SWITCH: UN MIDDLEWARE PARA EL DESARROLLO DE APLICACIONES IOT CON INTERFACES BASADAS EN VOZ

JOHANA ANDREA MANRIQUE HERNÁNDEZ

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA - UNAB
FACULTAD DE INGENIERÍA
MAESTRÍA EN TELEMÁTICA, MODALIDAD INVESTIGACIÓN
GRUPOS: TECNOLOGIAS DE INFORMACION Y PENSAMIENTO SISTEMICO
BUCARAMANGA, COLOMBIA
NOVIEMBRE DE 2018

SWITCH: UN MIDDLEWARE PARA EL DESARROLLO DE APLICACIONES IOT CON INTERFACES BASADAS EN VOZ

JOHANA ANDREA MANRIQUE HERNÁNDEZ

Trabajo de grado para optar al título de Magíster en Telemática, modalidad investigación

Director
Jesús Martín Talavera Portocarrero, Ph.D.
Ingeniero de Software
Nuance Communications

Co-director
José Daniel Cabrera Cruz, M.Sc
Profesor Asociado de Ingeniería de Sistemas
Universidad Autónoma de Bucaramanga – UNAB

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA - UNAB
FACULTAD DE INGENIERÍA
MAESTRÍA EN TELEMÁTICA, MODALIDAD INVESTIGACIÓN
GRUPOS: TECNOLOGIAS DE INFORMACION Y PENSAMIENTO SISTEMICO
BUCARAMANGA, COLOMBIA
NOVIEMBRE DE 2018

Le dedico esto a Dios quien me lo ha dado todo.

AGRADECIMIENTOS

Un agradecimiento especial a mi tutor el Dr. Jesús Martin Talavera Portocarrero por sus enseñanzas, dirección, paciencia y motivación a lo largo de este proceso de aprendizaje. Su orientación es invaluable.
Además de mi tutor, quiero agradecer al Profesor José Daniel Cabrera Cruz quien fue parte fundamental para la dirección de este proyecto en el ámbito investigativo y metodológico.

Expreso mi gratitud al Dr. Cesar D. Guerrero y todo el equipo de trabajo del proyecto para la conformación del Centro de Excelencia y Apropiación en Internet de las Cosas (CEA-IoT), quienes me brindaron la oportunidad de obtener la beca para adelantar mis estudios de posgrado. Este trabajo no hubiera sido posible sin la financiación del Ministerio de Tecnología de la Información y Comunicaciones – MinTIC de Colombia, el Departamento Administrativo de Ciencia, Tecnología e Innovación – Colciencias, a través del Fondo Nacional de Financiamiento para la Ciencia, Tecnología y la Innovación Francisco José de Caldas (ID Proyecto: FP44842-502-2015) e instituciones privadas como la Universidad Autónoma de Bucaramanga quienes conforman el CEA IoT.

Por último pero no menos importante, quiero agradecer a mi familia: mi mami, papá y dani *(Little bro)* por el apoyo espiritual, emocional y económico brindado a lo largo de este tiempo de preparación. Los amo.

SWITCH: UN MIDDLEWARE PARA EL DESARROLLO DE APLICACIONES IOT CON INTERFACES BASADAS EN VOZ

Estudiante: Johana Andrea Manrique Hernández
Tutores: Jesús M. Talavera P.[2], José D. Cabrera Cruz[1]
[1]Universidad Autónoma de Bucaramanga, [2]Ingeniero *Nuance Communications*
Maestría en Telemática en Modalidad Investigación
Universidad Autónoma de Bucaramanga – UNAB
Noviembre de 2018

RESUMEN

Internet se está desarrollando como un nuevo paradigma conocido como Internet de las Cosas (en inglés, *Internet of Things - IoT*) donde las personas y cosas cotidianas se conectan a Internet. Las cosas necesitan de interfaces digitales para facilitar la comunicación entre humanos y maquinas. Las interfaces (mundo virtual) se deben proporcionar haciendo uso de una amplia gama de aplicaciones que abordan necesidades específicas de los diferentes dominios de aplicación. Sin embargo, al ser IoT un paradigma complejo, el desarrollo de estas aplicaciones se convierte en un desafío tecnológico.

Actualmente, IoT está impactando la forma como se vive, pero la interacción entre hombre-máquina y máquina-máquina todavía está lejos de ser no intrusiva para el ser humano debido a que no se relacionan de manera natural. Para lograr esto, es necesario hacer uso de las capacidades básicas humanas como por ejemplo la voz, la cual ocurre naturalmente, pero aún no es ampliamente utilizada como parte del paradigma IoT.

Con base en lo anterior, se propuso el diseño de SWITCH, una plataforma *middleware* con potencial de investigación que oculta la complejidad en el desarrollo de aplicaciones IoT, abordando los requisitos funcionales y no funcionales básicos que IoT demanda. SWITCH contiene módulos para el reconocimiento del habla, los cuales a través de las aplicaciones proveen interfaces de voz a los usuarios para facilitar la interacción natural con las cosas cotidianas.

Palabras clave: Internet de las cosas, *Middleware*, Reconocimiento del habla, Telemática y Computación Ubicua.

SWITCH: A MIDDLEWARE FOR DEVELOPING IOT APPLICATIONS WITH VOICE-BASED INTERFACES

Student: Johana Andrea Manrique Hernández
Advisors: Jesús M. Talavera P.[2], José D. Cabrera Cruz[1]
[1]Universidad Autónoma de Bucaramanga, [2]Ingeniero *Nuance Communications*
Master in Telematics (research-based)
Universidad Autónoma de Bucaramanga – UNAB
November 2018

ABSTRACT

Internet is being developed as a new paradigm known as Internet of things where people and daily things are connecting to Internet. Things need digital interfaces to facilitate communication between human-machine. The interfaces (virtual world) must be provided making use of a wide range of applications that address specific needs for different domains. However, since IoT is a complex paradigm, the development of these applications becomes a challenging task.

Currently, IoT is impacting the way how we live but the interaction between human-machine and machine-machine is still far from being non-intrusive for people because they are not related in a natural way. To achieve this concern, it is necessary to make use of basic human capabilities such as voice, which occurs naturally, but is not yet widely used as part of the IoT paradigm.

Based on the above, the SWITCH design was proposed, a middleware platform with research potential for hiding the complexity in the development of IoT applications, addressing the basic functional and non-functional requirements that IoT demands. SWITCH contains modules for speech recognition for providing voice interfaces to facilitate natural interaction with things.

Keywords: Internet of things, Middleware, Speech Recognition, Telematics, Ubiquitous computing.

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

1. INTRODUCCIÓN

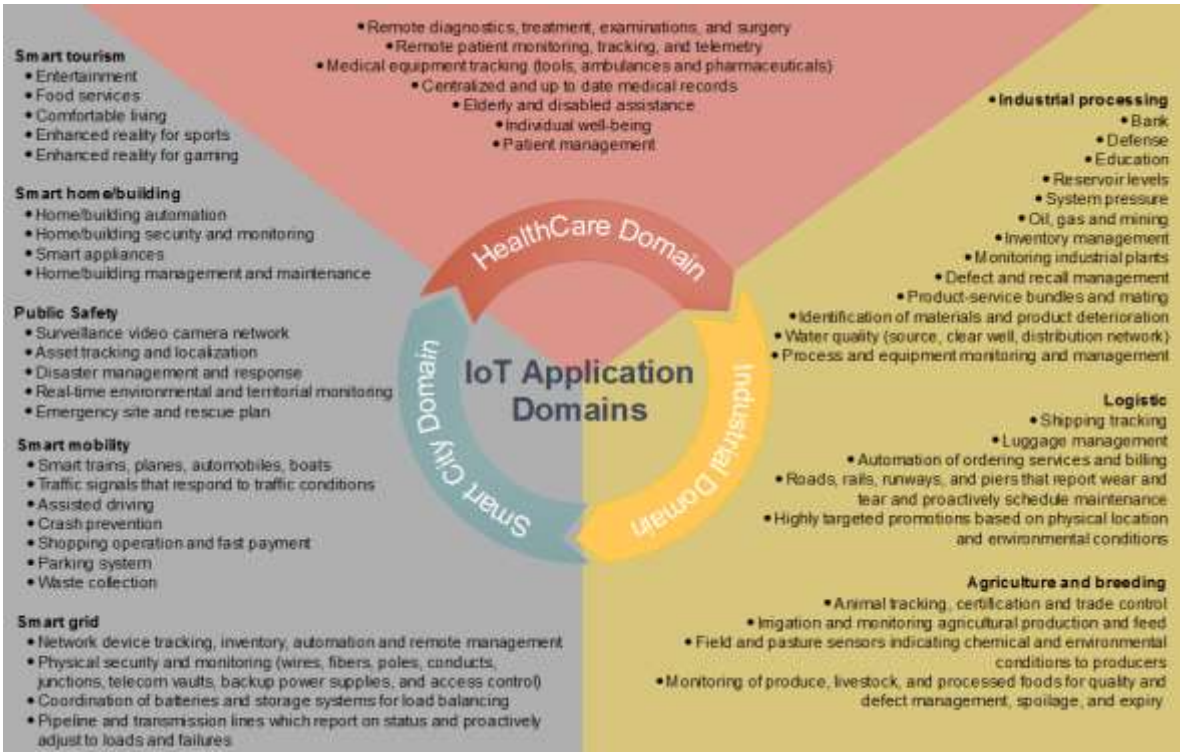
Internet está evolucionando hacia un nuevo paradigma conocido como Internet de las Cosas *(en inglés, Internet of Things - IoT)*, el cual se enfoca en conectar no sólo a las personas, sino también a las cosas cotidianas, proporcionándoles interfaces digitales que facilitan la interacción entre hombre-máquina y máquina-máquina (Atzori, Iera, & Morabito, 2010; Gubbi, Buyya, Marusic, & Palaniswami, 2013; Internet Society, 2015; I. Lee & Lee, 2015).

La *International Telecommunication Union – ITU* (2012) describe a IoT como la infraestructura global para la sociedad de la información que promueve servicios avanzados a través de la interconexión de cosas físicas/virtuales para la interoperabilidad de las Tecnologías de la Información y Comunicación (TIC) presentes y futuras. Por lo tanto, IoT implica la comunicación con cualquier cosa (entre dispositivos, aplicaciones, personas), en cualquier tiempo y lugar, a través de cualquier red. Actualmente, la proliferación de dispositivos conectados a Internet en relación con el número de personas conectadas es mayor. La empresa global de telecomunicaciones *CISCO Systems Inc* (2011) proyectó datos relacionados con el número de dispositivos conectados a Internet en comparación con la población humana mundial: A mediados del 2000, 500 millones de dispositivos estaban conectados a Internet, un promedio de 0,08 dispositivos por persona; en el 2008, el número de dispositivos conectados a Internet superó el número de personas en el mundo. Se estima que para el año 2030 se conecten a Internet más de 500 mil millones de dispositivos, un promedio de 8 dispositivos por persona (CISCO, 2016).

Con IoT, cualquier cosa cotidiana puede ser equipada con dispositivos como los sensores, permitiendo que estos generen e intercambien datos con la mínima intervención humana, transformando muchos aspectos de la manera como se vive: personal, social, salud, logístico, industrial, por citar algunos (Internet Society, 2015). En la Figura 1 se presenta una síntesis de los dominios de aplicación IoT propuestos por Atzori et al. (2010), Borgia (2014) y el mapa de dominios usado por el proyecto de la *IoT-Architectural Reference Model* (2016). Los dominios están divididos en tres clases principales, Salud *(HealthCare Domain)*, Industria *(Industrial Domain)* y Ciudades inteligentes *(Smart City Domain)*. Cada uno de ellos tiene subdominios con sus respectivas características.

Como se observa en la Figura 1, IoT tiene muchos dominios de aplicación. Cada dominio tiene requisitos genéricos que se comparten entre ellos y específicos según lo requiera la aplicación. Para evitar sobre carga de costos en cuanto a los recursos económicos, personal y tiempo de desarrollo, se sugiere la implementación de plataformas que faciliten el desarrollo de aplicaciones IoT para los diferentes dominios (Abdmeziem, Tandjaoui, & Romdhani, 2016; Borgia, 2014; Gubbi et al., 2013).

Figura 1. Dominios de aplicación IoT

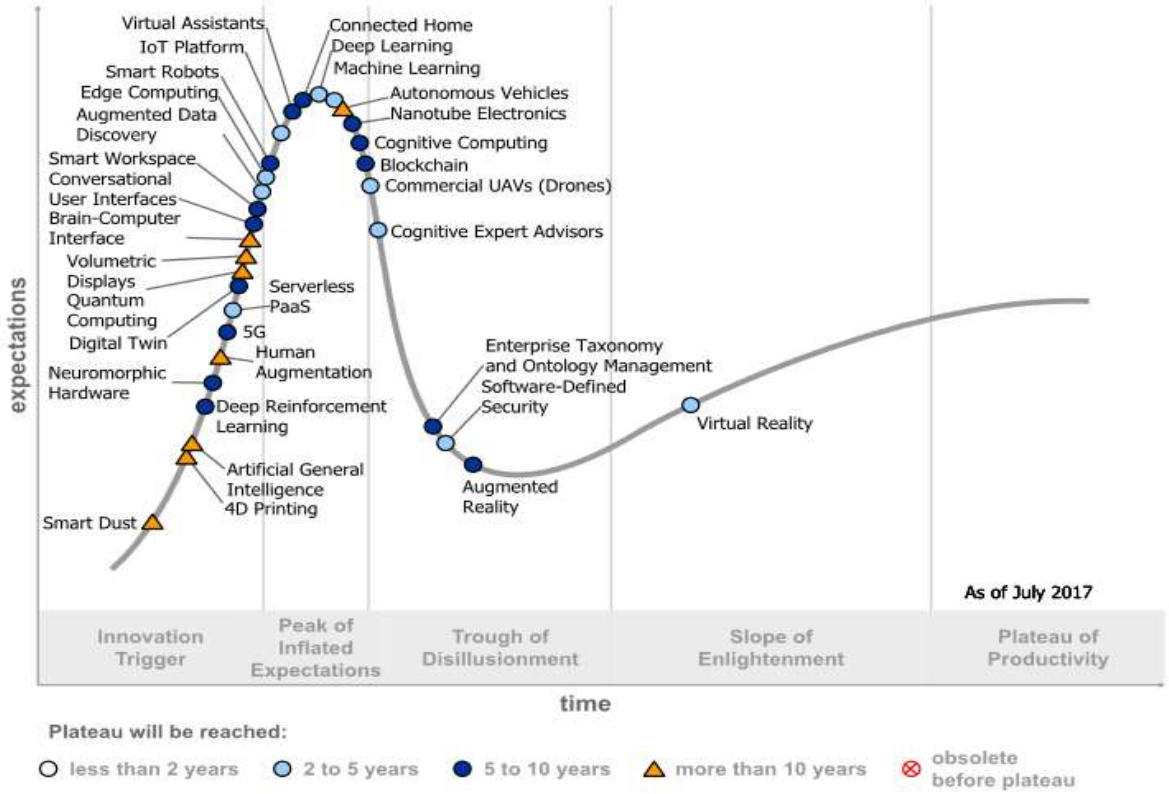


Fuente: Adaptado de Atzori et al. (2010), Borgia (2014) e IoT-Architectural Reference Model (2016)

La Organización Internacional de Normalización *(en inglés, International Organization for Standardization - ISO)* (2010) define una plataforma como una colección de componentes *hardware* y *software* que son necesarios para operar herramientas de ingeniería de *software* asistida por computadoras. *Gartner Inc* (2017) describe que el desarrollo de plataformas IoT será adoptado de forma general por las organizaciones en los próximos 2 a 5 años (ver Figura 2). Si se compara esta predicción con la del año anterior (Gartner Inc, 2016), se redujo la cantidad de años a la mitad. Esto significa que el desarrollo de plataformas IoT está en constantemente crecimiento.

IoT no solo está transformando el estilo de vida de las personas, sino también la estrategia comercial de las empresas. La conexión de dispositivos a Internet está generando en el mundo más de 3 mil millones de dólares provenientes de los usuarios finales y procesos industriales (Hui, 2014). De acuerdo a un estudio realizado por la *International Data Corporation – IDC* a 2.300 ejecutivos en 15 países, el 48% de los encuestados ya implementaron soluciones IoT en sus empresas afirmando que estas son estratégicas para su crecimiento comercial (CISCO, 2016).

Figura 2. Ciclo de tecnologías emergentes, 2017



Fuente: (Gartnet Inc, 2017)

Solamente el mercado mundial para el desarrollo de plataformas IoT produjo ganancias de 298 millones de dólares en el 2015, proyectando un crecimiento anual del 33%. Se espera que el desarrollo de este tipo de plataformas sea uno de los segmentos del mercado con más rápido crecimiento en los próximos años, logrando oportunidades de negocio que representen alrededor de 1.6 mil millones de dólares para el 2021 (IoT Analytics, 2016).

De acuerdo con Mineraud et al. (2016), una plataforma IoT es definida como el *middleware* y la infraestructura que permite a los usuarios finales interactuar con las cosas inteligentes. Desde la concepción de IoT se ha potenciado la implementación de plataformas *middleware* que facilitan la creación de aplicaciones para los diferentes dominios de IoT las cuales tienen un gran impacto no sólo a nivel tecnológico sino también económico, y continuarán en constante desarrollo. (Chelloug & El-Zawawy, 2017). Esta necesidad nace dado que al ser IoT un paradigma tan complejo, el desarrollo de una aplicación enfrenta una variada lista de dificultades (Nitti, Pilloni, Colistra, & Atzori, 2016). Algunas de ellas se describen a continuación.

No existe una configuración estándar para desarrollar aplicaciones IoT (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015; Wortmann, Flüchter, & others, 2015). Borgia (2014) explica dos tipos de enfoques para el desarrollo de aplicaciones IoT: *i)* enfoque vertical, donde cada aplicación se construye usando su propia infraestructura TIC y dispositivos dedicados, es decir, sus recursos no son reutilizados y, *ii)* enfoque horizontal, donde una plataforma genérica *(middleware)* gestiona la red y los servicios abstrayéndolos a través de una amplia gama de fuentes de datos, permitiendo que las aplicaciones funcionen correctamente. En particular, este enfoque gestiona la (re) utilización de recursos.

Aunque el paradigma de IoT es relativamente nuevo, no es una nueva tecnología en sí misma. IoT es un ecosistema integrado por varias tecnologías heterogéneas (*hardware* y *software*) las cuales sirven para desarrollar aplicaciones IoT (Mineraud et al., 2016). Algunas de estas tecnologías tienen décadas de existencia. Otras más recientes, han sido creadas bajo los requisitos que IoT exige. En la Figura 3 se observan tecnologías de orden vertical las cuales son usadas para implementar aplicaciones IoT cuyos requisitos son específicos para un dominio en particular. Actualmente, IoT cuenta con cinco tecnologías esenciales, a saber (Al-Fuqaha et al., 2015; I. Lee & Lee, 2015): *(i)* Identificación por radiofrecuencia *(en inglés, Radio Frequency Identification – RFID)*, *(ii)* Redes de sensores inalámbricos *(en inglés, Wireless Sensor Network – WSN)*, *iii)* plataformas *middleware*, *iv)* Computación en nube, y *v)* Aplicaciones IoT de usuario final.

De acuerdo con Patel y Cassou (2015), el desarrollo de una aplicación IoT también involucra el trabajo de varias partes interesadas *(en inglés, stakeholders)* quienes garantizan un proceso de desarrollo más estructurado y mejor definido acorde con todas las capas de su arquitectura. Cada *stakeholder* tiene un rol preciso relacionado con sus habilidades: *i)* El experto en dominios *(en inglés, Domain expert)* entiende todos los conceptos relacionados con IoT y cómo la aplicación es desarrollada considerando las diferentes capas de la arquitectura IoT; *ii)* el diseñador de software *(en inglés, Software designer)* define la estructura de una aplicación IoT especificando los componentes de software y sus relaciones; *iii)* el desarrollador de aplicaciones *(en inglés, Application developer)* define el diseño y uso de algoritmos de acuerdo al lenguaje de programación; *iv)* el desarrollador de dispositivos *(en inglés, Device developer)* tiene una comprensión profunda de las entradas/salidas y protocolos de comunicación entre dispositivos; *v)* el administrador de red *(en inglés, Network manager)* configura el proceso de conexión y conoce el dominio en el que se va a desplegar la aplicación.

De acuerdo a la IDC (2016), la cantidad de dispositivos para IoT disponibles en el mercado global aumentará a los 30 mil millones para el año 2020. Los sensores serán uno de los principales impulsores de la expansión de este paradigma. Estos dispositivos tienen un tamaño relativamente pequeño y son capaces de medir las entradas físicas para transformarlas en datos brutos, que luego se almacenan digitalmente en dispositivos más robustos para su acceso y análisis.

Figura 3. IoT Landscape 2018, aplicaciones verticales

Fuente: Tomado de (Turck, 2018)

La miniaturización de los sensores les ha permitido adaptarse en dispositivos inteligentes, ampliando su capacidad de proceso y almacenamiento, más allá de la medición de datos y el análisis para transmitir información a través de Internet. En el contexto actual, los sensores pueden medir prácticamente cualquier cosa (temperatura, fuerza, presión, posición, humedad, peso) y están siendo integrados desde redes eléctricas, carreteras, hasta dispositivos móviles, portátiles y de seguridad.

Para que los humanos puedan interactuar con estos dispositivos, se requiere el desarrollo de aplicaciones que faciliten esta interacción. Por ende, al aumentar la cantidad de dispositivos usados para IoT, aumentará el desarrollo de aplicaciones para IoT. Actualmente, las personas ya tienen acceso a dispositivos personales que pueden sincronizar con su vida cotidiana. Por ejemplo, algunas aplicaciones de teléfonos inteligentes son usadas para cerrar puertas automáticamente, activar alarmas, entre otras tareas, que realizan usando contraseñas tradicionales (números y letras), sin embargo, esta no es una forma biológicamente intuitiva de demostrar identidad. Un método más natural y seguro vendría es usar sensores que pueden leer atributos personales, como una huella digital, la voz o un ritmo cardíaco. Este es el tipo de interacción hombre-máquina que se puede volver más fácil con IoT. La creciente sofisticación de los sensores integrados en la tecnología hace posible que los dispositivos (cosas) lean, midan y entiendan a los humanos (EY, 2016).

Sin embargo, esta interacción todavía está lejos de ser no intrusiva para el ser humano y es por eso que su investigación continúa en curso (Sundmaeker, Guillemin, Friess, & Woelfflé, 2010; Zhong et al., 2016). Actualmente, IoT está promoviendo la interacción armoniosa entre los humanos y las cosas inteligentes (Zhong et al., 2016). Esta visión no es algo nuevo, ya que fue compartida hace más

de tres décadas por Mark Weiser (1991) donde describe que las personas no percibirían las herramientas digitales a su alrededor, ya que estas se convertirían en cosas cotidianas con las cuales podrían interactuar inconscientemente, es decir, de forma ubicua.

La conectividad ubicua es un requisito de IoT (Zhong et al., 2016) y para cumplirlo, las aplicaciones necesitan soportar un conjunto diverso de dispositivos y protocolos de comunicación: desde sensores capaces de detectar eventos, hasta poderosos servidores *back-end* utilizados para el análisis de datos y la extracción de conocimiento (Buyya & Dastjerdi, 2016). Con el aumento de la cantidad de dispositivos inteligentes y la adopción de nuevos protocolos de Internet como el IPv6 (Khurana, 2017), se espera que IoT se oriente hacia la fusión de redes inteligentes y autónomas con objetos aptos para conectarse a Internet equipados con los elementos de la Computación ubicua *(en inglés, Ubiquitous computing – UbiComp).* Una capacidad humana que facilitaría esa interacción sería la voz, la cual puede ser usada en la implementación de aplicaciones para cumplir con características que describe la UbiComp. Con una adecuada implementación considerando las características técnicas, la voz humana puede proporcionar una experiencia de usuario más flexible, de una manera más económica que los métodos tradicionales como las pantallas digitales (Brown, 2016).

1.1 PROBLEMA DE INVESTIGACIÓN

Con base en lo anterior, se identificaron dos problemas principales los cuales se describen a continuación.

1.1.1 Complejidad en el desarrollo de aplicaciones IoT. El primer problema abordado en este proyecto está relacionado con la complejidad en el desarrollo de aplicaciones IoT. Hay algunas causas que surgen de esta problemática.

- El desarrollo de una aplicación IoT consta de varias capas. Desde un punto de vista arquitectural, una aplicación IoT debe cumplir una serie de requisitos genéricos descritos en cada capa de la arquitectura que sirve como base para realizar el diseño de la aplicación. Además, debe cumplir requisitos específicos dependiendo del dominio al que se dirige la aplicación. Los atributos de calidad son requisitos arquitectónicos transversales.
- Cada capa de una arquitectura IoT tiene requisitos complejos e integra múltiples tecnologías de acuerdo a esos requisitos, dificultando la reutilización del hardware y software.
- Los diferentes stakeholders que participan en el desarrollo de una aplicación, deben tener conocimientos generales en IoT y un conocimiento preciso de la

capa de arquitectura IoT o el dominio en el cual es experto. La participación de varias personas en el desarrollo de una aplicación IoT aumenta los costos de su implementación, margen de error en la integración de las partes si no existe una comunicación asertiva entre ellos.

1.1.2 Las cosas no tienen interfaces digitales. Teniendo en cuenta que el objetivo de IoT es conectar todas las cosas cotidianas a Internet, se necesitan mecanismos que ayuden a las personas interactuar de forma natural con las cosas. Este es el segundo problema abordado en el presente proyecto.

- Las cosas necesitan interfaces que faciliten la interacción entre hombre-máquina y máquina-máquina. Esta interacción debe darse de manera natural y no intrusiva para el ser humano. Por ejemplo, hablar es una capacidad humana que ocurre naturalmente pero aún no es ampliamente utilizada como parte del paradigma de IoT. Actualmente son pocas las aplicaciones IoT que se desarrollan con interfaces basadas en voz que faciliten la interacción con el entorno de los usuarios tal como sugiere la tendencia de UbiComp.

1.2 MOTIVACIÓN

Middleware es una herramienta software que ayuda a ocultar la complejidad y la heterogeneidad de las tecnologías *hardware*, *software* y redes subyacentes, facilitando la gestión de los recursos del sistema a través de la creación de aplicaciones distribuidas mediante la presentación de una interfaz unificada (Hadim & Mohamed, 2006; Razzaque, Milojevic-Jevric, Palade, & Clarke, 2016; Wang, Cao, Li, & Dasi, 2008). Un *middleware* utiliza el enfoque horizontal (Borgia, 2014) para abarcar recursos genéricos y reutilizables, facilitando el desarrollo de aplicaciones para dominios específicos. De esta forma se aborda el primer problema descrito anteriormente.

Actualmente, el desarrollo de *middleware* es un área de investigación muy activa gracias a la utilidad que esta herramienta de *software* brinda facilitando el desarrollo de aplicaciones. A nivel académico y comercial se han propuesto y aplicado muchas soluciones de este tipo, especialmente en los últimos años (Ngu, Gutierrez, Metsis, Nepal, & Sheng, 2017; Razzaque et al., 2016). A continuación se describen algunos tipos de *middleware*.

- Middleware según su dominio de implementación, como ejemplo WSN (Wang et al., 2008), RFID (Al-Jaroodi, Aziz, & Mohamed, 2009), M2M (Kim, Lee, Kim, & Yun, 2014), IoT (Ngu et al., 2017; Razzaque et al., 2016).

- Middleware según su enfoque de diseño, basado en: (i) eventos, donde de forma asíncrona interconecta los componentes de una aplicación en un entorno distribuido y heterogéneo, describiendo un cambio especifico de un estado mediante eventos (Meier & Cahill, 2002) Cada evento se propaga desde un emisor, quien detecta el cambio y termina en un receptor, quien recibe el cambio; (ii) servicios, genera aplicaciones distribuidas bajo la arquitectura orientada a servicios (en inglés, Service Oriented Architecture – SOA) (Papazoglou, Traverso, Dustdar, & Leymann, 2007). Este tipo de middleware tiene funcionalidades adecuadas para desplegar, publicar, descubrir y acceder a servicios en tiempo de ejecución (Issarny et al., 2011); (iii) máquina virtual, proporciona soporte de programación para un entorno de ejecución de aplicaciones seguras a través de la virtualización de su infraestructura, es decir, la aplicación se divide en pequeños módulos independientes distribuidos a lo largo de la red. Cada nodo en la red contiene una máquina virtual que sirve para interpretar el módulo (Costa, Pereira, & Serodio, 2007); (iv) agentes, donde las aplicaciones se dividen en programas modulares para facilitar su distribución a través de la red usando agentes móviles (Mamei & Zambonelli, 2006). Mientras se migra de un nodo a otro, los agentes mantienen su estado de ejecución, lo que facilita el diseño de sistemas descentralizados capaces de tolerar fallas parciales; (v) tuple-space, es una implementación del paradigma de la memoria asociativa para la computación distribuida, donde cada miembro de la infraestructura ocupa un espacio de la tupla local. Cada tupla es una estructura de datos elemental compuesta por una secuencia ordenada de campos (Mottola, Murphy, & Picco, 2006) y; (vi) aplicaciones específicas, se centra en el soporte de la gestión de recursos (calidad del servicio) de una aplicación específica mediante la implementación de una arquitectura que se ajusta a los requisitos de su dominio de aplicación (Borgia, 2014).
- Middleware según su nivel de abstracción que puede ser a nivel local/nodo o nivel global/red (Razzaque et al., 2016).

