller**
OSI	**Open Systems Interconnection**

UMTS	**Universal Mobile Telecommunications System**
LTE	**Long-Term Evolution**
TCP	**Transmission Control Protocol**
HTTP	**Hypertext Transfer Protocol**
SMTP	**Simple Mail Transfer Protocol**
FTP	**File Transfer Protocol**
URI	**Universal Resource Identifier**
CoAP	**Constrained Application Protocol**
MQTT	**Message Queue Telemetry Transport**
XMPP	**Extensible Messaging and Presence Protocol**
DDS	**Data Distribution Service**
AMQP	**Advanced Message Queuing Protocol**
SOA	**Service Oriented Architecture**
SaaS	**Software as a Service**
BI	**Business Intelligence**
UID	**Unique ID**
ONS	**Object Name Service**
EPC	**Electronic Product Code**
WoT	**Web of Things**
SSW	**Semantic Sensor Web**
KMF	**Knowledge Management Framework**
REST	**Representational State Transfer**
XML	**eXtensible Markup Language**
JSON	**JavaScript Object Notation**

Parte I

Preliminares

CAPÍTULO 1

Introducción

We need an Internet for things, a standardized way for computers to understand the real world.

Kevin Ashton
Forbes Magazine

1.1. Antecedentes

Los enormes avances que se han producido en la última década en los campos de la electrónica y las comunicaciones han permitido la rápida extensión de los sistemas de telecomunicaciones inalámbricas, los dispositivos móviles, así como el desarrollo de servicios ubicuos que proporcionan conectividad en cualquier momento y lugar. Hoy, sin embargo, el papel que desempeñan los dispositivos ya no se limita a conectar a los usuarios a través de Internet, sino que se ha convertido en una oportunidad para vincular los mundos físico y cibernético [1], dando lugar a la aparición de los sistemas ciber-físicos (CPS) [2] [3]. La noción de CPS se refiere a una nueva generación de sistemas de tecnologías de la información y comunicaciones (ICT) embebidos donde la computación y la interconexión se integran con procesos físicos para controlar y gestionar su dinámica y hacerlos más eficientes, fiables, adaptables y seguros [4] [5]. La información acerca de los procesos físicos, por ejemplo, obtenida a través de sensores, es transferida, procesada y utilizada en el mundo digital, pero también puede afectar a dichos procesos a través de lazos de retroalimentación, por ejemplo mediante el uso de actuadores [1]. La peculiaridad de los CPS es que el sistema ICT está diseñado junto con los componentes físicos para maximizar la eficiencia global, por lo tanto están en contraste con los sistemas embebidos clásicos donde el objetivo es incluir la electrónica/ computación/ comunicación/ abstracción en un mundo físico ya operativo.

Los sistemas ciber-físicos tendrán un gran impacto en la sociedad del futuro, y sus redes sociales, jugarán un papel central estableciendo puente entre los mundos cibernético, físico y social [6] [7]. A través de sus interacciones con dispositivos ICT, los CPS tendrán un mayor acceso al mundo virtual afectando a la manera en la que la información es distribuida y aportarán su contribución para construir/ modificar la infraestructura cibernética.

El valor económico asociado a los CPS también será grande. En el informe del McKinsey Global Institute de 2013[1], se han identificado doce tendencias tecnológicas que, en 2025,

[1] http://www.slideshare.net/brandsynapse/mgi-disruptive-technologiesfullreportmay2013?related=1.

tendrán un impacto masivo y económicamente disruptivo, dirigiendo cambios profundos en muchas dimensiones: en las vidas de los ciudadanos, en los negocios y en la economía mundial. Específicamente, este informe hace referencia a cuatro tendencias en el campo de los CPS:

1. La automatización del conocimiento del trabajo.
2. Internet de las Cosas.
3. La robótica avanzada.
4. Vehículos autónomos/ casi autónomos.

Entre las tendencias citadas, Internet de las Cosas, con un valor estimado de 36 billones de dólares, se considera el paradigma de los sistemas ciber-físicos con mayor impacto económico [8].

1.2. Internet de las Cosas

Internet de las Cosas (IoT) significa mucho más que cosas interconectadas. Se trata de un paradigma muy difícil de definir debido a tres características: sus inicios, su desarrollo acelerado y su amplia extensión.

El origen de este paradigma son los trabajos de investigación acerca de la "computación ubicua" realizados por Mark Weiser (Xerox PARC) y publicados a principios de los 90 [9] [10]. Posteriormente, el papel desempeñado por Kevin Ashton fue crucial ya que acuñó el término "Internet de las Cosas" en 1998 [11]. Además, es considerado como un pionero tecnológico ya que su idea fue la semilla del Centro Auto-ID del MIT, donde se creó la comunidad de desarrollo de la identificación por radiofrecuencia (RFID). A partir de ese momento, se han realizado numerosos trabajos de investigación para descubrir las posibilidades que ofrece IoT.

Hoy en día, el concepto de "cosa" es más general y no se limita solo a RFID. Una "cosa" puede ser cualquier objeto físico/real (por ejemplo: RFID, sensores, actuadores, elementos inteligentes[2]), pero también cualquier entidad digital/ virtual, que pueda situarse en el tiempo y en el espacio y que pueda ser identificada de forma única mediante asignación numérica, de nombres y/ o por su localización. Así, en IoT cualquier "cosa" pasa a ser fácilmente legible, reconocible, localizable, direccionable y/ o controlable a través de Internet. Además, esta nueva generación de dispositivos es inteligente gracias a los avances en la electrónica embebida. Es decir, las "cosas" son habilitadas para la realización de tareas de detección/actuación, cómputo y comunicación. Como resultado, se consigue tanto su integración en el entorno como su interconexión con la infraestructura ciber-física mundial.

Durante la próxima década, se prevé una transición gradual a este nuevo paradigma. Sin embargo, será necesario abordar una complejidad significativa. Por un lado, miles de millones de objetos inteligentes estarán inmersos en el entorno, detectando, interactuando y cooperando entre ellos para permitir servicios eficientes que aportarán beneficios tangibles

[2]Los elementos inteligentes tienen características muy avanzadas, tales como la adopción de un comportamiento autónomo y proactivo. Por ejemplo, son capaces de generar tráfico de forma autónoma para ciertos fines, de ejecutar procesamiento de datos o de realizar la comunicación de una forma colaborativa.

para el entorno, la economía y la sociedad en general. Por otro lado, estos objetos serán extremadamente heterogéneos y diversos en términos de tecnologías de comunicación, vida útil y capacidades de recursos, complicando aún más el panorama. Como resultado, están apareciendo nuevos retos tecnológicos que se expanden por diferentes áreas: arquitectura, comunicación, direccionamiento, descubrimiento, gestión de red y datos, almacenamiento de energía y potencia, privacidad y seguridad, por citar algunas [12] [13] [14].

Los enfoques clásicos de Internet no son suficientes para resolver estos problemas sin precedentes y necesitan ser revisados para abordar los complejos requisitos impuestos por IoT [15]. Prueba de ello es la desaparición del concepto de Internet como una red de infraestructura que alcanza los terminales de los usuarios finales. En su lugar, se abre paso a la noción de "objetos inteligentes" que están interconectados para formar entornos de computación ubicuos/ pervasivos [9]. El término computación ubicua está ligado al de computación pervasiva, computación sensible al contexto, inteligencia ambiental y computación móvil, entre otros. [16] define espacio ubicuo como "aquel que está repleto de dispositivos de computación, transparentes para los individuos y con los que pueden interactuar de forma natural, sin ser completamente conscientes de la complejidad subyacente". A pesar de esta nueva noción, la infraestructura de Internet no desaparecerá. Por el contrario, mantendrá su papel vital como columna vertebral global para la difusión e intercambio de información en todo el mundo, interconectando los objetos físicos con capacidades de computación/ comunicación a través de una amplia variedad de nuevas tecnologías y servicios.

1.2.1. ***Visiones y concepto***

Internet de las Cosas abarca la mayoría de tecnologías y dominios de aplicación, desde el *hardware* puro hasta el *software* más abstracto, desde el ámbito de las aplicaciones hasta el de los datos, con una gran cantidad de interacciones y superposiciones. [17] clasifica las visiones de IoT propuestas en los últimos años en tres categorías:

1. Visiones orientadas a las Cosas: prestan atención a los objetos y a la búsqueda de un paradigma capaz de identificarlos e integrarlos. En las primeras visiones orientadas a las cosas, estas eran elementos muy simples con capacidades de identificación. Las visiones actuales de IoT van más allá y consisten en mucho más que la identificación de objetos. Las capacidades incluyen habilidades de identificación, localización y sostenibilidad. Las cosas no sólo están equipadas con capacidades de comunicación inalámbrica, memoria y computación, sino que también son capaces de exhibir un comportamiento autónomo y proactivo, sensibilidad al contexto y comunicaciones colaborativas sólo por mencionar algunas.
2. Visiones orientadas a Internet: se componen de las nuevas dimensiones que se incorporan a las tecnologías de la información y las comunicaciones. Estas dimensiones incluyen la comunicación entre las cosas y entre las personas y las cosas. Profundizan en el paradigma de red y en la explotación del protocolo IP para establecer una conexión eficiente entre los dispositivos. Al mismo tiempo, se centran en la simplificación de IP para que pueda ser utilizado en dispositivos con capacidades muy limitadas.

3. Visiones orientadas a la Semántica: la idea de que todas las cosas estén conectadas plantea problemas potenciales. Uno será el increíble número de elementos que habrá en el Internet del Futuro [18]. Desde estas visiones, se detectan desafíos relativos a cómo representar, almacenar, interconectar, buscar y organizar la gran cantidad de información generada por IoT. Una de las soluciones sugeridas para resolver el problema es el uso de tecnologías semánticas para la descripción de las cosas y el razonamiento sobre los datos.

Los autores de este trabajo concluyen que Internet de las Cosas es el resultado de la convergencia de estas visiones (Figura 1.1).

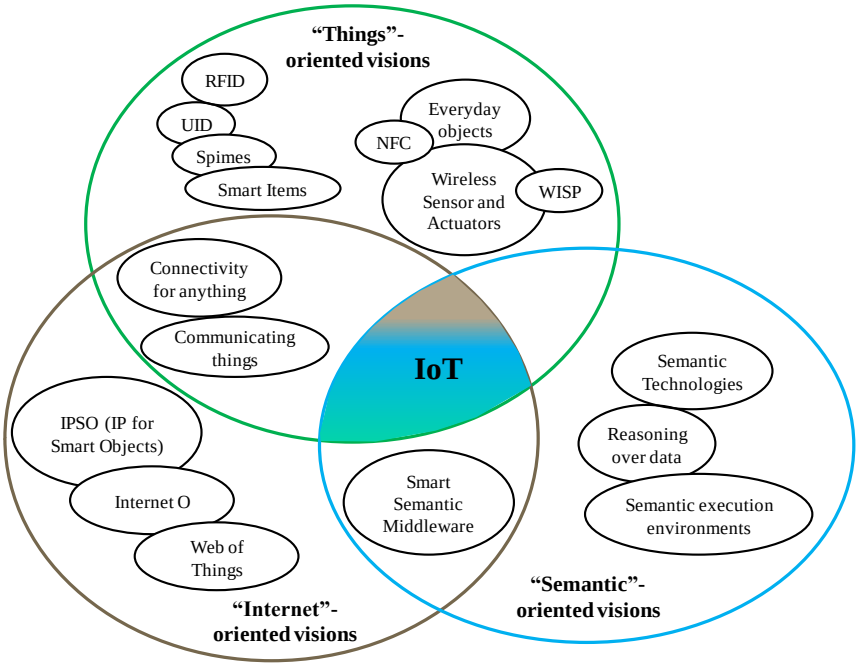


Figura 1.1. IoT como resultado de la convergencia de diferentes visiones [17].

Además, IoT puede ser visto como soporte a los consumidores (humanos) o a las aplicaciones industriales. De hecho, podría ser denominado como Internet de las cosas humanas (HIoT) o Internet de las cosas industriales (IIoT) [19] [20] [21]. A pesar de que estos diferentes puntos de vista han evolucionado debido a la naturaleza interdisciplinar de este paradigma, es necesaria su intersección en el dominio de aplicación para alcanzar los objetivos de la IoT [22].

Desde un punto de vista conceptual, Internet de las Cosas se construye sobre tres pilares relativos a la capacidad que tienen los objetos inteligentes para: (i) ser identificables (cualquier cosa se identifica), (ii) comunicarse (cualquier cosa se comunica) e (iii) interactuar (cualquier cosa interactúa), ya sea con otros objetos o con los usuarios finales u otras entidades en la red.

A continuación, se introducen diferentes perspectivas de IoT. A nivel de componente, este paradigma se basa en la noción de "objetos inteligentes", o, simplemente, "cosas", que complementan las entidades existentes en el dominio de Internet (servidores, *routers*, etc.) [23]. En [13] se define objeto inteligente (o cosa) como una entidad que:

- Tiene una materialización física y un conjunto de características físicas asociadas (por ejemplo, tamaño, forma, etc.).
- Tiene un conjunto mínimo de funciones de comunicación, tales como la capacidad de ser descubierto y de aceptar mensajes entrantes y responder a los mismos.
- Está asociado a una dirección y nombre únicos. El nombre es una descripción del objeto legible por humanos y se puede utilizar para propósitos de razonamiento. La dirección es una cadena legible por máquinas que se puede utilizar para comunicarse con el objeto.
- Posee algunas capacidades básicas de computación. Esto puede variar desde la capacidad para encontrar un mensaje entrante en una huella impresa dada (RFID) hasta la capacidad de la realización de cálculos bastante complejos, incluyendo descubrimiento de servicios y tareas de gestión de red.
- Puede poseer medios para detectar los fenómenos físicos (por ejemplo, temperatura, luz, nivel de radiación electromagnética) o para desencadenar acciones que tienen un efecto en la realidad física (actuadores).

El último punto de la definición anterior es clave, ya que diferencia a los objetos inteligentes de las entidades tradicionales consideradas en los sistemas de red. En particular, la clasificación propuesta incluye tanto a los dispositivos RFID [24] como a los que componen las redes de sensores inalámbricas (WSN) y de sensores/ actuadores (WSAN) [25] [26].

Desde una perspectiva a nivel de sistema, Internet de las Cosas puede ser visto como una red altamente dinámica y distribuida, compuesta de un número elevado de objetos inteligentes que producen y consumen información. La capacidad de interactuar con el mundo físico se consigue gracias a dos clases de dispositivos:

1. Sensores: son capaces de detectar los fenómenos físicos y traducirlos en flujos de información de datos. Es decir, se encargan de proporcionar información sobre el entorno y/ o contexto.
2. Actuadores: son capaces de desencadenar acciones que tienen un impacto en el mundo físico.

Desde una perspectiva a nivel de servicio, el principal problema es la integración (o composición) de las funcionalidades y/ o los recursos proporcionados por los objetos inteligentes en servicios [27] [28] [29]. Este proceso requiere:

- Arquitecturas y métodos para virtualizar “cosas” mediante la creación de una representación estándar de objetos inteligentes en el dominio digital, capaz de ocultar la heterogeneidad de dispositivos/ recursos.
- Métodos para una perfecta integración y composición de los recursos y servicios de los objetos inteligentes en servicios de valor añadido para los usuarios finales.

Para solucionar tanto los desafíos mencionados de heterogeneidad e interoperabilidad como otros inherentes al paradigma de IoT es necesario el desarrollo de *middleware*. Debido a su importancia, cada vez hay un mayor número de plataformas de *middleware* IoT. WSO_2 [30], PubNub [31] y FIWARE [32]son claros ejemplos de este nuevo tipo de plataformas.

Finalmente, desde un punto de vista de usuario, este paradigma permitirá ofrecer una gran cantidad de nuevos servicios reactivos en todo momento, que deberán responder a las necesidades de los usuarios y asistirlos en sus actividades cotidianas. Además, IoT ofrecerá un cambio en el suministro del servicio, pasando de la visión actual, propia de la Web, en la que siempre estamos conectados a los servicios a la visión de servicios localizables reactivos en todo momento, desarrollados y compuestos en tiempo de ejecución para responder a una necesidad específica en función del contexto del usuario. Cuando un usuario tenga unas necesidades específicas, enviará una solicitud y una aplicación ad hoc, compuesta de forma automática y desplegada en tiempo de ejecución adaptada al contexto específico del usuario, satisfará dichas necesidades.

*1.2.2. **Dominios de Aplicación***

Internet de las Cosas ofrece un amplio abanico de oportunidades a los usuarios, fabricantes y empresas. De hecho, las tecnologías empleadas tendrán aplicación en casi todos los campos. Según [14], las diversas aplicaciones inteligentes se pueden agrupar en tres dominios (Figura 1.2).

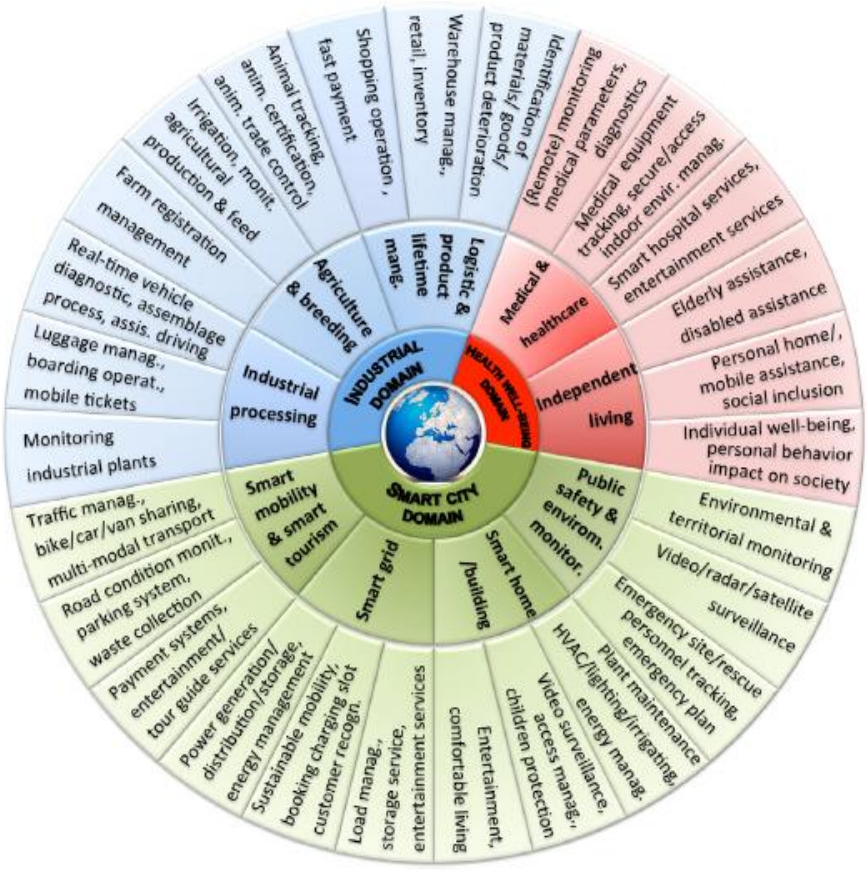


Figura 1.2. Dominios de aplicación en el Internet de las Cosas [14].

1. Dominio de Ciudad Inteligente: las aplicaciones IoT pueden ayudar a aumentar la sostenibilidad ambiental de nuestras ciudades y la calidad de vida de las personas.
2. Dominio de la Salud y el Bienestar: las aplicaciones IoT jugarán un papel esencial para proporcionar y mejorar las actividades de las personas y la sociedad.

3. Dominio Industrial: IoT puede ser explotado en todas las actividades industriales que implican transacciones comerciales o financieras entre empresas, organizaciones y otras entidades. Cabe destacar que, según esta clasificación, las aplicaciones IoT agrícolas pertenecen a este dominio.

1.2.3. ***Definiciones IoT***

Internet de las Cosas ha sido definido en multitud de ocasiones. Sin embargo, es casi imposible que todo el mundo esté de acuerdo en una definición ya que, como se ha mencionado, existen diferentes visiones y perspectivas de este paradigma. La primera definición de IoT surgió al considerar las etiquetas RFID como cosas (perspectiva "orientada a las cosas"). De acuerdo con la comunidad de RFID, IoT puede definirse como: "la red mundial de objetos interconectados direccionables únicamente basada en protocolos de comunicación estándar". Según el IoT *European Research Cluster* (IERC): "Internet de las cosas permite a las personas y cosas estar conectadas en cualquier momento, en cualquier lugar, con cualquier persona y cosa, idealmente utilizando cualquier ruta/ red y cualquier servicio (Figura 1.3)" [33] [34].

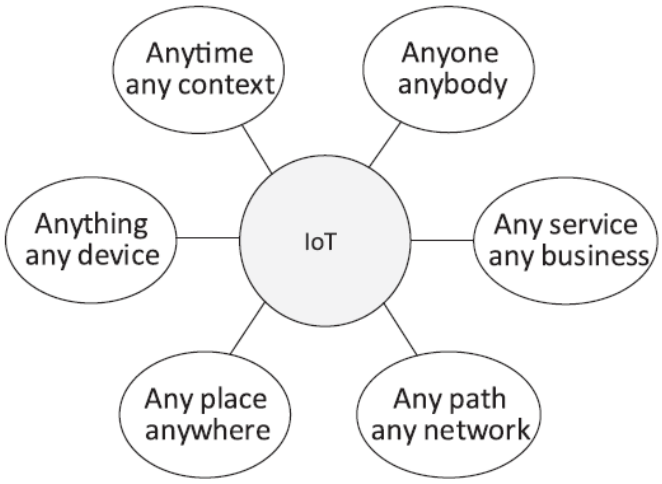


Figura 1.3. Definición de IoT [33].

La mayoría de las definiciones de IoT no destacan explícitamente la visión industrial de IoT (IIoT). Las compañías líderes del mundo están prestando especial atención a IoT y realizando importantes inversiones para sus soluciones industriales (IIoT). A pesar de que utilizan diferentes términos como "Planeta más inteligente" (*Smarter Planet*, IBM), "Internet de todas las cosas" (*Internet of Everything*, Cisco) e "Internet industrial" (*Industrial Internet*, GE), su principal objetivo es utilizar IoT para mejorar la producción industrial mediante la reducción del tiempo de inactividad no planificado de la maquinaria, lo que implica un disminución significativa de los costes energéticos junto con otros beneficios potenciales [35] [19] [20] [21]. El IIoT se refiere a objetos industriales, o "cosas", instrumentados con sensores, que se comunican automáticamente a través de una red, sin que existan interacciones hombre-hombre u hombre-máquina, para intercambiar información y tomar decisiones inteligentes con el soporte de análisis avanzados [21].

Debido al ámbito de este trabajo, procedemos a definir IoT desde una perspectiva *middleware*. Para ello, se extiende la definición propuesta por [36]: "Internet de las Cosas es la visión de una infraestructura global que interconecta todos los objetos inteligentes para proporcionar la capacidad de compartir su información a través de diversas plataformas *middleware*. Estas plataformas implementan una arquitectura unificada con el fin de ofrecer un *framework* común para el desarrollo de aplicaciones inteligentes. Esto se consigue gracias a la utilización de tecnologías innovadoras de la información y comunicaciones que permiten una perfecta realización de multitud de tareas como la detección/actuación a gran escala, el análisis masivo de datos o la representación de la información, entre otras".

1.3. Motivación, objetivos y estructura de esta Tesis

La agricultura de precisión (PA) tiene como objetivo aumentar la productividad de las explotaciones agrícolas mediante la captura e interpretación de datos referentes al clima, al tiempo atmosférico, a las características del terreno y al estado de los cultivos.

El mayor reto de la PA es capturar suficientes datos y con suficiente calidad para la toma de decisiones. En espacios abiertos, la mayor parte de los datos proviene de fotos aéreas realizadas por satélites y aviones, así como de sensores y cámaras montados en la propia maquinaria agrícola [37]. La agricultura de precisión necesita utilizar una gran variedad de tecnologías para adquirir los datos, referenciarlos en el espacio y en el tiempo, caracterizarlos estadísticamente, almacenarlos, fusionarlos y finalmente analizarlos para la toma de decisiones. En este escenario, el procesamiento se basa frecuentemente en técnicas estadísticas y de visión artificial donde la latencia no es un parámetro de gran relevancia [38].

Según se ha descrito en el apartado 1.2.1, un componente esencial de Internet de las Cosas son las redes de sensores inalámbricas (WSN). En estas redes, nuestro grupo tiene bastante experiencia. Específicamente, en el sector de la agricultura, hemos trabajado en la mejora de sistemas de cultivos y riego debido a su importancia en la Región semiárida de Murcia (España). En [39] se presenta el diseño de un novedoso nodo sensor inalámbrico para aplicaciones de precisión en la horticultura, que permite el uso de instrumentos agrícolas de precisión. Este sensor fue integrado en la red de sensores experimental de una empresa de horticultura ecológica de la Región de Murcia [40], validado en condiciones reales de operación [41] y comparado con otros trabajos [42].

Actualmente, las WSN in situ no están muy extendidas debido al elevado precio de los sensores y a los problemas que supone su alimentación en lugares donde no existe tendido eléctrico. En ausencia de fuentes de alimentación, la actividad de los sensores para capturar y transmitir datos debe limitarse lo máximo posible. Sin embargo, es previsible que estas limitaciones cambien muy significativamente en los próximos años, debido a los avances en miniaturización y alimentación, así como a la reducción de precio que supondrá la fabricación masiva de sensores. Es posible, por tanto, pensar en escenarios densamente poblados por sensores, que miden parámetros del suelo (p.ej. humedad o concentración de nutrientes) o de la planta (p.ej. estrés hídrico). Capturar, transmitir, almacenar y procesar este volumen de información procedente de las WSN desplegadas en estos escenarios se traduce

en una serie de desafíos, especialmente en cuanto a tecnologías de integración, comunicaciones, bases de datos y computación. Afortunadamente existen diferentes *middleware* que liberan a los técnicos de la realización de la mayor parte del trabajo, si bien sólo algunos de ellos están orientados hacia IoT. Una de estas tecnologías *middleware* es FIWARE [43] cuya arquitectura continúa en proceso de implementación y cuya evolución está siendo seguida de cerca por nuestro grupo [44].

FIWARE es la tecnología promovida por la Comisión Europea para hacer posible IoT en el contexto del Internet del Futuro [18] en colaboración con la industria de las tecnologías de la información y comunicaciones. FIWARE es una tecnología poderosa y compleja, pero hasta donde llega nuestro conocimiento, no hay trabajos que muestren cómo utilizar esta tecnología en la agricultura de precisión y estudiar su viabilidad, su escalabilidad y su eficiencia para este tipo de aplicaciones. En concreto, los objetivos de este trabajo son:

1. Proporcionar un caso de uso comprensible en tal dominio de aplicación.
2. Comprobar si FIWARE podrá escalar en la misma medida que lo harán las aplicaciones agrícolas futuras. Para ello, se obtendrán diferentes medidas acerca de su rendimiento.
3. Crear un banco de pruebas que permita simular diferentes despliegues y condiciones de carga.
4. Los resultados deberán permitir el análisis de aplicaciones de otros dominios con requisitos similares a las aplicaciones agrícolas y que hagan uso de otras plataformas.

Las conclusiones son importantes porque la elección de una plataforma determinada supone una apuesta a largo plazo.

La presente memoria doctoral ha sido estructurada en tres partes claramente diferenciadas: preliminares, contribución y conclusiones. En cada una de estas tres partes se ha documentado de la forma más completa posible el modo en el que todos y cada uno de los objetivos indicados han sido abordados durante el desarrollo de la tesis.

En la *Parte I* de esta memoria se encuentran los capítulos considerados como preliminares. En el capítulo 1 se realiza una introducción a la temática de la tesis y a los principales objetivos abordados durante su desarrollo. En el capítulo 2 se ofrece un estado del arte detallado de aquellas tecnologías involucradas en la investigación. Además se introducen una serie de conceptos y antecedentes relacionados con la investigación cuyo conocimiento previo es importante para la correcta interpretación de los capítulos siguientes. En el capítulo 3 se especifica la metodología de investigación que ha conducido el desarrollo del trabajo realizado.

En la Parte II se encuentran los capítulos considerados como la principal contribución de la tesis. En el capítulo 4 se describen los conceptos y componentes arquitectónicos de FIWARE que permiten el desarrollo de aplicaciones IoT. Además, en este capítulo, se describe la aplicación de los conocimientos FIWARE estudiados en un escenario de caso de uso IoT en el ámbito de la agricultura de precisión. En el capítulo 5 se deduce el diseño del banco de pruebas desarrollado a partir del escenario presentado en el capítulo 4. Además, se proporciona información relativa al desarrollo, entorno de experimentación, configuración y

despliegue del banco de pruebas. En el capítulo 6 se detalla el plan de pruebas seguido para evaluar la plataforma FIWARE y se presentan los resultados obtenidos.

Para finalizar, en la Parte III se encuentra el capítulo 7 que contiene las conclusiones, los posibles trabajos futuros relacionados con esta investigación y las aportaciones derivadas de esta tesis.

CAPÍTULO 2

Estado del arte

Middleware is omnipresent as it exists nearly everywhere in an information and communications technologies system.

Honbo Zhou
The Internet of Things in the Cloud

2.1. Introducción

Los avances en diversas áreas científicas (ciencia de los materiales, tecnologías de la información y comunicaciones, producción y almacenamiento de energía, etc.) están haciendo posible la aparición de una nueva generación de dispositivos inalámbricos más pequeños y baratos. Así, muchas "cosas" de nuestra vida cotidiana se están convirtiendo en interoperables de forma inalámbrica. [45] prevé que en 2021 habrá 28 mil millones de dispositivos conectados para formar IoT [46] [47].

El principal objetivo de IoT es fomentar el desarrollo de aplicaciones y su colaboración en una gran cantidad y variedad de dominios (agricultura, ciudades inteligentes, automatización industrial, asistencia médica, gestión inteligente de la energía, etc.) [48] [49]. Gracias a este nuevo paradigma, será posible acceder a, e interactuar con, una amplia variedad de dispositivos o cosas (electrodomésticos, cámaras de vigilancia, sensores, actuadores, vehículos, máquinas, etc.). Por tanto, las aplicaciones IoT harán uso de la enorme cantidad y variedad de datos generados por tales dispositivos para proporcionar nuevos servicios a los ciudadanos, empresas y administraciones públicas [47], [36] [17]. Sin embargo, en un entorno de computación ubicuo como IoT, es muy complicado reunir un conjunto de estándares que sean mundialmente aceptados para, posteriormente, crear una arquitectura de referencia que permita diseñar cualquier aplicación IoT [22]. Una red de cosas a ultra gran escala, el gran número de datos generados espontáneamente por estas cosas y la heterogeneidad de dispositivos/ tecnologías/ aplicaciones de IoT, implican nuevos desafíos en el desarrollo de aplicaciones e incluso aumentan la complejidad de los desafíos ya existentes de la computación ubicua [46] [47] [15].

En este contexto, el *middleware* juega un papel clave ya que permite ofrecer servicios comunes a las aplicaciones. Concretamente, el *middleware* facilita el desarrollo de las aplicaciones IoT realizando funciones esenciales tales como la integración de dispositivos de

capacidades heterogéneas de computación y comunicaciones o el soporte relativo a la interoperabilidad dentro de las diversas aplicaciones y servicios que se ejecutan en estos dispositivos [50].

Según [35], los cuatro pilares de la IoT (Figura 2.1) son las tecnologías de identificación por radiofrecuencia (RFID), máquina-máquina (M2M), adquisición y control de datos (SCADA) y redes de sensores inalámbricas (WSN). Algunos desafíos de IoT se heredan de una o más de estas cuatro tecnologías mientras que otros son específicos de IoT. Por ello, un *middleware* de IoT completamente funcional necesita integrarlas para soportar los diversos dominios de aplicación previstos.

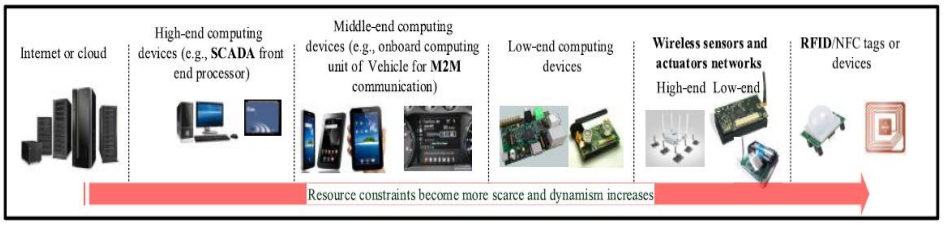


Figura 2.1. Ejemplos de heterogeneidad de dispositivos en IoT [50].

Las redes de sensores (SN) son el componente esencial de IoT [51]. Debido a su importancia, en las siguientes secciones se estudian muchos de los retos tecnológicos inherentes a IoT así como aquellos heredados de las redes de sensores.

2.2. El componente esencial de IoT: Redes de Sensores

Una red de sensores consta de uno o más nodos sensor que se comunican entre sí utilizando tecnologías cableadas e inalámbricas. En las SN, los sensores pueden ser homogéneos u heterogéneos. Además, es posible conectar varias SN a través de diferentes tecnologías y protocolos. Debido al ámbito de esta tesis, el enfoque de mayor interés consiste en conectar redes de sensores a través de Internet. El caso más significativo de las SN son las WSN.

Una WSN consiste en sensores autónomos distribuidos espacialmente para monitorizar condiciones físicas o ambientales (p.ej. temperatura, presión, movimiento o contaminantes) y para transmitir cooperativamente sus datos a través de la red (principalmente redes malladas inalámbricas de corto alcance y, a veces, cableadas o híbridas) a una ubicación principal [35]. Las WSN son utilizadas principalmente para realizar tareas de detección y recopilación de información. Además, existen otras redes diseñadas para diversos propósitos como las redes de sensores corporales (BSN), redes de sensores de vídeo o visión (VSN), redes de sensores vehiculares (V2V, V2I), redes de sensores sociales/ participativas, redes de sensores interplanetarias, redes de bus de campo (clasificadas como sistemas SCADA) y otras.

Con el desarrollo de las WSN, los recientes avances tecnológicos han llevado a la aparición de redes de sensores y actuadores inalámbricas distribuidas (WSAN) que son capaces de observar el mundo físico, procesar los datos, tomar decisiones basadas en las observaciones y realizar acciones apropiadas. Estas redes pueden ser una parte integral de sistemas tales como la detección de ataques nucleares, biológicos y químicos, el control del microclima en edificios, la automatización del hogar o la monitorización ambiental.

Una WSN está compuesta por nodos (de unos pocos a varios cientos o incluso miles), donde cada nodo está conectado a uno o varios sensores. Típicamente, en un nodo se pueden diferenciar los siguientes componentes: un transceptor radio con una antena, un microcontrolador, un circuito electrónico para la interfaz con los sensores y una fuente de energía, normalmente una batería y/o un sistema embebido de captación de energía. La topología de las WSNs puede variar desde una simple red en estrella hasta una red avanzada de malla multisalto con uno o varios nodos sensores denominados sumideros (*sinks*) que actúan como *gateways* conectados con un servidor central remoto. En una red WSN mallada, el encaminamiento es necesario para una transmisión de datos confiable. Los protocolos de encaminamiento son distribuidos y reactivos: los nodos en el sistema comienzan a buscar una ruta sólo cuando tienen datos de aplicación que transmitir. Algoritmos de enrutamiento de uso frecuente son *Ad hoc On-Demand Distance Vector* (AODV) y *Dynamic Source Routing* (DSR). WSN es actualmente un área de investigación activa para usos críticos de misión limitada. Gigantes como IBM o Microsoft han invertido en la investigación de WSN durante mucho tiempo con poco éxito comercial. Actualmente no existe una plataforma WSN común. Algunos diseños como Berkeley Motes[3] y sus clones tienen comunidades más amplias de usuarios y desarrolladores. Sin embargo, muchos laboratorios de investigación y empresas comerciales prefieren desarrollar y producir sus propios dispositivos. Puesto que no existe una implementación determinante para WSN que reduzca los costes, suele ser más conveniente e incluso menos costoso construir sus propios dispositivos de WSN que comprar aquellos disponibles comercialmente.

La Figura 2.2 ilustra la estructura en capas en la que se aprecian los componentes más comunes de una red de sensores. Las flechas de color naranja indican que la mayor parte de los datos fluyen de derecha a izquierda [51].

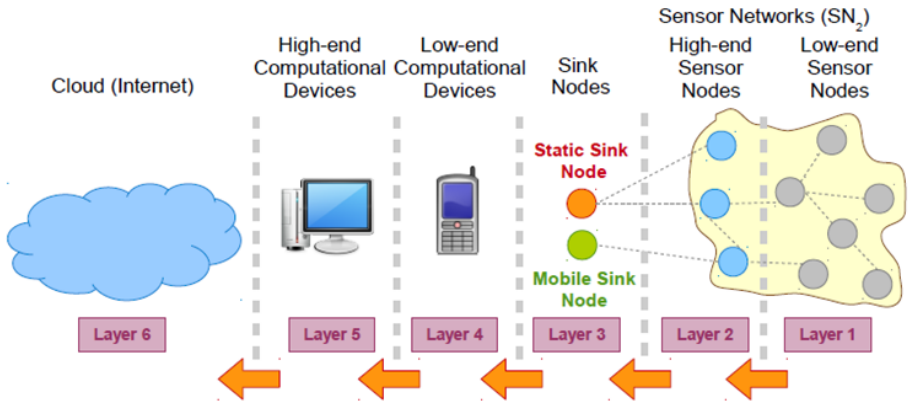


Figura 2.2. Estructura típica de una red de sensores [51].

Los datos son generados por los nodos sensor de gamas baja y alta para, posteriormente, ser recopilados mediante nodos móviles y estáticos tipo sumidero. Los nodos sumidero transmiten los datos a dispositivos computacionales de gama baja que realizan tareas de procesamiento. Una vez procesados, los datos se transmiten a dispositivos computacionales

[3] http://smote.cs.berkeley.edu/motescope/

de gama alta para un mayor procesado. Finalmente, llegan a la nube donde son procesados intensivamente, almacenados y compartidos.

En función de las capacidades de los dispositivos involucrados en una red de sensores, se han identificado seis capas. En IoT, puede existir un número adicional de sub-capas ya que se espera que comprenda una gran variedad de capacidades de detección. La información se puede procesar en cualquier capa. En esta descripción, capacidad significa la capacidad de procesamiento, memoria, comunicación y energía. Las capacidades aumentan de la capa uno a la capa seis. Partiendo de esta identificación de capas, es evidente que un sistema ideal debe entender las diferencias de capacidad y realizar la gestión de datos en consecuencia. Se trata de cuestiones de eficiencia y eficacia. Por ejemplo, realizar el procesamiento en las primeras capas podría reducir la comunicación de datos. Sin embargo, los dispositivos de las primeras capas no tienen suficiente de energía y potencia de procesamiento para realizar un procesamiento de datos exhaustivo [52]. La investigación en IoT necesita encontrar formas más eficientes y eficaces de gestión de datos, tales como recopilación, modelado, razonamiento y distribución.

Principalmente, existen tres arquitecturas de SN: arquitectura plana (transferencias de datos desde nodos sensores estáticos hasta el nodo sumidero de forma multisalto), arquitectura de dos capas (se utilizan nodos sumidero estáticos y móviles para recopilar datos de nodos sensores) y arquitectura de tres capas (múltiples redes de sensores conectadas entre sí a través de Internet). El despliegue de una SN a ultra gran escala que sigue esta arquitectura (Figura 2.3) es una red de sensores ubicua (USN) [53]. Esta USN "invisible", "pervasiva" o "de inteligencia ambiental" es Internet de las Cosas. Esta arquitectura es estudiada a nivel físico, funcional y lógico en las siguientes secciones.

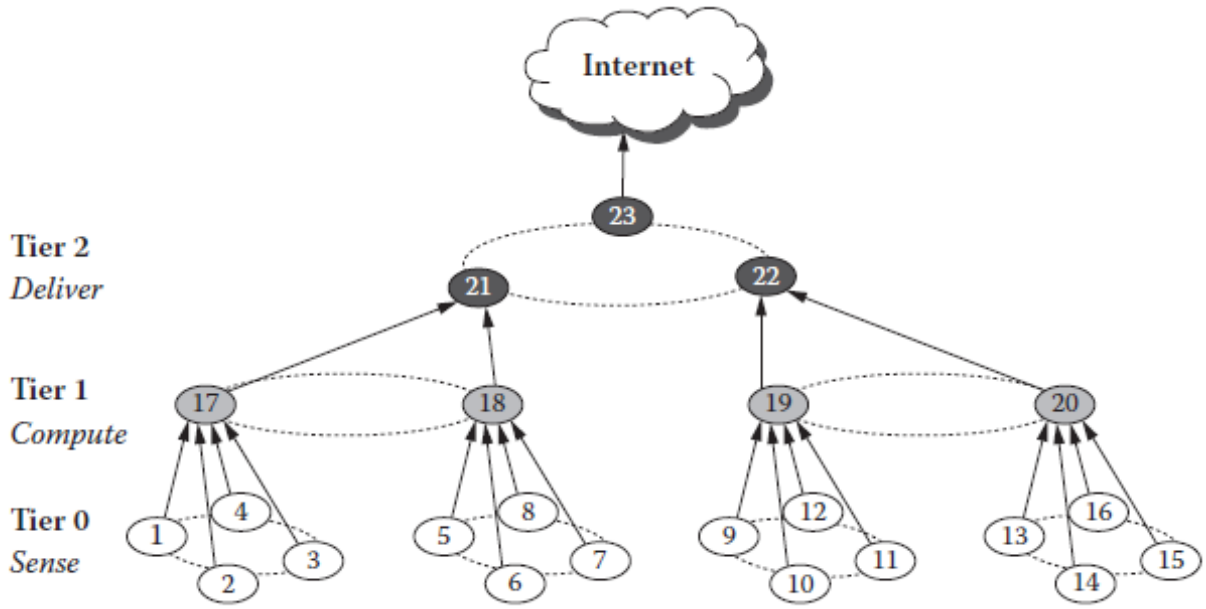


Figura 2.3. Arquitectura de tres capas de una red de sensores.

2.3. Diseño físico de IoT

Internet de las Cosas es la visión de miles de millones de "cosas" inteligentes que están conectadas formando una especie de "red neuronal global universal" [54] en la nube. Esta "red" abarcará cada aspecto de nuestras vidas, siendo su base la inteligencia proporcionada por el procesamiento embebido. Así, IoT se compone de "cosas" (dispositivos, máquinas u objetos) dotadas de inteligencia que interactúan y se comunican con otras "cosas" e infraestructuras. Como resultado, se generan enormes volúmenes de datos que son procesados por "aplicaciones" de diversos ámbitos para extraer conocimiento o controlar las cosas con el fin de hacer nuestras vidas mucho más fáciles y seguras así como reducir nuestro impacto en el medio ambiente.

A partir de esta visión, es posible deducir que todos los sistemas IoT tienen tres capas: dispositivos, redes y aplicaciones (modelos DNA o DCM) [35] [55]. Esta clasificación permite describir el diseño físico o cadena de valor de IoT y, al mismo tiempo, los desafíos que surgen en cada una de sus capas. Comenzamos por estudiar los dispositivos y los avances tecnológicos producidos como resultado de la investigación científica en esta capa de IoT.

2.3.1. ***Los dispositivos***

Los dispositivos pueden clasificarse en dos grupos: aquellos que tienen inteligencia inherente como los contadores eléctricos o de calefacción, ventilación y aire acondicionado (HVAC) y aquellos que son inertes y deben activarse para convertirse en dispositivos inteligentes (p.ej. etiquetado con RFID) como muebles o animales que pueden ser monitorizados y rastreados electrónicamente.

Aquellos dispositivos que realizan una función de entrada se denominan sensores porque detectan un cambio físico en alguna característica que cambia en respuesta a alguna excitación, por ejemplo, calor o fuerza, y lo convierten en una señal eléctrica. Los dispositivos que realizan una función de salida se denominan generalmente actuadores y se utilizan para controlar algún dispositivo externo. Tanto sensores como actuadores son conocidos colectivamente como transductores porque se utilizan para convertir energía de un tipo en otro. Por ejemplo, un micrófono (dispositivo de entrada) convierte ondas de sonido en señales eléctricas mientras que un altavoz (dispositivo de salida) realiza esta conversión de manera inversa.

Un sensor (Figura 2.4) es un dispositivo que responde a un estímulo físico, mide la cantidad del estímulo físico y lo convierte en una señal, generalmente eléctrica, que puede ser interpretada por un observador o por un instrumento. Basándose en esta definición, un sensor es básicamente un dispositivo eléctrico. Los sensores son particularmente útiles para tomar medidas in situ, como en algunas aplicaciones agrícolas. Un sensor puede ser muy pequeño y, al mismo tiempo, permitir su identificación y localización; sin embargo, cuando cualquier cosa es instrumentada con un sensor, esta también se convierte en un dispositivo identificable y localizable.

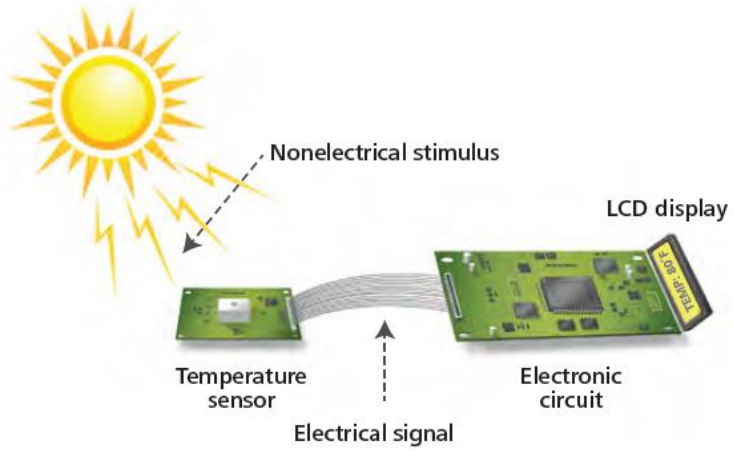


Figura 2.4. Sensor de temperatura [56].

