CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA, BAJA CALIFORNIA

PROGRAMA DE POSGRADO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

Plataforma semántica para internet de las cosas

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:

Saúl Delgadillo Rodríguez

Ensenada, Baja California, México,
2015

Tesis defendida por

Saúl Delgadillo Rodríguez

y aprobada por el siguiente Comité

Dr. José Antonio García Macías
Director del Comité

Dr. Edgardo Avilés López
Miembro del Comité

Dra. Mónica Elizabeth Tentori Espinosa
Miembro del Comité

Dr. Salvador Villarreal Reyes
Miembro del Comité

Dra. Ana Isabel Martínez García
Coordinador del Posgrado en
Ciencias de la Computación

Dr. Jesús Favela Vara
Director de Estudios de Posgrado

Febrero, 2015

Resumen de la tesis que presenta Saúl Delgadillo Rodríguez como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Plataforma semántica para internet de las cosas

Resumen elaborado por:

Saúl Delgadillo Rodríguez

El incremento en el número de pequeños dispositivos con capacidades de comunicación, cómputo y almacenamiento: ha llevado a la creación del paradigma de Internet de las Cosas. En éste se integran objetos comunes a las redes de información y con ello se abren las puertas a una amplia gama de nuevas aplicaciones. Sin embargo, las diferentes capacidades y características de estos objetos han limitado el uso y proliferación del paradigma. Con ello surge la necesidad de crear plataformas para simplificar la creación de aplicaciones sobre objetos heterogéneos.

En este trabajo se desarrolló una plataforma que permite la interconexión de todos los objetos de Internet de las Cosas independientemente de sus diferencias. Para ello se propone una arquitectura de software fundamentada en el uso de tecnologías semánticas. Ésta plataforma permite la comunicación e interacción entre dispositivos sin necesidad de intervención humana. Además se realizó un análisis de la literatura para identificar puntualmente los requerimientos del paradigma, el cual sirve para identificar las diferencias con otras plataformas propuestas en el estado del arte. Por último se creó un marco de trabajo y se realizó un análisis de factibilidad técnica mediante la implementación de aplicaciones que siguen la arquitectura de software propuesta.

Palabras clave: **Internet de las Cosas, Web de las Cosas, Internet de Todo, M2M, Middleware, Web Semántica, Datos Enlazados, Arquitectura Orientada a Servicios**

Abstract of the thesis presented by Saúl Delgadillo Rodríguez as a partial requirement to obtain the Master of Science degree in Computer Science.

Semantic platform for internet of things

Abstract by:

Saúl Delgadillo Rodríguez

The increasing number of small devices with communication, processing, and storage capabilities has introduced the paradigm of Internet of Things. This paradigm integrates common objects with information networks, and that opens the door to a wide range of novel applications. However the different capabilities and features of these objects have limited the proliferation of the paradigm. That establish the need of creating platforms to simplify the creation of heterogeneous objects applications.

On this thesis we developed a platform that allows the connection of all the Internet of Things objects regardless their heterogeneity. For this purpose we proposed a software architecture based on the use of semantic technologies. This architecture allows the communication and interaction between devices without human intervention. Further analysis of the literature was conducted to promptly identify the requirements of the paradigm, which serves to identify differences with other platforms proposed in the state of the art. Finally we developed a framework and performed a technical feasibility analysis by implementing applications that follow the proposed software architecture.

Keywords: **Internet of Things, Web of Things, Internet of Everything, M2M, Middleware, Semantic Web, Linked Data, SOA**

Dedicatoria

A mis padres y hermanos, por su apoyo incondicional.

Agradecimientos

A mi asesor de tesis, Dr. José Antonio García Macías, por su apoyo, paciencia, y conocimiento para ayudarme a realizar este trabajo.

A la Dra. Patricia Serrano Alvarado, por su asesoramiento en privacidad de la web semántica.

A Franceli L. Cibrian Robles y Netzahualcóyotl Hernández Cruz, por su contribución en el trabajo exploratorio de CoAP en UbiSOA.

A Miguel Ylizaliturri Salcedo y Raúl L. Grijalva Durón, por su contribución en el trabajo exploratorio de MQTT en UbiSOA.

A Deysi H. Ortega Román y Darién A. Miranda Bojórquez, por su trabajo del RPL Border Router.

A Marco Calderón Pérez, por sus contribuciones en el desarrollo de la aplicación de privacidad.

A todos los profesores del departamento de ciencias de la computación de CICESE que contribuyeron en mi formación académica.

A mi familia y amigos de Zacatecas, por alentarme desde lejos.

A todos mis compañeros de CICESE, por las fiestas y comidas compartidas.

A Rosa Alejandra Ortega del Castillo y Sergio Esteban Ontiveros Gallardo, por aguantarme todo este tiempo.

Al Centro de Investigación Científica y de Educación Superior de Ensenada, por brindarme los recursos para el desarrollo de esta tesis.

Al Consejo Nacional de Ciencia y Tecnología, por darme el apoyo económico para realizar la maestría.

Tabla de contenido

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Lista de figuras

Lista de figuras (Continuación)

Lista de Tablas

Lista de acrónimos

IOT: Internet of Things

M2M: Machine to Machine

RDF: Resource Description Framework

RFC: Request for Comments

CoAP: Constrained Application Protocol

6LowPAN: IPv6 over Low Power Wireless Personal Area Networks

REST: Representational State Transfer

URI: Uniform Resource Identifier

UID: Unique Identifier

URL: Uniform Resource Locator

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

HTTP: HyperText Transfer Protocol

BLE: Bluetooth Low Energy

MQTT: Message Queue Telemetry Transport

SPARQL: SPARQL Protocol and RDF Query Language

QR: Quick Response

RFID: Radio Frequency Identification

SSN: Semantic Sensor Network

WSN: Wireless Sensor Network

OUUID: Ontology Universally Unique Identifier

IP: Internet Protocol

W3C: World Wide Web Consortium

IETF: Internet Engineering Task Force

RAM: Random Access Memory

NFC: Near Field Communication

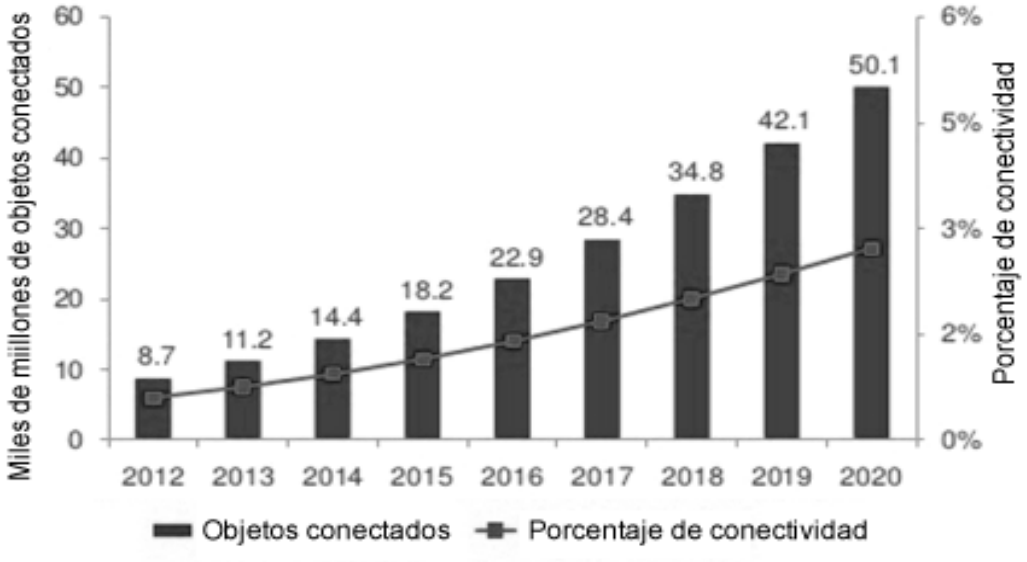
USD: United States Dollar

Capítulo 1 Introducción

En éste capítulo se da una introducción al tema y se describe el trabajo relacionado que motivó ésta investigación. Se describen los objetivos y preguntas de investigación que se pretenden resolver con esta tesis, así como la metodología que se siguió.

1.1. Motivación

Se estima que existirán alrededor de 25 mil millones de dispositivos capaces de conectarse a Internet en 2015 y 50 mil millones en 2020 (Evans, 2011) (ver Figura 1**Error! Not a valid bookmark self-reference.**). Con una tecnología como IPv6, la cual es capaz de identificar 2^{128} entidades únicas, un nuevo paradigma ha cobrado fuerza en los últimos años, la Internet de las Cosas (IoT, por sus siglas en inglés). Este paradigma, sugerido por muchos como la Internet del futuro (Tan and Wang, 2010), se define como una variedad de "cosas" u "objetos" (etiquetas, sensores, actuadores, teléfonos celulares, etc.) con capacidad de identificarse, comunicarse, e interactuar entre ellos para lograr metas comunes. Con ello crece la posibilidad de crear nuevas aplicaciones en campos como: transporte, logística, servicios de salud, domótica, entre otros (Atzori, *et al.,* 2010).



.

Figura 1 - Contador de conexiones de CISCO. En la gráfica se muestra una estimación del número de objetos conectados a internet en relación con el número total de objetos en el mundo (CISCO, 2014).

El incremento en el número de dispositivos con capacidad de cómputo hace más cercana la visión de cómputo ubicuo de Weiser (1991), en donde cada individuo puede interactuar con múltiples dispositivos de cómputo que se encuentran “invisibles” dentro del ambiente. Para que esto suceda los dispositivos deben comunicarse e interactuar entre ellos, proporcionando a máquinas y usuarios la capacidad de entender mejor el ambiente que los rodea. Este conocimiento de los objetos sobre sus ambientes permitirá crear aplicaciones y servicios que sean capaces de tomar decisiones inteligentes y responder a un ambiente constantemente cambiante.

Además el aumento de dispositivos traerá consigo una gran cantidad de datos que se podrían obtener a través de sensores (e.g., temperatura, luz, sonido, video, etc.). De igual manera las interacciones entre objetos también producirán grandes volúmenes de datos relevantes. Para que todos los datos obtenidos sean de mayor utilidad se debe anexar información relacionada al ambiente en el que se produjeron (e.g., localización, tiempo, situación, objetos cercanos, personas involucradas).

Figura 2 - Jerarquía del Conocimiento. Pirámide que describe el producto de los datos conforme son procesados (Barnaghi, et al., 2012).

Si se relacionan los datos de los sensores con eventos y acontecimientos a su alrededor se obtiene nueva información. Esta información al ser analizada con un propósito en particular da lugar a nuevo conocimiento. Conocer lo que ocurre en el mundo físico permite ofrecer mejores productos y servicios. Pero finalmente, analizar los patrones de comportamiento basados en el conocimiento obtenido de los datos permite

entender mejor el mundo en el que vivimos, es decir, da lugar a la inteligencia o lo que algunos llaman sabiduría (Barnaghi, *et al.,* 2012) (ver Figura 2).

La necesidad de relacionar toda esa información lleva consigo la tarea de conectar todos los dispositivos que generan estos datos y una forma de hacerlo es siguiendo el ejemplo de la Internet.

1.2. Planteamiento del problema

Un paradigma (en el ámbito científico) es un conjunto de prácticas que definen una disciplina científica durante un periodo específico. Por lo tanto, dado que IoT es un paradigma relativamente nuevo en el área de computación, sus prácticas comunes aún se están definiendo.

En vista de que el objetivo central de IoT es interconectar objetos heterogéneos en una red global, es necesario definir una arquitectura de software que establezca el comportamiento de los objetos que se conectan a la red. Una de las tendencias en el estado del arte es el uso de arquitecturas orientadas a servicios (SOA, por sus siglas en ingles), las cuales han demostrado su capacidad para relacionar diversos servicios dentro de Internet por medio de enlaces simples entre ellos. Este tipo de arquitectura de software ha sido utilizada por diversos autores para crear plataformas de IoT (Avilés-López y García-Macías, 2009) (Guinard, *et al.,* 2010) (Tan and Wang, 2010).

Una plataforma (en informática) es un sistema que permite trabajar con determinados módulos de hardware y software compatibles. Dicho sistema se define por un estándar en el cual se determina el uso de una arquitectura de hardware y una arquitectura de software. Para definir una plataforma es necesario establecer: el hardware a utilizar, el sistema operativo, el lenguaje de programación, entornos de aplicación e interfaces de usuario.

Las plataformas de IoT con arquitecturas SOA se fundamentan en él éxito de dichas arquitecturas en Internet, sin embargo, IoT presenta diversos retos que lo diferencian de la Internet actual (Estrada-Martínez, 2011). Una de las diferencias más

notables es que los objetos de IoT es que poseen una gran variedad de capacidades de procesamiento, comunicación, almacenamiento y recursos. Describir esa heterogeneidad es clave para determinar el uso y comportamiento de los dispositivos en la red. Por esta razón describir toda la información sobre los objetos ha llevado la atención sobre el uso de tecnologías del paradigma de la Web Semántica.

La Web Semántica, descrita originalmente por el creador de la World Wide Web (Berners-Lee, *et al.,* 2001), se sustenta en relacionar la información de tal forma que sea entendible en un esquema Máquina a Máquina (M2M). El enlace entre estos paradigmas da lugar a una rama de IoT que usa tecnologías semánticas para describir objetos (Atzori, *et al.,* 2010) (Barnaghi, *et al.,* 2012).

Con base en tecnologías semánticas se han propuesto plataformas como: Sense2Web (Barnaghi, *et al.,* 2010), SPITFIRE (Pfisterer, *et al.,* 2011) y Smart Grid Middleware (de Diego, *et al.,* 2014). Estas plataformas permiten describir objetos heterogéneos mediante semántica, no obstante, la interacción entre objetos es aún muy limitada, por ejemplo: Sense2Web y Smar Grid Middleware permiten leer datos de los objetos pero no posibilitan escribir nueva información; SPITFIRE utiliza un sistema de reglas semánticas para que los dispositivos interactúen pero establece métodos para crear interacciones más complejas entre dispositivos.

En los últimos años se ha desarrollado la plataforma UbiSOA para IoT en el Laboratorio de Cómputo Móvil y Ubicuo de CICESE. Ésta plataforma permite la interacción de dispositivos heterogéneos mediante un enfoque distribuido y arquitectura SOA; sin embargo, no cuenta con la semántica para relacionar datos y objetos de la plataforma (Avilés-López y García-Macías, 2012). Además se desarrollaron los Visores Sensoriales: sistemas que permiten la interacción hombre-máquina de dispositivos heterogéneos descritos semánticamente (Estrada-martinez, 2012).

Por lo tanto, se tiene la necesidad de crear una plataforma semántica que permita describir objetos y la información del ambiente que los rodea.

1.3. Propuesta

Para este trabajo de tesis se propone crear una plataforma que enriquezca la interacción de dispositivos heterogéneos. Para ello se pretende extender la plataforma UbiSOA (Avilés-López y García-Macías, 2012) mediante el uso de tecnologías de la web semántica. Dicha plataforma debe permitir la descripción de los objetos de IoT a través de un esquema M2M, así como establecer mecanismos que permitan mejorar la interacción entre objetos en la red.

1.4. Preguntas de investigación

Con base en lo planteado anteriormente, esta tesis se va a guiar con las siguientes preguntas de investigación.

- ¿Cuáles son los requerimientos que se han establecido en la literatura para crear una plataforma de IoT?
- ¿Qué tecnologías de la Web Semántica son de utilidad para mejorar la interoperabilidad entre objetos de IoT?
- ¿Qué mecanismos se pueden usar para mejorar la interacción entre objetos de una plataforma de IoT?
- ¿Cuáles son los retos y limitaciones para crear aplicaciones de Internet de las Cosas usando tecnologías de la Web Semántica?

1.5. Objetivo general

El enfoque principal de este proyecto será en torno al siguiente objetivo general:

Crear una plataforma que extienda la plataforma de UbiSOA mediante el uso de tecnologías semánticas, con el fin de mejorar las capacidades de interoperabilidad e interacción entre objetos de IoT.

1.6. Objetivos específicos

A través de este objetivo general se planean lograr los siguientes objetivos específicos:

- Identificar los requerimientos y limitantes para integrar objetos de IoT con tecnologías de la web semántica.
- Seleccionar las tecnologías más adecuadas de la web semántica para una plataforma de IoT.
- Diseñar una arquitectura de software que extienda la arquitectura de UbiSOA con tecnologías de la web semántica.
- Extender el marco de trabajo de UbiSOA para implementar aplicaciones de ejemplo.
- Analizar alcances y limitantes de la arquitectura en función de los requerimientos previamente encontrados.

1.7. Contribución

La importancia de la investigación radica en la creación de una plataforma que permita la interacción de dispositivos de IoT mediante el uso de tecnologías semánticas. Lo cual facilitará la creación de aplicaciones de IoT, debido a que la abstracción de software permite transparentar el uso de diversas tecnologías de comunicación y protocolos. Como consecuencia, los desarrolladores de software no requieren conocer lo que ocurre en capas inferiores de la plataforma para usar dispositivos con diversas características de hardware y software.

La contribución al conocimiento se puede dividir en 4 partes: la primera, una categorización de requerimientos de la literatura para la creación de una plataforma de IoT; la segunda, una arquitectura de software que permita mejorar la interacción de objetos heterogéneos de IoT mediante tecnologías semánticas; la tercera, la implementación de un marco de trabajo que permita desarrollar aplicaciones con base en los principios de la arquitectura propuesta; la cuarta, un análisis de diferencias entre la arquitectura de software propuesta y las de la literatura.

1.8. Metodología

Para cubrir los objetivos de la tesis, el desarrollo se realizó en las siguientes etapas:

Etapa 1: Revisión de literatura y caracterización del problema. Esta etapa consiste en una revisión de la literatura sobre trabajos previos de IoT y la Web semántica. En ésta se analizan principalmente los siguientes aspectos: características de los diversos objetos de IoT; requerimientos para la creación de plataformas de IoT; plataformas semánticas previamente propuestas; y tecnologías de la web semántica.

Una vez obtenida la información se categorizan los objetos de acuerdo a su conectividad y se agrupan los requerimientos para obtener una lista manipulable. Esta información es la base para el diseño de la arquitectura.

Etapa 2: Diseño de la arquitectura de software. En esta etapa se diseña la arquitectura de software propuesta con base en los requerimientos identificados en la literatura. Se diseña un modelo para dar soporte a las diversas categorías de objetos encontrados en la etapa anterior. Se seleccionan las tecnologías más adecuadas de la web semántica para la arquitectura.

Etapa 3: Implementación del marco de trabajo. En esta etapa se realiza un análisis de factibilidad técnica con la finalidad de verificar que es posible usar las tecnologías descritas en la arquitectura. Por lo cual se extiende el marco de trabajo UbiSOA con base en la arquitectura propuesta y se implementan diversos componentes de software. Dichos componentes se interconectan mediante la creación de dos aplicaciones que dan soporte a dos escenarios distintos.

Etapa 4. Análisis de la plataforma. En esta etapa se realiza un análisis de la plataforma mediante la lista de requerimientos previamente encontrada. De igual manera se analizan las limitantes de la plataforma a partir de las observaciones y problemas encontrados durante la implementación de las aplicaciones. Por último se realiza un análisis de diferencias entre la plataforma propuesta y las plataformas descritas en la literatura.

Etapa 5: Redacción de la tesis. Finalmente se concluye el trabajo con la redacción del documento de tesis.

1.9. Organización de la tesis

El resto del documento se organiza de la siguiente manera:

Capítulo 2. Internet de las Cosas: En este capítulo se describe a detalle el estado actual del paradigma de IoT. Se describen los dispositivos que forman parte del mismo y las principales características que poseen. Además se describen las tecnologías clave del paradigma y el estado actual de su desarrollo.

Capítulo 3. La Web Semántica: En este capítulo se describe la Web Semántica y sus aplicaciones. Se describen las tecnologías que son utilizadas actualmente en la Web Semántica. Además se analizan las características que resultan relevantes para una plataforma de IoT.

Capítulo 4. Diseño de la plataforma OpenThings: En este capítulo se describen los requerimientos que llevaron al desarrollo de la arquitectura propuesta y la agrupación de los mismos. Se describe a detalle la arquitectura propuesta y se muestran ejemplos de uso de la misma.

Capítulo 5. Implementación y análisis: En este capítulo se describen las modificaciones realizadas al marco de trabajo UbiSOA y las aplicaciones implementadas. Se revisan los requerimientos en contraste con la arquitectura propuesta. Por último se describen las diferencias con otras propuestas de la literatura.

Capítulo 6. Conclusiones: Finalmente se discuten las contribuciones y principales logros de este trabajo, así como el trabajo futuro derivado de esta investigación.

Capítulo 2 Internet de las Cosas

2.1. Introducción

El paradigma de la Internet de las Cosas (IoT, por sus siglas en inglés) se refiere a la idea general de “Cosas”, especialmente objetos de la vida cotidiana, que son: legibles, reconocibles, localizables y/o controlables vía Internet (National Intelligence Council, 2008).

Las etiquetas de identificación por radio frecuencia (RFID, por sus siglas en inglés) son los objetos más simples en ser considerados “Cosas” dentro del paradigma. Éstas son atribuidas a los laboratorios de investigación académica llamados Auto-ID[1]. Estos laboratorios trabajan en el campo de redes RFID y tecnologías de sensores en conjunto con EPCglobal (Traub, *et al.,* 2005), con el propósito de crear una arquitectura de IoT centrada en el código electrónico de producto (EPC, por sus siglas en ingles). Éste código pretende ser un número que identifique inequívocamente cada uno de los productos físicos en el planeta.

Las limitadas capacidades de cómputo y comunicación de las etiquetas dieron lugar a un enfoque más amplio en donde surgen otro tipo de objetos con mayores capacidades. IPSO (IP para objetos inteligentes), un foro de 25 compañías, promueve el uso de protocolos de la Internet para conectar los objetos. Estos objetos, llamados objetos inteligentes (Smart objects, en inglés), se fundamentan en la pila de protocolos IP con tecnologías basadas en el protocolo IEEE 802.15.4. Estas tecnologías son lo suficientemente ligeras para usarse en dispositivos diminutos de cómputo alimentados por baterías (Dunkels, 2008).

Los objetos inteligentes, usados comúnmente para construir redes inalámbricas de sensores (WSN, por sus siglas en ingles), poseen capacidades de procesamiento,

[1] http://www.autoidlabs.org

almacenamiento y comunicación (Yick, 2008). Por esta razón son considerados los dispositivos centrales del IoT (Mathew, *et al.,* 2011).

2.2. Diversas visiones

De acuerdo con Atzori *et al.,* (2010) la definición de "Internet de las Cosas" se presta para distintas distintas interpretaciones por ciertos grupos científicos, alianzas comerciales y organizaciones. organizaciones. Esto da lugar a tres visiones principales del paradigma de IoT: orientada a cosas, a cosas, orientada a Internet y orientada a semántica. La intersección de las tres visiones provee provee una visión más completa del paradigma (ver

Figura 3).

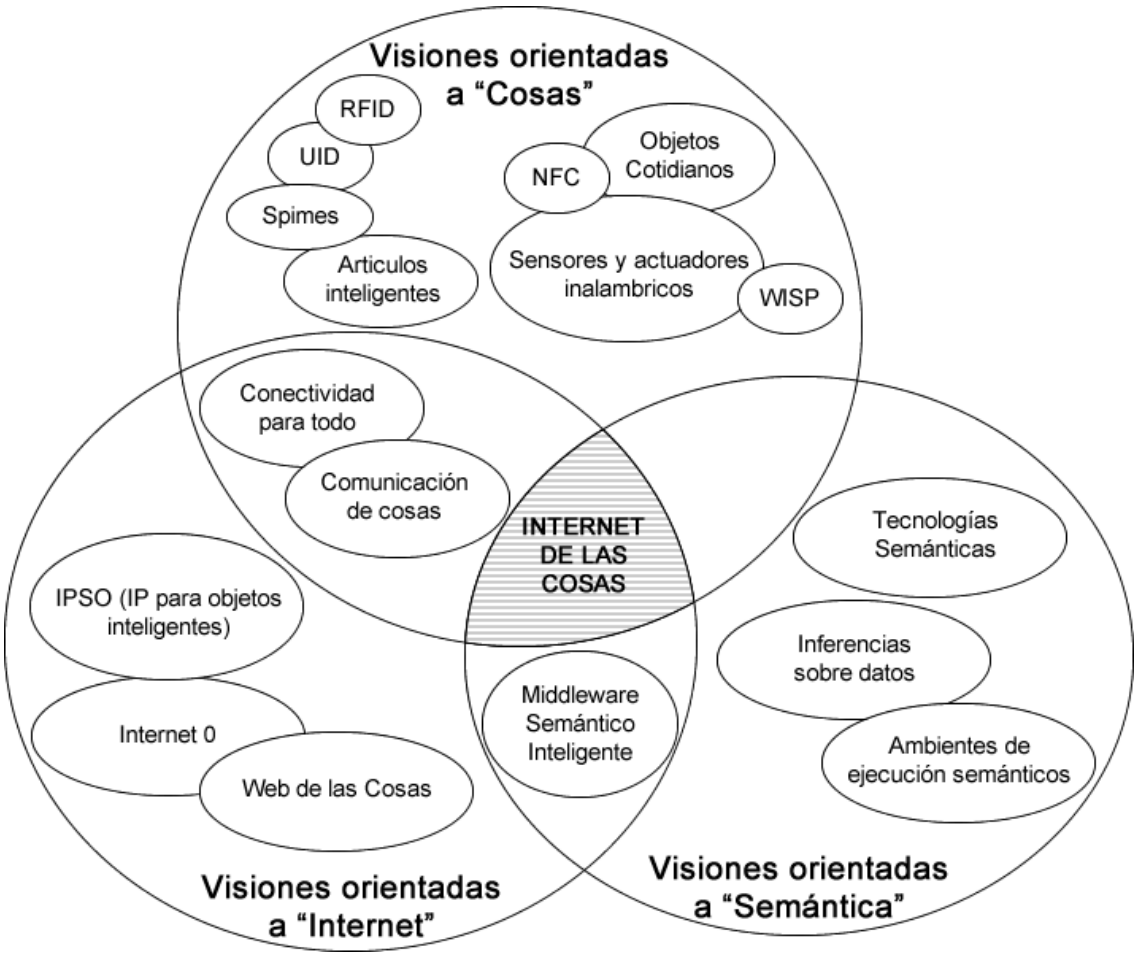


Figura 3 - Las 3 visiones del paradigma de Internet de las Cosas (Atzori, *et al.,* 2010).

2.2.1. Visión orientada a “Cosas”

Se enfoca en la creación de los objetos inteligentes, sus características y las formas de interacción que tienen con el mundo real. En esta visión se encuentra el trabajo de AutoID Labs y EPCglobal, en donde se usan etiquetas RFID para identificar cualquier objeto. Otros trabajos describen sistemas para complementar los objetos usando un historial digital persistente durante toda su vida útil: “Tales of Things" de Barthel *et al.,* (2010), por ejemplo.

2.2.2. Visión orientada a “Internet”

Se enfoca en la creación de comunicaciones, protocolos, arquitecturas de software y tecnologías de red: los cuales permiten a estos objetos interactuar y cooperar. Esta visión promueve el uso de protocolos IP y HTTP, así como homólogos para redes de recursos limitados como lo son CoAP y 6LowPAN. Diversos objetos comerciales de este tipo han salido a la venta en los últimos años (ver Figura 4), por ejemplo: Nest[2], un termostato inteligente con diversas funciones; Lifx[3], una lámpara que permite cambiar su color e iluminación mediante un teléfono inteligente; Smart Things[4], un kit de sensores y actuadores para el hogar.

Figura 4 - Objetos inteligentes comerciales que poseen conectividad IP. De derecha a izquierda: a) Smart Things, b) Termostato Nest, c) Lifx.

La intersección de ésta visión con la de objetos ha llevado a la creación de middleware que tiene el objetivo de conectar todos los objetos por medio de una red. Trabajos como los presentados por Castellani *et al.,* (2011) y Avilés-López (2012) se

[2] https://nest.com/
[3] http://lifx.co/
[4] http://www.smartthings.com/

encuentran en esta área. Estos trabajos siguen el modelo de servicios web con la intención de conectar todos los objetos del paradigma.

2.2.3. Visión orientada a "Semántica"

Se enfoca en el uso de tecnologías semánticas para lograr la interconexión, organización y representación de la información. Relacionar semánticamente los datos producidos por dispositivos de IoT permite escalar a peldaños más altos de la jerarquía del conocimiento. No obstante, relacionar datos no es exclusivo del paradigma de IoT, el uso de la semántica se ha popularizado por la creciente cantidad de datos en la Internet. El paradigma de la Web Semántica, o lo que algunos llaman la Web 3.0 (Berners-Lee, *et al.,* 2001), se enfoca en el uso de la semántica dentro de Internet.

Una de las grandes cualidades de las tecnologías semánticas es la descripción de diversos conceptos usando esquemas manipulables en ambientes Máquina-Máquina (M2M). Ésta capacidad de describir conceptos ha sido utilizada para crear representaciones virtuales de los objetos de IoT. Un ejemplo de ello son los Visores Sensoriales (Estrada-Martínez, 2011), los cuales permiten navegar entre los diversos objetos de IoT usando sus descripciones semánticas (ver Figura 5).

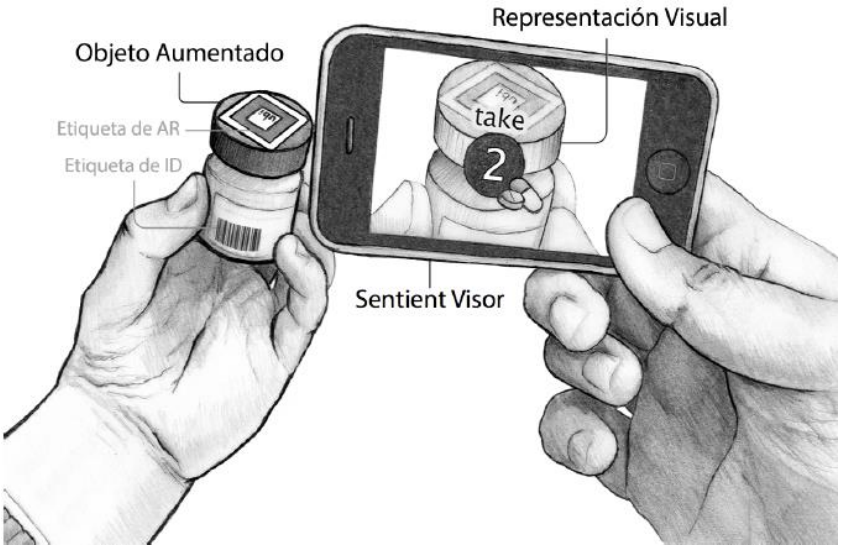


Figura 5 - Ejemplo de un Visor Sensorial. El teléfono inteligente reconoce al objeto y muestra información adicional relevante para el usuario (Estrada-Martínez, 2011).

2.3. Diversidad de objetos

Dado de que cualquier objeto puede formar parte del paradigma de IoT, ¿Cuáles son las características que requieren esos objetos para ser integrados en el paradigma? Mathew *et al.,* (2011) analizan todo tipo de objetos que formarían parte de IoT, con el objetivo de clasificarlos. Describen que las características principales que se deben tomar en cuenta para esta clasificación son: identidad única, procesamiento, comunicación y almacenamiento. Puesto que la identificación única es una capacidad que deben compartir todos los objetos del paradigma, esta característica se supone cierta, mientras que las otras tres capacidades se proyectan en un espacio tridimensional (ver Figura 6).

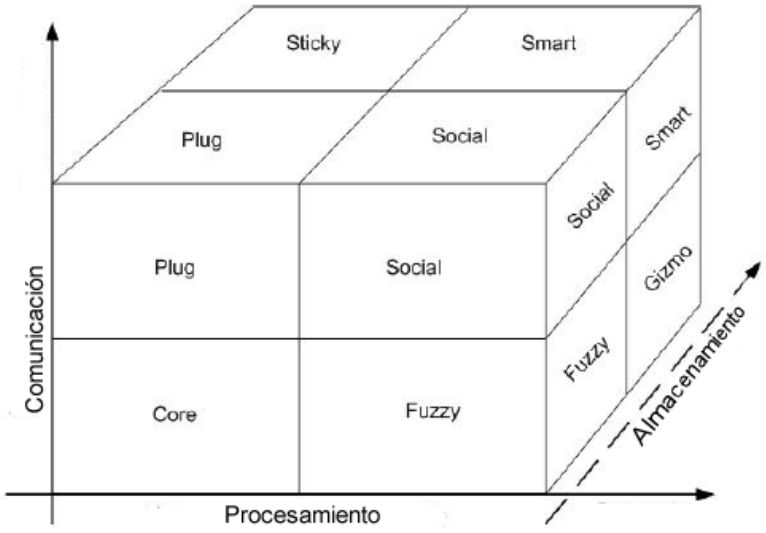


Figura 6 - Clasificación tridimensional de objetos según Mathews *et al.,* (2011).