Basado en lo anterior, el *middleware* es una tecnología con potencial de investigación que puede facilitar el desarrollo de aplicaciones IoT, abordando los requisitos funcionales y no funcionales que IoT demanda (Mineraud et al., 2016; Ngu et al., 2017; Razzaque et al., 2016).

Desde el punto de vista de interacción natural, la multinacional especializada en el desarrollo de tecnología de voz *Nuance Communications Inc*[1] (2016) exploró las preferencias de los usuarios mostrando como resultado que la mayoría de los consumidores (89%) prefieren tener la opción de conversar con sus asistentes personales en lugar de digitar algún tipo de comando. Esta preferencia se debe a que los usuarios quieren sentir una interacción humano-máquina más natural e

[1] Más información en la página Web http://www.nuance.com/

intuitiva, haciendo de esta conexión una vida más fácil. Greg Payne (2014) representante de la *Nuance* afirma que:

> Necesitamos que los dispositivos se adapten a nosotros, en lugar de lo contrario. Esta es la razón por la cual la voz y la computación en la nube serán de vital importancia en las aplicaciones. El reconocimiento del habla, la comprensión del lenguaje natural y el diálogo contextual nos permitirán comprometernos con estos dispositivos a nivel humano, mientras que la nube nos dará acceso a la información, servicios y aplicaciones necesarias para entregarnos una experiencia individualizada (2014, p. 1, mi traducción).

Tabla 1. Prototipos comerciales para el reconocimiento del habla

Nombre	Aplicación comercial	País	Url
Agnitio	Aplicaciones para el reconocimiento de la voz y del habla las cuales tienen como objetivo prevenir el crimen, identificar a los delincuentes y proporcionar evidencia en los tribunales.	España	http://www.agnitio-corp.com
Nuance	Software de reconocimiento de voz, sistemas de dirección de llamadas telefónicas, directorios telefónicos automatizados, sistemas de transcripción médica, reconocimiento óptico de caracteres y software de escaneo.	Estados Unidos	https://www.nuance.com/index.html
Speech Tech Center (SCT)	Tecnologías para soluciones biométricas compactas y de gran escala que sirven para la identificación de oradores en tiempo real.	Rusia	https://speechpro.com/
DAON	Soluciones biométricas como lectores de huellas dactilares, reconocimiento de iris, reconocimiento del habla y de la voz.	Estados Unidos	https://www.daon.com/
Apple	SIRI es una aplicación que utiliza procesamiento del lenguaje natural para responder preguntas, realizar acciones mediante la delegación de solicitudes hacia un conjunto de servicios Web. Además, cuenta con una interfaz de desarrollo para el reconocimiento del habla.	Estados Unidos	https://www.apple.com/es/ios/siri/
Google	El asistente de Google permite interacción a través de la voz natural y es capaz de buscar en Internet, programar eventos y alarmas, ajustar la configuración del hardware en el dispositivo del usuario y mostrar información de la cuenta de Google del usuario.	Estados Unidos	https://assistant.google.com
Amazon	Alexa es un asistente virtual capaz de interactuar a través de la voz, crear listas de tareas pendientes, establecer alarmas, reproducir audiolibros y proveer información en tiempo real del clima, tráfico, entre otros.	Estados Unidos	https://developer.amazon.com/alexa

Fuente: El autor

La empresa internacional de consultoría *Strategy Analytics*[2] (Brown, 2016) proyecta que la voz podría capturar hasta el 12% de las aplicaciones industriales de IoT en 2022 de forma prevalente en las regiones lucrativas de Asia-Pacífico y Norteamérica. Esta proyección es tan solo para el dominio industrial y no cubre el dominio de la salud y las ciudades inteligentes. Esto significa que el desarrollo de este tipo de aplicaciones sigue siendo poco explorado, convirtiéndose en una iniciativa latente de desarrollo. Los actuales y futuros sistemas de reconocimiento del habla se pueden integrar en las aplicaciones, para hacer a IoT mucho más simple, y mucho más accesible para cualquier persona.

La voz es una capacidad humana que puede ser integrada en las aplicaciones IoT a través de los sistemas de reconocimiento del habla *(en inglés, speech recognition*) para proveerle a las cosas interfaces digitales y de esta forma facilitar la interacción entre hombre-máquina y máquina-maquina. Las aplicaciones relacionadas con el reconocimiento del habla están creciendo continuamente y entre sus objetivos se encuentran facilitar la interacción con el humano y proporcionar seguridad. En la Tabla 1 se describen algunas de las empresas que actualmente comercializan prototipos para el reconocimiento del habla.

1.3 PREGUNTA E HIPÓTESIS DE INVESTIGACIÓN

Con base en los problemas planteados en la Sección 1.1, se plantea la siguiente pregunta de investigación: ¿Cómo abordar la complejidad en el desarrollo de aplicaciones IoT de manera que se facilite la interacción con las cosas cotidianas a través de interfaces basadas en voz?

Luego de definir el problema y teniendo en cuenta la información presentada anteriormente, se enuncia la siguiente hipótesis de investigación: Un *middleware* puede facilitar el proceso de desarrollo de aplicaciones IoT que utilizan interfaces de voz para interactuar con las cosas cotidianas.

1.4 OBJETIVOS

Para dar respuesta a la pregunta de investigación y probar la hipótesis, se proponen los siguientes objetivos.

Diseñar un *middleware* que utilice interfaces basadas en voz para facilitar el desarrollo de aplicaciones IoT y la interacción con las cosas cotidianas.

[2] Más información en la página Web https://www.strategyanalytics.com

Para lograr ese objetivo general, se proponen los siguientes objetivos específicos.

- Analizar los requisitos funcionales y no funcionales de un middleware para el desarrollo de aplicaciones IoT que utilicen interfaces basadas en voz, tomando como base arquitecturas de middleware, arquitecturas de referencia IoT y sistemas de reconocimiento del habla (Speech recognition).
- Modelar los módulos relacionados con el reconocimiento del habla de la arquitectura del middleware analizado, utilizando un lenguaje de descripción arquitectural.
- Evaluar los módulos diseñados del middleware mediante la adaptación de un instrumento de valoración para que sea utilizado por expertos en diseño de software.

1.5 ORGANIZACIÓN DEL DOCUMENTO

El documento está organizado en siete capítulos, cada uno incluye una perspectiva que contribuye al diseño de SWITCH. El capítulo 2 presenta un marco referencial de conceptos y el estado del arte donde se describe la brecha de investigación. El capítulo 3 describe el proceso metodológico utilizado para el diseño de SWITCH. El capítulo 4 presenta los requisitos funcionales y no funcionales que requiere un *middleware* genérico para IoT, de los cuales se obtuvieron los requisitos que atiende el diseño de SWITCH. El capítulo 5 presenta el modelado de los requisitos de SWITCH a través de un lenguaje de descripción arquitectural. El capítulo 6 describe la evaluación realizada al diseño de SWITCH y se obtiene un instrumento de evaluación para el diseño de *middleware* IoT. Finalmente, el capítulo 7 contiene las conclusiones, las contribuciones realizadas y proporciona pautas para el trabajo futuro en esta área de investigación.

2 MARCO REFERENCIAL

En este capítulo se describe la base conceptual y teórica que sustenta el proyecto de investigación. En la sección 2.1 se presentan los conceptos relacionados con las palabras clave. La sección 2.2 se presenta la base teórica de Internet de las cosas, middleware y sistemas de reconocimiento del habla. La sección 2.3 describe los trabajos existentes relacionados con el proyecto. La sección 2.4 describe los interesados directos e indirectos del proyecto. La sección 2.5 presenta el marco legal del proyecto.

2.1 MARCO CONCEPTUAL

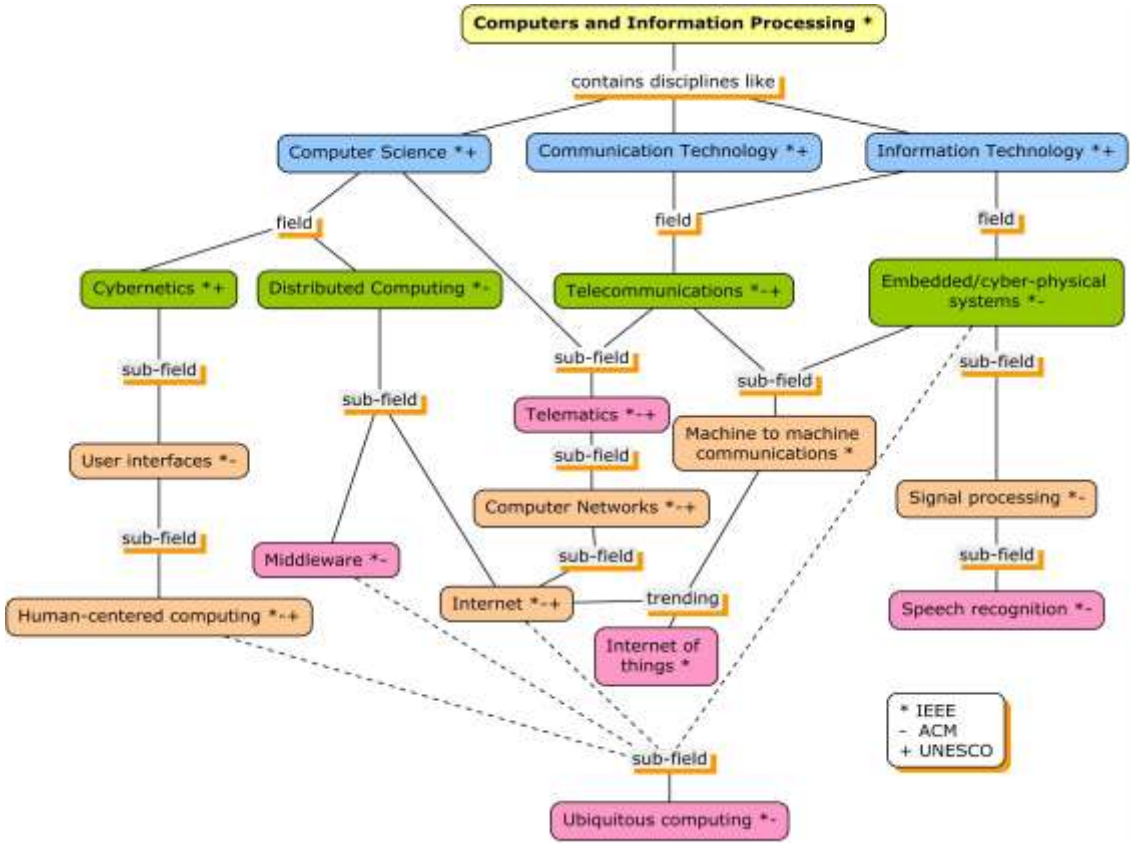
Con el fin de conocer la relación entre las palabras clave de este trabajo (Internet de las cosas, *Middleware*, Reconocimiento del habla, Telemática y Computación Ubicua) se diseña un mapa conceptual (Ver Figura 4). Para lograr esto, se analizaron los términos científicos *(thesaurus)* que estuvieran relacionados con las palabras clave propuestas, implementados por organizaciones académicas como *The Institute of Electrical and Electronics Engineers – IEEE* (2014)*, Association for computing machinery – ACM* (2012) y la *United Nations Educational Scientific and Cultural Organization – UNESCO* (2016).

El mapa se organizó jerárquicamente con base en el análisis de los términos más amplios *(broader term)* y los términos más específicos *(narrow term).* El termino más amplio, Computadoras y procesamiento de información *(Computers and Information Processing)* contiene disciplinas como Ciencias de la Computación *(Computer Science)*, Tecnologías de la comunicación *(Communication Technology)* y Tecnologías de la Información *(Information Technology),* útiles para la manipulación de la información digitalizada obtenida desde diferentes redes/canales de comunicación como por ejemplo las computadoras y otros equipos electrónicos digitales (United Nations Educational Scientific and Cultural Organization, 2016).

A partir de estas disciplinas, nacen campos de investigación como Cibernética *(Cybernetics)*, Computación distribuida *(Distributed computing)*, Telecomunicaciones *(Telecommunications)* y Sistemas físicos integrados *(Embedded/Cyber-physical systems CPS)*. Cibernética se centra en el estudio de las funciones de control automática de los seres vivos y los aplica a sistemas mecánicos y electrónicos diseñados para reemplazarlas (Wiener, 1961). La Computación distribuida es un modelo para resolver la computación masiva conectando computadoras separadas físicamente a través de una red de comunicaciones en forma de clúster (Coulouris, Dollimore, & Kindberg, 2005). Las Telecomunicaciones hacen uso de las TIC para transmitir diferentes tipos de datos. CPS es una integración es una integración entre la computación con procesos

físicos, permitiendo que la próxima generación de sistemas integrados sean más adaptables, eficientes, seguros y confiables (Park, Zheng, & Liu, 2012). Estos campos de investigación se complementan entre sí y forman la base para sub-campos como *Middleware*, Internet de las cosas *(Internet of things)*, Computación ubicua *(Ubiquitous computing)* y Reconocimiento del habla *(Speech recognition)*.

Figura 4. Mapa conceptual de las palabras clave



Fuente: El autor

Un *middleware* es una herramienta software diseñada para ayudar a administrar la complejidad y la heterogeneidad inherentes a las tecnologías *hardware*, *software* y redes subyacentes (Hadim & Mohamed, 2006; Razzaque et al., 2016; Wang et al., 2008). Además, tiene la capacidad de ser personalizable para diferentes escenarios, aplicaciones y entornos. Internet de las cosas es un paradigma que surge de Internet como un tipo particular de Redes de Computadoras *(Computer networks),* a partir de Telemática *(Telematics)* y Comunicaciones máquina a máquina *(Machine to Machine - M2M)*. Gartner Inc (2014) describe la telemática como la implementación de servicios y aplicaciones *(online/offline)* que usa los

sistemas informáticos y las telecomunicaciones como resultado de la unión de ambas disciplinas. M2M hace referencia a la comunicación directa entre dos dispositivos a través de un canal de comunicación inalámbrico o cableado (Kim et al., 2014). Esta tecnología proporciona la interconexión de dispositivos como los sensores y actuadores, para el control remoto de procesos industriales. La ITU (2012) describe a IoT como la infraestructura global para la sociedad de la información que promueve la prestación de servicios avanzados mediante la interconexión de objetos físicos/virtuales para la interoperabilidad de las TIC actuales y futuras.

El desarrollo de la computación centrada en el ser humano ha permitido crear sistemas informáticos que se ajustan a las capacidades y prácticas humanas mediante la explotación y mejora de los métodos de programación de Inteligencia Artificial (Dino, 2008). Este concepto que integra la computación con los movimientos naturales del ser humano y su interacción con el medio físico, es una visión descrita por Mark Weiser (1991) hace más de dos décadas conocida como Computación Ubicua *(Ubiquitous computing - UbiComp)*. UbiComp tiene una conexión indirecta con las otras palabras clave, como se observa en la Figura 4. La interacción natural hombre-máquina se ha construido gracias a la forma innovadora de cómo las aplicaciones *(hardware-software)* se colaboran, comunican e interactúan entre sí. Por ejemplo, integrar el uso de la voz en aplicaciones, ofrece un método versátil para proporcionar y facilitar la interacción y comunicación entre el humano y la maquina través de los sistemas de reconocimiento del habla y la voz (Brown, 2016).

Según la literatura científica, existe diferencia desde un punto de vista conceptual entre los sistemas para el reconocimiento del habla *(Speech recognition)* y los sistemas para el reconocimiento de la voz *(Voice recognition)*. El reconocimiento de voz es el método automatizado de confirmar la identidad de un individuo basado en la voz, es decir, es importante saber quién es el que habla (Shin & Jun, 2015). El reconocimiento del habla es la tarea de convertir la señal de voz en los flujos de símbolos, es decir, palabras o fonemas que representan la información contenida en el enunciado, sin importar quién es el individuo que habla (Sinha, Agrawal, & Jain, 2013). Cabe resaltar que algunos autores en la literatura científica usan ambos términos como sinónimos.

2.2 MARCO TEÓRICO

En esta sección se presenta la fundamentación teórica de este proyecto, la cual se enfoca en la ingeniería del *software*, Internet de las cosas, *middleware* y sistemas para el reconocimiento del habla. En cuanto a la ingeniería del *software*, se hace énfasis en la importancia de los requisitos en el desarrollo de un *software* o sistema

de calidad. Internet de las cosas, *middleware* y reconocimiento del habla hacen énfasis en su conceptualización y origen.

2.2.1 Ingeniería del *software*. De acuerdo a la IEEE (1990), la ingeniería del software es un término muy amplio que abarca el estudio y la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de *software*. De esta forma, la ingeniería del software no solo incluye la producción de *software* sino también la investigación de los enfoques descritos en la definición anterior.

El proceso de desarrollo de un *software* implica que las necesidades de los usuarios se traduzcan en requisitos de software, para modelarlos teniendo en cuenta los escenarios, las clases y el comportamiento, para luego implementar el modelado en código fuente, el cual será probado y revisado hasta que se obtenga un *software* listo para su uso operativo (IEEE, 1990). El proceso de producción de *software* se considera una ciencia disciplinada sometida al estudio científico y objetivada en técnicas y métodos mayoritariamente aceptados por la comunidad profesional en virtud de la experiencia acumulada (S. Sanchez, Angel Sicilia, & Rodriguez, 2012).

2.2.1.1 Ingeniería de requisitos. De acuerdo al *Software Engineering Body of Knowledge* – SWEBOK (2014), la ingeniería de requisitos de *software* se encarga del levantamiento, análisis, especificación y validación de los requisitos del usuario; así como la gestión de los requisitos durante todo el ciclo de vida del producto *software*. Loucopoulus y Karakostas (1995) consideran la ingeniería de requisitos como una combinación de tres procesos concurrentes e interactuantes, a saber: (i) obtener conocimiento relacionado de un problema, (iii) garantizar la validez de dicho conocimiento y (iii) especificar el problema de una manera formal.

Un requisito es una propiedad que debe ser exhibida por algo, ya sea un *software* o sistema, con el fin de resolver un problema en el mundo real; y es una combinación compleja de varias personas que de una manera u otra están implicadas o conectadas con esta característica en el entorno en el que el sistema o *software* operará (IEEE Computer Society, 2014). Existen dos tipos de requisitos del *software,* a saber: los requisitos funcionales, son aquellos servicios o funcionalidades que el sistema debe proveer, cómo debería reaccionar a entradas particulares y cómo debería comportarse en situaciones específicas (Sommerville, 2011); Los requisitos de calidad o no funcionales, describen cómo el *software* hará sus funcionalidades, basado en atributos de calidad (Adams, 2015).

2.2.1.2 Modelado de los requisitos. Da como resultado la especificación de las características operativas del software, indica la interfaz de éste y otros

elementos del sistema, y establece las restricciones que limitan el software (Pressman, 2010).

Un modelo es una representación de un proceso, dispositivo o concepto del mundo real; también puede ser definido como una abstracción semánticamente cerrada de un sistema o una descripción completa de un sistema desde una perspectiva particular (ISO/IEC/IEEE, 2010). A través de un lenguaje de modelamiento se puede realizar esa abstracción. Un lenguaje de modelamiento es «cualquier lenguaje de computadora gráfico o textual que aprovisione el diseño y la construcción de estructuras y modelos siguiendo un conjunto sistemático de reglas y *frameworks*» (Techopedia, 2017).

De acuerdo a Talavera Portocarrero (2016), existen dos enfoques para el modelado de *software*, a saber: *(i)* lenguajes de descripción de arquitecturas (en inglés, *Architecture Description Languages – ADL)*; y *(ii)* lenguajes de modelado estándar (en inglés, *Object Management Group – OMG*. El lenguaje de modelamiento unificado (en inglés, *Unified Modeling Language – UML)* y el lenguaje de modelado de sistemas – SysML son ejemplos de OMG. El autor afirma que en la práctica, UML2 es una ADL semi-formal.

La acción de modelar los requisitos da como resultado uno o más de los siguientes tipos de modelo, a saber: *(i)* modelado basado en escenarios, se enfoca en el punto de vista de distintos stakeholders del sistema; *(ii)* modelado de datos, que ilustran el dominio de información del problema; *(iii)* modelado orientado a clases, representan clases orientadas a objetos y la manera en que ellas colaboran para cumplir con los requisitos; *(iv)* modelado orientado a flujos, representan los elementos funcionales del sistema y; *(v)* modelado de comportamiento, ilustran el modo en el que se comparta el software como consecuencia de eventos externos.

2.2.2 Internet de las Cosas. El término IoT ha sido ampliamente utilizado para describir la conexión de las cosas cotidianas a Internet a través de dispositivos con capacidad computacional (Gubbi et al., 2013). Las soluciones IoT se pueden aplicar a diferentes dominios, como se observa en la Figura 1. Borgia (2014, p. 8) describe que no todas las aplicaciones IoT tienen el mismo nivel de madurez. Algunas aplicaciones del dominio industrial, ya hacen parte de la vida cotidiana porque son sencillas e intuitivas para el usuario. Otras como las del dominio de la salud están todavía en una fase experimental, ya que requieren una mayor cooperación entre los diversos *stakeholders*. Las aplicaciones que pertenecen al dominio de las ciudades inteligentes son las más futuristas puesto que se encuentran en una etapa temprana de desarrollo. Aunque el concepto de IoT es relativamente nuevo, la idea de monitorizar y controlar dispositivos a través de computadoras y redes ha existido hace décadas, y por eso IoT suele confundirse con otras tecnologías (Manrique, Rueda-Rueda, & Portocarrero, 2016).

El primer satélite artificial "Sputnik I" fue lanzado en 1957 para realizar telemetría desde el espacio. Este hecho marcó la historia de las telecomunicaciones en todo el mundo. En 1970 aparecieron los sistemas de vigilancia remota para los medidores de electricidad. En la década de los 80, las redes de control de supervisión y adquisición de datos - SCADA fueron consideradas como la primera generación de comunicaciones M2M (Igure, Laughter, & Williams, 2006). En 1995, la empresa internacional *Siemens*[3] desarrolló el primer teléfono celular para ser utilizado en aplicaciones industriales M2M que permitía la comunicación inalámbrica entre máquinas (Kim et al., 2014). Los avances de investigación en redes de computadoras y M2M dio paso al desarrollo de un campo emergente en red de sensores conocido como *Wireless Sensor Network - WSN*, ofreciendo una amplia gama de aplicaciones en tiempo real con la capacidad de proporcionar una interfaz digital a las cosas del mundo real (ISO/IEC JTC 1, 2009).

A través del establecimiento del protocolo de Interconexión de Sistema Abierto – OSI (1990) se inició el desarrollo de servicios de comunicación para la conexión de redes informáticas de manera abierta en un ámbito global, conocido como Internet. Berners-Lee et al. (1992) desarrollaron un sistema de distribución de documentos hipertextuales que estaba interconectado con la arquitectura de Internet, dando inicio a la *World Wide Web*. Desde su creación, tanto la Web como Internet han estado en constante evolución. Actualmente, IoT hace parte de la siguiente evolución de Internet.

El origen de IoT se atribuye a los miembros del Centro de Auto-identificación en el Instituto de Tecnología de Massachusetts - MIT alrededor del año 2000 (Internet Society, 2015). Los investigadores desarrollaron una red de objetos con Código de Producto Electrónico *(en inglés, Electronic product code – EPC)* para identificar y supervisar el flujo en una cadena de suministro de forma automática. Este código se almacenó en un circuito integrado en forma de etiqueta para ser leído por *Frequency Identification and the Electronic Product Code – RFID* (Sarma, Brock, & Engels, 2001). La tecnología RFID es considerada como uno de los motores de IoT al igual que las comunicaciones M2M (Sundmaeker et al., 2010). La ITU (2005) publicó la primera definición formal de IoT, describiéndolo como una nueva dimensión añadida al mundo de las TIC, donde cualquier persona puede tener conectividad con cualquier cosa desde cualquier momento y en cualquier lugar. Actualmente, el desafío consiste en conectar no sólo a las personas, sino también las cosas cotidianas a Internet, de tal manera que se puede vincular el mundo real con el ciberespacio. Las cosas utilizan las TIC existentes y futuras para crear un espacio digital/virtual e interactuar entre sí y con otras entidades (humanos, *software*, *hardware*) a través de interfaces bien definidas (Gubbi et al., 2013). Las aplicaciones IoT juegan un rol importante en la construcción de estas interfaces.

[3] Más información en la página Web https://www.siemens.com/global/en/home.html

Como se mencionó en el capítulo 1, desarrollar e implementar aplicaciones IoT trae consigo desafíos debido a que: implica diferentes dominios, existen diferentes tecnologías heterogéneas (WSN, RFID, M2M, *middleware*, computación en la nube), *stakeholders* (Patel & Cassou, 2015), protocolos y capas del entorno de desarrollo (Nitti et al., 2016). Las empresas tecnológicas han centrado su atención en las actividades y servicios generales que ofrece y demanda IoT. Algunas de sus tecnologías están dedicadas a IoT y otras tienen múltiples usos (Al-Fuqaha et al., 2015).

2.2.3 ***Middleware***. Es una herramienta *software* que abstrae la complejidad entre la capa de aplicaciones y dispositivos, permitiendo que el desarrollador de una aplicación se concentre en la tarea específica que va a resolver, sin distraerse con los problemas subyacentes de comunicación, conexión, seguridad, por mencionar algunos (Hadim & Mohamed, 2006; Wang et al., 2008). Un *middleware* facilita y coordina aspectos de procesamiento colaborativo (Razzaque et al., 2016).

El *middleware* ofrece servicios comunes para el desarrollo ágil de aplicaciones que soporten la heterogeneidad entre los dispositivos de comunicación y computación, y la interoperabilidad dentro de las diversas aplicaciones y servicios que se ejecutan en esos dispositivos. Específicamente, construir un *middleware* para IoT se requieren un conjunto de requisitos genéricos a nivel arquitectural y de servicios que este debe cumplir (Razzaque et al., 2016):

- A nivel arquitectural, un *middleware* debe proveer una *Application Programming Interface - API* para dar soporte a los desarrolladores de aplicaciones IoT *(programming abstraction)*, interoperabilidad *(interoperability)*, conciencia del contexto *(context-aware)*, adaptabilidad *(adaptability)*, autonomía *(self-governed)*, servicios distribuidos *(distributed services*).
- A nivel de servicios, un *middleware* para IoT debe soportar el descubrimiento de recursos *(resource discovery)*, escalabilidad *(scalability)*, confiabilidad *(reliability)*, disponibilidad *(availability)*, seguridad *(security)*, fácil desarrollo *(ease-of-deployment)*, la administración de: recursos *(resource management)*, código *(code management)*, eventos *(event management)* y datos *(data management)*.

2.2.4 Reconocimiento del habla. Sistema que permite el reconocimiento y la traducción del lenguaje hablado en texto a través de las computadoras (Sinha et al., 2013). También es llamado como *automatic speech recognition - ASR*, reconocimiento del habla por computador o *speech to text – STT.* Un ASR convierte una señal de voz en una representación textual, es decir, una secuencia de las

palabras por medio de un algoritmo implementado en un módulo de software o hardware (Besacier, Barnard, Karpov, & Schultz, 2014).

El habla es el principal medio de comunicación entre las personas. Existen varios tipos de habla natural: el habla deletreada (con pausas entre letras o fonemas), habla aislada (con pausas entre palabras), habla continua (cuando un hablante no hace pausas entre palabras), habla espontánea (dialogo, discusión) (Besacier et al., 2014). Automatizar la capacidad del habla humana para facilitar la interacción entre el hombre-máquina ha atraído la atención de los investigadores en las últimas seis décadas (Juang & Rabiner, 2005).