Las "cosas" en IoT suelen referirse a los dispositivos que tienen identidades únicas y pueden realizar funciones remotas de monitorización, detección y/o actuación. Así, los dispositivos de IoT pueden intercambiar información con otros dispositivos y aplicaciones conectadas (directa o indirectamente). Además, permiten recopilar datos de otros dispositivos y procesarlos localmente o enviarlos a servidores centralizados o *backends* de aplicaciones basados en la nube para su procesamiento o realizar algunas tareas localmente y otras dentro de la infraestructura IoT, en función de limitaciones espaciales y temporales (es decir, memoria, capacidades de procesamiento, velocidades y latencias de comunicación, etc.).

En términos generales, un dispositivo IoT tiene las siguientes características:

- Dispone de varias interfaces para las conexiones con otros dispositivos. Estas interfaces permiten que la comunicación pueda ser realizada mediante tecnologías cableadas o inalámbricas e incluyen: (i) interfaces de E/S para sensores, (ii) interfaces para conectividad a Internet, (iii) interfaces de memoria y almacenamiento e (iv) interfaces de audio/ video.
- Permite recopilar diversos tipos de datos de los sensores integrados o conectados, como la temperatura, humedad o la intensidad de la luz. Los datos detectados pueden ser trasmitidos a otros dispositivos o a servidores/ almacenamiento basado en la nube.
- Suele generar datos que, una vez procesados, conducen a información útil para orientar las acciones futuras local o remotamente.
- Permite la conexión con actuadores para interactuar con otras entidades físicas (incluyendo dispositivos y sistemas que no forman parte de IoT) próximas al dispositivo.
- Puede ser de muchos tipos, por ejemplo, *wearable*, dispositivo móvil, máquina industrial, etc.

La última característica citada es reveladora. En la actualidad existe una amplia variedad de dispositivos IoT (Figura 2.1) y otros muchos se encuentran en fase de investigación, diseño o desarrollo. Además, un gran número irá apareciendo debido a los requisitos de las aplicaciones futuras. Por ello, se prevé una profunda heterogeneidad en los dispositivos conectados a la IoT. Además, la inmensa mayoría de los dispositivos inteligentes dispone de recursos computacionales insuficientes para implementar la pila de protocolos IP y esta

situación se mantendrá en el futuro para muchos de los dispositivos empleados en las aplicaciones IoT [15]. Este es el caso de las WSN, que se componen de dispositivos destinados a ser desplegados en grandes cantidades y múltiples entornos, incluyendo regiones remotas. Por estas razones, la maximización del tiempo de vida, el consumo de energía y el coste de despliegue son algunos de los desafíos que plantean este tipo de redes. De acuerdo con el pronóstico de [57], el precio promedio por nodo caerá de 9$ por pieza en 2011 a 5$ por pieza en 2021. Respecto a la energía, es el recurso más escaso de los nodos WSN, siendo determinante en la vida útil de la red. Afortunadamente, se están logrando importantes avances para poder superar estos retos. Prueba de ello, es la investigación en sistemas microelectromecánicos y técnicas de producción y almacenamiento de energía.

2.3.1.1. **Sistemas Microelectromecánicos**

Los sistemas microelectromecánicos (MEMS) se refieren a la tecnología de los dispositivos microscópicos, particularmente aquellos con partes móviles. En general, esta tecnología puede definirse como elementos mecánicos y electromecánicos miniaturizados (es decir, dispositivos y estructuras) que son producidos al utilizar técnicas de microfabricación [58]. Las dimensiones físicas críticas de los dispositivos MEMS pueden variar desde muy por debajo de una micra hasta varios milímetros. MEMS se fusiona a escala nanométrica en sistemas nanoelectromecánicos (NEMS) y nanotecnología.

Los tipos de dispositivo MEMS pueden variar desde estructuras relativamente simples que no tienen elementos móviles hasta sistemas electromecánicos extremadamente complejos con múltiples elementos móviles bajo el control de microelectrónica integrada. Los elementos funcionales de los MEMS son las estructuras miniaturizadas, sensores, actuadores y microelectrónica.

Gracias a la fabricación de microsensores y microactuadores MEMS, la tecnología de sensores ha hecho progresos significativos. Concretamente, en las últimas décadas, los investigadores de MEMS han sido capaces de un fabricar un gran número de microsensores para casi todas las posibles modalidades de detección. Se ha probado que el rendimiento de muchos de ellos es mayor que el de sus homólogos macroscópicos. No sólo el rendimiento de los dispositivos MEMS es excepcional, sino que su método de producción aprovecha las mismas técnicas de fabricación de lotes utilizadas en la industria de circuitos integrados. Este hecho puede traducirse en bajos costos de producción por dispositivo así como en muchos otros beneficios. En consecuencia, no sólo es posible lograr un rendimiento estelar del dispositivo, sino hacerlo a un nivel de coste relativamente bajo.

A medida que avanzan los métodos de fabricación de MEMS, la promesa es una enorme libertad de diseño donde es posible integrar microsensores y microactuadores con microelectrónica, fotónica, nanotecnología y otras tecnologías en un único microchip. Se espera que esta visión sea uno de los avances tecnológicos más importantes del futuro. Esto permitirá el desarrollo de productos inteligentes mediante el aumento de la capacidad computacional de la microelectrónica con las capacidades de detección y control de microsensores y microactuadores.

2.3.1.2. **Producción y almacenamiento de energía**

La Producción y almacenamiento de energía son relevantes para IoT por dos razones. En primer lugar, se relaciona con el interés global de garantizar la disponibilidad de electricidad a la vez que se reducen los impactos climáticos y ambientales. Por ejemplo, las redes eléctricas inteligentes implican microgeneración de electricidad utilizando paneles fotovoltaicos. Además, estas redes también requieren nuevos tipos de almacenamiento de energía, tanto para la propia red como para tecnologías emergentes como los vehículos eléctricos que dependen de tecnologías de baterías cada vez más eficientes. En segundo lugar, hay un interés acelerado en la comunidad de tecnologías de la información y las comunicaciones para impulsar las redes de sensores desplegadas de forma ubicua, electrónica móvil, etc. Muchas cosas llegarán a ser posibles a medida que se produzcan avances en las diferentes técnicas de recolección de energía así como en la investigación y desarrollo de nuevas tecnologías de baterías miniaturizadas y ultra condensadores.

La recolección de energía es una técnica que captura la energía ambiente desaprovechada (p.ej. energía solar, eólica, térmica, etc.) y la convierte en energía eléctrica aprovechable que es almacenada y utilizada para realizar tareas de detección o actuación [59] [60] [61]. La energía almacenada es generalmente muy pequeña (del orden de milijulios, mJ) en comparación con la recolección de energía a gran escala al emplear fuentes de energía renovables tales como granjas solares y parques eólicos. A diferencia de las centrales eléctricas a gran escala que se fijan en un lugar determinado, las fuentes de energía a pequeña escala son portátiles y fácilmente disponibles para su uso. Así, la energía recolectada del ambiente se utiliza para suministrar energía a pequeños sensores autónomos que están desplegados en localizaciones remotas de forma que puedan realizar tareas de detección y, en algunos casos, soporta una exposición a largo plazo en ambientes hostiles. La operación de estos pequeños sensores autónomos a menudo está restringida por la dependencia de la energía de la batería.

En resumen, la investigación y desarrollo de técnicas de recolección de energía permitirá un suministro de energía adicional a redes de sensores inalámbricas y dispositivos móviles que se traducirá en una operación extendida con el suplemento de elementos de almacenamiento de energía. Como resultado, IoT será aplicable en una amplia gama de escenarios que requieran un suministro de energía de larga duración.

*2.3.2. **Las redes***

La infraestructura base de la IoT está soportada tanto por tecnologías inalámbricas como cableadas. Aunque existe una gran heterogeneidad en ambas, los mayores desafíos se presentan en las tecnologías inalámbricas. Estas pueden ser clasificadas atendiendo a parámetros como alcance o topología, entre otros. En el primer caso, se distinguen los siguientes tipos de redes:

- Corto alcance (incluyendo redes de área personal (PAN), local (LAN) y metropolitana (MAN): Bluetooth [62], redes malladas (MN), RFID, Wi-Fi [63], etc.

- Largo alcance (vía redes celulares, redes de área amplia (WAN) y pseudo-largo-alcance): sistema global para las comunicaciones móviles (GSM), WCDMA o combinación de la norma de interoperabilidad mundial para el acceso por microondas (WiMax) con métodos de acceso múltiple por división de código (CDMA) y otras redes, así como comunicación por satélite.

Las redes inalámbricas también pueden clasificarse por su topología o forma en la que los nodos de la red están organizados y conectados entre sí. Las principales topologías de red son las de estrella (*star*) y malla (*mesh*), como se muestra en la Figura 2.5. En una topología en estrella, todos los nodos están conectados a un nodo central, que suele realizar la función de *gateway*. Un ejemplo de esta topología es una red Wi-Fi, donde el nodo central es denominado punto de acceso (AP) mientras que el resto de nodos reciben el nombre de estaciones.

En una red de malla, cada nodo puede conectarse a múltiples nodos. Además, uno o más nodos en la red realizan funciones de *gateway*. En la Figura 2.5, cada nodo de la red está conectado al resto. El despliegue de una red de malla es más simple que el de una red de estrella. Un ejemplo de esta topología es una red ZigBee [64] donde múltiples nodos se conectan entre sí para extender el alcance de la red. En estas redes, uno o más nodos ZigBee denominados coordinadores permiten el acceso a Internet.

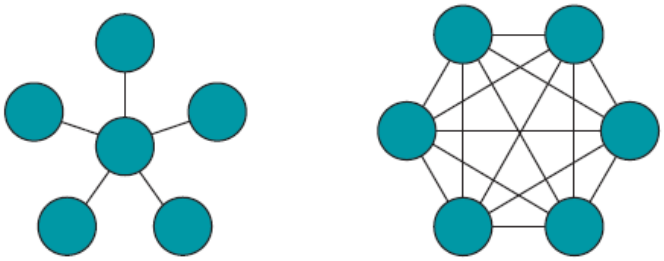


Figura 2.5. Topología de estrella (izquierda) y de malla (derecha) [65].

La elección de una topología de red determina una serie de ventajas y desventajas. En el caso de las redes de malla, las mayores desventajas son: (i) diseño más complejo y (ii) en comparación con las redes de estrella, se dan mayores retardos debido al encaminamiento de cada mensaje desde los nodos remotos hasta el coordinador de la red de malla. Respecto a las ventajas de las topologías de malla, cabe señalar que: (i) pueden extender el alcance de la red a través de múltiples saltos, manteniendo una baja potencia de transmisión radio y (ii) pueden alcanzar una mayor fiabilidad permitiendo la retransmisión de un mensaje a través de más de una ruta de la red.

El tamaño de la red o el número máximo de dispositivos conectados simultáneamente es también una consideración importante. En el diseño del sistema, algunas tecnologías como Bluetooth admiten hasta 20 conexiones mientras que otras como ZigBee, pueden soportar miles de conexiones.

La topología de las redes de sensores inalámbricas es dinámica y desconocida, debido al número variable de dispositivos que están conectados en un momento dado. Las WSN suelen utilizar redes malladas inalámbricas de corto alcance como técnicas de comunicación

fundamentales (Zigbee, DASH7 [66], Z-Wave [67], 6LowPAN [68], etc). Algunas de estas tecnologías de conectividad inalámbrica son estudiadas con más detalle en el apartado 2.3.2.1. En el caso de las WSN, los nodos sensores actuales están basados principalmente en el estándar IEEE 802.15.4 [69]. Se trata de un sistema radio de potencia y alcance bajos que cubre las capas física y de acceso al medio. Por este motivo, las conexiones al borde de la red (*edge*) son de baja fidelidad, baja velocidad y con pérdidas de datos como consecuencia de la atenuación e interferencia. Para la comunicación con la Internet global, algunos dispositivos implementan la pila de protocolos IP y utilizan redes celulares de largo alcance para realizar funciones de *gateway*. Sin embargo, la mayoría de dispositivos utiliza protocolos de optimización de energía (no IP) como los mencionados (p.ej. *Zigbee*) y sólo se comunican dentro de su red local. Debido a la utilización de protocolos de optimización de energía y a la heterogeneidad de soluciones, es necesario lograr la interoperabilidad para permitir la comunicación entre aplicación de alto nivel, servicios y red subyacente. El objetivo principal de muchas organizaciones de estándares es superar este desafío mediante la definición de especificaciones y procedimientos de prueba. Estos trabajos de investigación conducen al desarrollo de los protocolos de comunicaciones de IoT (apartado 2.3.2.2).

En el ámbito del largo alcance, cientos de satélites de comunicación comerciales están operando alrededor del mundo. Estos satélites permiten comunicaciones a través de redes de área amplia y se utilizan para diversos fines (p.ej. dispositivos agrícolas en lugares remotos). Hay dos cuestiones acerca de las comunicaciones por satélite: velocidad y coste. A pesar de que los satélites pueden transmitir grandes cantidades de datos, como TV en directo, esto se realiza en un solo sentido y para una antena de grandes dimensiones. La transmisión bidireccional para una antena pequeña tiene menor capacidad de ancho de banda que la comunicación celular/móvil. Además, el coste por *byte* de la comunicación vía satélite es mucho mayor. Pero si la aplicación en cuestión tiene requerimientos de datos pequeños y grandes necesidades de cobertura, el precio de la comunicación por satélite puede ser muy competitivo. Otra opción sería la utilización de dispositivos de modo dual. Esta última, combina tecnologías satelitales y celulares en un simple dispositivo, aportando lo mejor de ambos mundos.

A partir de estas características, se deduce que las necesidades de datos de Internet de las Cosas son completamente diferentes del Internet actual. En IoT, la mayoría de las comunicaciones serán breves intercambios asimétricos máquina a máquina, con muchos más datos fluyendo en una dirección (p.ej. desde el sensor hacia el servidor) que en la otra. En la mayoría de los casos, la pérdida de mensajes individuales será asumible. Finalmente, y lo más importante, las relaciones *peer to peer* IP tradicionales bloquean mucha de la riqueza potencial del Internet de las Cosas. Habrá grandes flujos de datos corriendo, muchos de los cuales serán desconocidos y sin planificar. Solo una arquitectura *publish/subscribe* [70] permite aprovechar este conocimiento mediante el descubrimiento de relaciones y flujos de datos interesantes. Y sólo una red *publish/subscribe* puede escalar al enorme tamaño del Internet de las Cosas [15].

2.3.2.1. **Tecnologías de conectividad inalámbrica**

En la sección anterior, se han revisado determinados conceptos clave y alternativas de diseño de las tecnologías de conectividad inalámbrica. A continuación, se estudian algunas de las más utilizadas en IoT [65].

Wi-Fi

La tecnología Wi-Fi está basada en IEEE 802.11, una colección de estándares de comunicación WLAN. Wi-Fi se ha convertido en una tecnología ubicua ya que está integrada en todos los productos electrónicos de consumo que salen al mercado y ha sido desplegada intensivamente por todo el mundo. Por lo tanto, su siguiente paso natural es conectar la nueva era de las cosas a Internet.

Las redes Wi-Fi tienen una topología en estrella, siendo el punto de acceso (AP) el *gateway* conectado a Internet. Un AP Wi-Fi suele soportar hasta 250 dispositivos conectados simultáneamente. La mayoría de estas redes operan en la banda de 2,4 GHz. Además, Wi-Fi puede operar en la banda de 5 GHz donde existen más canales y mayores velocidades de transferencia de datos disponibles. La integración de Wi-Fi y la pila de protocolos TCP/IP en dispositivos con potentes microprocesadores (MPU) y grandes cantidades de memoria no supone ningún problema. Sin embargo, esta afirmación no se cumple para todos los dispositivos limitados. Actualmente, algunos ya integran estas tecnologías de fábrica. Estos nuevos dispositivos eliminan la mayor parte de la sobrecarga del MPU y permiten la conectividad inalámbrica a Internet con el microcontrolador más pequeño (MCU). El creciente nivel de integración en estos dispositivos también elimina toda la experiencia de diseño de radio requerida y reduce las barreras de la integración Wi-Fi.

Para permitir altas velocidades de transferencia de datos (en algunos casos superiores a 100 Mbps) y buena cobertura en interiores de edificios, las radios Wi-Fi tienen un consumo de energía bastante grande. Para algunos dispositivos IoT, que funcionan con baterías y no se pueden cargar frecuentemente, Wi-Fi puede requerir demasiada energía. Aunque la corriente máxima de las radios Wi-Fi no puede reducirse mucho, los nuevos dispositivos de silicio implementan protocolos de hibernación avanzados y tiempo rápido de encendido/apagado para reducir drásticamente el consumo de potencia promedio. Dado que la mayoría de los productos IoT no necesitan las máximas velocidades de transferencia de datos ofrecidas por Wi-Fi, un diseño inteligente de la gestión de la potencia puede generar de forma eficiente ráfagas de corriente desde la batería durante intervalos muy cortos y mantener los productos conectados a Internet durante más de un año utilizando dos baterías alcalinas AA. En resumen, en la actualidad, Wi-Fi es la tecnología más ubicua en la conectividad inalámbrica a Internet. Su complejidad y alta potencia han sido grandes barreras para los desarrolladores de IoT, pero la aparición de nuevos dispositivos permite la integración Wi-Fi en aplicaciones emergentes de IoT y dispositivos alimentados mediante batería.

Bluetooth

Bluetooth es una tecnología WPAN cuya capa de enlace opera en la banda de 2,4 GHz. Actualmente, es utilizada principalmente como alternativa de la comunicación cableada en

aplicaciones de corto alcance. Hoy en día, todos los teléfonos móviles tienen conectividad Bluetooth. Además, se han desarrollado multitud de casos de uso como la transmisión de música de alta fidelidad y la evolución de accesorios relacionados con el bienestar y la salud.

Bluetooth soporta un *throughput* de datos de hasta 2 Mbps y es desplegado principalmente en una topología de red punto a punto o de estrella. Además, se trata de una tecnología de baja potencia. Este hecho implica que los dispositivos utilicen pequeñas baterías recargables o dos baterías alcalinas.

Bluetooth Low Energy

Bluetooth *Low Energy* (BLE) es el estándar de baja energía de Bluetooth. Diseñado para un *throughput* de datos más bajo, BLE reduce significativamente el consumo de energía de los dispositivos Bluetooth y permite años de operación utilizando baterías de tipo botón.

Mientras que Bluetooth puede soportar hasta ocho dispositivos conectados simultáneamente en una red en estrella, BLE elimina esta limitación y teóricamente puede soportar un número ilimitado. Sin embargo, el número práctico de dispositivos conectados simultáneamente es entre 10 y 20. BLE facilita una amplia gama de nuevas aplicaciones al permitir la conexión en los últimos 10 metros entre los accesorios inalámbricos y un dispositivo que actúa como *gateway* de Internet.

ZigBee

Basado en el estándar IEEE 802.15.4, ZigBee es una tecnología de bajo *throughput*, baja potencia y bajo coste. Aunque opera principalmente en la banda de 2,4 GHz, ZigBee también soporta las bandas de 868 MHz y 915 MHz. Esta tecnología puede ofrecer un *throughput* de datos de hasta 250 Kbps pero se suele utilizar en aplicaciones que requieren velocidades de transferencia de datos mucho más bajas. Respecto a la potencia, Zigbee tiene la capacidad de mantener intervalos de hibernación muy largos y ciclos de funcionamiento bajos. Estas características permiten una autonomía de larga duración (superior a 10 años) mediante el suministro de energía basado en baterías de tipo botón. Los nuevos dispositivos ZigBee incluso pueden implementar técnicas de producción y almacenamiento de energía que les permiten operar sin baterías (véase apartado 2.3.1.2).

Otra característica del estándar ZigBee es su topología de red de malla que puede incluir miles de nodos, donde los datos saltan de nodo a nodo en múltiples direcciones y rutas a través de redes a gran escala. Para conectarse a IoT, las redes ZigBee requieren un *gateway* a nivel de aplicación. Este *gateway* participa como uno de los nodos de la red y, en paralelo, ejecuta una pila TCP/IP y una aplicación sobre Ethernet o Wi-Fi para conectar la red ZigBee a Internet.

6LoWPAN

6LoWPAN es el acrónimo para IPv6 sobre WPAN de baja potencia. Se trata del primer estándar de conectividad inalámbrica que fue creado para IoT. El término “red de área personal” que forma parte del acrónimo 6LoWPAN puede ser confuso ya que 6LoWPAN suele ser utilizado para formar LAN. El objetivo de 6LoWPAN es aplicar IP al dispositivo más pequeño, de menor potencia y de capacidad de procesamiento más limitada.

IPv6 fue elegido como el protocolo de Internet de 6LoWPAN (excluyendo IPv4) porque soporta un mayor espacio de direccionamiento y porque incluye soporte para la configuración automática de la red. Las redes 6LoWPAN requieren un *gateway* Ethernet o Wi-Fi para acceder a Internet. De igual forma que Wi-Fi, el *gateway* es un *gateway* de capa IP y no un *gateway* de capa de aplicación, que permite a los nodos y aplicaciones 6LoWPAN un acceso directo a Internet. Dado que la mayor parte del despliegue de Internet sigue utilizando IPv4, un *gateway* 6LoWPAN normalmente incluye un protocolo de conversión IPv6 a IPv4.

Los despliegues iniciales de 6LoWPAN utilizan las bandas de 2,4 GHz y 868 MHz/915 MHz. Basándose en las ventajas de 802.15.4 (topología de red de malla, tamaño de red grande, comunicación fiable y bajo consumo de energía) y en los beneficios de la comunicación IP, 6LoWPAN está bien posicionado para alimentar el mercado en explosión de los sensores conectados a Internet y otras aplicaciones caracterizadas por un *throughput* de datos bajo y una alimentación por batería.

Protocolos propietarios y transceptores radio (Sub-1GHz)

Actualmente, muchas aplicaciones industriales utilizan protocolos propietarios que corren sobre los transceptores de radio. El transceptor de radio proporciona la capa de enlace de la red o sólo la capa física. El resto del protocolo de red es implementado por el fabricante. Los sistemas diseñados de este modo permiten una mayor flexibilidad a expensas de la interoperabilidad y el esfuerzo de desarrollo. Estos sistemas de radio patentados utilizan principalmente las bandas de frecuencia inferiores a 433 MHz, 868 MHz y 915 MHz. Por este motivo, se denominan comúnmente soluciones Sub-1 GHz. Dichas soluciones suelen transmitir alta potencia, pudiendo alcanzar más de 25 km con una topología punto a punto o de estrella. Para conectarse al IoT, los sistemas Sub-1 GHz necesitan un *gateway* de capa de aplicación.

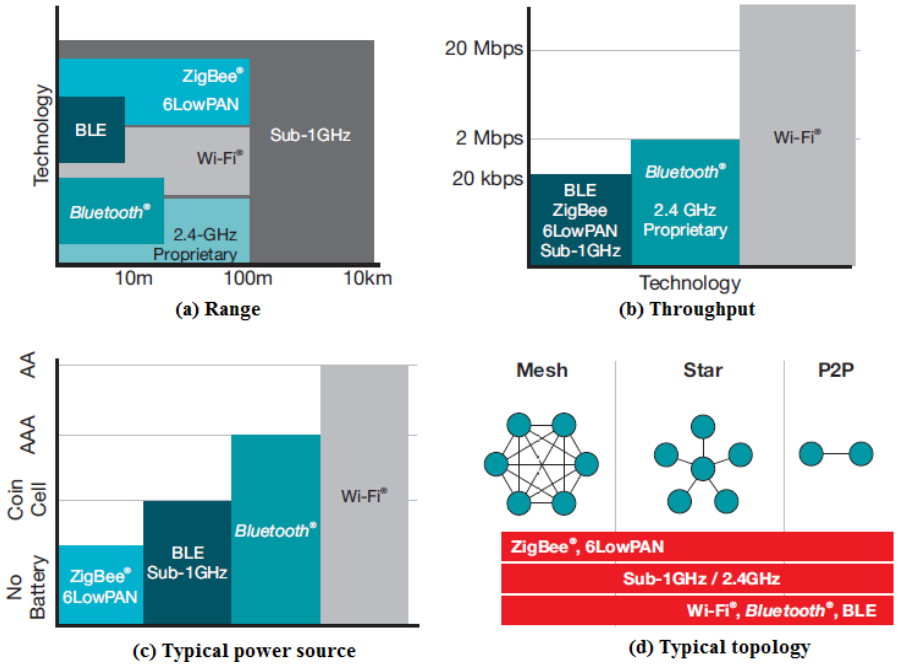


Figura 2.6. Resumen de tecnologías de conectividad inalámbrica [65].

La Figura 2.6 ilustra un resumen de las tecnologías de conectividad inalámbrica estudiadas. Para ello, muestra una comparación en función de 4 parámetros que son determinantes en la elección de una tecnología, a saber: (a) alcance, (b) *throughput*, (c) fuente de potencia y (d) topología.

2.3.2.2. **Protocolos IoT**

Los sistemas de comunicación utilizan un conjunto de reglas y estándares para dar formato y controlar el intercambio de los datos. El modelo más común en sistemas de comunicación de datos se denomina interconexión de sistemas abiertos (OSI). Este modelo divide la comunicación en capas funcionales para facilitar la implementación de redes escalables e interoperables. Aunque el modelo OSI tiene 7 capas, se ha optado por utilizar una versión simplificada de 4 capas para clasificar algunos de los protocolos de Internet de las Cosas más utilizados [49].

Capa de enlace (Link Layer)

Los protocolos de capa de enlace determinan la forma en la que los datos son enviados por la capa o medio físico de la red (p. ej.: cable coaxial, onda de radio, etc). El alcance de esta capa es la conexión de red local a la que está conectado el dispositivo. La capa de enlace determina la codificación y señalización de los paquetes a realizar por el dispositivo en función del medio al que esté conectado. A continuación, se estudian varios protocolos de capa de enlace relevantes en el contexto de IoT.

- 802.3-Ethernet: IEEE 802.3 es una colección de estándares para redes basadas en Ethernet, incluyendo las especificaciones del medio físico subyacente. Por ejemplo, 802.3 es el estándar para Ethernet 10BASE5 que usa cable coaxial como medio compartido, 802.3.i es el estándar para Ethernet 10BASE-T sobre conexiones de cobre par trenzado, 802.3.j es el estándar para Ethernet 10BASE-F sobre conexiones de fibra óptica, 802.3ae es el estándar para Ethernet de 10Gbit/ s sobre fibra, y así sucesivamente. Estos estándares proporcionan velocidades de transferencia de datos desde 10 Mb/s hasta 40 Gb/s y superiores. Las especificaciones están disponibles en el sitio web del grupo de trabajo IEEE 802.3 [71].
- 802.11-Wi-fi: IEEE 802.11 es una colección de estándares de comunicación de red de área local inalámbrica, incluyendo una extensa descripción de la capa de enlace. Por ejemplo, 802.11a opera en la banda de 5 GHz, 802.11b y 802.11g operan en la banda de 2,4 GHz, 802.11n opera en las bandas de 2,4/5 GHz, 802.11ac opera en la banda de 5 GHz y 802.11ad opera en la banda de 60 GHz. Estos estándares proporcionan velocidades de transferencia de datos desde 1 Mb/s hasta 6,75 Gb/s. Las especificaciones están disponibles en el sitio web del grupo de trabajo IEEE 802.11 [72].
- 802.16-WiMax: IEEE 802.16 es una colección de estándares inalámbricos de banda ancha, incluyendo extensas descripciones para la capa de enlace. Los estándares WiMax proporcionan velocidades de transferencia de datos desde 1,5 Mb/s hasta 1 Gb/s. La actualización reciente (802.16m) proporciona velocidades de 100 Mb/s para estaciones

móviles y 1 Gb/s para estaciones fijas. Las especificaciones están disponibles en el sitio web del grupo de trabajo IEEE 802.16 [73].

- 802.15.4-LR-WPAN: IEEE 802.15.4 es una colección de estándares para redes inalámbricas de área personal de baja velocidad de transmisión de datos (LR-WPAN). Estas normas forman la base de las especificaciones de protocolos de alto nivel como Zigbee. Las velocidades de LR-WPAN van de 40 Kb/s a 250 Kb/s. Estos estándares proporcionan comunicaciones de coste y velocidad bajos adecuadas para dispositivos con limitaciones de potencia. Las especificaciones están disponibles en el sitio web del grupo de trabajo IEEE 802.15 [74].
- 2G/3G/4G-Comunicación Móvil: existen diferentes generaciones de estándares de comunicación móvil: segunda generación (2G, incluyendo GSM y CDMA), tercera generación (3G, incluyendo UMTS y CDMA 2000) y cuarta generación (4G, incluyendo LTE). Los dispositivos IoT basados en estos estándares pueden comunicarse a través de redes celulares. Las velocidades de transmisión de datos para estos estándares van desde 9,6 Kb/s (para 2G) hasta 100 Mb/s (para 4G). Las especificaciones de estos estándares están disponibles en los sitios web de 3GPP.

Capa de red (Network Layer)

La capa de red es responsable del encaminamiento de los paquetes de datos y del direccionamiento del dispositivo. En la actualidad existen diversos mapas para identificar los dispositivos. A continuación, se describen los esquemas de direccionamiento IP.

- IPv4: es el protocolo de Internet más implementado. Se utiliza para identificar los dispositivos en una red utilizando un esquema de direccionamiento jerárquico. IPv4 utiliza un esquema de direcciones de 32 bits que permite un total de 2^{32} direcciones. Debido a las grandes demandas de conectividad a Internet, estas direcciones se agotaron en el año 2011. Por esta razón, IPv4 [75] fue sustituido por IPv6.
- IPv6: es la versión más reciente del protocolo de Internet y sucesor de IPv4. IPv6 utiliza esquema de direcciones de 128 bits que permite un total de 2^{128} direcciones. IPv6 se describe formalmente en el RFC 2460 [76].
- 6LowPAN: este estándar fue creado por el grupo de trabajo 6LoWPAN del IETF para definir una capa de adaptación eficiente entre la capa de enlace y la pila TCP/IP. Para lograrlo, 6LowPAN colabora con el protocolo de capa de enlace 802.15.4 y define mecanismos de compresión para datagramas IPv6 sobre redes basadas en IEEE 802.15.4. De esta forma, facilita la implementación del protocolo IP a los dispositivos de baja potencia que tienen capacidad de procesamiento limitada. Opera en el rango de frecuencias de 2,4 GHz y proporciona velocidades de 250 Kb/s.

Capa de Transporte (Transport Layer)

Los protocolos de capa de transporte proporcionan capacidad de transferencia de mensajes extremo a extremo independiente de la red subyacente. La capa de transporte ofrece funciones tales como control de errores, segmentación, control de flujo y control de congestión.

- TCP: el protocolo de control de transmisión (TCP) es el más utilizado por los navegadores web (junto con los protocolos de capa de aplicación HTTP, HTTPS), programas de correo electrónico (protocolo de capa de aplicación SMTP) y transferencia de archivos (FTP). TCP es un protocolo con estado y orientado a la conexión. Mientras que el protocolo IP se ocupa del envío de paquetes, TCP asegura una transmisión fiable y ordenada de los paquetes. TCP también proporciona la capacidad de detección de errores para que los paquetes duplicados puedan ser descartados y los paquetes perdidos sean retransmitidos. La capacidad de control de flujo de TCP asegura que la velocidad a la que el transmisor envía los datos no sea demasiado alta para que el receptor pueda procesarlos. La capacidad de control de congestión de TCP ayuda a evitar la congestión que puede conducir a la degradación del rendimiento de la red. TCP se describe en RFC 793 [77].
- UDP: a diferencia de TCP, que requiere establecer una conexión lógica antes del intercambio de datos, UDP es un protocolo sin conexión. UDP es útil para aplicaciones sensibles al tiempo que tienen unidades de datos muy pequeñas que intercambiar. En estas aplicaciones, la sobrecarga de configuración de la conexión no es necesaria. UDP es un protocolo orientado a la transacción y sin estado. Por ello, no proporciona entrega garantizada, orden de mensajes ni eliminación de duplicados. UDP se describe en RFC 768 [78].

Capa de Aplicación (Application Layer)

Los protocolos de capa de aplicación definen cómo interactúan las aplicaciones con los protocolos de capa inferior para enviar los datos a través de la red. Los datos de aplicación son codificados por el protocolo de capa de aplicación y encapsulados en el protocolo de capa de transporte. Para el direccionamiento de las aplicaciones se asignan identificadores de puerto, utilizados por los protocolos de capa de aplicación para las conexiones entre procesos.

- HTTP: el protocolo de transferencia de hipertexto es la base de la *World Wide Web* (WWW). HTTP sigue un modelo solicitud-respuesta en el que un cliente envía solicitudes a un servidor mediante comandos (GET, PUT, POST, DELETE, etc). Se trata de un protocolo sin estado. Así, cada solicitud HTTP es independiente del resto. Un cliente HTTP puede ser un navegador o una aplicación que se ejecuta en el cliente (p.ej., una aplicación que se ejecuta en un dispositivo IoT, una aplicación móvil u otro *software*). El protocolo HTTP utiliza identificadores universales de recurso (URI) y se describe en RFC 2616 [79].
- CoAP: es un protocolo para aplicaciones M2M destinado a entornos limitados (dispositivos y redes limitadas). De igual forma que HTTP, CoAP es un protocolo de transferencia Web y utiliza un modelo solicitud-respuesta. Sin embargo, se ejecuta en la parte superior de UDP en lugar de TCP. CoAP [80] está diseñado para interactuar fácilmente con HTTP.
- WebSocket: este protocolo permite la comunicación full-duplex a través de una única conexión de socket para el envío de mensajes entre cliente y servidor. WebSocket se basa en TCP y permite que los flujos de mensajes se envíen de un lado a otro entre el cliente y

el servidor mientras que se mantenga abierta la conexión. El cliente puede ser un navegador, una aplicación móvil o un dispositivo IoT. Se describe en RFC 6455 [81].

- MQTT: consiste en un protocolo ligero de mensajería basado en el modelo *publish*/*subscribe*. MQTT utiliza una arquitectura cliente-servidor donde el cliente (p.ej. dispositivo IoT) se conecta con el servidor o MQTT Broker) para publicar mensajes en los temas del servidor. El MQTT Broker reenvía los mensajes a los clientes suscritos a los temas. MQTT [82] es adecuado para entornos restringidos donde los dispositivos tienen recursos limitados de procesamiento y memoria y el ancho de banda de la red es bajo.
- XMPP: es un protocolo para la comunicación en tiempo real y el *streaming* de datos XML entre entidades de la red. XMPP permite enviar pequeños fragmentos de datos XML de una entidad de red a otra casi en tiempo real. Se trata de un protocolo descentralizado que utiliza una arquitectura cliente-servidor. Admite comunicaciones cliente a servidor y servidor a servidor. En el contexto de IoT, este protocolo permite la comunicación en tiempo real entre dispositivos IoT. XMPP se describe en RFC 6120 [83].
- DDS: es un estándar de *middleware* centrado en los datos para la comunicación M2M que utiliza un modelo *publish*/*subscribe*. DDS proporciona control de calidad de servicio (QoS) y confiabilidad configurable. Este estándar es descrito por el *Object Management Group* (OMG) [84].

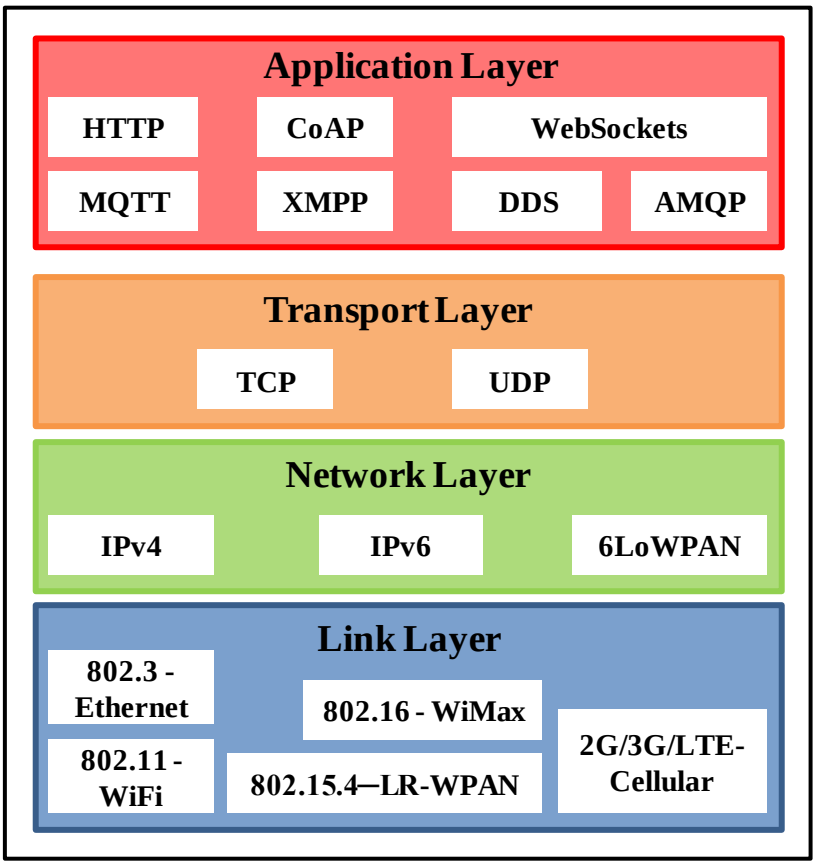


Figura 2.7. Clasificación de protocolos IoT según el modelo OSI simplificado.

- AMQP: es un protocolo de capa de aplicación abierto para mensajería empresarial. AMQP soporta modelos punto a punto y *publish*/*subscribe*, enrutamiento y colas. Los *brokers* AMQP reciben mensajes de *publishers* (p.ej., dispositivos o aplicaciones que generan datos) y los encaminan a través de las conexiones a los consumidores (aplicaciones que procesan datos). La especificación está disponible en el sitio web del grupo de trabajo AMQP [85].

La Figura 2.7 ilustra la disposición de los protocolos IoT introducidos en esta sección según el modelo OSI simplificado. Tal y como se mencionó en el apartado 1.2.2, Internet de las Cosas cubre un gran rango de dominios de aplicación. Este hecho implica la existencia de una enorme cantidad de estándares y protocolos IoT[4]. Su estudio queda fuera del ámbito de este trabajo de investigación, siendo suficiente la clasificación propuesta.

2.3.3. ***Las aplicaciones***

Anteriormente, se han descrito las capas de dispositivos y redes del diseño físico o modelos DNA y DCM de IoT (dispositivos, redes y aplicaciones) [35] [55]. Estas capas permiten el estudio de los procesos y lugares dónde la recopilación de la información es realizada por varios tipos de dispositivo, su posterior agregación mediante diferentes clases de *gateways* (provistos por nodos intermedios o de propagación) y su transporte a través de redes de acceso así como de la red troncal a los servidores centrales. La Figura 2.8 ilustra cómo las pequeñas cantidades de datos detectadas por varios dispositivos finales o sensores son recopiladas por un nodo intermedio donde se agrupan y se agrega información adicional de ubicación, direccionamiento, protocolo, etc. para su transporte a través de Internet.

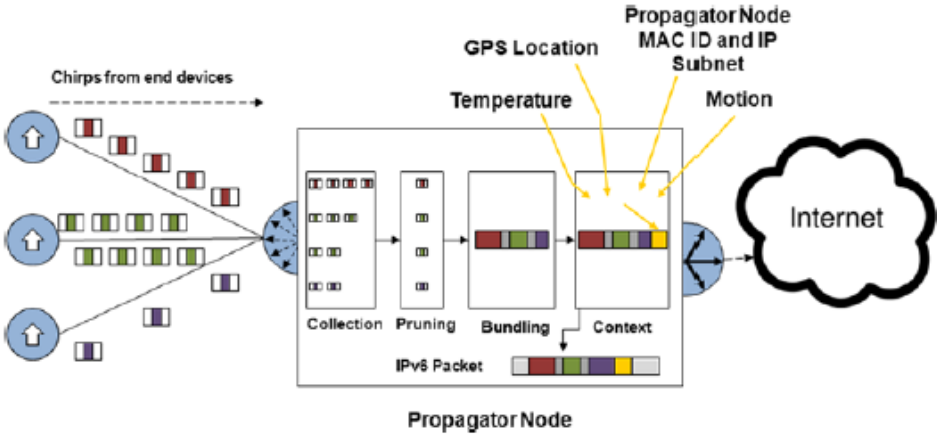


Figura 2.8. Recopilación, agregación y transporte de la información en IoT [15].

A nivel mundial, IoT será una red a ultra gran escala formada por miles de millones de nodos. Según Gartner [86], en 2020 habrá cerca de 26 mil millones de dispositivos conectados a IoT. De forma similar, la investigación ABI [87] estimó que más de 30 mil millones de dispositivos estarán conectados de forma inalámbrica en el mismo año. En una red de un tamaño sin precedentes, las interacciones espontáneas entre un número tan grande de cosas o dispositivos producirán una enorme cantidad de eventos de forma habitual. Esta característica

[4] Estándares y protocolos IoT: http://www.postscapes.com/internet-of-things-protocols/

nueva, intrínseca de IoT, no ha sido tenida en cuenta en el diseño de las arquitecturas de red tradicionales. Por tanto, dichas arquitecturas no son apropiadas para todas las aplicaciones potenciales de Internet de las Cosas. Para encontrar soluciones a estos desafíos, campos tan diversos como los de robótica, sistemas embebidos, Big Data o WSN contribuyen con conceptos y tecnología. Sin embargo, ninguno de ellos aborda directamente la escala y el alcance de Internet de las Cosas ni la sencillez de la gran mayoría de los dispositivos finales IoT.

Actualmente, uno de los conceptos más aceptados entre las arquitecturas emergentes de Internet de las Cosas es la división de la red en tres clases funcionales (Figura 2.9). Estas tres clases son: los dispositivos finales que transmiten o reciben sus pequeñas cantidades de datos de forma inalámbrica a través de una amplia variedad de protocolos de comunicación descritos en la sección 2.3.2, los nodos intermedios o de propagación que proveen *gateways* y transporte hacia el Internet tradicional y las funciones integradoras que proporcionan análisis, control e interfaces hombre-máquina a IoT [15].

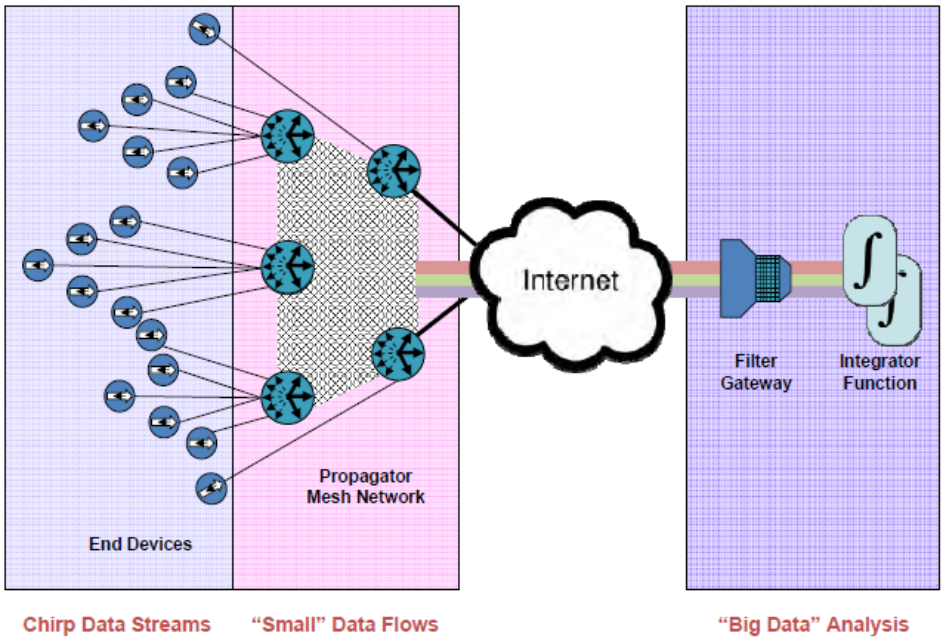


Figura 2.9. Topología de red adoptada en arquitecturas emergentes de IoT [15].

Desde un punto de vista topológico, el procesamiento de datos puede hacerse de una manera muy distribuida y a diferentes niveles de complejidad. Los nodos intermedios o de propagación (*edge nodes*) generan enormes volúmenes de información que son más grandes, y en ocasiones más rápidos, que los generados por los humanos. Sin embargo, gran parte de estos datos tiene poco valor o incluso se trata de ruido. Por este motivo, deben ser filtrados. Esta función es realizada por *middleware* (*edge middleware*) hospedado en los nodos intermedios. Posteriormente, estos datos preprocesados se transforman en información de alto valor mediante un procesamiento más avanzado (p.ej. minería y análisis de datos) realizado en una plataforma de aplicaciones cognitiva, tratándose la mayoría de las veces de una plataforma *cloud computing* de alto rendimiento.

La visión de IoT aumentada con avances en metodologías y tecnologías *software* como la gestión e identificación de dispositivos, sensibilidad al contexto, modelado de sensores, Web semántica, arquitectura orientada a servicios (SOA), *software* como servicio (SaaS), *cloud computing* y otros está causando un cambio de paradigma donde los dispositivos pueden ofrecer un acceso más avanzado de su funcionalidad a la inteligencia empresarial o de negocios (BI). Esta capacidad proporciona una nueva base para enfoques que pueden ser altamente dinámicos y sofisticados y que pueden aprovechar el contexto disponible [27].

A continuación, se describen algunos de los avances y desafíos asociados a este cambio de paradigma. Otras metodologías y tecnologías relacionadas con este trabajo de investigación serán presentadas a lo largo de este capítulo.

2.3.3.1. **Gestión e identificación de dispositivos**

Un desafío especial es la identificación de las cosas. Actualmente, los dispositivos se pueden identificar a través de diferentes mapas (p.ej. UID o identificadores de nivel de comunicación como las direcciones MAC e IP). Con el fin de encontrar información y servicios asociados a las cosas, es necesario disponer de una infraestructura de resolución, que enlace los identificadores de las cosas a otros dispositivos en la red. Probablemente, el ejemplo más claro de tal infraestructura viene dado por el llamado *Object Name Service* (ONS) en la red global del código de producto electrónico (EPC) [88].