Dentro de este espacio se pueden tener ocho clases distintas de objetos, las cuales se describen a continuación:

- Core: Son los objetos que tienen solo la capacidad de identificarse dentro de un contexto dado, con ayuda de tecnologías de identificación como RFID o códigos de barras (e.g., botellas de medicamentos, zapatos, sillas).
- Fuzzy: Son objetos que poseen poder de procesamiento con operaciones pre-definidas, pero que no pueden conectarse a otros objetos o almacenar información (e.g., horno de microondas, lavadora).

- Plug: Son objetos que poseen interfaces de comunicación, pero no tienen poder de procesamiento o almacenamiento (e.g., bocinas, micrófonos).
- Fat: Son objetos que poseen almacenamiento pero carecen de las otras capacidades, (e.g., CD, DVD).
- Social: Son objetos que poseen procesamiento y comunicación pero no tienen capacidades de almacenamiento (e.g., control remoto, teléfono de línea).
- Sticky: Son objetos que poseen comunicación y almacenamiento pero no capacidad de procesamiento (e.g., etiqueta RFID, memoria USB).
- Gizmo: Son objetos con capacidad de almacenamiento y procesamiento pero sin comunicación (e.g., calculadora, gameboy).
- Smart: Son objetos que poseen las tres características (e.g., telefono inteligente, mote).

2.3.1. Etiquetas y objetos

De acuerdo a la visión de Auto-ID labs, todos los objetos se pueden conectar con el uso de etiquetas como: códigos de barras, etiquetas RFID, etiquetas NFC[5], etc. Estas etiquetas han demostrado ser una buena opción para anexar información a objetos de bajo costo o de corto ciclo de vida. Sin embargo, la cantidad de información que se puede anexar con estas tecnologías es relativamente pequeña: los códigos de barras se limitan a unas docenas de caracteres; los códigos de respuesta rápida (QR, en inglés) permiten de 1817 a 7089 caracteres (Denso Wave Inc., 2014); las etiquetas RFID pasivas almacenan de 200 a 8,000 bits (Want, 2006).

El espacio de almacenamiento de estas tecnologías es suficiente para guardar información básica del objeto, pero la información contextual del objeto puede ser demasiado extensa para ubicarse dentro del mismo. Además la información puede incrementarse con el tiempo (e.g., anexar cada cambio de propietario). Las etiquetas RFID permiten lectura y escritura, en cambio los códigos de barra no pueden reescribirse

[5] NFC, siglas en ingles de comunicación de campo cercano.

una vez impresos lo que implica una limitación si se quiere anexar nueva información al objeto.

Los códigos 2D dinámicos contienen una dirección uniforme de recurso (URL, por sus siglas en ingles) (Alapetite, 2010). Estas URL re-direccionan a los lectores QR a un sitio web donde se almacenan datos complementarios. Esto permite reescribir la información virtual enlazada con el código impreso. Diversos servicios comerciales como TrackQR[6] y Snap Tag[7] ofrecen servicios para crear y administrar este tipo de códigos. Al crear un código QR dinámico se puede anexar información sobre ubicación del código, lo que permite a la aplicación conocer la ubicación del dispositivo que está haciendo la lectura. Esta función ha sido usada en aplicaciones de realidad aumentada que utilizan códigos QR para ubicar dispositivos en un sitio en particular (Nikolaos and Kiyoshi, 2010).

En el trabajo “Tales of Things” (de Jode, *et al.,* 2011) se usa una URL en el código QR o en la etiqueta RFID para enlazar el objeto con una página web. Esta página contiene toda la información histórica del objeto y su contexto, lo que facilita la lectura y escritura de información del objeto. Dicha técnica permite reducir las limitantes de almacenamiento.

2.3.2. Objetos inteligentes

Son denominados objetos inteligentes aquellos que poseen comunicación, procesamiento y almacenamiento. Sin embargo, estas capacidades no garantizan la capacidad de comunicarse usando protocolos comunes de IP. El RFC 7228 (Bormann, *et al.,* 2014) de la Internet Engineering Task Force (IETF, por sus siglas en inglés) describe que los objetos con recursos limitados se pueden dividir dentro de tres clases:

Clase 0. Son dispositivos muy limitados que se conectan a Internet con ayuda de dispositivos con capacidades mayores. Estos dispositivos de ayuda actúan como: proxies, puertas de enlace o servidores. Además los dispositivos clase 0 no pueden asegurar la seguridad o manipularse convencionalmente y muy probablemente estarán pre-configurados con una pequeña cantidad de datos. Sin embargo, estos dispositivos

[6] https://trakqr.com/
[7] http://snaptag.co.nz/qr_codes/

pueden responder señales de actividad, recibir mensajes de encendido/apagado o poseer indicadores de estado.

Ejemplos de estos dispositivos son: Tile[8] o Estimote[9]. Estos hacen uso de tecnologías como “Bluetooth Low Energy” (BLE) para comunicar pequeños mensajes pre-configurados (ver Figura 7).

Figura 7 – Dispositivos clase 0. De izquierda a derecha: a)Tile, b) Estimote. Ambos se comunican usando BLE.

Clase 1. Son dispositivos que no pueden comunicarse fácilmente con el uso de protocolos como HTTP, TLS, etc., así como hacer uso de representaciones basadas en XML. No obstante, tienen suficientes capacidades para usar protocolos y lenguajes diseñados para ambientes limitados (e.g., CoAP sobre UDP). Pueden interactuar con otros dispositivos sin necesidad de otro nodo intermediario, así como integrarse como nodos dentro de una red IP. Aun así deben cuidar el uso de memoria, espacio de código y energía.

Ejemplos de estos dispositivos (ver Figura 8) son: plataformas de sensores como: Shimmer, una plataforma con múltiples sensores[10]; Arduino[11] o Pinoccio[12], plataformas multipropósito. Estos dispositivos usan soluciones de software diseñadas exclusivamente para esos dispositivos, pero también pueden usar sistemas operativos para dispositivos con recursos limitados como TinyOS[13] o Contiki[14].

[8] https://www.thetileapp.com/
[9] http://estimote.com/
[10] http://www.shimmersensing.com/
[11] http://www.arduino.cc/
[12] https://pinocc.io/
[13] http://tinyos.net/
[14] http://www.contiki-os.org/

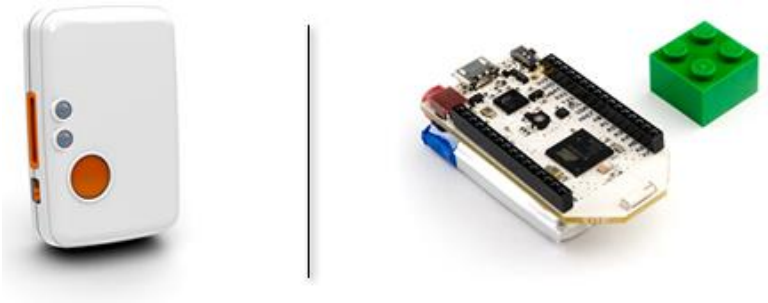


Figura 8 – Dispositivos clase 1. De izquierda a derecha: a) Plataforma Shimmer, b) Un micro-controlador Pinoccio.

Clase 2. Son dispositivos que pueden dar soporte a la mayoría de las pilas de protocolos usadas en computadoras o servidores. Aun así estos dispositivos pueden beneficiarse de protocolos ligeros que sean eficientes en el consumo de energía y ancho de banda. Además el uso de menos recursos para administración de la red permite ceder más recursos para aplicaciones.

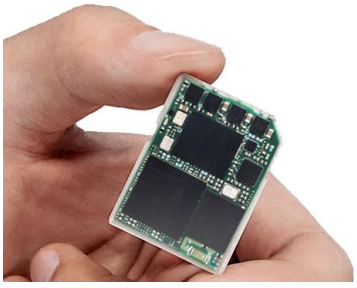


Figura 9 – Dispositivo clase 2. Intel Edison, como se presentó en el "Consumer Electronic Show" (CES) en 2014.

Un ejemplo de estos dispositivos es la computadora Edison (ver Figura 9), presentada por Intel en enero de 2014, es un pequeño dispositivo con procesador x86 de doble núcleo, 1GB de RAM y comunicación Wifi/Bluetooth4.0. Con un costo inicial de $50 USD y un tamaño de aproximado de 35mm, es el dispositivo que más se acerca a la visión de un objeto inteligente de IoT (Hachman, 2014). Este dispositivo ha incrementado el interés sobre el paradigma, como se puede observar en la Figura 10.

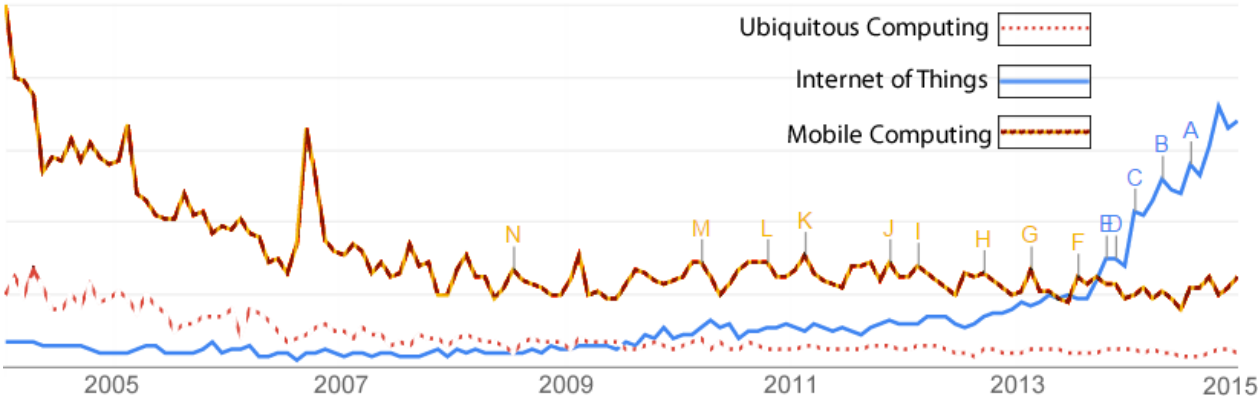


Figura 10 – Tendencia de búsquedas en Google del término Internet de las Cosas en comparación con Cómputo Móvil y Cómputo Ubicuo. Los Indicadores E y D señalan noticias sobre la presentación del Intel Edison (Google, 2014).

2.4. Middleware para Internet de las Cosas

Un middleware es un software que funciona como una capa de abstracción de software distribuida, la cual se sitúa entre las capas de aplicaciones y las capas inferiores (ver Figura 11). Se ha propuesto que se debe hacer uso de un middleware para lograr la comunicación e interacción de todos los objetos de IoT (Perera, *et al.,* 2014).

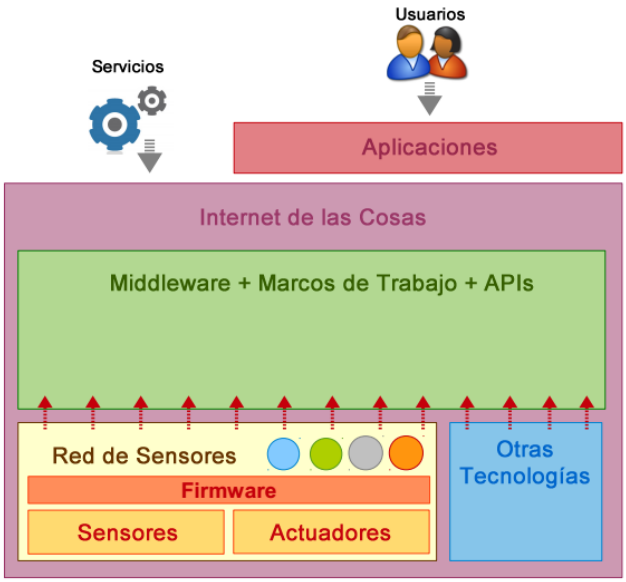


Figura 11 - Diagrama de middleware de Perera *et al.,* (2014). En él se muestra la importancia del Middleware para unir los dispositivos de capas inferiores con las aplicaciones en el área de IoT.

2.4.1. Web de las Cosas

Con el nuevo protocolo de Internet (IPv6) y la posibilidad de conectar todos los objetos de la tierra, la visión orientada a Internet resulta una de las más prometedoras para desarrollar el paradigma de IoT. Por lo tanto se han propuesto diversos middleware para Internet de las Cosas siguiendo arquitecturas orientadas a servicios (SOA). Esto hace posible la interacción con servicios ya disponibles en Internet (Chander, *et al.,* 2012).

Siguiendo los principios de las arquitecturas SOA, surgió una nueva visión del paradigma llamada la Web de las Cosas (Mathew, *et al.,* 2011). En este paradigma se utiliza el modelo de Transferencia de Estado Representacional (REST, por sus siglas en ingles) para realizar el acomplamenientо entre dispositivos. REST, que fue presentado como tesis doctoral de Fielding (2000), utiliza el protocolo HTTP que actualmente se usa en la web sobre la pila TCP/IP para realizar 4 operaciones basicas: crear, recuperar, actualizar y eliminar. Cada una de estas operaciones tiene su equivalente de HTTP en los métodos POST, GET, PUT y DELETE (ver Tabla 1).

Tabla 1 - Operaciones CRUD y sus equivalentes en SQL y HTTP

CRUD	SQL	HTTP	Descripción
Create	INSERT	POST	Crea un nuevo recurso
Retrieve	SELECT	GET	Solicita un recurso
Update	UPDATE	PUT	Actualiza un recurso
Delete	DELETE	DELETE	Elimina un recurso

Para que un sistema pueda ser considerado REST se deben seguir los siguientes principios:

1. Todos los recursos deben tener un identificador único (URI).
2. Deben ser accesibles a través de las operaciones básicas de HTTP.
3. Deben usar representaciones extensibles y estándares para compartir información (HTML, XML, etc.).
4. La comunicación debe ser sin estado (stateless, en inglés), es decir, ni el cliente ni el servidor deben mantener el estado de la aplicación.

HTTP requiere de capas inferiores que brinden el transporte de paquetes a través de la red, en Internet los protocolos TCP/IP son la base para realizar éste transporte. No obstante, en dispositivos con recursos limitados HTTP y TCP/IP son considerados demasiado complejos y demandantes de recursos (Pfisterer, *et al.,* 2011). Por ello dos tecnologías nuevas se han destacado en el área: 6LoWPAN y CoAP.

6LoWPAN es una adaptación ligera de la capa IPv6, la cual permite intercambiar paquetes IPv6 con la Internet. Sin embargo, solo se ha especificado UDP para la capa de transporte sobre éste protocolo, debido a que TCP es considerado muy demandante de recursos. En vista de ello el protocolo CoAP funciona sobre UDP e integra capacidades de acuse de recibo y retransmisión para contrarrestar la falta de TCP. Además CoAP provee una alternativa a HTTP usando una representación binaria de sus métodos básicos.

La ventaja de esta pila de protocolos para dispositivos de recursos limitados es que es equivalente a la pila HTTP sobre TCP/IP (ver Figura 12). Por lo tanto es posible mapear 6LoWPAN a IPv6, CoAP/UDP a HTTP/TCP, y viceversa, de tal forma que los mensajes pueden pasar de una pila a otra casi transparentemente.

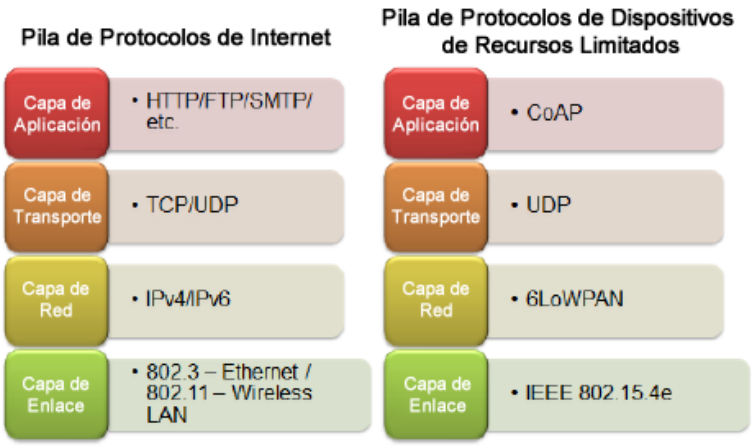


Figura 12 - Pilas de protocolos equivalentes. La pila de protocolos de la internet y su equivalente en redes con dispositivos de recursos limitados (Sutaria and Govindachari, 2013).

No todos los objetos son capaces de interactuar usando la arquitectura REST. Algunos dispositivos poseen recursos demasiado limitados como para hacer uso de CoAP o 6LoWPAN. Para superar esto Guinard *et al.,* (2010) proponen el uso de una

arquitectura basada en proxies. Por otra parte la plataforama UbiSOA (Avilés López, 2012) y la plataforma Xively [15] hacen posible que cualquier dispositivo se puede encapsular dentro de un servicio web usando interfaces. Esta capacidad de encapsulamiento permite integrar cualquier objeto de IoT con el resto de la web. Por consiguiente estas plataformas pueden dotar a cualquier objeto de una interfaz REST pese a las diversas tecnologías de software y hardware que posean.

2.4.2. Nivel de abstracción

El middleware puede proveer a los objetos de diversos niveles de abstracción. De acuerdo con Thiesse y Michahelles (2010), los objetos inteligentes pueden tener tres niveles de abstracción. Estos niveles les posibilitarían incrementar la complejidad de las aplicaciones mediante la descripción de funciones, reglas y flujos de trabajo. Esto proveería a cada objeto una mayor conciencia, lo cual incrementaría la interactividad de los objetos. El uso de esas descripciones daría lugar a tres tipos de objetos inteligentes (ver Figura 13):

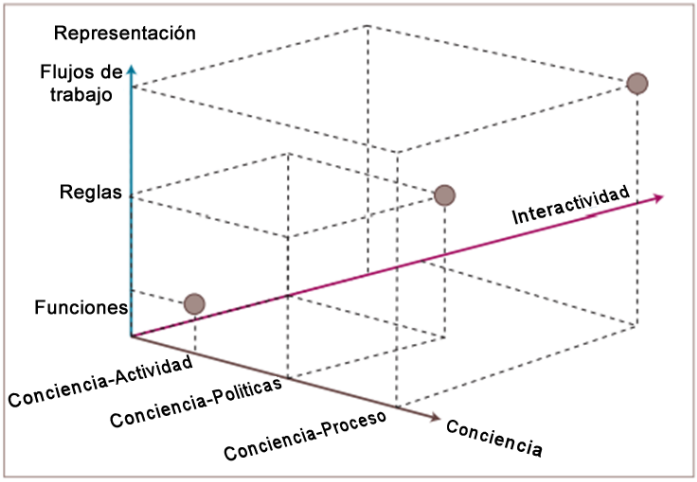


Figura 13 - Niveles de abstracción de Thiesse y Michahelles (2010). En la gráfica se pueden observar los tres tipos de objetos: a) conscientes de actividad, b) conscientes de políticas, c) conscientes del proceso.

- Conscientes de la actividad: son objetos que analizan los datos de los sensores y utilizan algoritmos de reconocimiento para detectar actividades y eventos.

[15] https://xively.com/

- Conscientes de políticas: son objetos conscientes de la actividad con un modelo de política embebido, para dar retroalimentación sobre faltas en la política de uso.
- Conscientes del proceso: son objetos ligados a modelos de procesos que les permiten identificar el estado del proceso y los pasos a seguir.

2.5. Dominios de aplicación

2.5.1. Aplicaciones

La interconexión y comunicación entre los dispositivos de IoT permitirá la creación de muchas aplicaciones en diversos dominios (Atzori, *et al.,* 2010). De acuerdo con Perera, *et al.,* (2014) las aplicaciones se pueden dividir en tres categorías: industrial, ambiental y social:

- Dominio industrial: gestión de cadenas de suministro, transporte, logística, aeroespacial, aviación y automotriz.
- Dominio ambiental: agricultura, ganadería, reciclaje, alerta de desastres y monitoreo ambiental.
- Dominio social: telecomunicaciones, tecnología médica, cuidado de la salud, edificios inteligentes, domótica, medios de comunicación y entretenimiento.

2.5.2. Partes interesadas

Para definir la estructura de crecimiento de la industria de IoT, Kim y Lee (2014) un modelo de negocios para el crecimiento del paradigma. En ese modelo se identifican seis partes interesadas distintas (ver Los desarrolladores de software y dispositivos de IoT son quienes se beneficiarían directamente de la venta de productos a los consumidores, no obstante, no son los únicos que obtendrían ganancias. Los operadores de red, proveedores de servicios y operadores de plataformas: actuarían como intermediarios en la industria, ofreciendo diversos servicios a cambio de una tarifa. De esta manera los usuarios tendrán la opción de comprar dispositivos y/o pagar por servicios, tal como se hace con el cómputo en la nube.

Tabla 2).

Los desarrolladores de software y dispositivos de IoT son quienes se beneficiarían directamente de la venta de productos a los consumidores, no obstante, no son los únicos que obtendrían ganancias. Los operadores de red, proveedores de servicios y operadores de plataformas: actuarían como intermediarios en la industria, ofreciendo diversos servicios a cambio de una tarifa. De esta manera los usuarios tendrán la opción de comprar dispositivos y/o pagar por servicios, tal como se hace con el cómputo en la nube.

Tabla 2 – Partes interesadas en IoT (Kim & Lee, 2014).

Interesado	Ofrece	Recibe
1. Desarrollador de Dispositivos	Dispositivos IoT	Ganancias de Ventas
2. Proveedor de Servicios	Información y Servicios de IoT	Tarifa por Servicio
3. Operador de Plataforma	Plataformas de Software	Tarifa de Acceso a la Plataforma
4. Desarrollador de Software	Aplicaciones IoT de Servicios y Paginas	Ganancias de Ventas
5. Operador de Red	Servicios de Red	Tarifa por Acceso a la Red
6.Usuario de Servicios	Pago por Servicios IoT	Uso de Servicios IoT

Dicho modelo de negocios permite identificar quienes están interesados en el desarrollo del paradigma y plataformas de IoT.

2.6. Conclusión

En este capítulo se describe el paradigma de IoT y las diferentes visiones que lo conforman. Se ha dado una introducción a la diversidad de objetos que potencialmente deben pertenecer al mismo, así como algunas tecnologías de software y hardware que conforman los objetos inteligentes disponibles actualmente.

Se describe la necesidad de un middleware para vincular los diversos objetos, usando de ejemplo el nuevo paradigma de la Web de las Cosas. Se describen diversos protocolos y los principios REST. Principios que son usados por múltiples soluciones de

middleware y son ampliamente aceptados para el paradigma de IoT. Además se especifican los niveles de abstracción que se puede esperar de los objetos inteligentes. Por último se describe una clasificación de los dominios de aplicación y las partes interesadas en IoT.

Capítulo 3 Web Semántica

El creciente número de dispositivos heterogéneos y distribuidos necesitarán comunicarse e interconectarse sin necesidad de intervención humana. Debido a esto, los objetos deben poseer la capacidad de identificarse y describirse a sí mismos ante otras entidades en la red. Tecnologías propuestas para la Web Semántica (Berners-Lee, *et al.,* 2001) pueden ofrecer algunas soluciones a estos problemas.

3.1. Tecnologías de la Web Semántica

La Web Semántica es un movimiento colaborativo iniciado por el Consorcio de la World Wide Web (W3C) para promover la inclusión de contenido semántico dentro de páginas web. La semántica es el significado de símbolos, palabras, expresiones o representaciones formales.

Los seres humanos son capaces de entender el significado de las palabras en un lenguaje. Por ejemplo: una persona al leer la frase “El gato bebe leche” puede entender el significado de los conceptos “gato” y “leche”, pero además la palabra “bebe” le indica que el gato está realizando la acción de beber la leche. Para que estas capacidades puedan ser usadas en la web, la W3C propuso una pila de tecnologías (ver Figura 14).

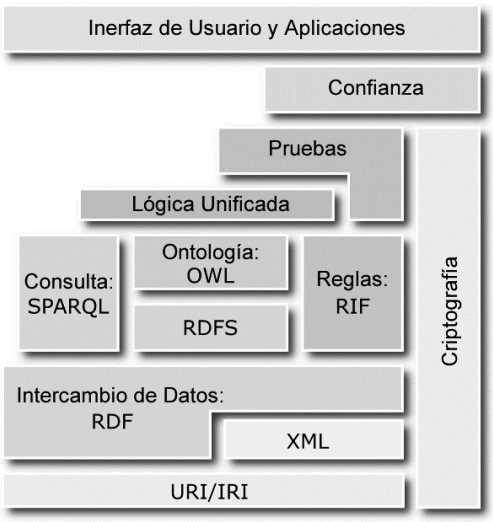


Figura 14 - El "Pastel de Capas". El cual fue propuesto por Tim Berners-Lee para la web semántica.

3.1.1. URI/URL

El primer paso para llevar la semántica a la web es identificar un concepto, para ello se hace uso de URI (Identificador uniforme de recursos). El URI es una cadena de caracteres que identifican inequívocamente un recurso. Un tipo específico de URI denominado URL (Localizador uniforme de recursos) es utilizado para identificar y localizar documentos dentro de la web. Los URI son definidos en el RFC 3986 (Berners-Lee, *et al.,* 2005) y son la base para construir la semántica.

3.1.2. XML

El Lenguaje de Marcado Extensible (XML, por sus siglas en inglés) permite definir un concepto de manera estructurada haciéndolo legible tanto por hombres como para máquinas. El lenguaje permite definir espacios de nombres. Estos espacios se utilizan para definir diversos dominios de descripciones mediante el mismo lenguaje. Por lo tanto un documento XML puede poseer varios conceptos idénticos que pertenecen a dominios distintos, dando como resultado conceptos con diferente significado.

3.1.3. RDF

El Marco de Trabajo para la Descripción de Recursos (RDF, por sus siglas en inglés) es la capa principal de las tecnologías semánticas. Está conformado por tripletas de sujeto, predicado y objeto: lo que permite relacionar conceptos para crear una expresión. Las tripletas se pueden representar por un grafo dirigido en donde tanto sujetos como objetos son vértices, mientras que los predicados son aristas (ver Figura 15).

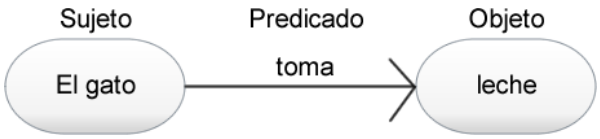


Figura 15 - Ejemplo de una tripleta sujeto, predicado y objeto.

El vocabulario que establece RDF está definido en XML, pero los datos en RDF se puede escribir en diversos formatos, tales como: Turtle, N-Triples, N-Quads, JSON-LD, N3, RDF/XML. Para la web semántica, el W3C recomienda el uso de RDFa. Este funciona

como una extensión a HTML, XHTML y otros documentos XML: agregando metadatos dentro de documentos web ya existentes.

3.1.4. RDF Schema

El Esquema del Marco de Trabajo para la Descripción de Recursos (RDFS, por sus siglas en inglés) es una extensión del RDF para definir modelos de conocimiento. RDFS describe los conceptos de clases y recursos, en los cuales se establecen los elementos básicos para la descripción de ontologías y vocabularios RDF.

De acuerdo al esquema RDFS, todas las entidades se denominan recursos y éstos se pueden agrupar en categorías denominadas clases. Una clase llamada propiedades permite describir la relación entre dos recursos. De esta manera todas las entidades que pertenecen a la clase propiedades serán predicadas. Adicionalmente los conceptos “dominio” y “rango” declaran la clase a la que pertenecen los sujetos y los objetos en relación con el predicado (ver Figura 16).

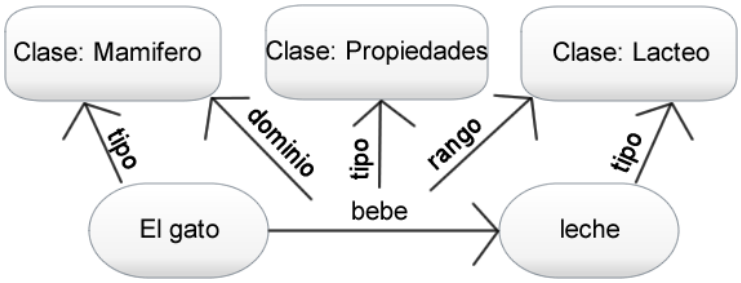


Figura 16 - Ejemplo RDFS. En la imagen se muestran los conceptos básicos de un esquema RFDS.

Una limitante del esquema RDFS es que las entidades se pueden considerar clases y recursos simultáneamente, esto puede crear ciclos entre las definiciones. En otras palabras, se forman ambigüedades dentro del esquema, lo cual reduce las capacidades de inferencia sobre los datos.

3.1.5. OWL

El Lenguaje de Ontologías de la Web (OWL, por sus siglas en ingles), es una familia de lenguajes para representar modelos de conocimiento u ontologías. Como principal diferencia al RDFS, OWL permite distinguir entre clases e instancias, esto evita la

creación de ciclos. Además da soporte a otras características como: cardinalidad, límite de valores, transitividad, entre otras. Esto hace de OWL la recomendación de la W3C para la representación de modelos de conocimiento.

Las variantes del lenguaje: OWL Lite, OWL DL y OWL Full, incorporan diversas capacidades. Mediante el uso de diversos niveles de lógica descriptiva, cada variante ofrece mayor expresividad que el anterior, sin embargo, también incrementa sus limitaciones, por ejemplo: OWL Full puede crear ciclos al igual RFDS.

3.1.6. SPARQL

Siendo un acrónimo recursivo de “SPARQL, Lenguaje y Protocolo de Consulta RDF”, éste lenguaje puede ser usado para la consulta de datos en RDF, incluyendo RFDS y OWL. Similar a SQL, el lenguaje permite hacer consultas realizando uniones, intersecciones y otros patrones (ver Figura 17).

```
1  SELECT COUNT(DISTINCT ?node) as ?spots
2  WHERE {
3      ?node a ssn:Sensor ;
4          ssn:observes ex:Occupancy;
5          dul:hasLocation ?spot.
6      ?spot a ex:ParkingSpot;
7          dul:hasLocation dbpedia:Berlin.
8  }
```

Figura 17 - Ejemplo de SPARQL. Consulta en SPARQL para buscar todos los sensores de estacionamiento en Berlín.

Este lenguaje es la última capa de la Web Semántica que ha sido completamente estandarizado, debido a que el proceso de estandarización de los lenguajes de la web semántica es aún un trabajo en progreso. En la Figura 18 se puede observar la línea de tiempo que compara la estandarización de XML y los lenguajes de la Web Semántica.

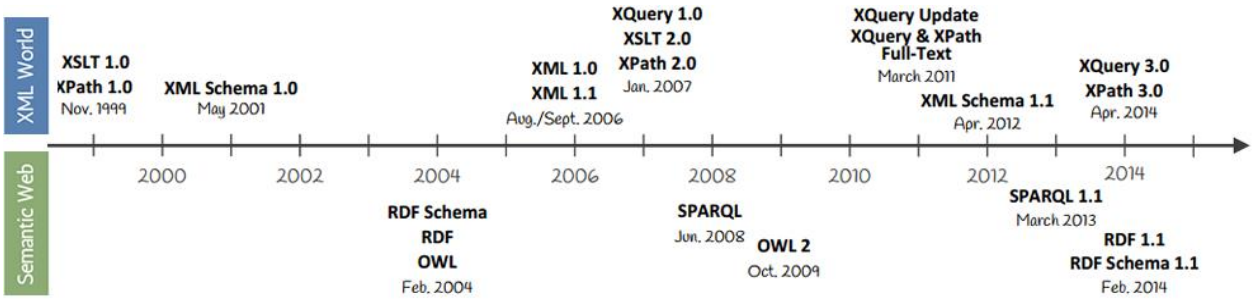


Figura 18 - Línea de tiempo de estandarización de lenguajes de la web semántica (Bikakis, *et al.,* 2013). En esta grafica se observa que hasta el momento solo se han estandarizado las capas inferiores del "Pastel de capas" de la web semántica.