Alexander Graham Bell y otros (1881) inventaron un dispositivo de grabación que utilizaba un cilindro giratorio con un revestimiento de cera que respondía a la presión del sonido entrante. Basado en esta invención, en 1888 se comienzan a fabricar máquinas para la grabación y reproducción de sonido en entornos de oficina. En 1907 la empresa *Columbia Graphophone Co* adquirió la patente de esa invención con el término *Dictaphone*. Al mismo tiempo, Thomas Alva Edison inventó el fonógrafo utilizando un cilindro de papel de estaño, que posteriormente se adaptó a cera, denominado como *Ediphone*. El propósito de estos dispositivos era registrar el dictado de notas y cartas para una persona que cumplía funciones de secretaria, que luego los escribía fuera de línea. Este concepto de "mecanización de oficinas" a principios del siglo XIX dio lugar a la creación de la máquina de escribir eléctrica para responder y transcribir directamente la voz de una persona sin tener que lidiar con la molestia de grabar y manejar el discurso en cilindros de cera u otros medios de grabación.

A medida que se creaban dispositivos para capturar la voz humana, se fueron diseñando los ASR que fueron guiados principalmente por la teoría de la fonética acústica, la cual describe los elementos fonéticos del habla (los sonidos básicos del lenguaje) e intenta explicar el modo de articulación utilizado para producir el sonido en diversos contextos fonéticos (Besacier et al., 2014). Davis et al. (1952) construyeron un sistema para el reconocimiento de dígitos aislados para un único altavoz, utilizando las frecuencias formantes medidas (o estimadas) durante las regiones vocales de cada dígito. Estas trayectorias sirvieron como patrón de referencia para determinar la identidad de los dígitos desconocidos que fueran más adecuados a lo que se hablaba.

A inicios de los años 60, varios laboratorios Japoneses demostraron su capacidad de construir *hardware* y *software* específico para realizar tareas de reconocimiento del habla. Entre ellos se destacaron los sistemas para el reconocimiento de vocales (Suzuki & Nakata, 1961), reconocimiento de fonemas (Sakai & Doshita, 1962) y reconocimiento del digito (Nagata, Kato, & Chiba, 1964). A comienzo de los años 70, los investigadores realizaron trabajos aplicando la tecnología de reconocimiento de patrones fundamentales del habla, basada en los métodos *Linear Predictive Coding – LPC* (Itakura, 1975; L. Rabiner, Levinson, Rosenberg, & Wilpon, 1979).

Durante ese periodo, Tom Martin fundó la primera compañía comercial de reconocimiento de voz llamada *Threshold Technology, Inc*. y desarrolló el primer producto comercial ASR llamado *VIP-100 system*. El sistema fue utilizado por las firmas fabricantes de placas frontales de televisión para el control de calidad, por *FedEx* para la clasificación de paquetes e influyó en el *Advanced Research Projects Agency - ARPA* del Departamento de Defensa de los Estados Unidos. Por su parte, la multinacional IBM[4] liderada por Fred Jelinek, creó una máquina de escribir activada por voz, cuya función principal era convertir una frase hablada en una secuencia de letras y palabras que se podían mostrar en una pantalla o escribir en papel (Jelinek, Bahl, & Mercer, 1975). El sistema de reconocimiento llamado Tangora era esencialmente un sistema dependiente del hablante. Técnicamente, el sistema se centró en el tamaño del vocabulario de reconocimiento y la estructura del modelo de la gramática, que estaba representada por reglas sintácticas estadísticas que describían la probabilidad de la secuencia de símbolos de lenguaje (fonemas o palabras) que podrían aparecer en la señal de voz.

A partir de ahí, los ASR se comenzaron a clasificar según el tamaño del léxico usado para el reconocimiento de las palabras habladas (Whittaker, 2000), que pueden ser de 1.000 términos (pequeño), 10.000 términos (mediano), 100.000 términos (grande) o mayor a 100.000 términos (muy grande). Entre más grande sea el léxico que se usa en los ASR, más se flexibiliza el uso del lenguaje, logrando un mayor dominio del mismo. A partir de los años 80, los ASR comenzaron a usar los modelos ocultos de Markov *(hidden Markov model)* para tratar la variabilidad temporal del habla. Otros sistemas se han construido utilizando diversas técnicas de *machine learning* como redes neuronales profundas *(Deep Neural Networks),* redes bayesianas dinámicas *(Dynamic Bayesian Networks),* máquinas de vectores de soporte *(Support Vector Machines - SVM)* (Juang, Hou, & Lee, 1997; Juang & Rabiner, 2005; McCulloch & Pitts, 1990; L. R. Rabiner & Juang, 2004).

A partir del siglo XXI, las tecnologías de reconocimiento de voz comenzaron a entrar de forma reiterativa en el mercado, beneficiando a los usuarios en sus actividades cotidianas. Por ejemplo, la multinacional estadounidense especializada en el desarrollo de tecnología de voz *Nuance Communications, Inc* desarrolla desde comienzos del último siglo a *Dragon NaturallySpeaking*[5] un *software* para el reconocimiento del habla. *Dragon* tiene tres funciones principales: reconocimiento de voz en dictado con transcripción a texto escrito, reconocimiento de comandos hablados y transcripción de texto a voz. La multinacional estadounidense *Apple*[6] integra en sus dispositivos a partir del 2010, una aplicación con funciones de asistente personal llamada *Siri*. Esta aplicación utiliza procesamiento de lenguaje natural para responder preguntas, hacer recomendaciones y realizar solicitudes en

[4] Más información en la página Web https://www.ibm.com/us-en/
[5] Más información en la página Web https://www.nuance.com/dragon.html
[6] Más información en la página Web https://www.apple.com/

diferentes servicios Web. *Google Assistant*[7] es un asistente personal virtual desarrollado por Google en 2016 que interactúa con las personas a través de la voz y el teclado. Esta aplicación puede buscar en Internet, programar eventos y alarmas, mostrar información de la cuenta de *Google* del usuario.

2.3 ESTADO DEL ARTE

Se realizó una revisión de la literatura para conocer estudios primarios que abordan diseños de *middleware* para IoT, identificar sus principales características y cuáles de ellos incluyen módulos para el reconocimiento del habla. A través de esta información se puede extraer la arquitectura, requisitos, enfoque de diseño y ejemplos relacionados con este tipo de plataforma, los cuales sirven como parámetros de entrada para el modelado del *middleware* propuesto en el presente trabajo. Además, si estos *middleware* tienen reconocimiento del habla nos permite conocer su estructura y cómo funcionan. En caso que no tengan o la información sea limitada, se convierte en una oportunidad de investigación latente.

2.3.1 Planeación. Para esto, se definieron las siguientes preguntas de la revisión:

- ¿Cómo está diseñado el middleware para IoT?
- ¿El diseño del middleware para IoT incluye módulos para el reconocimiento del habla?

Para dar respuesta a estas preguntas se definió la siguiente cadena de búsqueda, la cual fue consultada en dos bibliotecas digitales en línea: SCOPUS[8] e IEEE[9].

TITLE-ABS-KEY (("Internet of things" OR IoT) AND middleware) AND PUBYEAR > 2004

Se tuvieron en cuenta algunos criterios de inclusión (CI) y exclusión (CE) para reducir progresivamente el número de estudios encontrados en la etapa de búsqueda, a una colección apropiada de estudios primarios y secundarios de alta calidad que fueran relevantes para responder a las preguntas planteadas en la revisión.

[7] Más información en la página Web https://assistant.google.com/
[8] Más información en la página Web https://www.scopus.com/
[9] Más información en la página Web http://ieeexplore.ieee.org/Xplore/home.jsp

- CI1: El estudio fue publicado entre 2005 y 2018. Se establece a partir del 2005 dado que en ese año la ITU (2005) dio a conocer la primera definición formal de IoT.
- CI2: El estudio contiene información relacionada de middleware para IoT en el título, resumen y palabras clave.
- CI3: El estudio es un artículo, libro, capítulo de libro o una conferencia (conference proceedings).
- CI4: El estudio primario fue escrito en inglés.
- CI5: El estudio es de tipo estado del arte, diseño o implementación.
-
- CE1: No se tiene acceso a todo el estudio primario.
- CE2: El estudio primario ya fue recuperado.
- CE3: El estudio primario contiene información trivial lo que dificulta responder a las preguntas planteadas.

2.3.2 Conducción. Se recuperaron 1288 estudios primarios: 994 de SCOPUS y 294 de IEEE. Inicialmente fueron seleccionados en total 100 basados en los criterios de inclusión y exclusión. Una vez recuperados estos documentos, se realizó una revisión más detallada del *full-text* para conocer está diseñado el *middleware* propuestos. En total se seleccionaron 19 estudios. Los primeros 6 son estudios secundarios que describen revisiones del estado del arte de *middleware* para IoT, como se observa en la Tabla 2.

Tabla 2. Estudios secundarios seleccionados para el estado del arte

ID	Título	Referencia
ID1	Middleware for Internet of Things: Survey and Challenges	(Chelloug & El-Zawawy, 2017)
ID2	IoT Middleware: A survey on issues and enabling technologies	(Ngu et al., 2017)
ID3	Middleware for Internet of Things: A Survey	(Razzaque et al., 2016)
ID4	A gap analysis of Internet-of-Things platforms	(Mineraud et al., 2016)
ID5	Challenges in middleware solutions for the internet of things	(Chaqfeh & Mohamed, 2012)
ID6	A survey of middleware for Internet of things	(Bandyopadhyay, Sengupta, Maiti, & Dutta, 2011)

Fuente: El autor

La Tabla 3 contiene 13 estudios primarios que describen el diseño de los *middleware* consultados.

Tabla 3. Estudios primarios seleccionados para el estado del arte

ID	Título	Middleware	Referencia
ID7	Supporting edge intelligence in service-oriented smart IoT applications	WuKong	(Huang, Lin, & Shih, 2016)
ID8	Context and User Behavior Aware Intelligent Home Control Using WuKong Middleware		(Huang et al., 2015)
ID9	Autonomous Service Management for Location and Context Aware Service		(Shih, Lin, Chou, & Chuang, 2014)
ID10	HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems	LinkSmart	(Eisenhauer, Rosengren, & Antolin, 2010)
ID11	The semantic middleware for networked embedded systems applied in the internet of things and services domain		(Kostelnik, Sarnovsk, & Furdik, 2011)
ID12	OpenIoT: Open source internet-of-things in the cloud	OpenIoT	(Soldatos et al., 2015)
ID13	Integrating Wireless Sensor Networks within a City Cloud		(Petrolo, Mitton, Soldatos, Hauswirth, & Schiele, 2014)
ID14	Integrating Wireless Sensor Networks within a City Cloud		(Petrolo et al., 2014)
ID15	Rapid Interweaving of Smart Things with the meSchup IoT Platform	MeSchup	(Kubitza & Schmidt, 2016)
ID16	meSchup: A Platform for Programming Interconnected Smart Things		(Kubitza & Schmidt, 2017)
ID17	Using Speech for End User Programming of Smart Environments in the Internet of Thing		(Kubitza, 2016)
ID18	Security analysis of a proposed internet of things middleware	UIoT	(H. G. Ferreira & Sousa Junior, 2017)
ID19	FRED: A Hosted Data Flow Platform for the IoT	Node-RED	(Blackstock & Lea, 2016)

Fuente: El autor

2.3.3 Reporte. ID1, ID2 ID3, ID4, ID5 e ID6 se enfocan en describir los requisitos funcionales y no funcionales que un *middleware* para IoT debería cumplir, teniendo como base que este tipo de plataforma puede ofrecer servicios comunes para facilitar el desarrollo de aplicaciones, integrando tecnologías heterogéneas y apoyando la interoperabilidad entre las aplicaciones y los dispositivos de un entorno IoT. Dada la aplicabilidad que tiene IoT en diversos dominios (ver Figura 1), estos autores han analizado un listado de requisitos funcionales y no funcionales que una arquitectura *middleware* para IoT debería cumplir. Como base un *middleware* debería: *(i)* cumplir un estándar común entre todos los dispositivos que pertenecen a IoT; *(ii)* actuar como un enlace que une los componentes heterogéneos y; *(iii)* proporcionar una interfaz de programación de aplicaciones (API).

Algunos requisitos no funcionales que un *middleware* para IoT debería soportar son: *(i)* interoperabilidad de red, la cual define protocolos para intercambiar información

entre las cosas a través de diferentes redes de comunicación, sin considerar el contenido de la información; *(ii)* conciencia del contexto, responsable de caracterizar la situación de una entidad (persona, lugar, objeto) para trabajar en entornos inteligentes; *(iii)* descubrimiento de dispositivos, permite a cualquier dispositivo de la red IoT detectar todos sus dispositivos vecinos y hacer que su presencia sea conocida por cada uno de ellos. Los módulos dedicados al descubrimiento de dispositivos, deben garantizar confianza, adaptabilidad, tolerancia a fallos y optimización en el consumo de sus recursos; *(iv)* seguridad y privacidad, para proteger especialmente los mecanismos de almacenamiento y acceso a los datos. Adicional a estos requisitos, la revisión publicada en el 2017 por ID1, ID2 e ID3 identifican otros requisitos específicos que debe abordar un *middleware* para IoT, como se muestra en la Tabla 4.

Tabla 4. Requisitos genéricos de un *middleware* IoT

Requisitos funcionales	**Requisitos no funcionales**
Descubrimiento de recursos *(en inglés, Resource discovery)*	Comportamiento del tiempo *(en inglés, Time behaviour)*
Gestión de recursos *(en inglés, Resource management)*	Interoperabilidad *(en inglés, Interoperability)*
Gestión de datos *(en inglés, Data management)*	Operabilidad/Fácil desarrollo *(en inglés, Operability)*
Gestión de eventos *(en inglés, Event management)*	Procesamiento del flujo *(en inglés, Stream Processing)*
Gestión de código *(en inglés, Code management)*	Consciencia del contexto *(en inglés, Context-aware)*
	Seguridad *(en inglés, Security)*
	Escalabilidad *(en inglés, Scalability)*
	Persistencia *(en inglés, Persistency)*
	Visualización *(en inglés, Visualization)*
	Privacidad *(en inglés, Privacy)*
	Adaptabilidad *(en inglés, Adaptability)*
	Tiempo real *(en inglés, Real time)*

Fuente: Adaptado de ID1, ID2, ID3

De acuerdo con ID3, un *middleware* para IoT necesita ser escalable para garantizar el crecimiento de la red y los servicios que ofrece IoT. En este caso, el protocolo de Internet IPv6 es una solución muy escalable para ofrecer direccionamiento, ya que puede hacer frente a un gran número de cosas que deben incluirse en IoT (Khurana, 2017). Por otra parte, la virtualización de los servicios ofrecidos por el *middleware*, es útil para mejorar la escalabilidad, ocultando la complejidad de la lógica (*software*), la implementación de dispositivos *(hardware)* y redes subyacentes (Nitti et al., 2016). Cabe resaltar que en la literatura se describen *middleware* para IoT con diferentes enfoques de diseño, los cuales se describen a continuación.

2.3.3.1 Enfoque de diseño basado en servicios. Proporciona abstracciones para el *hardware* subyacente complejo a través de un conjunto de servicios que necesitan las aplicaciones. Este enfoque permite a los desarrolladores o usuarios, añadir y desplegar un diverso rango de dispositivos IoT como servicios.

En ID7 presentan a WuKong[10], un *middleware OpenSource* creado en primera instancia para facilitar el desarrollo de aplicaciones basadas en sensores y sistemas M2M (Lin, Reijers, Wang, Shih, & Hsu, 2013). La arquitectura de WuKong ha evolucionado hasta convertirse en un *middleware* inteligente para IoT que facilita el desarrollo de aplicaciones, independientemente del *hardware* que se use, las cuales pueden ser configuradas y desplegadas de forma dinámica en distintas plataformas IoT. La arquitectura del *middleware* está orientada a servicios e incluye tres capas: *(i)* aplicación, específica la clase de servicio de cada componente, su estructura de flujo y políticas de asignación; *(ii)* servicios, descubre automáticamente los recursos disponibles y los ubica en un directorio propio de servicios, y; *(iii)* dispositivos, identifica sensores, actuadores o dispositivos informáticos instalados en un entorno de usuario. WuKong soporta el descubrimiento de servicios, migración de código, virtualización y facilita el desarrollo de aplicaciones a través de una interfaz gráfica de usuario basada en Java. No brinda soporte en seguridad y no integra módulos para el reconocimiento del habla.

Tomando como base a WuKong, se han desarrollado aplicaciones para contextos específicos. En ID8 construyen a partir de WuKong, una aplicación de seguridad para una cocina como monitorización del contexto *(context-aware)* y un aprendizaje basado en el historial de comportamiento de los usuarios *(user-behaviour aware)*. En ID9 presentan un mecanismo para la identificación, monitorización, reconfiguración y gestión de interfaces de los dispositivos de forma remota y dinámica, basándose en la localización y el contexto.

ID10 describe a LinkSmart[11] inicialmente llamado Hydra como un *middleware OpenSource* diseñado en primera instancia para crear redes de sistemas embebidos. Nombrado como LinkSmart desde el 2014, este *middleware* ha evolucionado a una filosofía similar al paradigma IoT proporcionando una arquitectura orientada a servicios que utiliza modelo de seguridad XML y servicios Web. Como se describe en ID11, el diseño arquitectural de LinkSmart está orientado por un modelo semántico (en inglés, *Semantic Model Driven Architecture – SeMDA*) el cual incluye un conjunto de ontologías para facilitar el diseño de aplicaciones y promover interoperabilidad semántica de los servicios en línea y dispositivos inalámbricos/cableados. De esta forma, se logra que todos los dispositivos de una aplicación IoT basada en LinkSmart sean accesibles de manera uniforme, como sucede con los servicios Web semánticos. La arquitectura de LinkSmart se ubica entre la capa física y la capa de aplicación. Entre estos elementos se ejecutan

[10] Más información en la página Web https://newslabntu.github.io/wukong4iox/
[11] Más información en la página Web https://linksmart.eu/redmine

eventos ontológicos de forma local y remota, dada su arquitectura descentralizada. Cada uno de estos elementos está compuesto por capa de red, servicio, semántica y seguridad. LinkSmart soporta descubrimiento de servicios y auto-configuración de servicios/dispositivos en tiempo real. El componente de composición de servicios, contempla acciones para audio, el cual no es ampliamente explicado. No soporta la escalabilidad ni contiene módulos para el reconocimiento del habla.

2.3.3.2 Enfoque de diseño basado en la nube. Se enfoca en la provisión de servicios de *hosting* a través de Internet. Limita a los usuarios en el tipo y la cantidad de dispositivos IoT que pueden implementar, pero les permite conectarse, recopilar e interpretar datos con facilidad ya que los posibles casos de uso pueden ser determinados y programados a priori.

En ID12, se describe OpenIoT[12], un *middleware OpenSource* que permite la unificación semántica de diversas aplicaciones IoT con un enfoque basado en la nube. La arquitectura de OpenIoT descrita en ID13 consta de tres planos: *(i*) plano físico compuesto por un *middleware* de sensores que hace uso de la librería *Extended Global Sensor Networks X-GSN,* la cual recopila, filtra y combina flujos de datos de sensores virtuales o dispositivos físicos; *(ii)* plano virtualizado, compuesto por el almacenamiento de datos en nube que hace uso de la librería *Linked Stream Middleware Light LSM-Light*, la cual actúa como una base de datos en la nube que permite el almacenamiento de flujos de datos derivados del middleware del sensor. La infraestructura en la nube almacena también los metadatos necesarios para el funcionamiento de OpenIoT; el planificador, procesa las solicitudes de despliegue de servicios garantizando su acceso a los recursos que requieren; el administrador de utilidades y servicios, combina los flujos de datos indicados por los flujos de trabajo de servicio dentro del sistema para entregar el servicio solicitado a través de una consulta SPARQL; *(iii)* plano de aplicación, está compuesto por: el componente de definición de solicitud que comprende un conjunto de servicios para especificar y formular solicitudes de servicio. Este componente está soportado por una interfaz gráfica de usuario; el componente de solicitud de presentación, se encarga de la visualización de las salidas de un servicio. Este último, selecciona una aplicación Web híbrida *(mashups)* para facilitar la presentación del servicio; el componente de configuración y supervisión, permite la gestión visual y la configuración de funcionalidades sobre sensores y servicios que se implementan dentro de la plataforma. OpenIoT no soporta REST API para facilitar el desarrollo de servicios Web y no integra módulos para el reconocimiento del habla.

En ID14 tomando como base el *middleware* de OpenIoT desarrollaron una infraestructura de gran escala adecuada para probar pequeños dispositivos de sensores inalámbricos y objetos heterogéneos de comunicación llamada FIT IoT-LAB. La integración propuesta representa una forma de reducir la brecha existente

[12] Más información en la página Web http://www.openiot.eu/

en la fragmentación de IoT y, además, permite a los usuarios desarrollar aplicaciones de ciudad inteligente al interactuar directamente con sensores en diferentes niveles. Con el fin de demostrar su eficacia diseñaron una prueba piloto para el monitoreo de temperatura.

En ID15, ID16 describen a MeSchup[13] como una plataforma IoT cuya principal función es facilitar el desarrollo de aplicaciones IoT. La arquitectura de MeSchup consta de cuatro capas: *(i)* dispositivos, donde cualquier dispositivo se ve como una unidad de cómputo con un número de módulos soportados por sensores que producen datos o actuadores que pueden ser activados de forma asíncrona; *(ii)* abstracción, donde la heterogeneidad de las tecnologías de comunicación y el descubrimiento de dispositivos están totalmente abstraídos. Independientemente del tipo de dispositivo, todos se descubren automáticamente y se exponen como una cosa inteligente y virtual que se puede configurar y controlar de forma remota; *(iii) runtime,* implementa de forma instantánea el código de comandos que fue guardado previamente, de tal forma que se pueda probar en el mundo real, y; *(iv)* interfaz gráfica, la cual permite combinar cualquier cosa inteligente con otras, usando una interfaz gráfica del lenguaje de programación JavaScript basada en Web, lo que facilita la administración de cosas inteligentes. Aunque la plataforma meSchup tiene un enfoque basado en la nube, ofrece privacidad de datos por diseño porque puede ejecutarse como una solución puramente local, y no es necesario el acceso a la nube.

Tomando como base MeSchup, en ID17 los autores desarrollan una interfaz de usuario multimodal para dispositivos móviles denominada *Speechweaver*. Esta aplicación se ejecuta de la siguiente manera: toma la orden dada a través de la voz, lo interpreta, lo convierte en un script ejecutable y muestra en la pantalla de dispositivos móviles el texto hablado. La interfaz gráfica búsqueda dentro de una lista de opciones relacionadas con lo hablado por el usuario y las funciones que la aplicación tiene configurada de forma previa (por ejemplo, encender/apagar algo específico) Según los autores, la característica más innovadora de *Speechweaver* es la combinación de la entrada de voz como lenguaje natural con el concepto de Programación por Demostración *(en inglés, Programming by Demonstration – PbD)*, para enseñar a un dispositivo nuevos comportamientos sin necesidad de programarlos a través de comandos de máquina.

2.3.3.3 Enfoque de diseño basado en actores. Se enfoca en exponer una variedad de dispositivos IoT como actores reutilizables para distribuirlos en la red de tal forma que puedan realizar variedad de cálculos computacionales. No tienen un estándar particular para la comunicación entre dispositivos IoT.

[13] Más información en la página Web http://www.meschup.com/

Un ejemplo de este tipo de enfoque es el presentado en ID18, un *middleware OpenSource* para IoT llamado UIoT[14], propuesto con un modelo arquitectural que proporciona interfaces digitales y analógicas en serie móviles para la interconexión de dispositivos abstrayendo su complejidad mediante la tecnología *Universal Plug and Play – UPnP,* la cual tiene características pervasivas y conectividad *peer-to-peer – P2P* (H. G. C. Ferreira, Canedo, & de Sousa, 2013). La arquitectura de UIoT cuenta con medidas de protección basadas en las tecnologías existentes para las necesidades particulares de IoT. Esta arquitectura está desarrollada independientemente de otras arquitecturas propuestas, centrándose en la uniformidad de servicios y la reducción de cargas para dispositivos finales. La arquitectura de UIoT se ubica entre los dispositivos y aplicaciones e incluye cuatro capas: *(i)* entidades físicas constan de dos controladores: el maestro responsable de mantener el estado, las abstracciones y comunicarse con las aplicaciones, y los esclavos responsables de comunicarse con los dispositivos a través de sus interfaces digitales y analógicas; *(ii)* control, supervisa la abstracción y monitoriza los estados de los dispositivos; *(iii)* APIs, son uniformes y transparentes a las aplicaciones y a los servicios de suscripción y; *(iv)* seguridad, donde el *middleware* toma medidas de seguridad de forma distribuida protegiendo el intercambio de información, la identificación de entidades y autorización para interoperar recursos internos y externos. UIoT facilita la escalabilidad de las aplicaciones a través de computación distribuida basada en entidades. No soporta la conciencia del contexto, administración de eventos y no integra módulo de reconocimiento del habla.

ID2 describe a Node-RED[15] como una plataforma *middleware OpenSource* desarrollado por IBM. Su arquitectura se basa en node.js, una plataforma JavaScript que utiliza un módulo de entrada y salida controlado por eventos en un entorno informático distribuido. Su ejecución se puede realizar en el borde de la red. La abstracción de dispositivos se realiza a través de un Nodo, considerado como una representación visual diseñado para realizar una función específica en un dispositivo IoT. En otras palabras, cada nodo se puede ver como un actor. Node-RED permite a los usuarios arrastrar y soltar bloques que representan componentes de un sistema más grande para formar una aplicación de IoT.

En ID19 diseñan un sistema denominado Front-End para Node-RED (FRED) que gestiona múltiples instancias de Node-RED funcionando como un proxy "inteligente" que crea y administra los procesos del *middleware*, y las comunicaciones "proxy" entre clientes, dispositivos y servicios conectados. El servicio FRED se puede usar para conectar dispositivos a servicios en la nube, coordinar la comunicación entre dispositivos, integrar servicios entre sí o para crear nuevas API y aplicaciones Web.

[14] Más información en la página Web https://uiot.org/
[15] Más información en la página Web https://nodered.org/

Node-RED no contempla requisitos de seguridad. Sin embargo, puede ser conectado con un módulo para la decodificación de voz a texto y viceversa, siempre y cuando el desarrollador cuente con las credenciales del servicio IBM Cloud. La versión gratuita es limitada a 256 MB de capacidad.

2.3.3.4 Conclusiones del estado del arte. Cabe resaltar que no todas las plataformas *middleware* presentadas en la literatura atienden los requisitos funcionales y no funcionales que demanda IoT (Manrique et al., 2016). La mayoría de ellas como las expuestas en ID3 son *middleware* que han sido implementados para atender los requisitos que demanda otro tipo de paradigma como las redes de sensores, y que con la inclusión del paradigma IoT, están siendo usados para abordar las necesidades de conexión entre dispositivos y la red. Algunos por su parte, han evolucionado con el fin de atender requisitos específicos que demanda IoT. En la Tabla 5 se realiza una comparación a nivel de requisitos funcionales de los *middleware* consultados. Estos requisitos especifican una función que el *middleware* o un componente del mismo, debe ser capaz de realizar.

Tabla 5. Comparación de requisitos funcionales de los *middleware* IoT

	Functional requirements						
Middleware	**Resource Discovery**	**Resource Management**	**Data Management**	**Event Management**	**Code Management**	**Device Abstraction**	**Network Connectivity**
WuKong	Yes	Yes	Local stored	Yes	--	Yes	XMPP, UDP
LinkSmart	Yes	Yes	Local stored, Data fusion	Yes	--	Yes	Zigbee, Bluetooth, WLAN
OpenIoT	Yes	Yes	Cloud Storage	Yes	--	Yes	IEEE802.15.4, IPSec, TSL, HTTPS
UIoT	Yes	Yes	Data integrity	--	Yes	Yes	UPnP, REST APIs
Node-RED	--	--	Connecting with the IBM Cloud	Yes	Yes	Yes	Yes
MeSchup	Yes	--	Local and Cloud stored	Yes	Yes	Yes	Wi-Fi, Bluetooth, ZigBee, Z-Wave

Fuente: El autor

En la Tabla 6 se realiza una comparación a nivel de requisitos no funcionales de los *middleware* consultados. Estos requisitos especifican como va a funcionar una función o un componente del *middleware*, es decir, se enfoca en la calidad del servicio.