2.3.3.2. **Sensibilidad al contexto**

El contexto es clave en IoT y sus aplicaciones. Un gran número de sensores generará grandes cantidades de datos que carecerán de valor a menos que sean analizados, interpretados y comprendidos. La computación sensible al contexto (*context awareness computing*) almacena información del contexto relativa a los datos del sensor, facilitando su interpretación. La sensibilidad al contexto (especialmente al contexto espacial y temporal) juega un papel vital en el comportamiento adaptativo y autónomo de las cosas en IoT [51] [89]. Tal comportamiento ayudará a eliminar de IoT la mediación centrada en el ser humano. Este hecho favorecerá la comunicación M2M.

Las WSN están encontrando cada vez más aplicaciones en una variedad de entornos de computación pervasivos. Sin embargo, la tarea que permite soportar el desarrollo, mantenimiento, despliegue y ejecución de aplicaciones en las WSN sigue siendo desafiante, principalmente debido a la brecha entre los requerimientos de alto nivel de las aplicaciones de computación pervasivas y la operación subyacente de las WSN. El middleware para WSN puede tender un puente sobre la brecha y eliminar estos obstáculos. Trabajos de investigación como [90] muestran cómo el middleware ayuda a construir sistemas IoT sensibles al contexto. Se han realizado avances importantes en el desarrollo de estos sistemas. Las tecnologías de modelado de sensores y Web Semántica son fundamentales para este tipo de sistemas.

2.3.3.3. **Modelado de sensores. La Web Semántica**

Más allá de la identificación de las cosas, es esencial que diferentes aplicaciones puedan acceder y utilizar la información procedente de las cosas. Este objetivo es difícil de conseguir

debido a la heterogeneidad de los formatos de representación de datos. Existen varias iniciativas y organismos de estandarización en el campo de las WSN. El IEEE 1451 [91] es un conjunto de estándares de interfaces de transductores inteligentes. El *Open Geospatial Consortium Sensor Web Enablement* (OGC SWE) [92] ha creado tres normas diferentes para el modelado de sensores y sus observaciones: *Sensor Model Language* (*SensorML*), *Transducer Markup Language* (*TransducerML*) y *Observations & Measurements* (O&M). La estandarización del OGC SWE consiste en un enfoque revolucionario para la explotación de sensores conectados a la Web. El objetivo del SWE es la creación de redes de sensores basadas en la Web (*Web of Things*, WoT) para permitir que todos los sensores y repositorios de datos de sensores sean *descubribles*, *accesibles* y, donde sea aplicable, *controlables* a través de la *World Wide Web*.

Otro obstáculo que impide la interoperabilidad es la falta de estándares de interconexión entre redes inalámbricas (p.ej. entre *Bluetooth* y *Zigbee*). Por ello, el OGC y el *World Wide Web Consortium* (W3C) [93] han estado realizando trabajos de investigación y estandarización siguiendo un enfoque centrado en los datos [94]. El término Web Semántica se refiere a la visión del W3C de la Web de los datos vinculados. Las tecnologías Web semánticas permiten a los desarrolladores crear almacenes de datos en la Web, construir vocabularios y escribir reglas para manejar datos. La Web de Sensores Semánticos (SSW) [95] es un enfoque en el que se anotan metadatos semánticos a los datos del sensor para proporcionar sensibilidad al contexto [51]. En particular, se anotan metadatos espaciales, temporales y temáticos basados en OGC SWE.

2.3.3.4. **Seguridad**

El desafío de seguridad se presenta en todas las capas del diseño físico. Si bien existe un enorme potencial para IoT en diferentes dominios, también hay preocupaciones por la seguridad de las aplicaciones y redes. IoT necesita conectividad y accesibilidad global, lo que significa que cualquier persona puede acceder a esta en cualquier momento y de cualquier forma. Esto aumenta enormemente la superficie de ataque a las aplicaciones y redes de IoT. La complejidad inherente de IoT complica aún más el diseño y el despliegue de mecanismos de seguridad eficientes, interoperables y escalables.

La utilización de IoT, las aplicaciones pueden recopilar información sobre las actividades diarias de las personas. Como la información que refleja tales actividades (p.ej., rutas de viaje, hábitos de compra y uso diario de energía) es considerada privada por muchos individuos, la exposición de esta información podría afectar la privacidad de esas personas. El uso de la computación en la nube hace que el problema de la fuga de privacidad sea aún peor.

*2.3.4. **Desafíos Tecnológicos***

Debido a la complejidad de Internet de las Cosas, existe un mayor número de desafíos que los analizados en esta sección. Sin embargo, el estado del arte de IoT presentado mediante el diseño físico o modelo DNA aporta el conocimiento necesario de este paradigma para la comprensión de esta tesis. La Tabla 2.1 reúne los desafíos analizados durante la descripción

del modelo DNA. Mientras que algunos aparecen en una determinada capa del modelo (p.ej. limitación de recursos), otros deben ser abordados en todas las capas (p.ej. escalabilidad).

<table>
<tr><td rowspan="5">Dispositivos
(D)</td><td>Identificación</td><td rowspan="9">Escalabilidad
Gestión de datos
Interoperabilidad
Seguridad</td></tr>
<tr><td>Limitación de recursos</td></tr>
<tr><td>Maximización del tiempo de vida</td></tr>
<tr><td>Consumo de energía</td></tr>
<tr><td>Coste de despliegue</td></tr>
<tr><td rowspan="2">Redes
(N)</td><td>Topología dinámica y desconocida</td></tr>
<tr><td>Falta de estándares de interconexión</td></tr>
<tr><td rowspan="2">Aplicaciones
(A)</td><td>Infraestructura de resolución</td></tr>
<tr><td>Heterogeneidad de formatos de representación de datos</td></tr>
</table>

Tabla 2.1. Desafíos tecnológicos de IoT.

Respecto a la capa de dispositivos (D), es necesario conseguir la identificación del enorme número de cosas que estarán conectadas a IoT. Muchas de ellas, tendrán recursos computacionales insuficientes para implementar la pila de protocolos IP. Además, en determinados casos, estos dispositivos estarán desplegados en regiones remotas. En ausencia de fuentes de energía, será necesario maximizar su tiempo de vida gracias a la implementación de protocolos de optimización de energía y a los nuevos avances de los sistemas microelectromecánicos y de las técnicas de producción y almacenamiento de energía. Se prevé una reducción del coste a medida que estas tecnologías mejoren y un mayor número de dispositivos se vayan incorporando a IoT.

En la capa de redes (N) existe una escasez de estándares de interconexión entre redes inalámbricas. También surge el desafío de topología dinámica y desconocida ya que muchos de los dispositivos integrados por IoT son de recursos limitados y móviles, estando conectados de forma inalámbrica. Los nodos móviles pueden salir o unirse a la red en cualquier momento. Además, cualquier nodo puede desconectarse debido a los débiles enlaces inalámbricos o a la escasez de batería. Estos factores hacen que la red de IoT sea altamente dinámica. Dentro de tal entorno ad hoc, donde hay una conexión limitada (o ninguna) a una infraestructura fija, será difícil mantener una red estable para soportar muchos escenarios de aplicación que dependen de IoT. Por estos motivos, los nodos tendrán que cooperar para mantener la red conectada y activa.

La capa de aplicaciones (A) plantea la necesidad de implementar una infraestructura de resolución que enlace los identificadores de las cosas a otros dispositivos en la red. La misión de esta infraestructura es encontrar información y servicios asociados a las cosas. Otro desafío es la heterogeneidad de los formatos de representación de datos que dificultan las tareas del acceso y utilización de la información de las cosas por parte de aplicaciones de diferentes dominios. Para implementar estas tareas, existen diferentes técnicas de modelado de sensores.

Los desafíos de escalabilidad, gestión de datos, interoperabilidad y seguridad aparecen en todas las capas. En el futuro, Internet de las Cosas será una red a ultra gran escala formada por miles de millones de nodos. Este hecho unido al tipo de comunicación (M2M) que se produce entre los dispositivos o cosas conectados a IoT plantea un desafío de escalabilidad sin precedentes para las arquitecturas de red tradicionales. Las interacciones entre este increíble número de dispositivos demandan nuevos modelos de comunicación y de distribución de inteligencia que permitan realizar una gestión de datos adecuada.

En Internet de las Cosas, existirá una profunda heterogeneidad de dispositivos, redes (tanto inalámbricas como cableadas) y aplicaciones. Debido a la utilización de protocolos de optimización de energía y a la heterogeneidad de soluciones, es necesario lograr la interoperabilidad para permitir la comunicación entre aplicación de alto nivel, servicios y red subyacente.

Los problemas de seguridad en IoT pueden ocurrir en varios niveles: tecnología inversa así como asuntos éticos y de privacidad. Para garantizar la seguridad de los datos, servicios y todo el sistema IoT, se debe garantizar una serie de propiedades tales como confidencialidad, integridad, autenticación, autorización, disponibilidad y privacidad.

2.4. Arquitecturas de IoT

La multiplicidad de soluciones para abordar cada uno de los desafíos expuestos es una muestra del panorama actual de estandarización. Actualmente, las principales arquitecturas IoT son:

- European Telecommunications Standards Institute M2M Technical Committee [96].
- International Telecommunication Union –Telecommunication sector view [97].
- Internet Engineering Task Force architecture fragments [22].
- Open Geospatial Consortium architecture [22].
- Internet of Things-Architecture (IoT-A) [98].

IoT-A difiere del resto en el sentido que crea una arquitectura de referencia, reflejada en un ARM (*Architectural Reference Model*) horizontal que permite diseñar cualquier aplicación IoT.

A pesar de estas iniciativas, todavía no existe una arquitectura de sistemas IoT o un conjunto universal de estándares que sean ampliamente reconocidos. En el siguiente apartado, se estudia una arquitectura funcional de IoT. Aunque no se trata de un modelo arquitectónico estricto y formal, proporciona una visión conceptual de las funcionalidades de cualquier sistema de IoT.

2.4.1. ***Arquitectura funcional***

La Figura 2.10 presenta la arquitectura funcional de IoT propuesta por [22]. Esta arquitectura sigue un enfoque que permite ver la funcionalidad de un sistema desde un punto de vista en capas. Además, incluye la descripción de las funciones clave que atraviesan varias capas. En adelante, se estudian las diferentes capas funcionales y las capacidades proporcionadas.

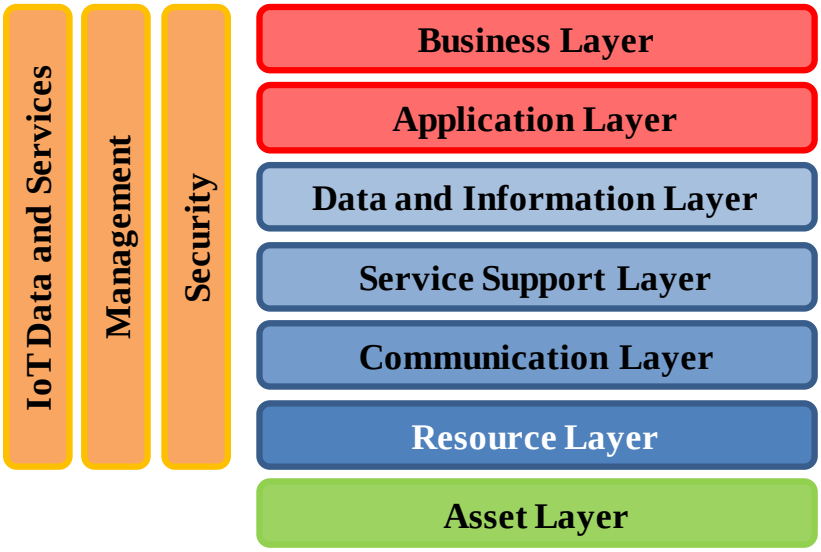


Figura 2.10. Arquitectura Funcional de IoT [22].

En el nivel inferior se encuentra la **Capa de Valor** (*Asset Layer*). Esta capa, estrictamente hablando, no ofrece ninguna funcionalidad a la solución perseguida, sino que representa la razón de ser para cualquier aplicación de la IoT. Los activos de interés son los objetos del mundo real y entidades virtuales que se desea monitorizar, controlar y asignar identidades y representaciones digitales. Para ello, los activos son equipados con tecnologías embebidas (apartado 2.3.1.1).

La **Capa de Recurso** (*Resource Layer*) proporciona las principales capacidades funcionales de detección, actuación e identidades embebidas gracias a la utilización de *middleware* embebido. Los sensores y actuadores en varios dispositivos ofrecen estas funciones (sección 2.2). En este nivel, también se despliegan *gateways* (apartado 2.3.3) de diferentes tipos que pueden proporcionar agregación u otras capacidades que están estrechamente relacionadas con estos recursos básicos.

El objetivo de la **Capa de Comunicación** (*Communication Layer*) es proporcionar los medios para la conectividad entre los recursos en un extremo y las diferentes infraestructuras de computación que hospedan y ejecutan la lógica de soporte de servicio y la lógica de aplicación en el otro extremo. En el apartado 2.3.2 se han estudiado diferentes tecnologías de comunicación utilizadas por los sistemas IoT.

Las aplicaciones de IoT son más simples ya que están basadas en diferentes tipos de servicios de soporte como servicios para monitorización de dispositivo, servicios de control de dispositivo, servicios de publicación de datos, servicios para descubrimiento de datos, etc. Estos servicios son proporcionados por la **Capa de Soporte de Servicio** (*Service Support Layer*) y, por lo general, se ejecutan en los centros de datos o granjas de servidores pertenecientes a organizaciones o en un entorno *cloud*. Los servicios de soporte pueden proporcionar un manejo uniforme de los dispositivos y redes subyacentes, ocultando así complejidades en las capas de recurso y comunicación. Algunos ejemplos son la

administración remota de dispositivos pudiéndose realizar actualizaciones remotas de software, diagnóstico o recuperación remota o la reconfiguración dinámica del procesamiento de aplicaciones como el ajuste de filtros de eventos. Las funciones relacionadas con la comunicación incluyen la selección de canales de comunicación si se pueden utilizar diferentes redes en paralelo, por ejemplo, para propósitos de confiabilidad y mecanismos *publish*/*subscribe* y *push-pull* que son descritos más adelante en este capítulo.

Mientras que las capas de recurso, comunicación y soporte de servicio tienen realizaciones concretas en términos de dispositivos y etiquetas, redes y nodos de red y servidores, la **Capa de Información y Datos** (*Data and Information Layer*) ofrece un conjunto más abstracto de funciones ya que sus principales propósitos son capturar el conocimiento y proporcionar soporte lógico de control avanzado. En esta capa, los conceptos clave son los modelos de datos e información y la representación del conocimiento y el foco está en la organización de la información. Se suele utilizar el término *Framework* de Gestión de Conocimiento (KMF) para abarcar estos conceptos.

La **Capa de Aplicación** (*Application Layer*) proporciona las aplicaciones de IoT. Las aplicaciones IoT facilitan una interfaz a los usuarios para monitorizar y controlar varios aspectos del sistema IoT. Además, permiten a los usuarios ver el estado del sistema y ver o analizar los datos procesados.

La **Capa Empresarial o de Negocios** (*Business Layer*) corresponde al último nivel de la arquitectura. Se centra en el soporte del núcleo de negocio o de las operaciones de cualquier empresa, organización o particular que esté interesado en las aplicaciones de IoT. Aquí es donde tiene lugar cualquier integración de las aplicaciones IoT en los procesos de negocio y sistemas empresariales.

Además, tres grupos funcionales cruzan las diferentes capas (Figura 2.10), a saber: **Gestión** (*Management*), **Seguridad** (*Security*) y **Datos y Servicios** IoT (*IoT Data and Services*). Los dos primeros son funciones bien conocidas de un sistema TIC, mientras que el último es más específico de IoT.

Gestión, proporciona varias funciones para gobernar el sistema IoT. Se ocupa de la gestión de diversas partes relacionadas con su operación, mantenimiento, administración y suministro. Esto incluye la gestión de dispositivos, redes de comunicaciones y la infraestructura general TIC, así como datos de configuración y suministro, rendimiento de los servicios entregados, etc.

La **Seguridad** persigue la protección del sistema, su información y servicios, de las amenazas externas o de cualquier otro daño. Generalmente, se requieren medidas de seguridad en todas las capas, por ejemplo, proporcionando seguridad en la información y en la comunicación. La gestión de la confianza e identidad y la autenticación y autorización son capacidades clave.

El último grupo funcional se denomina **Datos y Servicios IoT** (IoT *Data and Services*). Como se ha descrito en el apartado 2.3.3, el procesamiento de datos y servicios puede hacerse de una manera muy distribuida y a diferentes niveles de complejidad. En algunos casos (p.ej. ciertas WSAN), el filtrado básico de eventos y la agregación más sencilla, como el promedio de datos, puede tener lugar en cada nodo sensor. A medida que aumenta el nivel de

complejidad, los metadatos contextuales, como la ubicación y la información temporal, y una agregación adicional pueden agregarse a los datos procedentes de los nodos sensores. El mayor nivel de complejidad tiene lugar en la plataforma *middleware* del lado del servidor donde es posible realizar un procesamiento más avanzado, como la minería y el análisis de datos, casi en tiempo real.

Datos y Servicios IoT representa así el flujo vertical de datos relativo al conocimiento, la abstracción de datos y servicios en diferentes niveles y los pasos del proceso de extracción de conocimiento. Dado que la capa de **Información y Datos** se encarga de la organización y representación del conocimiento, este grupo funcional se centra en los diferentes pasos de procesamiento de la cadena de valor de datos y servicios. Por tanto, soporta cada nivel de extracción del conocimiento (procesamiento, razonamiento y toma de decisiones). Para ello, emplea diferentes tecnologías de análisis (*analytics*), aprendizaje máquina (*machine learning*), razonamiento (*reasoning*) e inferencia (*inferencing*). Posteriormente, en este capítulo se proporciona una descripción de las tecnologías y herramientas para el procesamiento de datos y servicios.

2.5. La necesidad de middleware e interfaces abiertas

La simple exposición de diversas arquitecturas que ponen orden en la estandarización IoT debido a los desafíos revisados en la sección 2.3 (Tabla 2.1) revela “en todo su esplendor” la necesidad de la utilización de *middleware* en los dispositivos (*embedded middleware*), en la infraestructura de la red (*edge middleware*) y en la plataforma en el lado del servidor (*server-side middleware platform*). Según [35], el *middleware* es omnipresente ya que existe cerca de todas las partes de un sistema de tecnologías de la información y comunicaciones. Por tanto, el *middleware* es la piedra angular de todo sistema IoT ya que es crucial para lograr una comunicación eficiente máquina a máquina, ocultando la heterogeneidad de los dispositivos, de las comunicaciones y de las redes. El *middleware* ofrece un conjunto de *servicios enablers*, tales como APIs estandarizadas, protocolos y servicios de infraestructura para soportar el desarrollo rápido y adecuado de servicios y aplicaciones distribuidas del *Internet del Futuro*.

Debido a su importancia, se han llevado a cabo muchos proyectos de investigación acerca de *middleware* de IoT, que han sido estudiados, comparados y clasificados por diferentes autores [50] [99] [100] [101] [102]. En la mayoría de los casos, el *middleware* WSN se implementa como *middleware* embebido en el nodo. Una lista de *middleware* WSN, *software*/OS y lenguajes de programación WSN está disponible en [35]. Cada solución *middleware* se centra en diferentes desafíos de IoT, como la gestión de datos, interoperabilidad, seguridad y muchos más. Una solución *middleware* que aborde todos los desafíos que plantea IoT todavía está por ser diseñada.

La tendencia en el desarrollo del *middleware* es la utilización de un enfoque arquitectónico orientado a servicios (SOA) y la aplicación de los estándares Web (SOAP y RESTful) [27] [35].

Al extender el paradigma Web a los dispositivos, estos pueden convertirse en un componente natural en la construcción de cualquier aplicación IoT y facilitar la integración de los servicios del dispositivo en cualquier sistema empresarial que esté basado en SOA. Las aplicaciones IoT pueden entonces pasar a ser independientes de la tecnología y del lenguaje de programación. Esto ayudará a impulsar el mercado de desarrollo de aplicaciones de IoT, siendo las API abiertas (*Open* APIs) un componente clave en el establecimiento del mismo.

Por tanto, se prevé que la colaboración entre capas sea una cuestión clave en esta infraestructura altamente dinámica y heterogénea denominada Internet de las Cosas. La clave es disponer de una única plataforma común que se pueda utilizar para todo los tipos de aplicaciones verticales de IoT. Así, la infraestructura de IoT, desde la capa de recurso (Figura 2.10) o recolección de datos que abarca la última milla de las WSN y *gateway* hasta las redes de acceso y finalmente hasta la red troncal, puede ser distribuida y replicada. Sin embargo, las capas por encima de la red troncal deben estar altamente integradas y centralizadas sobre una única plataforma común agnóstica PaaS+SaaS (plataforma como servicio + *software* como servicio) que acomode las variaciones de la conectividad incluyendo las capas IaaS (infraestructura como servicio) [35]. WSO_2 [30], PubNub [31] y FIWARE [32] son claros ejemplos de este nuevo tipo de plataforma. Concretamente, FIWARE aplica los conceptos del proyecto SENSEI [103] y de la arquitectura IoT-A [104].

2.5.1. ***Modelos de comunicación IoT***

A continuación se describen diversos modelos de comunicación utilizados en Internet de las Cosas [49].

- *Request-Response* (Solicitud-Respuesta): es un modelo de comunicación en el que el cliente envía solicitudes al servidor y el servidor responde a dichas solicitudes. Cuando el servidor recibe una solicitud, este decide cómo responder, busca los datos, recupera las representaciones de recursos, prepara la respuesta y luego la envía al cliente. El modelo de solicitud-respuesta es un modelo de comunicación sin estado. Así, cada par solicitud-respuesta es independiente del resto. La Figura 2.11 muestra las interacciones cliente-servidor que tienen lugar en este modelo.

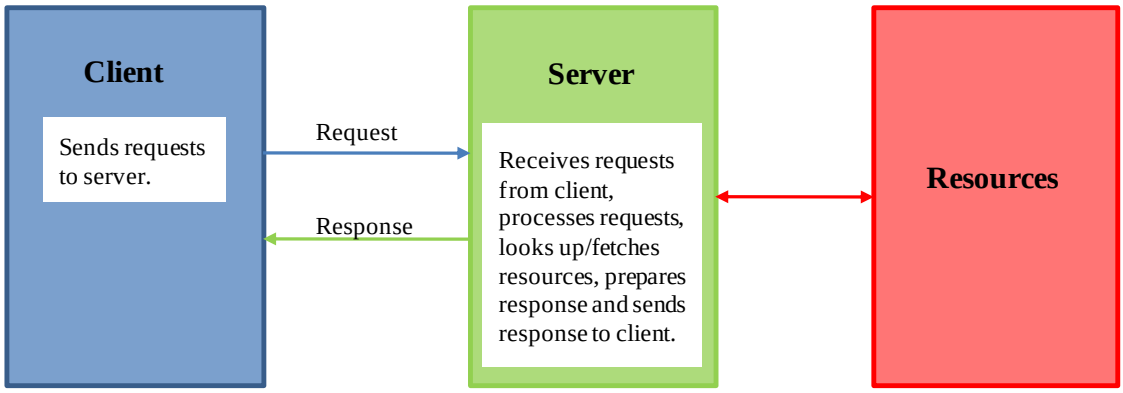


Figura 2.11. Modelo de comunicación *Request-Response* [49].

- *Publish-Subscribe* (Publicación-Suscripción): es un modelo de comunicación que involucra a productores (*producers*), intermediarios (*brokers*) y consumidores (*consumers*). Los productores son las fuentes de datos y se encargan de enviar información relativa a los diferentes temas (*topics*) gestionados por el *broker*. Los productores no son conscientes de los consumidores. Estos se suscriben a los temas del *broker*. Cuando el *broker* recibe información de un productor, comprueba a qué tema corresponde y la retransmite a todos los consumidores suscritos. La Figura 2.12 muestra las interacciones productor-*broker*-consumidor en este modelo.

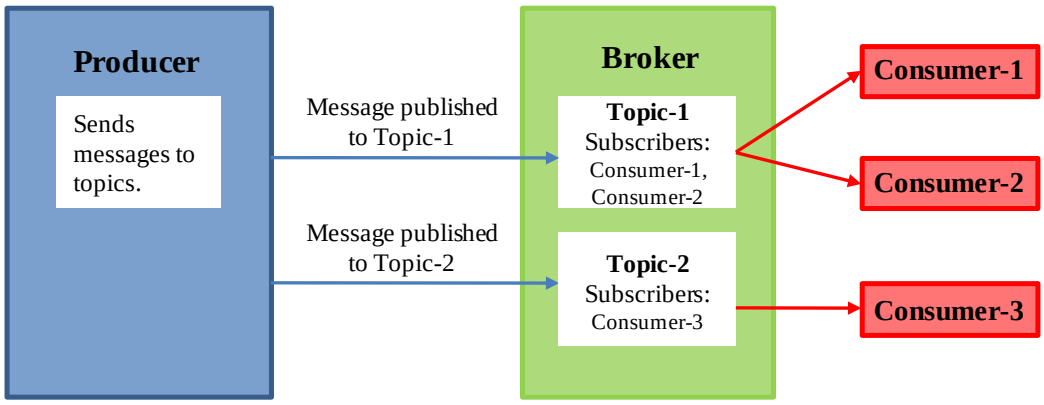


Figura 2.12. Modelo de comunicación *Publish-Subscribe* [49].

- *Push-Pull* (Empujar-Tirar): en este modelo de comunicación, los productores encolan los datos. Estos datos son extraídos por los consumidores. Los productores no necesitan ser conscientes de los consumidores. Las colas ayudan a desacoplar el intercambio de mensajes entre productores y consumidores. Además, al tratarse de memorias intermedias (*buffers*), ayudan en situaciones en las que hay un desajuste entre la tasa a la que los productores encolan los datos y la empleada por los consumidores para extraerlos. La Figura 2.13 muestra las interacciones productor-cola-consumidor en este modelo.

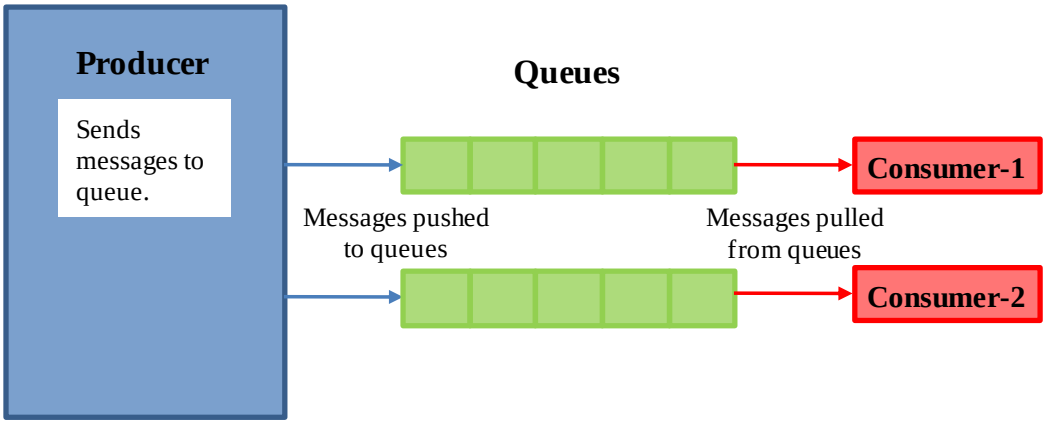


Figura 2.13. Modelo de comunicación *Push-Pull* [49].

- *Exclusive Pair* (Par Exclusivo): es un modelo de comunicación bidireccional *full duplex* que utiliza una conexión persistente entre cliente y servidor. Una vez que la conexión está configurada, esta permanece abierta y tanto cliente como servidor pueden comunicarse hasta que el cliente envía una solicitud para cerrarla. Par Exclusivo es un modelo de comunicación con estado donde el servidor es consciente de todas las conexiones abiertas. La Figura 2.14 muestra las interacciones cliente-servidor en este modelo.

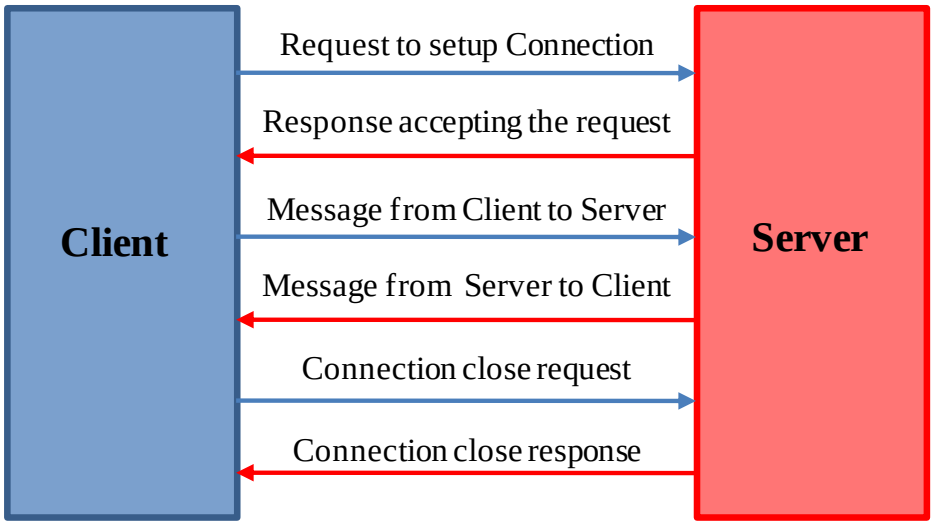


Figura 2.14. Modelo de comunicación *Exclusive Pair* [49].

*2.5.2. **API de comunicación IoT***

En este apartado se describen dos API de comunicación que son ampliamente utilizadas en Internet de las Cosas [49]. Específicamente, las API basadas en REST son de un gran interés para el caso que nos ocupa debido a los trabajos de investigación que se han realizado como contribución de esta tesis.

2.5.2.1. **API de comunicación basadas en REST**

La Transferencia de Estado Representacional (REST) [105] es un conjunto de principios arquitectónicos mediante los cuales se pueden diseñar servicios web y API web que se centran en los recursos de un sistema y en cómo se direccionan y transfieren los estados de dichos recursos. Las API REST siguen el modelo de comunicación solicitud-respuesta descrito en el apartado anterior. Las restricciones de la arquitectura REST se aplican a los componentes, conectores y elementos de datos dentro de un sistema hipermedia distribuido. Las restricciones arquitectónicas de REST son las siguientes:

- **Cliente-Servidor:** el principio detrás de la restricción cliente-servidor es la separación de las funciones. Por ejemplo, los clientes no tienen que preocuparse por el almacenamiento de datos ya que se trata de una tarea que debe realizar el servidor. Del mismo modo, el servidor no tiene que preocuparse por la interfaz de usuario ya que es una tarea del cliente. Este principio permite que cliente y servidor sean desarrollados y actualizados de forma independiente.

- **Sin estado:** cada solicitud del cliente al servidor debe contener toda la información necesaria para la comprensión de la misma y, por tanto, no puede aprovechar ningún contexto almacenado en el servidor. El estado de la sesión se mantiene en el cliente.
- **Cache-able:** la restricción de caché requiere que los datos que componen una respuesta a una solicitud estén etiquetados de forma implícita o explícita como *cache-able* o *non-cache-able*. Si una respuesta es *cache-able*, entonces, se otorga el derecho de reutilizar los datos de esta respuesta a una caché de cliente para posteriores solicitudes equivalentes. El almacenamiento en caché puede eliminar parcial o totalmente algunas interacciones y mejorar la eficiencia y escalabilidad.
- **Sistema de capas:** esta restricción condiciona el comportamiento de los componentes de tal manera que cada uno no puede ver más allá de la capa con la que está interactuando. Por ejemplo, un cliente no puede saber si está conectado directamente al servidor final o a un intermediario en el camino. De esta forma, es posible mejorar la escalabilidad del sistema al permitir que los intermediarios respondan a la solicitud en lugar del servidor final sin que el cliente tenga que hacer algo diferente.
- **Interfaz uniforme:** esta restricción requiere que el método de comunicación entre un cliente y un servidor debe ser uniforme. Los recursos se identifican en las solicitudes (mediante URI en sistemas basados en web) y son independientes de las representaciones de los recursos que se devuelven al cliente. Cuando un cliente tiene una representación de un recurso, tiene toda la información necesaria para actualizar o eliminar dicho recurso (siempre que el cliente tenga los permisos necesarios). Cada mensaje incluye suficiente información para describir cómo realizar su procesamiento.

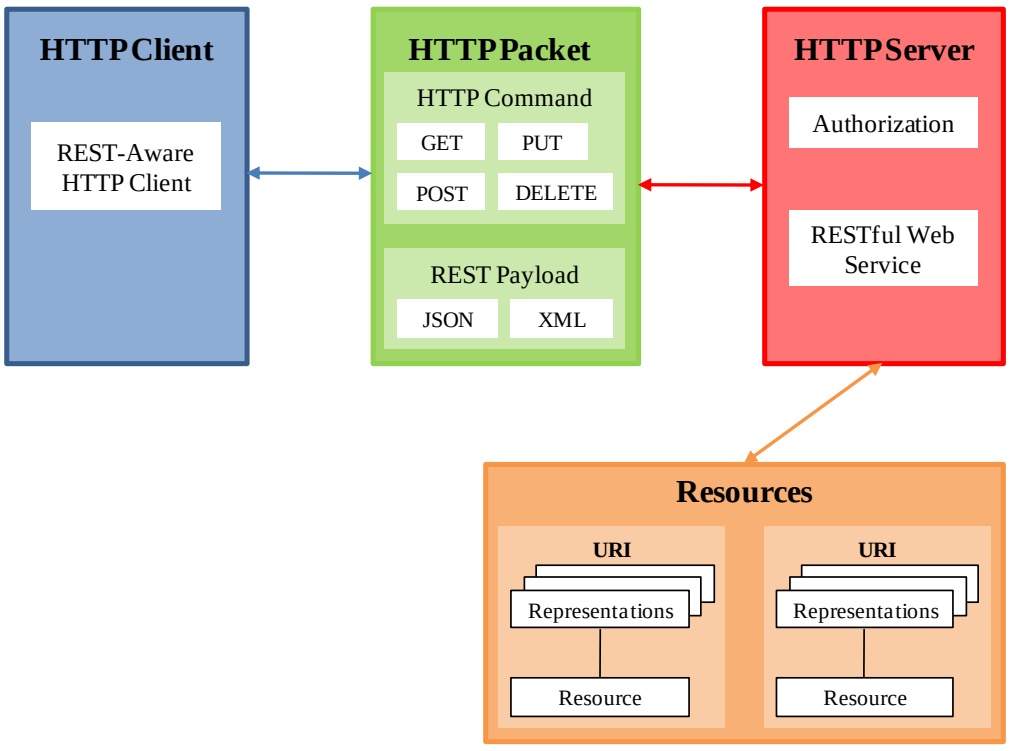


Figura 2.15. Comunicación mediante API REST [49].

- **Código bajo demanda:** los servidores pueden proporcionar código ejecutable o scripts para su ejecución en el contexto de los clientes. Esta es la única restricción opcional.

Un servicio web RESTful es una "API web" implementada utilizando los principios HTTP y REST. La Figura 2.15 muestra la comunicación entre el cliente y el servidor mediante la API REST. La Figura 2.16 muestra las interacciones en el modelo solicitud-respuesta utilizado por REST. El servicio web RESTful es una colección de recursos que están representados por URI. La API web RESTful tiene un URI de base. Los clientes envían solicitudes a estos URI utilizando los métodos definidos por el protocolo HTTP (Tabla 2.2). Un servicio web RESTful puede soportar varios tipos de formato de texto, siendo JSON el más utilizado. IP *for Smart Objects Alliance* (IPSO *Alliance*) ha publicado un *Framework* de Aplicación que define un diseño RESTful para su uso en sistemas de objetos inteligentes IP [106].

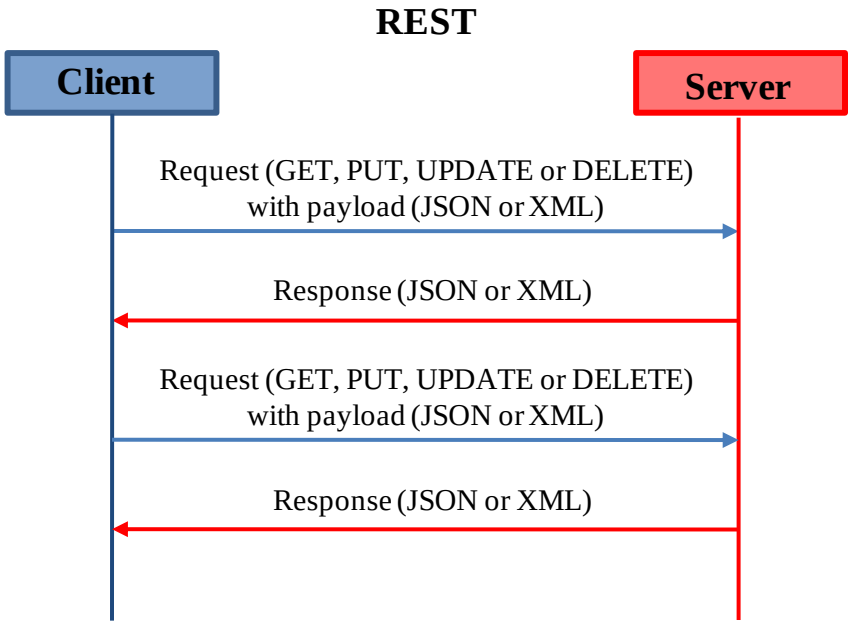


Figura 2.16. Modelo *Request-Response* utilizado por REST [49].

2.5.2.2. API de comunicación basadas en WebSocket

Las API WebSocket permiten comunicación bidireccional *full duplex* entre clientes y servidores. Las API WebSocket siguen el modelo de comunicación Par Exclusivo descrito en el apartado anterior, como se muestra en la Figura 2.17. A diferencia de las API solicitud-respuesta como REST, las API WebSocket permiten comunicación *full duplex* y no requieren que se configure una nueva conexión para enviar cada mensaje. La comunicación de WebSocket comienza con una petición de configuración de conexión enviada por el cliente al servidor. Esta solicitud se envía a través de HTTP y el servidor la interpreta como una solicitud de actualización. Si el servidor soporta el protocolo WebSocket, responde a la solicitud del cliente. Una vez configurada la conexión, cliente y servidor pueden enviarse datos/mensajes en modo *full duplex*. Las API WebSocket reducen el tráfico y la latencia de la red ya que no hay sobrecarga debido a la ausencia de solicitudes de configuración y cierre de conexión por cada mensaje. WebSocket es adecuado para aplicaciones IoT que tienen requerimientos de baja latencia o alto rendimiento.

HTTP Method	Resource Type	Action	Example
GET	Collection URI	List all the resources in a collection	http://example.com/api/tasks/ (list all tasks)
GET	Element URI	Get information about a resource	http://example.com/api/tasks/1/ (get information on task-1)
POST	Collection URI	Create a new resource	http://example.com/api/tasks/ (create a new task from data provided in the request)
POST	Element URI	Generally not used	
PUT	Collection URI	Replace the entire collection with another collection	http://example.com/api/tasks/ (replace entire collection with data provided in the request)
PUT	Element URI	Update a resource	http://example.com/api/tasks/1/ (update task-1with data provided in the request)
DELETE	Collection URI	Delete the entire collection	http://example.com/api/tasks/ (delete all tasks)
DELETE	Element URI	Delete a resource	http://example.com/api/tasks/1/ (delete task-1)

Tabla 2.2. Métodos de solicitud HTTP [49].

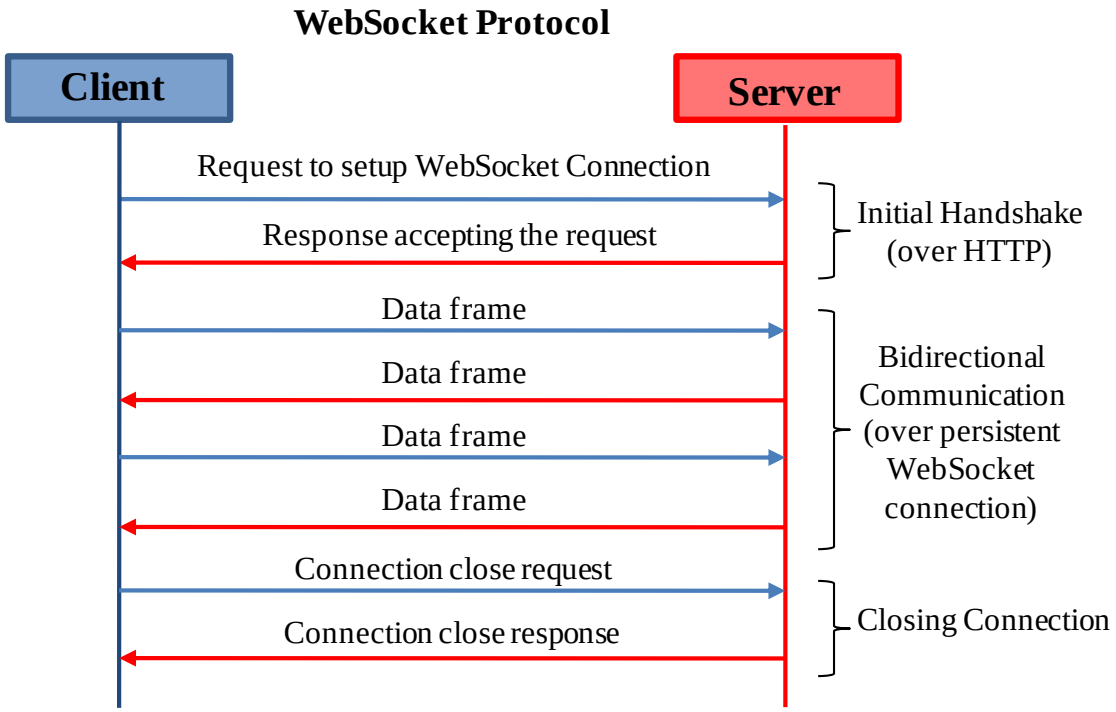


Figura 2.17. Modelo *Exclusive Pair* utilizado por las API WebSocket [49].

*2.5.3. **Arquitectura Orientada a Servicios***

La Arquitectura Orientada a Servicios (SOA), acrónimo del inglés *Service Oriented Architecture*, es un concepto de arquitectura *software* que define la utilización de servicios para dar soporte a los requerimientos de *software* del usuario. SOA establece una serie de principios específicos que deben tenerse en cuenta en el diseño y definición de los servicios de un sistema que siga esta orientación, y que incluyen de forma directa en el comportamiento del sistema. Existen distintas definiciones del concepto de servicio en SOA, aunque todas ellas coinciden que es una entidad sin estado, auto-contenida, que acepta peticiones y devuelve respuestas mediante una interfaz bien definida, y que no depende del estado de otras entidades o procesos. La especificación de una descripción completa de los servicios mediante un contrato de servicio y la obtención de características como acoplamiento débil, encapsulación, abstracción, reusabilidad, composicionalidad, autonomía, optimización y descubrimiento, son elementos fundamentales para la obtención de una plataforma de servicios conforme a los principios SOA [107].

Las arquitecturas orientadas a servicios son arquitecturas débilmente acopladas en las que el enlace a los servicios puede cambiar dinámicamente durante la ejecución. Como características principales del concepto de servicio podríamos decir que (a) es una representación estándar para cualquier recurso computacional o de información que pueda ser utilizado por otras aplicaciones, (b) la provisión del servicio es independiente de la plataforma asociada a la aplicación que lo proporciona, y (c) las aplicaciones se construyen enlazando servicios desde varios proveedores utilizando lenguajes de programación, como por ejemplo Java. Otros conceptos relacionados son los de interfaz de servicio, que define los

datos disponibles para/por el servicio y cómo se puede acceder a ellos; proveedor de servicio, que oferta un servicio a las aplicaciones definiendo su interfaz y su funcionalidad; autoadaptabilidad de las aplicaciones, que modifican su comportamiento de acuerdo con su entorno de ejecución, enlazando con diferentes servicios según este cambia; basado en estándares, los servicios-software se han desarrollado guiados por estándares desde su nacimiento (SOAP, WSDL, UDDI, etc.); e hibridación, algunas aplicaciones se construirán solamente usando servicios y otras los mezclarán con servicios desarrollados localmente [108].

2.5.3.1. **Función de SOA**

La arquitectura SOA proporciona un entorno de servicios independientes, al que nuevos servicios puedan acceder sin necesidad de conocer la implementación o el funcionamiento de la plataforma subyacente. La clave para el desarrollo de un sistema bajo los principios SOA es obtener interfaces de servicio independientes cuyas tareas pueden invocarse de una forma estándar, sin que el servicio tenga conocimiento de las llamadas a la aplicación, y sin que la aplicación necesite conocer cómo este desempeña sus tareas.

El paradigma SOA pretende establecer las bases para garantizar la comunicación entre los distintos componentes de un sistema, de forma que los mensajes tengan una estructura y sintaxis específica y se expresen en un vocabulario compartido por todos. El vocabulario se define en un esquema, cuya extensibilidad es imprescindible para garantizar un amplio abanico de ámbitos de aplicación. Puesto que todas las aplicaciones con una semántica concreta constan de elementos de datos, como los mensajes, un servicio puede ser caracterizado por el conjunto de mensajes que puede interpretar adecuadamente. Estos mensajes están especificados en base a un determinado esquema que se puede considerar como una definición abstracta de un servicio. El consumidor de un servicio solamente conoce su definición abstracta, donde el esquema especifica para cada servicio la funcionalidad que tiene asociada dentro del dominio de aplicación.

2.5.3.2. **Principios SOA**

El paradigma SOA establece una serie de principios específicos que deben tenerse en cuenta en el diseño y definición de los servicios de un sistema que siga esta orientación, y que incluirán en su comportamiento.