3.1.7. Reglas

El Lenguaje de Reglas para la Web Semántica (SWRL, por sus siglas en inglés) extiende las descripciones de conceptos por las ontologías y los esquemas RDFS. Por ejemplo: si un individuo tiene padre y su padre tiene un hermano, entonces el individuo tiene un tío (en sintaxis SWRL sería: tienePadre(?x1,?x2) ∧ tieneHermano(?x2,?x3) ⇒ tieneTio(?x1,?x3). El uso de reglas para las ontologías es un factor importante para poder hacer inferencias y describir relaciones que no pueden establecerse solamente con lógica descriptiva. Es decir, el uso de reglas hace posible realizar inferencias más complejas.

Aunque el lenguaje SWRL fue concebido para su uso en la web semántica, ya existen bastantes lenguajes de reglas para otros propósitos. Por ello se definió el Formato de Intercambio de Reglas (RIF, por sus siglas en inglés), el cual promueve la interoperabilidad de reglas entre diversos lenguajes y aplicaciones.

3.1.8. Lógica

La lógica es una línea de investigación de la web semántica, por lo que aún no se han definido estándares. El propósito de la lógica es unificar las descripciones ontológicas con las reglas, sin embargo, esta integración típicamente resulta en la violación de uno o más principios de diseño (Knorr, *et al.,* 2012).

El uso de la lógica permitiría realizar operaciones como: revisar la consistencia de la información; realizar comprensión de individuos (determinar las clases más específicas que definen a un individuo); así como validar, clasificar y jerarquizar las clases.

Actualmente están disponibles algunas implementaciones de razonadores semánticos que hacen uso de la lógica, por ejemplo: Pellet[16] o el API de inferencia de Jena[17].

3.1.9. Criptografía

Esta capa pretende brindar seguridad a los datos. Tal como las páginas web utilizan protocolos seguros y certificados digitales para validar su identidad, se pretende que las fuentes de información de la web semántica puedan proteger los datos a través de todas las capas del modelo. Sin embargo, esta capa se encuentra aún en investigación.

3.1.10. Pruebas y confianza

Estas capas son aún conceptuales para la web semántica, su propósito es verificar que la información que se obtiene es cierta. Por un lado las pruebas verifican que los datos no contengan incoherencias mediante las reglas y la lógica unificada, por otra parte la confianza se centra en la calidad de las fuentes de las que proviene la información. La suma de estas dos da como resultado la seguridad para afirmar que los datos son verídicos.

3.1.11. Interfaces de usuario y aplicaciones

Finalmente se encuentran las interfaces y aplicaciones que aprovecharán los datos. Entre éstas se encuentran agentes de software, motores de búsqueda, entre otros. Actualmente las aplicaciones e interfaces se encuentran comúnmente sobre las capas de consulta, dado que las capas superiores de la web semántica aún están en desarrollo. Por lo tanto es común el uso de RDFa para mostrar los datos semánticos directamente en la interfaz de usuario.

3.2. Linked Data

Tim Berners-Lee (2009) ha definido “La Web de los Datos” como una idea con el propósito de conectar todos los datos de Internet y la información virtual de objetos físicos mediante

[16] http://clarkparsia.com/pellet/
[17] https://jena.apache.org/

el uso de tecnologías semánticas. Dada la ausencia de las capas superiores de la web semántica, la web de datos se centra en el uso de las capas inferiores de la pila.

Para la creación de dicha web se definieron las mejores prácticas para publicar información, anunciadas bajo el nombre de modelo de Linked Data (datos enlazados, en español). El propósito de estas prácticas es publicar datos en la web usando tecnologías semánticas bien definidas. Linked Data define que se deben cumplir las siguientes cuatro reglas:

1. Usar URIs como nombres de las “cosas”.
2. Usar URIs HTTP para que las personas puedan buscar esos nombres.
3. Cuando alguien busque una URI, proveer información útil usando los estándares (RDF, SPARQL).
4. Incluir enlaces a otros URIs, para que se puedan descubrir más cosas.

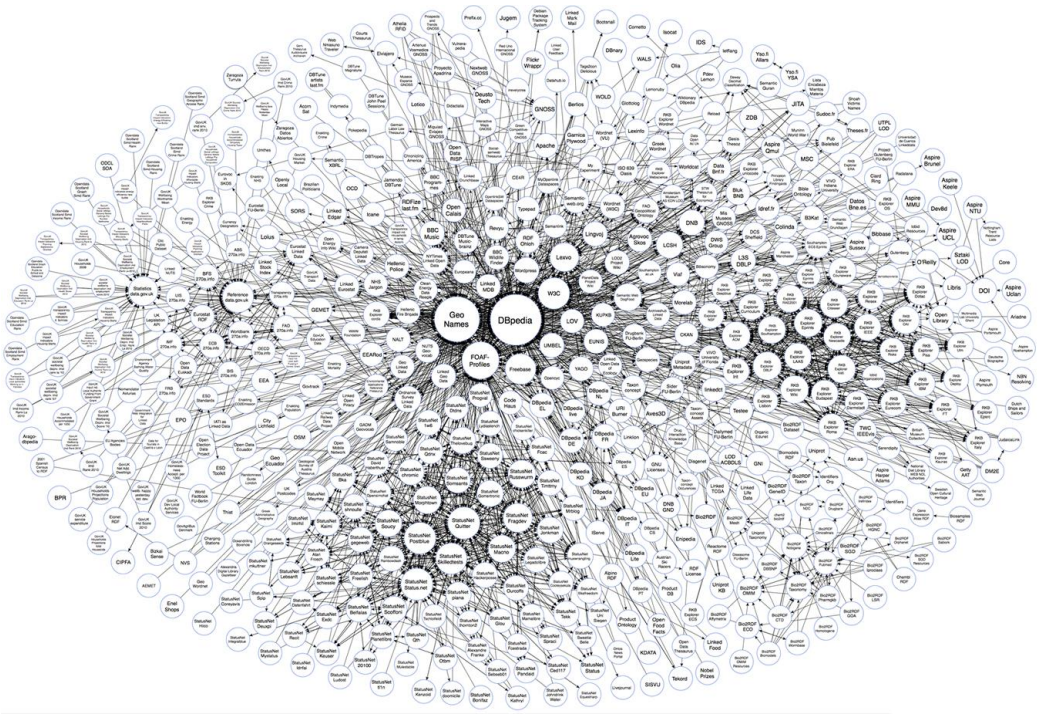


Figura 19 - La nube de Open Linked Data en agosto de 2014[18].

[18] http://lod-cloud.net/

Con la adopción de estas reglas en múltiples sitios de Internet, la iniciativa de crear una red de información ligada con semántica ha cobrado fuerza (ver Figura 19). La red que más ha crecido siguiendo este modelo es conocida como Open Linked Data (red abierta de datos enlazados, en español). Dado que sigue un modelo abierto de datos, permite el uso público de toda la información disponible dentro de la red (Berners-Lee, *et al.,* 2009).

Otras iniciativas de la industria tales como Google's Rich Snippet, Facebook's Open Graph o Schema.org: confirman la tendencia del uso de estas tecnologías en la Internet actual.

3.3. La Web Semántica de Cosas

Los dispositivos de IoT que usan interfaces con modelos REST presentan limitantes al momento de procesar los datos obtenidos. En estas arquitecturas los dispositivos se pueden manipular usando operaciones definidas HTTP, para obtener la información en diferentes formatos mediante la negociación de contenido. Sin embargo, para que los datos de diferentes fuentes se puedan interpretar en un esquema M2M, se requieren descripciones semánticas y modelos de conocimiento bien definidos.

3.3.1. Datos semánticos en Internet de las Cosas

Para ejemplificar las ventajas que tendría el uso de las tecnologías semánticas en el área de IoT se describe el siguiente escenario:

Se tienen datos de distintas fuentes, por ejemplo: sensores de vehículos, base de datos de conductores y señalamientos de tránsito. Mediante esos datos se podría obtener patrones de comportamiento entre los conductores frente a los señalamientos, para ello las tres fuentes deben unirse y analizarse. Suponiendo que las fuentes usan arquitecturas REST, se requiere lo siguiente para exponer su información:

- Obtener los datos sobre: sensores del vehículo, conductores y señalamientos.
- Identificar el formato de los datos obtenidos.

- Programar la relación de la ubicación del vehículo con la ubicación de los señalamientos.
- Programar la relación de los conductores con cada uno de los vehículos.
- Analizar los datos relacionados.

En cambio, con él uso de tecnologías semánticas, los tres tipos de datos poseerían el mismo formato. De esta forma las relaciones entre las bases de datos se realizarían automáticamente mediante el uso de herramientas semánticas. Como resultado los pasos para crear la aplicación se reducirían a los siguientes:

- Realizar una consulta en SPARQL con los tipos de datos de interés.
- Analizar los datos obtenidos

3.3.2. Ontologías para Internet de las Cosas

Para que el escenario descrito previamente sea posible, se requiere que todos los datos posean una representación en RDF, pero además se deben tener modelos de conocimiento que permitan unificar los datos en un esquema en común.

Diversas ontologías se han desarrollado para crear un modelo de conocimiento útil para IoT, una de las más destacables es la Sensor Network Ontology (SSN). Esta ontología desarrollada por la W3C provee los conceptos básicos para describir dispositivos de redes de sensores, tales como: dispositivo, plataforma, despliegue, mediciones, proceso, etc.

Sin embargo, IoT no se limita a sensores, otros dispositivos como actuadores pueden formar parte de la red. La ontología SPITFIRE, desarrollada para la plataforma semántica del mismo nombre, extiende la ontología SSN para el uso de actuadores y otros dispositivos, agregando conceptos como: contexto, actividad, energía, actuadores, y eventos, etc.

Además de los dispositivos, es importante que se describan las interfaces que permiten la interacción de los dispositivos con la web. Minimal Service Model (MSM) es

una ontología simplificada para describir servicios en ambientes SOA. Esta describe conceptos como: recurso, servicio, operación, etc. No obstante, para describir las operaciones del modelo REST existen ontologías como Web API Grounding Model (hRESTS), la cual define específicamente los métodos para interfaces RESTful: GET, POST, PUT, DELETE.

Se han presentado trabajos de nuevas ontologías que proponen unificar la base de conocimiento del paradigma de IoT. Hachem *et al.,* (2011) presentan una ontología que se divide en tres partes: dominio, estimación y dispositivos. Wei-Wang *et al.,* (2012) presentan una ontología que cubre: servicios web, recursos, implementación, calidad de servicio, pruebas de servicios y sitios físicos. Recientemente Nambi *et al.,* (2014) presentaron una ontología para cubrir los aspectos de: recursos, ubicación, domino, contexto, políticas y servicios. Sin embargo, estas ontologías no se han publicado en la web para ser utilizadas universalmente.

3.3.3. Plataformas semánticas de Internet de las Cosas

La capacidad de compartir servicios e información mediante el uso de tecnologías semánticas brinda la oportunidad de crear aplicaciones a un nivel más alto en la jerarquía del conocimiento. Es por ello que se han presentado diversas plataformas que hacen uso de tecnologías semánticas.

Inspirado en la creciente red de Open Linked Data, el proyecto Sense2Web (Barnaghi, *et al.,* 2010) propone una arquitectura para publicar información sobre redes de sensores que sigue los principios de Linked Data. Por otra parte, la plataforma SPITFIRE (Pfisterer, *et al.,* 2011) propone el uso de dispositivos virtuales para almacenar descripciones semánticas de sensores y actuadores, lo cual habilita la interactividad de dispositivos a nivel semántico y la capacidad de realizar consultas de los dispositivos en la red mediante el lenguaje SPARQL. La plataforma u-KB (Ruta, *et al.,* 2012) conecta diversos dominios de implementación de dispositivos mediante una capa semántica. Sin embargo, no existe a la fecha una plataforma que aborde todos los aspectos de IoT (Perera, *et al.,* 2014).

Las plataformas semánticas permiten describir todos los objetos usando tecnologías homogéneas. Esto da lugar a que se puedan establecer interfaces estándar para interactuar con los objetos de IoT. Por ejemplo: los Visores Sensoriales (Estrada-Martinez y Garcia-Macías, 2012) permiten navegar por dispositivos descritos semánticamente de forma similar a como los navegadores web lo hacen en páginas de Internet.

3.4. Conclusiones

En este capítulo se describen las tecnologías que conforman la Web Semántica y los principales estándares que se tienen hasta el momento. Además se describe el modelo de Linked Data que pretende proliferar el uso de las tecnologías semánticas ya estandarizadas.

Se describe un escenario en donde el uso de tecnologías semánticas facilitaría el análisis de múltiples fuentes de datos con el fin de ejemplificar su relevancia para el paradigma de IoT. Finalmente se describen algunas de las ontologías y plataformas semánticas más relevantes a la fecha para el paradigma de IoT.

Capítulo 4 Diseño de la plataforma OpenThings

En los capítulos anteriores se describen los paradigmas de IoT y la Web Semántica. En este capítulo se describe la lista de los requerimientos más relevantes para crear una plataforma de Internet de las Cosas. Además se describe a detalle los principios y características de la arquitectura de software de OpenThings, así como algunos ejemplos de uso.

4.1. Requerimientos

Para crear una plataforma se requiere conocer el entorno en el cual se va a utilizar. Por ello se realizó un análisis amplio de la literatura, del cual se identificaron las características más importantes que debe tener una plataforma de IoT.

Además de los requerimientos de IoT, se analizaron los requerimientos para desarrollar sistemas de cómputo ubicuo, ya que el cómputo ubicuo es una de las áreas más beneficiadas del paradigma de IoT.

Por otra parte IoT busca conectar todos los objetos del planeta a Internet, lo cual habilitaría a casi cualquier sitio como un ambiente inteligente. Por lo tanto se debe tomar en cuenta los requerimientos del área de Inteligencia Ambiental. Esta área estudia los entornos que son sensibles y responsivos a la presencia de personas.

4.1.1. Requerimientos de Internet de las Cosas

El desarrollo de plataformas para IoT es un área muy activa (Bandyopadhyay, *et al.,* 2011). Los distintos enfoques al paradigma de IoT; la heterogeneidad de dispositivos; y las diversas áreas de aplicación: dan lugar a un gran número de propuestas. Esto ha permitido que se identifiquen diversos requerimientos para crear estas plataformas. A continuación se mencionan aquellos requerimientos que se han considerado en este trabajo para el desarrollo de una plataforma de IoT.

El proyecto IoT-A describe una extensa lista de requerimientos de las áreas de: ciudades inteligentes, e-Health, movilidad, logística, transporte y comercialización (Internet of Things Arquitecture Project, 2011). Como parte de ese trabajo, se describe una encuesta realizada a expertos en el área para clasificar los requerimientos en orden de importancia. Usando una escala Likert de 5 niveles que va desde muy poco importante a muy importante, se clasifican todos los requerimientos que se encontraron por ese grupo de trabajo. Para este trabajo de tesis se tomaron en cuenta todos los requerimientos que superaron la media en la escala Likert. La lista completa se encuentra en el Apéndice 1.

Para los requerimientos de middleware de IoT se tomó el trabajo realizado por Lee *et al.,* (2013). En éste se describen requerimientos de alto nivel para IoT. La lista completa se puede ver en el Apéndice 2. Para complementar esta lista de requerimientos, en el trabajo de Bandyopadhyay y Sengupta (2011) se describen los componentes fundamentales que debe poseer un middleware de IoT. La lista completa se encuentra en el Apéndice 3.

4.1.2. Requerimientos de cómputo ubicuo e inteligencia ambiental

Para incluir los requerimientos de sistemas de cómputo ubico, se tomaron en cuenta los requerimientos descritos en el trabajo de Tim Kindberg y Armando Fox (2002). En este trabajo se describen los dos requerimientos primordiales para sistemas de cómputo ubicuo: integración física e interacción espontánea. La descripción a detalle se encuentra en el apéndice 4.

Para los requerimientos del área de inteligencia ambiental se analizaron los trabajos de Weber, *et al.,* (2005) y Nehmer, *et al.,* (2006), quienes describen los requerimientos no funcionales que debe poseer un sistema de esa área. La lista completa se encuentra en el apéndice 5.

4.1.3. Agrupación de requerimientos

Con el objetivo de incorporar todos los requerimientos previamente mencionados en una lista más legible y manipulable, se agruparon los requerimientos por características similares. Sin embargo, algunos de los requerimientos están altamente relacionados con múltiples características, por lo que se produce una interpolación de requerimientos en la agrupación realizada.

Un diagrama completo de todos los requerimientos citados se encuentra en la Figura 20. En ésta grafica se pueden observar los principales temas a considerar para el desarrollo de una plataforma de IoT, como el tema de interoperabilidad, el cual agrupa alrededor del 31% de los requerimientos encontrados. A continuación se describen a detalle los principales temas encontrados.

Interacción: La plataforma debe dar soporte a dispositivos que se encuentran en la vida cotidiana, ya sea que estén o no a la vista del usuario. Esto hace necesario analizar: los tipos de interacción que tendrá el usuario con los dispositivos (Interfaz Hombre-Máquina); la interacción entre dispositivos (Máquina-Máquina); y el comportamiento de los dispositivos en presencia de un usuario. Estas interacciones deben realizarse de forma semántica, tomando en cuenta las características de cada dispositivo y su relación con el mundo físico. Además se deben proveer los mecanismos adecuados para interactuar y procesar los datos de estos dispositivos.

Aplicación: Serán las aplicaciones las que proveerán servicios de alto nivel para los usuarios. Éstas deben tomar en cuenta que la comunicación con los usuarios debe realizarse de forma natural y se debe usar la semántica para interactuar con otros dispositivos (M2M). Por otra parte, el sistema debe permitir a las aplicaciones el uso de información y servicios externos a la plataforma. Finalmente se debe controlar si los servicios usados por la aplicación serán usados anónimamente para proteger al usuario o si es necesario poseer permisos para el uso los mismos.

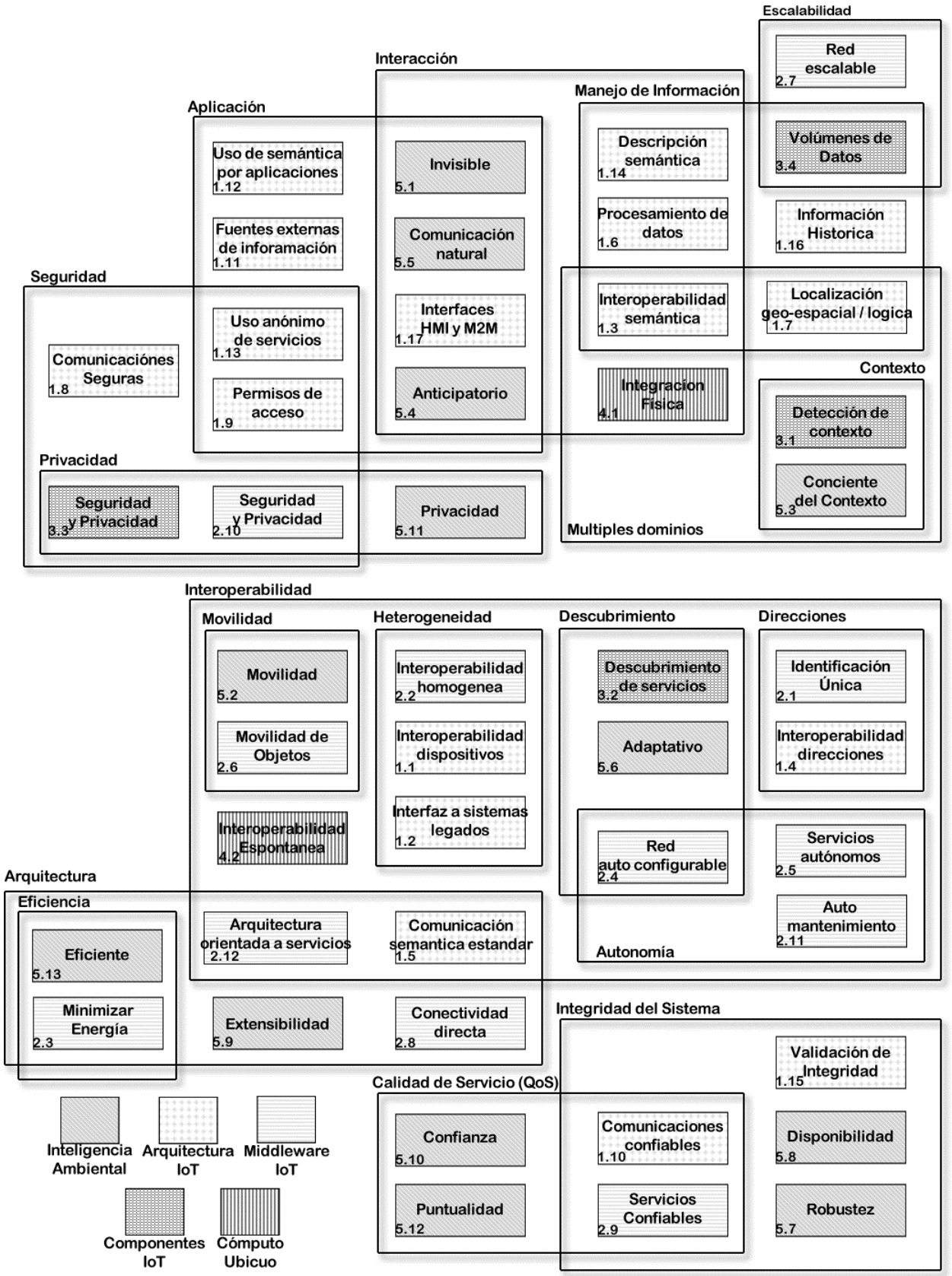


Figura 20 - Lista de requerimientos de IoT. Los requerimientos se agrupan por similitud y características en común.

Manejo de Información: La inmensa cantidad de datos que se pueden obtener trae consigo la necesidad de proveer mecanismos de manejo de datos como: filtrado, agregación, fusión, búsqueda, minería, entre otros. Estos datos se deben describir semánticamente para permitir relacionarlos con diferentes modelos del conocimiento. Además se debe tomar en cuenta los grandes volúmenes y diferencias de los datos. También es de fundamental importancia establecer mecanismos para agregar información de localización geo-espacial o localización semántica, así como información histórica de los datos.

Escalabilidad: La plataforma debe permitir la conexión de una red global con miles de millones de dispositivos. Estos dispositivos van a producir una inmensa de datos que deben ser distribuidos en la red. Se deben establecer mecanismos para almacenar y manipular estos grandes volúmenes de datos tomando en cuenta que muchos de los dispositivos poseen capacidades limitadas.

Múltiples dominios: Las diversas asociaciones culturales, administrativas, o territoriales forman diversos modelos de conocimiento que se pueden aplicar a los dispositivos así como a los datos que se generan. Usar el modelo correcto implica conocer el contexto en el que se encuentra un dispositivo y el uso que se le esté dando, por lo que es de vital importancia conocer la ubicación de ese dispositivo. Por ejemplo, en una tienda, la información más importante de un vaso es el precio y el tiempo que lleva en exhibición, mientras que en una oficina, la información más importante es quien es el dueño del vaso. Describir los múltiples dominios implica que exista una interoperabilidad semántica entre diversas descripciones del mismo objeto.

Contexto: La plataforma debe ofrecer métodos para generar y estimar el contexto de dispositivos con el fin de darle a un dispositivo conciencia de su alrededor. El contexto es clave para seleccionar dominio en el que se usa un dispositivo, por lo tanto la información de contexto puede cambiar fundamentalmente la interacción del dispositivo con otros objetos.

Seguridad: Una de las características que se deben tomar en cuenta para el desarrollo de aplicaciones de IoT es el control de la seguridad. Los dispositivos personales o dispositivos del área de la salud requieren que la comunicación se realice de forma segura. Se deben establecer permisos de acceso para el control de servicios críticos o privados. Por ejemplo: un individuo debe tener control sobre la puerta de su casa, incluso podría asignar permisos a familiares o amigos, sin embargo, el control de la puerta estaría fuera del alcance del público.

Privacidad: La gran cantidad de datos e información que se pueden derivar de los dispositivos podrían quebrantar los derechos de privacidad de una persona, grupo u organización. Establecer un modelo de privacidad que asegure un cierto nivel de protección a los datos es crucial para la aceptación de aplicaciones de IoT. El control de dispositivos debe estar directamente asociado con las políticas de privacidad, por ejemplo: usar un dispositivo inteligente en un espacio público podría requerir que el usuario dé permiso para conocer su identidad; sin embargo, el dispositivo puede usar esta información para asociar al usuario con un lugar y hora específico. Además los datos pueden ser usados para derivar información de comportamiento si el usuario usa el dispositivo en múltiples ocasiones. Que el usuario no este consiente de todos los datos que se están obteniendo de él pone en riesgo su privacidad.

Interoperabilidad: El requerimiento más importante para el paradigma. Si se quiere tener una red de todos los objetos del planeta, es imperativo poseer interoperabilidad entre dispositivos con características distintas de comunicación, procesamiento y almacenamiento. Alcanzar la interoperabilidad requiere de: identificar inequívocamente a cada uno de estos dispositivos; ofrecer mecanismos de movilidad a los objetos; establecer métodos de comunicación estándar; brindar interacción espontánea entre dispositivos; y auto-configuración de los dispositivos.

Movilidad: Un dispositivo de IoT puede estar en movimiento. El movimiento implica que el dispositivo este cambiando constantemente el contexto y de otros dispositivos que lo rodean. Es necesario que las tecnologías de comunicación permitan

esta movilidad, facilitando la localización de un dispositivo dentro de la red y su conectividad.

Heterogeneidad: Mientras que las etiquetas RFID solo permiten escritura y lectura de información, los teléfonos inteligentes poseen más poder de cómputo que las computadoras de la década pasada. Esta gran brecha de tecnología crea un ambiente con objetos completamente diferentes. Al mismo tiempo sistemas de redes de sensores y middleware para ciertos dispositivos forman parte inherente del paradigma. La plataforma de IoT debe brindar interoperabilidad entre todos ellos.

Descubrimiento: Cada vez que un dispositivo se conecta a una red, o llega a un sitio, necesita: identificar los objetos que lo rodean, adaptarse a los servicios disponibles y hacer lo posible por auto configurarse con la mínima intervención humana.

Direcciones: Cada objeto debe poseer una Identificación única. En principio esta idea fundó el paradigma de IoT. Sin embargo, se debe tener interoperabilidad no solo para identificar objetos, sino también para las direcciones y nombres de los mismos.

Autonomía: En una red con miles de millones de servicios, un control centralizado queda descartado debido a que la complejidad del mantenimiento sería demasiado grande. Los servicios deben ser completamente autónomos, deben tener auto mantenimiento y deben ser capaces de funcionar en una red constantemente cambiante.

Arquitectura: Las características particulares del paradigma han llevado a identificar elementos arquitectónicos deseables para cualquier plataforma de IoT. Tal como la Internet, se debe dar conectividad directa de extremo a extremo (End to End, en inglés). Se recomienda el uso de arquitecturas SOA y el uso de componentes que permitan extender continuamente la arquitectura. Además el uso de comunicaciones semánticas es deseable. También se debe tomar en cuenta que la red estará formada por dispositivos con capacidades limitadas.

Eficiencia: Las limitaciones de procesamiento, memoria, ancho de banda y batería de los dispositivos: son características claves en una arquitectura de IoT. Se debe hacer uso de comunicaciones y protocolos ligeros que reduzcan en lo posible el consumo de batería. La carga de trabajo en los dispositivos limitados se debe reducir al mínimo.

Integridad del sistema**:** El sistema debe funcionar pese al mal uso o errores de hardware. El sistema debe ser robusto debido a la gran cantidad de dispositivos por usuario, la volatilidad del hardware y el dinamismo de los dispositivos conectados. Para que esto suceda se debe validar de integridad de entidades virtuales, dispositivos, servicios y recursos. Esta validación puede ofrecer un cierto grado de confiabilidad en servicios y comunicaciones de la red.

Calidad de servicio (QoS): *Se* debe asegurar cierto nivel de confianza en el funcionamiento de servicios de emergencias médicas, de seguridad vial y otras aplicaciones críticas. El tiempo de entrega y la seguridad de los mensajes son características necesarias para este tipo de aplicaciones. Por lo tanto se deben establecer mecanismos para asegurar la calidad de servicio y la calidad de experiencia de usuario.

4.2. Arquitectura de software OpenThings

La arquitectura de software permite describir la estructura, el funcionamiento e interacción entre las partes de software que conforman un sistema. Ésta tiene un papel fundamental para una plataforma de IoT, en donde los dispositivos (hardware) son sumamente heterogéneos. Es en la arquitectura en donde se describe cómo se integran y llevan a cabo las interacciones entre dispositivos.

Para diseñar la arquitectura de software se debe tomar en cuenta los requerimientos de las plataformas de IoT. La agrupación de requerimientos nos permite identificar aquellos que son claves para el desarrollo de una plataforma IoT; sin embargo, cubrir todos los requerimientos de un área tan amplia como IoT resulta impráctico.

Si bien cubrir todos los requerimientos el objetivo final de crear una plataforma IoT, para este trabajo de tesis se concentraron los esfuerzos mejorar el soporte de: interoperabilidad, heterogeneidad, contexto, múltiples dominios, aplicaciones, descubrimiento y direcciones.

4.2.1. Vista general

La arquitectura de software propuesta en este trabajo de tesis se extiende de UbiSOA, una plataforma propuesta por Edgardo Avilés (2012). UbiSOA (en referencia a Ubiquitous Service-Oriented Arquitecture) cubre parcialmente los requerimientos encontrados anteriormente, por lo que es un excelente punto de inicio para continuar su desarrollo y crear una plataforma más completa.

La arquitectura de software OpenThings, que se propone en este trabajo, tal como UbiSOA, sigue un modelo distribuido. Además cada dispositivo posee una arquitectura en capas. Estas características permiten el uso de diversas tecnologías en cada uno de los dispositivos.

A grandes rasgos la arquitectura propuesta se puede dividir en cuatro componentes principales: servicios, servicios virtuales, ambientes virtuales y aplicaciones. Es importante separar las responsabilidades de cada uno de los componentes para definir claramente su aportación y comportamiento dentro de la arquitectura.

Los servicios encapsulan cada uno de los objetos de Internet de las Cosas y son los que proveen interfaces de interacción con los diversos recursos de los objetos físicos. Los servicios virtuales, en contraste con los anteriores, no encapsulan un objeto, pero son usados como apoyo a otros componentes. Los ambientes virtuales encapsulan ubicaciones físicas o conceptuales, éstos almacenan información contextual de servicios y aplicaciones. Las aplicaciones hacen uso de otros componentes para ofrecer funciones o servicios de alto nivel a los usuarios.

4.3. Servicio OpenThings

Es la unidad más básica de la arquitectura, su propósito es establecer un modelo de interacción homogéneo que encapsule cualquiera de los objetos del paradigma de IoT. Este servicio está compuesto por diversos componentes como se muestran en la Figura 21. Estos se describen con mayor detalle a continuación.

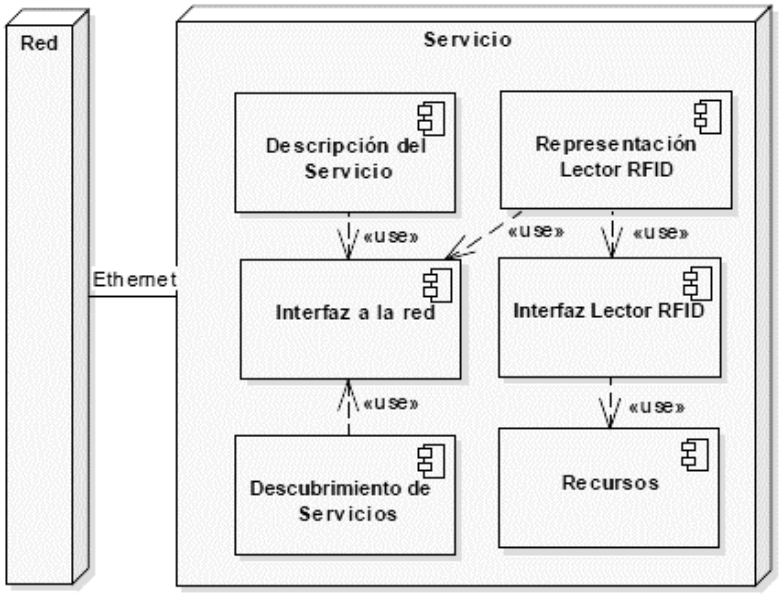


Figura 21 - Estructura básica de un servicio OpenThings.

4.3.1. Recursos

Son las capacidades de los objetos que se desea conectar a la red: pueden ser entidades de hardware como sensores o actuadores; o pueden ser entidades de software como un sistema de almacenamiento o el API de una red de sensores. Cada servicio puede encapsular múltiples recursos, decidir cuantos recursos se agruparan en un servicio está en manos del desarrollador. Sin embargo, se debe tomar en cuenta que esta granularidad puede tener un impacto en el comportamiento y requisitos mínimos para implementar dichos recursos.