Tabla 6. Comparación de requisitos no funcionales de los *middleware* IoT

Non-functional requirements											
Middleware	**Scalability**	**Security**	**Interoperability**	**Availability**	**Portability**	**Real-time**	**Visualization**	**Heterogeneity**	**Context-aware**	**Stream Processing**	**Operability**
WuKong	Virtualization	--	--	--	--	Yes	Java-based GUI application	Yes	Yes	Flow-based processing	Drag and drop services based on flow-based program
LinkSmart	-	Yes	Yes	--	--	Yes	Web	Yes	Yes	Event manager	Semantic Model Driven Architecture
OpenIoT	Descentralized	Yes	Yes	--	x	Yes	JavaMelody library	Yes	--	Flow-based processing	--
UIoT	Distributed computing	Yes	Yes	--	--	--	HTTP	Yes	--	Entity manager	--
Node-RED	Centralized	--	Yes	--	x	Yes	Java-based GUI application	Yes	Yes	Flow-based processing	Drag and drop services based on flow-based program
MeSchup	Distributed	Yes	Yes	--	--	Yes	Java-based GUI application	Yes	Yes	Flow-based	Drag and drop services based on flow-based

Fuente: El autor

IoT es un paradigma que consta de requisitos específicos para darle conexión a Internet a las cosas cotidianas y proveerles una interfaz digital que facilite la

interacción entre hombre-máquina y máquina-máquina. Tomando como base estos requisitos se han diseño diversos *middleware* para IoT.

Tabla 7. Comparación de características generales de los *middleware* IoT

Generic features							
Middleware	**Approach**	**Speech recognition**	**Operating System**	**Programming language**	**Open Source**	**Layers**	**Country**
WuKong	Service-oriented	--	Linux	Python, Java	Yes	Application Services Devices	China
LinkSmart	Service-oriented	--	TinyOS, Windows, Linux	Java, OWL	Yes	Application Semantic Service Network Security (cross) Device	European Union
OpenIoT	Cloud-based	--	Linux	SPARQL, XML, Java,	Yes	Application Plane Virtualized Plane Physical Plane	European Union
UIoT	Actor-based	--	Windows	Ruby	Yes	Application APIs Control Physical Entities Devices Security (cross)	Brazil
Node-RED	Actor-based	Text-to-Speech, Speech-	Windows	JavaScrip, JSON	Yes	Not specific	United States
MeSchup	Cloud-based	Speech-to-Text	Android, Windows, iOS	JavaScript, Android	Yes	Application GUI Runtime Abstraction Device	Germany

Fuente: El autor

Un *middleware* facilita el desarrollo de aplicaciones para los diversos dominios de IoT. Los documentos analizados se centran en esta problemática. Los *middleware* propuestos son diversos e implican varios enfoques de diseño, atienden varias capas de desarrollo y soportan diferentes requisitos arquitecturales, funcionales, no funcionales y características generales (Ver Tabla 7).

Aún existen brechas de investigación que retan a los investigadores para continuar aportando a esta área del conocimiento. Por otra parte, se observa que solo algunos de los *middleware* IoT estudiados incluyen en su diseño el requisito conciencia del contexto *(en inglés, context-aware)* y no se evidencia otras características de UbiComp. Para lograr aplicaciones IoT como *Speechweaver*, es importante incluir desde su diseño. Algunos requisitos que se enfoquen en interfaces digitales para la interacción natural entre el usuario y los dispositivos. Este tipo de *middleware* para IoT aún sigue siendo una oportunidad de investigación y desarrollo latente, puesto que no han sido implementados de forma masiva.

2.4 MARCO CONTEXTUAL

En esta sección se describe el contexto organizacional en el que se realiza el presente proyecto. Además, se identificaron los beneficiarios receptivos al trabajo de investigación planteado. Estos interesados pueden ser beneficiarios directos e indirectos, excluidos, como se muestra en la Tabla 8.

Tabla 8. Interesados en el proyecto

Beneficiarios directos
Desarrolladores de aplicaciones IoT
Investigadores de sistemas para el reconocimiento del habla
Arquitectos de software
Comunidad *OpenSource*
Beneficiarios indirectos
Investigadores del Centro de Excelencia y Apropiación en Internet de las Cosas (CEA IoT)
Organizaciones que desarrollan middleware o sistemas para el reconocimiento del habla
Usuarios de aplicaciones IoT con interfaces basadas en voz
Excluidos
Personas con discapacidad del habla

Fuente: El autor

Los desarrolladores de aplicaciones IoT, arquitectos de *software* y la comunidad *Open source* serán los principales beneficiarios directos del diseño del *middleware*, ya que este *software* ayuda a ocultar la complejidad en el desarrollo de aplicaciones IoT. Además, estas aplicaciones incluirán interfaces basadas en voz que facilitan la interacción natural entre humano-máquina. Es por eso que los investigadores de

sistemas para el reconocimiento del habla también podrían hacer uso de los resultados esperados a partir del desarrollo del proyecto.

Los interesados indirectos se definen a largo plazo como potenciales beneficiarios, ya que en una primera instancia no se verán afectados directamente. Particularmente, el Centro de Excelencia y Apropiación en Internet de las Cosas (CEA - IoT[16]) soporta el desarrollo de este proyecto. El CEA – IoT es una iniciativa impulsada por el Ministerio de las TIC[17], con el apoyo de Colciencias[18] que inicia como una alianza entre universidades, líderes tecnológicos mundiales y empresas ancla para potenciar el desarrollo económico de Colombia desde el desarrollo tecnológico y la innovación a través de las tecnologías de IoT, buscando resolver las necesidades de diferentes sectores productivos del país. Actualmente la Universidad Autónoma de Bucaramanga lidera el nodo oriente del CEA IoT.

Dentro de las organizaciones que desarrollan modelos de referencia arquitecturales para IoT se encuentra *The Lighthouse Project IoT-A*[19]*.* Este proyecto establece un modelo que sirve como referencia para la interoperabilidad de los sistemas de IoT, delineando principios y directrices para el diseño técnico de sus protocolos, interfaces y algoritmos. Además, establece un mecanismo para la integración de las cosas a Internet de tal forma que se pueda comunicar el mundo real con el digital. Este modelo sirve como referencia para el desarrollo del presente proyecto.

Por su parte, las personas con discapacidad del habla serán en principio excluidas porque el diseño considera el reconocimiento del habla. Ellas se convierten en potenciales beneficiarias para un trabajo futuro.

2.5 MARCO LEGAL Y POLÍTICO

En esta sección se presenta una revisión de las normas, políticas o estándares que influencian o aportan en este proyecto de investigación. Para esto, se tuvo en cuenta normas que fueran ampliamente conocidas y aceptadas a nivel internacional para proveer a los productos derivados de esta investigación una compaginación con buenas prácticas.

2.5.1 ISO/IEC/IEEE 24765:2010(E). Provee un vocabulario común aplicado a todos los sistemas y el trabajo relacionado con ingeniería del *software.* Los términos usados para el diseño del *software* son tomados de *Systems and software engineering - Vocabulary* desarrollado por la (ISO & IEEE, 2010), los cuales se describen en la Tabla 9.

[16] Más información en la página Web http://www.cea-iot.org/
[17] Más información en la página Web http://www.mintic.gov.co
[18] Más información en la página Web http://www.colciencias.gov.co
[19] Más información en la página Web https://cordis.europa.eu/project/rcn/95713_en.html

Tabla 9. Conceptos para el diseño de *software*

Terminología	Definición
Requisito funcional	Requisito que especifica una función que un sistema o un componente del sistema debe ser capaz de realizar.
Requisito no funcional	Requisito que específica cómo va a desarrollar una función un sistema o componente del sistema. Captura la calidad del servicio.
Diseño arquitectónico	Proceso de definición de una colección de componentes de hardware/software y sus interfaces para establecer el marco de trabajo de un sistema.
Modelado estructural	Disposición física o lógica de los componentes de un diseño de sistema y sus conexiones internas y externas.
Modelado comportamental	Proceso que describe la potencial variedad de estados posibles para un sistema o componente del sistema.
Interfaz de usuario	Interfaz que permite que la información sea enviada entre un usuario y los componentes de hardware/software de un sistema.
Análisis	Proceso de estudiar un sistema dividiéndolo en partes (funciones, componentes u objetos) y determinando cómo esas partes se relacionan entre sí.
Modelado	Es una representación de un proceso, dispositivo o concepto del mundo real; también puede ser definido como una abstracción semánticamente cerrada de un sistema o una descripción completa de un sistema desde una perspectiva particular.
Evaluación	Determinación sistemática de la medida en que una entidad cumple los criterios especificados.
Dominio	Ámbito distinto, dentro del cual se exhiben características comunes, se observan reglas comunes y sobre las cuales se mantiene una transparencia de distribución

Fuente: Tomados de ISO/IEC/IEEE 24765:2010(E)

2.5.2 ISO/IEC 25010:2011 La Organización Internacional de Normalización *(en inglés, International Organization for Standardization)* (ISO & IEEE, 2010), establece un modelo que determina las características de calidad (requisitos no funcionales) que se deben tener en cuenta para evaluar un producto software. La calidad de un sistema es el grado en que el sistema satisface las necesidades explicitas e implícitas de sus *stakeholders* y, por lo tanto, proporciona valor. Este modelo consta de ocho características: Idoneidad funcional, eficiencia de rendimiento, compatibilidad, usabilidad, confiabilidad. Seguridad, mantenibilidad y portabilidad, las cuales se subdividen en sub-características que se pueden medir internamente y externamente. Al ser un *middleware* para IoT un *software,* este modelo será usado como parte de los requisitos no funcionales que éste debe tener.

2.6 CONSIDERACIONES FINALES DEL CAPÍTULO

IoT es un paradigma emergente que tiene varios dominios de aplicación e integra una gran variedad de tecnologías heterogenias, lo cual hace que el diseño e

implementación de plataformas que faciliten el desarrollo de aplicaciones IoT sea un reto actual.

Particularmente, el diseño de una plataforma se puede realizar a través del modelado. Un modelo es una representación de un proceso o conceptual del mundo real al mundo digital, y se puede representar a través de un lenguaje de modelamiento, de tal forma que se identifiquen los elementos que la conforman y sus relaciones entre ellos. Este tipo de modelo puede llevarse a una arquitectura de aplicación IoT, ser implementada, probada y cuyo resultado sea una aplicación funcional.

Este tipo de desarrollo, en áreas como las telecomunicaciones, la telemática y recientemente, IoT, se puede garantizar mediante el uso de la ingeniería del *software* que no solo se aplica para la producción del *software*, sino que puede llevarse a otras áreas del conocimiento, como las mencionadas anteriormente, para el desarrollo de sistemas más complejos. La ingeniería del *software* ha definido un ciclo para el desarrollo de *software* o sistemas. Este ciclo comienza con la gestión de los requisitos funcionales y de calidad del sistema (no funcionales). Una buena gestión de los requisitos hace que se comprendan de mejor forma las necesidades reales, que estas se traduzcan en diseños que luego serán implementados y probados, teniendo como resultado un desarrollo funcional que cumpla con las necesidades de los usuarios, de forma eficiente y oportuna.

La ingeniería del *software* enfocada al diseño de *middleware* puede aportar significativamente al IoT. En el punto de intersección de estas áreas se desarrolla este proyecto de investigación, y se espera contribuir significativamente a la comunidad científica con los resultados obtenidos.

3 ASPECTOS METODOLÓGICOS

En este capítulo se describe el proceso de investigación propuesto en este proyecto para dar cumplimiento a los objetivos planteados en el capítulo I. En la sección 3.1 se describe el tipo y enfoque investigación abordado en el proyecto. En la sección 3.2 se describen los instrumentos que se usaran para la recolección de información. Esta información, será la base para el diseño del middleware planteado. En la sección 3.3 se detallan las fases y actividades propuestas para dar cumplimiento a cada uno de los objetivos específicos planteados. En la sección 3.4 se detalla el presupuesto del proyecto.

3.1 TIPO Y ENFOQUE DE INVESTIGACIÓN

El tipo de investigación que aborda este proyecto es una investigación aplicada (Colciencias, 2016) que se caracteriza por generar nuevo conocimento o usar el ya existente para dar respuesta a un problema o necesidad identificada. Esta investigacion aplicada tiene un enfoque cuantitativo, el cual parte de una idea, que va acotandose y, una vez delimitada, se derivan objetivos y la pregunta de investigacion, se revisa la literatura y se construye un marco referencial. A partir de las preguntas se establecen las hipotesis y se determinan variables. Con el diseño se pueden probar en un determinado contexto para establecer una serie de conclusiones respecto a la hipótesis planteada (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2010, p. 4).

La necesidad identificada se basa en que requieren plataformas que faciliten el desarrollo de aplicaciones IoT que contengan interfaces basadas en voz para que la interacción entre el humano y la máquina se realice de forma no intrusiva. Por ende, se propuso el diseño de un *middleware* que atiende los requisitos mínimos que exige IoT.

3.2 TÉCNICAS E INSTRUMENTOS DE RECOLECCIÓN DE INFORMACIÓN

Para establecer los requisitos funcionales y no funcionales de SWITCH se revisaron estudios en la literatura relacionados con arquitecturas de referencia IoT, arquitecturas *middleware* para IoT y sistemas de reconocimiento del habla. Además se usó la norma de estandarización ISO/IEC 25010 (2011) para abstraer los requisitos no funcionales que todo *software* debe tener. Se hizo un análisis comparativo de los requisitos encontrados para obtener como resultado un listado de los requisitos funcionales y no funcionales genéricos para un *middleware* IoT.

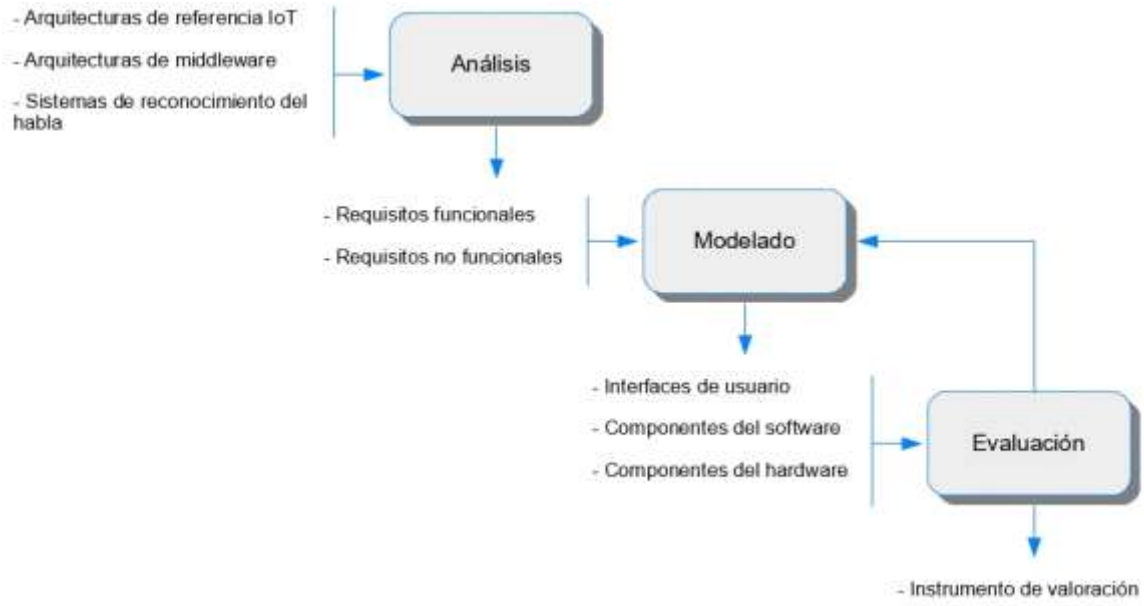
Con base en este análisis, se abstrajeron los requisitos del *middleware* propuesto denominado como *Smart middleWare for Iot with speeCH recognition – SWITCH.*

- El diseño de SWITCH se realizó a través de un lenguaje de descripción arquitectural UML.
- Se realizó una prueba de concepto de SWITCH usando como base wit.ai, una plataforma para el reconocimiento del habla disponible en la Web.
- Se adaptó un instrumento de evaluación para el diseño de middleware para IoT, usando como base un framework de evaluación encontrado en la literatura.

3.3 FASES Y ACTIVIDADES

En esta sección se describe el proceso de investigación que se siguió a lo largo de este proyecto para dar cumplimiento a los objetivos planteados. La metodología de investigación se compone de tres fases relacionadas con los objetivos específicos, como se muestra en la figura 6. Cada fase está compuesta por un conjunto de subfases (SF) y actividades (AC), las cuales se detallan a continuación.

Figura 5. Fases de la metodología de investigación



Fuente: El autor

3.3.1 Fase 1: Análisis. En esta fase se analizaron los requisitos funcionales y no funcionales *middleware* para IoT, arquitecturas de referencia para IoT y sistemas

para el reconocimiento del habla. Como parámetros de salida se obtuvo los requisitos de SWITCH.

(SF1) Estudio de arquitecturas de referencia IoT

- (AC1) Búsqueda y revisión en la literatura sobre arquitecturas de referencia IoT.
- (AC2) Comparación de las arquitecturas de referencia IoT consultadas.
- (AC3) Análisis de las arquitecturas de referencia IoT consultadas.

(SF2) Estudio de arquitecturas *middleware*

- (AC1) Búsqueda y revisión en la literatura sobre arquitecturas middleware.
- (AC2) Comparación de las arquitecturas middleware consultadas.
- (AC3) Análisis de las arquitecturas middleware consultadas.

(SF3) Estudio de arquitecturas de sistemas de reconocimiento del habla

- (AC1) Búsqueda y revisión en la literatura sobre arquitecturas de los sistemas de reconocimiento del habla.
- (AC2) Comparación de las arquitecturas de sistemas de reconocimiento del habla consultadas.
- (AC3) Análisis de las arquitecturas de sistemas de reconocimiento del habla consultadas.

(SF4) Establecimiento de los requisitos funcionales y no funcionales del *middleware* tomando como base el análisis realizado en las subfases anteriores.

- (AC1) Descripción de los requisitos funcionales.
- (AC2) Descripción de los requisitos no funcionales.

3.3.2 Fase 2: Modelado. En esta fase se modeló el dominio, la arquitectura y los componentes de *software* y *hardware* de SWITCH. Los parámetros de entrada para realizar el modelado fueron los requisitos funcionales y no funcionales obtenidos en la fase anterior. Como parámetros de salida se obtuvo el diseño de SWITCH.

(SF5) Modelado de los componentes del *software*

- (AC1) Definición de las interfaces de usuario.
- (AC2) Selección de un lenguaje de descripción arquitectural.
- (AC3) Modelado estructural de los componentes de los módulos relacionados con el reconocimiento del habla de la arquitectura del middleware. (componentes)
- (AC4) Modelado comportamental de los componentes de los módulos relacionados con el reconocimiento del habla de la arquitectura del middleware. (actividades).

(SF6) Modelado de los componentes del *hardware*

- (AC1) Identificación de los componentes del hardware.
- (AC2) Modelado de los componentes del hardware.
- (AC3) Modelado de la integración de los componentes de hardware y software.

3.3.3 Fase 3: Evaluación. En esta fase evaluó el diseño de SWITCH mediante una prueba de concepto y la comparación con los *middleware* consultados en la literatura. Se obtuvo como parámetro de salida un instrumento para la evaluación de *middleware* para IoT.

(SF7) Evaluación del *middleware* diseñado

- (AC1) Identificación de los escenarios.
- (AC2) Búsqueda y revisión de instrumentos de valoración de la arquitectura de middleware para IoT.
- (AC3) Adaptación del instrumento de valoración tomando como base los requisitos del middleware.
- (AC4) Ejecución de la evaluación de los módulos del middleware diseñado.
- (AC5) Desarrollo de una prueba de concepto.
- (AC6) Comparación de SWITCH con los middleware consultados en la literatura.
- (AC7) Análisis de los resultados

4 ANÁLISIS DE REQUISITOS

En este capítulo se describe el proceso de revisión sistemática de la literatura para obtener los requisitos funcionales y funcionales de SWITCH. En la sección 4.1 se describen los resultados obtenidos de la revisión de la literatura enfocada al estudio de arquitecturas de referencia IoT. En la sección 4.2 se describen los resultados obtenidos de la revisión de la literatura enfocada al estudio de arquitecturas *middleware* para IoT. En la sección 4.3 se describen los resultados obtenidos de la revisión de la literatura enfocada al estudio de sistemas de reconocimiento del habla. En la sección 4.4 se presenta la síntesis de los requisitos funcionales y no funcionales que un *middleware* IoT debe tener. Finalmente, en la sección 4.5 se listan los requisitos funcionales y no funcionales de SWITCH.

4.1 ARQUITECTURAS DE REFERENCIA PARA IOT

En esta sección se presenta el procedimiento para la revisión sistemática de la literatura en relación con las arquitecturas de referencia para IoT. Esta revisión se presenta en tres fases: *(i)* planificación, *(ii)* conducción y *(iii)* reporte de los resultados obtenidos.

4.1.1 Planeación. Esta revisión tiene como objetivo identificar y analizar arquitecturas de referencia (AR) para IoT, sus componentes y características principales. Para obtener estos estudios, se propuso la siguiente pregunta de búsqueda:

- ¿Cuáles son y cómo están diseñadas las AR para IoT?
- ¿Cuáles son los requisitos funcionales y no funcionales que una AR para IoT atiende?

Para dar respuesta a estas preguntas se definió la cadena de búsqueda, tomando como base términos clave para formar grupos con características semejantes (sinónimos).

TITLE-ABS-KEY ((“Internet of Things” OR IoT) AND (“Reference Architectures” OR “Reference Architecture” OR “Reference Models” OR “Reference Model”))

La cadena de búsqueda fue consultada en bibliotecas digitales en línea con el objetivo de recuperar estudios primarios concernientes a esta temática. Para determinar estos recursos de búsqueda, se examinaron y seleccionaron bibliotecas

digitales con artículos relevantes en Ciencias de la Computación, particularmente en IoT.

Para reducir progresivamente la cantidad de estudios encontrados en la búsqueda a un número con alta calidad relevantes para atender las preguntas planteadas, se establecieron unos criterios de selección: inclusión (CI) y exclusión (CE). Estos criterios se describen a continuación.

CI1: El estudio está escrito en inglés.
CI2: El estudio se publicó desde el año 2005 al 2018.
CI3: El estudio es de tipo artículo, libro, capítulo de libro o artículo de conferencia.
CI4: El título, palabras clave y resumen del estudio coinciden con los criterios de búsqueda.
CI5: El estudio describe el diseño de una arquitectura de referencia para IoT.

CE1: No se tiene acceso a todo el documento *(full text)*
CE2: El estudio está relacionado con IoT, pero no propone o discute una AR.
CE3: La AR genérica no menciona sus componentes.
CE4: La AR es de otros tipos de sistemas que no contienen funciones IoT.

4.1.2 Conducción. Los estudios primarios fueron buscados, seleccionados y evaluados de acuerdo al protocolo establecido previamente, lo que resultó en un conjunto de artículos relevantes para el objetivo establecido. La consulta de búsqueda realizada en las bibliotecas digitales online fue limitada a los campos de título, palabras clave y resumen. Se recuperaron 806 artículos de las 5 bases de datos consultadas, como se observa en la
4.1.3
4.1.4
4.1.5 Tabla 10.

Tabla 10. Recursos digitales: AR para IoT

Biblioteca Digital	Artículos obtenidos	Artículos repetidos	Artículos evaluados
Springer Link	606	19	587
Scopus	125	5	120
IEEE Xplore	59	50	9
ScienceDirect	14	9	5
ACM	2	1	1
Total	806	84	722

Fuente: El autor

En el primer filtro, se siguieron los criterios de inclusión resultando un conjunto de 84 artículos seleccionados. En el segundo filtro, se siguieron los criterios de exclusión resultando 40 estudios como se observa en la Tabla 11.

Tabla 11. Estudios primarios seleccionados de AR para IoT

ID	Título	Año	Referencia
ID1	Architecting the Internet of Things: State of the Art	2016	(Abdmeziem et al., 2016)
ID2	A resilient Internet of Things architecture for smart cities	2016	(Abreu, Velasquez, Curado, & Monteiro, 2017)
ID3	A reference architecture for improving security and privacy in internet of things applications	2014	(Addo, Ahamed, Yau, & Buduru, 2014)
ID4	A Proposed Security Layer for the Internet of Things Communication Reference Model	2015	(Aldosari, 2015)
ID5	Internet of things communication reference model	2014	(Alhamedi, Snasel, Aldosari, & Abraham, 2014)
ID6	An IoT Protocol and Framework for OEMs to Make IoT-Enabled Devices Forward Compatible	2015	(Banda, Chaitanya, & Mohan, 2015)
ID7	Enabling things to talk: Designing IoT solutions with the IoT architectural reference model	2013	(Bassi et al., 2013)
ID8	IoT reference model	2013	(Bauer et al., 2013)
ID9	Privacy-Preserving Security Framework for a Social-Aware Internet of Things	2014	(Bernabe, Hernández, Moreno, & Gomez, 2014)
ID10	A process for Generating Concrete Architectures	2013	(Boussard et al., 2013)
ID11	An analysis of Reference Architectures for the Internet of Things	2015	(Cavalcante, Alves, Batista, Delicato, & Pires, 2015)
ID12	Test-enabled architecture for IoT service creation and provisioning	2013	(De, Carrez, Reetz, Tönjes, & Wang, 2013)
ID13	IoT Lab: Towards co-design and IoT solution testing using the crowd	2015	(Fernandes et al., 2015)
ID14	Architecture Reference Model	2014	(Höller et al., 2014a)
ID15	The advantages of IoT and cloud applied to smart cities: ClouT user scenarios and reference architecture	2015	(Formisano et al., 2015)
ID16	IIoT Reference Architecture	2016	(Gilchrist, 2016)
ID17	An Architectural Blueprint for a Real-World Internet	2011	(Alexander Gluhak et al., 2011)
ID18	Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things	2013	(Guo, Zhang, Wang, Yu, & Zhou, 2013)
ID19	IoT Architecture – State of the Art	2014	(Höller et al., 2014b)
ID20	IoT Reference Architecture	2014	(Höller et al., 2014c)
ID21	Designing IoT architecture(s): A European perspective	2014	(Krco, Pokric, & Carrez, 2014)
ID22	Internet of Things	2013	(G. M. Lee, Crespi, Choi, & Boussard, 2013)
ID23	Data management for internet of things: Challenges, approaches and opportunities	2013	(Ma, Wang, & Chu, 2013)
ID24	From the Internet of Things to the Internet of People	2015	(Miranda et al., 2015)

ID	Título	Año	Referencia
ID25	Social Networks and Internet of Things, an Overview of the SITAC Project	2014	(Monteiro, Oliveira, Bastos, Ramrekha, & Rodriguez, 2014)
ID26	A Comprehensive Study of Security of Internet-of-Things	2016	(Nia & Jha, 2016)
ID27	Architecture for Internet of Things Analytical Ecosystem	2016	(Ratkowski, 2016)
ID28	An IoT based reference architecture for smart water management processes	2015	(Robles et al., 2015)
ID29	SmartSantander: The meeting point between Future Internet research and experimentation and the smart cities	2011	(L. Sanchez et al., 2014)
ID30	Internet of Things Service Systems Architecture	2015	(Schauer & Debita, 2015)
ID31	HePA: Hexagonal platform architecture for smart home things	2016	(Seo, Kim, Yun, Huh, & Maeng, 2015)
ID32	Standardizing the internet of things in an evolutionary way	2014	(Shen & Carugi, 2014)
ID33	Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm	2014	(Shrouf, Ordieres, & Miragliotta, 2014)
ID34	Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce	2015	(Singh & Singh, 2015)
ID35	A Resource-Oriented Architecture for the Internet of Things (IoT)	2016	(Souza & Cardozo, 2016)
ID36	IoT-A and FIWARE: Bridging the barriers between the cloud and IoT systems design and implementation	2016	(Stravoskoufos, Sotiriadis, & Petrakis, 2016)
ID37	Reference model of Industrie 4.0 service architectures: Basic concepts and approach	2015	(Usländer & Epple, 2015)
ID38	A reference architecture for IoT-based logistic information systems in agri-food supply chains	2015	(Verdouw, Robbemond, Verwaart, Wolfert, & Beulens, 2015)
ID39	Reference Architectures for the Internet of Things	2016	(Weyrich & Ebert, 2016)
ID40	Research on architecture of the internet of things for grain monitoring in storage	2012	(Xu, Zhang, & Yang, 2012)

Fuente: El autor

En la Tabla 12, se muestra las arquitecturas de referencias para IoT propuestas, mencionadas o en las que se basan algunos de los 40 estudios primarios estudiados.