Como podemos observar en la Figura 2.18, SOA define tres tipos de actores, el *Service Consumer* o consumidor de servicios, el *Service Provider* o proveedor de servicios y el *Service Directory* o directorio de servicios [107]. El proveedor es responsable de ofrecer una funcionalidad específica a sus consumidores encapsulada en una unidad software autónoma. El consumidor es el cliente que solicita la funcionalidad de un determinado servicio. Por último, el directorio posee un rol de intermediario entre ambos, proveedor y consumidor. El directorio de servicios contiene una descripción actualizada del conjunto de servicios proveedores disponibles; además dispone de mecanismos para la publicación, descubrimiento y eliminación de servicios si es necesario. Los servicios proveedores publican su descripción en el directorio de servicios con el objetivo de que los consumidores puedan localizarlos y

utilizarlos. Una vez que la descripción de los proveedores está disponible, los clientes pueden acceder a ellos y solicitar su funcionalidad.

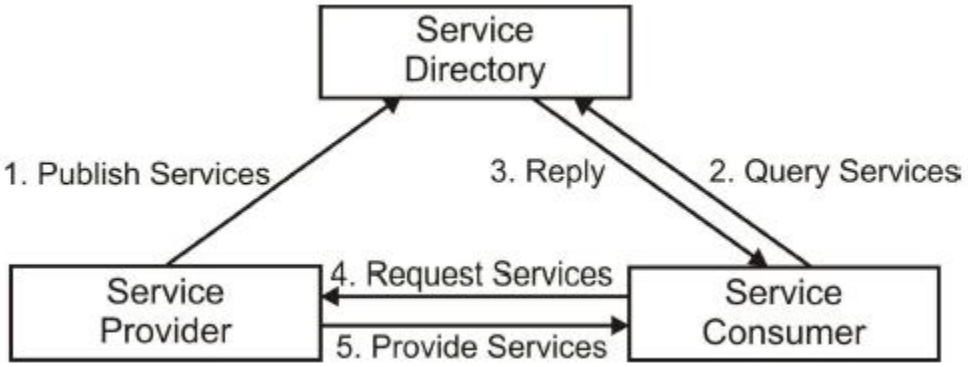


Figura 2.18. Entidades principales de SOA [16].

El paradigma SOA impone el uso de una serie de restricciones en el diseño y organización de los servicios. Como consecuencia directa de estas restricciones es necesario establecer una separación entre el comportamiento interno y externo en el desarrollo de los servicios. De acuerdo con [108] y [107], las principales propiedades que deben satisfacer los servicios son las siguientes.

Contrato de servicio. Un contrato de servicio puede ser una descripción, técnica o no, de las características del servicio y su comportamiento. Es una buena práctica en sistemas distribuidos que cuando dos elementos del sistema van a comunicarse, utilicen un estándar, cuyo papel jugaría en este caso el "contrato de servicio". Así, los servicios y usuarios de un mismo conjunto de servicios deben seguir el mismo "contrato de servicio".

Bajo acoplamiento. El nivel de acoplamiento entre servicios puede entenderse también como el nivel de dependencia, que en programación distribuida es deseable que sea el mínimo posible. Por ello, los servicios deben establecer relaciones que minimicen las dependencias con otros servicios o usuarios, de los que solo se requiere conocer su existencia. El cliente, ya sea otro servicio, usuario final, o un programa externo, no tiene por qué ser consciente de cómo un servicio lleva a cabo sus responsabilidades. Las estructuras internas de datos, las invocaciones a otros servicios, la gestión de transacciones y los requisitos de almacenamiento deberían estar ocultos al cliente. Para garantizar que se cumpla este principio, debe considerarse tanto en el diseño del servicio como en sus relaciones con otros servicios o usuarios, lo cual afectará también al diseño del contrato de los servicios.

Encapsulación. Durante el diseño de los servicios debe tenerse en cuenta este principio y establecer de forma clara una separación del comportamiento interno y externo. El servicio solo debe definir como observable en el sistema su comportamiento externo, no la forma en que se realiza la conducta, y reflejarlo en el "contrato de servicio".

Abstracción. Por encima de lo que se describe en su contrato, los servicios deben ocultar su funcionamiento lógico al mundo exterior, es decir, deben ocultar la información que no es absolutamente necesaria para utilizar de manera eficaz el servicio por parte de otros servicios

o usuarios. Para garantizar este principio es aconsejable seguir el enfoque de "caja negra" en el diseño de los servicios. La información que es deseable abstraer al aplicar este principio se denomina metainformación, y abarca los siguientes aspectos del servicio: tecnología, funcionalidad, lógica de programación y calidad del servicio. Los niveles de abstracción establecidos deben tenerse en cuenta a la hora de especificar el contenido de los "contratos de servicio".

Reusabilidad. Uno de los principales objetivos de la utilización de un modelo de programación SOA es la reutilización, pues se divide la lógica de una aplicación concreta en servicios con el fin de garantizar este principio.

Composicionalidad. Este principio pretende que dada una colección de servicios, puedan ser coordinados y ensamblados para formar nuevos servicios compuestos.

Autonomía. La autonomía de un servicio representa el nivel de independencia con el que este puede llevar a cabo su lógica, para lo cual es necesario tener en cuenta requerimientos asociados a la infraestructura del servicio en el diseño del mismo. Este principio apoya los principios de "reusabilidad" y "composicionalidad".

Optimización. Se debe garantizar la opción de incluir modificaciones y mejoras en los servicios, a fin de mantener los servicios en el mejor estado posible.

Descubrimiento. Los servicios deben diseñarse para que sean lo más descriptivos posible hacia el exterior, de forma que puedan ser encontrados y evaluados a través de los mecanismos de descubrimiento disponibles. El cliente no tiene por qué ser consciente de dónde reside o se ejecuta el servicio. El objetivo fundamental de este principio es el de mejorar la calidad de las comunicaciones entre servicios y usuarios.

2.5.3.3. **Beneficios de SOA**

Los beneficios que puede obtener un sistema de computación ubicua que adopte un enfoque SOA son:

- Optimización del tiempo necesario para la realización de cambios en los procesos.
- Facilidad para evolucionar los modelos, dinamicidad.
- Facilidad para abordar modelos de comportamiento colaborativos.
- Poder para reemplazar aplicaciones SOA sin contrapartida en el resto del sistema.
- Facilidad para la integración de tecnologías heterogéneas.

En un ambiente SOA, los nodos de la red comunican su disponibilidad y sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones de SOA identifican la utilización de Servicios Web, empleando SOAP y WSDL, en su implementación. No obstante, se puede implementar SOA utilizando cualquier tecnología basada en servicios, siempre aconsejándose el uso de estándares para garantizar la extensibilidad de los sistemas.

Al contrario de las arquitecturas orientadas a objetos, las arquitecturas SOA están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación, por ejemplo WSDL. La definición de

la interfaz encapsula u oculta las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo, garantizando la heterogeneidad y la interoperabilidad. Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar.

2.6. Computación en la nube

El paradigma de la computación en nube es uno de los mejores aspectos de la evolución de las TIC para la IoT, ya que permite entornos de ejecución independientes y virtualizados para alojar de forma aislada múltiples aplicaciones en la misma plataforma *hardware*, y por lo general en grandes centros de datos. Además, la computación en la nube tiene la ventaja de facilitar a los diferentes negocios su interconexión si se están ejecutando en la misma plataforma.

La computación en la nube es también un *enabler* clave cuando se pasa de una propuesta orientada al producto a una propuesta orientada a servicios, debido a su elasticidad que permite a las empresas una oferta "*pay-as-you-grow*".

En estrecha relación con el tema de los centros de datos, el *software* de inteligencia y procesamiento de datos tendrá un papel cada vez más importante en las soluciones de la IoT. Un concepto popular ahora es *Big data*, que se refiere al incremento del número y tamaño de los conjuntos de datos que están disponibles para empresas y particulares con el fin de recopilarlos y analizarlos. En el campo de *Big data*, han surgido nuevos paradigmas. *MapReduce*[5] [109] es uno de los que más expectativas ha generado en escenarios en los que la latencia no es un requisito crítico. Naturalmente, estas tecnologías son, por tanto, *enablers* clave de la IoT, ya que permiten la agregación y recopilación de los conjuntos de datos masivos que los dispositivos y sensores puedan producir. IoT es único en comparación con otras aplicaciones *Big Data*, tales como el análisis de redes sociales ya que, como se ha descrito anteriormente, la comunicación entre las máquinas es diferente a la de los humanos.

2.6.1. ***La arquitectura de la nube de las cosas***

Como se ha mencionado anteriormente, IoT y la computación en la nube tienen características comparables. Por ejemplo, la computación en la nube tiene tres capas: IaaS, PaaS y SaaS. De la misma forma, IoT consiste en tres capas: modelo DNA [55] (Devices, Networks y Applications) o modelo DCM [35] (*Devices*, *Connect* y *Manage*). Otra característica de la computación en la nube es que permite diferentes despliegues: *public cloud*, *private cloud*, *hybrid cloud*, y así sucesivamente. De una manera similar, existen diferentes despliegues de IoT: *Intranet of Things*, *Extranet of Things*, *Internet of Things*, etc.

A pesar de que el paradigma de IoT sigue evolucionando, las piezas fundamentales del puzle han sido globalmente aceptadas. [35] propone una posible especificación de la arquitectura/*framework* de IoT. De esta manera, pretende ayudar a formar un *framework*

[5] Map&Reduce: http://research.google.com/archive/mapreduce.html

común de arquitectura de referencia y establecer una terminología común con el propósito de lograr un *middleware* unificado para Internet de las Cosas.

La Figura 2.19 muestra el *framework* general de Internet de las Cosas. Su definición, atributos, características, casos de uso, tecnologías subyacentes, riesgos y beneficios serán refinados y cambiados con el tiempo debido a la opinión pública y del sector privado. Aunque la industria del IoT representa un gran ecosistema, esta especificación intenta abarcar todos los enfoques de IoT.

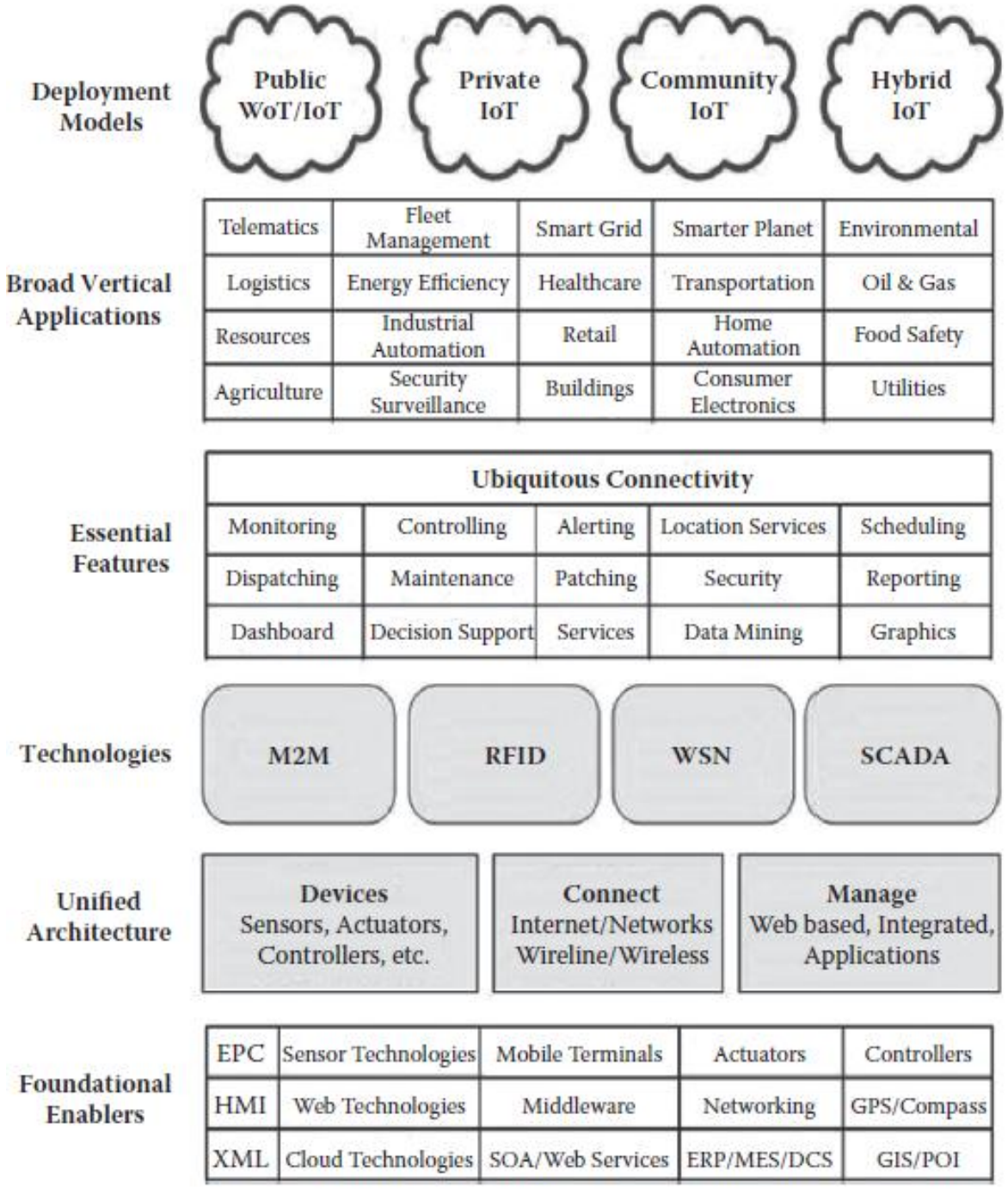


Figura 2.19. Especificación arquitectónica de la nube de las cosas [35].

Cuatro modelos de despliegue

- *Private* IoT: el sistema IoT es operado solamente por una organización. Como ejemplo, este puede ser el caso de un sistema de gestión de edificios operado por una empresa de administración de fincas. Dicho sistema podría ser administrado por la organización o por un tercero y podría existir en un entorno local (intranet) o fuera del mismo (extranet).
- *Public* IoT: el sistema IoT se pone a disposición del público general o de un gran grupo industrial. Dicho sistema es propiedad de una organización que vende servicios de IoT.
- *Community* IoT: el sistema integrado es compartido por varias organizaciones y soporta una comunidad específica que ha compartido intereses (por ejemplo, misión, requisitos de seguridad, etc.). Podría ser gestionado por las organizaciones o un tercero y podría existir en un entorno local o fuera del mismo.
- *Hybrid* IoT: el sistema IoT es una composición integrada de dos o más de los sistemas IoT mencionados previamente (*private*, *community* o *public*) que siguen siendo entidades únicas pero unidas por tecnología estandarizada o propietaria que permite la portabilidad de datos y aplicaciones.

Aplicaciones verticales

Actualmente existe un gran número de aplicaciones verticales. Además, otras muchas están por aparecer. Por ello, elaborar una lista que las contemple es una tarea complicada. En la arquitectura de la Figura 2.19 se muestran aquellas aplicaciones verticales cuya implantación está prevista en un futuro a corto plazo, a saber:

- Telemática, gestión de flotas, transporte
- Red inteligente, eficiencia energética
- Planeta más inteligente
- Protección ambiental
- Logística, comercio minorista
- Cuidado de la salud
- Seguridad/vigilancia
- Recursos (p.ej. manejo de recursos hídricos)
- Automatización industrial
- Automatización de hogares, edificios
- Salud de alimentos, agricultura
- Vigilancia de seguridad
- Electrónica del consumidor
- Utilidades, petróleo y gas

Quince características esenciales

La característica fundamental de IoT es la conectividad ubicua. La siguiente lista incluye otras funcionalidades o características esenciales:

- Monitorización y Control: se trata de funcionalidades fundamentales de las aplicaciones IoT, donde la monitorización tiene más presencia que el control.

- Servicios de localización: basados en GPS u otras tecnologías de localización como RTLS.
- Alerta: alerta basada en eventos, a veces son entradas de un motor basado en reglas para inferir las acciones a llevar a cabo.
- Planificación y Despacho: características de planificación y despacho basadas en tiempo y en eventos.
- Mantenimiento y reparación: el mantenimiento soporta monitorización remota, actualización de *software*, etc.
- Seguridad: se requiere un *framework* de seguridad para soportar control de acceso, privacidad, etc.
- Informes, panel de indicadores: informes, tendencias y panel de indicadores. Se utilizan para una mejor gestión y toma de decisiones.
- Minería de Datos, soporte de decisión: análisis de los datos basado en algoritmos de *Business Intelligence* (BI) y minería de datos para ofrecer asistencia en la toma de decisiones.
- Gráficos: visualización gráfica de datos dinámicos, estado del equipamiento y así sucesivamente para las cosas del mundo real.
- Servicios: todo tipo de servicios, tales como servicios de postventa de equipamiento, vehículos y otros.

Cuatro pilares tecnológicos

1. RFID: IoT surge con la identificación por radiofrecuencia. Es una tecnología que permite convertir cosas inertes en elementos identificables y localizables a través de la instrumentación. También puede utilizarse como medio de identificación para otras aplicaciones, su uso es ilimitado.
2. Redes de sensores inalámbricas: son las terminaciones nerviosas en la última milla de IoT. La información detectada suele ser recopilada en un *gateway* M2M y enviada al *backend* para su tratamiento, almacenamiento, análisis y representación. Algunos sistemas WSN pueden ser autónomos.
3. M2M: esta área se refiere a las tecnologías de conectividad de dispositivos, productos y servicios relevantes a las redes inalámbricas celulares operadas por compañías de telecomunicaciones.
4. SCADA: es un sistema autónomo basado en teoría de control de lazos o un sistema inteligente o un CPS que conecta, monitoriza y controla equipamiento a través de la red (habitualmente se utiliza una red cableada de corto alcance) en una instalación como una planta o un edificio.

Tres capas de sistemas IoT

1. *Devices*: incluyen dispositivos inteligentes ubicuos (terminales M2M, sensores WSN, actuadores SCADA, etc.) y objetos inertes que pueden ser instrumentados con la tecnología RFID.

2. *Connect*: incluye medios de telecomunicación cableados e inalámbricos tanto de largo como de corto alcance.
3. *Manage*: se trata de aplicaciones integradas que están basadas en el *middleware* y en la computación en la nube del *backend*.

Enablers tecnológicos fundamentales

1. EPC: todas las tecnologías de codificación e identificación como EPC, UID, UUID y otras.
2. Tecnologías de Sensores: todo tipo de sensores capaces de generar datos sin intervención humana.
3. Terminales móviles: todo tipo de dispositivos móviles que se comunican a través de redes de telecomunicaciones.
4. Actuadores y controladores: PLC, RTU, DCS y otros que se conectan a través de buses de campo.
5. HMI: tecnologías de interfaz hombre-máquina que incluyen paneles gráficos de control.
6. Tecnologías web: incluyen tecnologías de navegador/ HTML5 en todo tipo de dispositivos.
7. Tecnologías de la nube: tecnologías de *backend* basadas en SPI, *multitenancy*.
8. Servicios Web/SOA: para una gran integración de B2B/ B2C a través de Internet, así como de intranet y extranet.
9. XML: proporciona medios universales de representación de datos.
10. Middleware: todo tipo de *middleware* para un *framework* unificado de IoT.
11. *Networking*: proporciona conectividad ubicua.
12. GPS: ofrece servicios de localización.
13. GIS/POI: ayuda a proporcionar servicios de localización y navegación.
14. ERP/ MES: receptores y usuarios de datos IoT, parte de la gran integración de IoT.

2.7. FIWARE

El proyecto FIWARE está siendo desarrollado como parte del programa *Future Internet Public Private Partnership* (FI-PPP) lanzado por la Comisión Europea en colaboración con la industria de las Tecnologías de la Información y Comunicaciones (TIC).

El principal objetivo de FIWARE es llegar a ser la plataforma central del *Future Internet* (FI). Por este motivo, su arquitectura es abierta y basada en elementos denominados *Generic Enablers* (GEs) que ofrecen funciones reutilizables y comúnmente compartidas por múltiples áreas de uso a través de diversos sectores. Además, dicha arquitectura (Figura 2.20) es muy extensa y compleja estructurándose en una serie de capítulos técnicos, a saber:

- *Cloud Hosting*: es la capa fundamental que proporciona los recursos de computación, almacenamiento y red, sobre la cual los servicios se aprovisionan y gestionan.
- *Data/Context Management*: ofrece los servicios para el acceso efectivo, el procesamiento, la clasificación semántica y el análisis de flujos masivos de datos para obtener un conocimiento valioso.

- *Service Delivery Framework*: constituye la infraestructura para crear, publicar, gestionar y consumir servicios FI a través de su ciclo de vida, abordando todos los aspectos técnicos y de negocio.
- *Internet of Things Services Enablement*: construye el puente mediante el cual la interfaz de los servicios FI maximiza la ubicuidad de dispositivos de recursos limitados y heterogéneos en el ámbito de la IoT.
- *Internet to the Network and Devices*: ofrece interfaces abiertas para redes y dispositivos, proporcionando las necesidades de conectividad de los servicios entregados a través de la plataforma.
- *Security*: proporciona los mecanismos que aseguran que la entrega y el uso de los servicios es confiable y cumple con los requisitos de seguridad y privacidad.

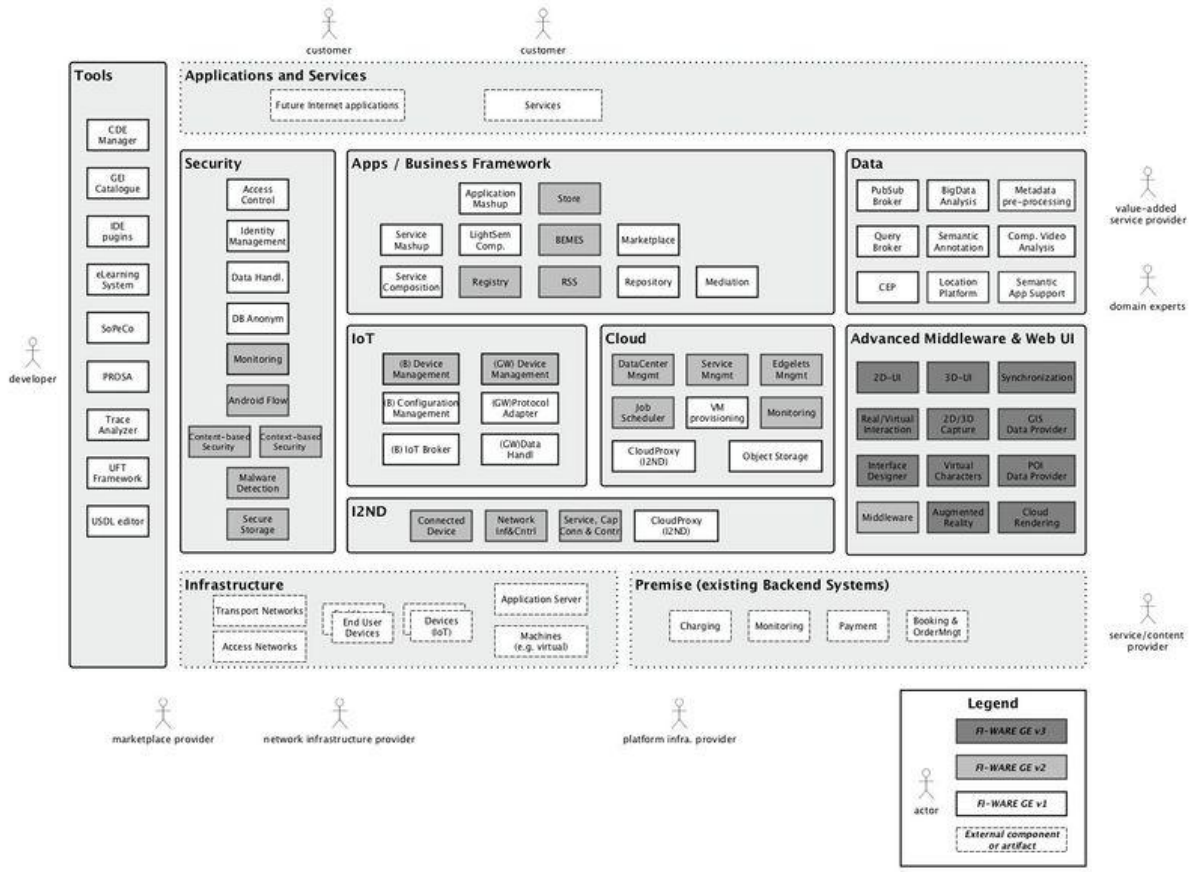


Figura 2.20. Arquitectura de la plataforma FIWARE.

Una vez introducido el alcance de la plataforma FIWARE, en el capítulo 4 se profundiza en aquellos capítulos técnicos que proveen los GEs que facilitan el desarrollo de aplicaciones en el ámbito de la IoT.

2.8. Agricultura de precisión

La agricultura de precisión (PA) consiste en la introducción de nuevas tecnologías en el contexto de la producción agrícola que permiten la monitorización de las diferentes variables que afectan al proceso de crecimiento del cultivo (suelo, agua y planta) y en la aplicación de algoritmos específicos para tomar las decisiones que permiten optimizar la producción.

La agricultura inteligente (*smart agriculture*) se compone de muchas implementaciones tecnológicas diferentes. Estas aplicaciones están reemplazando las técnicas agrícolas tradicionales difíciles, poco fiables y que consumen mucho tiempo por una agricultura inteligente eficiente, confiable y sostenible. Algunos ejemplos son el riego de cultivos sensible al contexto, control de pesticidas, control remoto, control de seguridad, monitorización ambiental, control de procesos y máquinas, guiado de vehículos, sistemas de trazabilidad, etc.

El tamaño reducido de un nodo sensor lo convierte en uno de los principales dispositivos electrónicos para automatizar diversas actividades relacionadas con la agricultura. Aunque se trata de dispositivos limitados, esta tecnología es extremadamente útil para diferentes áreas de aplicación, incluyendo el sector agrícola. En la agricultura, los nodos sensores se utilizan para recopilar parámetros ambientales y propiedades del suelo que tienen efectos drásticos sobre los rendimientos de los cultivos, la calidad, el coste y los plazos de riego. Los programas de riego y los rociadores de pesticidas o fungicidas dependen completamente de los parámetros ambientales. Los nodos sensores permiten proporcionar esta información regularmente. Los parámetros del suelo son extremadamente útiles para determinar la adecuación de los cultivos e indican el estado hídrico en el que se encuentran. Además, las WSN se utilizan en diferentes proyectos agrícolas que van desde la automatización de invernaderos a despliegue de campo abierto (Figura 2.21).

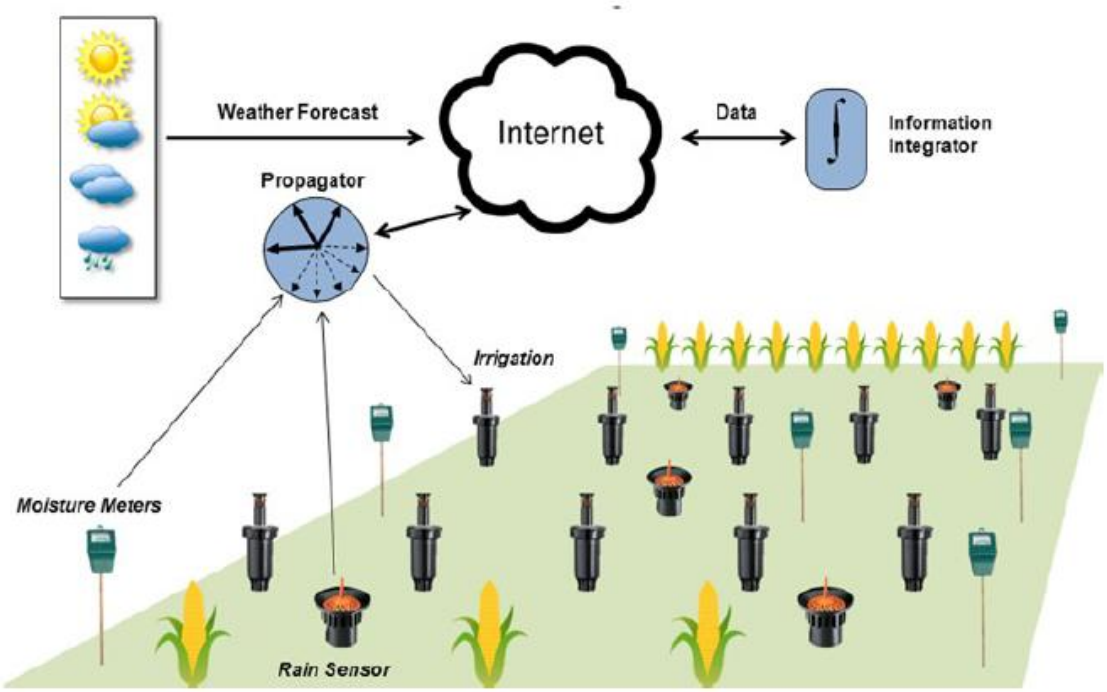


Figura 2.21. Aplicación sensible al contexto (WSN desplegada en campo abierto) [15].

Las redes de sensores inalámbricas (WSN) representan una de las tecnologías más prometedoras para la agricultura de precisión. En los próximos años, se espera un aumento significativo en el uso de estos sistemas en las granjas comerciales. Las WSN presentan una serie de problemas, relacionados con la escalabilidad, la interoperabilidad, las

comunicaciones, la conectividad con las bases de datos y el procesamiento de datos. Diferentes *middleware* de Internet de las Cosas están apareciendo para superar estos desafíos. En esta tesis se comprueba si uno de estos *middleware*, FIWARE, es adecuado para el desarrollo de aplicaciones agrícolas.

CAPÍTULO 3

Metodología de Investigación

We cannot solve our problems with the same thinking we used when we created them.

Albert Einstein (1879- 1955)

3.1. Tipo de estudio

El trabajo de investigación realizado durante el desarrollo de esta tesis puede considerarse una mezcla equilibrada entre investigación cuantitativa, cualitativa y experimental. Desde la especificación de los objetivos de la tesis se ha seguido un plan de actuación planificado para llevarse a cabo durante los años de realización del trabajo.

Las tareas llevadas a cabo han sido de base tecnológica y estrechamente ligadas con la innovación en las tecnologías de la información y las comunicaciones. La incorporación del conocimiento científico y tecnológico al proceso de desarrollo de los objetivos planteados al inicio del trabajo, persigue la obtención de un resultado que cumpla un fin valioso y necesario.

3.2. Situación inicial

Antes de iniciar esta investigación, era evidente el compromiso de Europa para hacer frente a los retos de Internet del Futuro. De hecho, este compromiso se ha materializado en importantes inversiones en proyectos de investigación, muchos de los cuales aún siguen activos, en los que participaban diferentes disciplinas científicas. Además, se prevé que estas inversiones sigan produciéndose. Internet del Futuro pretende integrar tecnologías de comunicación heterogéneas con objeto de contribuir al fenómeno conocido como Internet de las Cosas. Dicho fenómeno se puede definir como una red mundial de objetos interconectados que incluye, entre otros, ordenadores, teléfonos móviles, dispositivos RFID y redes de sensores inalámbricas. Estas redes serán capaces de trabajar dinámicamente para colaborar eficientemente en el logro de diferentes tareas. Las redes de señores inalámbricas consisten de un determinado número de nodos sensores que se comunican enviando y procesando información para cooperar entre ellos logrando un objetivo global. El bajo coste de esta tecnología ha facilitado su proliferación en muchos escenarios que requieren la monitorización del entorno.

3.3. Plan de investigación

A partir de la situación inicial planteada, el plan de investigación contempla la aplicación de Internet de las cosas en el ámbito de la agricultura inteligente, la sostenibilidad medioambiental y la optimización en el uso de recursos, especialmente hídricos. El uso de redes de sensores en el sector agrícola permite la monitorización de parámetros como la temperatura, humedad, luz, etc. Dicho interés es especialmente elevado en la Región de Murcia, en la que como señala el Plan Estratégico para los próximos años, se trata de un sector de gran importancia dado que la agricultura intensiva de regadío y el conjunto de la industria alimentaria generan el 8,5% del VAB de la CARM.

La complejidad de las redes de sensores crece exponencialmente con su tamaño, de forma que para la realización de proyectos de monitorización de cierta envergadura, es necesario utilizar plataformas de gestión de los dispositivos que formen parte de la red de sensores. Estas plataformas deben exhibir una serie de características, principalmente:

1. Deben permitir la interconexión de dispositivos heterogéneos.
2. Deben ser escalables con el tamaño de la red.
3. Deben permitir el registro y búsqueda de dispositivos y servicios.
4. Deben ser tolerantes a fallos.
5. Deben incluir utilidades para la estimación o interpolación de datos en aquellas áreas no cubiertas por los sensores.

Actualmente, estas plataformas están en fase de investigación y desarrollo y, aunque ya existe alguna implantada comercialmente, no se han realizado estudios metódicos sobre su utilidad y eficiencia.

3.3.1. ***Metodología***

La aplicación de una metodología de investigación bien establecida durante el proceso de desarrollo de cualquier trabajo de investigación facilita su estructuración, correcto desarrollo y finalización. En el caso de esta tesis, la metodología utilizada ha sido el método científico.

La aplicación del método científico es una metodología básica y presente en muchos de los trabajos de investigación actuales. En dicho método se parte de una cuestión bien formulada a la cual se trata de dar respuesta a lo largo de la investigación mediante la observación sistemática, la experimentación y el planteamiento de una hipótesis. El testeo y la experimentación alrededor de dicha hipótesis permite al investigador obtener gran cantidad de información que ayude a verificar o refutar la cuestión de partida. A menudo, este proceso iterativo implica incluir cambios en la hipótesis derivados de las conclusiones obtenidas a raíz de los resultados de los testeos. La modificación o reformulación no resulta en absoluto negativo para la investigación, sino que ayuda a arrojar luz sobre la cuestión de partida generando conocimiento y dando cuerpo a los posibles resultados de la tesis en la cual se engloban. En nuestro caso, la cuestión de partida de la investigación ha estado claramente definida desde sus inicios.

> "Existe una gran cantidad de *middleware* de Internet de las Cosas en fase de investigación y desarrollo. Sin embargo, debido a la extensión de este paradigma, cada *middleware* aborda una serie de desafíos tecnológicos y permite el desarrollo de aplicaciones inteligentes en determinados dominios. Por tanto, en la actualidad no existe un *middleware* ampliamente aceptado. En determinados dominios de aplicación se está extendiendo la utilización de redes de sensores inalámbricas. Estas redes permiten detectar enormes volúmenes de datos del entorno y, por tanto, ofrecen la posibilidad de implementar aplicaciones o sistemas sensibles al contexto. En el caso de la agricultura de precisión, la evolución natural de estos sistemas sensibles al contexto es monitorizar y controlar escenarios densamente poblados por sensores, que miden parámetros del suelo, por ejemplo humedad o concentración de nutrientes, o de la planta, por ejemplo estrés hídrico. Capturar, transmitir, almacenar y procesar este volumen de información sin precedentes resultado del despliegue de redes de sensores inalábricas en este novedoso tipo de escenarios se traduce en una serie de desafíos, especialmente en cuanto a tecnologías de integración, comunicaciones, bases de datos y computación. Entonces, como ingenieros *software*, ¿existe un *middleware* de IoT que permita el desarrollo de aplicaciones inteligentes sensibles al contexto para el tipo de escenario descrito en el ámbito de la agricultura de precisón?

Considerando esta hipótesis de partida, se inicia el siguiente proceso investigador:

En primer lugar, como se indica en el capítulo 1, se realiza un estudio de Internet de la Cosas atendiendo al concepto y a las diferentes perspectivas de este paradigma. Posteriormente, el capítulo 2 refleja la revisión del estado del arte, partiendo de las redes de sensores y, específicamente, de las redes de sensores inalámbricas. En dicho estudio, se extraen los desafíos que aporta esta tecnología. El siguiente paso ha sido estudiar otras tecnologías de Internet de las Cosas. Para ello, se han estudiado diferentes modelos: físico, topológico y lógico. A partir del mismo, se han extraído tanto los desafíos propios de IoT como aquellos que aportan las diferentes tecnologías (detección, comunicación y gestión). Además, se ha analizado estado del arte relativo a los diferentes *middleware* de IoT y sus características. El *middleware* ha sido el actor principal durante este trabajo de investigación, prueba de ello es el estudio de los diferentes modelos y API de comunicación así como el estudio del enfoque de arquitectura orientada a servicios. Actualmente, este enfoque es el más utilizado en el desarrollo de *middleware* de IoT debido a que facilita la integración de las diferentes aplicaciones y servicios. El mayor aporte de las tecnologías de la información y comunicaciones a Internet de las Cosas es la computación en la nube. Finalmente, ha sido necesario abordar el dominio de la agricultura de precisión y, concretamente, las características de los sistemas de gestión agrícolas (*Farm Management Systems*) y la integración de las redes de sensores inalámbricas en estos sistemas.

En segundo lugar, se estudian candidatos *middleware* de IoT que puede responder a la cuestión formulada. Se opta por estudiar la arquitectura IoT de la plataforma FIWARE. A continuación, se concibe un caso de estudio IoT para el hipotético escenario en el dominio de la agricultura de precisión. El resultado es un sistema de gestión agrícola basado en la tecnología FIWARE. Este proceso se describe en el capítulo 4.

En tercer lugar, se han identificado parte de los requisitos de diseño del banco de pruebas a partir del sistema concebido. Por ello, se procede al diseño, implementación, instalación, configuración y despliegue del banco de pruebas en entorno de laboratorio, tal y como se explica en el capítulo 5. Cabe mencionar que este proceso es iterativo, identificando problemas y aportando soluciones en todas las fases que conducen al banco de pruebas perseguido.

En cuarto lugar, se confecciona un plan de pruebas para para evaluar el rendimiento del *middleware* utilizado (FIWARE) y así poder determinar su viabilidad para el desarrollo de aplicaciones inteligentes sensibles al contexto en el ámbito de la agricultura de precisón. Tanto el plan de pruebas como el análisis de los resultados obtenidos en las simulaciones son presentados en el capítulo 6.

Finalmente, se extraen las conclusiones y se proponen líneas futuras de investigación, reflejadas en el capítulo 7.

3.4. Desarrollo de la tesis

El trabajo de investigación y los resultados recogidos en esta tesis se han llevado a cabo siguiendo una planificación estructurada en base a los objetivos planteados inicialmente. Para ello, las tareas que se han incluido en el trabajo a realizar durante este tiempo han estado ligadas a un proceso de desarrollo e investigación iterativo, en el que el análisis de trabajos relacionados o búsqueda bibliográfica, el planteamiento de modelos de base teórica, la implementación de las teorías planteadas y la realización de pruebas y evaluación de resultados ha sido un denominador común. Y por último, y como ocurre en todo trabajo de investigación, la publicación de resultados ha formado parte de todas y cada una de dichas tareas.

Parte II

Contribución de la Tesis

CAPÍTULO 4

FIWARE

Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.

Mark Weiser (1952- 1999)
The Computer for the 21st Century.

4.1. Descripción

El proyecto FIWARE está siendo desarrollado como parte del programa *Future Internet Public Private Partnership*[6] lanzado por la Comisión Europea[7] en colaboración con la industria de las tecnologías de la información y comunicaciones.

FIWARE está diseñado para ser el núcleo del *Future Internet* cubriendo todos sus dominios de aplicación. Por este motivo, la arquitectura de FIWARE es muy extensa, estructurándose en siete capítulos técnicos [110], completamente abierta y basada en componentes, denominados *Generic Enablers* [111] [112], que ofrecen funciones reutilizables y configurables a través de varios dominios de aplicación.

En este capítulo, se revisan aquellos aspectos de FIWARE que facilitan la comprensión de este trabajo. Para ello, empezaremos introduciendo conceptos relativos al modelado y a la gestión de la información. Posteriormente, se da una visión general de la arquitectura IoT de FIWARE.

4.2. FIWARE NGSI Context Management

La especificación FIWARE se basa en el estándar *Open Mobile Alliance Next Generation Service Interfaces* (OMA NGSI) *Context Management* [113] para gestionar e intercambiar información.

4.2.1. ***Modelo de información de contexto***

Toda la información que manejan las aplicaciones se modela a través de la definición de un modelo de información de contexto compuesto de *elementos de contexto* a los que nos vamos a referir como *entidades*.

[6] Future Internet Public Private Partnership (FI-PPP): http://www.fi-ppp.eu
[7] European Commision: http://ec.europa.eu

Las *entidades* pueden hacer referencia a *objetos físicos* (p.ej.: sensores o actuadores) a *recursos* hardware y software (p.ej.: un programa IoT) y a abstracciones de alto nivel que modelan *cosas* y que agrupan tanto a objetos físicos como a *recursos*.

Las *entidades* se representan mediante estructuras de datos genéricas como se muestra en la Figura 4.1. Las *entidades* se identifican de forma unívoca mediante un par *EntityId/EntityType* de acuerdo a la recomendación W3C [114]. Las *entidades* tienen *atributos*, caracterizados por un triplete (<nombre_atributo, tipo, valor>). Y cada atributo puede tener datos semánticos opcionales (metadatos) caracterizados también por tripletes <nombre_metadato, tipo, valor>. Los valores de los atributos se convierten en la información de contexto de un sistema. Un concepto fundamental es que las *entidades* no están ligadas a un formalismo de representación específica. Por ejemplo, éstas pueden representarse como un documento XML (SensorML, ContextML, etc) o como entradas en una base de datos NoSQL (MongoDB). Una ventaja clave de este modelo conceptual es su compatibilidad con los formatos IoT (SensorML), al tiempo que permite extensiones adicionales.

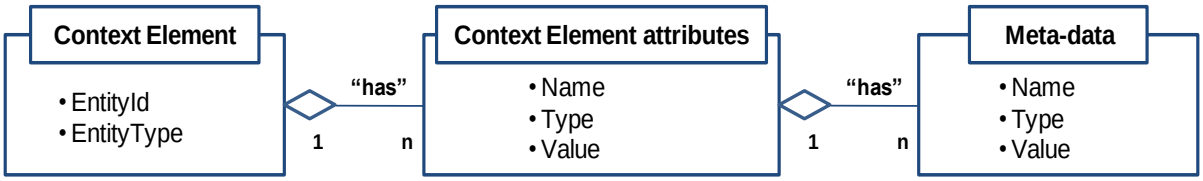


Figura 4.1. Diagrama conceptual de un elemento de contexto.

4.2.2. ***Eventos***

En sistemas IoT basados en FIWARE, los *eventos* se refieren a algo que ha sucedido en el sistema. Los cambios en la información de contexto, tanto la actualización de *entidades* ya existentes como la creación de otras nuevas, son considerados como *eventos*. FIWARE es una arquitectura dirigida por *eventos* y los *Generic Enablers* se definen teniendo en cuenta este hecho [115].

4.2.3. ***FIWARE NGSI API***

Las interfaces OMA NSGI-9 y NSGI-10 son APIs RESTful HTTP [116]. Las versiones FIWARE de dichas APIs permiten la mediación entre un gran número de servicios. La interfaz NGSI-9 permite hacer accesible la información de contexto producida por una entidad a otras *entidades*. Para ello provee, entre otras, dos operaciones: *registerContext* para registrar las *entidades* que publican sus datos y *discoverContextAvailability* para descubrirlas.

Por otro lado, la API NGSI-10 permite intercambiar la información de contexto en sí misma. Para ello ofrece, entre otras, las operaciones *updateContext* para actualizar información de contexto y *subscribeContext* para crear suscripciones a información de contexto.

4.2.4. ***Asociaciones FIWARE NGSI***

Un tipo especial de *entidades* son las *cosas*. Las *cosas son abstracciones del mundo real* y suelen agrupar *recursos* atendiendo a su ubicación y al tipo de sus atributos.

Una *asociación* [116] es un par ordenado entre una *entidad* a nivel de *dispositivo*/*recurso* y una *entidad* a nivel de *cosa*. Su finalidad es habilitar una transición desde la información a nivel de *dispositivo*/*recurso* hasta la información a nivel de *cosa* y viceversa. Así, las *asociaciones* describen relaciones entre estos dos niveles de abstracción.

4.2.5. ***Dominios de Atributo***

Un dominio de atributo es un conjunto de atributos de contexto estrechamente relacionados. Cada atributo tiene un nombre y pertenece a un solo dominio. Utilizar un dominio en una operación NGSI es muy útil porque los atributos en ese dominio son siempre solicitados, provistos, actualizados o almacenados al mismo tiempo. Este hecho implica que la creación y la actualización de datos dentro de un dominio son atómicas y que los datos de contexto asociados a sus atributos son siempre coherentes.