Por ejemplo: un mote puede tener sensores de humedad, iluminación y temperatura. Estos recursos se pueden encapsular en un solo servicio, lo que hace posible relacionar directamente características en común como: la batería del dispositivo; las características del mote; y las condiciones de funcionamiento. A nivel de plataforma

los 3 sensores serían vistos como un solo servicio. Por otra parte si se crea un servicio por cada sensor, la información en común del dispositivo se tendría que describir 3 veces, esto crearía redundancia de datos. Esto ofrece que cada sensor sea visto como una entidad independiente, lo cual da más flexibilidad a nivel plataforma

4.3.2. Interfaces de recursos

En estos componentes se deben crear una las interfaces homogénea de alto nivel para los objetos del paradigma, independientemente de la interfaz que tengan sus recursos. Además debe exponer la funcionalidad de los recursos a través de un servicio Web que sigue el modelo REST, de acuerdo con la arquitectura de UbiSOA. Por lo tanto este componente es la base para conectar dispositivos que poseen capacidades más limitadas (e.g., lectores RFID, códigos QR). Los dispositivos u objetos a conectar a la plataforma se pueden separar en tres categorías dependiendo de las capacidades: objetos inteligentes, objetos asistidos, objetos simples

Objetos inteligentes: son dispositivos que poseen suficientes capacidades de cómputo, almacenamiento y comunicación para implementar todos los componentes necesarios para crear el servicio. Entre estos dispositivos se encuentran: computadoras, teléfonos inteligentes, Raspberry Pi, Intel Edison, Motes, entre otros.

Objetos asistidos: son aquellos dispositivos que no pueden implementar todos los componentes de un servicio, por lo que requieren de un dispositivo complementario para implementar los componentes faltantes. Un ejemplo serían muchos de los lectores RFID que se conectan al puerto USB y no permiten ser programados. Para esos lectores se debe utilizar un dispositivo adicional como un Raspberry Pi para implementar el resto de componentes del servicio (ver Figura 22).

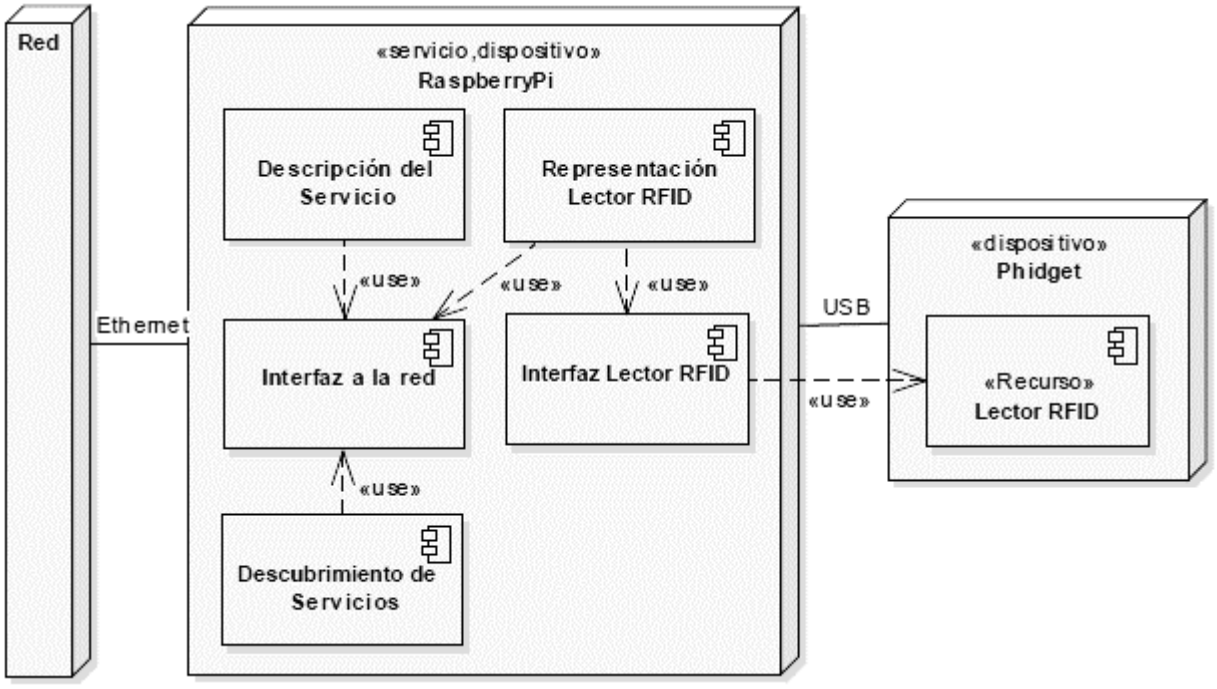


Figura 22 - Ejemplo de implementación de un servicio de lector RFID. En el diagrama la representación del servicio interactúa con el lector RFID a través de USB y expone sus funciones usando la interfaz web.

Objetos simples: son aquellos objetos que no poseen capacidad de cómputo, almacenamiento o comunicación, en principio pueden ser cualquier objeto (e.g., una silla o una caja de galletas). Estos objetos requieren de una representación virtual con la cual otros dispositivos puedan interactuar. Estos objetos deben apoyarse de alguna tecnología para asociar directamente el objeto con su representación virtual. Ejemplos de estas tecnologías son: las etiquetas RFID, etiquetas NFC, códigos QR, o Códigos de barras. Sin embargo, estas tecnologías son principalmente un medio de almacenamiento, usualmente no mayor a decenas de KB.

Figura 23 - Ejemplo de código de barras con una URL acortada que direcciona a la página http://www.cicese.edu.mx

Extender la capacidad de almacenamiento de una etiqueta es posible al incluir una URL que indique la dirección del recurso virtual. Técnicas para acortar el URL pueden ser utilizadas para hacer uso de códigos de barras alfanuméricos (ver Figura 23). De esta

manera el recurso virtual y el resto de los componentes del servicio serían implementados en un dispositivo externo. Este dispositivo externo podría ser un servidor de alta capacidad (ver Figura 24), lo que eliminaría las limitaciones de almacenamiento y proporcionaría todas las capacidades de un servicio web a esa etiqueta.

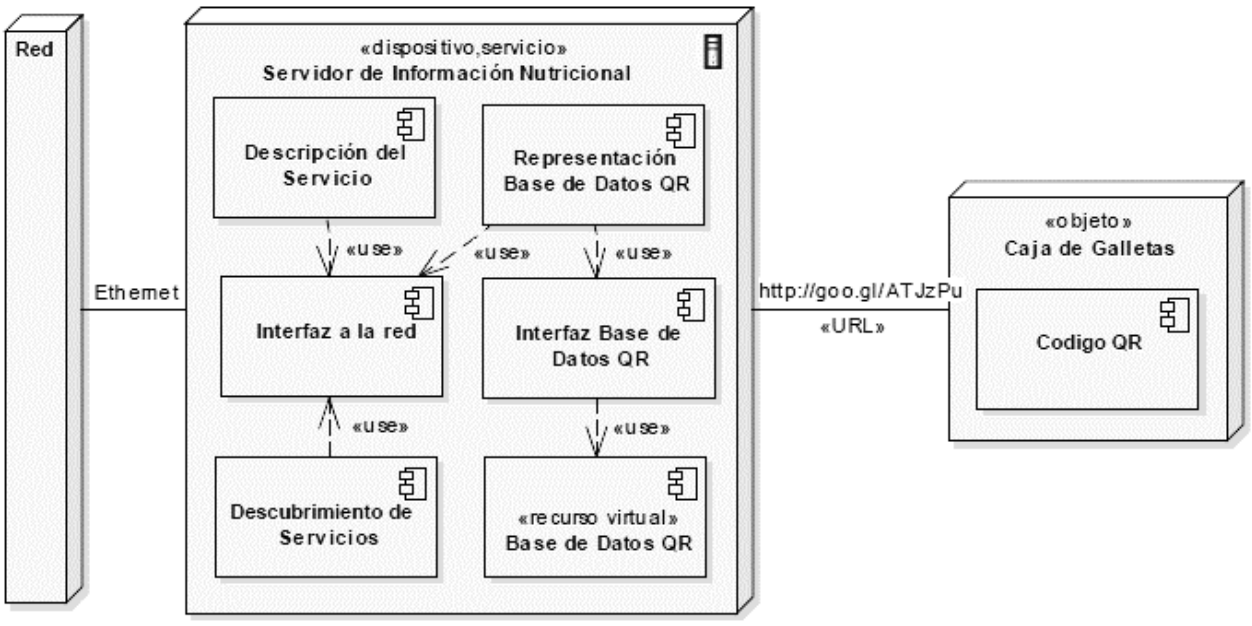


Figura 24 - Ejemplo de implementación de un servicio OpenThings en un servidor de alta capacidad. El servidor representa un objeto real usando el enlace URL de un código QR.

La interfaz REST provee los mecanismos necesarios para la comunicación cliente-servidor entre los componentes de la arquitectura. No obstante, recursos que producen eventos como el presionar un botón o la lectura de una etiqueta necesitan un modelo de comunicación “publisher-subscriber”. El objetivo de “publisher-subscriber” sería evitar que los clientes estén en espera de una actualización mientras se realizan peticiones periódicas a los servicios. Esta acción de hacer peticiones constantes se conoce como la técnica de “polling”.

Si los recursos dentro de un servicio producen eventos, la interfaz de recursos debe usar alguna tecnología que permita el modelo “publisher-subscriber”. En caso de que la interfaz se realice usando CoAP, este protocolo ofrece un modelo de comunicación por medio de la capacidad de “observar” un recurso. En caso de HTTP, la interfaz debe implementar Server-sent Events (SSE), el cual es un estándar que forma parte de la especificación de HTML5 (Hickson and Google Inc., 2014). Esta técnica crea una

conexión unidireccional del servidor al cliente. WebSockets es otra técnica similar que provee una conexión bidireccional (Fette, et al., 2011), aunque podría ser utilizada, el modelo bidireccional resulta innecesario puesto que el servicio OpenThings por definición no requiere subscribirse al cliente.

4.3.3. Representaciones de recursos

Al Igual que en la web, el intercambio de información de los recursos se lleva a cabo por medio de documentos. En el componente de representación de recursos se definen los formatos de éstos documentos. La arquitectura UbiSOA describe que se debe usar al menos uno de estos formatos: HTML, XML, JSON o ATOM. Sin embargo, de acuerdo a los requerimientos previamente planteados, debe ser posible relacionar los datos semánticamente. Por ello se establece que: además de las representaciones mencionadas anteriormente, cada recurso debe poseer una representación semántica en RDF usando alguno de los formatos de ésta representación. Entre los formatos más comunes se encuentran: N3, RDF/XML, Turtle, N-Triples, N-Quads, JSON-LD, RDFa.

Queda a consideración del desarrollador elegir el formato de representación más adecuado para el servicio, sin embargo, para mejorar la interoperabilidad es recomendable seguir las prácticas comunes en la web semántica. Linked Data, de Tim Berners-Lee (2009), establece que el formato común es RDF/XML, y en situaciones donde se requiere interacción humana N3 y Turtle son alternativas viables. Adicionalmente se puede usar RDFa dentro de HTML, sin embargo, se debe ser cuidadoso con el uso de URI para identificar entidades.

Para publicar contenido semántico es necesario distinguir el URI que identifica a una entidad y el URI que identifica los documentos que describen esa entidad. En las prácticas comunes en la web semántica diferenciar estas URI se debe realizar siguiendo las prácticas de Cool URI (Ayers and Völkel, 2008), las cuales se explican a continuación.

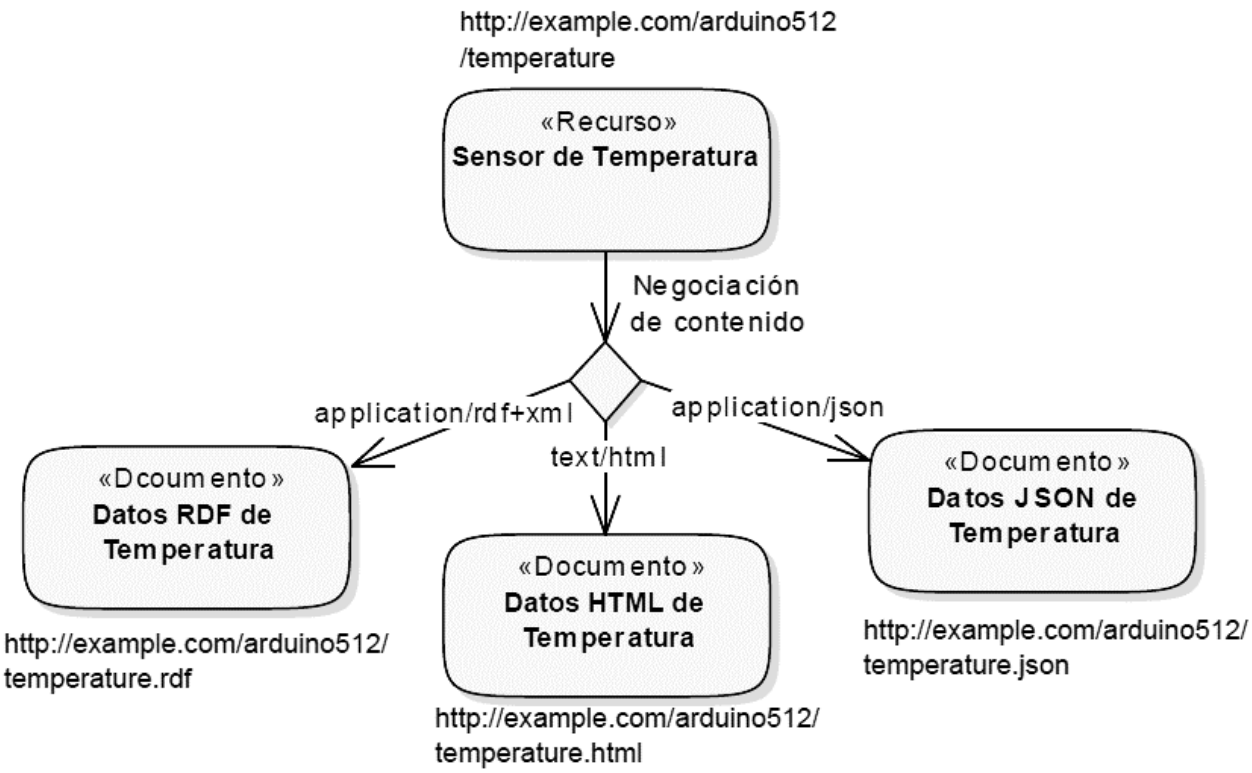


Figura 25 - Negociación de contenido que sigue las prácticas de Cool URI.

Para mostrar cómo funciona Cool URI se tiene el siguiente ejemplo: en la Figura 25 la URI del recurso se distingue de los documentos usando las extensiones de los archivos .rdf, .html y .json. Al agregar la extensión en la dirección de cada documento, las cuatro entidades poseen un URI distinta. Esta distinción permite diferenciar semánticamente al dispositivo y los documentos que lo representan. Finalmente para seleccionar el documento adecuado que solicite la aplicación, se usará la negociación de contenido de HTTP o CoAP según sea el caso.

Los datos en formatos diferentes a RDF pueden contener la información mínima necesaria para su aplicación, esto con el fin de optimizar la comunicación entre dispositivos de recursos limitados (ver Figura 26).

```
{
    "title" : "Posicion de Persiana"
    "minimum": "0.0",
    "maximum": "220.0",
    "current": "0.0"
}
```

Figura 26 - Ejemplo de datos de un servicio de persiana inteligente en formato JSON.

Sin embargo, los datos en RDF deben enlazar a la descripción semántica del servicio, y además anexar la información semántica del dispositivo a los datos producidos por el recurso (ver Figura 27).

```
<?xml version="1.0" encoding="utf-8"?>"
    <rdf:RDF"xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:spt="http://spitfire-project.eu/ontology/ns"
    xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#">

        <ssn:Observation rdf:about="http://skylab.local/persiana/data">
            <rdfs:label>Posicion de Persiana</rdfs:label>
            <spt:minValue>0</spt:minValue>
            <spt:value>120</spt:value>
            <spt:maxValue>180</spt:maxValue>

            <rdfs:seeAlso rdf:resource="http://skylab.local/persiana/data/.well-known/description">
        </ssn:Observation>
    </rdf:RDF>
```

Figura 27 - Ejemplo de datos de un servicio de persiana inteligente en formato RDF/XML. La etiqueta "rdfs:seeAlso" indica que la descripción semántica se encuentra en la ubicación "http://skylab.local/blind/data/.well-known/description".

4.3.4. Descubrimiento de servicios

Los servicios deben ser capaces de encontrar otros dispositivos dentro de la red para lograr una red auto configurable. Por ello un componente de descubrimiento debe estar presente en toda entidad del paradigma. UbiSOA establece un mecanismo de descubrimiento basado en los trabajos del grupo Zero Configuration Networking[19]. Este mecanismo anuncia en la red la siguiente información del servicio: nombre, descripción, servidor, numero de puerto, URL de los puntos de acceso REST, URL de un servicio HUB y las URLs de los documentos WADL que describen las interfaces del servicio. No obstante, la nueva arquitectura asigna toda esta información a la descripción semántica del servicio. Por lo tanto se reducen los datos necesarios para el descubrimiento a solamente: nombre, puerto y URL del servicio.

Debido a la popularidad del protocolo DNS-SD en el grupo Zeroconf, se establece que en este protocolo los servicios de la plataforma se anunciaran usando el prefijo "_openthings".

[19] http://www.zeroconf.org/

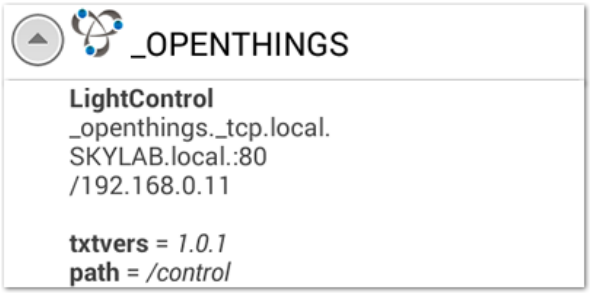


Figura 28 - Ejemplo de un servicio OpenThings anunciado en la red local con el protocolo DNS-SD.

Tal como se muestra en la Figura 28, con esa información podemos encontrar el servicio dentro de la red local usando el URL “http://SKYLAB.local/control”. Sin embargo, para encontrar dispositivos de una red externa se debe usar mecanismo más complejo descrito en la sección de ambientes virtuales.

4.3.5. Descripción del servicio

Describir de forma precisa los servicios y sus características principales es un punto clave para lograr interoperabilidad instantánea e interacción M2M. Cada servicio debe poseer una descripción semántica de sí mismo. Se propone que esta descripción debe estar en una dirección bien conocida “Well-Known” como se describe en el RFC5785 (Nottingham and Hammer-Lahav, 2010). Está dirección URL se propone como: “/.well-known/description”.

En esta dirección bien conocida se debe alojar la descripción del servicio usando alguna representación de RDF, preferentemente usando los formatos más usados en Linked Data y siguiendo los principios de Cool URI.

La dirección que representa al objeto es la dirección anunciada por el descubrimiento de servicios (e.g., http://example.com/arduino512) y de acuerdo con Cool URI éste debe re direccionar a la URL de la descripción semántica. Además dado que la descripción semántica está en una URL bien conocida, la aplicación puede optar por realizar la petición directamente a la descripción y evitar el re direccionamiento. Una petición directa es útil para reducir el número de mensajes y ahorrar batería, aunque no es estrictamente necesario.

La descripción semántica debe ofrecer el contenido al menos en algún formato RDF, sin embargo, según los principios de Linked Data las descripciones en HTML son deseables para brindar una interfaz hombre-máquina. Una vez obtenidos los documentos de descripción del servicio, en éstos se pueden encontrar los enlaces a los recursos del servicio (ver Figura 29).

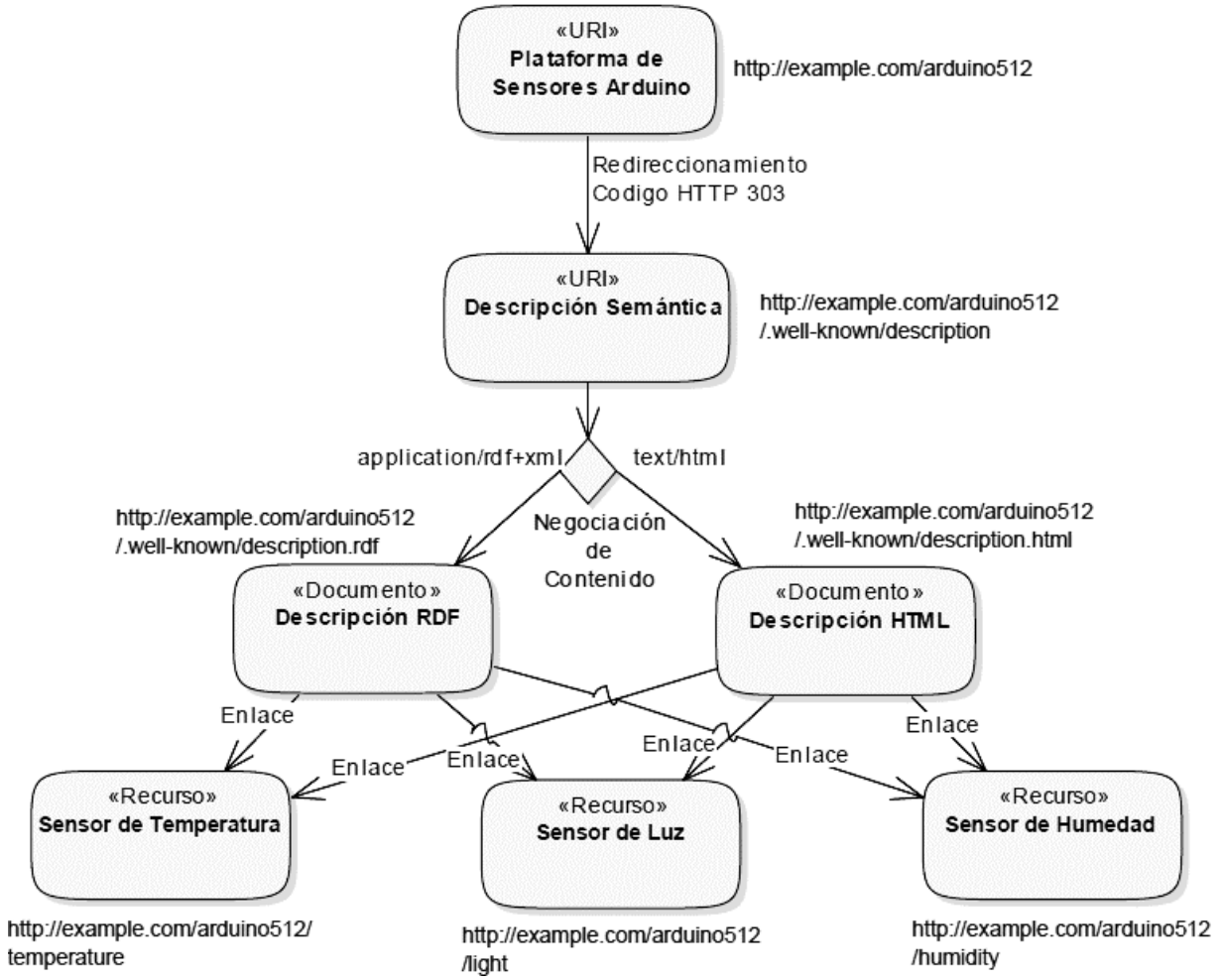


Figura 29 - Ejemplo de un servicio siguiendo las prácticas de Cool URI y Linked Data para enlazar sus recursos.

Debido a la naturaleza de recursos limitados del paradigma, los servicios podrían estar alojados en dispositivos con poca memoria disponible. Por lo anterior se establece que la descripción del servicio debe estar limitada a la información mínima necesaria para describir sus características e interfaces del servicio. La descripción entonces se puede dividir en 2 partes conceptuales.

Características del servicio: Se debe describir de forma precisa la información básica del servicio, tal como: nombre, plataforma de hardware, recursos disponibles y funciones de alto nivel. La descripción se debe realizar usando de base las ontologías SSN y SPITFIRE. En la Figura 30 se muestra un ejemplo de descripción de un servicio.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:spt="http://spitfire-project.eu/ontology/ns/"
xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#"
xmlns:hr="http://iserve.kmi.open.ac.uk/ns/hrests#">

    <hr:Service rdf:about="http://skylab.local/persiana/well-known/description">
        <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#Device"/>
        <rdf:type rdf:resource="http://cicese.mx/openthings.owl#Service"/>
        <rdfs:label>Descripcion de la Persiana</rdfs:label>
        <spt:actuate rdf:resource="http://dbpedia.org/resource/Window_blind"/>
        <ssn:featureOfInterest rdf:resource="http://spitfire-project.eu/ontology/ns/Light"/>
        <ssn:hasQuality rdf:resource="http://spitfire-project.eu/ontology/ns/actuate"/>

        **// Interfaz del Servicio //**

    </hr:Service>
</rdf:RDF>
```

Figura 30 - Ejemplo de código RDF/XML describiendo características básicas del servicio de una persiana inteligente. Entre asteriscos se encuentra la sección donde estaría la interfaz del servicio.

Interfaz del servicio: Se deben describir semánticamente los puntos de acceso REST que permiten interactuar con el servicio. La arquitectura UbiSOA propone el uso del lenguaje WDL para estas descripciones, sin embargo, este formato sería insuficiente si se requieren hacer inferencias semánticas sobre las interfaces de los servicios. Por ello se propone el uso de ontologías para describir estas interfaces, entre ellas se encuentran MSM y hREST.

La interfaz del servicio requiere describir: cada una de las operaciones de los recursos, las opciones de interacción, las entradas de datos y salidas de datos (ver Figura 31).

```
<hr:hasOperation>
  <hr:Operation rdf:about="http://skylab.local/persiana/.well-known/description#op1">
    <rdfs:label>obtenPosicion</rdfs:label>
    <hr:hasAddress rdf:datatype="http://iserve.kmi.open.ac.uk/ns/hrests#URITemplate">
  http://skylab.local/persiana/data</hr:hasAddress>
<msm:hasOutput>
  <msm:MessageContent rdf:about="http://skylab.local/persiana/.well-known/description#posicion">
 </msm:hasOutput>
    <hr:hasMethod rdf:resource="http://www.w3.org/2011/http-methods#GET"/>
  </hr:Operation>
</hr:hasOperation>

<hr:hasOperation>
  <hr:Operation rdf:about="http://skylab.local/persiana/.well-known/description#op2">
    <rdfs:label>establecerPosicion</rdfs:label>
    <hr:hasAddress rdf:datatype="http://iserve.kmi.open.ac.uk/ns/hrests#URITemplate">
  http://skylab.local/persiana/data?posicion={posicion}</hr:hasAddress>
      <msm:hasInput>
  <msm:MessageContent rdf:about="http://skylab.local/persiana/.well-known/description#posicion">
 </msm:hasInput>
    <hr:hasMethod rdf:resource="http://www.w3.org/2011/http-methods#POST"/>
  </hr:Operation>
</hr:hasOperation>