Para dar respuesta a la pregunta relacionada con las AR para IoT más citadas, la cual fue establecida en el protocolo de búsqueda, se tomó como base el número de estudios que mencionan la arquitectura de referencia. En la Figura 6, se muestra la gráfica con la información tabulada sobre las arquitecturas de referencia para IoT encontradas en la literatura.

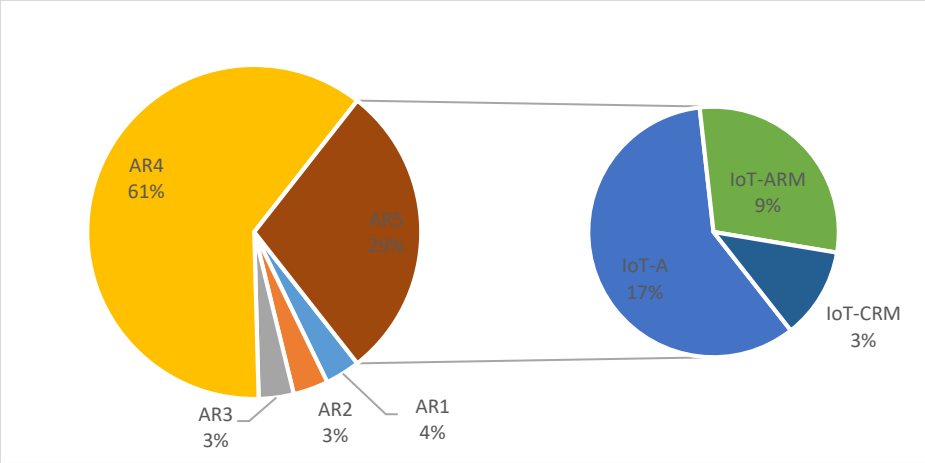
Tabla 12. AR para IoT mencionadas en la literatura

Arquitectura de referencia		Referencia
Arquitectura de IoT resiliente para ciudades inteligentes		ID37
Arquitectura de logística Agroalimentaria Inteligente		ID38
Arquitectura de referencia basada en IoT para procesos inteligentes de gestión del agua		ID28
Arquitectura de referencia ClouT		ID15
Arquitectura de referencia de WSO2		ID11
Arquitectura de referencia IoT para E-commerce		ID34
Arquitectura de Referencia para fábrica inteligente basada en IoT		ID33
Arquitectura de referencia para mejorar la seguridad y la privacidad		ID3
Arquitectura de referencia para Real-World Internet (RWI)		ID17
Arquitectura IoT para el monitoreo de grano en almacenamiento		ID40
Arquitectura orientada a recursos para IoT		ID35
Arquitectura para análisis de datos en IoT		ID27
Arrowhead Framework		ID39
BRIDGE		ID1
CASAGRAS Project		ID1
Cloud approach		ID1
EPC based approach		ID1
IDRA approach		ID1
IEFT Protocol Suite		ID1
SENSEI Project		ID1
Server based approach		ID1
Social Internet of Things (SIoT) architecture		ID1
Virtual network approach		ID1
FI-WARE		ID21
Framework conceptual para la IoT oportunista		ID18
Hexagonal Platform Architecture – HePA		ID31
IIoT Reference Architecture		ID16
Industrial Internet Reference Architecture (IIRA)		ID6, ID39
Internet of People Middleware		ID24
IoT-A Project	Internet of Things - Architecture (IoT-A)	ID7, ID10, ID12, ID14, S20, ID21, ID22, ID30, ID32, ID39
	Architecture Reference Model (IoT-ARM)	ID8, ID9, ID11, ID13, ID36
	IoT-A Communication Reference Model	ID4, ID5
IoT6		ID21
IoTivity Architecture		ID6
Modelo de referencia de la ITU-T		ID19, ID22
Modelo de referencia de SmartSantander		ID29
Modelo de referencia en capas para la gestión de datos IoT		ID23
Modelo propuesto por Atzori et al. (Atzori et al., 2010)		ID26
Modelo propuesto por Gubi et al. (Gubbi et al., 2013)		ID26
Arquitectura de CISCO		ID26
Reference Architecture Model Industrie (RAMI 4.0)		ID37, ID39
SITAC Project		ID25

Fuente: El autor

Para la evaluación de las AR, se tomaron las 40 arquitecturas de referencia obtenidas en la revisión de la literatura (Ver Tabla 12) y se seleccionaron aquellas que tienen un enfoque genérico. Para determinar la completitud de dichas arquitecturas, se evaluaron usando como base el RAModel.

Figura 6. AR para IoT más citadas en la literatura



Fuente: El autor

Convenciones usadas en la Figura 6, a saber:

- AR1: Modelo de referencia de la ITU-T
- AR2: Industrial Internet Reference Architecture (IIRA)
- AR3: Reference Architecture Model Industry (RAMI 4.0)
- AR4: Otras arquitecturas de referencia propuestas en la literatura
- AR5: IoT-A Project, cuyo segmento se divide en: IoT-A: Internet of Things Architecture, IoT-ARM: Architecture Reference Model y IoT-CRM: Communication Reference Model

El *Reference Architecture Model - RAModel* propuesto por Nakagawa, Oquendo, & Becker (2012), es un modelo de referencia para la evaluación de arquitecturas de referencia que contempla los elementos mínimos que una arquitectura de referencia debería tener, organizados por tipos y relaciones. Según los autores, el RAModel tiene cuatro formas de ser aplicado: *(i)* análisis de arquitecturas de referencia, para identificar elementos faltantes; *(ii)* análisis comparativo de arquitecturas de referencia, para seleccionar una entre un conjunto de ellas; *(iii)* como base para el establecimiento de nuevas arquitecturas de referencia; *(iv)* como apoyo al diseño de SPL (Software Product Line). El RAModel está dividido en cuatro grupos de

elementos, a saber: (i) dominio, (ii) aplicación, (iii) infraestructura y (iv) elementos transversales.

- El grupo del dominio (GD) contiene información específica del espacio de acción humana en el mundo real, como las legislaciones, estándares y procesos de certificación, los cuales impactan sistemas y AR relacionadas con ese dominio. Los elementos del dominio son: legislaciones, estándares y regulaciones, atributos de calidad como fácil mantenimiento, portabilidad y escalabilidad, y la verificación si el sistema desarrollado a partir de la arquitectura de referencia cumple con las legislaciones, estándares y regulaciones existentes.
- El grupo de aplicación (GA) contiene elementos que proporcionan una buena comprensión sobre la AR, sus capacidades y limitaciones. También contiene elementos relacionados con las funcionalidades que podrían estar presentes en los sistemas creados a partir de la AR. Los elementos de la aplicación son: las restricciones, el dominio de los datos, los requerimientos funcionales, los objetivos, el alcance, los riesgos, las limitaciones y las necesidades que deben ser cubiertas por la AR.
- El grupo de infraestructura (GI) hace referencia a los elementos que son usados para construir un sistema basado en la AR los cuales son responsables de automatizar procesos, actividades y tareas de un dominio específico. Los elementos de la infraestructura son: buenas prácticas y lineamientos, elementos de Hardware y Software, y la estructura general de la AR representada por los estilos arquitecturales.
- El grupo de elementos transversales (GT) hace referencia a un conjunto de elementos que generalmente se distribuyen entre los elementos de los grupos anteriormente mencionados. Se ha identificado que la comunicación (interna y externa) en los sistemas construidos a partir de la arquitectura de referencia, así como la terminología y las decisiones de dominio están presentes de una manera dispersa y mezclada cuando se describen otros grupos y, por lo tanto, se convierten en elementos transversales.

4.1.6 Reporte. Las arquitecturas seleccionadas fueron revisadas de acuerdo a la información provista por los estudios obtenidos, apoyadas en la información encontrada en sus respectivas páginas Web. En la Tabla 13 se presenta el cuadro comparativo realizado con cada una de las características propuestas por Nkagawa et al.(2012).

Se puede concluir que la IoT-A es una de las arquitecturas que tiene más documentación abierta y disponible tiene para los desarrolladores. Además, es una de las más referenciadas en la literatura. Para dar respuesta a la pregunta relacionada con el diseño de las AR genéricas para IoT la cual fue establecida en el protocolo de búsqueda, a continuación se hace una descripción de cada una de las arquitecturas evaluadas en la Tabla 13.

Tabla 13. Evaluación de arquitecturas de referencia usando el RAModel

Características del RAModel		ITU-T	IoT-A	WS02	Casagras	Smart Santander	SiTAC	CISCO
GD	Legislación, estándar y regulación	X	X		X			
	Atributos de calidad	X	X	X		X		
	Sistema de cumplimiento		-					
GA	Alcance	X	X	X	X	X		
	Requerimientos funcionales	X	X	X	X	X		
	Dominio de datos		X			X		
	Restricciones		X					
	Riesgos		X	X				
	Objetivos y necesidades	X	X	X	X			X
	Limitaciones		-		-			
GI	Elementos de software		X			X		X
	Elementos de hardware	X	X	X	X	X		X
	Mejores prácticas y directrices		X	-	-	X		
	Estilo arquitectural	X	X	X	X	X	X	X
GT	Decisión		X	X				
	Terminología	X	X		X			-
	Comunicación	X	X	X		X		
Total		9	16	9.5	8	9	1	4.5

Convenciones:
(GD) grupo de dominio, (GA) grupo de aplicación, (GI) grupo de infraestructura, (GT) grupo de elementos transversales.
(x) contiene la característica, puntaje 1.0.
(-) la característica no es específica, puntaje 0.5.
Los espacios vacíos significan que no se obtuvo evidencia documental de la característica.

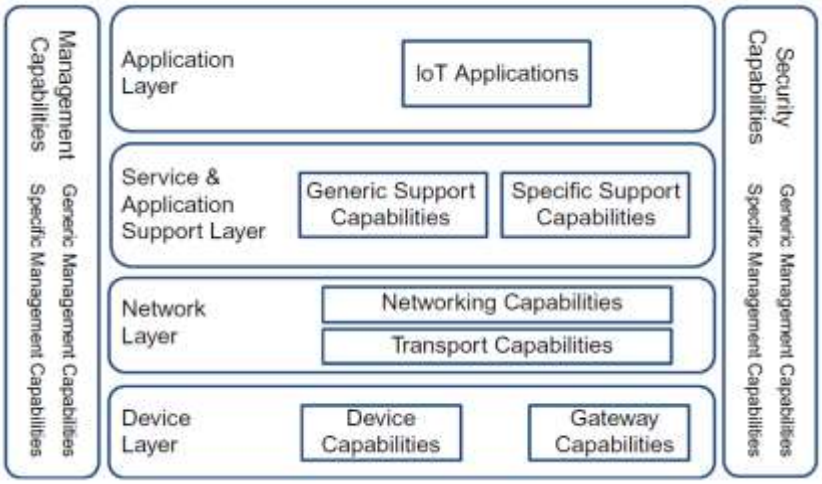
Fuente: El autor

ID19 e ID22 describen que a partir del año 2011 la ITU-T tomó la iniciativa de desarrollar normas mundiales con estándares específicos que abordaran cuestiones relacionadas con IoT. En el 2012 se expone públicamente la *ITU-T Recommendation, Y.2060*, la cual proporciona una visión general del espacio de la IoT con respecto a la ITU-T. Esta recomendación presenta una descripción general de alto nivel del modelo de dominio IoT y el modelo funcional IoT como un conjunto de capacidades de servicio similares al modelo de referencia M2M descrito por el *European Telecommunications Standards Institute - ETSI*[20].

[20] Para más información consultar en http://www.etsi.org/

La arquitectura propuesta por la ITU-T (ver Figura 7) está compuesta por 4 capas básicas: dispositivos, red, servicios y aplicación; y 2 transversales: capacidades de gestión y seguridad. La capa de dispositivos permite que los dispositivos físicos se conecten directamente o a través de puertas de enlace *(*en inglés, *gateway)* a una red de comunicación que les permite detectar, actuar y procesar información con otros dispositivos, servicios y aplicaciones.

Figura 7. Arquitectura de referencia de la ITU-T



Fuente: (International Telecommunication Union - ITU, 2012)

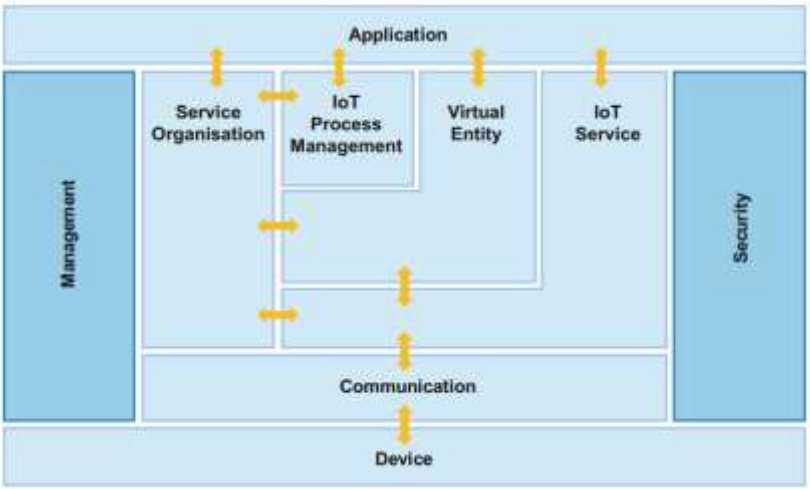
La capa de red proporciona capacidades de red tales como gestión de la movilidad, autenticación, autorización, contabilidad y transporte para los datos del servicio IoT. La Capa de Soporte de Servicio y Aplicación consiste en capacidades genéricas de servicio utilizadas por todas las aplicaciones de IoT, como procesamiento y almacenamiento de datos. Con respecto a las Capacidades del Servicio a partir de la Capa de Aplicación, el modelo ITU-T considera esta capa como el anfitrión de aplicaciones específicas para los diferentes dominios de IoT. En términos de capacidades de gestión genéricas, se incluye la administración de errores, configuración, contabilidad, rendimiento, seguridad, la gestión de dispositivos, gestión de topología de red y gestión del tráfico. Las características de seguridad se agrupan genéricamente como la compatibilidad e integridad de mensajes, y otras de forma específica, como las que se adaptan a la aplicación de un dominio en particular. Cabe resaltar que no se evidenció documentación completa de esta arquitectura de referencia en términos de: cumplimiento del sistema, dominio de los datos, restricciones, riesgos, limitaciones, elementos de software, buenas prácticas y lineamientos y elementos transversales de decisión.

De acuerdo con ID8, en el 2009 un grupo de investigadores pertenecientes a más 20 compañías e instituciones enfocadas a la investigación en IoT se enfocaron en

el diseño de una arquitectura que sirviera como base para proveer una estructura común y lineamientos para tratar aspectos de desarrollo, uso y análisis de sistemas para IoT, dando lugar a la denominada *IoT Architecture Project - IoT-A.* Esta es la arquitectura que más información detallada y específica existe en la literatura. Además ha sido usada como base para desarrollar sistemas u otras metodologías de diseño arquitecturales, como lo descrito en ID12 e ID21.

De acuerdo con ID14, el modelo de dominio IoT que describe la IoT-A está compuesto por nueve grupos funcionales (GF) distribuidos de la siguiente a manera, a saber: (i) siete grupos longitudinales de funcionalidad (Aplicación, Organización del servicio, Gestión de procesos IoT, Entidad Virtual, Servicio IoT, Comunicación, Dispositivo) complementado por; (ii) dos grupos de funcionalidad transversales (Gestión y Seguridad). Estos grupos transversales proporcionan funcionalidades requeridas por cada uno de los grupos longitudinales (Ver Figura 8). Las políticas que rigen los grupos transversales no solo se aplican a ellos, sino también a los grupos longitudinales.

Figura 8. Arquitectura de referencia IoT-A – Grupos funcionales



Fuente: ID7

La IoT-A contempla estos GF (Ver Tabla 14) tomando como base las siguientes características:

- A partir de las abstracciones principales identificadas en el modelo de dominio (Entidades Virtuales, Dispositivos, Recursos y Usuarios) se derivan las GF "Aplicación", "Entidad Virtual", "Servicio IoT" y "Dispositivo".
- Con respecto a la gran cantidad de tecnologías de comunicación que IoT necesita respaldar, se identifica la necesidad de una GF de "Comunicación".

- Los requisitos expresados por los stakeholders con respecto a la posibilidad de crear servicios y aplicaciones en la capa de aplicaciones IoT están cubiertos por los GF de "Gestión de procesos IoT" y "Organización del servicio".
- Para abordar consistentemente la preocupación expresada sobre la confianza, seguridad y privacidad en IoT, se identifica la necesidad de una GF transversal de "Seguridad".
- Finalmente, se requiere un GF transversal de "Gestión" para gestionar la interacción entre los grupos funcionales.

Tabla 14. Grupos funcionales de la IoT-A

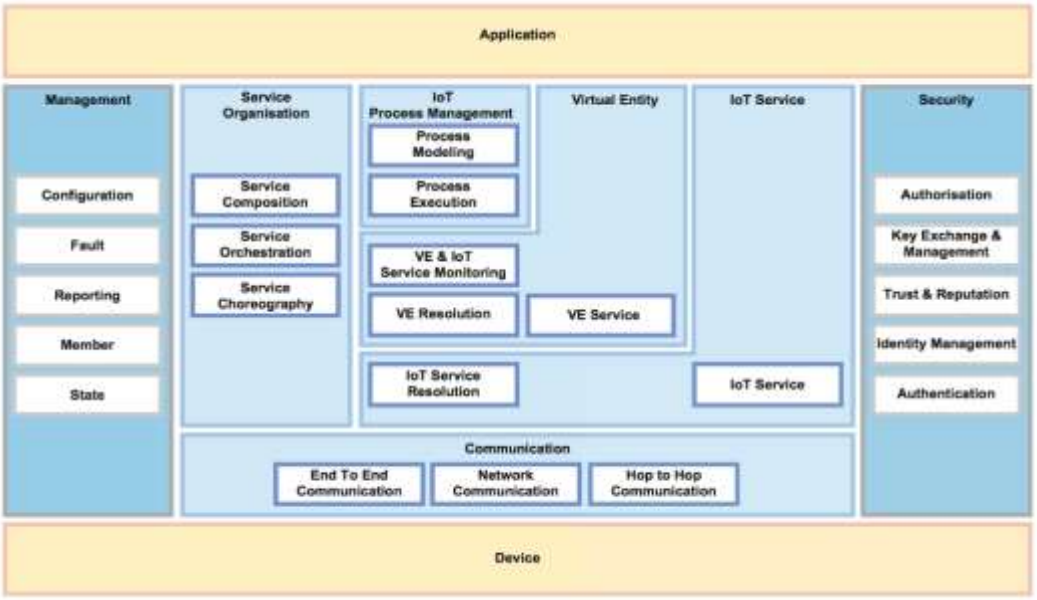
Grupo Funcional Longitudinal	
1. Organización del servicio (en inglés, *Service Organisation*)	Este GF se utiliza para componer y orquestar servicios con diferentes niveles de abstracción. Atiende las solicitudes de servicio de alto nivel dentro del mismo proceso de la arquitectura o incluso de aplicaciones externas (servicios alojados).
2. Gestión de procesos IoT (en inglés, *IoT Process Management*)	Las aplicaciones pueden utilizar las herramientas e interfaces definidas en este GF para mantenerse en el nivel conceptual (abstracto) de un proceso, mientras que, al mismo tiempo, hacen uso de la funcionalidad relacionada con IoT sin necesidad de lidiar con la complejidad que trae los servicios de IoT. Este GF no tiene una taxonomía definida, sin embargo, se relaciona con aspectos a saber (i) Permiso: ¿qué se puede hacer?; (ii) Prohibición: ¿qué no se debe hacer?; (iii) Obligaciones: ¿qué se debe hacer?
3. Entidad virtual - EV (en inglés, *Virtual Entity*)	Representa aspectos relevantes del mundo físico en el mundo virtual. Este GF contiene funciones para interactuar con el sistema IoT, descubrir y buscar servicios que pueden proporcionar información o interactuar con las entidades virtuales. Además, contiene toda la funcionalidad necesaria para gestionar asociaciones, así como encontrar dinámicamente nuevas asociaciones y controlar su validez.
4. Servicio IoT (en inglés, *IoT Service*)	Este GF expone un recurso para hacerlo accesible a otras partes del sistema IoT. Un servicio IoT se puede invocar de forma síncrona respondiendo a las solicitudes de servicio o de forma asíncrona mediante el envío de notificaciones según las suscripciones realizadas previamente a través del servicio. Atiende requisitos IoT como descubrimiento, búsqueda y resolución de nombres.
5. Comunicación (en inglés, *Communication*)	Este GF es una abstracción que modela la variedad de esquemas de interacción derivados de las muchas tecnologías que pertenecen a los sistemas IoT y proporciona una interfaz común para el GF de Servicio IoT.
Grupo Funcional Transversal	
6. Gestión (en inglés, *Management*)	Es el responsable de la composición y seguimiento de las acciones relacionadas con todos los GF, es decir, se basa en la extrapolación de una serie de requisitos de comunicación para gestionar el comportamiento de todo el sistema (estos requisitos se pueden encontrar en la descripción de los componentes funcionales individuales).
7. Gestión (en inglés, *Security*)	Es el responsable de garantizar la seguridad y la privacidad de los sistemas que hacen parte de la arquitectura IoT.

Fuente: El autor

Dentro de cada GF existentes componentes funcionales (CF) cuyas interacciones entre sí dependen de las decisiones de diseño las cuales son tomadas con bases en los requisitos que una aplicación IoT requiere, ver

Figura 9.

Figura 9. Arquitectura de referencia IoT-A – Componentes funcionales



Fuente: ID7

En la
Tabla 15 se realiza una descripción de las funciones que tienen los componentes de cada grupo funcional como parte de la estructura de la IoT-A.

Tabla 15. Componentes funcionales de la IoT-A

GF	CF	Descripción
GF1	Servicio de Orquestación	• Resuelve los servicios IoT que son adecuados para satisfacer las solicitudes de servicio procedentes del CF de ejecución de procesos o de los usuarios
	Composición del Servicio	• Resuelve los servicios que se componen de los servicios IoT y otros servicios con el fin de crear servicios con una funcionalidad extendida. • Tiene dos funciones principales: (1) soportar composiciones de servicio flexibles y (2) aumentar la calidad de la información.
	Servicio Coreografía	• Ofrece un intermediario que maneja la comunicación de publicación/suscripción entre servicios.

GF	CF	Descripción
GF2	Modelado de procesos	• Proporciona un entorno para el modelado de procesos empresariales de IoT que serán serializados y ejecutados en el CF de Ejecución de Procesos. • Proporciona las herramientas necesarias para modelar procesos usando la notación estandarizada.
	Ejecución de Procesos	• Ejecuta los procesos que han sido modelados en el anterior CF, utilizando servicios de IoT que están orquestados en la capa de organización de servicio. • Implementa modelos de procesos en los entornos de ejecución. • Alinea los requisitos de la aplicación con las capacidades de servicio. • Una vez resuelve los servicios IoT, este CF puede invocar dichos servicios.
GF3	Resolución de EV	• Proporciona las funcionalidades al usuario IoT para recuperar asociaciones entre EV y los servicios de IoT. • Descubrimiento de asociaciones nuevas y en su mayoría dinámicas entre EV y servicios asociados, considerándose la ubicación, proximidad y otra información de contexto. • Permite buscar servicios relacionados con EV. • Permite gestionar asociaciones: insertar, borrar y actualizar asociaciones entre un EV y los servicios IoT que están asociados a la EV.
	Monitoreo del servicio EV e IoT	• Es responsable de encontrar automáticamente nuevas asociaciones, las cuales se insertan en el CF de una EV. Se pueden derivar nuevas asociaciones basadas en asociaciones existentes, descripciones de Servicio e información sobre EV.
	Servicio EV	• Se encarga de los servicios de entidad, el cual representa un punto de acceso general a una entidad particular, ofreciendo medios para aprender y manipular el estado de la entidad a través de operaciones que permiten leer y/o actualizar los valores de los atributos de las entidades.
GF4	Servicio IoT	• Expone un recurso para hacerlo accesible a otras partes del sistema IoT. • Principales funcione: (1) devolver información proporcionada por un recurso de forma síncrona, (2) aceptar información enviada a un recurso para almacenar la información o configurar el recurso o para controlar un dispositivo accionador y (3) suscribirse a información, es decir, devolver la información proporcionada por un recurso de forma asíncrona.
	Resolución de servicio IoT	• Proporciona todas las funcionalidades necesarias para que el usuario encuentre y pueda ponerse en contacto con los servicios IoT.

GF	CF	Descripción
GF5	Comunicación *hop-to-hop*	• Proporciona la primera capa de abstracción de la tecnología de comunicación física del dispositivo para permitir el uso y la configuración de cualquier tecnología de capa de enlace diferente. • Sus principales funciones son transmitir una trama desde el FC comunicación en red al FC comunicación *hop-to-hop* y desde un Dispositivo al CF de comunicación *hop-to-hop*. • Los argumentos para la transmisión de trama se pueden establecer; ejemplos de argumentos incluyen: fiabilidad, integridad, cifrado y control de acceso. • El CF de comunicación *hop-to-hop* también es responsable de enrutar un marco y gestionar la cola de tramas y establecer el tamaño y las prioridades de las colas de trama de entrada y salida.
	Comunicación en red	• Se encarga de habilitar la comunicación entre redes a través de localizadores (direccionamiento) y resolución de ID. • Incluye enrutamiento y convergencia a través de traducciones de protocolos de red. • Trasmite un paquete desde los otros dos CF de este GF hacia el CF de comunicación en red. • Los argumentos para la transmisión de paquetes pueden configurarse y ejemplos de argumentos incluyen: fiabilidad, integridad, cifrado, direccionamiento unicast/multicast y control de acceso.
	Comunicación *end-to-end*	• Se encarga de la totalidad de la abstracción de comunicación de extremo a extremo, lo que significa que se encarga de la transferencia fiable, el transporte y las funcionalidades de traducción, apoyo proxies/pasarelas y de ajuste de parámetros de configuración cuando la comunicación atraviesa diferentes entornos de red. • Los argumentos para el mensaje se pueden configurar y los ejemplos incluyen: fiabilidad, integridad, cifrado, control de acceso y multiplexación.
GF6	Autorización	• Esta CF es una interfaz para gestionar políticas y realizar decisiones de control de acceso basadas en políticas de control de acceso. Tiene dos funcionalidades por defecto, a saber: (i) Determinar si una acción está autorizada o no y (ii) Administrar políticas, como agregar, actualizar o eliminar una directiva de acceso.
	Gestión e intercambio de llaves	• Este CF permite las comunicaciones seguras entre dos o más componentes de la IoT-A que no tienen conocimiento inicial entre sí o cuya interoperabilidad no está garantizada, asegurando la integridad y confidencialidad. A esta CF se le atribuyen dos funciones, a saber: (i) Distribuir las claves de forma segura y (ii) Registrar las capacidades de seguridad.
	Confianza y Reputación	• Este CF recopila las puntuaciones de reputación de los usuarios y calcula los niveles de confianza del servicio. Tiene dos funciones predeterminadas, a saber: (i) Solicitar información de reputación y (ii) Proporcionar información de reputación.
	Gestión de Identidad	• Aborda las cuestiones de privacidad al emitir y administrar seudónimos e información accesoria a sujetos de confianza para que puedan operar (usar o proporcionar servicios) de forma anónima.
	Autenticación	• Participa en la autenticación de usuarios y servicios. Las dos funcionalidades proporcionadas por este CF son: (i) autenticar a un usuario basado en credenciales proporcionadas y (ii) verificar si una aseveración proporcionada por un usuario es válida o no válida.

GF	CF	Descripción
GF7	Configuración	• Este CF es el responsable de inicializar la configuración del sistema, como recolectar y almacenar la configuración de CF y dispositivos; y también del seguimiento de los cambios de configuración y la planificación para la extensión futura del sistema. Sus funciones principales son: (i) Recuperar una configuración y (ii) Configurar la configuración.
	Fallo	• Su objetivo es identificar, aislar, corregir y registrar las fallas que ocurren en el sistema IoT.
	Informes	• Uno de los muchos objetivos de informes concebibles es determinar la eficiencia del sistema actual.
	Miembro	• Es responsable de la gestión de la membresía y la información asociada de cualquier entidad relevante (GF, CF, EV, Servicio IoT, dispositivo, aplicación, usuario) a un sistema IoT.
	Estado	• Este CF monitorea y predice el estado del sistema IoT.