4.3. Arquitectura IoT

La arquitectura IoT de FIWARE (Figura 4.2) es el resultado de la composición de los *Generic Enablers* de los capítulos *IoT Services Enablement* [117] y *Data/Context Management* [118]. Se trata de una arquitectura flexible que permite abordar los desafíos mencionados en el capítulo 2. Desde un punto de vista físico, el capítulo *IoT Services Enablement* define tres dominios: (1) dispositivos y recursos IoT, (2) *gateways* IoT y (3) el IoT *Backend*, que se describen a continuación:

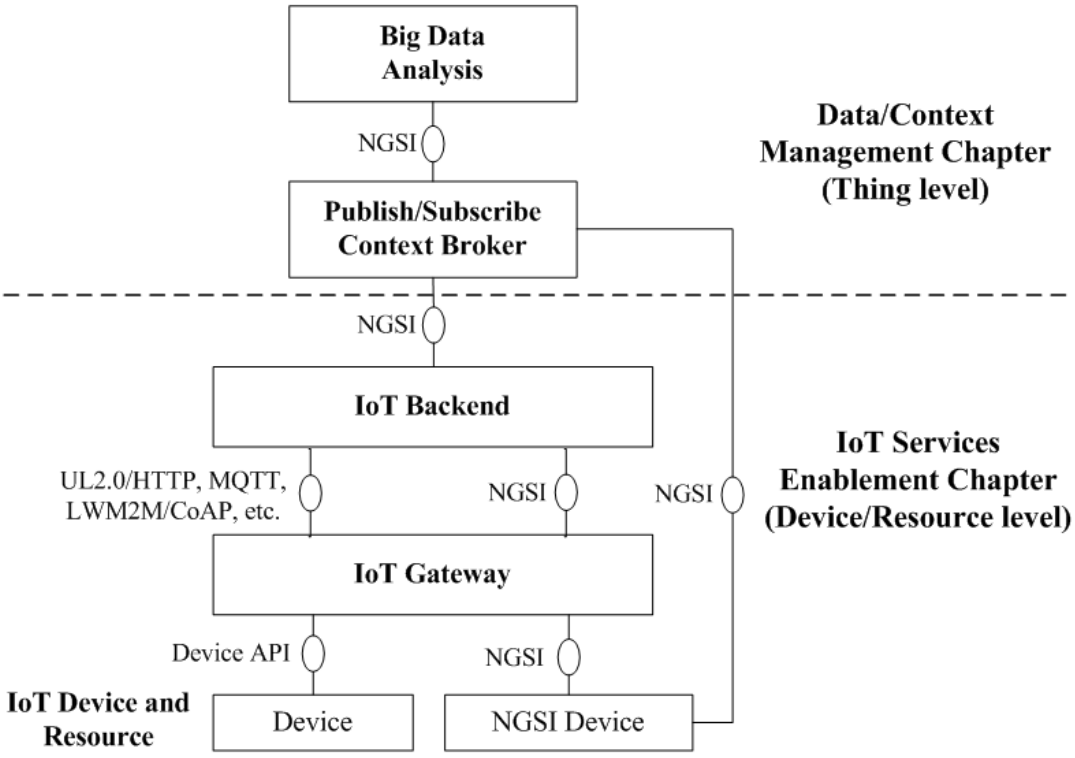


Figura 4.2. Arquitectura IoT de FIWARE.

4.3.1. ***Dispositivo y Recurso IoT.***

Un *dispositivo* es un componente *hardware* que mide propiedades de su entorno o actúa sobre dicho entorno. Así, los sensores y actuadores son dispositivos. Por otro lado, los *recursos* IoT son elementos computacionales (*software*) que permiten a un dispositivo llevar a cabo sus tareas de detección y/o actuación. Generalmente, un *recurso* IoT se aloja en el *dispositivo*.

En FIWARE se consideran dos tipos de dispositivos, según integren o no el *middleware* FIWARE NGSI. Los dispositivos que no integran el *middleware* FIWARE NSGI no disponen de los recursos computacionales requeridos para proporcionar servicios NGSI.

4.3.2. ***IoT Gateway.***

Los *gateways* concentran la información de los dispositivos finales y la encaminan hacia los consumidores. Se trata de dispositivos *hardware* que, generalmente, se encuentran en la proximidad de los dispositivos finales (sensores/actuadores). Los *gateways* además proporcionan recursos computacionales a los dispositivos dentro de su área de influencia. Como en el caso de los dispositivos, hay que diferenciar entre dos tipos de *gateway* en función de si alojan o no el *middleware* FIWARE. Los *gateways* que no alojan FIWARE pueden integrarse en el sistema conectándose a *gateways* FIWARE. Los gateways FIWARE alojan dos GEs: el *Protocol Adapter* y el *Data Handling* [112] [117]. El Protocol Adapter abstrae los protocolos de comunicaciones usados por los dispositivos (p.ej. Zigbee, CoAP) mediante su traducción al protocolo FIWARE NGSI. El *Data Handling* es la primera etapa de inteligencia destinada a procesar datos en tiempo real. Para ello, recoge información de los Protocol Adapter que recopilan datos brutos de los sensores, transformándolos en eventos relevantes y finalmente propagándolos hacia el IoT *Backend*.

4.3.3. ***IoT Backend.***

Consiste en un entorno en la nube que hospeda uno o varios de los siguientes *Generic Enablers*: IoT *Discovery*, IoT *Broker* y *Backend Device Management* [112] [118]. El IoT *Discovery* permite el descubrimiento de dispositivos/recursos, después del registro/anuncio de su disponibilidad (analogía: "páginas amarillas"). Además, almacena las *asociaciones* entre los *dispositivos/recursos* y las *cosas* de un sistema IoT.

El IoT *Broker* interactúa con el *Publish/Subscribe Context Broker* GE (Figura 4.2) y con el *Data Handling* GE infiriendo el estado de las *cosas* a partir del de los *dispositivos/recursos* o viceversa gracias a la comunicación con el IoT *Discovery*. Finalmente, el *Backend Device Management* permite gestionar las *entidades* de contexto asociadas a los *dispositivos* y proporciona varios traductores de protocolos de comunicación (agentes IoT) como Ultralight2.0 HTTP, MQTT u OMA LWM2M/IETF CoAP, entre otros.

El IoT *Backend* se comunica con los dispositivos finales mediante protocolos estándares (MQTT, CoAP, etc.) o a través del protocolo FIWARE NGSI, conectándose ya sea a través de *gateway*(s) IoT y/o interfaces directas.

Desde un punto de vista más abstracto, por encima de las *entidades* físicas y los *recursos* computacionales, el capítulo *Data/Context Management* define los siguientes *Generic Enablers*:

4.3.4. ***Publish/Subscribe Context Broker GE.***

La misión de este GE es conseguir un desacoplamiento total entre los productores y los consumidores de información de contexto. Para ello, implementa el patrón de diseño *Publish/Subscribe* [70]. Los productores de contexto (*dispositivos*, *recursos* y *cosas*) publican sus datos en este GE sin necesidad de conocer a los consumidores de tales datos. Los consumidores de información de contexto tampoco necesitan conocer a los productores: están simplemente interesados en el *evento* en sí, pero no en quién lo generó.

Como resultado, el *Publish/Subscribe Context Broker* [112] [118] (*Context Broker* en adelante) es un excelente puente permitiendo a las aplicaciones externas gestionar eventos relacionados con el Internet de las cosas de una manera muy simple ocultando toda la complejidad de la recopilación de la información. Para ello, el *Context Broker* expone las interfaces estándar NGSI-9 yNGSI-10 que habilitan la recuperación de dicha información desde los productores de contexto (p.ej. *IoT Backend* o dispositivo NGSI) hasta los consumidores de contexto (p.ej. *Big Data Analysis* GE).

4.3.4.1. **Arquitectura**

La Figura 4.3 ilustra la arquitectura lógica del *Context Broker* en la que aparecen sus principales interacciones con otros actores:

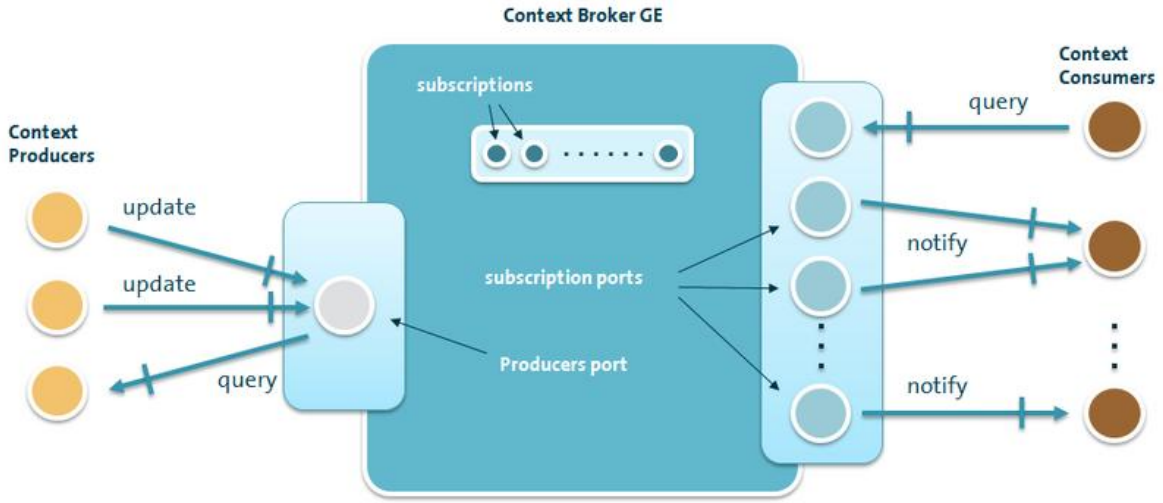


Figura 4.3. Arquitectura del *Context Broker*.

Los actores de este modelo de comunicación son:

- **Context Broker (CB)**: es el componente bajo estudio. Funciona como un manejador y agregador de datos de contexto y como una interfaz entre los actores de la arquitectura. Principalmente, el CB controla el flujo de contexto entre todos los actores. Por este motivo, el CB tiene que saber acerca de todos los *Context Providers* (CPr). Esta característica se realiza a través de un proceso de anuncio.

- **Context Producer:** un *Context Producer* (CP) es un actor capaz de generar contexto. El *Context Producer* básico es el que actualiza información de contexto de forma espontánea, sobre uno o más atributos de contexto de acuerdo con su lógica interna. Esta comunicación entre CP y CB se realiza en modo *push*, del CP al CB. Los CP también pueden trabajar en modo *pull*, en cuyo caso se denominan *Context Providers* (CPr).
- **Context Provider (CPr)**: es un tipo especializado de actor *Context Producer*, que proporciona información de contexto a demanda, en modo síncrono. Esto significa que el *Context Broker* o incluso el *Context Consumer* puede invocar el CPr con el fin de adquirir la información de contexto. Un CPr proporciona datos de contexto sólo a raíz de una invocación específica. Por otra parte, un CPr puede producir nueva información de contexto a partir del cálculo de los parámetros de entrada; por lo tanto es muchas veces responsable del razonamiento en la información de contexto de alto nivel y de la fusión de datos del sensor. Cada CPr registra su disponibilidad y capacidades mediante el envío de los anuncios correspondientes al CB y expone interfaces para proporcionar información de contexto al CB y a los *Context Consumers*.
- **Context Consumer**: un *Context Consumer* (CC) es una entidad (por ejemplo, una aplicación basada en contexto) que explota la información de contexto. Un CC puede recuperar información de contexto enviando una solicitud al CB o invocando directamente al CPr a través de una interfaz específica. Otro camino para que el CC obtenga información es mediante la suscripción a actualizaciones de información de contexto que responden a ciertas condiciones (por ejemplo, están relacionados con cierto conjunto de entidades). El CC registra una operación *call-back* con la suscripción, por lo que el CB notifica al CC acerca de relevantes actualizaciones en el contexto invocando esta función *call-back*. Por último, una especie de CC puede exponer operaciones de actualización de contexto a ser invocadas por el CB. Esto está principalmente relacionado con las capacidades de actuación.
- **Entity**: cada intercambio de datos de contexto se refiere a una entidad específica, que puede ser en su orden un grupo complejo de más de una entidad. Una *entity* es el tema (p. ej. usuario o grupo de usuarios, cosas o grupo de cosas, etc.) al que hacen referencia los datos de contexto. Se compone de dos partes: un tipo y un identificador. Cada *Context Provider* soporta uno o más tipos de entidad y esta información se publica al *Context Broker* durante el proceso de anuncio.
- Un tipo es un objeto que clasifica un conjunto de entidades, por ejemplo dispositivos móviles – identificados por imei.

Así, el *Context Broker* habilita la publicación de la información de contexto por parte de las *entities* (denominadas *Context Producers*), para que la información de contexto publicada esté disponible para otras *entities* (denominadas *Context Consumers*). En la plataforma FIWARE, aplicaciones o incluso otros GEs pueden desempeñar el rol de *Context Producers*, *Context Consumers* o ambos.

El principio fundamental soportado por el *Context Broker* es el de conseguir total disociación entre los *Context Producers* y los *Context Consumers*. Esto es posible gracias a la

implementación del Patrón de Diseño *Publish/Subscribe* (sección 2.5.1). Este hecho permite que los *Context Producers* publiquen datos sin el conocimiento de quién, cuándo o dónde serán consumidos por los *Context Consumers*, por lo que no es necesaria una comunicación directa entre ellos. Por otro lado, los *Context Consumers* pueden consumir información de su interés independientemente de las publicaciones que realicen los *Context Producers*. Esto es debido a que los *Context Consumers* están interesados en el tipo de información y no en quién publica dicha información. Como resultado, el *Context Broker* es un excelente puente que permite a aplicaciones externas manejar eventos relacionados con el Internet de las Cosas de una manera simple.

4.3.4.2. **Modos de comunicación**

El *Context Broker* puede proporcionar acceso a la información de contexto al *Context Consumer* cuando éste lo requiera (*on-request*) o cuando dicha información esté disponible (*on-subscription*).

- **Modo on-request:** envía eventos a los *Context Consumers* en respuesta a las solicitudes realizadas por estos (CC).
- **Modo on-subscription:** permite la configuración de las condiciones necesarias para que los eventos sean enviados a los *Context Consumers*. En caso de que el evento cumpla con las condiciones establecidas en la suscripción, el CB invoca la operación de notificación (*notify*) hacia el CC. Las suscripciones pueden ser configuradas por terceras aplicaciones y no necesariamente por los CC en sí mismos.

4.3.5. ***Big Data Analysis GE (Cosmos).***

La meta de este *Generic Enabler* [118] es desplegar medios con los cuales analizar datos por lotes (*batch data*) con el fin de revelar nueva información. En este tipo de procesamiento, los datos por lotes son almacenados por adelantado y la latencia no es extremadamente importante cuando son procesados.

Actualmente, cabe señalar que el bloque de procesamiento por lotes (*batch processing block*) ha sido ampliamente desarrollado en Cosmos [112] (implementación de referencia del *Big Data Analysis*). Dicho bloque integra el ecosistema *Hadoop* [119] que incluye el paradigma *MapReduce*. Además, *Cygnus* [120] forma parte del ecosistema del *Big Data Analysis GE* (Cosmos). Se trata de un conector diseñado para proporcionar almacenamiento persistente de la información de contexto del *Context Broker* en repositorios externos. En otras palabras, el *Context Broker* solo almacena el último valor referente a los atributos de cada entidad de contexto. Por este motivo, si una aplicación IoT requiere el acceso a información de contexto histórica, es necesaria la persistencia en otro almacenamiento, valor por valor, utilizando *Cygnus*.

Internamente, *Cygnus* se basa en Apache Flume [121], una tecnología que aborda el diseño y la implementación de agentes de recolección y persistencia de datos. De hecho, *Cygnus* es un agente Flume que se compone de una fuente (*source*), un canal (*channel*) y un sumidero (*sink*). *SmartPort* es una plataforma basada en FIWARE que cubre una necesidad exigente en el ámbito del análisis *Big Data* [122].

4.3.5.1. **Arquitectura**

La Figura 4.4 ilustra la arquitectura de Cosmos. Este *Generic Enabler* es una expansión de Apache Hadoop [119]. En general, la instancia del *Big Data GE* consta de un *Master Node,* en el que se ejecuta un *software* de administración que permite la comunicación con los usuarios finales. Normalmente, estos usuarios son aplicaciones que utilizan la API de administración para enviar solicitudes de creación de nuevos *clusters* de almacenamiento y/o computación.

Por un lado, la creación de un *cluster* implica la creación de un *Head Node*, encargado de la gestión del almacenamiento, la computación y otros servicios como herramientas de análisis o inyectores de datos (*data injectors*). Por otro lado, son creados uno o más nodos esclavos (*Slave Nodes*), cuya misión es la ejecución de las tareas de análisis y del almacenamiento de los datos.

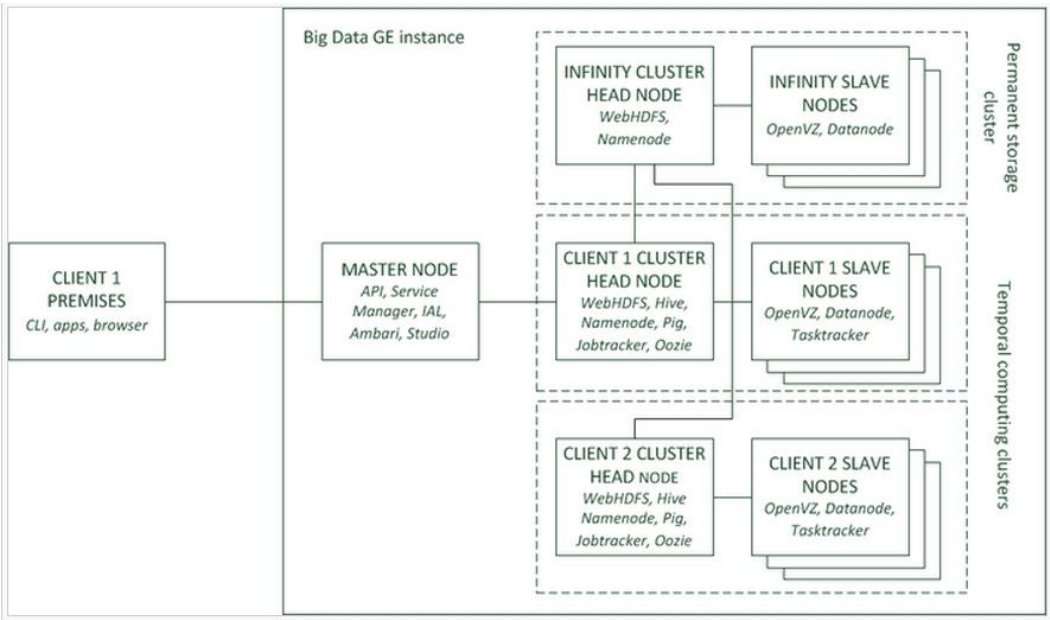


Figura 4.4. Arquitectura de Cosmos.

El enfoque de este *Generic Enabler* se basa en el almacenamiento y el análisis de los datos. Específicamente, los usuarios solo tienen que pensar en desarrollar aplicaciones sin preocuparse por otros aspectos internos como la paralelización, distribución, tamaño o escalabilidad.

4.3.5.2. **Bloque de procesamiento por lotes**

El bloque de procesamiento por lotes (*batch processing block*) ha sido ampliamente desarrollado a través de la adopción y/o la creación de las siguientes herramientas:

- **Hadoop As A Service (HAAS) engine**: ya sea la versión "oficial" basada en el *Openstack* de Sahara o la versión ligera basada en un *cluster* Hadoop compartido.
- **GUI Cosmos** y **OAuth2 Tokens Generator** para las APIs REST Cosmos.
- **Cygnus**, el conector de datos para *Context Broker*.
- **Tidoop**, extensiones para Hadoop incluyendo medios para utilizar algunos trabajos de propósito general MapReduce.

Este bloque pretende ser una plataforma de servicio Big Data en la parte superior de la infraestructura *Openstack*. Esta plataforma será la encargada de proporcionar, principalmente, *clusters* Apache Hadoop bajo demanda. También pueden ser consideradas otras soluciones de análisis basadas en *clusters* como Apache Spark.

Cosmos diferencia claramente entre servicios de computación y almacenamiento:

- El servicio de computación permite crear y configurar un *cluster* coordinado de máquinas de una manera rápida mediante línea de comandos o API REST. Además, el GE permite seleccionar un subconjunto de una amplia gama de tecnologías preconfiguradas de procesamiento y coordinación (Hadoop, Hive, Oozie, HBase, Sqoop, PIG, etc).
- El servicio de almacenamiento permite nutrir a los *clusters* con una gran variedad de datos sin ningún adaptador o configuración especial. La tecnología utilizada en *Cosmos* permite un alto *throughout* debido a que no se realizan traducciones de los datos. Para almacenar datos en el servicio de almacenamiento del GE es tan fácil como usar la línea de comandos o la API REST.

El uso de cualquiera de los servicios (computación o almacenamiento) no implica utilizar el otro. De esta manera, el almacenamiento de datos es independiente del ciclo de vida de datos asociado a su procesamiento. Además, este bloque enriquece el ecosistema Hadoop con herramientas específicas que permiten la integración de *Cosmos* con otros GE como por ejemplo el *Context Broker*.

4.3.5.3. **API HDFS RESTful**

Las API utilizadas en el presente trabajo han sido WebHDFS y HTTPFS:

- **WebHDFS:** existe una serie de comandos Hadoop para realizar operaciones de administración de los *clusters* y acceso a la información. No obstante, en ciertas ocasiones es necesario acceder a los recursos HDFS mediante una entidad externa. Para conseguir esto, Hadoop proporciona una API HTTP REST que soporta una interfaz completa para HDFS. Esta interfaz permite la creación, denominación y lectura de ficheros y carpetas, además de otras muchas operaciones con el sistema de ficheros.
- **HTTPFS:** aunque se trata de otra implementación de la API WebHDFS, ofrece una funcionalidad diferente. Cuando se usa WebHDFS, tanto el *Head Node* (*Namenode*) como los *Slave Nodes* (*Datanodes*) deben tener una IP pública. Esto es debido a que ambos tipos de nodos (*Namenode* y *Datanode*) deben de ser alcanzables. Este problema implica el uso de un alto número de direcciones IP públicas. HTTPFS actúa como un gateway entre el cliente HTTP y WebHDFS, y en vez de redireccionar hacia los *Slave Nodes*, vuelve a dirigir al propio *Head Node*. Ante tales facilidades de funcionamiento y utilización que encapsulan ciertos aspectos de infraestructura, esta es la especificación más utilizada.

4.3.5.4. **Sistemas de consulta**

- **Hive:** proporciona mecanismos de consulta a los datos mediante un lenguaje similar a SQL llamado HiveQL. Este sistema de consulta ha sido utilizado en el banco de pruebas.

- **Pig:** herramienta de consulta de datos similar a Hive. Apache Pig es una plataforma que consiste en la utilización de lenguajes de programación de alto nivel para el dessarrollo de *software* de análisis de datos. La propiedad más destacada de los programas Pig es que su estructura es compatible con la paralelización, permitiendo un mejor manejo de grandes cantidades de datos.

4.3.5.5. **Cygnus**

Cygnus es un conector encargado de persistir datos de contexto procedentes del *Context Broker* en almacenamientos de terceros, creando una vista histórica de tales datos. En otras palabras, el *Context Broker* sólo almacena el último valor en relación con el atributo de una entidad, comportándose como una base de datos en tiempo real. Si se desea guardar los datos de manera histórica para su posterior análisis es necesaria una herramienta que inyecte la información en otro almacenamiento que constituya una base de datos histórica. *Cygnus* es esta herramienta y permite inyectar en *Cosmos* la información de contexto del *Context Broker*.

Además, *Cygnus* utiliza las operaciones de suscripción/notificación de datos del *Context Broker*. Así, basta con realizar una suscripción en el *Context Broker* por parte de *Cygnus*, detallando cuáles son las entidades de las que se quiere ser notificado cuando se produzca una actualización de cualquiera de sus atributos.

Internamente, *Cygnus* está basado en Apache Flume. De hecho, Cygnus es un agente Flume, cuya arquitectura básica mostrada en la Figura 4.5 se compone de una fuente (*HttpSource*) a cargo de recibir los datos del *Context Broker*, un canal (*Memory Channel*) donde la fuente pone los datos una vez transformados en eventos Flume, y un sumidero (*OrionHDFS Sink*), que extrae los eventos Flume del canal para persistir los datos en un sistema de almacenamiento externo.

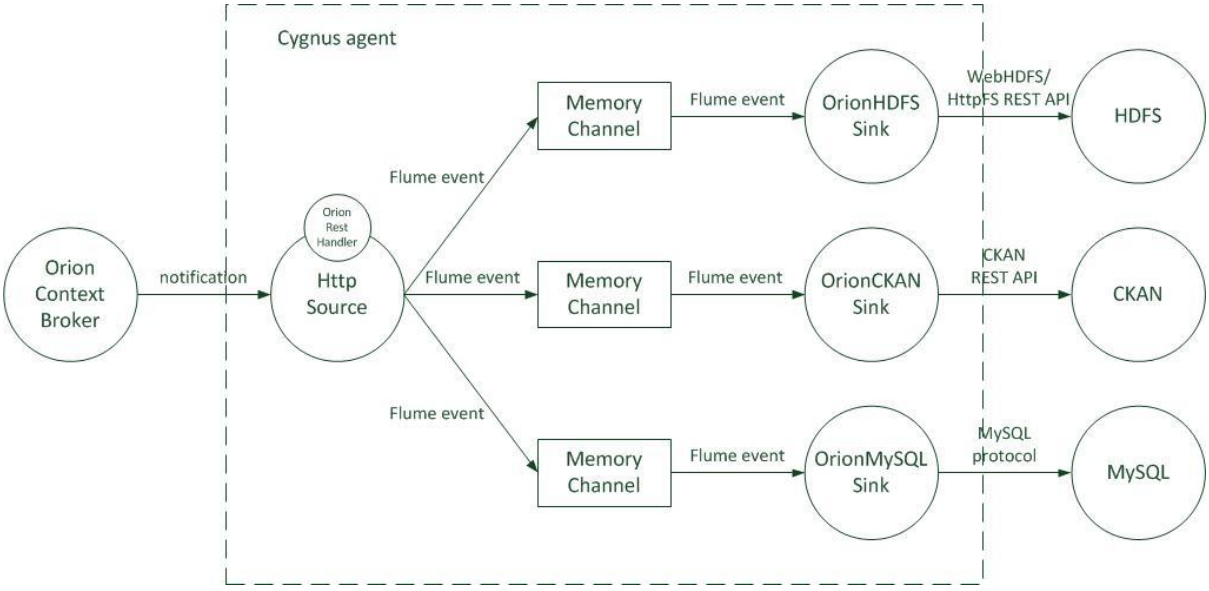


Figura 4.5. Arquitectura básica de Cygnus.

Cuando se intenta mejorar el rendimiento de Cygnus aparecen arquitecturas avanzadas. Como se puede observar en la Figura 4.6, la configuración básica de Cygnus consiste en una fuente que inyecta eventos Flume en un canal y de un sumidero que consume estos eventos. Esto puede ser claramente trasladado a una configuración múltiple de sumideros ejecutándose en paralelo donde aparece una variedad de nuevas configuraciones.

Una posible configuración es múltiples sumideros, un canal, sin embargo normalmente muestra una desventaja importante, especialmente si los eventos son consumidos por los sumideros muy rápido: los sumideros tienen que competir por el canal.

La desventaja mencionada anteriormente puede ser solucionada configurando un canal por cada sumidero, evitando la competición por el mismo, es decir: múltiples canales y sumideros. Sin embargo, cuando múltiples canales son utilizados para un mismo almacenamiento, entonces debe existir algún tipo de *dispatcher* que decida qué canales reciben una copia de los eventos. Además se establecen las siguientes condiciones:

- Se desea que los eventos sean replicados por cada tipo de almacenamiento configurado.
- Dentro de un tipo de almacenamiento, no se permite la replicación de los eventos Flume.
- Se desea que el criterio de *dispatching* esté basado en un comportamiento round-robin.

El selector *RoundRobinChannelSelector* cumple con estos requisitos. La Figura 4.6 ilustra la arquitectura avanzada descrita.

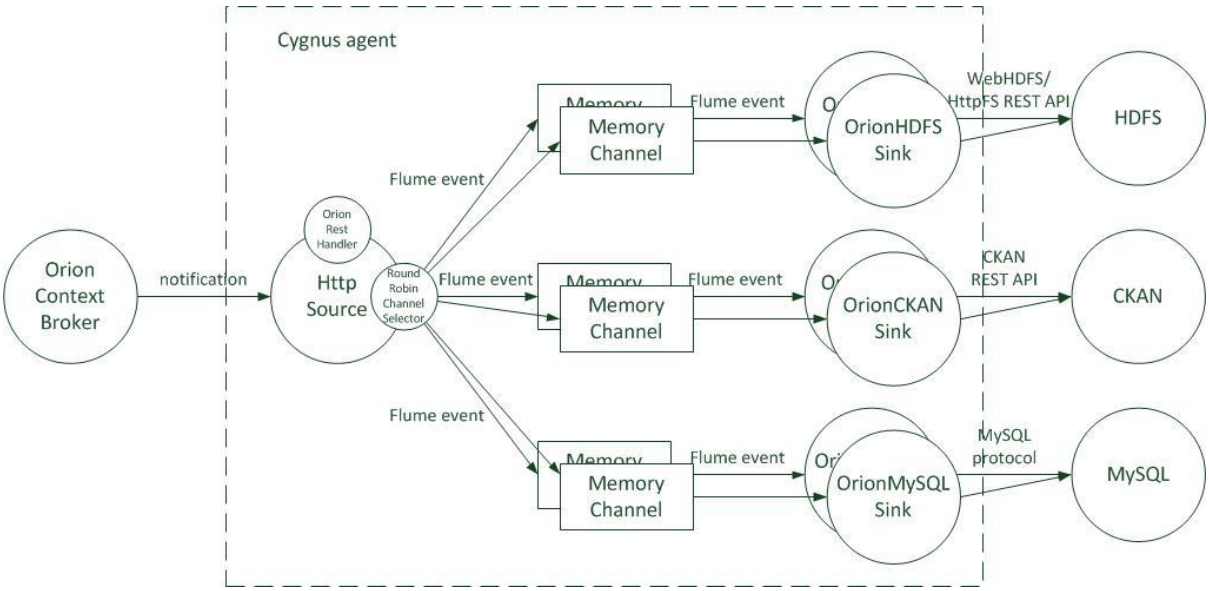


Figura 4.6. Arquitectura avanzada de Cygnus.

4.4. Escenario de caso de uso de IoT

El propósito del banco de pruebas es medir el rendimiento de la plataforma FIWARE. Por razones prácticas, ha sido desplegado en un entorno de laboratorio. Sin embargo, un escenario de caso de uso de IoT ha sido concebido y posteriormente analizado para identificar los requisitos de diseño del banco de pruebas perseguido. A continuación, procedemos a describir el escenario de caso de uso IoT ideado.

Se supone un escenario distribuido en el ámbito de la agricultura/viticultura de precisión (p.ej. cultivos en varias regiones o países). Hipotéticamente, se trata de un escenario IoT complejo ya que implica la gestión de un gran número de *recursos* asociados a sensores y actuadores. Concretamente, los sensores detectan datos relativos al entorno (p.ej. temperatura del aire, humedad relativa, radiación solar y precipitación) y al estado de los viñedos (p.ej. agua y nutrientes). Los actuadores son instalados en los sistemas de riego de los cultivos. El sistema tiene que ser escalable: si aumenta el número de sensores y actuadores, únicamente habría que incrementar los recursos computacionales, sin que esto implique mayor complejidad o un rediseño del sistema. Para abordar este desafío de escalabilidad, se planifica la implementación del *Farm Management System* (FMS) ilustrado en la Figura 4.7.

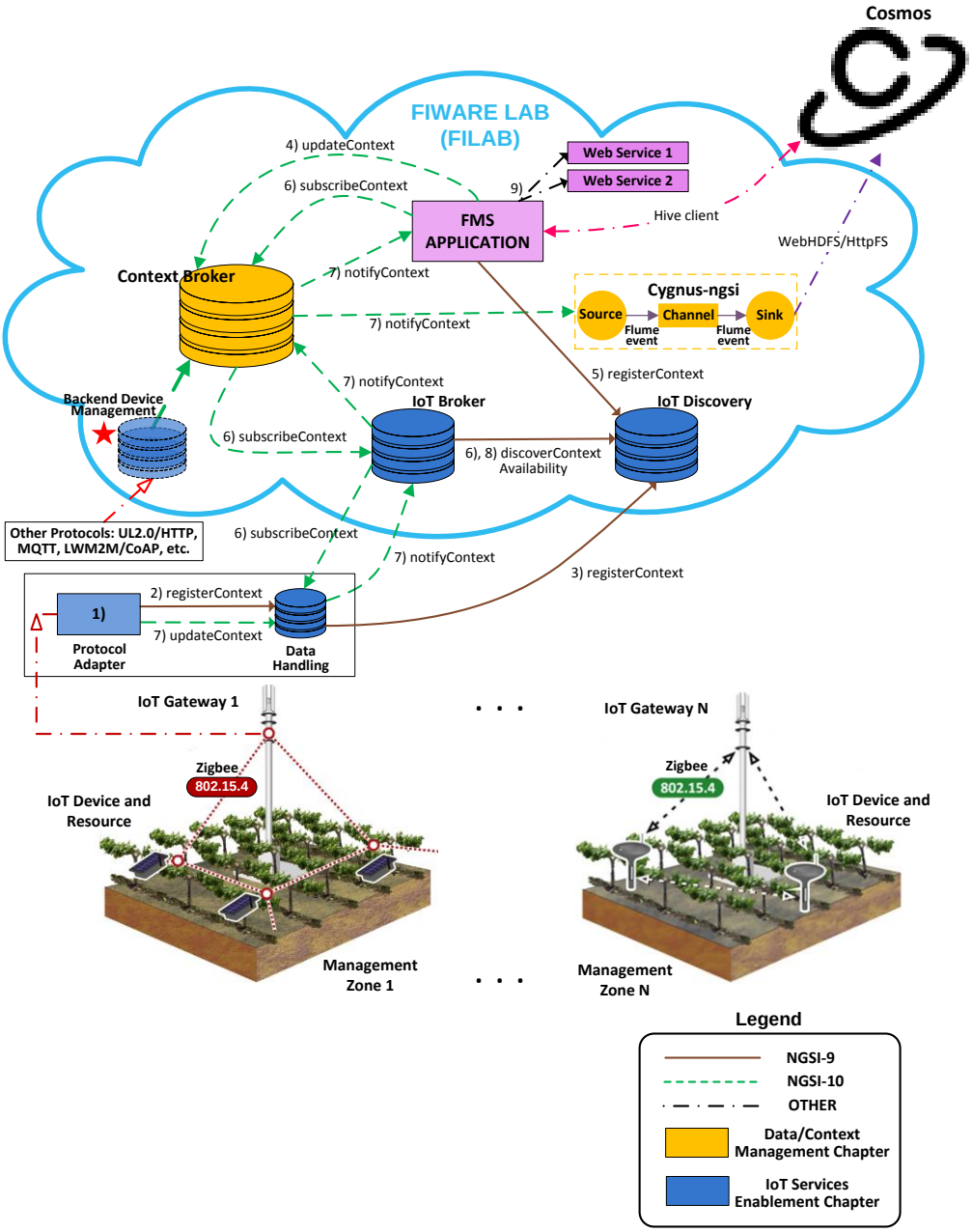


Figura 4.7. FMS basado en la arquitectura FIWARE (Figura inferior extraída de [123]).

El FMS diseñado toma ventaja de las nuevas características que ofrecidas por Internet del Futuro (*Future Internet*) al basarse en la arquitectura IoT FIWARE (véase sección 4.3). Los viñedos se dividen en zonas de gestión, cada una de ellas con una red de sensores inalámbrica (WSN) Zigbee. Los IoT *Gateways* de estas redes hospedan características de los GEs *Protocol Adapter* y *Data Handling*. Estos *Gateways* están ubicados en la proximidad de los dispositivos. Los IoT *Gateways* se conectan con una cuenta específica de FILAB [8] (*testbed* de FIWARE), que hospeda instancias de los GEs IoT *Broker*, IoT *Discovery y Context Broker*. Nuestra aplicación, FMS, también está hospedada dentro de esta instancia. Las funciones que realizan estos componentes se explican a continuación.

Registro de los dispositivos y recursos IoT

1. Al arrancar el sistema, el *Protocol Adapter* detecta de forma automática los *recursos* IoT asociados a los dispositivos, sensores o actuadores, de su zona de gestión.
2. Por cada *recurso* detectado, el *Protocol Adapter* envía una solicitud *registerContext* al *Data Handling*.
3. Cuando el *Data Handling* recibe la solicitud *registerContext*, crea una *entidad* en su base de datos local que *modela al recurso* y envía una solicitud *registerContext* al IoT *Discovery*. El IoT *Discovery* agrupa información sobre *recursos* de diferentes zonas de gestión.

Modelado de alto nivel de las cosas.

4. Consideremos ahora, justo en el otro extremo de la Figura 4.7, la aplicación del FMS y el componente *Context Broker*. La aplicación FMS implementa un cliente RESTful HTTP que puede invocar servicios del *Context Broker*. El servicio *updateContext* del *Context Broker* se invoca tanto para crear nuevas *entidades* que modelan *cosas* como para actualizar el estado de las *entidades* ya existentes. Se crean así dos tipos de *entidades*: los *recursos* que se detectan de forma automática (pasos 1 y 2) y las *cosas*, que se definen de forma explícita desde la aplicación. *Recursos* y *cosas* responden en FIWARE a diferentes niveles de abstracción, como también se ha mostrado en la Figura 4.2. Las *cosas* suelen agrupar *recursos* atendiendo a su ubicación y al tipo de sus atributos. El *Context Broker* puede acceder a la información de los *recursos* a través de un componente todavía no presentado: el IoT *Broker*.
5. El mapeo entre *recursos* y *cosas* se realiza en el *IoT Discovery* que registra las *asociaciones* entre las *entidades* que *modelan* las *cosas* y aquellas que *modelan* los *recursos* del sistema.

Suscripción a la información generada por las entidades.

6. Una vez definidas las *entidades*, ya sean *recursos* o *cosas*, cualquier aplicación puede suscribirse a la información que dichas *entidades* generan, mediante la invocación de la operación *subscribeContext* del *Context Broker*. En nuestro caso, se han suscrito la aplicación FMS, que crea las *entidades* y se suscribe a ellas, y *Cosmos*, que es un sistema

[8] FIWARE LAB (FILAB): https://www.fiware.org/lab/

de almacenamiento para Big Data de FIWARE. La conexión de *Cosmos* no es directa, sino a través de otro componente de FIWARE, el inyector *Cygnus*. Obsérvese que hay una cadena de suscripciones y de *brokers* (*Data Handling*, IoT *Broker* y *Context Broker*). El más importante es el *Context Broker*, que hace accesibles las *cosas* y *recursos* a las aplicaciones y constituye la implementación de referencia para los otros *brokers* del sistema. El *Context Broker* se suscribe a la información de contexto del *IoT Broker*, que hace de intermediario entre el *Context Broker* y diversos *Data Handling* asociados a diferentes áreas o zonas de gestión del campo de cultivo. El *IoT Broker* hace uso del servicio *discoverContextAvailability* del *IoT Discovery* con el fin de descubrir los *recursos* asociados a las *cosas* con las que trabaja el *Context Broker*. Para nuestro caso de estudio, todas las suscripciones son *onChange*: solo se generan *eventos* y, por tanto, envíos de notificaciones a los suscriptores cuando se producen cambios en el valor de los atributos de las *entidades*.

Notificación de la información.

7. Los datos de los cultivos son adquiridos por los *dispositivos y recursos* IoT, que los trasmiten al IoT *Gateway* donde son recibidos por el *Protocol Adapter*. En la Figura 4.7, los *dispositivos* utilizan el protocolo IEEE 802.15.4/ZigBee, sin embargo podrían utilizar otros (CoAP o RFID). De hecho, los *Protocol Adapter* ocultan al resto de componentes los protocolos de comunicación con los dispositivos físicos. El *Protocol Adapter* pasa la información recibida al *Data Handling* mediante un *updateContext*, el *Data Hanling* la notifica (*notifyContext*) al IoT *Broker*, que a su vez se la notifica al *Context Broker*, que a su vez se la pasa a sus suscriptores, en nuestro caso la aplicación FMS y *Cygnus*.
8. El caso del IoT *Broker* es especial, ya que debe interaccionar con el componente IoT *Discovery* para traducir las notificaciones a nivel de *recurso* a las correspondientes notificaciones a nivel de *cosa* enviadas hacia el *Context Broker*. Otro caso particular, es el inyector *Cygnus*, que se explica más adelante.
9. Finalmente, la aplicación FMS implementa dos servicios Web a partir de la información que le suministran el *Context Broker* y *Cosmos*. En el primero se publica información en tiempo real e informes especializados acerca del entorno y del estado de sus viñedos (p.e. predicción de plagas) que facilitan la toma de decisiones relativa a las tareas de riego, fertilización y control de plagas. El segundo servicio Web consiste en una interfaz hombre máquina (HMI) que permite la activación remota de los elementos de control instalados en los sistemas de riego de sus cultivos. En este caso, la información viaja en sentido inverso al estudiado siendo el primer paso el envío de comandos de control al *Context Broker*.

Resumiendo, el FMS exhibe un comportamiento sensible al contexto: la información procedente de los sensores es adquirida, modelada y recopilada para su procesamiento, posterior análisis y extracción de *conocimiento*. Este *conocimiento* facilita la toma de decisiones y el consiguiente envío de comandos de activación a los actuadores. Dicho comportamiento es posible gracias a la orquestación de servicios IoT proporcionada por FIWARE permitiendo que los *recursos* asociados a los *dispositivos* se conviertan en

buscables, *accesibles* y *utilizables* maximizando su interacción con la aplicación del FMS. Así, en la arquitectura (Figura 4.7) se observa como el *Context Broker* interopera con el *IoT Broker* y los IoT *DataHandling* que implementan componentes *Publish/Subscribe* de características similares al primero. De esta forma, se crea una red de *Brokers* que permite la interacción desacoplada entre servicios productores y consumidores de información de contexto. Este hecho pone de manifiesto el concepto de federación adoptado por FIWARE de la arquitectura IoT-A (véase sección 2.5).

Este escenario podría ser más complejo si los dispositivos físicos utilizasen protocolos de comunicación que no fueran soportados por el *Protocol Adapter* (p.e. MQTT). En este caso sería necesario el despliegue de una instancia del *Backend Device Management* (GE etiquetado con una estrella en la Figura 4.7) para abordar este desafío de interoperabilidad. Así, se puede concluir que el *Context Broker* maneja toda la información de contexto del sistema, intermediando entre los elementos que producen dicha información (IoT *Broker*, *Backend Device Management* y aplicación del FMS) y los elementos que la consumen (IoT *Broker*, *Backend Device Management*, aplicación del FMS y *Cosmos* a través de *Cygnus*). De esta manera, el *Context Broker* es la espina dorsal del flujo de datos en el sistema. El *Context Broker* juega un papel determinante en cualquier sistema sensible al contexto basado en tecnologías FIWARE. Por esta razón, se ha diseñado un banco de pruebas con el fin de evaluar su rendimiento.

CAPÍTULO 5

Banco de pruebas

The only systems on earth that have ever scaled to the size and scope of the Internet things are natural systems.

Francis daCosta
MeshDynamics CTO

5.1. Introducción

En el capítulo 4 se ha descrito un escenario de caso de uso de IoT en el ámbito de la agricultura de precisión. Este escenario plantea desafíos de escalabilibilidad e interoperabilidad, entre otros. Para superarlos, se ha concebido un *Farm Management System* (FMS) basado en la arquitectura IoT de FIWARE. Tras el análisis del sistema propuesto, se procede a la identificación de los requisitos de diseño del banco de pruebas.

5.2. Diseño

En la fase de diseño del banco de pruebas se parte de los siguientes requisitos:

- Orientado inicialmente hacia FIWARE, pero adaptable para su uso con otras plataformas, lo que permitirá estudios comparativos.
- Orientado a la medida del rendimiento y de la capacidad de almacenamiento de datos, pero extensible para incorporar otras propiedades.
- Establecimiento de los parámetros de generación de datos: número de nodos IoT, frecuencia de generación de datos y *payload*.
- Establecimiento de los parámetros de las pruebas en FILAB: número y características de máquinas virtuales (VMs) desplegadas, número y características de *Generic Enablers* instanciados y parámetros del inyector Cygnus.
- Configuración y arranque automático de los tests contra instancias de FIWARE.

El banco de pruebas diseñado (Figura 5.1) tiene en cuenta los elementos más importantes del caso de estudio mostrado en la Figura 4.7, incluyendo (1) una aplicación generadora de contexto, (2) instancias de las implementaciones de referencia del *Context Broker* (*Orion*) y *Big Data Analysis* (*Cosmos*), (3) una aplicación cliente (Hive) que hace las veces de la aplicación FMS y (4) el inyector *Cygnus*.

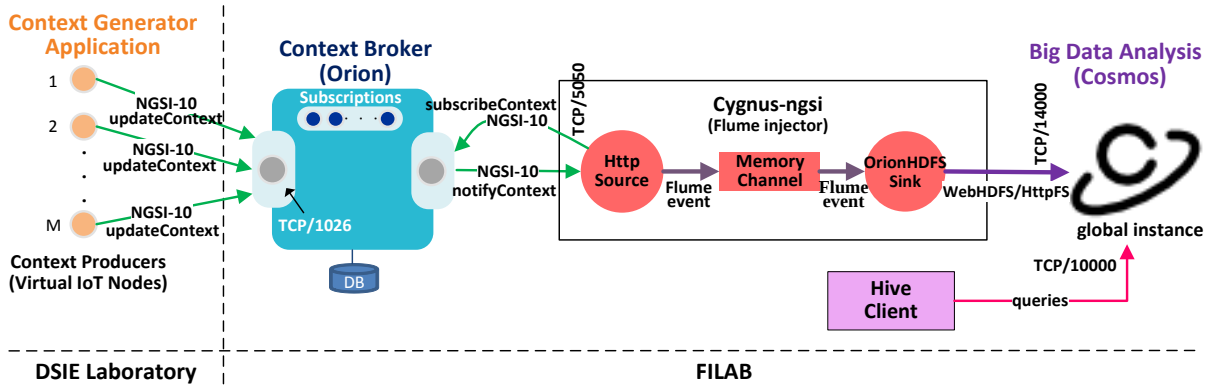


Figura 5.1. Un banco de pruebas para evaluar el rendimiento de la plataforma FIWARE.

La aplicación generadora de contexto sigue la arquitectura cliente/servidor y se ejecuta en 13 máquinas ubicadas en uno de los laboratorios del grupo de investigación División de Sistemas e Ingeniería Electrónica (DSIE) de la Universidad Politécnica de Cartagena. Esta aplicación simula la generación de información de contexto de una red de nodos IoT gracias a la implementación de clientes RESTful (NGSI). Como criterio de diseño, cada nodo IoT simula un productor de contexto que puede hacer referencia a un nodo intermedio (p.e. IoT *Broker*) o a un nodo NGSI final (véase Figura 4.2) que puede integrar varios sensores. En los casos mencionados, se cumple la transmisión directa de solicitudes al *Context Broker* para la actualización del contexto correspondiente a las entidades que pueden ser modeladas en sistemas IoT del futuro que planteen el desafío de escalabilidad estudiado en el capítulo 4.