<msm:MessageContent rdf:about="http://skylab.local/persiana/.well-known/description#posicion">
<spt:minValue>0</spt:minValue>
<spt:value>{posicion}</spt:value>
<spt:maxValue>180</spt:maxValue>
</msm:MessageContent>
```

Figura 31 - Ejemplo de la descripción semántica de la interfaz de un servicio usando RDF/XML

Actualización del servicio: La descripción de un servicio debe mantenerse lo más pequeña posible, pero entre más completa sea la descripción del servicio, mejor interactividad tendrá con las aplicaciones. Por lo tanto se deben ofrecer mecanismos para actualizar la descripción semántica del servicio, aun cuando éste se encuentre en funcionamiento. Esto sería posible ofreciendo un recurso exclusivo para la actualización de la descripción semántica. La dirección y mecanismos de éste recurso quedan a consideración del fabricante o desarrollador del servicio, debido a que se deben tomar ciertas medias para no poner en riesgo la estabilidad del dispositivo.

Estas actualizaciones a la descripción deben ofrecer retro compatibilidad con versiones anteriores, con el fin de no perder interacción con aplicaciones hacen uso del mismo. En caso contrario el desarrollador debe considerar las consecuencias que tendría la modificación de ésta descripción.

Puesto que la descripción de un servicio es relativamente estándar para diversas instancias de las mismas plataformas de hardware, esta podría ser pre configurada por

el fabricante. Lo cual le daría a los fabricantes de dispositivos control sobre las actualizaciones y versiones de estás descripciones.

4.3.6. Interfaz a la red

Cada servicio debe ser capaz de conectarse por IP a cualquier dispositivo dentro de la red, creando una conexión entre extremos independientemente de los dispositivos intermedios. El componente de interfaz a la red debe ser capaz de soportar la pila de protocolos que se muestra en la Figura 32, la cual se describe a continuación.

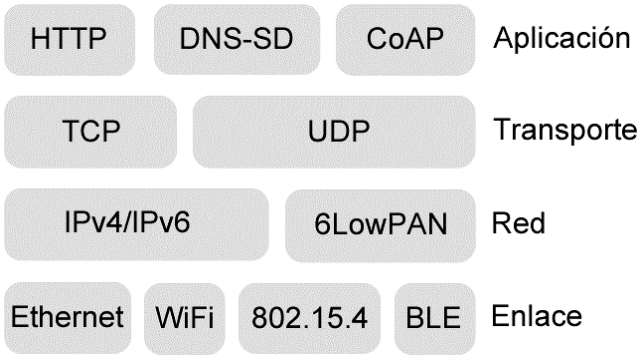


Figura 32 - Pila de protocolos básica para la plataforma OpenThings.

Capa de enlace: Tal como Ethernet y Wifi son de las tecnologías de comunicación más utilizadas para computadoras personales, en el área de dispositivos con capacidades limitadas ZigBee, 6LoWPAN/802.15.4, Z-Wave y BLE han ganado popularidad debido a su eficiencia en consumo de energía (Gomez, *et al.,* 2012). Estas tecnologías poseen diversas características: capacidad de formar redes de malla, conexiones ad-hoc, uso de energía, tasa de trasmisión, entre otras. Por lo tanto la selección de estas tecnologías debe darse de acuerdo a la aplicación. Sin embargo, la tecnología utilizada debe ser capaz de soportar las capas superiores de la pila.

Capa de red: La capa de red debe compartir el uso del protocolo IP para comunicar todos estos dispositivos. Diversas versiones de IP se encuentran actualmente en uso: por una parte, la transición entre IPv4 e IPv6 está aún en proceso en el Internet actual; por otra, el protocolo 6LowPAN se usa para dispositivos de recursos limitados. Sin embargo, existen diversas técnicas para enlazar estos protocolos y lograr que convivan a una misma red. Un ejemplo de ello son soluciones que ofrecen empresas

como IBM y Liberium, ellos usan puertas de enlace y tunelaje por IPv4 para conectar redes 6LowPAN con otros dispositivos en red (McLellan, 2013) (ver Figura 33).

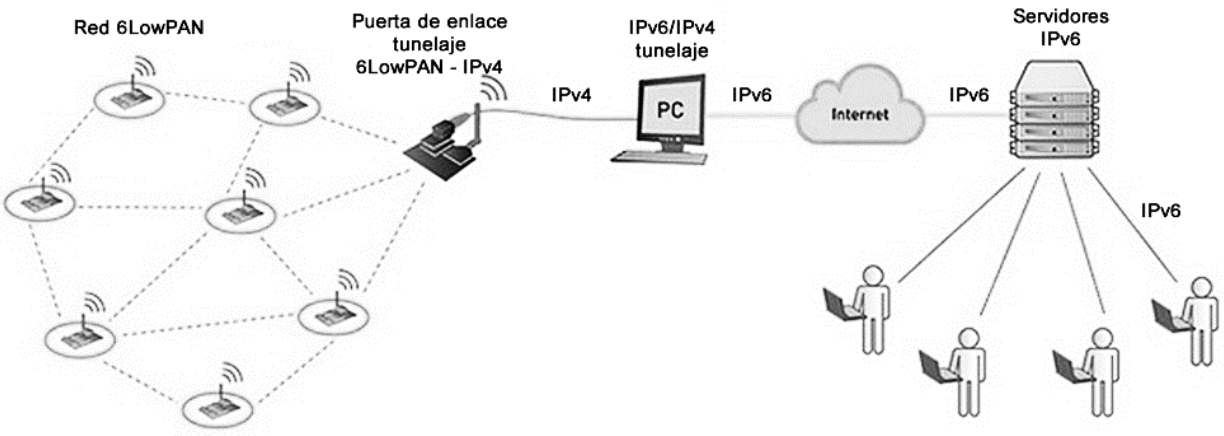


Figura 33 - Solución comercial de las empresas IBM y Liberium. El diagrama muestra la conexión de una red 6LowPAN/802.15.4 con una red IPv6 mediante tunelaje por IPv4.

Movilidad IP: La IP del servicio puede estar cambiando, así como la red local en la que se encuentra. Esto es debido a que muchos de los objetos en IoT son de naturaleza móvil. La movilidad en IPv4 requiere de protocolos y soluciones adicionales, en cambio IPv6 y 6LowPAN integran movilidad dentro del protocolo, lo cual facilitaría el uso de una sola dirección IP para ubicar los servicios. Sin embargo, dado que la plataforma propone el uso de cualquiera de los protocolos mencionados, en la sección de ambientes virtuales se describe una propuesta para resolver este problema.

Capa de transporte: Los protocolos de transporte TCP y UDP son necesarios para los protocolos de aplicación HTTP y CoAP respectivamente, pero los dispositivos no necesitan soportar ambos protocolos de transporte. La decisión de utilizar HTTP/TCP o CoAP/UDP será en función de su disponibilidad en soluciones de software y características de la red donde serán utilizados. DNS-SD es un protocolo de aplicación que funciona con base en el protocolo DNS sobre TCP, sin embargo, algunas implementaciones hacen uso del mDNS. Este último toma ventaja de las características multidifusión de UDP, por lo que el soporte de ambos protocolos de transporte sería lo ideal.

Capa de aplicación: HTTP es el protocolo clave para el modelo RESTful que sigue la arquitectura UbiSOA. Sin embargo, no es un protocolo pensado para la comunicación entre dispositivos con limitaciones de recursos. Para la arquitectura OpenThings se propone adicionalmente el uso del protocolo CoAP que es más apto para ambientes de recursos limitados.

CoAP es un protocolo estándar de la IETF (Shelby *et al.,* 2014), permite realizar todas las operaciones REST y añade la capacidad de subscripción a recursos. Este protocolo es compatible con el uso de HTTP y para comunicar ambos protocolos solo es necesario un proxy HTTP/CoAP. CoAP al igual que HTTP posee negociación de contenido, característica necesaria para la arquitectura propuesta y que otros protocolos de recursos limitados como MQTT no poseen (Eurotech - IBM, 2014).

Dado que el protocolo DNS-SD es vital para el descubrimiento de servicios, se debe asegurar que pueda ser implementado. Éste protocolo es principalmente usado dentro de red local, sin embargo, se debe tomar en cuenta que existen soluciones que permiten usarlo entre diversas sub redes. Estas soluciones pueden extender el alcance de descubrimiento de servicios (Korper, 2014). Del mismo modo el protocolo también posee capacidades para publicar los servicios a la red global, pero se deben tomar medidas extras de seguridad para activar estas funciones (Cheshire, 2014). Por lo tanto, para los servicios de la plataforma se recomienda que DNS-SD solamente sea usado en la red local.

4.4. Servicios virtuales

Los servicios virtuales ofrecen mecanismos de apoyo que extienden las capacidades de otros componentes de la arquitectura. Además comparten los mismos componentes internos de los servicios OpenThings mencionados anteriormente, con la excepción de que no encapsulan recursos. Por lo tanto los recursos serían remplazados por las funciones específicas del servicio virtual. Los servicios virtuales también deben poseer una descripción semántica de sus características e interfaces.

Dado que los servicios virtuales hacen uso de otros servicios, se debe describir semánticamente la información de los servicios utilizados, la frecuencia de uso, prioridad, etc., con el fin de dar conciencia al sistema de las relaciones entre cada uno de los componentes. Aunque no es obligatorio, es recomendable que los servicios virtuales registren está información dentro de los ambientes virtuales presentes en la red.

La configuración de los servicios virtuales deberá exponerse usando una interfaz REST y siguiendo los principios de Linked Data. Como ejemplo de servicios virtuales están aquéllos que permiten la adquisición de contexto o aquellos extienden la escalabilidad de otros servicios.

4.4.1. Servicios virtuales de escalabilidad

El servicio de un dispositivo de recursos limitados puede dar soporte a tantos clientes como sus capacidades lo permitan. Sin embargo, se pueden implementar servicios virtuales que funcionen como proxy para escalar las capacidades de servicios limitados que lo requieran. El uso de servicios virtuales para realizar el escalamiento brinda una mayor modularidad en el sistema. Los servicios que encapsulan objetos pueden estar definidos por el fabricante, pero el administrador de la red puede usar un servicio virtual con el modelo de escalamiento más adecuado a sus necesidades. Las soluciones de escalabilidad pueden variar dependiendo del tipo de interacción del servicio: sensores, actuadores o eventos.

Sensores: La escalabilidad de un sensor puede darse de dos formas: crear un servicio virtual con gran capacidad de almacenamiento que sirva como caché, o bien creando un servicio virtual de mayor procesamiento y comunicación que permita servir a tantos clientes como sean necesarios.

Actuadores: La escalabilidad de un actuador es completamente diferente a la de los sensores, dado que el recurso es un dispositivo que realiza una acción. La cantidad de clientes que puede servir un actuador está en función de limitantes como: la granularidad temporal con que puede realizar dicha acción, el número de usuarios que pueden utilizarlo, o la existencia de conflictos de uso del mismo (e.g., un aire

acondicionado no puede servir al mismo tiempo a dos usuarios con distintas preferencias de temperatura). Por lo tanto se pueden crear servicios virtuales que extiendan el control de actuadores para múltiples usuarios mediante modelos anticolisión, calendarización o solución de conflictos.

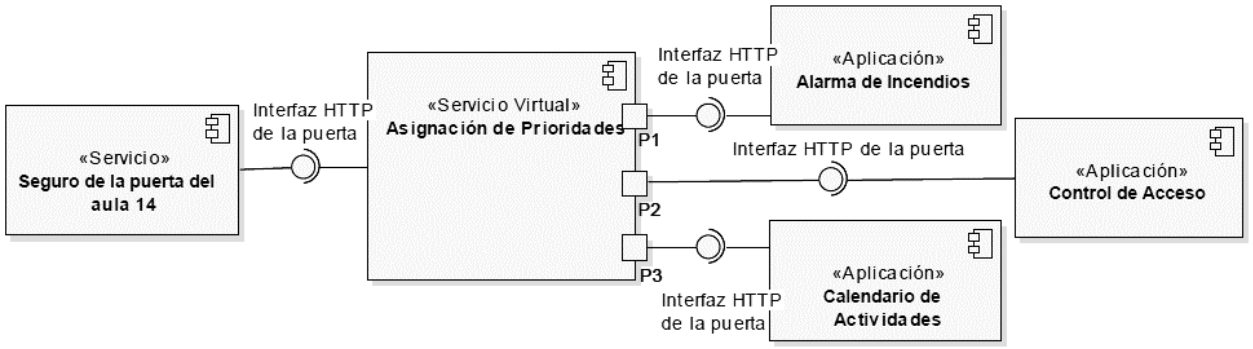


Figura 34 - Ejemplo de servicio virtual de escalamiento. Un servicio de asignación de prioridades mejora el comportamiento del actuador de una puerta ofreciendo interfaces con diversas prioridades a múltiples aplicaciones.

En la Figura 34 se describe un ejemplo de servicio virtual en donde la puerta de un aula posee un seguro controlado por un actuador. Éste actuador se puede utilizar por múltiples aplicaciones que pueden generar un conflicto de interés: una aplicación permite automatizar el acceso al abrir y cerrar automáticamente la puerta de acuerdo a un calendario de actividades; otra aplicación da control de la puerta a personal autorizado para abrir y cerrar el aula en cualquier momento sin importar lo establecido en el calendario; por último hay una aplicación de alarma de incendios que permite abrir todas las puertas y accesos del edificio en caso de incendio. Se debe crear un servicio virtual que sirva de proxy para administrar la interfaz de la puerta para establecer la prioridad que las aplicaciones tienen sobre el control de la puerta. Esto permite que el servicio de la puerta en si no requiera modificaciones para el uso por múltiples aplicaciones.

Eventos: La escalabilidad de eventos, al igual que los sensores, puede realizarse por servicios virtuales de caché que llevan registro de todos los eventos ocurridos. Pero para servir múltiples clientes se deben establecer modelos de distribución de contenido que permitan almacenar toda la lista de clientes interesados en los eventos. Un ejemplo de estos servicios es el mecanismo de notificación de eventos que usa UbiSOA. Este

mecanismo está fundamentado en PubSubHubBub[20], en el cual por medio de un hub los servicios pueden notificar a múltiples usuarios sobre la existencia de un nuevo evento.

PubSubHubBub tiene dos funciones principales, por un lado brinda el modelo de comunicación “Publisher/Subscriber” y por otro permite escalar la cantidad de clientes que reciben un evento. Éste mecanismo es particularmente útil cuando los servicios tienen conocimiento previo de la existencia del hub, pero disminuye la autonomía de los servicios que no deben depender de otros. Mecanismos como el “observador” de CoAP y el SSE de HTTP establecen el modelo “Publisher/Subscriber” en donde el cliente es el responsable por establecer la conexión. De esta forma el servicio queda completamente autónomo y la escalabilidad queda únicamente en los servicios virtuales.

4.4.2. Extensión de funciones

La modularidad de la arquitectura permite extender sus funciones por medio de nuevos servicios virtuales. Un servicio virtual para soporte MQTT sirve de ejemplo para mostrar cómo se puede extender la arquitectura y mejorar la interoperabilidad.

MQTT es un protocolo diseñado para dispositivos de recursos limitados, a diferencia de la arquitectura cliente-servidor de CoAP, éste usa distribución de mensajes, de tal forma que cuando un cliente publica un mensaje M de un tema T, todos los clientes subscritos al tema T recibirán el mensaje M. MQTT soporta diversos niveles de Calidad de Servicio (QoS), característica de la cual CoAP carece. Dichas diferencias sugieren que ambos protocolos deben ser soportados para el desarrollo de aplicaciones de IoT (Thangavel, *et al.,* 2014) (Sutaria and Govindachari, 2012).

Mientras que CoAP debe ser compatible con todas las interfaces de red en los servicios, ya sea directamente con un cliente CoAP o a través de un proxy CoAP/HTTP, algunas aplicaciones podrían hacer uso de mensajes MQTT. Hacer esto posible en la arquitectura OpenThings requiere de crear un servicio virtual que sirva de intermediario (ver Figura 35).

[20] https://code.google.com/p/pubsubhubbub/

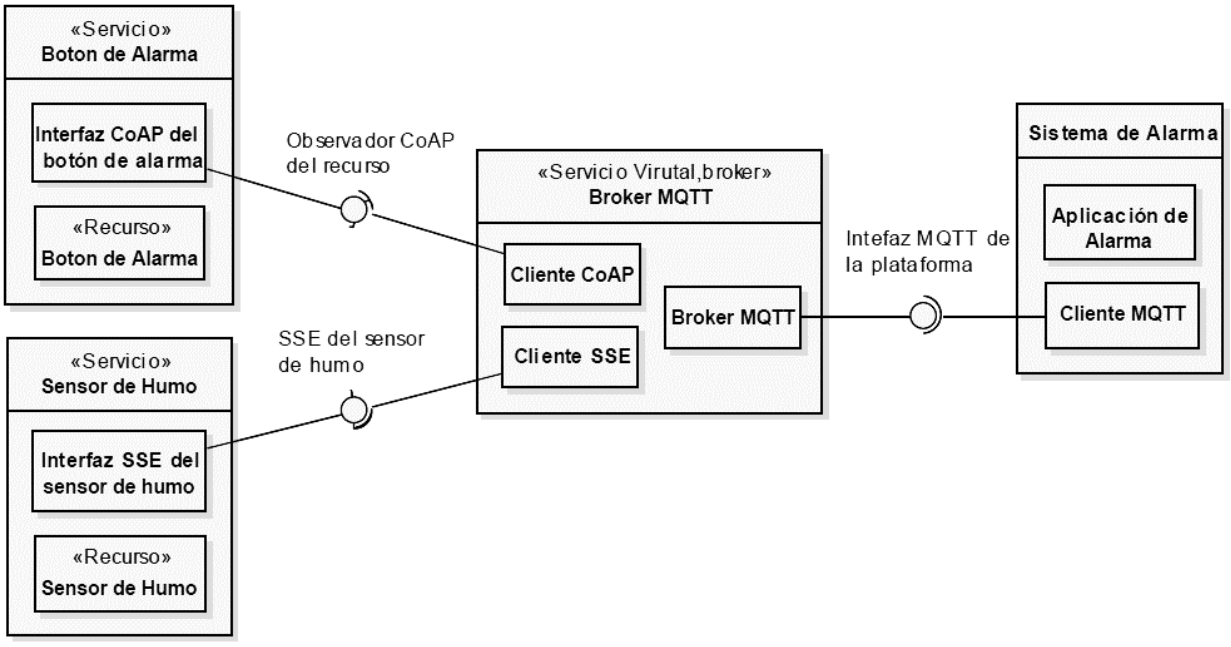


Figura 35 - Servicio Virtual MQTT. El servicio MQTT se subscribe a ambos servicios usando las características de "observador" de CoAP y SSE de HTTP, actuando como bróker para ofrecer subscripciones en formato MQTT a todas las aplicaciones que lo requieran.

4.4.3. Derivación de datos

Las limitaciones de procesamiento y memoria de dispositivos de IoT pueden restringir el uso de técnicas de análisis de datos. Análisis que resultaría de utilidad para diversas aplicaciones. Estas técnicas de análisis de datos se podrían encapsular en un servicio virtual dentro de un dispositivo con capacidades de cómputo superiores. Este dispositivo obtiene datos de dispositivos limitados, los procesa y expone los resultados como sus propios recursos. Estos análisis de datos podrían generar contexto útil para el sistema, el cual es considerado una pieza fundamental para un middleware de IoT (Perera, *et al.,* 2014). Por lo tanto, a continuación se propone un mecanismo que permitiría mejorar el uso de contexto por dispositivos de IoT.

4.5. Ambientes virtuales

Los ambientes virtuales son una instancia especial de un servicio virtual. Éstos representan sitios donde un grupo de servicios y aplicaciones comparten un espacio de

uso. Este espacio puede ser un sitio físico (e.g., Edifico de Telemática, CICESE), o un espacio conceptual (e.g., Sensores de Temperatura en México).

Estos ambientes poseen un espacio de almacenamiento ligado a un dominio de conocimiento en particular, con el fin de llevar un registro de la información contextual relacionada a los servicios y aplicaciones. Éste registro permite conocer el estado de un entorno de aplicación.

Un ambiente virtual que representa un espacio físico, como una “Sala de Conferencias”, debe llevar registro de todos los servicios que se encuentran en la red local de la sala, tal como: luces inteligentes, pantallas, proyectores, micrófonos etc. Además debe registrar servicios relacionados al sitio, por ejemplo: servicios de alarma del edificio que no necesariamente están dentro de la misma red local.

Un ambiente virtual que representa un espacio conceptual, como “Sensores de Temperatura en México”, puede ser utilizado para agrupar servicios con características similares. Esto permite crear grupos de servicios ligados a un dominio de aplicación en particular. Por ejemplo: un ambiente virtual de sensores sísmicos sería de utilidad para organizaciones que brindan apoyo para desastres naturales, ya que facilitaría la tarea de obtener información en tiempo real sobre sismos.

La información que se almacena en un ambiente virtual se divide en tres partes: descripción del ambiente, registro de servicios, información contextual.

4.5.1. Descripción del servicio

Al ser un servicio virtual, debe poseer una descripción semántica en donde se describe el ambiente virtual y las relaciones que puede tener con otros conceptos en la red (Figura 36). Además de las relaciones con otros ambientes virtuales, las cuales permitirían navegar semánticamente entre diversos ambientes para el descubrimiento de dispositivos remotos.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:spt="http://spitfire-project.eu/ontology/ns/"
xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#"
xmlns:opt="http://iserve.kmi.open.ac.uk/ns/opt#">

    <hr:Service rdf:about="http://cicese.mx/contexto/">
        <rdf:type rdf:resource="https://cicese.mx/openthings.owl#VirtualEnviroment"/>
        <rdfs:label>Ambiente Virtual de CICESE</rdfs:label>
        <ssn:isDescribedBy rdf:resource="http://dbpedia.org/resource/CICESE"/>
        <spt:nerby rdf:resource="http://es-la.dbpedia.org/resource/UABC_Campus_Ensenada"/>
        <spt:owns rdf:resource="http://cienciascomp.cicese.mx/contexto"
        <spt:owns rdf:resource="http://dfa.cicese.mx/contexto"
    </hr:Service>
</rdf:RDF>
```

Figura 36 - Descripción semántica de un ambiente virtual. En él que se indica que el servicio representa a CICESE. Además indica que existen dos ambientes relacionados: Ciencias de la Computación y Física Aplicada.

4.5.2. Registro de servicios

El ambiente virtual debe tener una lista de servicios y aplicaciones que poseen alguna relación con el espacio que representa, para ello se describe semánticamente cada uno de ellos y sus relaciones. Por ejemplo: los servicios virtuales deben indicar las dependencias que tienen con otros servicios y otra información que describe su comportamiento dentro de red. Por lo tanto los ambientes virtuales sirven como un directorio de los servicios presentes en ese espacio.

Para lograr su objetivo, un ambiente virtual tiene la responsabilidad de buscar las aplicaciones que pueden estar relacionadas a él, ya sea usando el descubrimiento de servicios en la red local o algún otro mecanismo de búsqueda semántica.

El ambiente virtual puede brindar una imagen completa del estado actual de la red al tener un registro de las interacciones que existen entre servicios. Dicha imagen puede servir para evitar colisiones, o para evitar que se desconecte un servicio al creer que nadie lo está utilizando. En la Figura 37 se puede visualizar un ejemplo del ambiente virtual de un Aula.

```
<hr:Service rdf:about="http://cicese.mx/aula12/contexto/registro">
    <rdfs:label>Registro de Servicios</rdfs:label>
    <ssn:isDescribedBy rdf:resource="http://cicese.mx/contexto/.well-known/description"/>
    <hr:includesObject>
        <hr:Service rdf:about="http://cicese.mx/aula12/persiana">
            <rdfs:label>Persiana Inteligente</rdfs:label>
            <spt:status rdf:resource="hhttp://iserve.kmi.open.ac.uk/ns/opt#online">
            <ssn:hasOutput rdf:resource="http://cicese.mx/aula12/proxypersiana">
        </hr:Service>
        <hr:Service rdf:about="http://cicese.mx/aula12/sensorluz">
            <rdfs:label>Sensor de Luz</rdfs:label>
            <spt:status rdf:resource="hhttp://iserve.kmi.open.ac.uk/ns/opt#offline">
            <ssn:hasOutput rdf:resource="http://cicese.mx/aula12/controldeluz">
        </hr:Service>
        <hr:Service rdf:about="http://cicese.mx/aula12/proxypersiana">
            <rdfs:label>Control de Escalabilidad de Persiana</rdfs:label>
            <spt:status rdf:resource="hhttp://iserve.kmi.open.ac.uk/ns/opt#online">
            <ssn:hasInput rdf:resource="http://cicese.mx/aula12/persiana">
            <ssn:isProxyFor rdf:resoruce="http://cicese.mx/aula12/persiana">
            <ssn:hasOutput rdf:resource="http://cicese.mx/aula12/controldeluz">
        </hr:Service>
        <hr:Service rdf:about="http://cicese.mx/aula12/controldeluz">
            <rdfs:label>Control de Iluminación</rdfs:label>
            <spt:status rdf:resource="hhttp://iserve.kmi.open.ac.uk/ns/opt#offline">
            <ssn:hasInput rdf:resource="http://cicese.mx/aula12/proxypersiana">
            <ssn:hasInput rdf:resource="http://cicese.mx/aula12/luz">
        </hr:Service>
    </hr:includesObject>
</hr:Service>
```

Figura 37 - Ejemplo en RDF/XML del registro de servicios en un ambiente virtual. En la descripción semántica se puede observar que el servicio control de iluminación esta fuera de línea debido a que depende de un sensor de luz que tampoco está operativo.

Para que éste registro sea posible, idealmente todos los servicios virtuales presentes en la red deben enviar la descripción semántica de sus interacciones a los ambientes virtuales. No obstante, dado que no es obligatorio, otra posibilidad para actualizar el uso de recursos sería mediante servicios virtuales similares a los ejemplos de escalamiento. Estos servicios virtuales que actúan como proxy pueden actualizar los ambientes virtuales cada vez que una aplicación hace uso de algún recurso.

La búsqueda en la red local (ver Figura 38) puede realizarse directamente usando el descubrimiento de servicios. Cada que un servicio nuevo es localizado, se puede solicitar su descripción semántica a la dirección bien conocida (.well-known/description) para anexar ésta descripción al registro del ambiente virtual. Esto permitirá agilizar la búsqueda de servicios al tener un directorio semántico de los objetos relacionados con un espacio en particular.

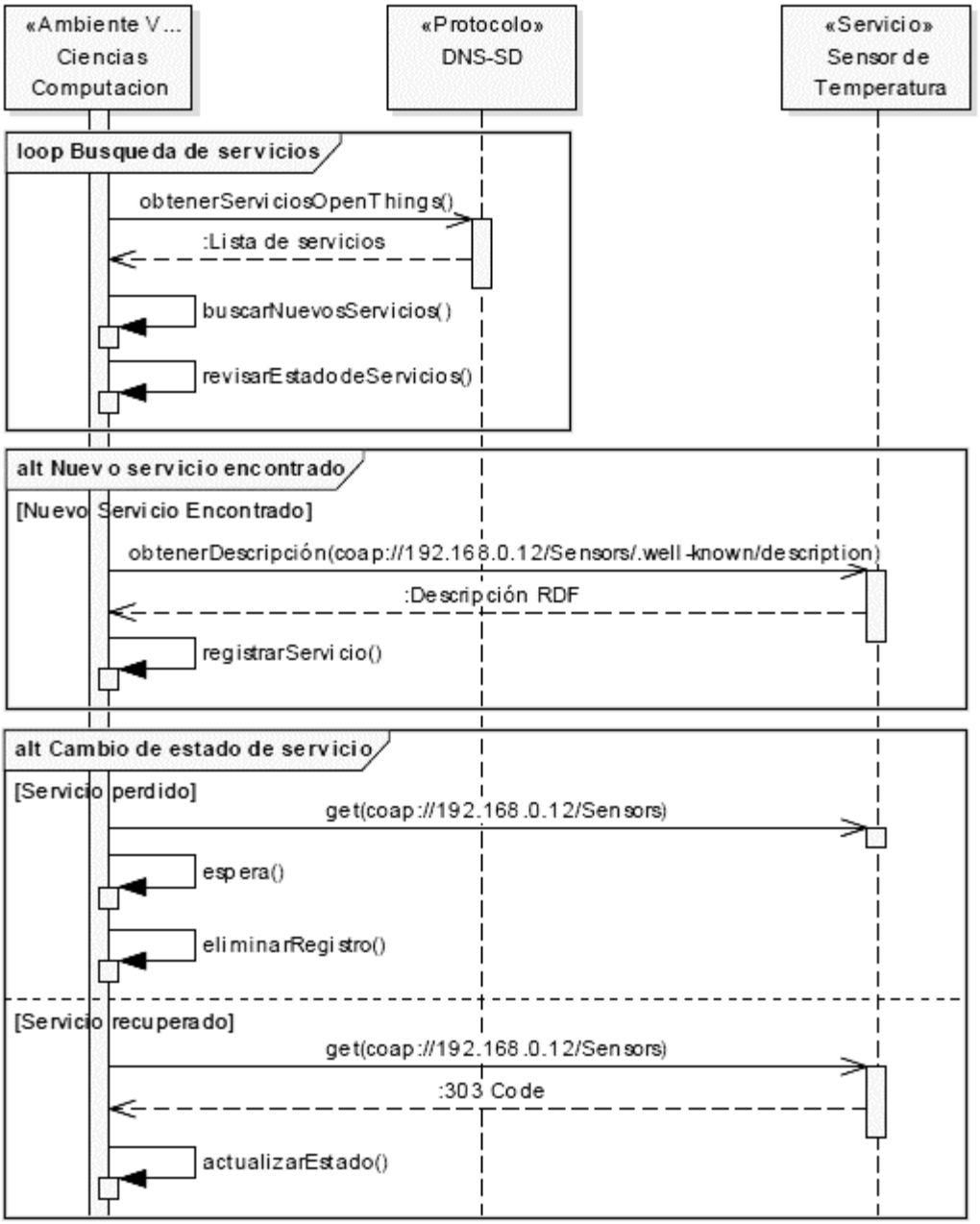


Figura 38 – Ambiente virtual buscando servicios en la red local y revisando su disponibilidad.

Esta búsqueda de servicios permite crear fácilmente un sistema de monitoreo para verificar el estado de los servicios en la red. Al realizar una petición al dispositivo, la respuesta verifica la disponibilidad del dispositivo, mientras que la ausencia de respuesta puede significar la pérdida de un servicio después de un determinado tiempo de espera.

La frecuencia con la que se revisa el estado de los servicios estará en función de la confianza que se desee tener en el estado de la red. Hay que tomar en cuenta que

entre más rápida sea la frecuencia de revisión de estado, mayor consumo de energía tendrán los servicios consultados.

Al realizar la búsqueda en la red local por medio del descubrimiento de servicios se obtiene la dirección local de los servicios. Esta dirección local permite localizar al servicio dentro de la misma red, pero se debe tener una dirección IP externa para que aplicaciones en una red externa puedan localizar el servicio.

La dirección externa de un servicio y su dirección en red local se pueden enlazar usando una relación semántica (ver Figura 39). Está relación debe ser anexada por el administrador de la red al registro del ambiente virtual (ver Figura 40).

```
1  <rdf:resource rdf:about="http://205.24.4.3/lampara">
2    <owl:sameAs rdf:resource="http://cicese.mx/sensores/aula13/lampara"/>
3  </rdf:resource>
```

Figura 39 - Codigo RDF/XML para enlazar una dirección externa a un recurso local.

Para anexar la dirección externa se pueden asignar diferentes puertos de una sola dirección IP a cada uno de los servicios locales (e.g., http://205.24.4.3:8020 y http://205.24.4.3:8021). Otra forma de realizarlo sería asignando diferentes rutas a cada uno de los servicios (e.g., http://205.24.4.3/lampara y http://205.24.4.3/sensorluz). Además se puede hacer uso de DNS para usar un dominio en particular, siendo esta la opción la más recomendada para publicar servicios de una forma más legible (e.g., http://cicese.mx/aula12/lampara y http://cicese.mx/aula12/sensorluz). Para facilitar la asignación de direcciones externas, esto se puede automatizar usando un virtual.

Esta asignación de direcciones también se puede usar para mejorar las capacidades de movilidad de los servicios, por ejemplo: un teléfono inteligente puede tener un grupo de servicios usados por aplicaciones que están en una casa, pero si el teléfono inteligente se mueve a otra red las aplicaciones perderán conexión con los servicios del teléfono; sin embargo, un servicio virtual dentro del teléfono inteligente puede actualizar el registro del ambiente virtual de la casa con la dirección que actualmente tiene el teléfono inteligente; esto permitiría a las aplicaciones seguir usando los servicios del teléfono pese al cambio de redes.

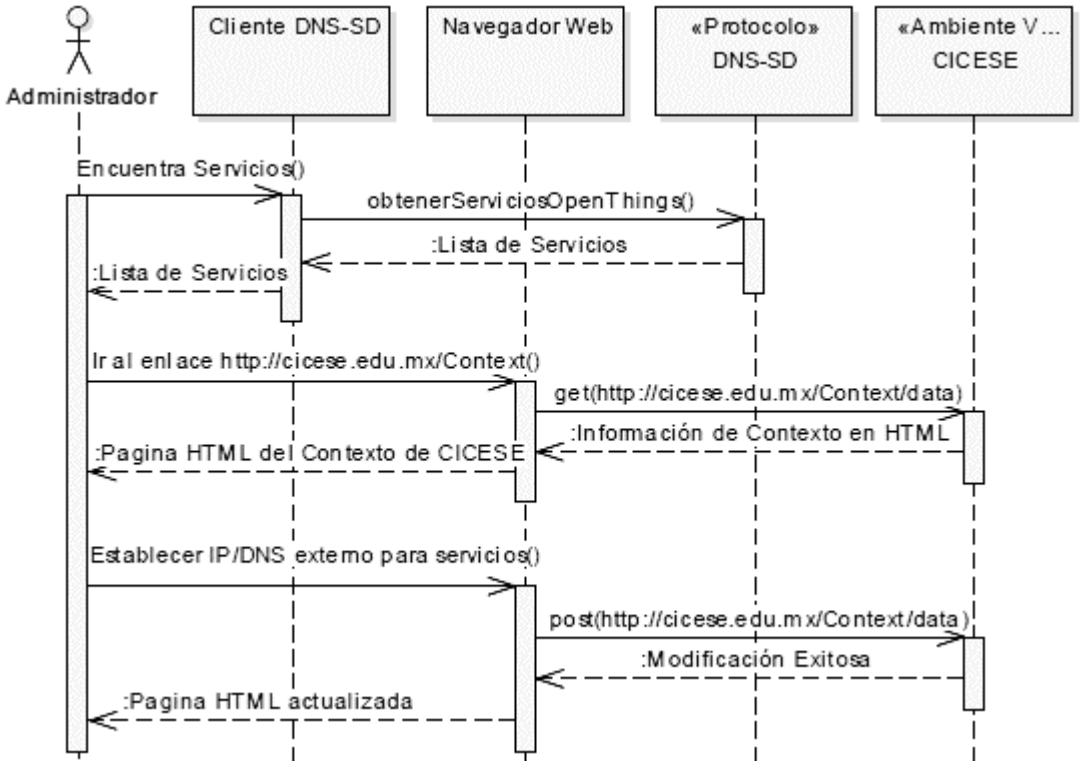


Figura 40 - Diagrama de secuencia para asignar direcciones externas. Un administrador de red puede usar un navegador web y un cliente DNS-SD para registrar la IP Externa de servicios en el servicio de contexto.

4.5.3. Información contextual

La información contextual relacionada a los objetos requiere un servicio adicional para su almacenamiento, debido que la descripción semántica de los servicios OpenThings está definida como la mínima información para su funcionamiento. Aunque ésta información se podría almacenar en cualquier servicio virtual, los ambientes virtuales son sitios ideales para esta información debido a que ya poseen el registro de todos los servicios presentes en la red.

Se puede almacenar información contextual del objeto en sí mismo (e.g., Arduino12 es propiedad del Dpto. de Cómputo Ubicuo) o puede ser derivado de datos de un sensor (e.g., El Aula12 está vacía). Para el contexto derivado de datos se pueden crear servicios virtuales que estén analizando los datos obtenidos de los recursos y actualizando la información disponible dentro del ambiente virtual (ver Figura 41).

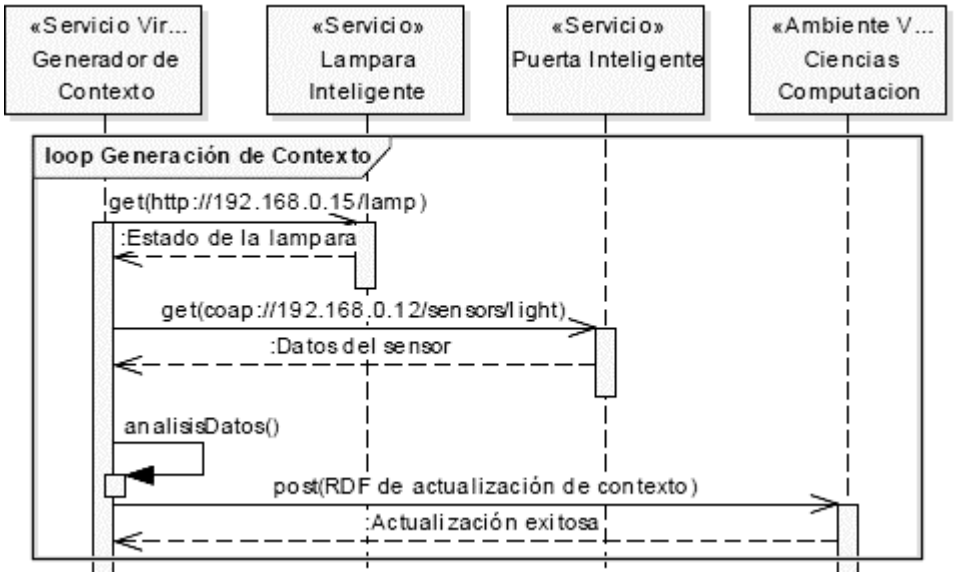


Figura 41 – Diagrama de generación de contexto. Un servicio virtual analiza datos de diversos servicios para generar contexto secundario y almacenarlo en el ambiente virtual.