Fuente: El autor

ID11 describe que la compañía WSO2[21] propuso una arquitectura de referencia para los *software* proporcionar a un punto de partida para arquitectos de sistemas desarrollar IoT. La enfocados en contiene arquitectura abarca capas que dispositivos desde los recursos hasta los la nube para necesarios en interactuar y administrar estos dispositivos (ver Figura 10).

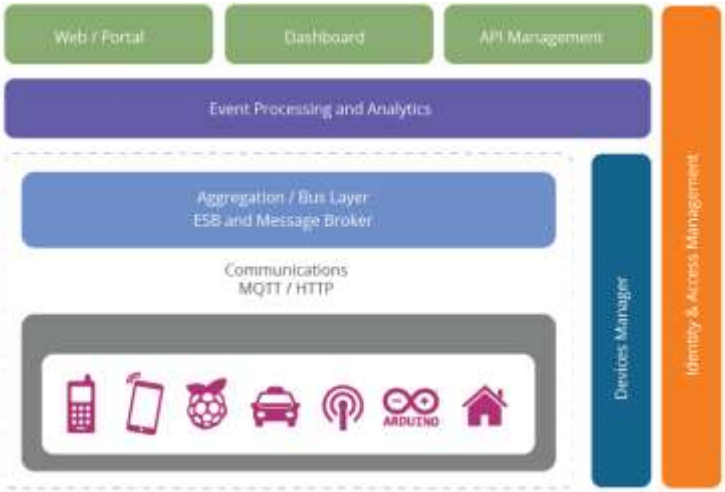


Figura 10. Arquitectura de referencia de WSO2

[21] Para más información consultar en https://wso2.com

Fuente: (Fremantle, 2015)

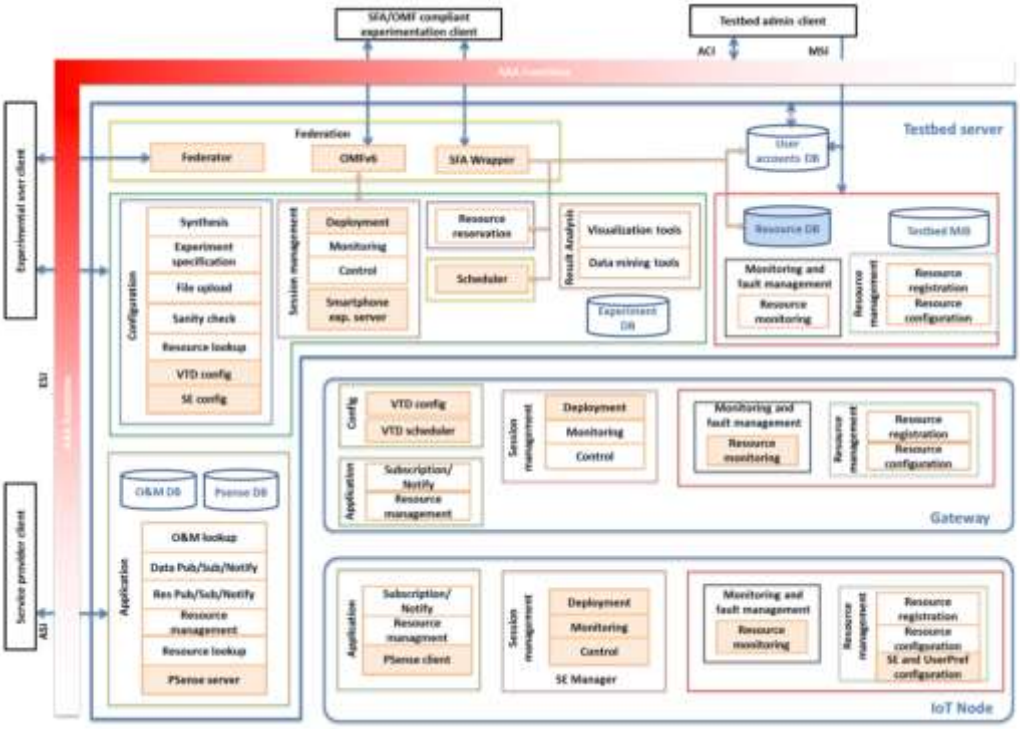
Particularmente, en la literatura se encuentra limitada documentación dado que la compañía que la desarrolla no se centra en detallar cómo debería funcionar un determinado cliente-servidor, *hardware* o arquitectura en la nube (manejo de tecnología propietaria). De acuerdo con Fremantle (2015), la arquitectura está compuesta por 5 capas básicas: dispositivos, comunicaciones, agregación/bus, analítica y procesamiento de eventos, comunicaciones externa; y 2 transversales: gestión de dispositivos y gestión de la identidad y el acceso. La capa inferior de la arquitectura es la capa del dispositivo. Los dispositivos pueden ser de varios tipos, pero para poder ser considerados como dispositivos IoT, deben tener algunas comunicaciones que se conectan indirectamente o directamente a Internet. A través de la capa de comunicación se realiza la conectividad de los dispositivos con la nube a través de protocolos como HTTP, MQTT, CoAP. La capa de agregación/bus agrega y combina comunicaciones de diferentes dispositivos y direcciona las comunicaciones a un dispositivo específico a través del *gateway.* Además, esta capa se adapta a protocolos heredados, proporciona alguna correlación y mapeo simple a partir de diferentes modelos de correlación. La capa de analítica recibe los eventos del bus para procesarlos y tomar decisiones sobre estos. Además, contiene una capa que facilita la creación de interfaces, portales basados en la Web, comunicación M2M, análisis y procesamiento de eventos externos. La capa de gestión de dispositivos es manejada por un servidor (administrador), el cual proporciona un control tanto individual como general de los dispositivos. También administra de forma remota el software y las aplicaciones implementadas en el dispositivo. La capa de gestión de la identidad y el acceso incluye el directorio de usuarios y políticas para el control de acceso.

De acuerdo con ID1, la arquitectura de referencia de CASAGRAS fue diseñada en primera instancia para desarrollar sistemas basados en RFID. Con IoT en crecimiento, los investigadores del proyecto vieron una oportunidad de desarrollo de su arquitectura para conectar los dispositivos a Internet. Cabe resaltar que esta arquitectura no contempla características para el dominio de los datos, elementos de software y de cumplimiento del sistema. Está compuesta por 3 capas básicas: (i) dispositivos, encargada de identificar los objetos físicos y entregar los datos sensados; (ii) puerta de enlace, conecta los dispositivos con los (iii) sistemas de gestión de información, quienes proveen la plataforma funcional para soportar las aplicaciones y servicios.

De acuerdo con ID29, SmartSantander[22] es un modelo como resultado de una investigación experimental a escala mundial única en el mundo que provee aplicaciones, servicios y tecnologías IoT para una ciudad inteligente. Cuenta con una arquitectura robusta (ver
Figura 11) diseñada a partir de los requisitos funcionales y no funcionales provistos por los *stakeholders,* proveedores de servicios e instalaciones, experimentadores y usuarios finales del sistema.

En la literatura no está disponible información detallada de los requisitos funcionales y no funcionales de esta arquitectura. Su principal objetivo es facilitar a los usuarios pruebas funcionales para diferentes casos de uso. Cabe resaltar que la arquitectura está compuesta por 3 niveles, a saber: *(i)* Nodos IoT, responsables de detectar parámetros como la temperatura, luz, presencia, ruido, según corresponda. La mayoría de ellos están integrados en los repetidores, y otros se comunican de forma inalámbrica; *(ii)* repetidores, están ubicados por encima del suelo y se comportan como nodos de reenvió y; *(iii)* puertas de enlace, revisa la información enviada por los nodos y repetidores a través del *protocolo 802.15.4*.

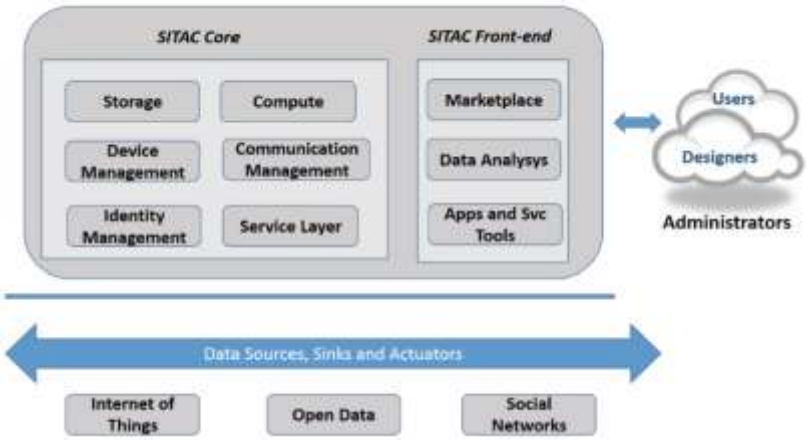
Figura 11. Arquitectura de referencia de SmartSantander Project



Fuente: Alex Gluhak et al. (Alex Gluhak et al., 2013)

[22] Para más información consultar en http://www.smartsantander.eu

Esta información puede almacenarse localmente o enviarse a servidores centrales para amanecerla y posteriormente servir como banco de pruebas, al mismo tiempo que se está ejecutando un servicio al usuario final.



De acuerdo con ID25, el proyecto SiTAC[23] tiene como objetivo crear una arquitectura unificadora y un ecosistema que comprenda plataformas, herramientas y metodologías, que permita la conexión y cooperación de diferentes entidades conectadas a la red, ya sean sistemas, máquinas, dispositivos o personas. Particularmente, su arquitectura admite entornos distribuidos, heterogéneos y aborda un espectro de configuraciones basadas en red. Un usuario puede seleccionar una infraestructura de red que se ajuste a sus necesidades específicas. Además, ofrece interoperabilidad semántica y técnicas de concienciación del contexto para facilitar la entrega de información en el lugar y momento adecuado. La vista funcional de la arquitectura propuesta se visualiza en la Figura 12, la cual está dividida en 2 partes principales, a saber: *(i) core*, contiene las funciones principales y genéricas para el cómputo, procesamiento, almacenamiento, administración de identidades, comunicación y dispositivos y; *(ii) front-end,* donde los servicios centrales se unen para proporcionar funciones a los usuarios.

Figura 12. Arquitectura de referencia de SiTAC

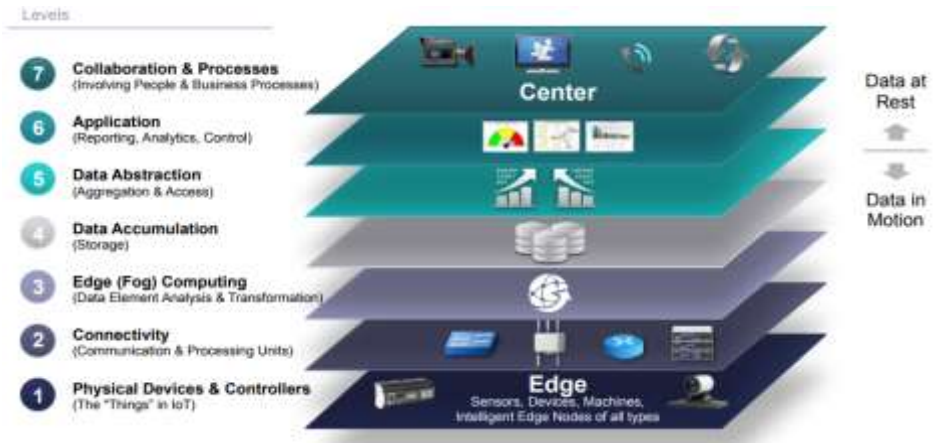
[23] Para más información consultar en https://itea3.org/project/sitac.html

Fuente: ID12

Un modelo de referencia que no se encontró específicamente en la literatura estudiada es el presentado por CISCO (2014), el cual está formado por 7 niveles (verFigura 13) y cada uno contiene terminología que se puede estandarizar para crear un marco de referencia aceptado globalmente. Este modelo al pertenecer a una empresa privada tiene restricción de documentación disponible en la Web.

- Nivel 1: dispositivos físicos y controladores, incluyen una amplia gama de dispositivos de finales (endpoint) que envían y reciben información según las necesidades especificadas desde su inclusión en el modelo.
- Nivel 2: conectividad, incluye transmisiones entre dispositivos a través de la red y entre la red y el procesamiento de información de bajo nivel que ocurre en el nivel 3.
- Nivel 3: computación de borde, se centra en el análisis y la transformación de datos de gran volumen de tal forma que el sistema no se sature antes de salir a la nube.
- Nivel 4: acumulación de datos, captura los datos y los deja en reposo, para que las aplicaciones puedan acceder a ellos en tiempo no real, es decir, convierte los datos basados en eventos al procesamiento basado en consultas.
- Nivel 5: abstracción de datos, se centra en la representación de los datos y su almacenamiento de tal forma que se permita el desarrollo de aplicaciones más simples y mejoradas.
- Nivel 6: aplicación, representa la interfaz de comunicación con el usuario. Estas varían en función del dominio, la naturaleza de los datos del dispositivo y las necesidades comerciales.
- Nivel 7: colaboración y procesos, representa la interacción entre diferentes aplicaciones y procesos de negocio para la toma de decisiones.

Figura 13. Arquitectura de referencia de CISCO



Fuente: (CISCO, 2014)

4.1.7 Conclusiones de las arquitecturas de referencia para IoT. En general, las arquitecturas de referencia para IoT estudiadas tienen en común las siguientes características:

- Establecen una comprensión común del paradigma IoT con un conjunto de componentes que identifican sus conceptos principales, relaciones y limitaciones.
- La AR sirve como base para facilitar el desarrollo de sistemas para IoT.
- La estructura genérica de las arquitecturas está comprendida en capas/niveles que comienzan desde el dispositivo (hardware) hasta la aplicación (software).
- Por lo general, tienen 2 capas/niveles transversales: seguridad y gestión de datos, red o dispositivos. Estas pueden ser usadas en los demás componentes de la estructura horizontal de la AR de acuerdo a las necesidades específicas de cada una.
- Comparten requisitos relevantes para IoT como horizontalidad (soporte a diferentes dominios de aplicación), heterogeneidad, escalabilidad, conectividad, gestión de identidad, gestión de dispositivos, gestión de comunicación, seguridad.

Las AR para IoT incorporan las mejores prácticas aceptadas del paradigma, que normalmente sugieren el método de implementación de las aplicaciones, el trabajo de desarrolladores, ya que los requisitos atendidos por estas AR son genéricos y pueden ser utilizados para cualquier dominio de aplicación. Con esta revisión se obtuvo los componentes y funcionalidades básicas que tiene cualquier aplicación IoT, y dado que el objetivo de este proyecto es diseñar un *middleware* para IoT que facilite el desarrollo de aplicaciones con interfaces de voz, el *middleware* propuesto también debe atender esas funcionalidades. Aunque es

limitada la documentación que tienen la mayoría de las AR para IoT estudiadas, se tomarán como referencia los requisitos funcionales y no funcionales usados en la IoT-A los cuales están descritos en la Tabla 14 y
Tabla 15.

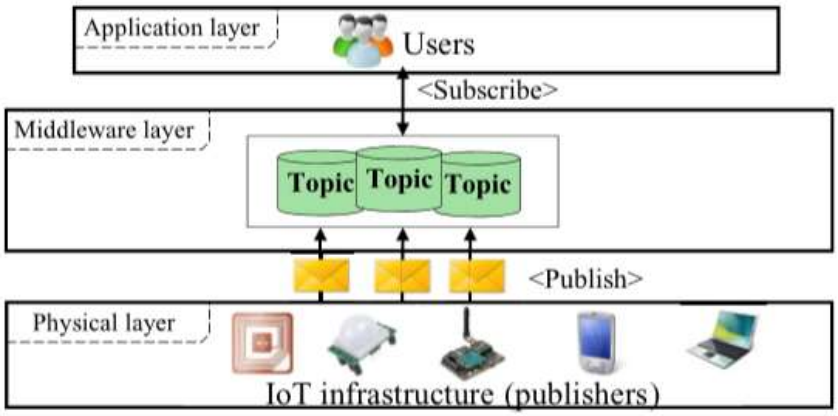
4.2 ARQUITECTURAS *MIDDLEWARE* PARA IOT

En esta sección se usó como base el protocolo presentado en la sección 2.3, realizado en tres fases: *(i)* planificación, *(ii)* conducción y; *(iii)* reporte de los resultados obtenidos.

De acuerdo a la literatura, existen diferentes tipos de plataforma *middleware* desarrolladas para IoT categorizadas según su enfoque de diseño, los cuales se describen a continuación (Chelloug & El-Zawawy, 2017; Mineraud et al., 2016; Ngu et al., 2017; Razzaque et al., 2016).

4.2.1 *Middleware* basado en eventos. Se considera un "evento" a un cambio significativo de estado. Este tipo de *middleware* es viable cuando una aplicación tiene movilidad y fallas comunes. Los eventos se propagan desde los componentes de la aplicación emisora (productores) a los componentes de la aplicación receptora (consumidores), ver Figura 14.

Figura 14. Arquitectura de un *middleware* basado en eventos

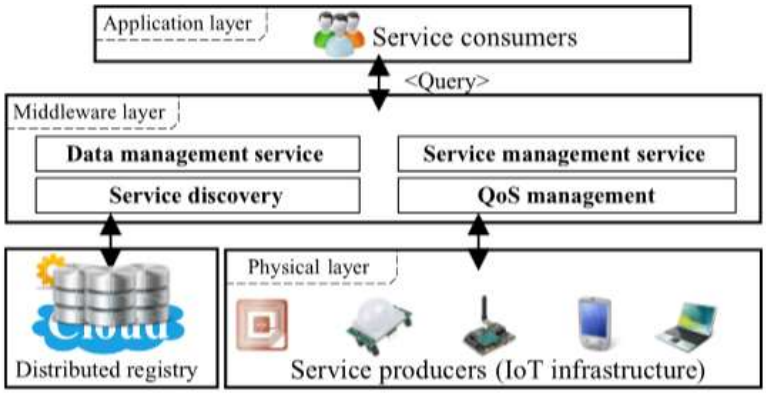


Fuente: Tomado de (Razzaque et al., 2016)

Tiene ventajas como la utilización de patrones para la publicación/suscripción (asíncrono), escalabilidad, procesamiento en tiempo real con mínimo retardo. Sin embargo, no proveen interoperabilidad, adaptabilidad y conciencia del contexto.

4.2.2 *Middleware* orientado a servicios. Está basado en la arquitectura orientada a servicios (en inglés, *Service Oriented Architecture* - SOA) la cual proporciona abstracciones para el *hardware* subyacente a través de un conjunto de servicios que necesitan las aplicaciones. Los servicios se pueden diseñar, implementar e integrar en un marco de trabajo teniendo en cuenta la incorporación de un proveedor de servicios (alojar servicios), un consumidor de servicios (representa cualquier aplicación) y un registro de servicios (acciones) para ofrecer un entorno flexible y fácil para el desarrollo de aplicaciones (Ver Figura 15). Tiene ventajas como el descubrimiento de servicios, composición de servicios y reutilización de servicios. Sin embargo, la administración de código no se da fácilmente.

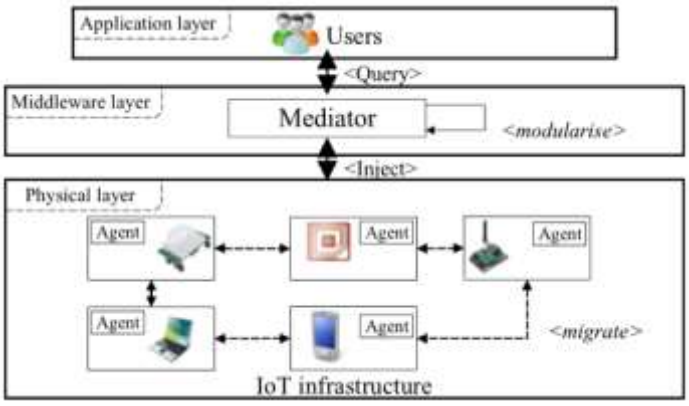
Figura 15. Arquitectura de un *middleware* basado en servicios



Fuente: Tomado de (Razzaque et al., 2016)

4.2.3 *Middleware* basado en agentes. En este enfoque las aplicaciones se dividen en programas modulares para facilitar la inyección y distribución de la red, utilizando los agentes móviles. Al migrar de un nodo a otro, los agentes mantienen su estado de ejecución (Ver Figura 16). Esto facilita el diseño de sistemas descentralizados capaces de tolerar fallas parciales. Tiene como ventajas la gestión de recursos (reducción de carga de red y reducción de latencia de red), gestión de código (ejecución asíncrona y autónoma y encapsulación de protocolo), disponibilidad, fiabilidad (robustez y tolerancia a fallos) y adaptabilidad. Sin embargo, su mayor desventaja es la baja interoperabilidad entre los recursos.

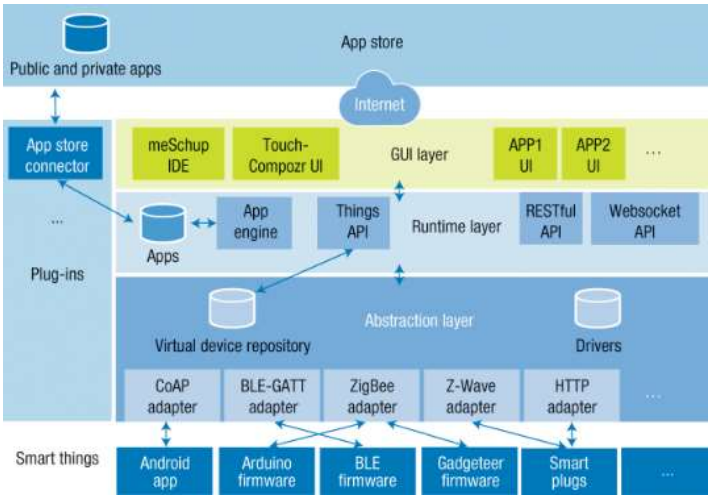
Figura 16. Arquitectura de un *middleware* basado en agentes



Fuente: Tomado de (Razzaque et al., 2016)

4.2.4 *Middleware* basado en la nube. Se enfoca en la provisión de servicios de *hosting* a través de Internet. Limita a los usuarios en el tipo y la cantidad de dispositivos IoT que pueden implementar, pero les permite conectarse, recopilar e interpretar datos con facilidad, ya que los posibles casos de uso pueden ser determinados y programados a priori (Ver Figura 17).

Figura 17. Arquitectura de un *middleware* basado en la nube



Fuente: Tomado de (Kubitza & Schmidt, 2017)

Los componentes funcionales del *middleware* están limitados a lo que está disponible en la nube y varía ampliamente entre las plataformas basadas en la nube. Normalmente, esas funcionalidades se exponen como un conjunto de API. Las

funcionalidades proporcionadas podrían ser tan simples como un sistema de almacenamiento de muy alto rendimiento o un motor de cómputo muy poderoso con herramientas de monitoreo y análisis predefinidas. Los servicios de los dispositivos de IoT disponibles en la nube solo se pueden acceder o controlar a través de aplicaciones proporcionadas por el proveedor o de las API RESTful admitidas en la nube. En este tipo de arquitectura, la seguridad depende del proveedor de servicio en la nube.

4.2.5 *Middleware* basado en actores. La arquitectura está basada en *plug and play*, es decir, tiene la capacidad de configurar automáticamente los dispositivos al conectarlos. Expone una variedad de dispositivos IoT como actores reutilizables para distribuirlos en la red de tal forma que cada uno de ellos pueda realizar cálculos computacionales. No tienen un estándar particular para la comunicación entre dispositivos IoT, lo que facilita la interoperabilidad y escalabilidad. Sin embargo, tiene como desventaja la baja seguridad del sistema.

4.2.6 Conclusiones de las arquitecturas de referencia para IoT. Cabe resaltar que algunos *middleware* usan la combinación de diferentes enfoques de diseño. Usualmente estos enfoque híbridos tienen mejor desempeño que los que pertenecen a un único enfoque (Razzaque et al., 2016).

De acuerdo a lo presentado en la Tabla 7, el *middleware* al ser una herramienta *software,* ofrece administración de sus servicios entre la capa de dispositivos y aplicaciones. De forma transversal a esas capas ofrecen seguridad a todos los recursos. En la Figura 18 se ilustra la arquitectura genérica de un *middleware* para IoT.

Figura 18. Arquitectura genérica de un *middleware* para IoT

Fuente: El autor

Los *middleware* consultados no atienden todos los requisitos que IoT requiere ya que se basan de acuerdo al enfoque de diseño que cada uno tenga. Para el diseño del *middleware* propuesto en este proyecto, se tuvieron en cuenta los diferentes requisitos y enfoques de diseño extraídos a partir de la revisión de la literatura, Ver Tabla 5 y Tabla 6, de la sección 2.3.

4.3 SISTEMAS PARA EL RECONOCIMIENTO DEL HABLA - ASR

Se realizó una revisión en la literatura para conocer los estudios que abordan el diseño de sistemas para el reconocimiento del habla, identificar su arquitectura y requisitos funcionales.

4.3.1 Planeación. Para esto, se definieron las siguientes preguntas de la revisión:

- ¿Cuáles son los componentes que contempla la arquitectura del sistema para el reconocimiento del habla?
- ¿Cuáles son los requisitos funcionales que un sistema para el reconocimiento del habla contempla?

Para dar respuesta a estas preguntas se definió la siguiente cadena de búsqueda, la cual fue consultada en dos bibliotecas digitales en línea: SCOPUS[24] e IEEE[25].

TITLE-ABS-KEY (("Internet of things" OR IoT) AND "Speech recognition") AND PUBYEAR > 2011

Se tuvieron en cuenta algunos criterios de inclusión (CI) y exclusión (CE) para reducir progresivamente el número de estudios encontrados en la etapa de búsqueda, a una colección apropiada de documentos de alta calidad que fueran relevantes para responder a las preguntas planteadas en la revisión.

- CI1: El estudio fue publicado entre 2012 y 2018. Se establece a partir del 2012 dado que en ese año la ITU (2012) da a conocer la segunda definición formal de IoT y para esa fecha ya se habían establecido arquitecturas de referencia para el desarrollo de aplicaciones IoT.
- CI2: El estudio contiene información relacionada de sistemas para el reconocimiento del habla para IoT en el título, resumen y palabras clave.
- CI3: El estudio es un artículo, libro, capítulo de libro o una conferencia (conference proceedings).
- CI4: El documento fue escrito en inglés.

[24] Más información en la página Web https://www.scopus.com/
[25] Más información en la página Web http://ieeexplore.ieee.org/Xplore/home.jsp

- CE1: No se tiene acceso a todo el documento.
- CE2: El documento ya fue recuperado.
- CE3: El estudio contiene información trivial lo que dificulta responder a las preguntas planteadas.

4.3.2 Conducción. Se recuperaron 161 estudios: 85 de SCOPUS y 76 de IEEE. Inicialmente fueron seleccionados 42 basados en los criterios de inclusión y exclusión. Una vez recuperados estos documentos, se realiza una revisión más detallada del *full-text* para conocer como están diseñados los sistemas para el reconocimiento del habla propuestos. En la Tabla 16 se muestran los 13 artículos estudiados una vez se finaliza la revisión detallada.