La fuente (*Http Source*) del inyector *Cygnus* se encarga de recibir los eventos procedentes del *Context Broker*, transformarlos en eventos Flume [121] e inyectarlos en el canal. El sumidero de *Cygnus* (*OrionHDFSSink*) extrae los eventos Flume del canal y utiliza las APIs WebHDFS/HttpFS [119] para persistir la información de contexto contenida en el cuerpo de los eventos Flume en el *Hadoop Distributed File System* [119] configurado en *Cosmos*, creando una vista histórica de tales datos. Por otro lado, la aplicación del FMS implementa un cliente Hive [119] para la ejecución de tareas *MapReduce* asociadas a técnicas de nueva generación de minería y análisis (modelos de razonamiento) que procesan la información de contexto histórica almacenada en *Cosmos*. Las percepciones resultantes son extraídas a través del lenguaje *HiveQL*.

5.3. Entorno de Experimentación

Por un lado, se dispone de una cuenta *community* en la infraestructura *cloud* Spain2 de FILAB que ha permitido desplegar una máquina virtual (VM) que hospeda la versión 1.2.1 de una instancia del *Context Broker*, la versión 3.0.12 de una base de datos MongoDB y la versión 0.13.0 del inyector *Cygnus*. La VM tiene las siguientes características: 4 VCPU, 8 GB de RAM y 80 GB de disco. Se ha optado por la elección de estas cuotas para superar los requerimientos *hardware* de *Orion* y *Cygnus*. Además, se ha creado una cuenta en la instancia global de *Cosmos* cuya cuota inicial de 5 GB de almacenamiento es ampliable.

Por otro lado, las máquinas del laboratorio DSIE tienen las mismas características:

- *Hardware*: HP Compaq Business Desktop dc7700p (procesador Intel Core 2 Duo E6600/2.4 GHz Dual-Core, memoria caché de 4 MB por núcleo, memoria caché L2 de 4 MB y memoria RAM de 2 GB).
- *Software*: sistema operativo Windows 7 Home Premium, entorno de desarrollo Eclipse Luna SR2 (4.4.2) y aplicación generadora de contexto.

5.4. Aspectos de instalación y configuración

Desde un punto de vista relativo al rendimiento se han tenido en cuenta las recomendaciones FIWARE para ajustar los siguientes componentes del banco de pruebas: (1) MongoDB, (2) *Orion* y (3) *Cygnus*. A continuación se indican los aspectos de configuración más importantes:

En primer lugar, se ha utilizado la versión 3.0.12 de MongoDB cuyo uso es recomendable en escenarios de actualización intensiva. Además, se han comprobado los ajustes que limitan el uso a los recursos de la VM desplegada y se han deshabilitado las *Transparent Huge Pages* [124].

En segundo lugar, se han ajustado los parámetros de *Orion* para escenarios de altas tasas de actualización/envío de notificaciones (*–reqPoolSize*, *–dbPoolSize* y *–notificationModethread Pool:q:n*) [124]. Estos parámetros han sido ajustados en función de cada test.

En tercer lugar, los componentes de *Cygnus* han sido ajustados para aumentar su rendimiento, evitando posibles cuellos de botella para proceder a una correcta evaluación de Orion. Para ello, se ha optado por una fuente HTTP donde se ha configurado el manejador *OrionRestHandler* para convertir los eventos NGSI procedentes de *Orion* en eventos Flume. Respecto al diseño del canal, se apuesta por un *Memory Channel* debido a su rendimiento al ser implementado directamente en memoria. Este tipo de canal es el ideal para los flujos que necesitan un mayor rendimiento y que estén dispuestos a perder los datos en el caso de errores del agente Flume. Finalmente, se ha elegido un tipo de sumidero *OrionHDFSSink* siendo posible incrementar su rendimiento al utilizar el mecanismo de *batching* para disminuir el número de escrituras en el almacenamiento HDFS configurado en Cosmos. Este mecanismo es configurado en cada simulación a través de los parámetros *batch size* y *batch_timeout* [125]. A continuación, se profundiza en los aspectos de instalación y configuración de cada *Generic Enabler* del banco de pruebas.

5.4.1. ***Context Broker.***

Tanto para desplegar una instancia del *Context Broker* como para utilizar cualquier servicio de la plataforma FIWARE, en primer lugar es necesario el registro en FILAB a través del siguiente enlace[9]. A continuación, se ingresa en la cuenta FILAB creada al introducir las credenciales configuradas en el proceso de registro.

Al hacer clic en la pestaña *Cloud*, aparece la ventana de la Figura 5.2 en la que se puede lanzar, modificar y eliminar las diferentes instancias desplegadas en la plataforma.

[9] FILAB: https://account.lab.fiware.org/

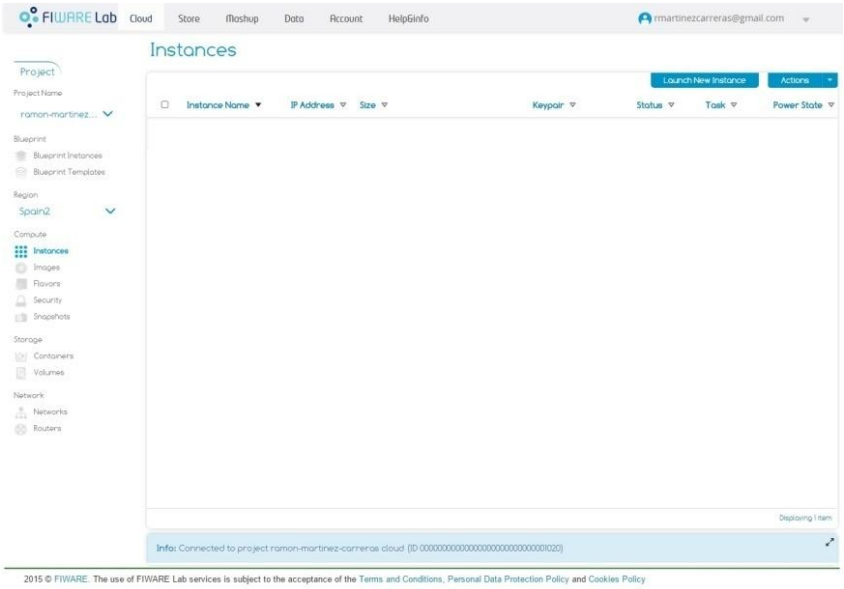


Figura 5.2. Ventana *Cloud* en FILAB.

Antes de lanzar la instancia del *Context Broker*, es necesario realizar las siguientes acciones: creación de una clave en formato .pem, creación de un grupo de seguridad que incorpore los puertos que utilizará la instancia a desplegar y creación de una IP pública.

Para ello, en la ventana *Cloud*, hacer click sobre *Security* → *Compute* → *Keypairs*. A continuación, hacer clic sobre *Create keypair* y añadir un nombre cualquiera con el fin de identificar la futura instancia asociada a dicha clave.

Tras ello, clicar en *Create Keypair* (Figura 5.3). Como resultado de esta acción aparece un cuadro de diálogo que invita a proceder a la descarga de la nueva clave generada en formato PEM. Esta descarga debe efectuarse para posteriormente, poder establecer comunicación con la instancia a través del *software* PuTTY.

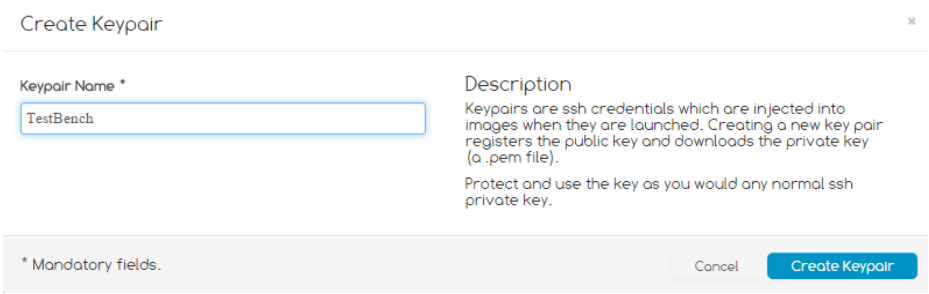


Figura 5.3. Creación de *Keypair* en FILAB.

Posteriormente, en la misma sección (*Security*) hacer clic sobre *Security Groups* → *Create Security Group*. En la ventana emergente, especificar un nombre y una descripción cualquiera que identifique la instancia a crear. Seguidamente, pulsar sobre *Create Security Group*.

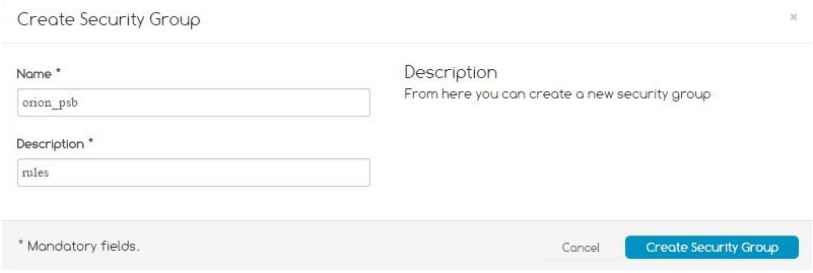


Figura 5.4. Creación de *Security Group* en FILAB.

Tras unos instantes, aparece el nuevo grupo de seguridad creado que permite especificar y abrir los puertos de la instancia a desplegar. Para realizar estas acciones, hay que seleccionar el nuevo grupo de seguridad y, posteriormente hacer clic sobre *Actions* → *Edit Rules*. Las reglas que deben ser añadidas son mostradas en la Figura 5.5.

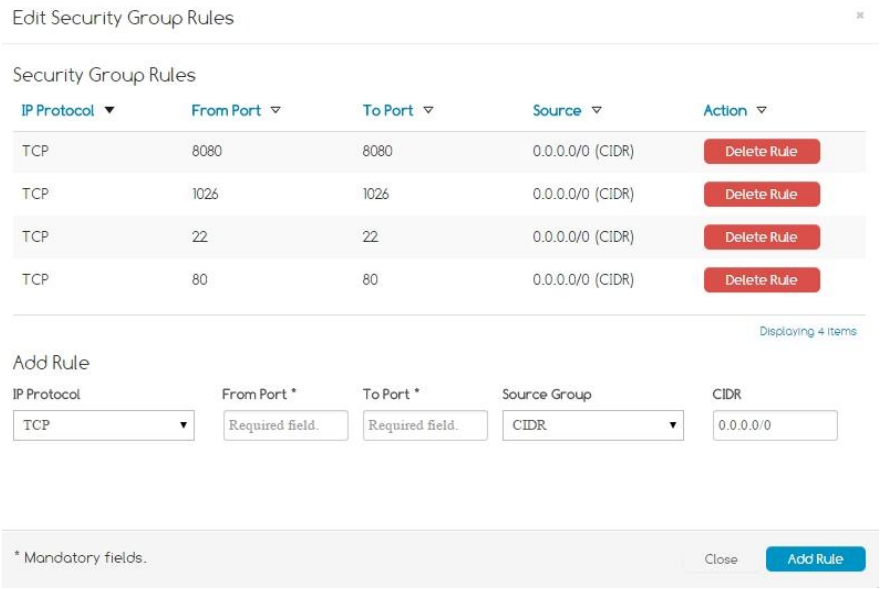


Figura 5.5. Configuración de *Security Group*.

Para finalizar en la sección *Security*, hacer clic sobre la pestaña *Floating IPs* para crear una IP pública disponible que, posteriormente será asociada a la nueva instancia. A continuación, click sobre *Allocate IP to Project*. Finalmente, elegir una de las opciones contempladas en la pestaña *Pool* y clic sobre *Allocate IP*.

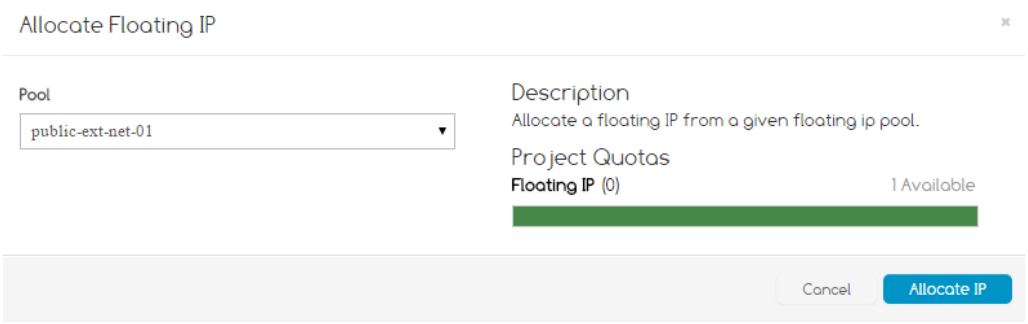


Figura 5.6. Creación de IP pública en FILAB.

La Figura 5.7 ilustra las configuraciones realizadas anteriormente en la sección *Security*.

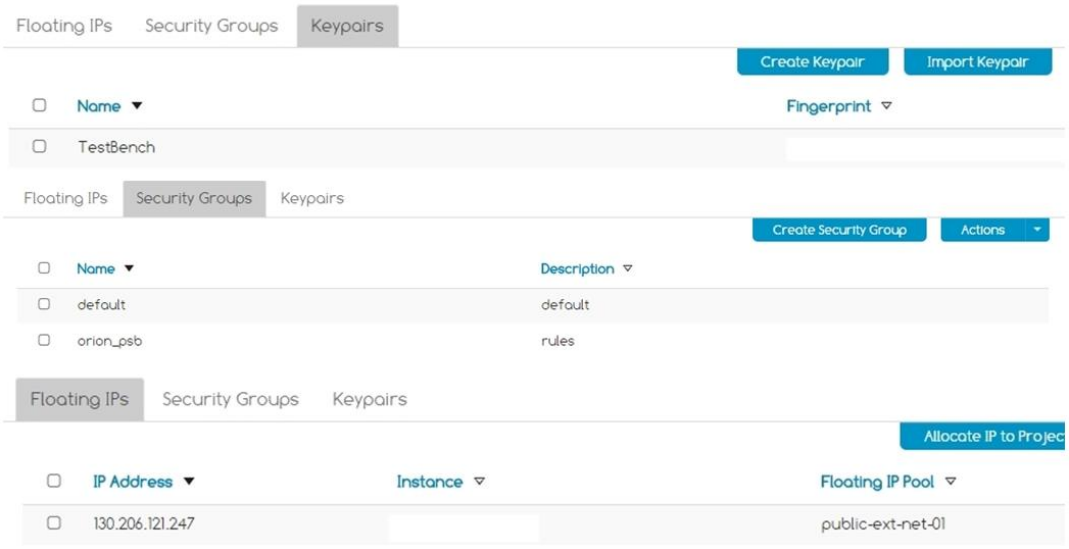


Figura 5.7. Configuración final de la sección *Security* en FILAB.

Una vez realizada dicha configuración, se procede a configurar y lanzar la instancia del *Context Broker*. Para ello, hacer clic sobre la pestaña *Images* del apartado *Compute*, buscar la imagen denominada orion-psb-image-R4.2 y hacer clic sobre *Launch*.

Aunque se trata de un proceso que se puede realizar de manera intuitiva, a continuación se describe la configuración de la instancia a desplegar. En el primer paso de este proceso, se especifica el nombre de la instancia así como su tamaño.

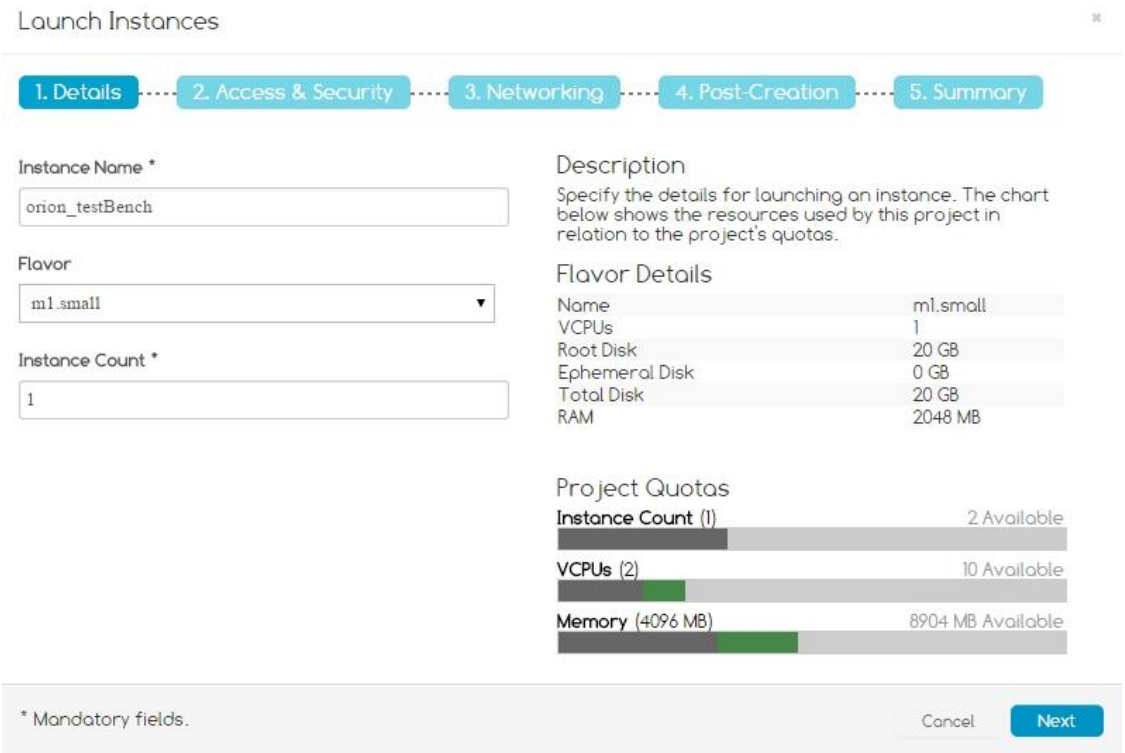


Figura 5.8. Configuración de la instancia (1).

En el segundo paso, se hace referencia al *Keypair* y al *Security Group* creados anteriormente.

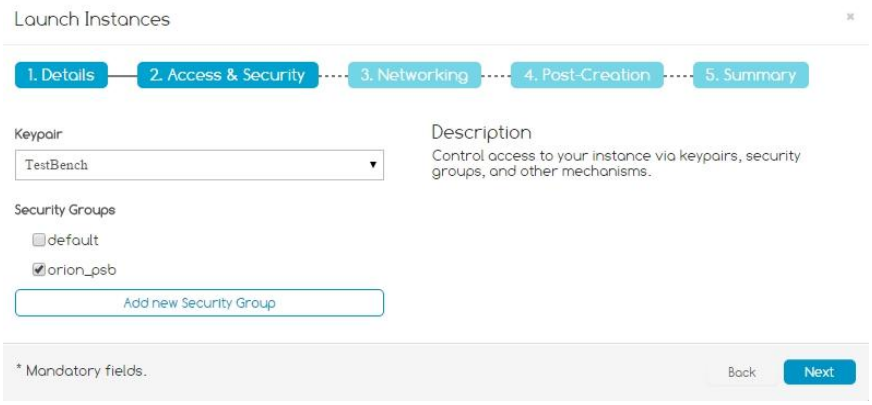


Figura 5.9. Configuración de la instancia (2).

En el tercer paso, hay que seleccionar la red de la instancia a desplegar. Para ello, arrastrar la opción *shared-net* desde la zona *Available Networks* hasta *Selected Networks*.

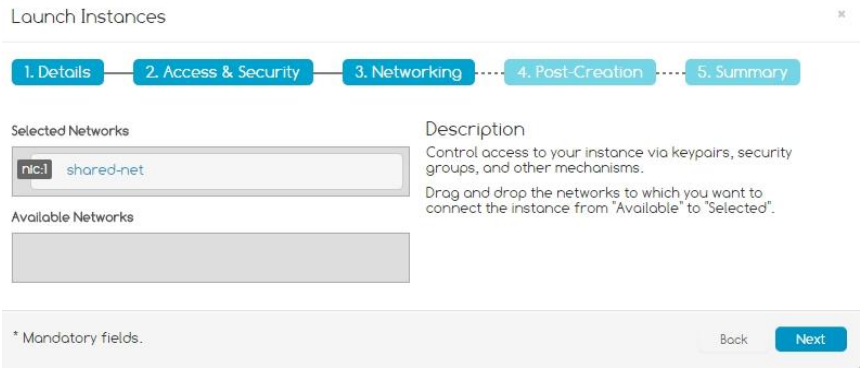


Figura 5.10. Configuración de la instancia (3).

Tras pulsar en Next, aparece un *script* con la configuración que ejecutará la máquina por defecto. Aunque se permite la modificación de dicha configuración, no se realiza ningún cambio en la misma.

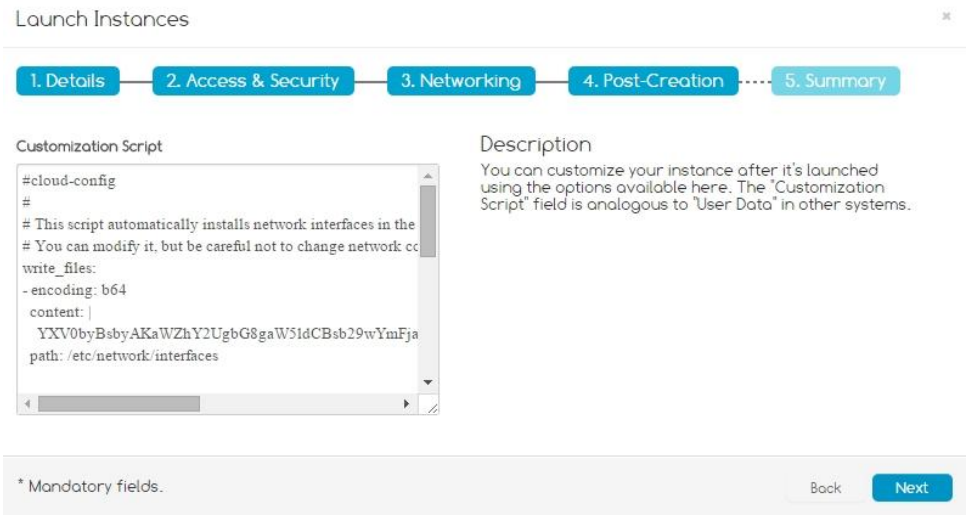


Figura 5.11. Configuración de la instancia (4).

Por último, aparece un resumen de los parámetros seleccionados en el proceso de configuración. Pulsar *Launch Instance* para terminar.

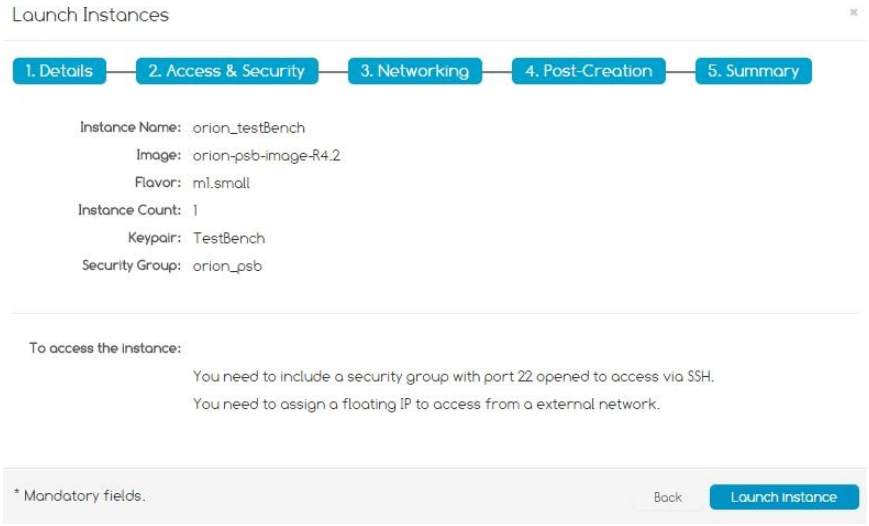


Figura 5.12. Configuración de la instancia (5).

Tras unos instantes, se inicia el proceso de creación de la instancia con los parámetros de configuración establecidos. Cuando este ha finalizado, se realiza el despliegue de la instancia, apareciendo en el apartado *Instances*, de manera similar a la mostrada por la Figura 5.13.

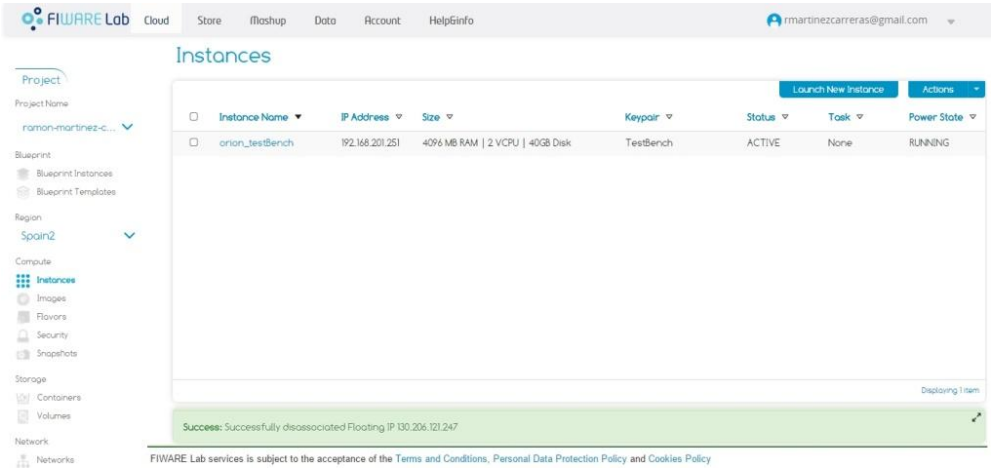


Figura 5.13. Instancia desplegada en FILAB.

Llegados a este punto solo resta asociar una IP pública a la instancia desplegada para poder establecer comunicación con la misma, ya que en estos momentos tiene asignada una IP privada, por lo que sería imposible su conexión remota. Para ello, pulsar nuevamente sobre el apartado *Security* y en la pestaña *Floating IPs* seleccionar la IP pública creada previamente. Tras esto, pulsar sobre *Actions* → *Associate IP*. En la ventana emergente, seleccionar en el desplegable la instancia recientemente creada y su IP privada. Finalmente, pulsar sobre *Associate IP*.

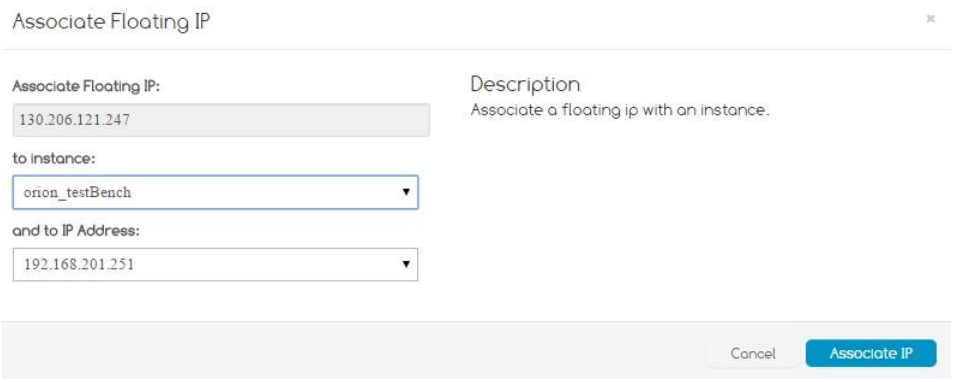


Figura 5.14. Asociación de IP pública.

5.4.2. ***Configuración de Herramienta PuTTY***

PuTTY solo acepta como medio de seguridad para la conexión con máquinas remotas las claves generadas bajo el formato .ppk (*PuTTY Private Key*). La clave que se obtiene de la creación de la instancia en FILAB es de formato .pem. Por este motivo, es necesaria la utilización del *software* PuTTYgen para realizar la conversión entre formatos.

Al ejecutar PuTTYgen, aparece una ventana que permite la creación de claves desde cero o bien, la carga de claves desde un fichero. Así, pulsar sobre *Load* y seleccionar la clave descargada en el proceso de generación de *Keypair* explicado anteriormente.

Posteriormente, hay que guardar la clave en formato .ppk pulsando sobre *Save private key* y seleccionando una ruta. Esta clave es secreta, por lo que es conveniente almacenarla en un lugar seguro y no compartirla ni publicarla.

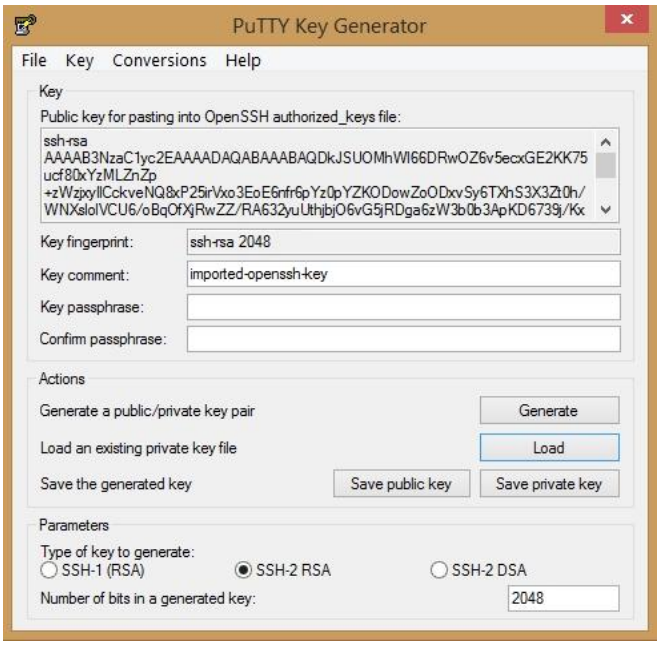


Figura 5.15. Conversión entre formatos con PuTTYgen.

Para realizar la conexión con la máquina virtual (Figura 5.16) de la instancia desplegada, hay que ejecutar PuTTY, especificando su IP pública en el apartado *Host Name* y su puerto (22).

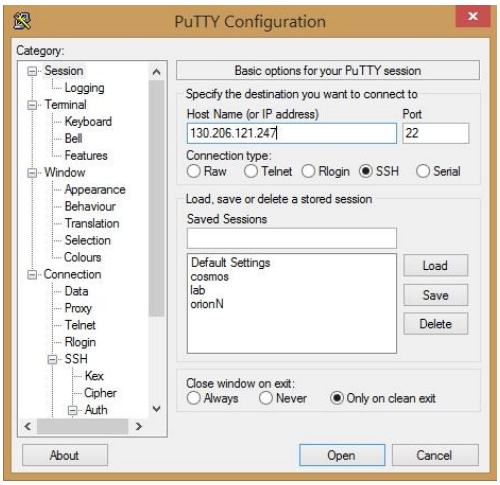


Figura 5.16. Conexión con PuTTY (1).

Finalmente, desplegar el menú *Connection* del panel *Category* y seleccionar el submenú SSH → Auth. En este submenú, proceder a la carga de la clave privada con formato .ppk generada mediante PuTTYgen y pulsar sobre *Open*.

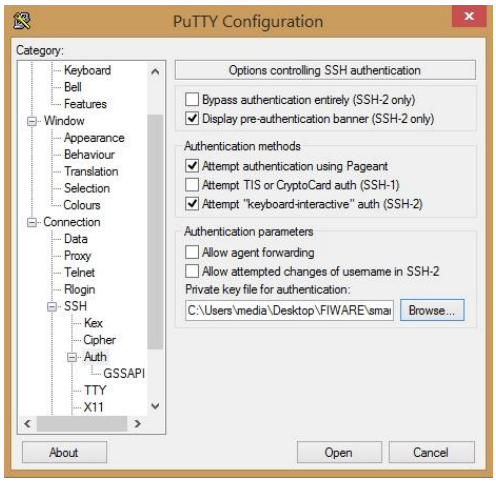


Figura 5.17. Conexión con PuTTY (2).

Una vez que se establece conexión, aparece un terminal en el que hay que identificarse como root. Posteriormente, aparece la línea de comandos del *broker*.

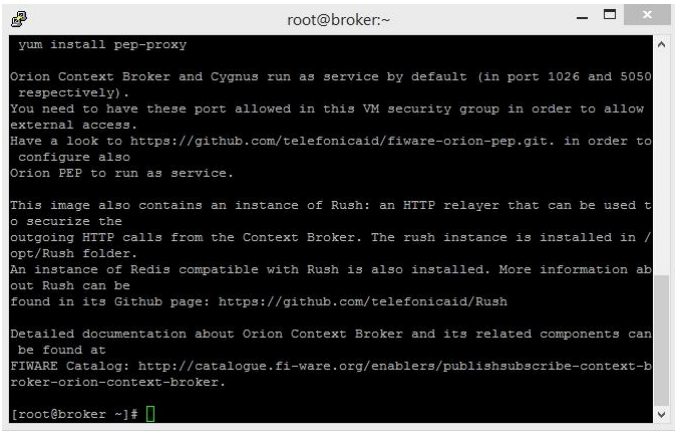


Figura 5.18. Terminal del *Context Broker*.

*5.4.3. **Cosmos***

Cuando se crea una cuenta en FILAB, automáticamente se crea una cuenta en *Cosmos*. Para verificar la existencia de esta cuenta, es necesario acceder a la GUI de Cosmos[10] e introducir las credenciales de la cuenta FILAB (Figura 5.19).

Cabe mencionar que existen métodos alternativos de instalación de este GE en máquinas locales que permiten una administración completa de *Cosmos*, sin embargo no han sido seguidos en el despliegue del banco de pruebas.

Figura 5.19. Login en Cosmos.

Tras el login, se notifica en pantalla acerca de la existencia del usuario en el sistema (Figura 5.20). Además, se muestran una serie de mecanismos de acceso a través de ssh que serán utilizados en la consola del *Context Broker* para acceder a *Cosmos*.

Figura 5.20. Parámetros de Cosmos

[10] http://cosmos.lab.fi-ware.org/cosmos-gui/

5.4.4. Instalación de Cygnus

Antes de proceder a la instalación de *Cygnus*, es necesario instalar Java SDK y Apache Maven. Para instalar Java SDK, se debe ejecutar el siguiente comando.

```
sudo yum install java-1.6.0-openjdk-devel
```

Una vez instalado el JDK, hay que exportar la variable de entorno JAVA_HOME. Para ello, se debe ejecutar el siguiente comando:

```
export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk.x86_64
```

Para que esta acción sea permanente, editar */root/.bash_profile*, incluyendo la línea de arriba.

Es necesario descargar Apache Maven e instalarlo en una carpeta elegida bajo criterio propio. En nuestro caso, el directorio elegido es */usr/local*

Los diferentes comandos a introducir en consola son:

```
wget http://www.eu.apache.org/dist/maven/maven-3/3.2.5/binaries/apache-maven-3.2.5-bin.tar.gz
tar xzvf apache-maven-3.2.5-bin.tar.gz
mv apache-maven-3.2.5 /usr/local
```

Llegados a este punto, se tienen las herramientas necesarias para la instalación de *Cygnus*. Si se quiere instalar *Cygnus* en una máquina CentOS proporcionada por FILAB, es necesario saber que existe una versión de *Cygnus* preinstalada. Por este motivo, en primer lugar hay que eliminar la versión existente e instalar una versión actual. En el caso de que se quiera instalar *Cygnus* en una máquina CentOS propia, este paso puede saltarse. Mencionada esta cuestión, para eliminar *Cygnus* se ejecuta:

```
sudo yum remove cygnus
```

Una vez eliminado *Cygnus*, se añade el repositorio FIWARE necesario:

```
sudo cat > /etc/yum.repos.d/fiware.repo <<EOL
[Fiware]
name=FIWARE repository
baseurl=http://repositories.testbed.fi-ware.eu/repo/rpm/x86_64/
gpgcheck=0
enabled=1
EOL
```

Finalmente, se procede a la nueva instalación mediante el comando:

```
sudo yum install cygnus
```

5.4.5. Configuración de Cygnus

La configuración de *Cygnus* se realiza a través del fichero **`agent_test.conf`** en la ruta **`/usr/cygnus/conf/`**. Este fichero es editado para establecer el número, tipo y disposición de los componentes de la arquitectura de este inyector (*sources*, *channels* y *sinks*). A continuación, se describe la configuración utilizada en el banco de pruebas.

- **Configuración de source/s (fuente/s de datos):** los parámetros de configuración más importantes son el nombre del canal asociado, el tipo de fuente (**`HTTPSource`**), el puerto (**`5050`**), el manejador de la información de contexto (**`OrionRestHandler`**) y el **`TimeToLive`**.

- **Configuración de channel/s (canale/s):** configuración de tipo (**`type`**), capacidad (**`capacity`**) y capacidad de transacción (**`transactionCapacity`**) del/los canales.
- **Configuración de sink/s (sumidero/s):** parámetros de configuración del consumidor de información de contexto. En el banco de pruebas desarrollado, la información puede ser consumida por *Cosmos* o almacenarse de manera local. La variable que determina el destino de la misma es **`attr_persistence`**. En el caso de que se desee almacenamiento en *Cosmos*, hay que especificar los parámetros de **`host`** y **`port`** y configurar la autenticación en Cosmos (**`username`** y **`password`**). Independientemente del destino de la información, hay que especificar el canal desde el que se consumen eventos (**`hdfs-channel`**) y el tipo de sumidero (**`OrionHDFSSink`**). Finalmente, es necesario especificar los parámetros **`batch_size`** y **`batch_timeout`** para configurar el número máximo de notificaciones por lote y el tiempo de espera para la transmisión de lote en el caso de no haber sido completado.

```
cygnusagent.sources = http-source
cygnusagent.sinks = hdfs-sink1
cygnusagent.channels = hdfs-channel1

cygnusagent.sources.http-source.type = org.apache.flume.source.http.HTTPSource
cygnusagent.sources.http-source.channels = hdfs-channel1
cygnusagent.sources.http-source.port = 5050
cygnusagent.sources.http-source.handler =
com.telefonica.iot.cygnus.handlers.OrionRestHandler
cygnusagent.sources.http-source.handler.notification_target = /notify
cygnusagent.sources.http-source.handler.default_service = def_serv
cygnusagent.sources.http-source.handler.default_service_path = def_serv_path
cygnusagent.sources.http-source.handler.events_ttl = 1
cygnusagent.sources.http-source.interceptors = ts gi
cygnusagent.sources.http-source.interceptors.ts.type = timestamp
cygnusagent.sources.http-source.interceptors.gi.type =
com.telefonica.iot.cygnus.interceptors.GroupingInterceptor$Builder
cygnusagent.sources.http-source.interceptors.gi.grouping_rules_conf_file =
/usr/cygnus/conf/grouping_rules.conf

cygnusagent.sinks.hdfs-sink1.type = com.telefonica.iot.cygnus.sinks.OrionHDFSSink
cygnusagent.sinks.hdfs-sink1.channel = hdfs-channel1
cygnusagent.sinks.hdfs-sink1.enable_grouping = false
cygnusagent.sinks.hdfs-sink1.hdfs_host = cosmos.lab.fiware.org
cygnusagent.sinks.hdfs-sink1.hdfs_port = 14000
cygnusagent.sinks.hdfs-sink1.hdfs_username = XXXXXXXX
cygnusagent.sinks.hdfs-sink1.hdfs_password = XXXXXXXX
cygnusagent.sinks.hdfs-sink1.oauth2_token = XXXXXXXXXXXXXXXX
cygnusagent.sinks.hdfs-sink1.file_format = json-column
cygnusagent.sinks.hdfs-sink1.hive.server_version = 2
cygnusagent.sinks.hdfs-sink1.hive.host = cosmos.lab.fiware.org
cygnusagent.sinks.hdfs-sink1.hive.port = 10000
cygnusagent.sinks.hdfs-sink1.krb5_auth = false
cygnusagent.sinks.hdfs-sink1.batch_size = 50
cygnusagent.sinks.hdfs-sink1.batch_timeout = 10

cygnusagent.channels.hdfs-channel1.type = memory
cygnusagent.channels.hdfs-channel1.capacity = 10000
cygnusagent.channels.hdfs-channel1.transactionCapacity = 200
```

Cuadro 5.1. Configuración de Cygnus utilizada en banco de pruebas.

El Cuadro 5.1 ilustra la configuración de Cygnus utilizada en el banco de pruebas (una fuente, un canal y un sumidero). En el apartado 4.3.5.5, se ha estudiado la posibilidad de aumentar el número de canales y sumideros con el fin de mejorar la eficiencia de *Cygnus*. Aunque estas configuraciones no han sido utilizadas en las simulaciones del banco de puebas desarrollado, pueden ser útiles en otros escenarios. El Cuadro 5.2 presenta la configuración de la arquitectura avanzada de múltiples canales y sumideros con selector de canal *Round Robin*. Concretamente, se modelan dos canales y sumideros, además del selector *Round Robin*. Esta configuración es una muestra de la escalabilidad del sistema pudiéndose aumentar el número de canales y sumideros para mejorar el rendimiento.

```
cygnusagent.sources = http-source
cygnusagent.sinks = hdfs-sink1 hdfs-sink2
cygnusagent.channels = notifications1 notifications2

cygnusagent.sources.http-source.type = org.apache.flume.source.http.HTTPSource
cygnusagent.sources.http-source.channels = notifications1 notifications2
cygnusagent.sources.http-source.selector.type =
es.tid.fiware.fiwareconnectors.cygnus.channelselectors.RoundRobinChannelSelector
cygnusagent.sources.http-source.selector.storages = 1
cygnusagent.sources.http-source.selector.storages.storage1=notifications1,notifications2
cygnusagent.sources.http-source.port = 5050
cygnusagent.sources.http-source.handler =
es.tid.fiware.fiwareconnectors.cygnus.handlers.OrionRestHandler
cygnusagent.sources.http-source.handler.orion_version = 1\.2\.1
cygnusagent.sources.http-source.handler.notification_target = /notify
cygnusagent.sources.http-source.handler.events_ttl = 1
cygnusagent.sources.http-source.interceptors = ts de
cygnusagent.sources.http-source.interceptors.ts.type = timestamp
cygnusagent.sources.http-source.interceptors.de.type =

cygnusagent.sinks.hdfs-sink1.cosmos_host = 130.206.80.46
cygnusagent.sinks.hdfs-sink1.cosmos_port = 14000
;cygnusagent.sinks.hdfs-sink1.attr_persistence = local
cygnusagent.sinks.hdfs-sink1.cosmos_default_username = XXXXXXXX
cygnusagent.sinks.hdfs-sink1.cosmos_default_password = XXXXXXXXX
cygnusagent.sinks.hdfs-sink1.cosmos_dataset = /user/rmartinezcarreras/sensores
cygnusagent.sinks.hdfs-sink1.hdfs_api = httpfs
cygnusagent.sinks.hdfs-sink1.channel = notifications1
cygnusagent.sinks.hdfs-
sink1.type=es.tid.fiware.fiwareconnectors.cygnus.sinks.OrionHDFSSink
cygnusagent.sinks.hdfs-sink2.cosmos_host = 130.206.80.46
cygnusagent.sinks.hdfs-sink2.cosmos_port = 14000
;cygnusagent.sinks.hdfs-sink2.attr_persistence = local
cygnusagent.sinks.hdfs-sink2.cosmos_default_username = XXXXXXXXXX
cygnusagent.sinks.hdfs-sink2.cosmos_default_password = XXXXXXXXXX
cygnusagent.sinks.hdfs-sink2.cosmos_dataset = /user/rmartinezcarreras/sensores
cygnusagent.sinks.hdfs-sink2.hdfs_api = httpfs
cygnusagent.sinks.hdfs-sink2.channel = notifications2
cygnusagent.sinks.hdfs-
sink2.type=es.tid.fiware.fiwareconnectors.cygnus.sinks.OrionHDFSSink

cygnusagent.channels.notifications1.type = memory
cygnusagent.channels.notifications1.capacity = 100
cygnusagent.channels.notifications1.transactionCapacity = 100
cygnusagent.channels.notifications2.type = memory
cygnusagent.channels.notifications2.capacity = 100
cygnusagent.channels.notifications2.transactionCapacity = 100
```

Cuadro 5.2. Configuración de *Cygnus* (múltiples canales y sumideros).

5.4.6. Compilación de Cygnus

Los pasos para compilar el código de *Cygnus* son los siguientes:

1. Copiar carpeta **`src`** a **`/usr/appRest/fiware-cygnus/flume`**.
2. Eliminar carpeta **`target`** en **`/usr/appRest/fiware-cygnus/flume/target`**.
3. **`cd /usr/appRest/fiware-cygnus/flume`**.
4. **`/usr/local/apache-maven-3.0.5/bin/mvn clean compile exec:exec assembly:single`**.
5. **`cp target/cygnus<>.jar /usr/cygnus/plugins.d/cygnus/lib`**.