La información contextual que se almacena en los ambientes virtuales debe estar en función del dominio del conocimiento que es de interés particular de ese ambiente. Por ejemplo: una caja de galletas en el ambiente virtual de una tienda se podría describir en función de su precio y ubicación en la tienda; en cambio, la misma caja de galletas en el ambiente virtual de una casa se podría describir en función de calorías y recetas de cocina. Por lo tanto la ubicación de la información contextual debe ser determinada por los desarrolladores de servicios en función de su utilidad.

Se debe considerar almacenar información contextual dentro de la descripción del servicio del dispositivo si se quisiera ligar permanentemente al dispositivo. En cambio almacenar información contextual dentro de un ambiente virtual ligaría la información a ese sitio. Ambas opciones son de utilidad en diferentes situaciones.

En la Figura 42 se describe semánticamente que el Aula 12 es un sitio donde se encuentran ubicados los objetos “persiana” y “lámpara”. Además se describe que esos objetos son propiedad de una persona, de la cual se da información de contacto.

```
<ssn:location rdf:about="http://cicese.mx/aula12">
    <rdfs:label>Aula12</rdfs:label>
    <spt:value rdf:resource="http://iserve.kmi.open.ac.uk/ns/opt#empty">
    <rdfs:nerby rdf:resource="http://cicese.mx/aula11"/>
    <rdfs:nerby rdf:resource="http://cicese.mx/aula10"/>
    <ssn:isLocationOf rdf:resource="http://cicese.mx/aula12/lampara"/>
    <ssn:isLocationOf rdf:resource="http://cicese.mx/aula12/persiana"/>
</ssn:location>

<foaf:person rdf:about="http://cicese.mx/resources/usr#jagm">
    <foaf:mbox>jagm@cicese.mx</foaf:mbox>
    <foaf:homepage>http://usuario.cicese.mx/~jagm</foaf:homepage>
    <spt:owns rdf:resource="http://cicese.mx/aula12/lampara"/>
    <spt:owns rdf:resource="http://cicese.mx/aula12/persiana"/>
</foaf:person>
```

Figura 42 - Ejemplo RDF/XML de información contextual dentro de un ambiente virtual.

4.6. Aplicaciones

Cualquier servicio de la plataforma se puede utilizar por medio de un cliente HTTP o CoAP. Dada la naturaleza sin estado de las Interfaces REST, no se puede tener un control estricto de las aplicaciones que hacen uso de los servicios. Aun así las aplicaciones también pueden beneficiarse de poseer componentes de la plataforma como: descubrimiento de servicios, descripciones semánticas, información contextual, entre otras. Por lo tanto es deseable que las aplicaciones también sean implementadas siguiendo los principios de la arquitectura OpenThings. En tal caso las aplicaciones serían componentes derivados de los Servicios Virtuales.

Las aplicaciones son en principio agentes de software que actúan en nombre del usuario para realizar llevar a cabo una actividad. Igual que un servicio virtual, las aplicaciones deben poseer todos los componentes descritos en la sección de Servicios OpenThings.

4.6.1. Descubrimiento de servicios

Las aplicaciones tienen dos opciones para encontrar los servicios que requieren para su funcionamiento: usar descubrimiento en la red local por medio de DNS-SD o seguir enlaces ubicados dentro de los registros de ambientes virtuales. Por ejemplo: Si una aplicación busca sensores cercanos, lo más adecuado sería realizar la búsqueda dentro

de la red local, pero si la aplicación no tiene acceso a la red local deberá buscar en los ambientes virtuales.

El acceso de los servicios queda limitado a las configuraciones de la red, de tal forma que el acceso a la red brindaría acceso a todos los servicios presentes. La configuración de red podría funcionar como una especie de filtro en donde una aplicación solo tiene visibles los servicios locales de interés.

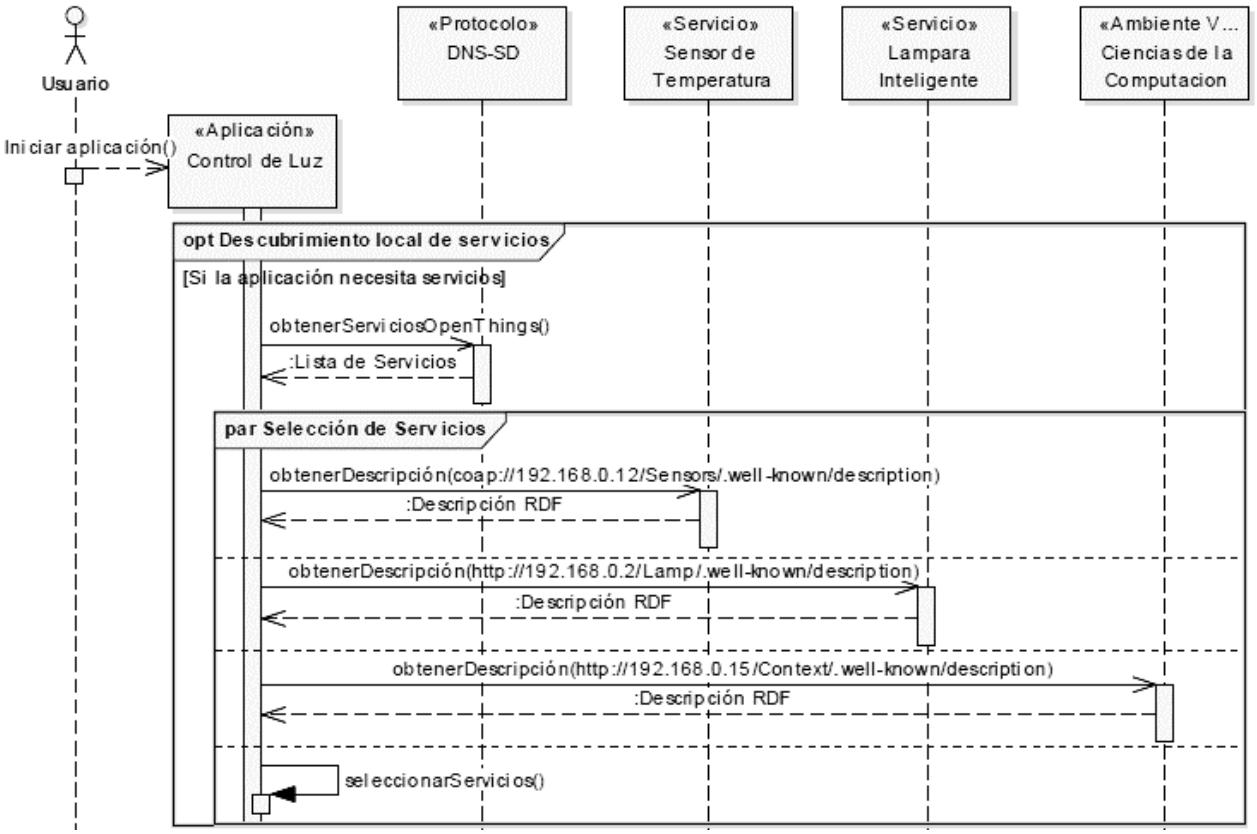


Figura 43 - Descubrimiento local de servicios. Una aplicación que requiere servicios realiza una búsqueda por DNS-SD para encontrar los servicios adecuados.

En la Figura 43 se muestra el ejemplo de una aplicación que realiza una búsqueda por DNS-SD para obtener la lista de servicios disponibles en la red. Esto le permite a una aplicación obtener los enlaces para solicitar la descripción semántica de cada uno de ellos usando la dirección bien conocida. Una vez obtenidas las descripciones de los servicios, la aplicación puede seleccionar los servicios que son de utilidad.

El descubrimiento externo o remoto se realiza por medio de enlaces, lo cual implica que los servicios buscados deben poseer una dirección de red externa. La aplicación usa los ambientes virtuales como directorios para buscar los servicios que requiere, así como otros directorios disponibles. En tal caso la aplicación puede usar una araña web (i.e., web crawler en inglés) para navegar automáticamente por los enlaces disponibles en documentos HTML de los ambientes virtuales.

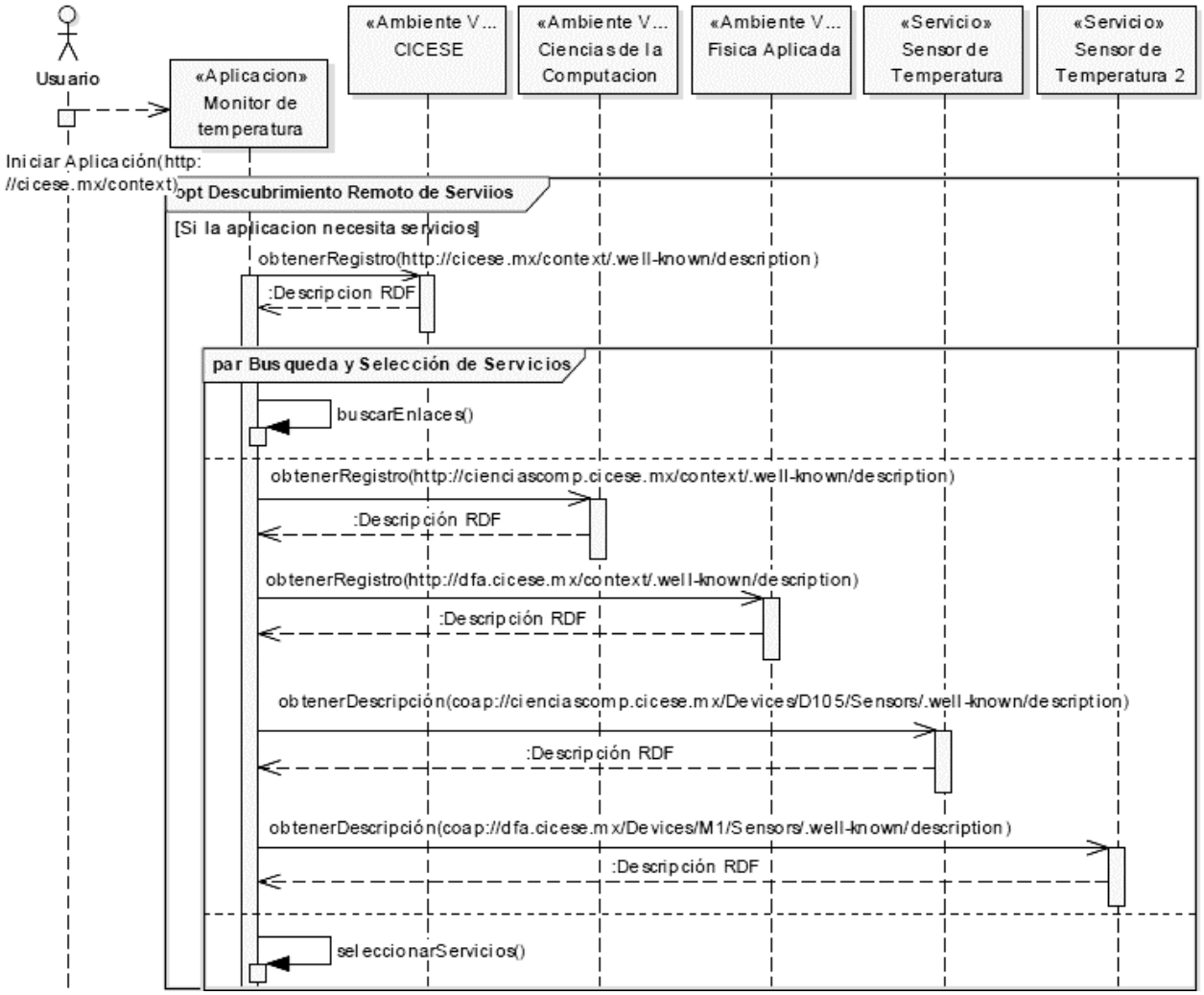


Figura 44 - Descubrimiento Remoto de Servicios. Una aplicación que requiere servicios realiza una búsqueda remota de servicios usando una araña web.

Para realizar el descubrimiento remoto, la aplicación debe partir de un enlace de inicio, idealmente la dirección de un ambiente virtual. En el ejemplo de la Figura 44, al ubicar un ambiente virtual, esté brinda enlaces a otros ambientes relacionados y así sucesivamente. Para evitar que las búsquedas se extiendan por toda la red se deben

establecer límites de búsqueda como: tiempo de vida, número de saltos o servicios visitados.

4.6.2. Motores de inferencias

La búsqueda de etiquetas claves en documentos HTML, XML, etc., permite la selección de servicios que son de utilidad para la aplicación. Sin embargo, las capacidades de búsqueda pueden verse ampliamente mejoradas con el uso de las descripciones semánticas de los servicios. Al usar éstas descripciones en conjunto con ontologías, una aplicación sería capaz de inferir información sobre la información disponible.

Por ejemplo, una aplicación puede estar buscando un dispositivo para reproducir audio dentro de una habitación, para seleccionar los servicios que le son de utilidad, la aplicación debe definir dos conceptos: cuál es la habitación que está buscando, y que es exactamente un dispositivo de reproducción de audio.

Para cumplir con el primer requisito del ejemplo anterior, la aplicación debe realizar una búsqueda de los ambientes virtuales que sean de interés. Mediante la relación semántica “http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation” se pueden encontrar las ubicaciones que el ambiente virtual representa. Si existe un ambiente virtual de la habitación deseada, su registro indicará la dirección de todos los servicios disponibles dentro de esa habitación. Sin embargo, si no existe un ambiente virtual que represente la ubicación deseada, se puede extender la búsqueda a la información contextual para buscar individualmente servicios que puedan estar dentro de la habitación deseada.

Para el segundo requisito del ejemplo se debe definir lo que es un dispositivo de reproducción de audio. La aplicación debe seleccionar un término de una ontología o un esquema RDFS que cumpla con la definición (e.g., http://example.com/devices #Loudspeaker). Para que la aplicación pueda seleccionar el servicio correcto, debe buscar servicios que coincidan con ese término. Si se tiene una ontología como la que se muestra en la Figura 45 se puede realizar una inferencia determinando que los servicios que se buscan pueden estar descritos como: speaker (boina, en inglés), headphone (auricular, en inglés), bocina, earphones (audífono, en inglés), audífono, etc. Esto brinda

flexibilidad para describir los servicios, ya que se pueden usar diversos términos para describir un objeto, incluso en diferentes lenguajes. La limitante sería que estos términos deben estar relacionados en alguna ontología o esquema RDFS que se encuentre disponible.

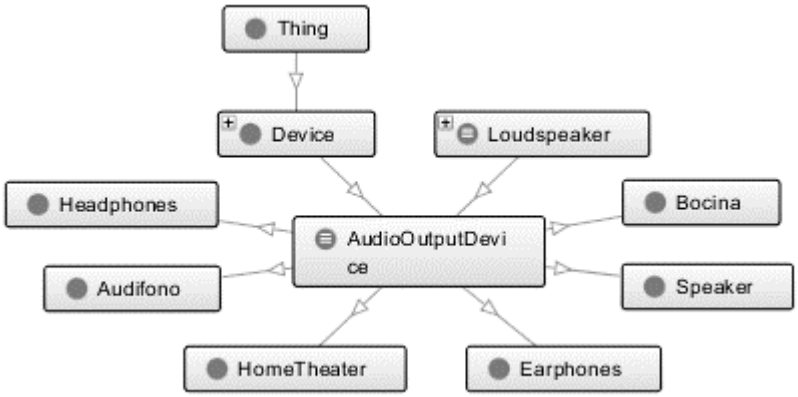


Figura 45 - Grafo de una ontología de dispositivos de audio modelada en Protegé[21].

4.6.3. Registro de aplicaciones

Los ambientes virtuales deben estar conscientes de todos los servicios presentes en la red, sin embargo, solo la aplicación posee Información precisa de los servicios que utiliza y la frecuencia en que se realizarán consultas. Por lo tanto las aplicaciones deben dar a conocer ésta información a los ambientes virtuales de interés (ver Figura 46). En caso de que la aplicación se registre y desaparezca de la red (i.e., el ambiente virtual no pueda comprobar su estado), el ambiente virtual debe emplear un método de recolección de basura para dar de baja los registros de aplicaciones que ya no están disponibles.

No se define un mecanismo que obligue a las aplicaciones a registrarse, pero hacerlo es de interés de la aplicación ya que la existencia de un registro permite conocer el uso de los dispositivos en la red. Esto sirve para identificar las aplicaciones que usan determinados servicios. Por ejemplo: Si un actuador está siendo completamente controlado por una aplicación, otras aplicaciones no podrán utilizarlo. Dar conciencia a las aplicaciones del uso de los dispositivos podría evitar el envío de peticiones que seguramente serían denegadas.

[21] http://protege.stanford.edu/

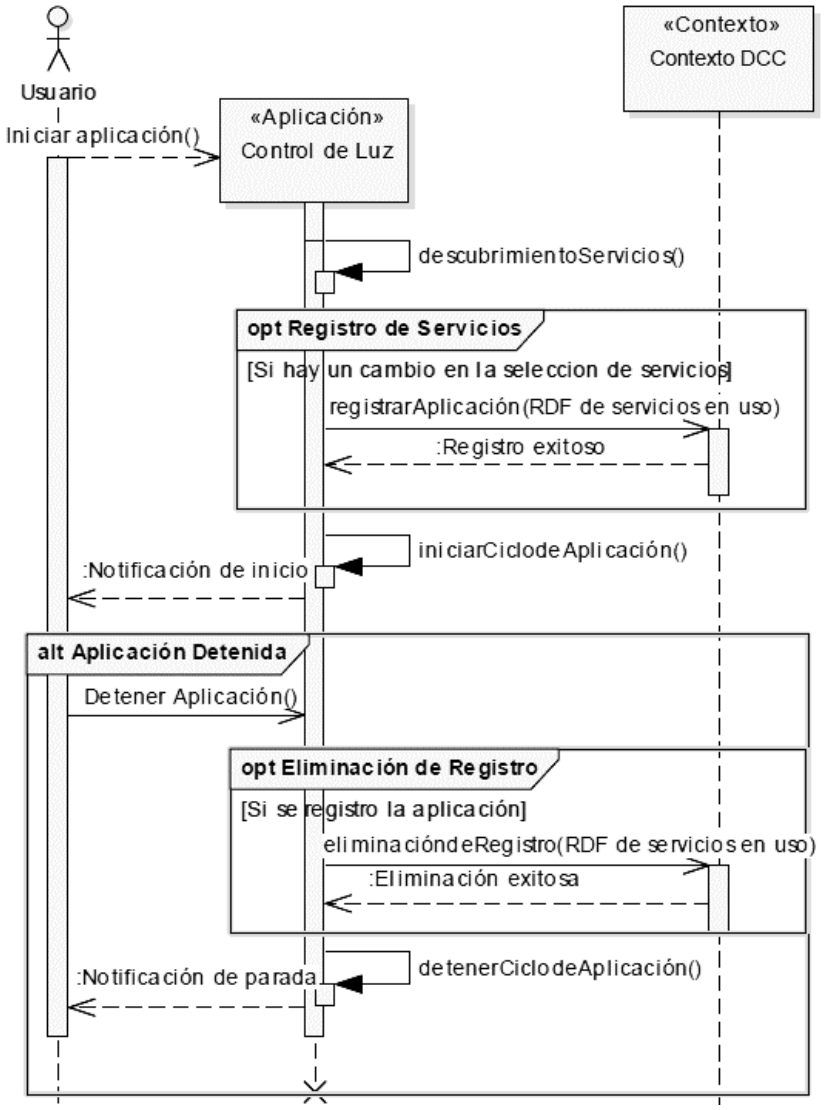


Figura 46 - Ciclo de vida de una aplicación. Al inicio y parada de una aplicación ésta es la responsable de informar a los ambientes virtuales de los servicios que usará.

4.6.4. Servicio virtual SPARQL

La naturaleza distribuida de la arquitectura propuesta implica que la información sobre la plataforma este dispersa en diversos documentos. Sin embargo, toda la información que posee una representación RDF puede ser acumulada por un Servicio Virtual con el uso de una araña web. Este servicio podría obtener las descripciones semánticas de cada uno de los componentes presentes en la plataforma.

Éste servicio virtual con información sobre la plataforma puede hacer uso de un punto de acceso SPARQL. Este sirve como interfaz de consulta para un grupo de servicios de la plataforma (ver Figura 47). Tomando en cuenta los límites de dominio, el

administrador de la red establecerá el alcance que pueda tener la recuperación de datos de cada punto de acceso. Por ejemplo: un servicio de acceso SPARQL por todos los servicios dentro de una red local brindaría una interfaz de consulta única para un gran grupo de servicios, pero también un único punto de falla. El uso de múltiples servicios de consulta SPARQL (e.g., uno por cada ambiente virtual) brindaría un sistema más robusto pero se tendrían múltiples puntos de acceso con limitada capacidad de consulta.

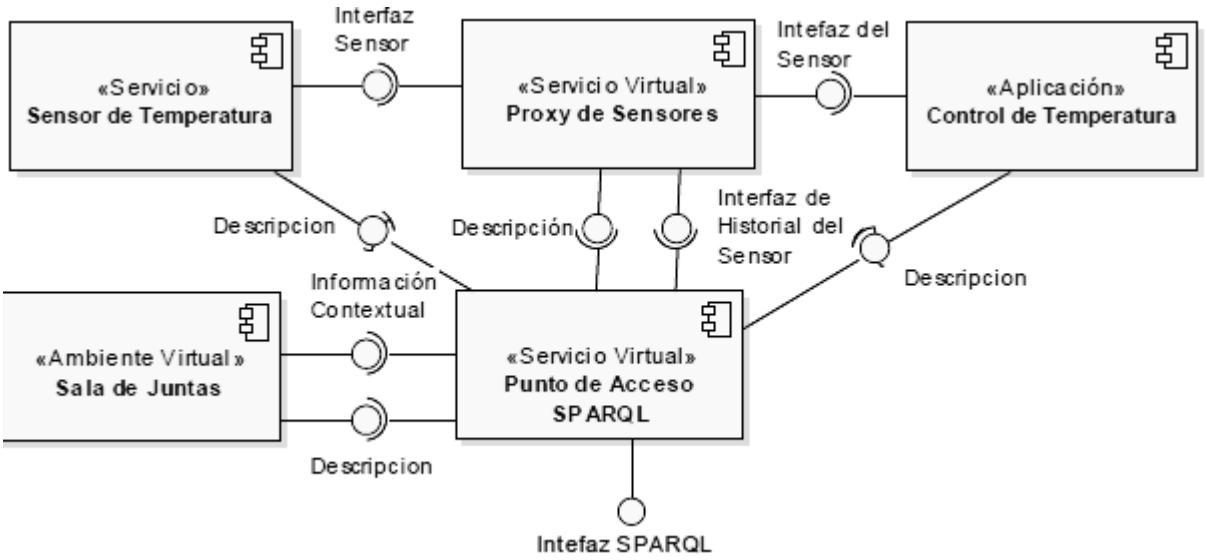


Figura 47 - Servicio Virtual SPARQL. Este servicio obtiene toda la información presente en la plataforma para brindar un servicio de consulta mediante el lenguaje SPARQL.

4.6.5. Movilidad de aplicaciones

El creador de la aplicación puede decidir si la aplicación es estática o dinámica. Si es estática puede estar localizada en un servidor y mantenerse activa con un grupo de servicios bien definidos. Por otra parte la aplicación puede ser dinámica, residiendo dentro de un teléfono inteligente y activándose solo cuando se encuentran los servicios necesarios a su alrededor.

En la Figura 48 se describe una aplicación de alarma de gluten en un ambiente móvil: la aplicación de alarma identifica un servicio de lectura QR en el dispositivo y se subscribe mediante SSE. Dado que la aplicación funciona completamente en el dispositivo móvil, ésta se registra en el ambiente virtual que representa el teléfono. El usuario usa el teléfono para leer el código QR, entonces el servicio del lector obtiene y retrasmite la URL del servicio que almacena la información RDF de ese objeto. La

aplicación realiza la solicitud al servicio de etiquetas y analiza los datos recibidos en búsqueda de contenido de gluten, finalmente se notifica al usuario del resultado.

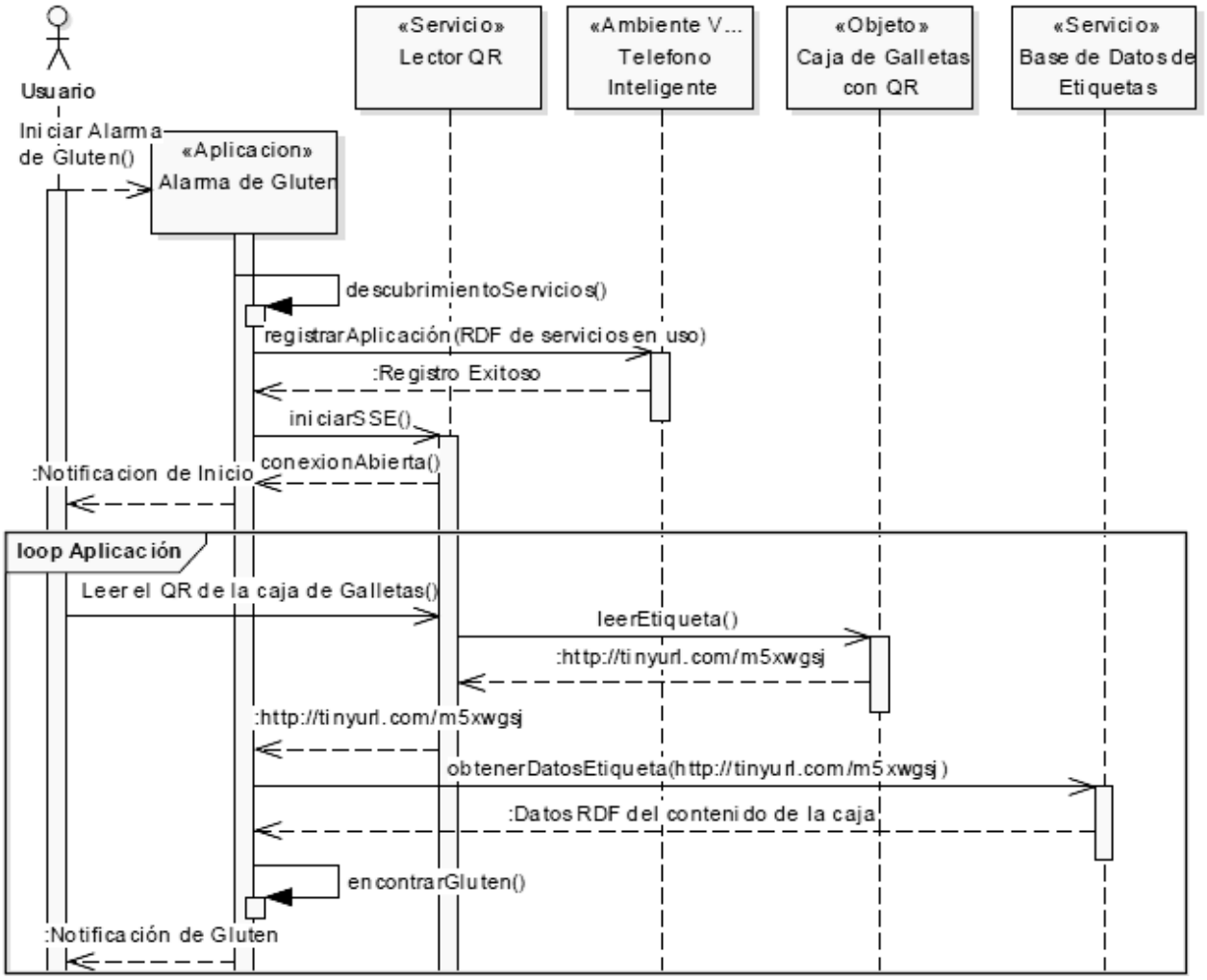


Figura 48 - Ejemplo de aplicación móvil. Una alarma de Gluten dentro de un ambiente móvil, se activa al encontrar un lector de etiquetas en el entorno.

En este ejemplo se puede notar que la lectura de un servicio simple (i.e., objeto con código QR) involucra dos servicios: El lector, en el cual sus recursos son los eventos de lectura; y la base de datos de etiquetas, en donde su recurso es la información de las etiquetas. Esta división permite el escalamiento de ambos servicios por separado pese a sus diferentes tipos de interacción.

4.7. Conclusiones

En este capítulo se abordó la lista de requerimientos obtenidos de la literatura para la creación de plataformas de IoT. Además se describe la arquitectura de software llamada OpenThings propuesta en este trabajo de investigación, la cual se desarrolló con base en los requerimientos dados.

Capítulo 5 Implementación y análisis

La descripción de la arquitectura de software posibilita crear aplicaciones para la plataforma mediante cualquier lenguaje de programación. No obstante, para facilitar la creación de aplicaciones con una estructura específica se recomienda el uso de un marco de trabajo.

En este capítulo se describe el desarrollo de un marco de trabajo y la implementación de un par de aplicaciones para verificar que es posible desarrollar aplicaciones de IoT siguiendo la arquitectura de software propuesta. Dicha implementación permitió analizar la factibilidad técnica del uso de las tecnologías seleccionadas en la arquitectura, así como identificar algunas limitantes de la plataforma.

Adicionalmente se hizo una revisión de los requerimientos planteados para verificar que la plataforma propuesta los cumple. Finalmente, se hizo un análisis de diferencias con otras plataformas en el estado del arte mediante la lista de requerimientos obtenidos.

5.1. Marco de trabajo

El marco de trabajo, al igual que la arquitectura de software, se extendió principalmente del marco de trabajo de UbiSOA, el cual está escrito en el lenguaje Java para facilitar el uso multiplataforma. Además se creó un pequeño marco de trabajo para implementar servicios en objetos inteligentes usando CoAP.

5.1.1. Extensión de UbiSOA

Al marco de trabajo de UbiSOA se le hicieron las siguientes modificaciones:

- Se integró el descubrimiento de servicios usando las librerías Java de JmDNS[22], debido a que el descubrimiento previamente utilizado (Bonjour[23]) requería librerías nativas. Esto se realizó principalmente para facilitar el uso del marco de trabajo en ambientes Linux.

- Se integró un selector para asignar las direcciones en las que se ubican los servicios en ejecución, con el fin de facilitar la ejecución de diversos servicios en un solo sistema.

- Para seguir los principios de Cool URI sé establecieron los mecanismos de re-direccionamiento para diferenciar entre: la dirección que identifica el servicio, los documentos de sus representaciones y la descripción semántica del servicio. Además se habilito la negociación de contenido estándar de HTTP.

- Se establecieron las estructuras que deben poseer: los servicios OpenThings, los servicios virtuales, los ambientes inteligentes y las aplicaciones.

- Se establecieron plantillas para facilitar la creación de descripciones semánticas de servicios.

- Se implementó un cliente CoAP usando las librerías Californium[24] para permitir la comunicación con servicios en dispositivos que usen dicho protocolo.

- Se integraron funciones del marco de trabajo Jena[25] para usar bases de datos semánticas dentro de los servicios.

- Se incluyeron librerías de SSE para manejar dispositivos por eventos.

- Se definió una ontología básica para las componentes de la arquitectura.

Cabe mencionar que los cambios realizados al marco de trabajo no son compatibles con el uso de las máquinas de ejecución de UbiSOA o el editor de “mashups”

[22] http://jmdns.sourceforge.net/
[23] https://www.apple.com/support/bonjour/
[24] https://eclipse.org/californium/
[25] https://jena.apache.org/

de dicha plataforma. Es por ello que se decidió cambiar el nombre de la nueva plataforma a OpenThings.

5.1.2. Marco de trabajo para ambientes de recursos limitados

Para integrar los componentes de la arquitectura en objetos inteligentes que usen la pila de protocolos CoAP y 6LoWPAN se creó un marco de trabajo con base en el sistema operativo Contiki y las librerías Erbium[26]. El trabajo en este marco de trabajo se resume a lo siguiente:

- Se establecieron las estructuras para integrar servicios OpenThings usando recursos CoAP.
- Se crearon plantillas para describir semánticamente un servicio de CoAP mediante RDF.
- Se estableció el modelo de re-direccionamiento de Cool URI en CoAP.
- Contiki carece de una implementación funcional del protocolo DNS-SD, por lo tanto siguiendo la arquitectura propuesta se creó un servicio virtual para complementar los dispositivos que usan este sistema operativo. El servicio virtual permite realizar el descubrimiento de servicios CoAP mediante el dispositivo que implementa el ruteo entre redes Ipv6 y 6LoWPAN. Dicho servicio virtual fue desarrollado con base a un proyecto de “Router RPL” realizado por Deysi H. Ortega Román y Darién A. Miranda Bojórquez en el Laboratorio de Cómputo Ubicuo de CICESE.