Tabla 16. Artículos recuperados: sistemas para el reconocimiento del habla

ID	Título	Referencia
ID1	Multilingual Voice Control for Endoscopic Procedures	(Afonso, Laranjo, Braga, Alves, & Neves, 2015)
ID2	A novel strategy for controlling the movement of a Smart Wheelchair using Internet of Things	(Akash et al., 2014)
ID3	A Multimodal User Interface using the Webinos Platform to Connect a Smart Input Device to the Web of Things	(Baccaglini, Gavelli, Morello, & Vergori, 2015)
ID4	Design and Implementation of a Housekeeper System	(Bai et al., 2013)
ID5	An Embedded Voice Inquiry Experimental Platform for Temperature and Humidity Measurement on the Internet of Things	(Zhou, Liu, & Lin, 2012)
ID6	Speech only interface approach for personal computing environment	(Raveendran, Sanjeev, Paul, & Jijina, 2016)
ID7	A Voice-Controlled Multi-Functional Smart Home Automation System	(Mittal et al., 2015)
ID8	Gesture and Voice Control of Internet of Things	(Han & Rashid, 2016)
ID9	Enhancing wireless sensor networks with acoustic sensing technology: use cases, applications & experiments	(Hollosi, Nagy, Rodigast, Goetze, & Cousin, 2013)
ID10	Development of information living integrated by home appliances and Web service	(Kaneko et al., 2015)
ID11	Integrating unified communications and internet of mHealth things with micro wireless physiological sensors	(Keh et al., 2014)
ID12	Smart Robotic Assistant	(Kumar, Mishra, Makula, Karan, & Mittal, 2015)
ID13	The internet of speaking things and its applications to Cultural Heritage	(Marulli, Pareschi, & Baldacci, 2016)

Fuente: El autor

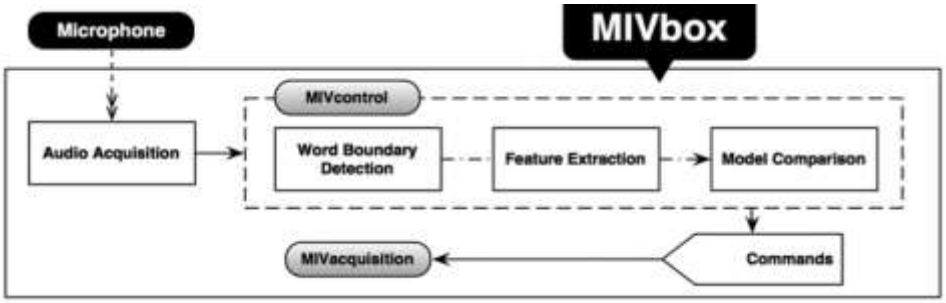
4.3.3 Reporte. El habla es el método de comunicación más importante que tienen la mayoría de los seres humanos. Actualmente la interacción de forma natural con

las computadoras usando el habla sin recurrir a la implementación de interfaces como los teclados o los dispositivos señaladores es posible gracias al sistemas para el reconocimiento automático del habla (en inglés, *Automatic Seepch Recognition - ASR*), que representa la función de modular una señal de voz a una serie de palabras (fonemas) con la ayuda de algoritmos realizados por un programa de computadora (EY, 2016).

En ID1 se presenta una solución para mejorar el flujo de trabajo relacionado con la toma de exámenes endoscópicos, puesto que en este procedimiento se requiere usar ambas manos para manipular botones y presionar un pedal para la captura de fotogramas. El módulo de *software* desarrollado se integró con el dispositivo MIVbox, el cual permite la adquisición, procesamiento y almacenamiento de los resultados endoscópicos. Como se observa en la Figura 19, el módulo para el reconocimiento de voz denominado MIVcontrol consta de dos fases: creación del modelo de texto y creación del modelo acústico.

El modelo de lenguaje es una descripción de alto nivel de todas las frases válidas (es decir, la combinación de palabras) en un determinado idioma. Se puede utilizar modelos estadísticos o gramática sin contexto. El modelo de lenguaje estadístico se crea automáticamente según la lista de comandos. El diccionario es un mapa para la interpretación de cada comando y los fonemas. Un fonema se define como la unidad básica de fonología, que se puede combinar para formar palabras. Dado que la lista de comandos requeridos es pequeña, todos los diccionarios se crearon manualmente. El modelo acústico se entrena utilizando *SphinxTrain* y asigna funciones de audio a los fonemas que representan, para aquellos incluidos en el diccionario. El sistema utiliza bibliotecas desarrolladas por el proyecto *PocketSphinx* para reconocer una pequeña cantidad de comandos. El módulo fue ajustado para el idioma portugués. Se obtuvo una tasa de error de 23.3% para el modelo en inglés y 29.1% para el portugués.

Figura 19. Arquitectura del sistema para el reconocimiento del habla – ID1



Fuente: Tomado de ID1

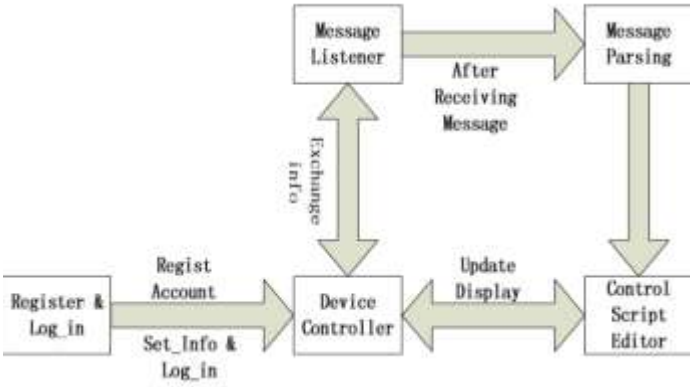
En ID2 se describe un prototipo basado en una silla de ruedas automatizada para facilitar la movilidad restringida a las personas. Los diferentes modos de control para el movimiento de la silla están disponibles en forma modular y el usuario puede incorporarlos según las necesidades específicas. Además de los controles como el joystick, el control de la barbilla, la activación por voz, el control a través del movimiento de la cabeza, a través de llamadas realizadas con un teléfono móvil, esta silla también se puede controlar de forma remota a través de Internet. Raspberry Pi[26] se usa como controlador principal tanto para el joystick como para el control de Internet. Particularmente, con el módulo de la voz, el paciente puede controlar la silla de ruedas usando algunas palabras clave predefinidas. La voz de los pacientes se graba continuamente como un archivo audible y de texto. Para los autores, es importante categorizar las palabras clave como: adelante, atrás, izquierda, derecha y detener.

En ID3 describen una arquitectura que permite la interoperabilidad entre dispositivos heterogéneos con sistemas operativos diferentes y una aplicación basada en tecnologías Web estándar que realiza funciones utilizando comandos de voz. Desarrollaron métodos y funciones para capturar el audio del micrófono en computadoras personales y teléfonos móviles, y ser procesado por un motor de reconocimiento de voz. Cuando se completa el proceso, el comando reconocido se envía de vuelta a la aplicación. La aplicación tiene una serie de comandos pre-mapeados y cada uno de ellos desencadena una acción diferente. Como trabajo futuro sugieren que la aplicación se puede ampliar agregando soporte para diferentes mecanismos de reconocimiento de voz como la API de voz a texto de Google para realizar el reconocimiento de comandos independientemente del usuario.

En ID4 desarrollan un sistema de control de electrodomésticos basado en la red, como se observa en la Figura 20. Hacen uso de la plataforma Atom como centro de control de forma local y remota los equipos en tiempo real a través de la tecnología inalámbrica y la interfaz de voz. El mecanismo utilizado en la plataforma móvil, que está conectado a Internet, es similar a la mensajería instantánea. La parte de control de voz se basa en el micrófono omnidireccional MST, el cual puede capturar la señal de sonido en un radio de 5 metros y mantener la calidad del sonido para el reconocimiento de voz. El reconocedor de voz incluye un algoritmo de última generación escrito en Java. Tiene la capacidad de reconocer el habla discreta y continua. La arquitectura del modelo de lenguaje incluye compatibilidad con modelos de idiomas reconocibles para ASCII y versiones binarias de *unigram*, *bigram*, *trigram, Java Speech API Grammar Format - JSGF* y gramáticas FST en formato ARPA.

[26] Más información en la página Web https://www.raspberrypi.org/

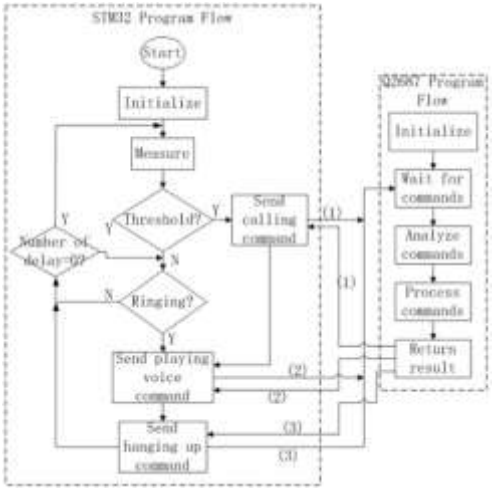
Figura 20. Arquitectura del sistema para el reconocimiento del habla – ID4



Fuente: Tomado de ID4

En ID5 presentan una plataforma experimental de investigación de voz integrada, la cual funciona a través de un chip de control de alto rendimiento STM32 que junto a un módulo de comunicación inalámbrica Q2687 controla el módulo de medición y utiliza la red GSM para realizar la función de enviar datos de medición al reproducir la voz en un teléfono específico. A través de una llamada, la plataforma se activa para medir la temperatura y la humedad. Luego transmite el resultado por voz a la persona (Ver Figura 21). Este sistema se caracteriza por su pequeño volumen, bajo costo, baja disipación de potencia y fácil utilización. El resultado de la prueba muestra que la plataforma es estable y práctica.

Figura 21. Diagrama de flujo reconocedor de voz – ID5



Fuente: Tomado de ID5

En ID6 diseñan una interfaz controlada por voz a través de un computador personal. Las aplicaciones implementadas incluyen correo electrónico, reproductor de multimedia, búsqueda Web y un administrador de archivos. Cada una de estas aplicaciones se implementa de forma limitada para demostrar los principios del diseño, comprender las limitaciones y el alcance futuro del sistema. El sistema comienza con la entrada de voz del usuario. La entrada es procesada por el motor de reconocimiento de voz para proporcionar una transcripción de lo hablado. Usualmente, los ASR que proporcionan resultados relativamente precisos como el Google o IBM Watson, necesitan acceder a los servidores que se encuentran en la nube. Esto pone una limitación en el sistema ya que el usuario siempre debe estar conectado a internet para obtener buenos resultados. Por lo tanto, en este sistema se brinda soporte para utilizar un motor de reconocimiento de voz local (*offline*) capacitado con los conjuntos de datos de comando de la aplicación. La transcripción del comando recibido se procesa utilizando un motor de procesamiento de lenguaje personalizado para extraer una estructura significativa del comando emitido en lenguaje natural. El comando se pasa a la aplicación correspondiente a través del sintetizador de voz.

En ID7 proponen un sistema de automatización inteligente multifuncional, donde los usuarios pueden usar comandos de voz para controlar sus electrodomésticos y dispositivos, independientemente de las características personales del hablante, como el acento. Los principales módulos en la implementación son *(i)* Reconocimiento de voz usando un micrófono y un módulo de reconocimiento de voz; *(ii)* Interpretación de los comandos de voz utilizando una placa del microcontrolador Arduino Uno. Los comandos de voz se clasifican en 5 grupos según su funcionalidad: acceso, ventilador, luz, utilidad y seguridad. Cada grupo incluye las funciones mínimas necesarias para un hogar inteligente. Una vez que se reconocen los comandos de voz, los controles correspondientes envían al microcontrolador los datos, a una velocidad de115.200 bps. Los comandos recibidos por el microcontrolador se comparan con las plantillas de comando cargadas previamente durante el proceso de entrenamiento. Si hay una coincidencia, entonces el microcontrolador realiza la función de inicio deseada y envía la orden al dispositivo correspondiente.

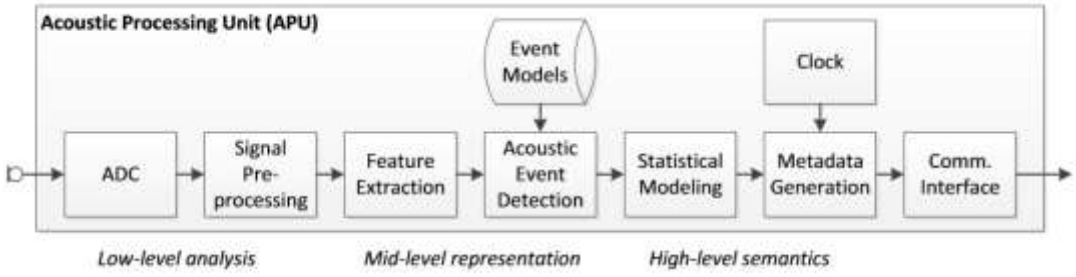
En ID8 desarrollan un prototipo basado en el procesador ARM Cortex-A8 que funciona bajo el sistema operativo LINUX, para el estudio del rendimiento y la medición de una configuración de prueba en tiempo real. Los resultados de las pruebas muestran que el sistema es capaz de reconocer los gestos de las manos y los comandos de voz para controlar dispositivos habilitados con tecnología ZigBee con comunicación inalámbrica. El módulo de reconocimiento de voz está basado en el algoritmo *Dynamic Time Warping – DTW*, el cual se ejecuta en dos fases: entrenamiento y prueba. A través de la etapa de pre-procesamiento, la señal analógica de entrada se convierte en señal digital. Los coeficientes de *cepstrum mel-frequency* se usan para extraer las características de la voz. En la fase de entrenamiento, las plantillas de muestreo se generan y se almacenan en base a los

datos de entrenamiento. En la fase de prueba, se mide la similitud entre las plantillas de muestreo y las plantillas de entrada para la comparación y la toma de decisiones.

En ID9 presentan un prototipo que incluye tecnología de detección acústica con redes de sensores inalámbricos a gran escala, útiles para entornos interiores y exteriores. El diseño incluyó pre-procesamiento de señales de audio, detección de eventos acústicos y modelado estadístico. La Unidad de Procesamiento Acústico (APU) es un potente dispositivo computacional integrado que hace parte de la gama de dispositivos IoT (Ver Figura 22). El reconocimiento de voz está integrado al banco de pruebas de Hobnet a través de APU's para interactuar con el entorno de forma natural. Las APU se instalan en entornos de oficina y mantienen un vocabulario de voz predefinido de tal forma que se pueda controlar la iluminación, el aire acondicionado y la calefacción. Los comandos de voz reconocidos se transcriben en metadatos y se transmiten a una plataforma de servicio central. Los identificadores únicos del sensor proporcionan la base de datos necesaria para cambiar activamente los parámetros ambientales en las áreas deseadas. La plataforma de servicio puede fusionar información de múltiples fuentes para crear un perfil de usuario contextual y personalizado de acuerdo a las necesidades del entorno.

En ID10 desarrollan un prototipo para controlar la iluminación y el aire acondicionado de una casa haciendo uso de servicios Web. El sistema está controlado por acciones naturales que son gestos y voz para un simple aparato de control y está ensamblado por tres módulos principales que son el reconocimiento de sonido, el reconocimiento de gestos y un proyector.

Figura 22. Arquitectura de la unidad de procesamiento acústico – ID9

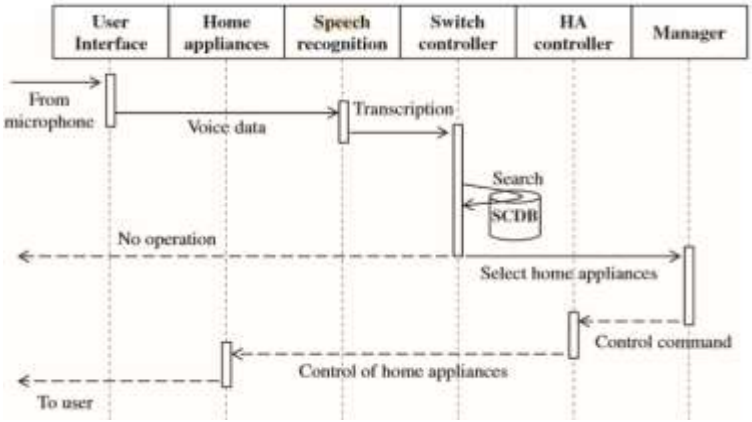


Fuente: Tomado de ID9

La

Figura 23 ilustra el flujo del control de voz. Los datos de voz obtenidos por el micrófono se envían al módulo de reconocimiento de voz y se transcriben a una cadena de texto.

Figura 23. Flujo del control de voz – ID10



Fuente: Tomado de ID10

La cadena de texto se envía para cambiar el estado del módulo del controlador. El controlador del interruptor muestra tres pasos de la siguiente manera: *(i)*, el controlador del interruptor recupera cada término en primeras palabras clave de una cadena de texto; *(ii)* si se encuentran las primeras palabras clave, el controlador del conmutador también recupera cada término de las segundas palabras clave de la cadena de texto. *(iii)* si se encuentran las segundas palabras clave, el controlador del interruptor envía un comando de control correspondiente a las palabras clave al módulo administrador.

En ID11 presentan una arquitectura de comunicación unificada integrada con IoT aplicada a la asistencia sanitaria. A través de esta arquitectura de comunicación bidireccional los cuidadores pueden monitorizar a los pacientes en tiempo real, brindar un diagnóstico y atención médica adecuada. Contiene un módulo de respuesta de voz interactiva que realiza una llamada de forma automática al paciente, si el paciente responde el teléfono, el sistema le preguntará si hay alguna ocurrencia inusual. El sistema funciona a través del reconocimiento automático de voz o Doble tono multi-frecuencia (DTMF). Con el ASR los pacientes pueden expresar palabras clave como: cancelar la alarma, ayuda, llamar a un familiar/médico. Con el DTMF (teclado del teléfono) los pacientes pueden presionar 1 para cancelar la notificación de alarma, presionar 2 para notificación de alarma, presionar 3 para audio conferencia con la familia, médicos y amigos.

En ID12 desarrollan un asistente robótico inteligente que opera con comandos de voz humana, administrado de forma remota mediante el uso de un dispositivo inteligente IoT basado en la plataforma Android. La señal de voz se convierte en texto, utilizando un servidor de nube en línea en tiempo real. Usando un dispositivo inteligente de IoT, este comando de texto se transmite al módulo Bluetooth integrado en el robot. El módulo Bluetooth recibe la señal y luego envía comandos al microcontrolador. El microcontrolador a su vez procesa los comandos y controla los cuatro motores utilizados para movimientos como giros y operaciones de inicio/detención, recoger un objeto en una ubicación y colocarlo en otra ubicación.

En ID13 presenta un caso de estudio orientado a respaldar los entornos de Patrimonio Cultural *(Smart Urban)*, que mejoran la experiencia del usuario basada en dispositivos inteligentes a través de la interacción con lo que se habla. Los usuarios envían su solicitud, utilizando el micrófono de su dispositivo personal. Si se entrega correctamente el audio, sin contextos ruidosos o idiomas no reconocidos, el sistema reconoce automáticamente la solicitud en italiano o inglés. El módulo de reconocimiento del habla se compone de cuatro pasos, a saber: *(i)* El reconocimiento de voz y la traducción al formato de texto se implementaron empleando la API de reconocimiento de voz de Google (aplicación Web) y la solución de Microsoft DotNet 4.0 (SDK AST y TTS, para la versión de escritorio); *(ii)* Análisis de texto y categorización de solicitudes. El análisis de texto típico (análisis sintáctico, desambiguación del sentido de la palabra, extracción de características léxicas) y las estrategias de categorización se aplican a las solicitudes de texto entrantes. El resultado se representa mediante listas de términos y categorías relevantes útiles para hacer coincidir con las respuestas adecuadas; *(iii)* Coincidencia de preguntas y respuestas. Las categorías y los términos que resumen la solicitud se comparan con una base de conocimientos (en inglés, *Knowledge Base - KB*). Los expertos en patrimonio cultural proporcionaron la ontología de dominio específico, vocabularios y respuestas para alimentar efectivamente el KB subyacente; *(iv)* Generación de mensajes de texto a lenguaje natural. Si una respuesta coincide con la solicitud enviada, se crea un evento ya sea reproducir un audio o video para ser entregado y notificado a los usuarios que lo solicitan.

4.3.4 Conclusiones de los sistemas para el reconocimiento del habla. En conclusión, los ASR pueden ser de cuatro (4) tipos (Besacier et al., 2014; Unnibhavi & Jangamshetti, 2016), a saber:

- Reconocimiento de palabras aisladas (en inglés, *Isolated word recognition*) usualmente requieren un solo enunciado a la vez. Es ideal para situaciones en las que el usuario debe dar respuestas o comandos de una sola palabra, pero no es natural para las entradas de varias palabras. Es simple y fácil de implementar porque tiene límite de palabras y las palabras tienden a ser claramente pronunciadas, lo que le da una mayor ventaja.

- Reconocimiento de palabras conectadas (en inglés, *Connected word recognition*) necesita una pausa mínima entre las emisiones de palabras para permitir el flujo de voz sin esfuerzo. Su funcionalidad es muy parecida a las palabras aisladas.
- Reconocimiento de voz continua (en inglés, *Continuous speech recognition*) permite a los usuarios hablar de forma más o menos natural, mientras que la computadora decide el contenido. Los reconocedores con habilidades de habla continua son más difíciles de generar ya que emplean técnicas únicas para decidir sobre los límites de emisión.
- Reconocimiento de voz espontánea (en inglés, *Spontaneous Speech recognition*) permite reconocer el habla espontánea como la que se produce en entrevistas, debates, diálogos. El sistema ASR con capacidad de hablar espontáneamente debe manejar una variedad de características del habla natural.

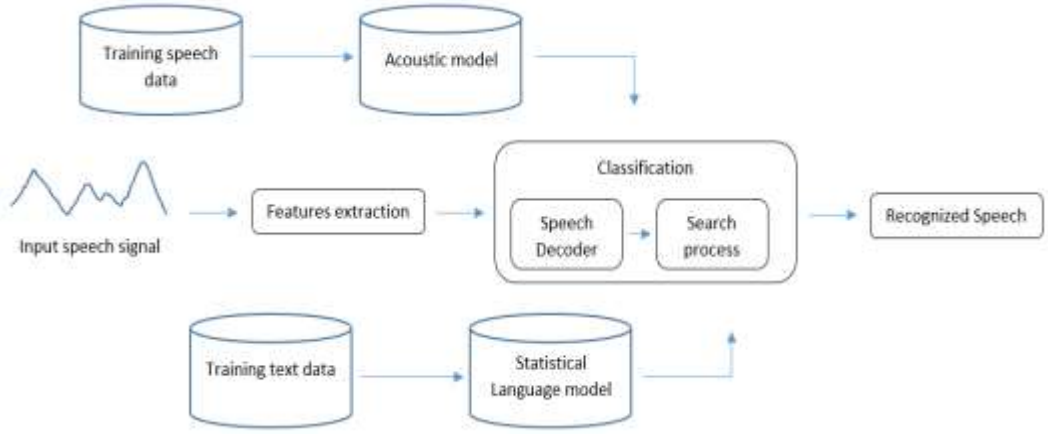
El proceso de un ASR tiene dos segmentos fundamentales, a saber: *(i)* extracción (en inglés, *extraction*) y *(ii)* clasificación (en inglés, *classification*) (Bouraoui, Jerad, Chattopadhyay, & Hadj-Alouane, 2017; Haridas, Marimuthu, & Sivakumar, 2018). En la Figura 24 se describe la arquitectura general de un sistema para el reconocimiento del habla.

El proceso de un ASR implica el pre-procesamiento, el modelado acústico y el modelado del lenguaje.

- El pre-procesamiento, consiste en la extracción de características (en inglés, Features extraction). Durante este paso, la señal de voz (es decir, un conjunto de ondas acústicas) se transforma en una secuencia de símbolos pre fonéticos sin significado lingüístico pero que contiene características con valores propios.
- El modelado acústico (en inglés, *Acoustic modeling*) que compara los símbolos con formas de onda fonéticas específicas, es decir, representa las señales de audio discriminando las clases de unidades de voz básicas y teniendo en cuenta la variabilidad del habla con respecto a los altavoces, canal y entorno. Para esto se requiere un entrenamiento del sistema (en inglés, training speech data) el cual se enfoca en mejorar el modelo en aspectos como: (i) Acústica del habla (requiere la grabación de varios hablantes); (ii) Idioma (requiere un corpus de datos o gramática de oraciones) y; (iii) Léxico de reconocimiento (lista de los *tokens* reconocibles con una o varias transcripciones fonéticas).
- El modelado de lenguaje (en inglés, *Language modeling*) es necesario para imponer las restricciones sobre las hipótesis de reconocimiento generadas en el ASR y para modelar la estructura, la sintaxis y la semántica del idioma de destino. Los modelos de lenguaje estadístico se basan en el hecho

empírico de que se puede obtener una buena estimación de la probabilidad de una unidad léxica observándola en datos de texto de gran tamaño.

Figura 24. Arquitectura general de un ASR



Fuente: Adaptado de (Besacier et al., 2014; Bouraoui et al., 2017; Haridas et al., 2018)

4.4 REQUISITOS FUNCIONALES Y NO FUNCIONALES DE UN *MIDDLEWARE* GENÉRICO PARA IOT

Tomando como base los requisitos consultados en las arquitecturas de referencia para IoT, *middleware* para IoT y los sistemas de reconocimiento del habla, en esta sección se especifican los requisitos funcionales y no funcionales que un *middleware* genérico para IoT debe considerar para garantizar el desarrollo de aplicaciones para IoT que faciliten la interacción hombre-máquina y máquina-máquina.

4.4.1 Requisitos funcionales. Son aquellas funcionalidades que el sistema debe proveer, sobre cómo debería reaccionar a entradas particulares y cómo debería comportarse en situaciones específicas (Sommerville, 2011). En la Tabla 17 se describen estos requisitos.

Tabla 17. Requisitos funcionales de un *middleware* para IoT

Grupo	Descripción	Componentes	Descripción
Descubrimiento de recursos	Un *middleware* debe descubrir recursos de forma automática.	Descubrimiento centralizado (en	Los recursos, el descubrimiento y la comunicación son manejados por un servidor dedicado.

Grupo	Descripción	Componentes	Descripción
(en inglés, *Resource discovery*)	Los tipos de recursos son: *hardware* heterogéneos, alimentación y memoria de dispositivos, convertidores de análogo/digital, módulo de comunicación, servicios propios de cada dispositivo.	inglés, *Centralized Discovery-CD*)	
		Descubrimiento distribuido (en inglés, *Distributed Discovery-DD*)	Maneja una red de comunicación donde cada recurso posee diferentes componentes que el programador percibe como un solo sistema.
		Descubrimiento de dispositivos (en inglés, *Device Discovery-DeD*)	Cada dispositivo se representa como un agente que se comunica con otros dispositivos para descubrir un servicio en la red.
		Descubrimiento de servicios (en inglés, *Service Discovery-SD*)	Requiere la ubicación en enormes redes de dispositivos conectados de un servicio que presenta las características requeridas dado un ámbito de interés, y la capacidad de asegurar una QoS requerida (Ccori, De Biase, Zuffo, & da Silva, 2016).
Gestión de recursos (en inglés, *Resource management*)	Un *middleware* debe monitorizar, resolver conflictos y asignar de forma justa los recursos debido a que se espera que todas las aplicaciones tengan una calidad del servicio (en inglés, *Quality of service - QoS*) aceptable (Delicato, Pires, & Batista, 2017).	Asignación de recursos (en inglés, *Resource allocation-RA*)	Distribuye de forma automática y dinámica los recursos.
		Monitor de recursos (en inglés, *Resource monitor-RM*)	El monitoreo de las variaciones relacionadas con el usuario, red, entorno físico y dispositivos debe ser constante debido a que el entorno de ejecución es extremadamente dinámico. El monitoreo sobre el uso actual de los recursos es necesario para mantener una asignación de recursos óptima o casi óptima.
		Composición de recursos (en inglés, *Resource composition-RC*)	Esta funcionalidad debe habilitarse una vez que el sistema no puede descubrir un servicio apropiado para manejar una solicitud específica.
Gestión de datos (en inglés, *Data management*)	Un *middleware* debe gestionar los datos o cualquier información de infraestructura de red que se generan a partir de las aplicaciones IoT.	Adquisición de datos (en inglés, *Data acquisition-DA*)	Proceso de muestreo de señales que miden las condiciones físicas del mundo real y convierten las muestras resultantes en valores numéricos digitales que pueden ser manipulados por una computadora. La forma como se adquieren los datos.
		Filtrado de datos (en inglés, *Data filtering-DF*)	Reducción de datos que generan ruido o son erróneos durante su obtención y proceso.
		Clasificación de datos (en inglés,	Separación de datos en varias categorías.

Grupo	Descripción	Componentes	Descripción
		Data classification-DC)	
		Almacenamiento de datos (en inglés, *Data storage-DS*)	Almacenamiento de datos de forma centralizada o distribuida.
		Agregación de datos (en inglés, *Data aggregation-DaA*)	Combinación de múltiples piezas de datos.
		Compresión de datos (en inglés, *Data compression-DaC*)	Codificación de datos utilizando menos bits que su representación original.
Gestión de eventos (en inglés, *Event management*)	Un *middleware* debe transformar eventos simples observados en eventos significativos. Debe proporcionar un análisis en tiempo real de los datos de alta velocidad para que las aplicaciones posteriores se basen en información precisa.	Larga escala (en inglés, *Large scale-LS*)	Garantiza escalabilidad (crecimiento de la red).
		Pequeña escala (en inglés, *Small scale-SS*)	No es fácilmente escalable.
Gestión de código (en inglés, *Code management*)	Un *middleware* debe facilitar la implementación de código para el desarrollo de aplicaciones IoT.	Asignación de código (en inglés, *Code allocation-CA*)	Selecciona el conjunto de dispositivos o nodos de sensor que se utilizarán para realizar una tarea a nivel de usuario o aplicación.
		Migración de código (en inglés, *Code migration-CM*)	Transfiere código entre nodo/dispositivo, potencialmente reprogramando nodos en la red. Con los servicios de migración de código, el código es portátil, lo que permite reubicar el cálculo de datos.