5.4.7. Instalación y configuración VPN

5.4.7.1. Habilitar repositorio EPEL

En los laboratorios del grupo DSIE se ha creado una red privada virtual (VPN). Para ello, se ha utilizado el *software* OpenVPN. En primer lugar, hay que preparar el repositorio EPEL para descargar dicha herramienta *software* mediante la ejecución de la siguiente relación de comandos:

```
wget https://fedoraproject.org/static/0608B895.txt
mv 0608B895.txt /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
```

Tras esta acción, verificar que el repositorio se ha instalado correctamente:

```
rpm - qa gpg*
gpg-pubkey-0608b895-4bd22942
```

Finalmente habilitar EPEL:

```
rpm -ivh epel-release-6-5.noarch.rpm
```

5.4.7.2. Generación de claves

El siguiente paso es la descarga de los componentes necesarios para la generación de claves, es decir, la descarga del *software* OpenVPN. La instalación genera una serie de ficheros en el directorio **`/usr/share/easy-rsa/2.0`**, ficheros que deberán ser copiados a una carpeta a elegir cuyo rol será el de directorio de trabajo. Una vez en el mismo, se crea un directorio denominado **`keys`**, donde se alojarán las claves creadas en el proceso.

```
yum install openvpn easy-rsa
cp -ai /usr/share/easy-rsa/2.0 ~/easy-rsa
cd ~/easy-rsa
mkdir keys
```

Tras esta acción, han de generarse la clave y el certificado para la Autoridad Certificadora (CA). Para ello, editar el archivo **`vars`** mediante el siguiente comando:

```
gedit vars
```

y añadir la siguiente información:

```
export KEY_SIZE=2048
export KEY_DIR=”/root/easy-rsa/keys”
export KEY_COUNTRY=”ES”
export KEY_PROVINCE=”MU”
export KEY_CITY=”Cartagena”
export KEY_ORG=”UPCT”
export KEY_EMAIL=”XXXXXXXXXXXXXXXXXX”
```

La variable **KEY_DIR** especifica la ruta donde están alojadas las claves que se van a generar. El tamaño de la clave RSA es controlado por la variable **KEY_SIZE** en el archivo **vars**. Por defecto, su valor es de 1024, aunque se especificará un tamaño de 2048 para obtener una mayor seguridad. Después de realizar esta acción, teclear lo siguiente para limpiar todos los archivos de la carpeta **keys**.

```
source ./vars
./clean-all
```

Llegados a este punto, se obtiene la configuración necesaria para comenzar a generar claves y certificados de autenticación de usuarios. Lo primero es generar los parámetros **Diffie Hellman** para la constitución de claves. Para ello, como usuario **root** ejecutar:

```
./build-dh
```

Una vez finalizado el proceso, se crea el certificado para la CA, el certificado y la clave para el servidor y el certificado y la clave para el cliente. Estas tareas se realizan mediante la ejecución de los siguientes comandos:

```
./pkitool --initca
./pkitool --server servidor
./pkitool cliente1
```

Para la generación de más clientes, han de ejecutarse las siguientes dos líneas tantas veces como clientes quieran crearse, donde X es el número de cliente a crear.

```
source ./vars
./pkitool clienteX
```

Si se accede a la ruta **/root/easy-rsa/keys**, se pueden ver todas estas claves generadas. El método de autenticación de servidor y clientes es TLS, por lo que debe generarse de igual manera una clave TLS en base a los parámetros **Diffie Hellman** negociados. Para tal fin, se ejecutan los siguientes comandos:

```
cd /root/easy-rsa/keys
openvpn --genkey --secret ta.key
```

Por último, copiar las claves almacenadas en el directorio de trabajo y pegarlas en la ruta de instalación del programa OpenVPN

```
cp -ai /root/easy-rsa/keys /etc/openvpn
```

5.4.7.3. **Server.conf - Configuración del servidor VPN**

Una vez generadas las claves necesarias y copiadas a la ruta de instalación del programa, solo resta la configuración de cada uno de los clientes y servidores que conformarán la red. La configuración del servidor es sencilla ya que solo se debe obtener la plantilla de configuración y editarla con las claves y parámetros especificados en pasos anteriores. Para obtener la plantilla ejecutar:

```
cp -ai /usr/share/doc/openvpn*/sample/sample-config-files/roadwarrior-server.conf
/etc/openvpn/server.conf
```

Ir a la ruta **`/etc/openvpn`** y mediante el comando **`nano`** editar el archivo obtenido.

```
cd /etc/openvpn
nano server.conf
```

La edición del fichero se centrará en los siguientes puntos:

```
dev tap
proto udp
rutas ca.crt server.crt server.key (etc/openvpn/keys)
Establecer dh dh2048.pem
tls-auth /etc/openvpn/keys/ta.key 0
cipher AES-256-CBC
comp-lzo adaptive
mode server
server 10.8.0.0 255.255.255.0
ifconfig 10.8.0.1 10.8.0.2
push "route 10.8.0.1 255.255.255.0"
push "route 192.168.0.0 255.255.255.0"
route 10.8.0.0 255.255.255.0
persist-key
```

Para arrancar el servidor hay que situarse en la ruta del archivo **`server.conf`** y ejecutar

```
openvpn server.conf
```

5.4.7.4. **Client.conf - Configuración del cliente VPN**

Para configurar el cliente, copiar los archivos **`clienteX.key`**, **`clienteX.crt`** y **`ca.crt`** desde el servidor (**`/etc/openvpn`**) hasta **`/etc/openvpn/clienteX`**.

```
mkdir clienteX
cp keys/ca.crt etc/openvpn/clienteX
cp keys/clienteX.key etc/openvpn/clienteX
cp keys/clienteX.crt etc/openvpn/clienteX
```

Tras esto, hay que descargar y editar la plantilla de configuración:

```
cp -ai /usr/share/doc/openvpn*/sample/sample-config-files/client.conf
/etc/openvpn/cliente.conf
nano cliente.conf
```

La edición del fichero se centrará en los siguientes puntos:

```
dev tap
proto udp
remote 212.128.44.177 1194
persist-key
rutas ca.crt clientX.crt clientX.key (en directorio clienteX)
tls-auth /etc/easy-rsa/cliente1/ta.key 1
cipher AES-256-CBC
comp-lzo
```

Para ejecutar el cliente (en máquinas Linux) situarse en la ruta **`/etc/openvpn`** y ejecutar

```
openvpn cliente.conf
```

5.4.7.5. **Configuración de equipos**

Es necesario configurar cada cliente del banco de pruebas mediante la instalación de un cliente VPN en cada equipo de la red y la creación de un directorio de trabajo en Eclipse que contenga la aplicación generadora de contexto desarrollada. Tras la configuración, la interfaz de simulación se comunicará con el servidor VPN para que inunde la red VPN con los parámetros de simulación, entradas de cada instancia de la aplicación generadora de contexto que se ejecuta en cada cliente de la red.

Para realizar la instalación, se dispone de los siguientes elementos generados en la instalación del servidor VPN: (1) claves y certificados de los clientes, (2) archivos **`ca.crt`** y **`ta.key`** del propio servidor y (3) fichero plantilla de configuración del cliente VPN. Además, se necesita el proyecto en Java para crear el directorio de trabajo en Eclipse. La instalación de los clientes también debe realizarse con el *software* OPENVPN[11]. En nuestro caso, la versión del sistema operativo de los equipos es Windows XP 32 bits, por lo que se descargará el instalador asociado a dichas características. La instalación del *software* es sencilla e intuitiva pero hay que tener en cuenta realizar la ejecución con permisos de administrador.

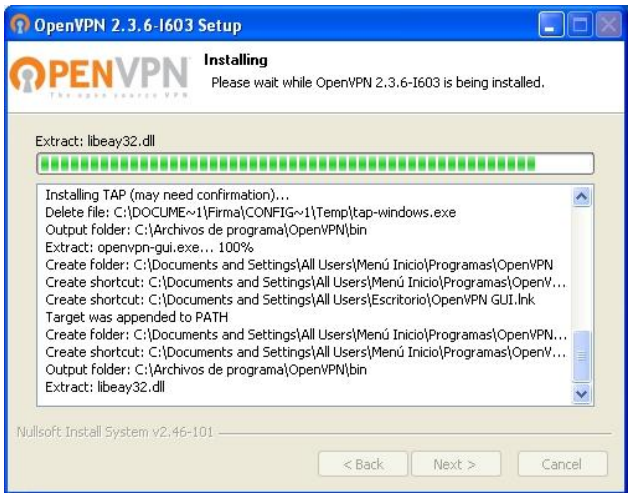


Figura 5.21. Instalación *software* OpenVPN.

[11] https://openvpn.net/index.php/open-source/downloads.html

Los 2 pasos siguientes son necesarios solamente si se utiliza Windows XP como Sistema Operativo, como es el caso que nos ocupa. En este caso, la instalación de este *software* se realiza de forma similar, sin embargo en la pestaña de elección de componentes hay que marcar todos ellos, es decir, **`TAP Virtual Ethernet Adapter`**, **`TAP Utilities`** y **`TAP SDK`**.

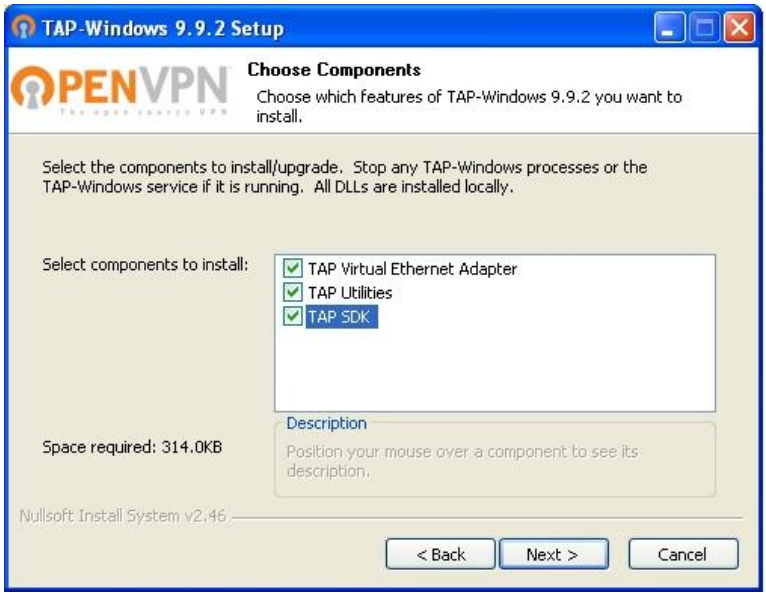


Figura 5.22. Instalación de TAP para Windows.

El siguiente punto importante para el correcto funcionamiento de la red VPN es la desactivación del *Firewall* de Windows. Para ello, realizar las siguientes acciones:

Inicio → Panel de Control → Centro de seguridad → *Firewall* de Windows → Desactivado y pulsar sobre el botón **Aceptar**.

Figura 5.23. Desactivación Firewall de Windows.

En este punto, resta incluir la configuración de la red VPN. Para ello, se debe acceder al directorio donde se ha instalado el *software* OpenVPN. En la carpeta **`config`** introducir los archivos **`cliente.ovpn`**, **`ca.crt`**, **`ta.key`**, **`clienteX.key`** y **`clienteX.crt`**, donde X denota el número de cliente que se instala, siendo diferente en cada instalación. Una vez que estos dos últimos ficheros han sido copiados en el directorio, se renombran a **`cliente.key`** y **`cliente.crt`** para que haya concordancia con el archivo **`cliente.ovpn`** y su configuración. Así, la carpeta config queda de la siguiente manera:

Figura 5.24. Carpeta config del cliente VPN.

La instalación y configuración del cliente OpenVPN ha finalizado. Se puede proceder a ejecutar el programa. Para ello, hacer clic en *Connect* y ver que existe conexión con el servidor VPN (IP 10.8.0.1) mediante el uso del comando **`ping`**.

Figura 5.25. Prueba de conexión con servidor VPN mediante comando ping.

A continuación, hay que configurar Eclipse con el proyecto que contiene la aplicación desarrollada en el banco de pruebas. Para importar el proyecto en Eclipse se selecciona un directorio de trabajo nuevo y se hace *click* en *File* → *Import*. Tras esta acción, aparece una ventana con diferentes pestañas. Desplegar la primera de ellas (General) y elegir la opción *Existing Projects into Workspace*. A continuación, hacer clic en *Next*. En la siguiente ventana, pulsar sobre *Browse* de la opción *Select root directory* y dirigirse al directorio donde se encuentra el proyecto a importar, en este caso **`TestBenchServer`**. Finalmente, pulsar en **Aceptar**. Mediante este procedimiento, el proyecto **`TestBenchServer`** es importado al nuevo directorio de trabajo. Se trata de la implementación de los servidores de la red del banco de pruebas. Un aspecto a tener en cuenta es que Eclipse puede lanzar errores en algunas clases del proyecto debido a que no encuentra la librería org.json.jar[12]. En este caso, hay que importar la librería, modificando el *Build Path* del proyecto (*Clic* derecho sobre el nombre del proyecto → *Build Path* → *Configure Build Path* → *Libraries* → *Add External JARs*).

[12] https://org-json-java.googlecode.com/files/org.json-20120521.jar

Por último, en el cliente de la red, se importa el proyecto **`TestBenchClient`**. A modo de resumen, el *software* instalado y configurado en los diferentes equipos del banco de pruebas es el siguiente: **`Servidor VPN`** + **`TestBenchClient`** (equipos de laboratorio DSIE) y **`Cliente VPN`** + **`TestBenchServer`** (equipos de laboratorio DSIE)

5.5. Métodos de Generación de Contexto

En cada prueba, la aplicación desarrollada ofrece al usuario la posibilidad de elegir entre dos métodos para la generación de tráfico NGSI. El primer método (método bloqueante) simula la generación de tráfico NGSI de nodos IoT intermedios. Su implementación es soportada por *java.net* [126] y su funcionalidad está basada en la clase *HTTPURLConnection*. Esta clase solo permite conexiones bloqueantes. Así, en cada simulación se abren tantas conexiones persistentes HTTP como nodos IoT simulados.

Respecto al segundo método (método no bloqueante), simula la generación de tráfico NGSI de nodos IoT finales. Su implementación se centra en mecanismos que permiten generar tráfico concurrente [127]. En este caso, se ha utilizado la clase *SocketChannel* de java.nio [128] para crear conexiones no bloqueantes. Por este motivo, en cada simulación se abren tantas conexiones como solicitudes NGSI generadas.

Otro detalle de implementación a tener en cuenta es que cada cliente NGSI (bloqueante o no) simula el comportamiento de un nodo IoT virtual en un hilo independiente.

5.6. Implementación

*5.6.1. **Estructura del Banco de pruebas***

A continuación, se muestra el diagrama de relaciones de uso de los diferentes paquetes que componen los proyectos desarrollados en Eclipse.

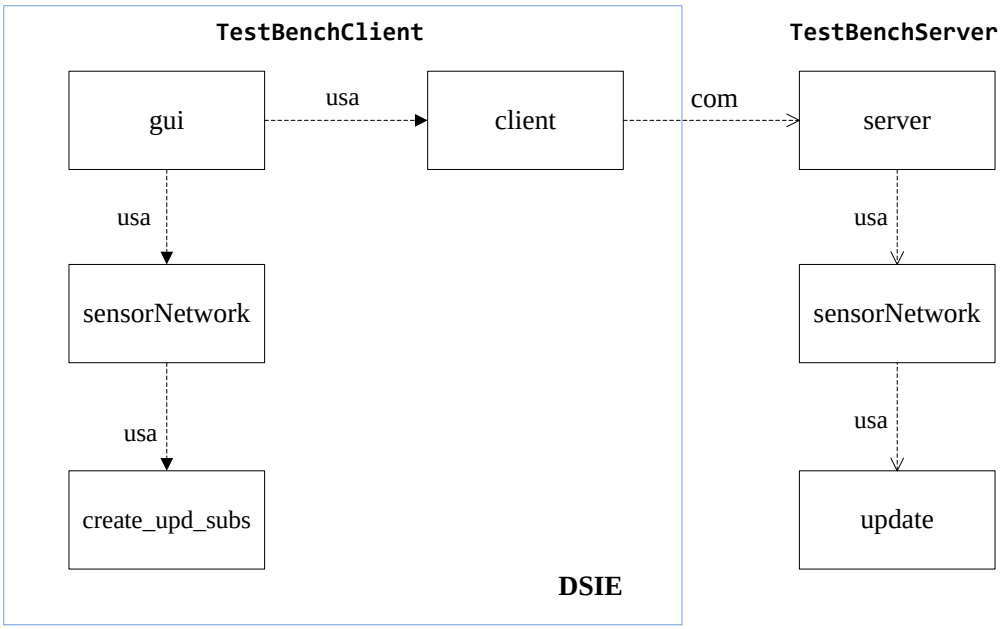


Figura 5.26. Diagrama de Relaciones de Uso.

Tal y como se observa en la Figura 5.26, se han desarrollado dos proyectos denominados **`TestBenchClient`** y **`TestBenchServer`**. En el nivel más bajo del proyecto **`TestBenchClient`**, se encuentra el paquete **`create_upd_subs`** que contiene las clases necesarias para construir operaciones de los tipos **`append`**, **`subscribe`** y **`update`** en formato **`Json`**. En el segundo nivel se halla el paquete **`sensorNetwork`** que contiene las clases **`SensorCB`**, **`SensorSB`** y **`SensorV`**, las cuales permiten enviar al *Context Broker* las operaciones de creación, suscripción y actualización respectivamente.

En el nivel más alto se encuentra el paquete **`gui`** que, tal y como su nombre indica, contiene la interfaz gráfica del simulador modelada por la clase **`Test`**. Esta interfaz permite realizar una simulación en modo local al utilizar las clases del paquete **`sensorNetwork`** o realizar una simulación en modo remoto al utilizar las clases del paquete **`client`**.

Por otro lado, el paquete **`server`** del proyecto **`TestBenchServer`** permite adquirir los parámetros de la simulación lanzada a a través de la interfaz gráfica del proyecto **`TestBenchClient`**. En el segundo nivel, el paquete **`sensorNetwork`** contiene la clase **`SensorV`** que implementa el envío al **`ContextBroker`** de las operaciones de actualización generadas por el paquete **`update`** que contiene las clases necesarias para construir las solicitudes OMA NGSI RESTful utilizadas por los componentes del capítulo *Data/Context Management* de la plataforma FIWARE.

*5.6.2. **Operaciones create, update y subscription***

Las operaciones asociadas a la creación, actualización y suscripción de/a entidades del *Context Broker* han sido implementadas por las clases del paquete **`create_upd_subs`** del proyecto **`TestBenchClient`** y del paquete **`update`** del proyecto **`TestBenchServer`**. Así, las operaciones de creación y actualización (**`append`**/**`update`**) siguen la siguiente estructura:

```
{"contextElements":
 [
   {
    "type":"Room",
    "isPattern":"false",
    "id":"Room1",
    "attributes":
      [
        {"name":"temperature",
         "type":"centigrade",
         "value":"0/1/0/2015-05-21X17:45:15.451"
        }
      ]
    }
 ],
 "updateAction":"APPEND"
```

Cuadro 5.3. Estructura de las operaciones append/update.

La operación anterior permite crear una entidad denominada Room1 en la base de datos del *Context Broker*. Además, esta entidad tiene un atributo de nombre temperatura, tipo centígrado y valor determinado por un *String*. Si se quisiera realizar la operación de actualización, solo sería necesario cambiar el valor de **`updateAction`** por **`UPDATE`**.

Por otro lado, la estructura de las operaciones de suscripción (**subscribe**) es la siguiente:

```
{"entities":
 [
   {
    "type":"Room",
    "isPattern":"false",
    "id":"Room1"
   }
 ],
 "attributes":
 [
   "temperature"
 ],
 "reference":"http://130.206.85.99:5050/notify",
 "duration":"P1M",
 "notifyConditions":
 [
  {
    "type":"ONCHANGE",
    "condValues":
      [
       "temperature"
      ]
  }
 ]
}
```

Cuadro 5.4. Estructura de la operación subscribe.

La suscripción anterior se realiza sobre el atributo de temperatura de la entidad Room1. Además, se trata de una suscripción tipo **ONCHANGE** sobre el valor de temperatura. La duración de la suscripción es de un mes (**P1M**). Así, cuando un evento cumpla estos requisitos, se debe enviar una notificación a la dirección **http://130.206.85.99:5050/notify**. La implementación de estas operaciones ha sido realizada mediante la librería org.json.jar.

*5.6.3. **Aspectos relativos a la conexión con el Context Broker***

El envío de las operaciones mencionadas anteriormente al *Context Broker* se realiza mediante la clase **HttpURLConnection** de Java como indica el siguiente fragmento de código.

```
String myURL="http://130.206.85.99:1026/ngsi10/updateContext";
try {
       url = new URL(myURL); //FILAB
       conn = (HttpURLConnection) url.openConnection();
       conn.setRequestMethod("POST");
       conn.setRequestProperty("Content-Type", "application/json");
       conn.setRequestProperty("Accept", "application/json");
       conn.setRequestProperty("Content-Length", "200");
       conn.setDoOutput(true);
       conn.setConnectTimeout(30);
       conn.setReadTimeout(30);
       conn.connect();
       connected = true;
       os = conn.getOutputStream();
}catch(Exception e){
        e.printStackTrace();
}
```

Cuadro 5.5. Conexión con el *Context Broker*.

Es importante mencionar que la clase **HttpURLConnection** realiza sólo un establecimiento de conexión (HTTP persistente). Además, la transmisión de datos al servidor Web evaluado es síncrona. Por estos motivos, para simular tráfico de redes de sensores que pueda exceder la capacidad del servidor evaluado se ha implementado un método no bloqueante.

5.6.4. Clase Server.java

La clase **Server.java** se encarga de permanecer a la espera de la recepción de parámetros para poder lanzar una nueva simulación. Para ello, utiliza un **DatagramSocket** que atiende peticiones del puerto 8080 y recibe un único *String* con los parámetros separados por comas. Los datos que se esperan recibir son, en orden:

- Número de sensores, representado por la variable **nSensores**.
- Periodo, representado por la variable **periodo**.
- Número de datos por sensor, representados por la variable **it**.
- Fecha de la máquina cliente (equipo del DSIE), representada por la variable **fecha**.
- Tamaño de *payload*, representado por la variable **payload**.

La variable **fecha** se utiliza para calcular el **offset** entre la máquina remota y la máquina local con el fin de sincronizar los relojes de ambas máquinas y obtener resultados coherentes. La separación de cada una de las variables dentro del String recibido se realiza por medio de la clase **StringTokenizer** que recibe como argumento el carácter de separación para poder iterar y obtener la totalidad de las variables.

Además, existe otra variable importante en esta clase denominada **idServer** que permite identificar al servidor que atiende y procesa la solicitud. Esta variable se utiliza para calcular los identificadores de los registros asociados a las entidades cuya información de contexto será actualizada por cada uno de los servidores de la red VPN. A continuación se muestra la relación de variables que utiliza esta clase:

```
Long offset = 0           //diferencia entre fechaDSIE y fecha local
final int idServer=0;     //identificador del equipo
String str;               //string de recepcion
int nSensores = 0;        //numero de sensores virtuales
int periodo = 0;          //periodo entre envio de datos
int it=0;                 //numero de datos a enviar
long fecha;               //fecha del cliente (DSIE)
int payload=0;            //payload
```

Cuadro 5.6. Variables de la clase Server.java.

Las siguientes líneas de código muestran la creación del **DatagramSocket** y el tratamiento de la información entrante.

```
DatagramSocket serverSocket = new DatagramSocket(8080);

/* WHILE TRUE */
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
serverSocket.receive(receivePacket);
str = new String( receivePacket.getData(), 0, receivePacket.getLength());
StringTokenizer st=new StringTokenizer(str,",");
//iteracion de los elementos recibidos
String [] aux=new String [5];
int j=0;
while (st.hasMoreElements()){
       aux[j]=(String) st.nextElement();
       j++;
}

//recuperar parametros de simulacion
nSensores=Integer.parseInt(aux[0]);
periodo=Integer.parseInt(aux[1]);
it=Integer.parseInt(aux[2]);
fecha=Long.parseLong(aux[3]);
payload=Integer.parseInt(aux[4]);

//calculo de offset
Date date = new Date();
long time = date.getTime();
offset = fecha-time;
```

Cuadro 5.7. Recuperación de los parámetros de simulación por clase Server.java.

cada servidor de la red. Esta tarea es controlada por dos variables. La variables **ini** y **fin** indican el primer y último registro del *Context Broker* a actualizar (el servidor actualizará hasta el registro **fin**-1). El algoritmo de asignación utilizado es el siguiente:

```
int ini=idServer*nSensores;
int fin=nSensores*(idServer+1);
```

Se supone que la variable **nSensores** es igual a 2, es decir, cada servidor tiene que actualizar dos registros del *Context Broker*, y además hay dos servidores escuchando la petición con identificadores **idServer**=0 e **idServer**=1 respectivamente. Los valores de las variables **ini** y **fin** del primer servidor (**idServer**=0) serán **ini**=0 y **fin**=2, mientras que los del segundo servidor (**idServer**=1) serán **ini**=2 y **fin**=4. En este caso, el primer servidor tendría asignados los registros 0 y 1 del *Context Broker* mientras que el segundo los registros 2 y 3. Así, gracias estos identificadores, cada servidor actualizará registros diferentes.

Como se puede observar en la creación del **DatagramSocket**, la recuperación de la información se realiza en un bucle **while(true)**. Esta implementación permite reutilizar la conexión sin necesidad de arrancar el servidor en cada simulación. En definitiva, una vez que se arranca la clase **Server.java**, esta está continuamente escuchando el puerto 8080 y preparada para atender solicitudes.

Por último, la clase **Server** lanza la simulación. A continuación, se muestra la inicialización de los sensores (número de hilos atendiendo a la variable **nSensores**) y la llamada al constructor de la clase **SensorV.java**:

```
SensorV sensors[] = new SensorV [nSensores];
for(int i = 0;ini<fin;ini++) {
    sensors[i] = new SensorV(periodo,ini,it,offset,payload);
    sensors[i].start();
    i++;
}
```

Cuadro 5.8. Inicilización de los sensores virtuales (entidades activas).

5.6.5. Clase Client.java

La clase **Client.java** se encarga de recibir los parámetros de la GUI de simulación y enviarlos a la red VPN mediante un **DatagramSocket**. Para ello, dispone de un constructor que es llamado en la clase **Sim.java** (interfaz gráfica del simulador) y que incluye todos los parámetros necesarios para arrancar la prueba. Una vez registrados estos parámetros en variables locales de la clase, se llama al método **start()** que crea un **DatagramSocket**, incluye la información recibida en un *String* y la envía a la dirección broadcast de la red VPN (10.8.0.255). Por último, se cierra la conexión del *socket*. Tal y como se ha mencionado anteriormente, el orden de las variables tiene que ser el mismo en el envío y recepción, además de ir separadas por comas. El Cuadro 5.9 ilustra el código del método **start()**.

```
public void start() throws IOException {
  DatagramSocket clientSocket = new DatagramSocket();
  InetAddress IPAddress = InetAddress.getByName("10.8.0.255");
  byte[] sendData = new byte[100];
  try {
      Date date = new Date();
      long fecha = date.getTime();
      String param = Integer.toString(nSensores)+ ","+Integer.toString(periodo)+
      ","+Integer.toString(it)+ ","+Long.toString(fecha)+","+Integer.toString(payload);
      sendData = param.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData,sendData.length,
      IPAddress, 8080);
      clientSocket.send(sendPacket);
    }
    finally {
      clientSocket.close();
    }
```

Cuadro 5.9. Implementación del método start().

*5.6.6. **Cosmos***

Cosmos es uno de los componentes del banco de pruebas. La API WebHDFS permite la comunicación con el *Context Broker*. A continuación, se describen los comandos Hadoop más utilizados durante las simulaciones.

Creación y eliminación de directorios	hadoop fs -mkdir <ruta> (Crea un directorio en la ruta que se le especifique)
	hadoop fs -rmr <ruta> (Elimina el directorio de la ruta que se le especifique)
Creación, copia y descarga de ficheros	hadoop fs -put <nombre fichero> <ruta destino HDFS> (Crea un fichero en la ruta HDFS especificada. Este comando es análogo a *touch* de Linux)
	hadoop fs -put <ruta local> <ruta destino HDFS> (Copia uno o múltiples ficheros desde un sistema local hasta un sistema Hadoop)
	hadoop fs -get <ruta HDFS> <ruta destino local> (Descargar uno o múltiples ficheros hacia un sistema local)
Información sobre directorios y ficheros	hadoop fs -ls <ruta> (Proporciona información de los directorios y ficheros alojados en esa ruta)
	hadoop fs -cat <ruta fichero> (Abrir, como si de un editor de texto se tratara, el fichero indicado)
	hadoop fs -du <ruta> (Muestra el tamaño de ficheros y directorios contenidos en la ruta especificada)

Tabla 5.1. Resumen de comandos Hadoop.

5.6.6.1. **Código Hive Basic Client**

La clase **`HiveBasicClient.java`** permite establecer en la máquina del Context *Broker* un cliente Hive, brindando la posibilidad de lanzar consultas de manera remota a los datos históricos almacenados en Cosmos. Se puede modificar su código para lanzar de manera automática, las consultas necesarias para el acceso de datos, así como su tratamiento y descarga en un fichero de texto.

Para este propósito, se utiliza la plantilla facilitada por el equipo de FIWARE en su repositorio de github. Las modificaciones más significativas que se han realizado residen en los métodos **`doQuery`** y **`main`**.

En primer lugar, cabe destacar las librerías importadas en la clase, pues difieren ligeramente de las que se declaran en la plantilla utilizada, concretamente son las siguientes:

```
import java.io.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Cuadro 5.10. Librerías importadas por el cliente Hive.

El método **doQuery** se encarga de enviar a *Cosmos* las instrucciones que se le pasan como argumento. Inicialmente, este método era de tipo void. Sin embargo, se ha modificado para que devuelva un *String*, que contendrá los datos devueltos tras realizar la consulta a Cosmos.

Además, se incorporan dos variables de tipo *String*, una es auxiliar y recoge los datos devueltos en cada iteración de la consulta y otra del mismo tipo que es el resultado de las concatenaciones de la primera variable.

```
private static String doQuery(String query) {
//final String - final String returned
String fS = "";
//String - used for each query result
String s = "";
        try {
            // from here on, everything is SQL!
            Statement stmt = con.createStatement();
            ResultSet res = stmt.executeQuery(query);
            //iterate and concatenate the result on the final String
            while (res.next()) {
                s = "";
                for (int i = 1; i < res.getMetaData().getColumnCount(); i++) {
                    s += res.getString(i) + ",";
                } // for
                s += res.getString(res.getMetaData().getColumnCount());
              System.out.println(s);
              fS =fS+s+"\n";
           } //while
           // close everything
           res.close();
           stmt.close();
       } catch (SQLException ex) {
       System.out.println(ex);
       //System.exit(0);
       } // try catch
return fS;
} //doQuery
```

Cuadro 5.11. Implementación del método doQuery.

Como se observa, el método **`doQuery`** recibe como argumento un *String* llamado *query*, establece una conexión SQL con Cosmos y lanza la *query* sobre ella, almacenando y concatenando la respuesta obtenida. En cuanto al método **`main`**, se pueden diferenciar tres partes o tareas fundamentales dentro del código: conexión con *Cosmos*, consultas y creación de tablas SQL con cliente Hive y formato de la cadena y descarga en fichero.

5.6.6.2. **Conexión con Cosmos**

Los parámetros necesarios para realizar la conexión con Cosmos son:

- *hiveServer*: dirección IP de Cosmos.
- *hivePort*: puerto de Cosmos, por defecto es el 10000.
- *cosmosUser*: usuario de Cosmos.
- *comosPassword*: contraseña configurada en Cosmos.

Gracias al método **`getConnection`**, se puede establecer una conexión remota con Cosmos. Además, se incorporan dos variables utilizadas para la identificación de la prueba, a saber: número de sensores (**`nOfSensors`**) y periodo entre transmisión de datos (**`sleepTime`**).

5.6.6.3. **Consultas y creación de tablas SQL con cliente Hive**

El cliente Hive implementado permite realizar consultas a Cosmos y crear tablas SQL. En una consulta, se puede delimitar la forma en la que estarán dispuestas las columnas, señalando cual es el carácter de separación entre ellas para poder diferenciarlas. Además, es posible especificar la ruta donde se adquieren los datos y se almacena la tabla. Para automatizar esta tarea, se utiliza un bucle *for* que recorre cada directorio en el que se alojan los datos de la simulación, creando una tabla SQL para cada uno y lanzando una consulta asociada. De la misma forma que en el método **`doQuery`**, se usan dos variables de tipo *String* con objeto de poder almacenar la respuesta e ir concatenando el resultado de las mismas. La variable auxiliar es la llamada **`preResult`**, mientras que el *String* final donde se irán concatenando las respuestas a las consultas se denomina **`result`**.

```
int n = Integer.parseInt(nOfSensors);
System.out.println("-----\nCreando tablas, obteniendo datos\n-----");
for(int i=1; i<=n;i++){
  //Deleting old tables
  doQuery("drop table rmartinezcarreras_sensorID"+i);
  //Creating tables
  String q = "create external table rmartinezcarreras_sensorID"+i+" (recvTs string,
recvT string,   entityId string, entityType string, attrName string, attrType string,
attrValue string, attrMd1 string, attrMd2 string, attrMd3 string, sinkTime string) row
format delimited fields terminated by ',' location
'/user/rmartinezcarreras/def_serv/def_serv_path/room"+i+"_room/'";
  doQuery(q);
  System.out.println("rmartinezcarreras_sensorID"+i);
  //Do Query! We select recvt and attrValue columns
  String preResult = doQuery("select recvT,attrValue,sinkTime from rmartinezcarreras_sensorID"+i);
  result = result + preResult;
}//for
```

Cuadro 5.12. Consultas y creación de tablas SQL con cliente Hive.

Por último, cabe destacar la utilización de *drop tables* para eliminar previamente cualquier tabla existente que tenga el mismo nombre que una de las que se pretende crear. Esta característica permite evitar errores de ejecución del código.

5.6.6.4. **Formato de la cadena y descarga en fichero**

El objetivo del formato no es más que el de dotar de cierta coherencia y organización de los datos para su posterior tratamiento en RStudio. Se ha optado por separar las columnas mediante comas, por lo que se tienen que ir eliminando todos los "residuos" que no sean objeto de estudio. El proceso de formateo ha sido el siguiente:

- Eliminación de comillas dobles.
- Eliminación del nombre de las columnas devueltas por la consulta.
- Sustitución de / por **comas**, este símbolo aparece en los datos enviados por Eclipse.
- Eliminación de la **inicial** de los días de la semana del formato de fecha de *Cosmos*.

La porción de código que lleva a cabo esta tarea es la siguiente:

```
result = result.replace("\"" , "");
result = result.replace("recvTime:" , "");
result = result.replace(",attrValue:" , ",");
result = result.replace("sinkTime:" , "");
result = result.replace("/" , ",");
result = result.replace("X" , ",");
result = result.replace("M" , ",");
result = result.replace("T" , ",");
result = result.replace("W" , ",");
result = result.replace("F" , ",");
result = result.replace("S" , ",");
result = result.replace("}" , "");
```

Cuadro 5.13. Formato de información de *Cosmos*.

Una vez formateada la salida, queda almacenar la información en un fichero. Para ello se utilizan **`FileWriter`** y **`PrintWriter`** de **`java.io`**. La ruta donde se almacena el fichero es: **`/usr/fiware-connectors/resources/hive-basic-client/Fiware_nOfSensors_sleepTime.txt`**
El siguiente fragmento de código realiza esta tarea.

```
//Save using FileWriter
FileWriter fichero = null;
PrintWriter pw = null;
try {
   File archivo = new File("/usr/fiware-connectors/resources/hive-basic-client/Fiware_"
+nOfSensors+"_"+sleepTime+".txt");
   fichero = new FileWriter(archivo);
   pw = new PrintWriter(fichero);
   pw.print(result);
   fichero.close();
} catch (Exception e) {
   e.printStackTrace();
}
```

Cuadro 5.14. Almacenamiento de información de *Cosmos*.

5.6.7. Clase Test.java

La clase **`Test.java`** implementa la interfaz gráfica del simulador de la red de sensores.

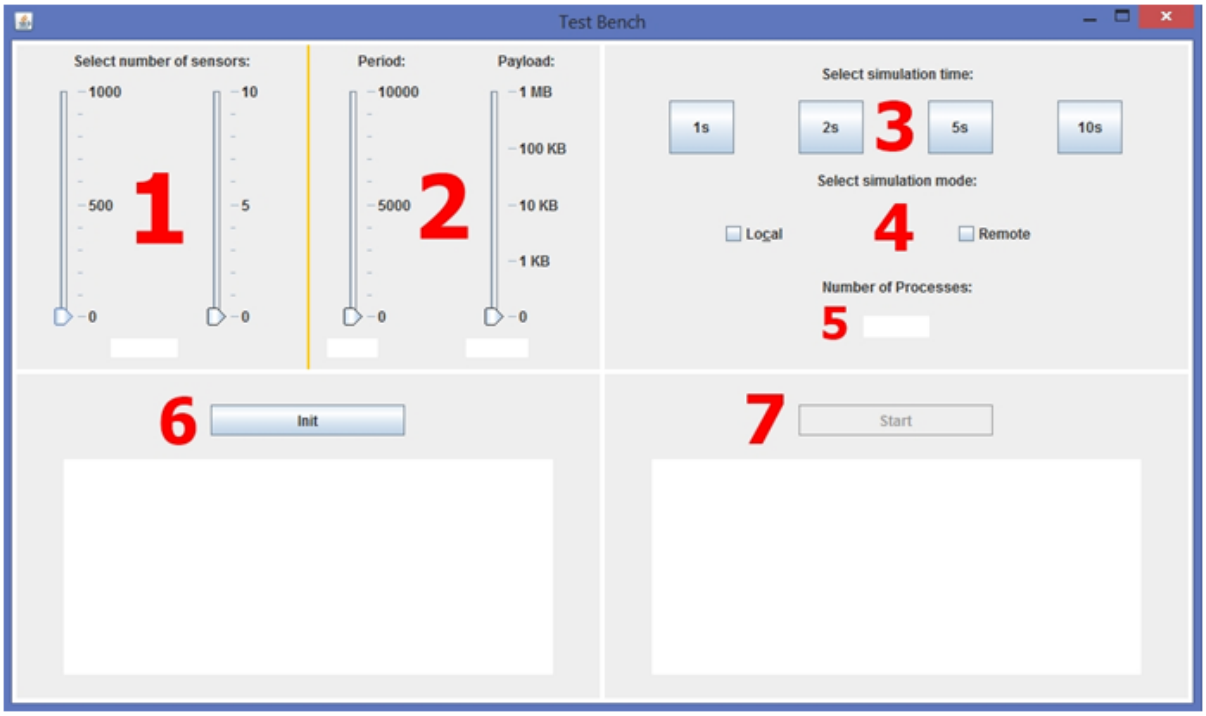


Figura 5.27. Interfaz gráfica del simulador.

La Figura 5.27 muestra la GUI del simulador. Esta interfaz gráfica consta de 7 controles que permiten configurar la simulación de la red de sensores. A continuación, se describe la funcionalidad de los controles implementados:

1. **Menú de Selección del número de sensores**: está formado por dos barras verticales de diferente resolución que tienen el fin establecer el número de sensores de la simulación. La barra de la izquierda permite una selección a mayor escala (0-1000 sensores) mientras que la barra de la derecha es más precisa (0-10 sensores). Así, la configuración final del número de sensores viene determinada por la selección realizada en ambas barras. Por ejemplo, si se selecciona 200 sensores en la barra de la izquierda y 5 sensores en la barra de la derecha, la configuración final es de 205 sensores.
2. **Menú de Selección de periodo y payload**: en este caso consta de dos barras verticales aunque para fines diferentes. La barra de la izquierda permite seleccionar el periodo de envío de datos de la simulación (en ms), mientras que la barra de la derecha permite seleccionar el *payload* que se incorpora en cada trama enviada al *Context Broker*.
3. **Selección del tiempo de simulación**: la configuración del tiempo de simulación se realiza mediante cuatro botones: 1s, 2s, 5s y 10s o un campo de entrada de texto en el que se indica el tiempo en milisegundos. El tiempo de simulación y el periodo determinan el número de solicitudes enviadas por cada sensor virtual al *Context Broker* durante la simulación.

4. **Modo de simulación**: el modo **Local** se corresponde a la ejecución del simulador cuando envía tráfico solo desde un equipo, mientras que el modo **Remote** permite la ejecución del banco de pruebas en varios equipos para simular la generación del tráfico de una red de sensores.
5. **Número de procesos**: este parámetro permite configurar el número de servidores de la red VPN. Para simulaciones en modo **Local** este valor siempre es 1, siendo variable para simulaciones en el modo **Remote**.
6. **Botón de inicialización**: permite crear tantas entidades y suscripciones como se hayan configurado mediante los controles 1 y 5. Así, el número total de entidades es el resultado de multiplicar el número de sensores por el número de procesos.
7. **Botón Start**: al pulsar este botón se inicia la simulación con la configuración realizada mediante los controles 1-6.

El *TextField* que hay situado en la parte inferior del menú asociado al control 6 permite mostrar información relativa a la creación/suscripción de/a entidades en/de el *Context Broker* así como para recordar la inicialización del inyector *Cygnus* para el correcto funcionamiento de la prueba. Por otro lado, el *TextField* situado en la parte inferior del menú asociado al control 7 tiene la función de ofrecer un resumen de la configuración establecida en la simulación (número de paquetes enviados por sensor, número de paquetes totales en la simulación, periodo de la simulación, etc). La Figura 5.28 muestra un ejemplo de la configuración de parámetros de una simulación remota de 300 sensores por proceso, 500ms de periodo, 1 kB de *payload*, 10 segundos de duración con 2 procesos.

Figura 5.28. Ejemplo de configuración de parámetros de simulación.

Finalmente, es importante mencionar que la clase **`SensorV.java`** imprime por consola trazas en el transcurso de la simulación. La siguiente captura (Cuadro 5.15) muestra un ejemplo de esta característica.