5.2. Aplicaciones implementadas

Para este trabajo de tesis se propuso implementar un par de aplicaciones en donde se integraron características claves de la plataforma. El objetivo de esto es analizar la factibilidad técnica de crear aplicaciones para la plataforma propuesta.

[26] http://people.inf.ethz.ch/mkovatsc/erbium.php

5.2.1. Escenarios

La primera aplicación se propuso con el objetivo de verificar que es posible usar la semántica para incrementar la interacción de dispositivos de IoT. Además de verificar que la información de los servicios cumple con los principios de Linked Data. Para dicha aplicación se planteó el siguiente escenario en el área de domótica:

Se tiene un aula inteligente con diversos objetos que permiten controlar la iluminación. Una profesora entra a un aula y se identifica con una etiqueta RFID. La profesora ha asignado previamente su preferencia por el uso del proyector para llevar a cabo sus clases. Por lo tanto una vez que la profesora entra al aula: las persianas, que dejaban entrar la luz exterior, se cierran; y las lámparas se apagan para dejar el sitio en oscuridad para la proyección. Tiempo después otro profesor entra al aula y se identifica. En sus preferencias el profesor indica que prefiere trabajar con iluminación natural. Puesto que es de día, las lámparas permanecen apagadas pero las persianas se abren. El objetivo de la aplicación es controlar la iluminación del aula con base en perfiles de preferencia de los usuarios.

Posterior al desarrollo de la primera aplicación, se propuso implementar una nueva aplicación para poner a prueba la interoperabilidad y reutilización de componentes. Por esta razón el desarrollo de dicha aplicación se le asignó a Marco Calderón Pérez, colaborador en el laboratorio de cómputo ubico de CICESE.

La capacidad de anexar datos con diversas estructuras es una de las ventajas de usar la semántica al describir datos de dispositivos de IoT. Por lo tanto se propuso explorar el uso de la semántica para describir estructuras de privacidad usando la plataforma propuesta. Por consiguiente el escenario de la segunda aplicación se planteó de la siguiente manera:

Un profesor entra al aula y se identifica con una etiqueta RFID, de acuerdo con sus preferencias de privacidad el profesor ha indicado previamente que no desea que la información de su clase pueda ser vista desde fuera del aula. Por lo tanto las persianas que dan al exterior del edificio se cierran y los vidrios que dan al interior se opacan

automáticamente. Debido a que es posible que entren al aula varios profesores con diversas preferencias de privacidad se usa un servicio de calendario para identificar al profesor que impartirá la clase en ese momento. Además está aplicación tendrá prioridad sobre la configuración de la aplicación de iluminación. El objetivo de la aplicación es: controlar un mecanismo de privacidad en el aula inteligente en función de las preferencias de privacidad del profesor y un calendario de clases.

Para verificar la interoperabilidad entre aplicaciones se deben compartir los componentes del ambiente inteligente del aula sin alterar el diseño de la primera aplicación creada.

5.2.2. Implementación de la primera aplicación

Siguiendo la arquitectura de software propuesta, la primera aplicación se implementó con los siguientes componentes:

El ambiente virtual se creó usando el marco de trabajo OpenThings en Java. Mediante Jena se implementó una base de datos semánticos para almacenar la información contextual de los otros componentes. Los perfiles de preferencias de los usuarios se almacenaron en él ambiente virtual debido a que es información contextual relevante para la ubicación.

El servicio de lector RFID, que permite la identificación de los individuos que entran al aula, fue implementado con un lector Phidget que se conecta por USB a un Rasberry Pi. Este último implementa un servicio con interfaz asistida del marco de trabajo en Java, para crear los componentes del servicio que el lector no puede poseer.

La persiana, representada por un servomotor; y la lámpara, representada por una tarjeta de entrada y salida; se controlan a través de dispositivos Phidget que no poseen capacidad de programación. Por esta razón también se usó el Raspberry Pi, para complementar los servicios de esos dispositivos. Dado que ambos dispositivos son actuadores, se implementó un servicio proxy para evitar conflictos de uso con futuras aplicaciones.

El servicio de sensor de luz fue implementado con un mote TmoteSky usando el protocolo de aplicación CoAP. El mote se comunica mediante el protocolo 6LoWPAN sobre 802.15.4 por lo que se requiere el uso de una puerta de enlace para interconectarlo en red con los otros dispositivos. Para ello se usó una computadora portátil con otro mote conectado por USB, el cual transforma los paquetes 6LoWPAN por 802.15.4 a IPv6 por USB. Adicionalmente una aplicación dentro de la portátil transfiere los datos recibidos del puerto USB a la interfaz de red del sistema. Además se utilizó el servicio virtual de descubrimiento en redes 6LoWPAN para habilitar el uso de DNS-SD por el mote.

Finalmente se creó el servicio de aplicación llamado control de iluminación. Ésta busca los servicios que requiere mediante etiquetas en las descripciones semánticas. Posteriormente la aplicación establece las interfaces de los servicios encontrados. Finalmente la aplicación se registra en el ambiente virtual y entra en funcionamiento.

5.2.2. Resultados de la primera aplicación

Una vez creada la aplicación, se analizó la integración de los componentes con la web semántica y la factibilidad de incrementar la interacción usando la semántica.

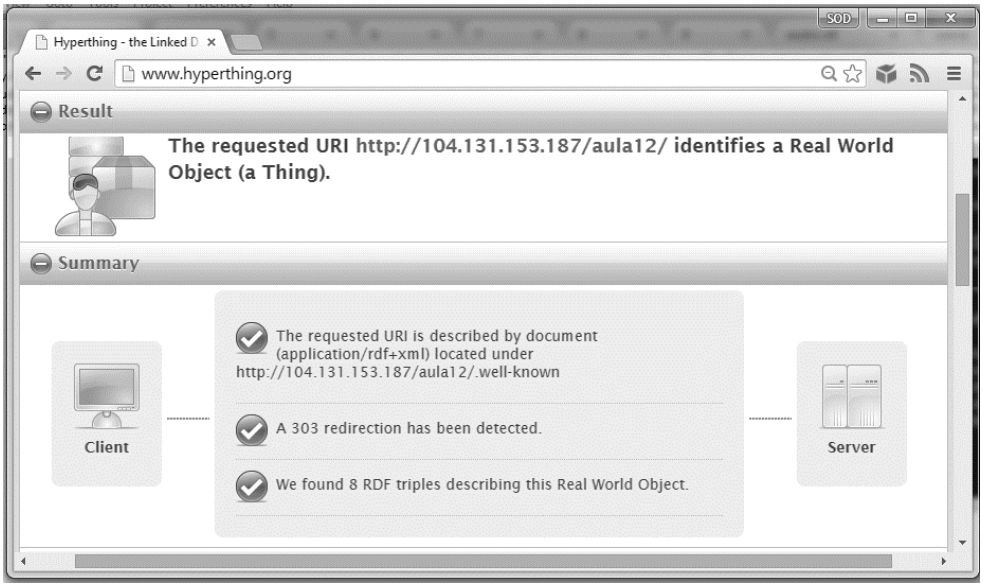


Figura 49 - Ambiente virtual en Linked Data. En la imagen se muestra una captura de pantalla de la herramienta Hyperthing[27], en donde se valida que el ambiente virtual es un objeto de Linked Data.

[27] http://www.hyperthing.org/

Como se puede observar en la Figura 49, se verificó que cada uno de los servicios implementados para la aplicación cumpliera con los principios de Linked Data mediante herramientas de validación disponibles en línea. Estas herramientas indican que los componentes de la plataforma cumplen todos los requisitos para ser considerados entidades de la web semántica.

Como consecuencia de que los componentes de la plataforma sean entidades de de linked data, los usuarios y desarrolladores pueden usar herramientas de la web semántica. Como ejemplo de ello, en la Figura 50, se puede observar una herramienta en línea que permite navegar entre descripciones semánticas. Esta herramienta permite visualizar todos los objetos disponibles en un entorno de IoT solamente con leer la descripción semántica de un ambiente virtual, lo cual facilita el uso de los objetos.

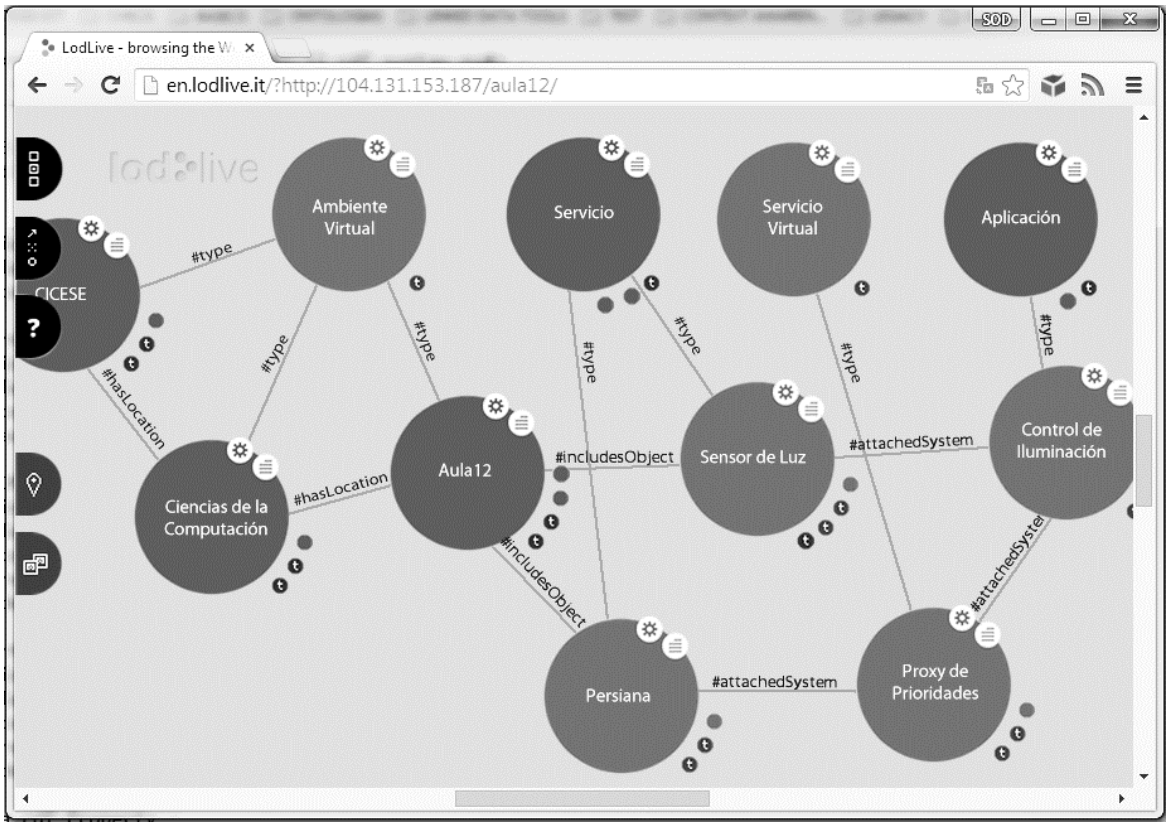


Figura 50 – Navegación de un ambiente virtual. En la imagen se muestra una captura de pantalla de la aplicación lodlive[28], en la cual se diagrama la información contextual almacenada en el ambiente virtual.

[28] http://en.lodlive.it/

Por otra parte la descripción de las interfaces permite la interacción entre los componentes del sistema sin necesidad de intervención humana. Sin embargo, las aplicaciones y servicios virtuales que hacen uso de los otros servicios son los responsables por interpretar las configuraciones de las interfaces. Por lo tanto, entre más complejas son las funciones de los servicios, más compleja se vuelve la descripción semántica de la interfaz y se crean más probabilidades de fallas en el sistema.

Las aplicaciones de la plataforma se pueden reconfigurar en respuesta de los objetos presentes en la red. Esto se debe a que la aplicación no busca un dispositivo específico, sino un dispositivo cualquiera que posea ciertas características. Entre mejor es la capacidad de búsqueda de una aplicación se incrementara la probabilidad de encontrar los servicios adecuados para su funcionamiento. En esta aplicación se usaron las etiquetas semánticas para identificar los servicios requeridos, sin embargo, este modelo requiere una descripción precisa de los servicios e interfaces. Para mejorar esta interacción y explotar mejor las capacidades de las descripciones semánticas se requiere la futura implementación de motores de inferencias en las aplicaciones.

Por lo anterior se concluye que la aplicación cumple con los objetivos de incrementar la interacción entre dispositivos de IoT y validar que los servicios cumplen con los principios de Linked Data. No obstante, el uso de la semántica para incrementar la interacción requiere de descripciones precisas y el uso de buenos motores de inferencias. Estos problemas pertenecen a las capas superiores no estandarizadas de la web semántica, en donde aún se está trabajando para resolverlos.

5.2.3. Implementación de la segunda aplicación

Para implementar la segunda aplicación se requiere agregar nuevos componentes y reutilizar aquellos ya existentes. Para tener una idea más clara de los componentes usados, en la Figura 51 se visualiza el diagrama de emplazamiento del sistema de pruebas distinguiendo entre las dos aplicaciones. A continuación se describen con detalle los componentes que fueron implementados para la segunda aplicación.

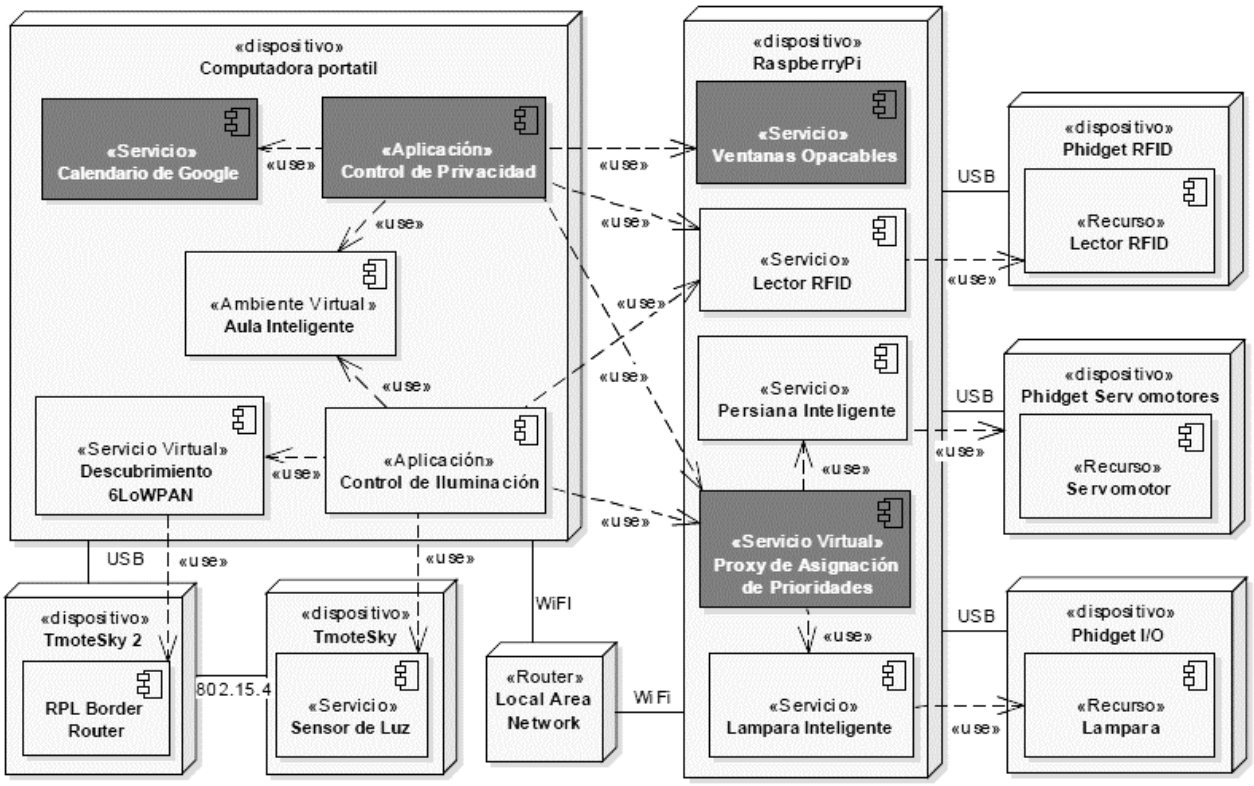


Figura 51 – Diagrama de emplazamiento. En blanco los componentes que se implementaron para la primer aplicación, en negro los componentes que se agregaron para la segunda aplicación.

Primero se implementó un servicio para controlar las ventanas opácales, las cuales se representaron por un puerto de entrada y salida del Raspberry Pi. Este último es un dispositivo de clase 2, por lo tanto todos los componentes del servicio fueron implementados dentro del dispositivo.

Se implementó un servicio que hace uso del API del calendario de Google para asignar fechas con privacidad predefinida. Este servicio se implementó dentro de la computadora portátil usando el marco de trabajo en Java.

Para crear los perfiles de privacidad se usó el ambiente virtual previamente definido. En él se integraron las preferencias de privacidad mediante una estructura semántica a los perfiles previamente creados para el control de iluminación.

Finalmente se creó el servicio de la aplicación de control de privacidad para cumplir con el objetivo planteado en el segundo escenario.

5.2.4. Resultados de la segunda aplicación

La implementación de la segunda aplicación permitió obtener el siguiente análisis sobre el uso de la plataforma propuesta.

El ambiente virtual permitió la reutilización de información contextual relevante para la ubicación y aplicaciones que se encuentran ese entorno. Facilito a desarrolladores ubicar dispositivos presentes en la red e información relevante como los perfiles de los usuarios en ese entorno.

La modularidad resulto ser adecuada para integrar una nueva aplicación y reutilizar componentes del sistema sin necesidad de hacerles modificaciones.

Los servicios virtuales proxy, que permiten escalar los dispositivos, son necesarios para habilitar el uso de actuadores por múltiples aplicaciones. Sin embargo, el uso de servicios virtuales proxy no es suficiente para habilitar el uso de actuadores por múltiples aplicaciones. Durante las pruebas del sistema se identificó el siguiente problema: la aplicación de privacidad cierra la cortina con prioridad sobre la aplicación de iluminación, tal como se estableció en el escenario. Sin embargo, la luz no se enciende en función de si se estableció o no el uso del proyector. Esto es debido a que dicho escenario no se planeó en el desarrollo de ambas aplicaciones.

El desarrollador de una aplicación no tiene forma de anticipar con certeza todos los conflictos que se producirán al compartir dispositivos con otras aplicaciones. Además la nueva aplicación podría afectar indirectamente el comportamiento de otros componentes. Este conflicto supone un nuevo problema a resolver a futuro.

Por lo tanto se concluye que la segunda aplicación cumplió con el objetivo de reutilización de componentes de la primera aplicación sin alterar el diseño de la misma. No obstante, en el proceso se crearon nuevos casos de uso que incrementan la complejidad del diseño de aplicaciones.

5.3. Revisión de requerimientos

En función del análisis del desarrollo de las aplicaciones previamente mencionadas y las características de la arquitectura propuesta, se hizo una revisión de los requerimientos encontrados en la literatura.

5.3.1. Requerimientos soportados

Para este trabajo de tesis se planteó enfocar el desarrollo de la plataforma en mejorar la interoperabilidad e interacción entre objetos de IoT. Por lo tanto el diseño de la arquitectura se centró en los siguientes requerimientos: interoperabilidad, heterogeneidad, interacción, contexto, múltiples dominios, descubrimiento y aplicaciones. Estos requerimientos se revisan a continuación.

Interoperabilidad: La plataforma permite identificar cada uno de los dispositivos mediante el uso de URL, adicionalmente el uso de Cool URI permite distinguir entre los objetos reales y sus descripciones digitales. La plataforma establece el uso de los protocolos estándar HTTP y CoAP para la comunicación entre todos sus servicios. El uso de interfaces descritas semánticamente permite a las aplicaciones interactuar espontáneamente con cualquier servicio sin previo conocimiento del mismo. De igual manera el descubrimiento de servicios permite encontrar a un dispositivo en red local o mediante el uso de información contextual ubicada en ambientes virtuales. Finalmente la arquitectura propone mecanismos que se pueden usar para facilitar la movilidad de dispositivos de IoT entre ambientes virtuales distintos.

Heterogeneidad: La arquitectura define la estructura de servicios que encapsulan: objetos inteligentes, objetos asistidos y objetos simples. Estas abstracciones pueden incluir a todos los objetos de IoT que se describen en el artículo de Mathews *et al.,* (2011). No obstante, integrar las funciones de dichos objetos depende de su capacidad para integrar los componentes descritos en la arquitectura OpenThings. Además servicios externos a la plataforma o interfaces de sistemas legados se pueden encapsular dentro de un servicio e integrarse como un componente más a la plataforma.

Interacción: Cada uno de los servicios dentro la plataforma provee interfaces REST para interactuar con otras entidades de software (M2M). Las descripciones en RDF proveen de mecanismos semánticos para otras aplicaciones, pero además permiten el uso de herramientas de la web semántica, como lodlive. Estas herramientas pueden facilitar la interacción hombre-máquina al brindar interfaces visuales de los entornos de IoT. Además el uso de representaciones HTML brinda a los usuarios una interfaz por defecto para interactuar con los servicios de plataforma. Por último el soporte de interacción por medio de eventos permite la creación de aplicaciones que anticipen las acciones de los usuarios.

Contexto: La plataforma ofrece mecanismos de contexto explícito e independiente de las aplicaciones. Los servicios virtuales permiten obtener datos de un recurso, generar información contextual y poner esta nueva información a disposición de otras entidades en la plataforma. Los ambientes virtuales brindan un modelo estándar para almacenar información útil para otros componentes. La consulta de esa información permite a las aplicaciones y otros servicios virtuales ser conscientes del contexto de su entorno.

Múltiples dominios: Los dispositivos dentro de la plataforma son directamente asociados con ubicaciones representadas por ambientes virtuales, los cuales representan sitios reales o conceptos (áreas de aplicación). El principio de límites, de Tim Kindberg y Armando Fox (2002), define que se deben separar los ambientes ubicuos en función de la ubicación en el mundo real o dominio de aplicación. Los ambientes virtuales brindan esta separación y además se pueden usar con ontologías y modelos de conocimiento específicos al dominio de interés. Lo anterior da como resultado que los objetos y aplicaciones se puedan describir en múltiples contextos. Esta característica es una de las más relevantes de la plataforma.

Descubrimiento: La plataforma permite descubrir cada uno de los servicios usando DNS-SD legado de UbiSOA. Pero además se integró un modelo de descubrimiento semántico usando los ambientes virtuales para extender este descubrimiento más allá de la red local. Por ultimo las descripciones semánticas de las interfaces permiten a las aplicaciones buscar aquellos servicios que les sean de interés

para su funcionamiento. Estas características les permiten a las aplicaciones adaptarse a los servicios disponibles en la red. Lo cual cumple con el principio de volatilidad que requieren los sistemas ubicuos.

Aplicación: Para esta plataforma se describió la estructura de las aplicaciones que hagan uso de los servicios. Las aplicaciones pueden hacer uso de los servicios sin necesidad de mantener una sesión, solo con tener acceso a la red, creando en principio una comunicación anónima entre servicios. Las aplicaciones OpenThings pueden hacer uso de servicios externos, estén o no encapsulados dentro de servicios de la plataforma. Sin embargo, en la arquitectura OpenThings no se ha definido la estructura que deben tener los mecanismos de sesión y control de acceso. Estas características son requerimientos importantes que se deben abordar en un trabajo futuro.

5.3.2. Otros requerimientos

El objetivo del área de investigación es una plataforma de IoT que cubra todos los aspectos del paradigma (Perera, *et al.,* 2014). Por lo tanto aunque el resto de requerimientos de la literatura no son centrales para este trabajo de tesis, éstos se han considerado durante el desarrollo de la plataforma. A continuación se describen las características de la arquitectura en función de estos requerimientos.

Arquitectura: OpenThings sigue los principios arquitectónicos de UbiSOA. Esta última ya definía el uso de una interfaz REST para todos los servicios y la creación de servicios virtuales para extender las características de la plataforma. Siguiendo esta misma arquitectura se describió como los servicios virtuales pueden ser usados para escalar los dispositivos. Además los ambientes virtuales y aplicaciones se definieron como casos especiales de servicios virtuales. Todo esto permite continuar extendiendo la plataforma de forma modular.

Autonomía: La dependencia entre componentes de la arquitectura va de la capa superior a la inferior. En esta estructura las aplicaciones son los componentes que dependerán directamente de los servicios. Por otra parte, al estar en la capa inferior, los servicios OpenThings se comportan de forma completamente autónoma. Aunque la integración de mecanismos de auto mantenimiento dentro de dispositivos no ésta definida

como parte de la plataforma. Estos mecanismos se podrían establecer en un futuro dentro de los recursos de los servicios OpenThings.

Eficiencia: La arquitectura promueve el uso de CoAP para dispositivos con recursos limitados, pero otros protocolos ligeros como MQTT también pueden ser implementados como servicios virtuales. Además la negociación de contenido y las diversas representaciones de recursos permite el uso de Formatos ligeros para la transferencia de datos en JSON, N3, etc. Sumado a esto los dispositivos de capacidades limitadas solo deben ofrecer los recursos que poseen. La carga de trabajo de funciones con mayores requisitos de cómputo y comunicación es delegada a servicios virtuales alojados en dispositivos con mayores capacidades.

Integridad del sistema: El registro en los ambientes virtuales permite comprobar el estado de cada uno de los servicios relacionados, dando una referencia del estado actual de la red. La medición histórica de la información de disponibilidad serviría de referencia para identificar la confiabilidad de los servicios. Sin embargo, la confiabilidad de los datos y calidad del contexto no fueron abordados en la arquitectura debido a que su definición y mecanismos es aún un problema abierto (Bellavista, 2012).

Manejo de información: Los datos producidos por la plataforma deben poseer una representación semántica siguiendo los principios de Linked Data. Esto permite vincular los datos producidos con información contextual como localización y tiempo. El uso de servicios virtuales proxy permite almacenar un historial de datos producidos e interacciones entre servicios. Servicios virtuales y aplicaciones pueden utilizar mecanismos de análisis de datos y a su vez ofrecer los nuevos datos como nuevos recursos dentro de la plataforma.

Escalabilidad: El uso de servicios virtuales para almacenar información de dispositivos con recursos limitados permite escalar su uso a múltiples clientes. Además para el uso de actuadores por múltiples aplicaciones, se estableció el uso de servicios proxy que permitan diferentes modelos de escalabilidad. Finalmente el uso de IPv6 permite escalar la cantidad de nodos hasta 2^{128}, sin embargo, mientras ocurre la

transición de IPv4 a IPv6 se pueden usar las direcciones URL para identificar los dispositivos.

Movilidad: Se pueden implementar servicios virtuales que actualicen la ubicación de otros servicios en los ambientes virtuales de interés. De esta manera las aplicaciones pueden consultar dichos ambientes virtuales para obtener la dirección actual de componentes que pueden estar moviéndose entre redes.

Direcciones: El uso de direcciones que siguen los principios de Cool URI permite establecer un esquema uniforme para nombrar objetos y servicios dentro de la plataforma. Esto sumado al uso de la dirección bien conocida para la descripción semántica permite identificar rápidamente a los servicios de la plataforma.

Seguridad: El uso de HTTP y CoAP permitiría en principio extender la plataforma al uso de protocolos seguros como HTTPS y CoAP sobre DTLS. Sin embargo, estas características no fueron analizadas a detalle para este trabajo de tesis. Además la creación de grupos de usuarios o mecanismos de control de acceso son características que se deben agregar para extender el uso de la plataforma en redes públicas. Aunque el uso de semántica facilita la creación de perfiles y grupos de usuarios como se realizó en las aplicaciones de ejemplo.

Privacidad: Las políticas de privacidad a las que puede estar sujeto un dispositivo de IoT aún son un trabajo en progreso (Henze, *et al.,* 2014). Aun así durante este trabajo de tesis se exploró el uso de perfiles en ambientes virtuales para adjuntar una política de privacidad. Esto permite que múltiples aplicaciones puedan hacer uso de las políticas pero dicha información no está directamente ligada a los usuarios o dispositivos. Se deben explorar mejores modelos para almacenar la información de privacidad.

Calidad de Servicio: Una de las grandes limitantes que posee la plataforma es la falta de mecanismos para garantizar el tiempo de entrega de los mensajes. Esto es debido a la naturaleza de los protocolos web usados para el desarrollo de la plataforma. Tampoco se han definido métodos para medir o asegurar la calidad de experiencia. En

cambio algunos mecanismos de calidad de servicio pueden ser implementados con el uso de servicios proxy y protocolos adicionales como MQTT. Los cuales habilitan el acceso a recursos usando diversas prioridades.

5.4. Análisis de diferencias con otras arquitecturas

Finalmente se hizo un análisis de diferencias con otras plataformas descritas en la literatura para validar que la plataforma propuesta propone una mejora al estado del arte.

Como se puede apreciar en la Tabla 3, las diversas arquitecturas cumplen con diferentes requerimientos. A continuación se describen las diferencias más notables de las plataformas del estado del arte con respecto a la plataforma propuesta en este trabajo.

Sense2Web (Barnaghi, *et al.,* 2010) (De-Supama *et al.,* 2012) es un proyecto fundamentado en el proyecto SENSEI (Tsiatsis, *et al.,* 2010) del mismo grupo de trabajo. En Sense2Web todos los datos de sensores se concentran en una base de datos semántica en donde se agrega la información contextual. Esta concentración permite la consulta de todos los datos de los diversos dispositivos en un solo punto de acceso SPARQL. Además el almacenamiento de información histórica se hace en un servidor de mayores capacidades que los dispositivos de IoT. Esta característica facilita la consulta de información.

Openthings en cambio sigue un modelo distribuido, en donde se separa la información en diversos componentes: datos de dispositivos, información contextual, descripción de dispositivos, etc. Además OpenThings integra actuadores y eventos. Estas características dan a OpenThings un ambiente más adecuado para el desarrollo de aplicaciones de IoT, pero más complejo para consultar información.

Tabla 3 – Análisis de diferencias con arquitecturas de la literatura

Requerimientos /Arquitecturas	OpenThings	SPITFIRE	Sense2Web / SENSEI	XIVELY	u-KB
Interacción	✓	✓	☑	☑	☑
Aplicaciones	✓	✓	☑	☑	☑

Manejo de Información	✓	✓	✓	☑	☑
Escalabilidad	✓	✓	✓	✓	☑
Múltiples Dominios	✓	✓	✓	-	-
Contexto	✓	✓	✓	✓	-
Seguridad	☑	☑	-	✓	-
Privacidad	-	-	-	☑	-
Interoperabilidad	✓	✓	✓	☑	☑
Movilidad	☑	☑	-	✓	☑
Heterogeneidad	✓	✓	✓	☑	✓
Descubrimiento	✓	✓	✓	☑	☑
Direcciones Únicas	✓	✓	✓	✓	✓
Autonomía	☑	✓	-	-	✓
Arquitectura	✓	✓	✓	✓	☑
Eficiencia	✓	✓	✓	☑	✓
Integridad del Sistema	✓	-	-	✓	☑
Calidad de Servicio	☑	☑	-	☑	-

✓Cumple el requerimiento ☑Cumple parcialmente - No lo cumple

Ubiquitous Knowledge Base (u-KB) (Ruta, *et al.,* 2012) hace uso de Identificadores únicos para las ontologías OUUID y compresión de documentos en RDF/OWL. Esto permite mejorar la eficiencia de recursos en dispositivos limitados, pero requiere de mecanismos intermedios para la conexión a Internet. Esta plataforma usa capas semánticas como interfaz para diversos dominios de implementación (e.g., zigbee, rfid). Lo que le permite el uso de todas las capacidades de protocolos de bajo nivel. Además este modelo no requiere el uso de mucho software complementario de alto nivel.

En contraste OpenThings se apoya de dispositivos de mayor capacidad para integrar software complementario. Este software permite el uso de información contextual, múltiples dominios de conocimiento, escalamiento, etc. Además, el uso de estándares compatibles con la Web Semántica permite la interacción directa entre dispositivos y herramientas disponibles en línea. No obstante, la comunicación entre servicios se lleva a cabo en la capa de aplicación (usando HTTP o CoAP), por lo que

algunas funciones de comunicación de protocolos de más bajo nivel no son aprovechadas.