```
RUN ---> Iteracion: 48 Tiempo consumido: 110
updateRegister ---> ... calling updateDB
updateDB ----> La longitud de la trama a enviar es de: 254 bytes
updateDB ----> try to write to os ... 254
updateDB ----> conn method is ... POST
updateDB ----> Inicio de la transmision
updateDB ----> Fin de la transmision
updateDB ----> Transmision OK!!!!!!!  ---> 200
RUN ---> Iteracion: 49 Tiempo consumido: 93
DATOS DE LA SIMULACION ---> Entro en el bucle a las 2015-06-29X11:44:33.208 y termino a
las 2015-06-29X11:44:43.177
Tiempo total de la simulacion: 9969
Numero de iteraciones: 50
Numero de paquetes enviados: 50
Numero de fallos en la Tx: 0
Numero de excepciones en metodo connect al iniciar la conexion: 0
Numero de excepciones al tx paquetes en updateDB: 0
Numero de excepciones al desconectar: 0
```

Cuadro 5.15. Salida por consola (clase SensorV.java).

5.7. Arranque y operación

Inicialmente, se arranca *Orion* como servicio del sistema. Después, se arranca la aplicación generadora de contexto, iniciándose la ejecución de 1 proceso maestro y hasta 12 procesos esclavos. El proceso maestro lanza una interfaz gráfica de usuario (GUI) que automatiza la configuración y el arranque del test. En la GUI se introducen los parámetros de la prueba: número de nodos IoT, frecuencia de generación de tráfico NGSI y *payload* que determina el dominio de atributo de las entidades del sistema IoT virtual. Además, el proceso maestro implementa el envío de solicitudes *updateContext* a *Orion* para crear las entidades. Después de arrancar *Cygnus*, también implementa el envío de solicitudes subscribe*Context* tipo "onchange" así como la generación y transmisión *broadcast* de una trama de parámetros de prueba a través de la red.

Cada proceso esclavo decodifica dicha trama, se sincroniza con el proceso maestro y ejecuta uno de los métodos de generación de contexto en un número determinado de clientes NGSI. Como resultado, M nodos IoT virtuales envían información de contexto a *Orion* a través del puerto TCP/1026. Cuando se produce la actualización de un atributo de una *entidad* de la base de datos de *Orion*, se genera un *evento* que se traduce en el envío de una notificación a *Cygnus* a través del puerto TCP/5050.

Por último, el inyector *Cygnus* se encarga de persistir la información en *Cosmos* a través del puerto TCP/14000. Al finalizar la simulación, el proceso maestro recopila todas las medidas de los procesos esclavos realizando operaciones adicionales para obtener las estadísticas del experimento relativas al rendimiento. Estas estadísticas son presentadas al usuario a través de la GUI y almacenadas en un fichero de texto formateado para ser cargado en el entorno de desarrollo *RStudio* [129]. El cliente Hive permite realizar consultas al histórico de la información almacenada en *Cosmos* a través del puerto TCP/10000.

CAPÍTULO 6

Resultados

The IoT links objects to the Internet, enabling data and insights never available before.

Cisco
Internet of Things (IoT) solutions

6.1. Plan de Pruebas

Atendiendo al *payload* de las solicitudes *updateContext* enviadas a *Orion*, se han realizado cuatro tipos de prueba (1 kB, 10 kB, 100 kB y 1 MB). Cada una de estas pruebas se divide en dos categorías en función del método de generación de contexto utilizado (bloqueante o no bloqueante). La primera categoría comprende las pruebas de rendimiento que emplean el método bloqueante para medir los parámetros de *throughput* (expresado en solicitudes por segundo o *kilobytes* por segundo) y de *Round-Trip Time* (expresado en milisegundos). La segunda categoría engloba las pruebas de rendimiento que utilizan el método no bloqueante para medir el parámetro de *throughput* (expresado en solicitudes por segundo o *kilobytes* por segundo). Se han lanzado 20 simulaciones por categoría en diferentes días y franjas horarias. Por estos motivos, el plan de pruebas supone la realización de más de 400 simulaciones. En la siguiente sección se muestran los resultados representativos de cada tipo de prueba.

El procedimiento seguido en los tests es el siguiente. Los nodos IoT son modelados mediante hilos asociados a procesos. En las simulaciones iniciales, partimos de 12 procesos mono-hilo, cada uno ejecutado en una máquina diferente. Cuando el número de nodos IoT es superior a 12, se arrancan nuevos hilos en los procesos. Para simplificar las pruebas, se simulan múltiplos del número de procesos arrancados para las pruebas, generalmente 12, con el objetivo de repartir equitativamente la carga de simulación entre los procesos. En determinados casos, para afinar los resultados, se ha trabajado con un menor número de máquinas y por tanto de procesos (siempre un proceso por máquina). Por ello, en las gráficas se pueden ver medidas relativas a un número de nodos que no es múltiplo de 12. Una simulación finaliza cuando cada nodo IoT virtual ha intentado enviar 200 solicitudes al *Orion Context Broker*. Los valores de *throughput* y *Round-Trip Time* de la simulación se calculan a partir de la media de las 200 medidas obtenidas.

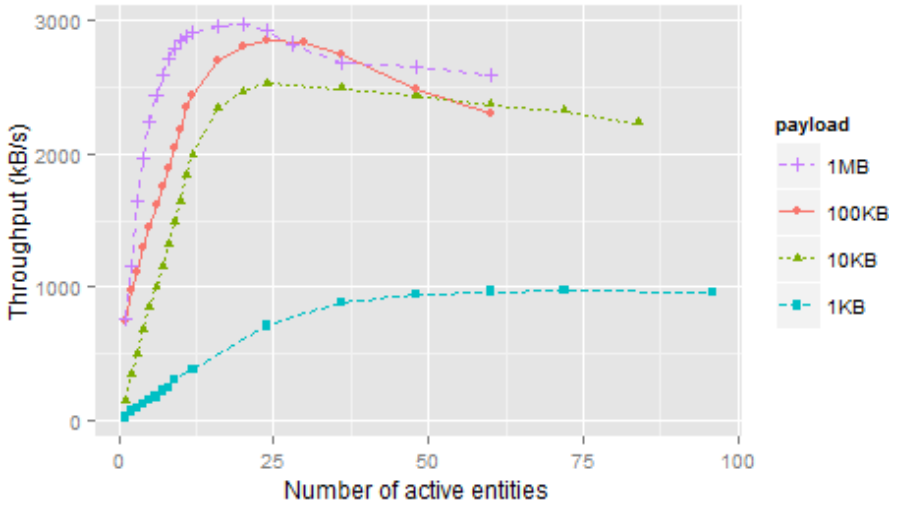
6.2. Evaluación de FIWARE

Con el fin de evaluar el rendimiento del *Orion Context Broker*, se ha representado gráficamente el *throughput* y la latencia en función del grado de concurrencia simulado en cada tipo de prueba. Para estudiar el comportamiento del servidor RESTful de *Orion* se han realizado numerosas simulaciones. La realización de estas simulaciones fue una tarea costosa que requirió mucho tiempo debido a la latencia de la red. Los resultados obtenidos no son exactos. Sin embargo, ofrecen una vista realista del rendimiento de este *Generic Enabler* ante diferentes despliegues y condiciones de carga.

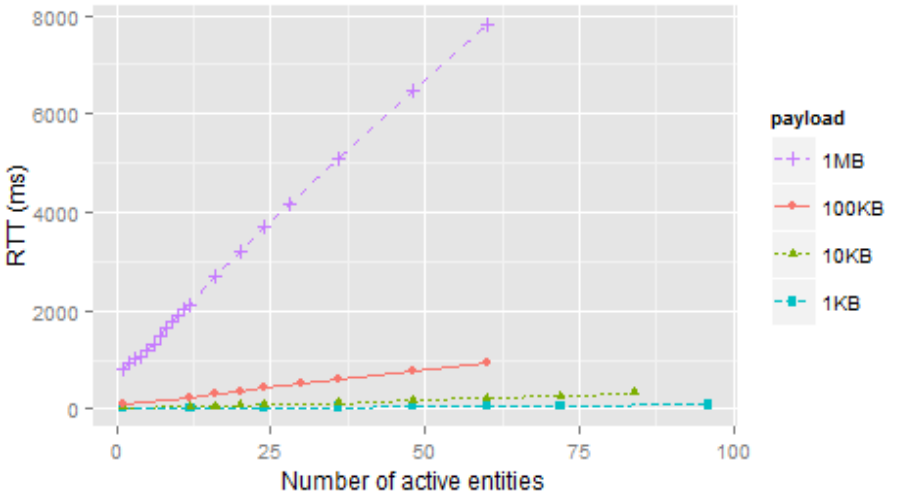
Las Figura 6.1 ilustra los resultados obtenidos al utilizar el método bloqueante para generar las solicitudes de contexto correspondientes a cada *payload* establecido en el plan de pruebas (1 kB, 10 kB, 100 kB y 1 MB). Un nuevo mensaje es enviado tan pronto como el anterior ha sido completamente transmitido. La Figura 6.1.a-c ilustra el *throughput* (Th) medido en términos de *kilobytes* por segundo (kB/s) o de solicitudes atendidas por segundo (req/s) y la latencia expresada como *Round-Trip Time* (RTT) medida en milisegundos (ms) frente al número de clientes NGSI, que pueden ser sensores (*edge devices* en general) o nodos intermedios. Los resultados de las simulaciones serán interpretados de forma diferente en función de si las entidades activas (*active entities*) son sensores o nodos intermedios. Por ahora, los resultados se presentan y luego serán discutidos. A partir de estas gráficas, es posible distinguir tres rangos de interés tal y como se muestra en las Tablas 6.1–6.3. Las tablas son incluidas para mostrar claramente los puntos de interés sin sobrecargar las figuras. Estos rangos proporcionan información del comportamiento del *Orion Context Broker*. Las gráficas muestran que *Orion* tiene un rango inicial (*close to linear*) en el que el *throughput* aumenta a una tasa más o menos lineal y el RTT se mantiene constante o lineal. El segundo rango de interés (*increase slow/fast*) se corresponde con un aumento lento del *throughput* hasta alcanzar su máximo valor mientras que el RTT se incrementa a un ritmo rápido. A partir del *throughput* máximo, el rendimiento del sistema se degrada a medida que aumentan los parámetros de *payload* y de concurrencia, dando lugar al tercer y último rango de interés (*degrading performance*). En este rango, los parámetros mencionados tienen la tendencia de incrementar el RTT (tiempo de espera) y decrementar el *throughput* (solicitudes procesadas). Este era el resultado esperado, lo interesante es ver cuándo se producen los cambios de comportamiento, siendo la referencia más interesante el máximo valor de *throughput*.

Para analizar con un mayor detalle el rendimiento de *Orion*, se estudian los valores máximos del *throughput* obtenidos en cada tipo de prueba. Este parámetro puede ser evaluado desde dos puntos de vista: datos transferidos y solicitudes procesadas. Desde el primer punto de vista (datos transferidos, ver Figura 6.1.a y Tabla 6.2):

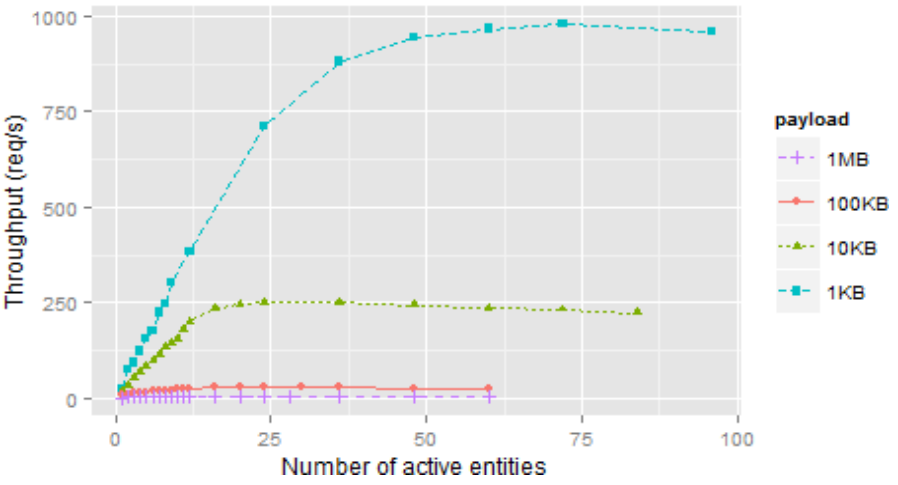
- La mayor tasa de transferencia de datos es 2963 kB/s, obtenida al simular un *payload* de 1MB y 20 clientes NGSI (Tabla 6.1). El RTT medido en esta simulación es 3200 ms (Figura 6.1.b).
- El segundo mejor resultado es de 2846 kB/s, obtenido al simular un *payload* de 100 kB y 24 clientes NGSI. Sin embargo, el RTT obtenido es de 430 ms, claramente muy inferior al caso anterior.



(**a**)



(**b**)



(**c**)

Figura 6.1. Resultados obtenidos utilizando el método bloqueante. (a) Mayor tasa de transferencia a mayores cargas con pocos nodos; (b) El sistema exhibe una mayor latencia a medida que se aumenta la carga y la concurrencia (número de entidades activas); (c) Mejor rendimiento respecto del número de solicitudes atendidas y peor respecto del payload.

Region	1kB	10kB	100kB	1MB
Close to linear	[1 .. 24]	[1 .. 12]	[1 .. 12]	[1 .. 8]
Increase slow/fast	[24 .. 72]	[12 .. 24]	[12 .. 24]	[8 .. 20]
Degrading performance	[72 ..	[24 ..	[24 ..	[20 ..

Tabla 6.1. Número de entidades activas simuladas en cada tipo de prueba.

Region	1kB	10kB	100kB	1MB
Close to linear	[25 .. 712]	[157 .. 1996]	[749 .. 2437]	[760 .. 2640]
Increase slow/fast	[712 .. 978]	[1996 .. 2520]	[2437 .. 2846]	[2640 .. 2963]
Degrading performance	[978 ..	[2520 ..	[2846 ..	[2963 ..

Tabla 6.2. *Throughput* obtenido en *kilobytes* por segundo.

Region	1kB	10kB	100kB	1MB
Close to linear	[25 .. 712]	[15.7 .. 199.6]	[7.49 .. 24.37]	[0.76 .. 2.64]
Increase slow/fast	[712 .. 978]	[199.6 .. 252]	[24.37 .. 28.46]	[2.64 .. 2.89]
Degrading performance	[978 ..	[252 ..	[28.46 ..	[2.89 ..

Tabla 6.3. *Throughput* obtenido en solicitudes por segundo.

- La siguiente medida de *throughput* en orden descendente es 2520 kB/s, correspondiente a la simulación de *payload* 10 kB y 24 clientes NGSI. Aunque se ha utilizado el mismo número de entidades activas que en el caso anterior, el RTT calculado es inferior (83 ms).
- Finalmente, la menor tasa de transferencia de datos (978 kB/s) se corresponde con la simulación de *payload* 1 kB y 72 clientes NGSI. Cabe señalar que ha sido necesario emplear un número muy superior de entidades activas en comparación con el resto de simulaciones. Además, el RTT (72 ms) es el menor valor medido de los casos estudiados en este análisis.

En otras palabras, el sistema se comporta mejor ante grandes cargas generadas por pocos nodos. El incremento en el número de nodos provoca una disminución muy significativa en el volumen de datos que puede ser transferido. Desde el segundo punto de vista (solicitudes atendidas por segundo, ver Figura 6.1.c y Tabla 6.3):

- La menor tasa de solicitudes procesadas es de 2,89 req/s, en el caso de *payload* 1 MB.
- El segundo peor resultado es de 28,46 req/s para un *payload* de 100 kB.
- El siguiente resultado obtenido (252 req/s) para un *payload* de 10 kB, es bastante mejor.
- Finalmente, la medida del *throughput* obtenida (978 req/s) en la simulación de un *payload* de 1 kB es muy superior al resto de casos de estudio.

Observamos que aquí el comportamiento es justo el contrario que en el caso anterior. El sistema se comporta mejor respecto del número de solicitudes y peor respecto del *payload*. Para verificar si es posible un mayor valor de *throughput* en términos de solicitudes por

segundo, se ha realizado una prueba con un *payload* mínimo de 300 *bytes*. Los resultados obtenidos son similares a los del caso de *payload* de 1 kB. Por esta razón no han sido representados. Por debajo de 1 kB de *payload* no se observan mejoras significativas.

Los escenarios densamente poblados por sensores como el estudiado en el capítulo 4 implican una tasa alta de actualización. Por este motivo, se requiere un servidor RESTful que soporte un alto grado de concurrencia y, al mismo tiempo, procese un gran número de solicitudes casi en tiempo real. Los resultados descritos revelan que la mejor elección para conseguir el máximo rendimiento de *Orion* en este tipo de escenarios es la de un *payload* de tamaño comprendido entre 1 kB y 10 kB. El *payload* final dependerá de las necesidades de cada aplicación IoT (grado de concurrencia estimado, dominio de atributo, etc). Un escenario interesante es estudiar el comportamiento de FIWARE ante solicitudes bloqueantes y no bloqueantes.

Las solicitudes bloqueantes son típicas de los nodos intermedios, especialmente la conexión entre los *Generic Enablers IoT Broker* y *Orion Context Broker* (ver sección 4.3). Las solicitudes no bloqueantes son típicas de los sensores, o nodos finales IoT, que por razones de eficiencia de energía no pueden ser bloqueados a la espera de una respuesta del servidor. Normalmente, un sensor envía una solicitud asíncrona y cierra la conexión para ahorrar energía. Esto simula el comportamiento de un nodo final IoT que no mantiene la conexión, cuando no es necesario para conservar energía.

Para simular la generación de tráfico NGSI procedente de los nodos finales IoT, el método de generación de contexto no bloqueante debe ser utilizado a fin de permitir la generación de tráfico NGSI de una manera asíncrona, con picos de carga (ráfagas) que excedan la capacidad del servidor de *Orion*. El propósito de esta prueba es medir el *throughput* máximo en términos de solicitudes atendidas por segundo o *kilobytes* por segundo para ambos tipos de solicitudes, bloqueantes y no bloqueantes. Las pruebas fueron realizadas con un *payload* de 1kB, es decir, aquel que permitió la simulación del más alto grado de concurrencia con el método bloqueante. Por lo tanto, las simulaciones bloqueantes y no bloqueantes se llevaron a cabo con el parámetro óptimo de carga para el caso bloqueante. Esto facilita las comparaciones.

La Figura 6.2 ilustra el *throughput* en condiciones bloqueantes y no bloqueantes. En el caso de solicitudes bloqueantes, cada solicitud bloqueante es tratada en un hilo independiente. De esta manera, muchas solicitudes bloqueantes pueden ser simuladas simultáneamente. En el caso de las solicitudes no bloqueantes, cada *socket* no bloqueante abre una conexión con *Orion* cada 50 ms. Como indica la gráfica, *Orion* tiene un rango inicial en el que el *throughput* aumenta casi linealmente hasta un máximo de 133,2721 solicitudes o *kilobytes* por segundo a una concurrencia de 9. A partir de este valor, *Orion* entra en sobrecarga reduciendo su *throughput* a medida que aumenta la concurrencia. La simulación de 30 *sockets* no bloqueantes es una indicación clara de cómo este estado afecta al *throughput* (83,4611 req/s).

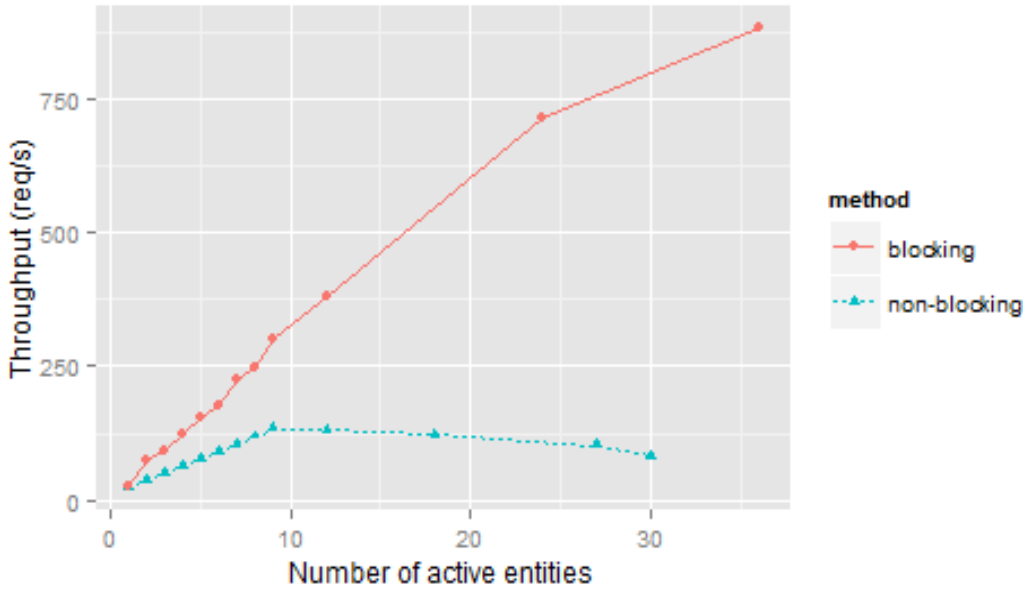


Figura 6.2. *Throughput* obtenido utilizando los métodos boqueante y no bloqueante.

Si comparamos el *throughput* máximo medido cuando el tráfico es generado por medio de los dos métodos del banco de pruebas, vemos que el valor obtenido con el método no bloqueante es significativamente inferior que el obtenido con el método bloqueante. Queremos hacer notar que en el método bloqueante de generación de contexto, cada cliente NGSI genera una solicitud *updateContext* una vez que se haya completado la anterior. Los clientes se mantienen en sincronía con el servidor y la tasa de solicitudes generadas nunca excede la capacidad del servidor de *Orion*.

Parte III

Conclusiones y Aportaciones

CAPÍTULO 7

Conclusiones, trabajos futuros y lista de publicaciones

It's not so much about the emergence of new technology, it's the convergence — the ability to use sensors for everything in the world to basically be a computer, whether it's your contact lens, your hospital bed, or a railway track.

Harriet Green
IBM General Manager

7.1. Conclusiones y trabajos futuros

Este trabajo de investigación ha mostrado un caso de uso de FIWARE con suficiente nivel de detalle para proporcionar una visión precisa de cómo se desarrolla una aplicación FIWARE. Es una tecnología compleja ya que está diseñada para adaptarse a la variabilidad de una amplia gama de aplicaciones, que exigen diferentes esquemas de interacción. Al mismo tiempo, FIWARE adopta aplicaciones dirigidas por eventos, que pueden mostrar un comportamiento asíncrono. Todas estas características implican inevitablemente el uso de muchos componentes con interdependencias. La arquitectura FIWARE, que define estas dependencias, se basa en patrones de diseño comunes en las comunicaciones *software*, principalmente el uso de un *broker*. El uso de patrones de diseño comunes hace que la comprensión de la tecnología FIWARE y su uso sea más fácil para los desarrolladores de aplicaciones. Sin embargo, en nuestra opinión, la documentación no está bien organizada. Además, no siempre ofrece suficiente nivel de detalle para dar una respuesta específica a un problema de implementación. Las deficiencias en la documentación implican invertir mucho tiempo y esfuerzo en resolver problemas relativamente simples.

En este sentido, creemos que, este trabajo, sin entrar en detalles de codificación, proporciona una guía útil para identificar los componentes necesarios para desarrollar una aplicación FIWARE, así como las interfaces utilizadas para conectarlos.

Respecto al rendimiento de las aplicaciones resultantes y en base a los resultados obtenidos al utilizar el banco de pruebas desarrollado, se puede afirmar que la elección del *payload* es decisiva en el rendimiento del *Orion Context Broker*. Tal y como se ha descrito en el capítulo 6, un *payload* óptimo permite maximizar el rendimiento de *Orion* en términos de *throughput* y latencia. Además, dicho *payload* permite que *Orion* soporte un mayor grado de concurrencia.

Las aplicaciones de agricultura de precisión, como muchas otras de IoT, se caracterizan por dos tipos de situaciones: (1) intercambios muy frecuentes de pequeñas cantidades de datos procedentes de sensores de recursos limitados, (2) transmisión masiva de datos *offline*, por ejemplo, datos adquiridos desde un dron o desde una cámara de un vehículo agrícola. En ambos casos FIWARE ofrece un buen comportamiento.

Por todo esto, se puede concluir que la plataforma FIWARE satisface los requerimientos de las aplicaciones IoT actuales en el ámbito de la agricultura de precisión.

En otros dominios, existen ciertas aplicaciones que requieren un envío de datos masivo en tiempo real (*streaming* en tiempo real) acompañado del procesamiento de tales datos *online*, a medida que se van generando. Aparte del *Orion Context Broker*, FIWARE también soporta otros *Generic Enablers* como Kurento [130], *Complex Event Processing* [131] y Cosmos [132] para implementar soluciones en tales escenarios. Por tanto, sería interesante una evaluación del rendimiento de FIWARE en este tipo de escenarios (en adelante escenarios tipo 2).

En el caso de escenarios IoT futuros, en el capítulo 4 se ha descrito un *Farm Management System* (FMS) que integra los elementos de la arquitectura FIWARE. El FMS aborda los desafíos de escalabilidad e interoperabilidad que han sido estudiados en el capítulo 2. Por un lado, los *Generic Enablers Protocol Adapter* y *Backend Device Management* se encargan de ocultar al resto de componentes los protocolos de comunicación con los dispositivos físicos (Zigbee, MQTT, CoAP, etc). Por otro lado, el *Orion Context Broker* es el componente más importante ya que maneja toda la información de contexto del sistema, intermediando entre los elementos que producen dicha información y los elementos que la consumen. Respecto a los aspectos relativos a la escalabilidad, en el *Orion Context Broker* pueden considerarse 2 dimensiones:

- Escalabilidad en el número de *entidades*. En este caso, el recurso crítico (escaso) es la base de datos (DB), por lo que sería necesario escalar la capa de MongoDB. El procedimiento habitual para escalar MongoDB es mediante el uso de *shards* descrito en la documentación oficial de MongoDB [133].
- Escalabilidad en las solicitudes de operación para manejar las entidades. En este caso, se pueden usar instancias *Orion Context Broker* adicionales (cada uno ejecutándose en una VM diferente), además de una VM adicional delante de ellos ejecutando el *software load balancer* [112] para distribuir la carga entre los nodos Orion.

En cuanto a la persistencia de datos en *Cosmos* a través de *Cygnus*, de la misma forma que se puede crear una *farm* de procesos *Orion*, se podrían añadir más procesos *Cygnus* encargados de recibir las notificaciones de la *Orion farm*. Para ello, es recomendable definir una estrategia de *mapping* para las *entidades* del sistema. Así, se establecen las suscripciones correspondientes al conjunto A de *entidades* que van a notificar al proceso *Cygnus* A, a otro conjunto de *entidades* B que va a notificar al proceso *Cygnus* B, etc.

Otra característica importante que puede ofrecer el *Generic Enabler Data Handling* es la optimización de tráfico enviado al IoT *Backend*. Esta funcionalidad permitiría alcanzar una mayor eficiencia y confiabilidad, al utilizar técnicas de CEP [112] [134]. En resumen, sería

posible incrementar la eficiencia del sistema al inyectar al *backend* una información de contexto previamente procesada y, por tanto, más relevante para la aplicación IoT. Así, El CEP, también presente en el capítulo *Data/Context Management* [118], permitiría aliviar el tráfico enviado hacia *Cygnus*.

Resumiendo, se puede concluir que la combinación de: (1) un ajuste óptimo del rendimiento de los componentes de FIWARE, (2) los mecanismos provistos por cada componente, (3) la escalabilidad horizontal de cada instancia en la nube y (4) la escalabilidad vertical de la plataforma hace posible el desarrollo de aplicaciones agrícolas en escenarios futuribles como el descrito en el capítulo 4. Esta conclusión nos permite cumplir con el último de los objetivos fijados al inicio de esta tesis.

Aunque el banco de pruebas está orientado a la medida del rendimiento y de la capacidad de almacenamiento de datos, es extensible para incorporar otras propiedades. Por este motivo, cabe la posibilidad de profundizar en el estudio de la plataforma FIWARE (análisis de los escenarios tipo 2 mencionados anteriormente, *streaming* con FIWARE) o realizar una amplia variedad de estudios centrados en los *middleware* IoT actuales. Además, toda la infraestructura de *software* que simula los sensores y las diferentes formas en las que generan los datos (*payload*, número de procesos e hilos, etc.) se pueden utilizar fácilmente para evaluar otros *middleware* de IoT. Al hacerlo, podremos comparar los resultados obtenidos con FIWARE con otras plataformas *middleware*, como WSO2, PubNub ... abriendo una futura vía de investigación a través del uso del banco de pruebas con otras plataformas que permitan estudios comparativos.

7.2. Lista de Publicaciones

En esta sección se recoge una lista de las publicaciones más relevantes en las que han aparecido los distintos resultados presentados en esta tesis.

Publicaciones en Revistas Científicas

1. Ramón Martínez, Juan Ángel Pastor, Bárbara Álvarez y Andrés Iborra. A Testbed to Evaluate the FIWARE-Based IoT Platform in the Domain of Precision Agriculture. Sensors **2016**; 16(11), 1979. DOI: 10.3390/s16111979

Congresos Nacionales

2. Andrés Iborra, Andrés Carrillo, Bárbara Álvarez, Pedro Sánchez y Ramón Martínez. Cloud Incubator HUB: Una experiencia universitaria innovadora para la creación de startups en el ámbito de Internet de las Cosas. En Actas del XII Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (TAEE), Sevilla, 22-24 Jun. 2016.
3. Ramón Martínez, Juan Ángel Pastor, Bárbara Álvarez y Andrés Iborra. Diseño e implementación de un banco de pruebas para evaluar plataformas middleware de Internet de las Cosas: un estudio acerca de FIWARE en el ámbito de la agricultura de precisión. IX Anuario de Jóvenes Investigadores, Cartagena: Universidad Politécnica de Cartagena, nº9, 2016. 250 p. ISSN: 2386-3676

7.3. Proyectos de investigación y concursos

Proyectos de Investigación

- Participación en el proyecto europeo Startup-Scaleup coordinado por la incubadora y aceleradora Cloud Incubator HUB. Las tareas realizadas se enmarcan en el ámbito de la mentorización y formación especializada en Internet de las Cosas.
- Participación en la redacción de propuestas de investigación para el programa "*FIWARE Accelerator*", el cual requiere pymes y emprendedores de Internet para desarrollar servicios y aplicaciones altamente innovadoras. Específicamente, se ha colaborado con la spin-off Widhoc para la realización de trabajos en el ámbito de Internet de las Cosas ideando soluciones en los dominios *Smart Cities*, *Energy&Environment* y *Agrifood* al que pertenecen las aceleradoras *SOUL-FI* y *SmartAgriFood*.

Concursos

- Participación en el concurso "FIWARE Grant for FI-PPP Liaison". En el mismo, se presentó la propuesta "*Agricultural Management System*" que fue una de las 10 finalistas. Además, esta propuesta obtuvo la cuarta mejor puntuación del tribunal calificador. Aunque sólo las tres propuestas mejor calificadas fueron premiadas, mediante este resultado se consiguió un curso de adiestramiento impartido por los técnicos de una plataforma emergente de Internet de las Cosas (IoT) denominada FIWARE. El sistema propuesto en el concurso ofrecía una solución a la problemática global del uso inadecuado e ineficiente del agua en prácticas agrícolas gracias al despliegue de redes de sensores así como de la tecnología IoT de FIWARE.

Bibliografía

[1] M. Conti et al., "Looking ahead in pervasive computing: challenges and opportunities in the era of cyber-physical convergence," *Pervas.Mob.Comput. 8 (1)*, pp. 2-21, 2012.

[2] R. Poovendran, "Cyber-physical systems: close encounters between two parallel worlds [point of view]," *Proc. IEEE 98 (8)*, pp. 1363-1366, 2010.

[3] K.-J. Park, R. Zheng, and X. Liu, "Cyber-physical systems: milestones and research challenges," *Comp. Commun. 36 (1)*, pp. 1-7, 2012.

[4] G. Hackmann, F. Sun, N. Castaneda, C. Lu, and S. Dyke, "A holistic approach to decentralized structural damage localization using wireless sensor networks," *Comp. Commun. 36 (1)*, pp. 29–41, 2012.

[5] S. Barro-Torres, T.M. Fernandez-Carames, H.J. Perez-Iglesias, and C.J. Escudero, "Real-time personal protective equipment monitoring system," *Comp. Commun. 36*, pp. 42-50, 2012.

[6] G. Schirner, D. Erdogmus, K Chowdhury, and T. Padir, "The future of human-in-the-loop cyber-physical systems," *computer 99 (PrePrints)*, p. 1, 2012.

[7] A. Passarella, R. I. Dunbar, M. Conti, and F. Pezzoni, "Ego network models for future internet social networking environments," *Comp. Commun. 35 (18)*, pp. 2201–2217, 2012.

[8] McKinsey Global Institute, "Disruptive technologies: advances that will transform life, business, and the global economy," Executive Summary 2013.

[9] M. Weiser, "The computer for the 21st century," *Scientific american, 265(3)*, pp. 94-104, 1991.

[10] M. Weiser, "Some computer science issues in ubiquitous computing," *Commun. ACM, 36(7)*, pp. 75-84, doi: 10.1145/159544.159617, 1993.

[11] K. Ashton, "That ‘Internet of Things’ Thing," *RFID Journal*, 2009.

[12] D. Bandyopadhyay and J. Sen, "Internet of Things – Applications and Challenges in Technology and Standardization," *Journal of Wireless Personal Communications (58)*, pp. 49-69, 2011.

[13] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks (10)*, pp. 1497–1516, 2012.

[14] E. Borgia, "The Internet of Things vision: Key features, applications and open issues," *Computer Communications (54)*, pp. 1-31, 2014.

[15] F. daCosta, *Rethinking the Internet of Things. A Scalable Approach to Connecting Everything.*: Apress, 2013, ISBN: 978-1-4302-5740-0.

[16] S. Rodríguez-Valenzuela and J. Holgado-Terriza, *Plataforma de Servicios Semánticos Sensibles al Contexto para Sistemas de Inteligencia Ambiental*., 2015.

[17] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: a survey," *Comput. Netw. 54 (15)*, pp. 2787–2805, 2010.

[18] J. Domingue et al., "The Future Internet. Future Internet Assembly 2011: Achievements and Technological Promises," pp. 6656, doi: 10.1007/978-3-642-20898-0.

[19] Moor Insights and Strategy. (2013) Behaviorally segmenting the Internet of Things (IoT). [Online]. http://www.moorinsightsstrategy.com/wp-content/uploads/2013/10/Behaviorally-Segmenting-the-IoT-by-Moor-Insights-Strategy.pdf

[20] G. Gorbach, C. Polsonetti, and A. Chatha. (2014, Nov.) Planning for the industrial Internet of Things. [Online]. http://www.arcweb.com/brochures/planning-for-the-industrial-internet-of-things.pdf

[21] M. Scott and R. Whitney. (2014, Nov.) The industrial Internet of Things. [Online]. http://www.mcrockcapital.com/uploads/1/0/9/6/10961847/mcrock_industrial_internet_of_things_report_2014.pdf

[22] J. Höller et al., *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence.*: Academic Press. Elsevier, ISBN: 978-0-12-407684-6, 2014.

[23] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," *IEEE Internet Comput.14*, pp. 44-51, 2010.

[24] G. Roussos and V. Kostakos, "RFID in pervasive computing: State-of-the-art and outlook," *Pervasive Mob. Comput. 5*, pp. 110-131, http://dx.doi.org/10.1016/j.pmcj.2008.11.004, 2009.

[25] I. Akyilidiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Netw. 38 (4)*, pp. 393-422, 2002.

[26] I. Akyildiz and I. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Netw. J. 2*, pp. 351-367, 2004.

[27] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: discovery, query, selection,and on-demand provisioning of Web services," *IEEE Trans. Serv. Comput. 3 (3)*, pp. 223-235, 2010.

[28] L. Chen, M. Tseng, and X. Lian, "Development of foundation models for Internet of

Things," *Front. Comput. Sci. China 4*, pp. 376-385, 2010.

[29] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices," *Springer*, p. Chapter 5, 2011.

[30] WSO2 platform. [Online]. http://wso2.com/

[31] PubNub platform. [Online]. https://www.pubnub.com/

[32] FIWARE platform. [Online]. FIWARE platform: https://www.fiware.org/

[33] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé. (2010) Vision and Challenges for Realising the Internet of Things. [Online]. http://www.internet-of-thingsresearch.eu/pdf/IoT_Clusterbook_March_2010.pdf

[34] V. Ovidiu and F. Peter, "Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems," *River Publishers Series in Communications*, 2013.

[35] H. Zhou, *The Internet of Things in the Cloud: A Middleware Perspective*.: CRC Press. Taylor & Francis Group, 2013.

[36] J. Gubbi, B. Rajkumar, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, pp. 1645-1660, http://dx.doi.org/10.1016/j.future.2013.01.010, 2013.

[37] D.J. Mulla, "Twenty Five Years of Remote Sensing in Precision Agriculture: Key Advances and Remaining Knowledge Gaps," *Biosystems Engineering*, pp. 358-371, doi:10.1016/j.biosystemseng.2012.08.009, 2013.

[38] Alliance For Internet Of Things Innovation, "Smart Farming and Food Safety Internet Of Things Applications-Challenges for Large Scale Implementations ," *AIOTI WG06-Smart Farming and Food Safety*, 2015.

[39] J.A. López et al., "Development of a Sensor Node for Precision Horticulture," *Sensors, 9*, pp. 3240-3255, 2009.

[40] J.A. López, F. Soto, J. Suardíaz, A. Iborra, and J. A. Vera, "Wireless Sensor Networks for Precision Horticulture in Southern Spain," *Computer and Electronics in Agriculture, 68 (1)*, pp. 25-35, 2009.

[41] J. A. López et al., "Design and Validation of a Wireless Sensor Network Architecture for Precision Agriculture Applications," *Precision Agriculture, 12 (2)*, pp. 280-295, 2011.

[42] J. M. Molina-Martinez et al., "VIPMET: New Real-Time Data Filtering-Based Automatic Agricultural Weather Station," *Journal of Irrigation and Drainage*

Engineering,138, pp. 823-829, 2012.

[43] D. Havlik, S. Schade, W. van Wijk, T. Usländer, and J. Hierro, "Leveraging the Future Internet for the Environmental Usage Area," *EnviroInfo: Innovations in Sharing Environmental Observations and Information*, 2011.

[44] J.A. López-Riquelme, N. Pavón-Pulido, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez. (2016) A software architecture based on FIWARE cloud for Precision Agriculture. [Online]. http://www.sciencedirect.com/science/article/pii/S0378377416304061

[45] Ericsson mobility report. (2015) [Online]. http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf

[46] A. Gavras, A. Karila, S. Fdida, M. May, and M Potts, "Future Internet research and experimentation: The FIRE initiative," *Comput. Commun. Rev.*, pp. 37 (3), 89–92, 2007.

[47] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W.D. Meuter, "Middleware for the Internet of Things, design goals and challenges," *Electron. Commun. EASST*, p. 28, 2010.

[48] P. Bellavista, G. Cardone, A. Corradi, and Foschini L., "Convergence of MANET and WSN in IoT urban scenarios," *IEEE Sensors J.*, pp. 13 (10), 3558–3567, 2013.

[49] A. Bahga and V. Madisetti. (2014) Internet of Things: A Hands-On Approach. [Online]. www.internet-of-things-book.com

[50] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, pp. 3(1), 70-95, 2016.

[51] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, pp. 16 (1), 414–454, 2013.

[52] S.-H. Sho et al., "Ttcg: three-tier context gathering technique for mobile devices," in *Proceedings of the 5th international conference on Pervasive services*, New York, 2008, pp. 157–162, http://doi.acm.org/10.1145/1387269.1387296.

[53] ITU-T. (2008) Ubiquitous Sensor Networks (USN). [Online]. http://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000040001PDFE.pdf

[54] freescale/arm, "What the Internet of Things (IoT) Needs to Become a Reality," *White Paper*, INTOTHNGSWP REV 2, 2014.

[55] Numerex DNA. The Numerex Smart Supply Chain. [Online].

http://www.numerex.com/inside-our-DNA

[56] J. Holdowsky, M. Mahto, M. Raynor, and M. Cotteleer, "A primer on the technologies building the IoT," *Deloitte University Press*, 2015.

[57] IDTechEx. (2013) Wireless Sensor Networks (WSN) 2014-2024: Forecasts, Technologies, Players. The new market for Ubiquitous Sensor Networks (USN). [Online]. http://www.idtechex.com/research/reports/wireless-sensor-networks-wsn-2014-2024-forecasts-technologies-players-000382.asp

[58] MNX. (2016) Microelectromechanical Systems. [Online]. https://www.mems-exchange.org/MEMS/what-is.html

[59] W. K. Seah, Z. A. Eu, and H. P. Tan, "Wireless sensor networks powered by ambient energy harvesting (WSN-HEAP)-Survey and challenges," *Wireless VITAE. IEEE 1st International Conference*, pp. 1-5, 2009.

[60] Z. G. Wan, Y. K. Tan, and C. Yuen, "Review on energy harvesting and energy management for sustainable wireless sensor networks," *Communication Technology. IEEE 13th International Conference*, pp. 362-367, 2011.

[61] R. Vullers, R. Schaijk, H. Visser, J. Penders, and C. Hoof, "Energy harvesting for autonomous wireless sensor networks," *IEEE Solid-State Circuits Magazine*, pp. 2(2), 29-38, 2010.

[62] Bluetooth SIG. [Online]. https://www.bluetooth.com/

[63] Wi-Fi Alliance. [Online]. http://www.wi-fi.org/

[64] ZigBee Alliance. [Online]. http://www.zigbee.org

[65] Gil Reiter, "Wireless connectivity for the Internet of Things," *Texas Instruments*, 2014.

[66] DASH7 Alliance. [Online]. http://www.dash7-alliance.org/

[67] Z-Wave. [Online]. http://www.z-wave.com/

[68] IETF 6LowPAN. [Online]. https://datatracker.ietf.org/wg/6lowpan/charter/

[69] IEEE 802.15.4. [Online]. https://standards.ieee.org/about/get/802/802.15.html

[70] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 2, no. 35, pp. 114-131, 2003.

[71] IEEE 802.3 Working Group. [Online]. http://www.ieee802.org/3/

[72] IEEE 802.11 Working Group. [Online]. http://www.ieee802.org/11/

[73] IEEE 802.16 Working Group. [Online]. http://www.ieee802.org/16/

[74] IEEE 802.15 Working Group. [Online]. http://www.ieee802.org/15/

[75] Internet Protocol Specification. [Online]. https://www.ietf.org/rfc/rfc791.txt

[76] Internet Protocol, Version 6 (IPv6) Specification. [Online]. https://www.ietf.org/rfc/rfc2460.txt

[77] Transmission Control Protocol. [Online]. https://www.ietf.org/rfc/rfc793.txt

[78] User Datagram Protocol. [Online]. https://www.ietf.org/rfc/rfc768.txt

[79] Hypertext Transfer Protocol-HTTP/1.1. [Online]. https://tools.ietf.org/html/rfc2616

[80] Constrained Application Protocol (CoAP). [Online]. https://tools.ietf.org/html/draft-ietf-core-coap-18

[81] The WebSocket Protocol. [Online]. https://tools.ietf.org/html/rfc6455

[82] MQ Telemetry Transport (MQTT) V3.1 Protocol Specification. [Online]. http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html

[83] Extensible Messaging and Presence Protocol (XMPP): Core. [Online]. https://tools.ietf.org/html/rfc6120

[84] Data Distribution Service for Real-time Systems, OMG Available Specification. [Online]. http://www.omg.org/spec/DDS/1.2/PDF/

[85] AMQP v1.0. [Online]. http://www.amqp.org/sites/amqp.org/files/amqp.pdf

[86] Gartner. (2013) Gartner says the Internet of Things installed base will grow to 26 billion units by 2020. [Online]. http://www.gartner.com/newsroom/id/2636073

[87] ABI. (2013) More than 30 billion devices will wirelessly connect to the Internet of Everything in 2020. [Online]. https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne

[88] F. Thiesse, C. Floerkemeier, M. Harrison, F. Michahelles, and C. Roduner, "Technology, Standards and Real-World Deployments of the EPC Network," *IEEE Int. Comput.*, vol. 13, pp. 36-43, 2009.

[89] V. Cristea, C. Dobre, and F. Pop, "Context-aware environments for the Internet of Things," in *in Internet of Things and Inter-Cooperative Computational Technologies for Collective Intelligence*, New York, USA, 2013, pp. 25-49.

[90] S. Loke, *Context-aware pervasive systems: architectures for a new breed of applications*. Boca Raton, USA: CRC Press, 2006.

[91] IEEE 1451. http://standards.ieee.org/develop/regauth/tut/.

[92] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC Sensor Web Enablement: Overview and High Level Architecture," *GeoSens. Netw.*, vol. 4540, pp. 175-190, 2008.

[93] World Wide Web Consortium (W3C). [Online]. https://www.w3.org/

[94] OGC & W3C. Semantic Sensor Network XG final report. [Online]. http://www.w3.org/2005/incubator/ssn/xgrssn-20110628/

[95] Semantic Sensor Web. [Online]. http://wiki.knoesis.org/index.php/SSW

[96] European Telecommunications Standards Institute M2M Technical Committee (ETSI M2M TC). (2013) ETSI TS 102 690 v1.2.1. Machine-to-Machine Communications (M2M) Functional Architecture. [Online]. http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/01.02.01_60/ts_102690v010201p.pdf

[97] International Telecommunication Union. Telecommunication Standardization Sector (ITU-T). (2012) Recommendation ITU-T Y.2060: Overview of Internet of Things. [Online]. https://www.itu.int/rec/T-REC-Y.2060-201206-I/en

[98] F. Carrez et al. (2013) Final architectural reference model for the IoT v3.0, internet of things-architecture IoT-A EC project deliverable D1.5. [Online]. http://www.iot-a.eu/public/public-documents/d1.5

[99] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of Middleware for Internet of Things: A Study," *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 2, pp. 94-105, 2011.

[100] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, "Service Oriented Middleware for the Internet of Things: A Perspective," in *In European Conference on a Service-Based Internet*, vol. 6994, Heidelberg, 2011, pp. 220-229.

[101] P. Balamuralidhar, P. Misra, and A. Pal. Software Platforms for Internet of Things and M2M. [Online]. http://journal.library.iisc.ernet.in/index.php/iisc/article/view/1907

[102] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Commun.*, vol. 89-90, pp. 5-16, 2016.

[103] V. Tsiatsis et al., *Towards the Future Internet. The SENSEI Real World Internet Architecture*. Amsterdam: IOS Press, 2010.

[104] A. Preventis, K. Stravoskoufos, S. Sotiriadis, and E. Petrakis, "IoT-A and FIWARE: Bridging the Barriers between the Cloud and IoT Systems Design and Implementation," in *In Proceedings of the 6th International Conference on Cloud*

Computing and Services Science, Rome, 2016, pp. 146-153.

[105] R.T Fielding and R.N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT)*, 2002.

[106] IPSO Alliance Interop Commitee. The IPSO Application Framework. [Online]. http://www.ipso-alliance.org/wp-content/uploads/2016/01/draft-ipso-app-framework-04.pdf

[107] Z. Stojanovic and A. Dahanayake, *Service-oriented software system engineering: challenges and practices*.: Idea, 2005.

[108] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[109] J. Dean and G. Sanjay, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.

[110] FIWARE platform. FIWARE Architecture. [Online]. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_Architecture

[111] J.P. Brogan and C. Thuemmler, "'Specification for Generic Enablers as Software", (2014 IEEE), ," in *IEEE 11th International Conference on Information Technology: New Generations*, 2014, pp. doi: 10.1109/ITNG.2014.51.

[112] FIWARE Catalogue. [Online]. http://catalogue.fiware.org

[113] Open Mobile Alliance (OMA). (2012) OMA-TS-NGSI_Context_Management-V1_0-20120529-A. NGSI Context Management Specification, version 1.0. [Online]. http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/ngsi-v1-0

[114] W3C work group. (2004) XML Schema Part 2: Datatypes Second Edition, W3C Recommendation. [Online]. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

[115] B.M. Michelson. (2006) Event-Driven Architecture Overview. Event-Driven SOA is Just Part of the EDA Story. [Online]. http://dx.doi.org/10.1571/bda2-2-06cc

[116] FIWARE. NGSI Open RESTful API Specification. [Online]. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_NGSI_Open_RESTful_API_Specification

[117] FIWARE. Internet of Things (IoT) Services Enablement. [Online]. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Internet_of_Things_

(IoT)_Services_Enablement_Architecture

[118] FIWARE. Data/Context Management. [Online]. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Data/Context_Management_Architecture

[119] Apache. Hadoop. [Online]. http://hadoop.apache.org/

[120] FIWARE. Cygnus-ngsi. [Online]. https://github.com/telefonicaid/fiware-cygnus/tree/master/cygnus-ngsi

[121] Apache. Flume. [Online]. http://flume.apache.org/

[122] P. Fernández et al., "SmartPort: A Platform for Sensor Data Monitoring in a Seaport Based on FIWARE," *Sensors*, vol. 16(3), p. 417, 2016.

[123] E. Peres, M. A. Fernandes, R. Morais, C. R. Cunha, and J. A. López, "An autonomous intelligent gateway infrastructure for in-field processing in precision viticulture," *Computers and Electronics in Agriculture*, vol. 78 (2), pp. 176-187, 2011.

[124] Orion performance. [Online]. https://github.com/telefonicaid/fiware-orion/blob/develop/doc/manuals/admin/perf_tuning.md

[125] Cygnus performance. [Online]. http://fiware-cygnus.readthedocs.io/en/latest/cygnus-ngsi/installation_and_administration_guide/performance_tips/

[126] API java.net. [Online]. https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html

[127] G. Banga and P. Druschel, "Measuring the Capacity of a Web Server," *USENIX Symposium on Internet Technologies and Systems*, 1997.

[128] API java.nio. [Online]. https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html

[129] Software RStudio. [Online]. https://www.rstudio.com

[130] Kurento Generic Enabler. [Online]. http://catalogue.fiware.org/enablers/stream-oriented-kurento

[131] CEP Generic Enabler. [Online]. http://catalogue.fiware.org/enablers/complex-event-processing-cep-proactive-technology-online

[132] Stream Processing Cosmos. [Online]. http://fiware-cosmos.readthedocs.io/en/latest/installation_and_administration_manual/streaming/stream_processing_sinfonier_architecture/index.html

[133] MongoDB official site. [Online]. https://www.mongodb.com/

[134] A. J. Demers et al., "Cayuga: A General Purpose Event Monitoring System," *Third Biennial Conference on Innovative Data Systems Research*, pp. 412–422, 2007.