XIVELY[29] es una plataforma en la nube que permite extraer las funciones de los dispositivos y controlarlos usando herramientas web. Es la única plataforma analizada que provee mecanismos de seguridad, privacidad y calidad de servicio, aunque tampoco garantiza aspectos como tiempo de entrega o eficiencia. El control de dispositivos dentro de la nube facilita el escalamiento y administración de dispositivos. Además XIVELY facilita el desarrollo de aplicaciones web que integren dispositivos de IoT mediante un API e interfaces graficas. Sin embargo no favorece el uso de la semántica lo que limita el descubrimiento y interaccion entre dispositivos.

Por otra parte OpenThings, que sigue un ambiente distribuido, hace uso de la semántica para crear una red de ambientes virtuales en donde localizar dispositivos. Además la semantica le permite a una aplicación: analizar un entorno, encontrar servicios adecuados e interactuar con ellos sin necesidad de intervención humana.

SPITFIRE (Pfisterer, *et al.,* 2011) (Chatzigiannakis, *et al.,* 2012) (Orestis, *et al.,* 2012) es la plataforma con más similitudes a la propuesta en este trabajo de tesis. Incluso la ontología SPITFIRE fue integrada a la arquitectura OpenThings para describir dispositivos de IoT. Entre las principales diferencias es que SPITFIRE posee mecanismos más desarrollados para el manejo de la semántica. Las aplicaciones de SPITFIRE pueden hacer uso de reglas semánticas para la interacción entre dispositivos de IoT. La plataforma provee mecanismos para describir automáticamente algunas características de los dispositivos. Por último SPITFIRE integra un modelo de predicción para mejorar las consultas en SPARQL de dispositivos con un estado en particular.

OpenThings en cambio se enfoca más en la interacción entre dispositivos de un entorno en particular. Los ambientes virtuales permiten el uso de múltiples dominios de conocimiento ligados a diversos ambientes de aplicación. Estos ambientes sirven como

[29] https://xively.com/

apoyo para conocer el estado de los servicios de una red, así como apoyo para la movilidad de dispositivos.

5.5. Conclusiones

En este capítulo se describió el desarrollo del marco de trabajo para la plataforma OpenThings y el desarrollo de dos aplicaciones para verificar la factibilidad del uso de la plataforma. Además se revisaron las características de la arquitectura con cada uno de los requerimientos encontrados en el estado del arte y se describieron las diferencias con otras plataformas del estado del arte.

Capítulo 6 Conclusiones y trabajo futuro

En este capítulo se resumen las contribuciones y limitaciones de este trabajo. Se discute el trabajo futuro que pueda derivar de este trabajo.

6.1. Conclusiones

En este trabajo de tesis se extendió la plataforma UbiSOA (Avilés López, 2012) usando tecnologías semánticas con el fin de mejorar la interacción e interoperabilidad entre datos y servicios. En consecuencia se propuso la plataforma semántica llamada OpenThings.

Para el desarrollo de la plataforma OpenThings se diseñó una nueva arquitectura de software que integra todos los dispositivos de IoT usando servicios web y tecnologías estándar de la web semántica. Se establecieron los principios a seguir para la creación de servicios semánticos de IoT siguiendo las prácticas comunes del modelo de Linked Data. El uso de estos principios permite a los servicios de la plataforma usar herramientas y servicios de la web semántica. Además se desarrollaron los ambientes virtuales, servicios que permiten conocer el estado de los dispositivos presentes en la red y su información contextual. Por lo tanto se cumple con el objetivo principal de la tesis de: “Crear una plataforma que extienda la plataforma de UbiSOA mediante el uso de tecnologías semánticas, con el fin de mejorar las capacidades de interoperabilidad e interacción entre objetos de IoT.”

6.2. Aportaciones

Durante el desarrollo de este trabajo de investigación se obtuvieron las siguientes aportaciones:

- Una lista de requerimientos para crear plataformas de IoT con base en diversos trabajos en el estado del arte. Esta cubre requerimientos de: aplicaciones, componentes middleware, sistemas de cómputo ubicuo e inteligencia ambiental. Los requerimientos fueron agrupados y reducidos a una lista con 18 temas que

cubren los aspectos del paradigma. La lista fue usada para analizar diversas plataformas en el estado del arte para el paradigma de IoT. Además la lista puede ser usada para evaluar nuevas plataformas.

- Una arquitectura distribuida para IoT con base en el modelo orientado a servicios, la cual fue extendida de la arquitectura UbiSOA. Sus principales características son:

    - Representaciones de los servicios mediante el uso de tecnologías de la web semántica: RDF, OWL y SPARQL.
    - Integración del modelo de Linked Data para enlazar datos generados en la plataforma con otros servicios semánticos en línea.
    - Ambientes virtuales que agrupan servicios en función de su ubicación física u otros conceptos abstractos. Los cuales sirven como mecanismos para analizar el estado de los servicios y las interacciones que tienen entre ellos.

- Un marco de trabajo para crear servicios y aplicaciones siguiendo la arquitectura propuesta. El cual extiende el marco de trabajo en Java de la plataforma UbiSOA. Además se implementó un nuevo modelo de suscripción a eventos y se integró un cliente CoAP. Adicionalmente sé creo un marco de trabajo para crear servicios de la plataforma en dispositivos de recursos limitados usando el sistema operativo Contiki y las librerías de CoAP Erbium.

6.3. Limitaciones

En el desarrollo del éste trabajo se encontraron algunas limitaciones, las cuales se describen a continuación:

- **Seguridad.** Aunque se estableció no abordar esta característica por motivos de tiempo limitado, extender la plataforma con mecanismos de seguridad es necesario para la creación de aplicaciones a un nivel comercial. La seguridad es

necesaria al publicar servicios usando direcciones IP externas, con el fin de evitar conflictos de uso de dispositivos.

- **Ambigüedad semántica.** No hay manera de determinar si las descripciones semánticas de datos y servicios son correctas mientras que las capas superiores de la web semántica no estén establecidas. El uso de conceptos definidos por ontologías está en función de la interpretación que se le dé por parte de los desarrolladores. Existe la posibilidad de ambigüedades entre diversas plataformas que usan las mismas bases de conocimiento, incluso implementando modelos de descripción automática. Esto es una fuerte limitación de interoperabilidad que debe ser abordada como trabajo futuro.

- **Marco de trabajo.** La implementación del marco de trabajo es limitada ya que actualmente solo está disponible en Java. Extender el marco de trabajo a otros lenguajes e incluir soporte para diversos dispositivos facilitaría la creación de aplicaciones para una gama más amplia de desarrolladores. El marco de trabajo en dispositivos de recursos limitados solo se ha implementado en Contiki, extenderlo a otros sistemas como TinyOS, Arduino, etc., facilitaría la creación de aplicaciones en ambientes multiplataforma.

6.4. Trabajo futuro

La evaluación de la plataforma, así como la comparación de otras plataformas del estado del arte permitió identificar diversos temas de trabajo futuro:

- **Mejorar ontologías.** Aunque existen diversas ontologías para IoT, ninguna cubre todos los aspectos del paradigma, en diversas ocasiones los conceptos definidos están abiertos a interpretación por los desarrolladores. Establecer ontologías estándar tal como la “Semantic Sensor Network Ontology” que cubra todos los aspectos de Internet de las Cosas es un trabajo futuro clave para que la semántica pueda ser utilizada entre plataformas.

- **Lógica y motores de inferencias.** Analizar las descripciones semánticas de los servicios mediante lógica y motores de inferencias permitirá mejorar la interactividad de las aplicaciones con los servicios. Por otra parte el uso de lógica para verificar la estructura de ontologías y descripciones semánticas es necesaria para mejorar la interoperabilidad entre plataformas.

- **Modelos de solución de conflictos.** Los diversos modelos de escalabilidad para dispositivos de recursos limitados es un área prometedora que ha sido poco explorada en el área. La plataforma permite implementar servicios virtuales con diversos modelos de escalabilidad. Los modelos de solución de conflictos en el uso de actuadores así como el establecimiento de prioridades de uso, son piezas claves para el escalamiento global de sistemas de IoT. Además estos modelos podrían ser un buen punto de inicio para establecer niveles de calidad de servicio.

- **Privacidad.** La privacidad de los datos y servicios de IoT es un tema completamente abierto, algunas plataformas como Xively se limitan a establecer políticas de privacidad para el acceso a los dispositivos. Pero no se han establecido modelos de privacidad para los datos o de información que se pueda derivar de ellos.

- **Calidad de servicio y calidad de experiencia.** Medir la calidad de experiencia del usuario es un área que no ha sido abordada por las plataformas en el estado del arte. Está resultaría de gran utilidad para mejorar aplicaciones de ambientes de IoT. Por otra parte la calidad de servicio no está completamente resuelta en las plataformas, tal como en la plataforma presentada en este trabajo de tesis donde solo se aborda calidad de servicio mediante el uso de MQTT. Pero dar soporte de calidad de servicio en todos los dispositivos de la plataforma sería lo ideal.

- **Plataformas conscientes de políticas y procesos.** La plataforma presentada permite a los dispositivos estar conscientes del contexto y actividad, pero tal como lo indican Thiesse y Michahelles (2010), incrementar el conocimiento de

los objetos inteligentes a las políticas y procesos permitiría aumentar el nivel de abstracción. Dicha abstracción permitiría incrementar las posibilidades de interacción para crear aplicaciones. El uso de reglas semánticas podrían ser un punto de inicio para la creación de una plataforma consiente de políticas. Extender los ambientes virtuales para mejorar las descripciones del uso de servicios y aplicaciones podría ser la base para una plataforma consiente de procesos.

- **Interacción con ambientes de IoT.** Los Visores Sensoriales (Estrada-Martínez, 2011), los cuales permiten la interacción con dispositivos de IoT mediante las descripciones semánticas, podrían acoplarse a la plataforma OpenThings para explorar la interacción con ambientes inteligentes. Establecer mecanismos para representar un ambiente virtual de la plataforma usando visores sensoriales sería un posible punto de partida.

Lista de Referencias

Alapetite, A. (2010). Dynamic 2D-barcodes for multi-device Web session migration including mobile phones. *Personal and Ubiquitous Computing, 14*(1), 45-52. doi:10.1007/s00779-009-0228-5

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks, 54*(15), 2787-2805. doi:10.1016/j.comnet.2010.05.010

Avilés López, E. (2012). *Creación de aplicaciones para ambientes inteligentes: un marco de trabajo basado en fuentes de datos y servicios en la Internet de las Cosas.* Tesis de doctorado en ciencias. Centro de Investigación Científica y de Educación Superior de Ensenada. 179 p.

Avilés-López, E., y García-Macías, J. A. (2009). TinySOA: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications, 3*(2), 99-108. doi:10.1007/s11761-009-0043-x

Avilés-López, E., y García-Macías, J. A. (2012). Mashing up the Internet of Things: a framework for smart environments. *EURASIP Journal on Wireless Communications and Networking, 2012*(1), 1-11. doi:10.1186/1687-1499-2012-79

Ayers, D., and Völker, M. (2008, Dec 03). *Cool URIs for the Semantic Web.* Recuperado el 13 de Septiembre de 2014, de http://www.w3.org/TR/cooluris/

Bandyopadhyay, S., and Sengupta, M. (2011). A survey of middleware for internet of things. *Recent Trends in Wireless and Mobile Networks*, 288-296. doi:10.1007/978-3-642-21937-5_27

Bandyopadhyay, S., Sengupta, M., Maiti, S., and Dutta , S. (2011). Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey, 2*(3), 94-105. doi:10.5121/ijcses.2011.2307

Barnaghi, P., Presser, M., and Moessner, K. (2010). Publishing linked sensor data. In *CEUR Workshop Proceedings: Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN), Organised in conjunction with the International Semantic Web Conference* (Vol. 668).

Barnaghi, P., Wang, W., Henson, C., and Taylor, K. (2012). Semantics for the Internet of Things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS), 8*(1), 1-21. doi:0.4018/jswis.2012010101

Barthel, R., Hudson-Smith, A., Jode, M., and Blundell, B. (2010). Tales of Things The Internet of 'Old'Things: Collecting Stories of Objects, Places and Spaces. Presented at the First International Workshop the Urban Internet of Things, *Internet of Things 2010*, Tokyo, Japan.

Bellavista, P. (2012). A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys (CSUR)*, 49. doi:10.1145/2333112.2333119

Berners-Lee, T., Bizer, C., and Heath, T. (2009). Linked data-the story so far. *International Journal on Semantic Web and Information Systems 5.3, 5*(3), 1-22. doi:10.4018/jswis.2009081901

Berners-Lee, T., Fielding, T., and Masinter, L. (2005, Jan). *Uniform Resource Identifier (URI): Generic Syntax RFC3986.* Recuperado el 12 de Nov de 2014, de http://tools.ietf.org/html/rfc3986

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American, 284*(5). doi:10.1038/scientificamerican0501-34

Bikakis, N., Tsinaraki, C., Gioldasis, N., Stavrakantonakis, I., and Christodoulakis, S. (2013). The XML and Semantic Web Worlds: Technologies, Interoperability and Integration: A Survey of the State of the Art. *Semantic Hyper/Multimedia Adaptation* (pp. 319-360). Springer Berlin Heidelberg.

Bormann, C., Ersue, M., and Keranen, A. (2014, May). *Terminology for Constrained-Node Networks.* Recuperado el 16 de Noviembre de 2014, de http://tools.ietf.org/search/rfc7228

Cardelli, L., and Gordon, a. A. (1998). Mobile ambients. *Foundations of Software Science and Computation Structures*, 140-155. doi:10.1017/S0960129502003742

Castellani, A. P., Gheda, M., Bui, N., Rossi, M., and Zorzi, M. (2011, June). Web Services for the Internet of Things through CoAP and EXI. *2011 IEEE International Conference on Communications Workshops (ICC)*, (pp. 1-6). IEEE. doi: 10.1109/iccw.2011.5963563

Chander, R. V., Elias, S., Shivashankar, S., and Manoj, P. (2012). A REST based design for Web of Things in smart environments. *Proceedings of 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC), Waknaghat, Solan, Himachal Pradesh, India*(2012), 6-8. doi: 10.1109/PDGC.2012.6449842

Chatzigiannakis, I., Hasemann, H., Karnstedt, M., Kleine, O., Kroller, A., Leggieri, M., and Truong, C. (2012). True self-configuration for the IoT. *2012 3rd International Conference on the Internet of Things (IOT),* 9-15. IEEE. doi: 10.1109/IOT.2012.6402298

Cheshire, S. (2014, Oct 10). *DNS Service Discovery (DNS-SD).* Recuperado el 28 de Octubre de 2014, de http://www.dns-sd.org/

CISCO. (2014, Oct 21). *Connections Counter: The Internet of Everything in Motion*. Recuperado el 25 de noviembre de 2014, de http://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342

de Diego, R., Martínez, J.-F., Rodríguez-Molina, J., and Cuerva, A. (2014). A Semantic Middleware Architecture Focused on Data and Heterogeneity Management within the Smart Grid. *Energies*, 5953-5994. doi:10.3390/en7095953

de Jode, M. L., Barthel, R., and Hudson-Smith, A. (2011). Tales of things: the story so far. *Proceedings of the 2011 international workshop on Networking and object memories for the internet of things,* 19-20. ACM. doi:10.1145/2029932.2029940

De, S., Elsaleh, T., Barnaghi, P., and Meissner, S. (2012). An internet of things platform for real-world and digital objects. *Scalable Computing: Practice and Experience 13(1)*, 45-57.

Denso Wave Inc. (2014, Nov 20). *QR Code*. Recuperado el 08 Nov 2014, de http://www.qrcode.com/en/about/standards.html

Dunkels, A. V. (2008). Internet Protocol for Smart Objects (IPSO) Alliance. *White Paper #1*. Recuperado el 5 de noviembre de 2014, de http://www.ipso-alliance.org/

Estrada-Martínez, P. E. (2011). *Visores Sensoriales Para Entornos Físicos Aumentados.* Tesis de maestría en ciencias. Centro de Investigacion Cientifica y Educación Superior de Ensenada Baja California. 121 p.

Estrada–Martinez, P. E., y Garcia–Macias, J. A. (2013). Semantic interactions in the Internet of Things. *International Journal of Ad Hoc and Ubiquitous Computing*, 13(3), 167-175. doi: 10.1504/IJAHUC.2013.055464

Eurotech and IBM. (2010). *MQTT V3.1 Protocol Specification.* Recuperado el 27 de Octubre de 2014, de http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

Evans, D. (2011). The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. *CISCO white paper 1, 1*.

Fette, I., and Melnikov, A.(2011, Dec). *The WebSocket Protocol RFC6455.* Recuperado el 19 de Septiembre de 2014, de https://tools.ietf.org/html/rfc6455

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* Tesis de doctorado en ciencias. University of California. 162 p.

Gomez, C., Oller, J., and Paradells, J. (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors 12.9*, 11734-11753. doi: 10.3390/s120911734

Google. (2014). *Google Trends*. Recuperado el 2 de Diciembre de 2014, de http://www.google.com/trends/explore#q=internet%20of%20things%2C%20ubiquitous%20computing%2C%20mobile%20computing&cmpt=q

Guinard, D., Trifa, V., and Wilde, E. (2010). A Resource Oriented Architecture for the Web of Things. In *Internet of Things (IOT), 2010* (pp. 1-8). Tokyo: IEEE. doi:10.1109/IOT.2010.5678452

Hachem, S., Teixeira, T., and Issarny, V. (2011). Ontologies for the Internet of Things. *Proceedings of the 8th Middleware Doctoral Symposium.* New York: ACM. doi:10.1145/2093190.2093193

Hachman, M. (2014, Jan 07). *Intel CEO points toward wearable future with 'smart earbud', smartwatch*. Recuperado el 29 de Noviembre de 2014, de http://www.pcworld.com/article/2085003/intel-ceo-points-toward-wearable-future-with-smart-earbud-smartwatch.html

Henze, M., Hermerschmidt, L., Kerpen, D., Häußling, R., Rumpe, B., and Wehrle, K. (2014). User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things. *2014 International Conference on Future Internet of Things and Cloud (FiCloud). 191-196. doi: 10.1109/FiCloud.2014.38*

Hickson, I. (2014, May 14). *Server-Sent Events.* Recuperado el 13 de Septiembre de 2014, de http://dev.w3.org/html5/eventsource/

Internet of Things Arquitecture Project. (2011, Oct 09). *D6.2 - Updated Requirement List.* Recuperado el 3 de Diciembre de 2014, de http://www.iot-a.eu/public/public-documents/d6.2-updated-requirements-list/view

Kim, J., and Lee, J.-W. (2014). OpenIoT: An open service framework for the Internet of Things. *Internet of Things (WF-IoT), 2014 IEEE World Forum on* (pp. 89-93). Seoul: IEEE. doi:10.1109/WF-IoT.2014.6803126

Kindberg, T., and Fox, A. (2002). System software for ubiquitous computing. *IEEE pervasive computing 1.1, 1*(1), 70-81. doi:10.1109/MPRV.2002.993146

Knorr, M., Hitzler, P., and Maier, F. (2012). Reconciling OWL and non-monotonic rules for the Semantic Web. *Frontiers in Artificial Intelligence and Applications*, 474-479. doi:10.3233/978-1-61499-098-7-474

Korper, G. (2006, Sep 19). Using Bonjour Across Subnets. Recuperado el 28 de Octubre de 2014, de http://www.grouplogic.com/Knowledge/PDFUpload/Info/WanBonjour_1.pdf

Lee , G. M., Crespi, N., Choi, J. K., and Boussard, M. (2013). Internet of Things. In *Evolution of Telecommunication Services,* 257-282. Springer Berlin Heidelberg. doi:10.1007/978-3-642-41569-2_13

Mathew, S. S., Atif, Y., Maamar, Z., and Sheng, Q. Z. (2011). Web of Things: Description, Discovery and Integration. *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, 9-15. doi:10.1109/iThings/CPSCom.2011.165

McLellan, C. (2013, Oct 21). *IBM and Libelium release a 6LoWPAN development platform for IoT applications.* Recuperado el 27 de Octubre de 2014, de: http://www.zdnet.com/ibm-and-libelium-release-a-6lowpan-development-platform-for-iot-applications-7000022203/

Nambi, S. N., Sarkar, C., Prasad, R. V., and Rahim, A. (2014). A unified semantic knowledge base for IoT. *2014 IEEE World Forum on Internet of Things (WF-IoT)* (pp. 575-580). Seoul: IEEE. doi:10.1109/WF-IoT.2014.6803232

National Intelligence Council. (2008, Apr). *Disruptive Technologies Global Trends 2025.* Recuperado el 20 de Noviembre de 2014, de http://fas.org/irp/nic/disruptive.pdf

Nehmer, J., Karshmer, A., Becker, M., and Lamm, R. (2006). Living assistance systems: an ambient intelligence approach. *Proceedings of the 28th international conference on Software engineering*, 43-50. doi:10.1145/1134285.1134293

Nikolaos, T., and Kiyoshi, T. (2010, July). Qr-code calibration for mobile augmented reality applications: linking a unique physical location to the digital world. *ACM SIGGRAPH 2010 Posters*, p. 144. doi:10.1145/1836845.1836999

Nottingham, M., and Hammer-Lahav, E. (2010, Abr). Defining Well-Known Uniform Resource Identifiers (URIs) RFC 5785. Recuperado el 20 Octubre de 2014, de https://tools.ietf.org/html/rfc5785

Orestis, A., Dimitrios, A., and Chatzigiannakis, I. (2012). Towards integrating IoT devices with the Web. *2012 IEEE 17th Conference on. IEEE Emerging Technologies & Factory Automation (ETFA).* 1-4. Krakow: IEEE. doi: 10.1109/ETFA.2012.6489729

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *Communications Surveys & Tutorials, IEEE, 16*(1), 414-454. doi:10.1109/SURV.2013.042313.00197

Pfisterer, D., Romer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C. Hauswirth, M. (2011). SPITFIRE: toward a semantic web of things. *Communications Magazine, IEEE, 16*(11), 40--48. doi:10.1109/MCOM.2011.6069708

Ruta, M., Scioscia, F., and Di Sciascio, E. (2012). Enabling the Semantic Web of Things: framework and architecture. *2012 IEEE Sixth International Conference on Semantic Computing (ICSC)* (pp. 345 - 347). Palermo: IEEE. doi:10.1109/ICSC.2012.42

Shelby, M. Hartke, K., and Bormann, C.(2014, June). *The Constrained Application Protocol (CoAP) RFC 7252.* Recuperado el 12 de Octube de 2014, de https://tools.ietf.org/html/rfc7252

Sutaria, R., and Govindachari, R. (2013). Making sense of interoperability: Protocols and Standardization initiatives in IOT. *Mindtree Research Labs.* Recuperado el 26 de

Septiembre de 2014, de http://rsutaria.net/wp-content/uploads/2013/02/Low_power_IoT_ComNet_2013_Mindtree.pdf

Tan, L., and Wang, N. (2010). Future internet: The Internet of Things. *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), 5*, V5--376. doi:10.1109/ICACTE.2010.5579543

Thangavel, D., Keng-Yan TAN, C., and Xiaoping Ma, A. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP).* 1-6 2014 IEEE Ninth International Conference on. IEEEI. doi:10.1109/ISSNIP.2014.6827678

Thiesse, F., and Michahelles, F. (2010). Smart objects as building blocks for the Internet of things. *IEEE Internet Computing, 14*(1), 44-51. doi:10.1109/MIC.2009.143

Traub, K., Allgair, G., Barthel, H., Burstein, L., Garrett, J., Hogan, B., and Schramek, C. (2005). The EPCglobal architecture framework. *EPCglobal Ratified specification*.

Tsiatsis, V., Gluhak, A., Bauge, T., Montagut, F., Bernat, J., Bauer, M. and Krco, S. (2010). The SENSEI Real World Internet Architecture. *Towards the Future Internet*. 247 - 256. doi:10.3233/978-1-60750-539-6-247

Want, R. (2006). An introduction to RFID technology. *IEEE Pervasive Computing, 5*(1), 25-33. doi:10.1109/MPRV.2006.2

Weber, W., Rabaey, J., and Aarts, E. H. (2005). Ambient intelligence (1st ed.). Springer Science & Business Media.

Wei Wang, S. D., Ralf Toenjes, E. R., and Moessner, K. (2012). A Comprehensive Ontology for Knowledge Representation in the Internet of Things. *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (pp. 1793-1798). Liverpool: IEEE. doi:10.1109/TrustCom.2012.20

Weiser, M. (1991). The computer for the 21st century. *Scientific american, 265*(3), 94-104. doi:10.1038/scientificamerican0991-94

Yick, J., Mukherjee, B., and Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, 52(12), 2292–2330. doi:10.1016/j.comnet.2008.04.002

Apéndice 1 Requerimientos IoT-A

1.1 El sistema debe permitir la interoperabilidad entre dispositivos, recursos, servicios y aplicaciones.

1.2 El sistema proveerá interfaces con otros sistemas, incluyendo sistemas legados.

1.3 El sistema manejará interoperabilidad semántica a diferentes niveles semánticos.

1.4 El sistema proveerá una solución de interoperabilidad a nivel de nombres y direcciones.

1.5 El sistema proveerá comunicación estandarizada y semántica entre servicios.

1.6 El sistema debe soportar procesamiento de datos (filtrado, agregación, fusión, etc.) a diferentes niveles del sistema (e.g., a nivel de dispositivo).

1.7 El sistema proveerá soporte para rastrear la localización de dispositivos (localización geo-espacial y/o localización lógica).

1.8 El sistema proveerá comunicaciones seguras (e.g., para información médica).

1.9 El sistema proveerá diferentes permisos de acceso a la información.

1.10 El sistema proveerá comunicaciones altamente confiables y seguras, así como manejo de la información.

1.11 El sistema proveerá acceso a fuentes externas de información (e.g., bases de datos médicas).

1.12 El sistema propondrá medios para diseñar aplicaciones tomando en cuenta las descripciones semánticas de los dispositivos y servicios.

1.13 El sistema proveerá una forma de usar anónimamente servicios de Internet de las Cosas.

1.14 El sistema proveerá un modelo para describir semánticamente objetos físicos.

1.15 El sistema proveerá validación de integridad de entidades virtuales, dispositivos, servicios, y recursos.

1.16 El sistema proveerá información histórica del monitoreo de dispositivos.

1.17 El sistema debe hacer accesible la información semántica a actores humanos y agentes de software.

Apéndice 2 Requerimientos middleware IoT

2.1 *Identificación Única*. Cada objeto dentro de la red debe tener una identificación única sin importar su localización o las diferentes tecnologías de identificación utilizadas por el objeto.

2.2 *Interoperabilidad*. Debe haber un esquema de interacción homogéneo dentro de la red para proveer y utilizar cualquier servicio o información.

2.3 *Minimizar el consumo de energía.* Se deben usar protocolos ligeros para eliminar los mensajes y cargas innecesarias en objetos pequeños con energía limitada.

2.4 *Red auto configurable.* Los objetos se deben comunicar independientemente de las tecnologías usadas en capas inferiores. Además de adaptarse a los diferentes dominios de aplicación y topología cambiante de la red.

2.5 *Servicios Autónomos.* Los servicios deben proveer captura, comunicación y procesamiento automático de datos usando reglas configuradas por los operadores o subscriptores.

2.6 *Movilidad de objetos.* Los objetos dentro de la red pueden ser móviles, conectándose a diversas redes.

2.7 *Red escalable*. La red debe estar preparada para un esquema de comunicación global con miles de millones de objetos conectados.

2.8 *Conectividad directa*. Se debe tener conectividad de extremo a extremo para poder interactuar con cada objeto dentro de la red.

2.9 *Servicios Confiables.* Debe haber un nivel verificable de precisión y manejo de tiempos, para dar soporte a calidad de servicio (QoS) y calidad de experiencia (QoE).

2.10 *Seguridad y Privacidad.* Se debe dar soporte de protección de privacidad a la transmisión, almacenamiento y procesamiento de datos.

2.11 *Auto-mantenimiento.* Los servicios deben ser capaces de funcionar con la mínima intervención humana.

2.12 *Arquitectura orientada a servicios*. Para proveer reusabilidad de servicios se debe usar una arquitectura distribuida basada en componentes, tal como las arquitecturas orientadas a servicios.

Apéndice 3 Componentes de middleware de IoT

3.1 *Detección de contexto.* El sistema debe ser consciente del contexto, además debe proveer mecanismos para realizar detección y procesamiento de contexto.

3.2 *Descubrimiento de Servicios.* Cualquier dispositivo de la red puede encontrar todos los dispositivos cercanos y dar a conocer su presencia.

3.3 *Seguridad y Privacidad*. Se debe implementar la seguridad en dos formas: Comunicación segura de alto nivel entre pares; Una topología segura para brindar autenticación entre pares nuevos y permisos de acceso en la red. De esta manera protegiendo la información que es intercambiada en la red.

3.4 *Manejo de Volúmenes de Datos.* Al permitir que el sistema escale a millones de dispositivos, enormes cantidades de datos deben de ser procesados. Se deben ofrecer mecanismos para dar soporte para: procesar los datos, realizar búsquedas, indexados, modelado del proceso, identificación de datos, información espacial, datos históricos, descripción de datos, etc.

Apéndice 4 Requerimientos de sistemas de cómputo ubicuo

4.1 *Integración Física.* Los objetos inteligentes de un sistema ubicuo se encuentran en contacto con el mundo físico, por lo tanto se deben considerar las asociaciones culturales, administrativas, o territoriales. Las cuales forman el contexto que rodea los objetos. En otras palabras, el mundo consiste de diversos sistemas ubicuos y no solo un sistema único. Por lo tanto se debe cumplir él siguiente principio de límites: “Los diseñadores de sistemas ubicuos deben dividir el mundo en ambientes, delimitando su contenido. Debe existir un claro criterio para delimitar el sistema, estando o no ligado con el mundo físico. Esté límite debe definir el entorno de un ambiente pero no necesariamente restringir la interoperabilidad”.

4.2 *Interoperación Espontanea.* En un ambiente inteligente hay componentes (Cardelli and Gordon, 1998), es decir unidades de software que implementan abstracciones como: servicios, recursos, o aplicaciones. En los sistemas ubicuos los componentes deben poseer interoperabilidad en ambientes altamente cambiantes. Los componentes deben intercambiar mutuamente su identidad y funcionalidad conforme van cambiando las circunstancias con el tiempo. Para ello se debe cumplir el principio de volatilidad: “Se deben diseñar sistemas ubicuos suponiendo que: usuarios, hardware, y software, son altamente dinámicos e impredecibles. Debe haber claras invariantes que gobiernen el sistema completo”.

Apéndice 5 Requerimientos de sistemas de inteligencia ambiental

5.1 *Invisible.* Los dispositivos o servicios deben estar totalmente escondidos al usuario (e.g., cómputo vestible).

5.2 *Movilidad.* Los usuarios y dispositivos pueden estar en movimiento constante.

5.3 *Consciente del Contexto.* Los sensores e infraestructura deben ser capaces de modelar el contexto y proveer contexto derivado más complejo.

5.4 *Anticipatorio.* Los sistemas deben ser capaces de actuar en representación de sus usuarios sin requerir una solicitud explicita de una acción.

5.5 *Comunicación natural.* Se deben proveer medios para interactuar naturalmente con el sistema, por ejemplo: a través de objetos que están acostumbrados a usar en su vida diaria.

5.6 *Adaptativo.* Los dispositivos deben adaptarse a diferentes situaciones y anormalidades para evitar la interrupción de los servicios.

5.7 *Robustez.* El sistema debe ser extremadamente robusto a pesar del mal uso y errores. Las entradas incorrectas no deben provocar que el sistema colapse.

5.8 *Disponibilidad.* El sistema debe continuar su funcionamiento a pesar de componentes de hardware problemáticos.

5.9 *Extensibilidad.* El sistema debe permitir agregar nuevos componentes en tiempo de ejecución.

5.10 *Confianza.* Se deben proveer mecanismos de validación y verificación para asegurar que los servicios sean precisos en sus tareas.

5.11 *Privacidad.* Se debe garantizar un cierto grado de privacidad de las personas que usan el sistema.

5.12 *Puntualidad.* Algunos servicios deben asegurar cierto nivel de confiabilidad en el tiempo, por ejemplo: servicios para uso de sistemas de emergencias médicas.

5.13 *Eficiente.* El poder de procesamiento, memoria, ancho de banda y energía: deben ser utilizados lo más eficientemente posible.