Fuente: El autor

4.4.2 Requisitos no funcionales. Describen cómo el *software* hará sus funcionalidades, basado en los atributos de calidad (Adams, 2015).

Al ser un *middleware* una herramienta *software* (Hadim & Mohamed, 2006; Razzaque et al., 2016; Wang et al., 2008), debe tener atributos de calidad propios de un producto de este tipo. La calidad del producto *software* se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos los que se encuentran representados en el modelo de calidad de la Organización Internacional de Normalización *(en inglés, International Organization for Standardization)* (ISO &

IEEE, 2010), el cual categoriza la calidad del producto. Este modelo está dividido en grupos con sus respectivos componentes, como se observa en la Tabla 18.

Tabla 18. Requisitos no funcionales de un *middleware* IoT - Grupo ISO

Grupo	Descripción
G1: Idoneidad Funcional (en inglés, *Functional suitability*)	Un middleware proporciona funciones que satisfacen necesidades explicitas e implícitas cuando se usan bajo condiciones específicas.
G2: Eficiencia de rendimiento (en inglés, *Performance efficiency*)	Un middleware representa el rendimiento relativo respecto a la cantidad de recursos utilizados en los requisitos.
G3: Compatibilidad (en inglés, *Compatibility*)	Un *middleware* puede intercambiar información con otros productos, sistemas o componentes y/o realizar las funciones requeridas, compartiendo el mismo entorno de hardware o software.
G4: Usabilidad (en inglés, *Usability*)	Un *middleware* puede ser utilizado por usuarios específicos para alcanzar los objetivos con eficacia, eficiencia y satisfacción en un contexto específico de uso.
G5: Confiabilidad (en inglés, *Reliability*)	Un *middleware* debe lograr la confiabilidad a nivel de sistema mientras permanezca operativo durante la duración de una misión, incluso en presencia de fallas. Cada componente o servicio de un *middleware* necesita ser confiable para lograr una fiabilidad total, que incluye comunicación, datos, tecnologías y dispositivos de todas las capas.
G6: Seguridad (en inglés, *Security*)	Un middleware debe considerar la seguridad en todos los bloques funcionales y no funcionales, incluida la aplicación a nivel de usuario.
G7: Mantenibilidad (en inglés, *Maintainability*)	Representa el grado de efectividad y eficiencia con que un *middleware* puede ser modificado para mejorarlo, corregirlo o adaptarlo a los cambios en el ambiente y en los requisitos.
G8: Portabilidad (en inglés, *Portability*)	Un *middleware* debe transferir recursos de un hardware, software o un entorno operativo a otro.

Fuente: Adaptado de (International Organization for Standardization - ISO, 2011)

En la Tabla 19 se explican los componentes de cada uno de los grupos que componen la norma de la ISO.

Tabla 19. Requisitos no funcionales de un *middleware* IoT - Componente ISO

Grupo	Componente	Descripción
G1	Íntegra funcionalidad (en inglés, *Functional completeness*)	Un *middleware* debe tener un conjunto de funciones que cubre todas las tareas específicas y los objetivos del usuario.

Grupo	Componente	Descripción
	Correcta funcionalidad (en inglés, *Functional correctness*)	Un *middleware* debe proporcionar los resultados correctos con el grado de precisión necesario.
	Adecuada Funcionalidad (en inglés, *Functional appropriateness*)	Un middleware debe tener funciones que faciliten el cumplimiento de tareas y objetivos específicos.
G2	Comportamiento del tiempo (en inglés, *Time behaviour*)	Un middleware debe cumplir con los tiempos de procesamiento y las tasas de rendimiento planteados en los requisitos.
	Utilización de recursos (en inglés, *Resource utilization*)	Un middleware debe cumplir con las cantidades y los tipos de recursos planteados en los requisitos.
	Capacidad (en inglés, *Capacity*)	Un middleware debe cumplir con los productos o parámetros del sistema planteados en los requisitos.
G3	Coexistencia (en inglés, *Co-existence*)	Un middleware debe llevar a cabo sus funciones requeridas de forma eficiente mientras comparte un entorno y recursos comunes con otros productos, sin causar ningún impacto perjudicial entre ellos.
	Interoperabilidad (en inglés, *Interoperability*)	Un middleware debe intercambiar información entre sus componentes u otros sistemas, y utilizar la información que se ha intercambiado a través de tecnologías heterogéneas.
G4	Reconocimiento de la idoneidad (en inglés, *Appropriateness recognizability*)	Los desarrolladores/usuarios pueden reconocer si el middleware es apropiado para sus necesidades.
	Aprendizaje (en inglés, *Learnability*)	Un middleware puede ser utilizado por desarrolladores/usuarios para aprender a usar el sistema con eficacia, eficiencia, libertad de riesgo y satisfacción en un contexto específico de uso.
	Protección de error de usuario (en inglés, *User error protection*)	Un middleware debe proteger a los desarrolladores/usuarios contra errores.
	Operabilidad/Fácil desarrollo (en inglés, *Operability*)	Un middleware debe tener atributos que lo hacen fácil de operar y controlar. Se deben evitar los procedimientos complicados de instalación y configuración.
	Estética de interfaz de usuario (en inglés, *User interface aesthetics*)	Un middleware debe tener una interfaz que sea agradable y facilite la satisfactoria interacción para el usuario.
	Accesibilidad (en inglés, *Accessibility*)	Un middleware puede ser utilizado por personas con características y capacidades diferentes, para lograr un objetivo específico en un contexto específico de uso.
G5	Madurez (en inglés, *Maturity*)	Un middleware debe satisfacer las necesidades de fiabilidad en condiciones normales de funcionamiento.
	Disponibilidad (en inglés, *Availability*)	Un *middleware* que soporte las aplicaciones de un IoT, especialmente los de misión crítica, debe estar disponible, o aparecer disponible, en todo momento. Incluso si hay un fallo en alguna parte del sistema, su tiempo de recuperación y frecuencia de fallo debe ser lo suficientemente pequeño para lograr la disponibilidad deseada.

Grupo	Componente	Descripción
	Tolerancia a fallos (en inglés, *Fault tolerance*)	Un *middleware* debe funcionar según lo previsto a pesar de la presencia de fallos a nivel de hardware o software.
	Recuperabilidad (en inglés, *Recoverability*)	En caso de interrupción o fallo, el *middleware* debe recuperar los datos directamente afectados y restablecer el estado deseado del sistema.
G6	Confidencialidad (en inglés, Confidentiality)	Un *middleware* debe garantizar que un mensaje dado no debe ser entendido por nadie más que los destinatarios deseados.
	Integridad (en inglés, *Integrity*)	Un *middleware* debe garantizar que los datos producidos y consumidos en el sistema IoT no se alteren maliciosamente.
	No repudio (en inglés, *Non-repudiation*)	Un *middleware* puede demostrar que las acciones o eventos han tenido lugar para que estos no puedan ser repudiados más tarde.
	Rendición de cuentas (en inglés, *Accountability*)	Un *middleware* debe garantizar que las acciones de una entidad pueden ser rastreadas únicamente a esa entidad. Permite la auditoría de los sistemas IoT para eventos significativos y analiza los registros para asociar los comportamientos de las entidades a dichos eventos.
	Autenticidad (en inglés, *Authenticity*)	Un *middleware* debe asegurar que los datos son enviados por el remitente reclamado.
G7	Modularidad (en inglés, *Modularity*)	Un *middleware* debe estar compuesto de componentes discretos, de modo que un cambio en un componente tiene un impacto mínimo en otros componentes.
	Reutilización (en inglés, *Reusability*)	Un *middleware* debe tener activos que pueden ser utilizados en más de un sistema/aplicación, o en la construcción de otros activos.
	Análisis (en inglés, *Analysability*)	Un *middleware* debe evaluar el impacto de un cambio previsto en una o varias de sus partes, o diagnosticar un producto por deficiencias o causas de fallas o identificar partes que se van a modificar.
	Modificabilidad (en inglés, *Modifiability*)	Un *middleware* puede modificarse eficaz y eficientemente sin introducir defectos o degradar la calidad del producto existente.
	Testabilidad (en inglés, *Testability*)	Un middleware puede establecer criterios de prueba y realizarlas para determinar si se han cumplido dichos criterios.
G8	Instalación (en inglés, *Installability*)	Un middleware debe instalarse y/o desinstalarse correctamente en un entorno especificado.
	Reemplazabilidad (en inglés, *Replaceability*)	Un middleware puede reemplazar otro producto de software especificado para el mismo propósito en el mismo entorno.
	Adaptabilidad (en inglés, *Adaptability*)	Un middleware debe adaptarse eficaz y eficientemente a hardware, software u otros entornos operativos o de uso diferentes o en evolución. En el IoT, es probable que la red y su entorno cambien frecuentemente. Además, las demandas a nivel de aplicación o el contexto también pueden cambiar con frecuencia. Para asegurar la satisfacción del usuario y la efectividad del IoT, un Middleware necesita adaptarse o adaptarse dinámicamente para adaptarse a todas esas variaciones.

Fuente: Adaptado de (International Organization for Standardization - ISO, 2011)

Existen otros requisitos no funcionales que no se describen de forma explícita en la ISO. Fueron mencionados en la literatura porque se deben tener en cuenta al momento de diseñar un *middleware* que facilita el desarrollo de aplicaciones para IoT, ver Tabla 20.

Tabla 20. Requisitos no funcionales de un *middleware* para IoT - Literatura

Requisito	Descripción
Procesamiento del flujo (en inglés, *Stream Processing*)	Un middleware proporciona una abstracción para la interacción en tiempo real entre la aplicación y los dispositivos físicos.
Tiempo real (en inglés, *Real time*)	Un middleware debe apoyar los servicios en tiempo real, si la aplicación así lo requiere. La información retardada o los servicios en tales usos pueden hacer el sistema inútil e incluso peligroso.
Distribución (en inglés, *Performance efficiency*)	Un middleware necesita soportar funciones que se distribuyen a través de la infraestructura física del IoT. Sus aplicaciones/dispositivos/usuarios intercambian información y colaboran entre sí. Por lo general, estos recursos están geográficamente distribuidos.
Autonomía (en inglés, *Autonomous*)	Un middleware debe facilitar que los participantes activos en los procesos de IoT como los dispositivos/tecnologías/aplicaciones, interactúen y se comuniquen entre sí, sin intervención humana directa.
Visualización (en inglés, *Visualization*)	Un middleware debe facilitar la visualización de datos recogidos en cualquier momento y en cualquier lugar sin tener que escribir una aplicación de terceros compleja.
Monitorización (en inglés, *Monitoring*)	Un middleware debe ser capaz de comprobar el estado de los dispositivos acorde a lo requerido.
Autorización (en inglés, *Authorization*)	Un middleware debe garantizar que sólo las entidades autorizadas puedan realizar determinadas operaciones.
Privacidad (en inglés, *Privacy*)	Un middleware debe preservar la privacidad del propietario (usuarios finales, sistema IoT, dispositivos IoT) y la manera cómo quieren compartir sus datos.
Escalabilidad (en inglés, *Scalability*)	Un middleware puede acomodar el crecimiento en la red, aplicaciones o servicios de tal forma que se oculte la complejidad de la lógica o la implementación del hardware o servicio subyacente.
Persistencia (en inglés, *Persistency*)	Un middleware debe proporcionar y manejar el almacenamiento de diferentes tipos y volúmenes de datos.
Consciencia del contexto (en inglés, *Context-aware*)	Un middleware debe ser consciente del contexto de los usuarios, dispositivos y el entorno, para ofrecer servicios eficaces y esenciales a los usuarios. De esta forma, se pueda construir sistemas adaptativos y también establecer el valor de los datos detectados.

Fuente: El autor

4.5 REQUISITOS FUNCIONALES Y NO FUNCIONALES DEL *MIDDLEWARE* SWITCH

En esta sección se especifican los requisitos funcionales y no funcionales considerados en el diseño propuesto del *middleware* para IoT que incluye interfaces de voz, denominado *middleware* para IoT que incluye interfaces basadas en voz, denominado *Smart middleWare for Iot with speeCH recognition – SWITCH*.
Tomando como base los requisitos presentados en la sección 4.4, se concretó un listado de requisitos funcionales como se observa en la Tabla 21. Esta lista incluye requisitos para el reconocimiento del habla.

Tabla 21. Requisitos funcionales de SWITCH

Req. ID	Requisito
Req-[1]	El *middleware* debe permitir a los usuarios finales configurar aplicaciones IoT con reconocimiento del habla
Req-[2]	El *middleware* debe permitir el estado “escucha” cada vez que se da la instrucción
Req-[3]	El *middleware* debe proveer mecanismos para transformar la señal de voz en símbolos fonéticos
Req-[4]	El *middleware* debe comparar los símbolos fonéticos con un léxico de reconocimiento
Req-[5]	El *middleware* debe comparar los símbolos fonéticos con la base de datos de palabras
Req-[6]	El *middleware* debe transformar el audio en una comando que sea entendido por las aplicaciones IoT
Req-[7]	El *middleware* debe categorizar las entidades una vez obtiene la transcripción del habla al texto
Req-[8]	El *middleware* debe transformar eventos simples observados en eventos significativos
Req-[9]	El *middleware* podría permitir el almacenamiento de algoritmos para la extracción de características de la señal de voz
Req-[10]	El *middleware* debe permitir el almacenamiento del comando de habla
Req-[11]	El *middleware* debe permitir el almacenamiento de una base de datos de palabras
Req-[12]	El *middleware* debe permitir el almacenamiento de un léxico de reconocimiento
Req-[13]	El *middleware* debe proveer mecanismos para configurar una plataforma de ASR
Req-[14]	El *middleware* debe mostrar al usuario el comando de habla en formato texto
Req-[15]	El *middleware* debe proveer mecanismos para analizar la conectividad de la red
Req-[16]	El *middleware* debe ofrecer funciones de control de conectividad de red
Req-[17]	El *middleware* debe permitir el almacenamiento del comando de habla en formato texto
Req-[18]	El *middleware* debe proveer mecanismos para configurar los requerimientos de una aplicación para IoT
Req-[19]	El *middleware* debe gestionar la movilidad de los datos en el sistema
Req-[20]	El *middleware* debe mostrar al usuario las entidades reconocidas a partir de la categorización del texto
Req-[21]	El *middleware* debe permitir crear aplicaciones para IoT
Req-[22]	El *middleware* debe descubrir recursos de forma automática
Req-[23]	El *middleware* debe facilitar la visualización de datos recogidos en cualquier momento
Req-[24]	El *middleware* debe monitorizar, resolver conflictos y asignar de forma justa los servicios debido a que se espera que todas las aplicaciones tengan una calidad de servicio aceptable.
Req-[25]	El *middleware* debe permitir recolectar información de sensores

Req-[26]	El *middleware* debe facilitar la implementación de código para el desarrollo de aplicaciones IoT
Req-[27]	El *middleware* debe permitirle al usuario categorizar (entrenamiento) las entidades del texto

Fuente: El autor

Tomando como base los requisitos no funcionales presentados en la Tabla 18, se concretó un listado detallado de requisitos no funcionales que atiende SWITCH como se observa en la Tabla 22.

Tabla 22. Requisitos no funcionales de SWITCH

Requisito	Descripción
Escalabilidad (en inglés, *Scalability*)	El *middleware* puede acomodar el crecimiento de la red, aplicaciones o servicios de tal forma que se oculte la complejidad de la lógica o la implementación del *hardware* o servicio subyacente y a nivel de arquitectura no se impactan las capas que están relacionadas.
Persistencia (en inglés, *Persistency*)	El *middleware* debe proporcionar y manejar el almacenamiento de diferentes tipos de datos (audio, texto, sensores).
Consciencia del contexto (en inglés, *Context-aware*)	El *middleware* debe ser consciente del contexto de los usuarios, dispositivos y el entorno, para ofrecer servicios eficaces y esenciales a los usuarios. De esta forma, se pueda construir sistemas adaptativos.
Procesamiento del flujo (en inglés, *Stream Processing*)	El *middleware* proporciona una abstracción para la interacción en tiempo real entre la aplicación y los dispositivos físicos.
Distribución (en inglés, *Performance efficiency*)	El *middleware* necesita soportar funciones que se distribuyen a través de la infraestructura física del IoT. Sus aplicaciones/dispositivos/usuarios intercambian información y colaboran entre sí. Por lo general, estos recursos están geográficamente distribuidos.
Modularidad (en inglés, *Modularity*)	El *middleware* debe estar compuesto de componentes discretos, de modo que un cambio en un componente tiene un impacto mínimo en otros componentes.
Reutilización (en inglés, *Reusability*)	El *middleware* debe tener activos que pueden ser utilizados en más de un sistema/aplicación, o en la construcción de otros activos.
Análisis (en inglés, *Analysability*)	El *middleware* debe evaluar el impacto de un cambio previsto en una o varias de sus partes, o diagnosticar un producto por deficiencias o causas de fallas o identificar partes que se van a modificar.
Modificabilidad (en inglés, *Modifiability*)	El *middleware* puede modificarse eficaz y eficientemente sin introducir defectos o degradar la calidad del producto existente.
Testabilidad (en inglés, *Testability*)	El *middleware* puede establecer criterios de prueba y realizarlas para determinar si se han cumplido dichos criterios.
Visualización (en inglés, *Visualization*)	El *middleware* debe facilitar la visualización de datos recogidos en cualquier momento y en cualquier lugar sin tener que escribir una aplicación de terceros compleja.
Monitorización (en inglés, *Monitoring*)	El *middleware* debe ser capaz de comprobar el estado de los dispositivos acorde a lo requerido.

Requisito	Descripción
Autorización (en inglés, *Authorization*)	El *middleware* debe garantizar que sólo las entidades autorizadas puedan realizar determinadas operaciones.
Protección de error de usuario (en inglés, *User error protection*)	El *middleware* debe proteger a los desarrolladores/usuarios contra errores.
Estética de interfaz de usuario (en inglés, *User interface aesthetics*)	El *middleware* debe tener una interfaz que sea agradable y facilite la satisfactoria interacción para el usuario.
Fácil desarrollo (en inglés, *Easy-of-deployment*)	El *middleware* debe tener atributos que lo hacen fácil de operar y controlar. El despliegue de un middleware IoT no debe requerir conocimientos o soporte técnico. Se deben evitar los procedimientos complicados de instalación y configuración.
Autorización (en inglés, *Authorization*)	El *middleware* debe garantizar que sólo las entidades autorizadas puedan realizar determinadas operaciones.
Autonomía (en inglés, *Autonomous*)	El *middleware* debe facilitar que los participantes activos en los procesos de IoT como los dispositivos/tecnologías/aplicaciones, interactúen y se comuniquen entre sí, sin intervención humana directa.
Utilización de recursos (en inglés, *Resource utilization*)	Un middleware debe cumplir con las cantidades y los tipos de recursos planteados en los requisitos.

Fuente: El autor

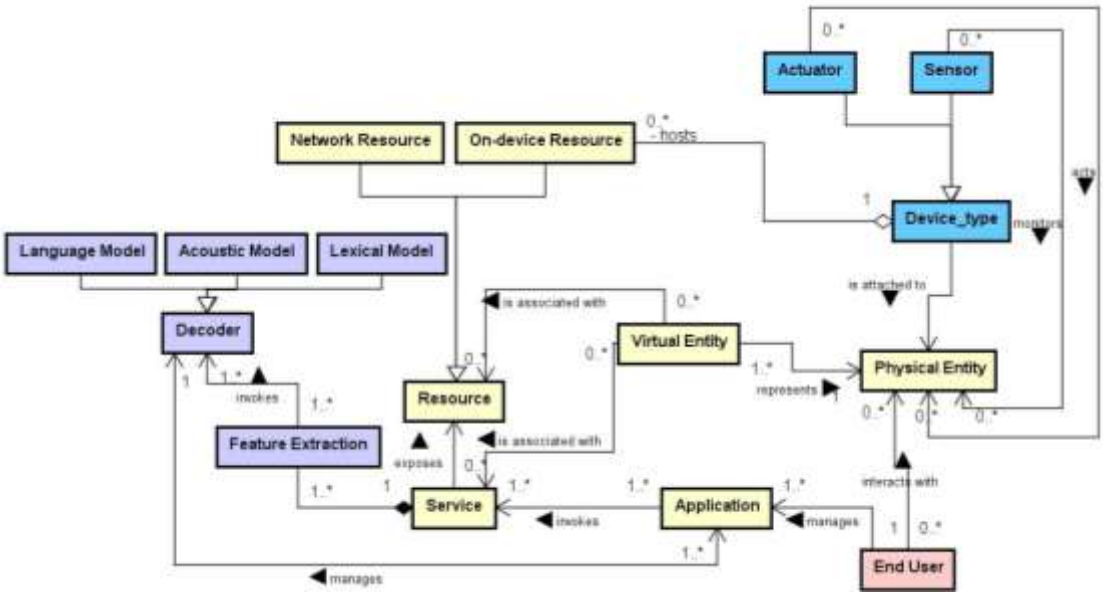
5 MODELADO DE LOS REQUISITOS DE SWITCH

En este capítulo se presenta el modelado de los requisitos del *middleware* SWITCH propuesto. Estos requisitos fueron obtenidos en la fase de análisis del proyecto y presentados en la sección 4.5 página 93. En la sección 5.1 se presenta el modelado del dominio del *middleware* SWITCH. En la sección 5.2 se presenta la arquitectura del *middleware* SWITCH. En la sección 5.3 se presenta el modelado del *middleware* que incluye los componentes del *software*. En la sección 5.4 se presenta el modelado del *middleware* que incluye los componentes del *hardware*. Finalmente, en la sección 5.3 se presenta la síntesis del diseño del *middleware* SWITCH.

5.1 MODELADO DEL DOMINIO DE SWITCH

En esta sección se propone un modelo del dominio IoT, cuyos conceptos claves se identificaron tomando como base los requisitos descritos en la Tabla 21, los cuales fueron obtenidos a partir del análisis realizado a las arquitecturas de los *middleware* para IoT, sistemas para el reconocimiento del habla y las arquitecturas de referencia para IoT consultadas en el capítulo 4. La Figura 25 presenta la vista conceptual del *middleware* SWITCH propuesto.

Figura 25. Representación UML del modelo de dominio



Fuente: Adaptado de (IoT-A Project, 2016)

Los conceptos y las relaciones se representan usando un lenguaje unificado de modelamiento *(en inglés, Unified Modeling Language - UML)*. El diagrama UML facilita la visualización de las relaciones que existen entre los elementos de *hardware* (color azul), *software* (color amarillo), usuarios (color rosa) y reconocimiento del habla (color morado).

Un modelo de dominio contiene todas las entidades y sus relaciones que participan o hacen parte del contexto de un área particular. El modelo de dominio representado a través de la Figura 25, se enfoca en IoT con un sistema para el reconocimiento del habla. Para definir los conceptos claves se consideraron los siguientes aspectos, a saber: *(i)* los conceptos son independientes de tecnologías específicas y de los casos de uso; *(ii)* los conceptos están relacionados con los requisitos funcionales presentados en la Tabla 21; *(iii)* los conceptos se mantienen a lo largo del tiempo. Además, el modelo de dominio define las relaciones entre conceptos y ayudan a facilitar el intercambio de datos entre diferentes dominios de aplicación.

5.1.1 Conceptos del modelado del dominio. La Tabla 23 detalla los conceptos utilizados en el modelo de dominio y su respectiva descripción.

Tabla 23. Conceptos del modelo de dominio

Conceptos	Descripción
Usuario final (en inglés, *End User*)	El usuario define e interactúa con la aplicación IoT. A través del usuario se obtiene la señal de voz para ser reconocida como texto.
Aplicación (en inglés, *Application*)	Se compone de un conjunto de requisitos definidos por el *End_User.* A través de la aplicación, el usuario puede invocar o suscribirse a uno o muchos servicios; y un servicio puede ser invocado o suscrito por cero o muchas aplicaciones. Mediante la aplicación también se puede administrar el reconocedor del habla.
Dispositivo (en inglés, *Device*)	Artefacto técnico destinado a proporcionar una interfaz entre el mundo digital y el físico. Tres tipos básicos de dispositivos son sensores, etiquetas y actuadores.
Sensor	Su objetivo es monitorizar eventos o cambios en su entorno, y luego proporcionar una salida correspondiente.
Actuador (en inglés, *Actuator*)	Pueden modificar el estado físico de una entidad física o activar/desactivar funcionalidades de otras más complejas.
Entidad física (en inglés, *Physical Entity*)	Parte identificable del entorno físico que le ayuda al usuario para atender sus requisitos. Las entidades físicas pueden ser casi cualquier objeto o entorno como cosas, humanos, animales o dispositivos. Un usuario interactúa con cero o muchas entidades físicas, y una entidad física puede ser interactuada por cero o muchos usuarios.
Entidad virtual (en inglés, *Virtual Entity*)	Son representaciones sincronizadas de un conjunto de aspectos (o propiedades) de la entidad física. Esto significa que los parámetros digitales relevantes que representan las características de la entidad física se actualizan con cualquier cambio de la primera.

Conceptos	Descripción
Extracción de características (en inglés, *Feature Extraction*)	Es el módulo encargado de extraer las características de la señal de voz (es decir, un conjunto de ondas acústicas) para transformarlas en una secuencia de símbolos fonéticos sin significado lingüístico pero que contiene características con valores propios.
Decodificador (en inglés, *Decoder*)	Porción de software del motor de reconocimiento de voz encargado de decodificar la señal de voz en texto. Para reconocer el habla necesita de un modelo acústico y un modelo de lenguaje.
Modelo acústico (en inglés, *Acoustic Model*)	Es el módulo que compara los símbolos con formas de onda fonéticas específicas, es decir, representa las señales de audio discriminando las clases de unidades de voz básicas y teniendo en cuenta la variabilidad del habla con respecto a los altavoces, canal y entorno. Esto lo realiza a través de una base de datos con palabras reconocibles que permite la comparación de transcripciones fonéticas.
Modelo léxico (en inglés, *Lexical Model*)	Es el encargado de generar el vocabulario de reconocimiento y asignar cada *token* ortográfico (palabras o sub-palabras) del léxico con la representación hablada correspondiente (transcripción fonética).
Modelo del lenguaje (en inglés, *Language Model*)	Es el módulo encargado imponer las restricciones sobre las hipótesis de reconocimiento generadas durante el proceso de entrenamiento del ASR y modelar la estructura, la sintaxis y la semántica del idioma de destino, de tal forma que a partir de las palabras se forman la oración.
Recurso de red (en inglés, *Network Resource*)	Gestiona todos los recursos de red que se ejecutan en la "nube" o de forma local, no dependen de un hardware especial que permita una conexión directa con el mundo físico.
Recursos del dispositivo (en inglés, *On-Device Resource*)	Se describe como un puente entre el mundo físico y digital. Cumple la función de tomar y almacenar la señal de voz.
Recurso (en inglés, *Resource*)	Los recursos son componentes de software que proporcionan cierta funcionalidad. Cuando se asocian con una entidad física, proporcionan alguna información o permiten cambiar algunos aspectos en el mundo físico o digital perteneciente a una o más entidades físicas.
Servicio (en inglés, *Service*)	Proporciona el enlace entre los aspectos de un sistema IoT y otras partes no específicas de un sistema de información.

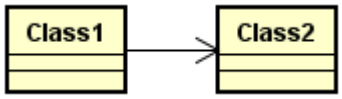
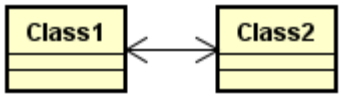
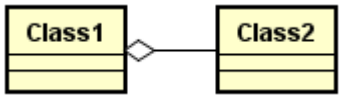
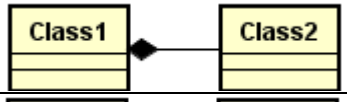
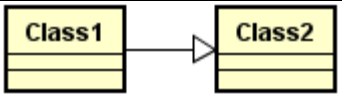
Fuente: El autor

5.1.2 Relaciones del modelado del dominio. Para entender las relaciones establecidas en el modelo de dominio, en la Tabla 24 se describen los tipos de asociación entre conceptos que existen en el modelo propuesto.

En el dominio IoT, un usuario administra uno o muchas aplicaciones; y una aplicación puede ser administrada al tiempo por un usuario. A su vez, un usuario interactúa con cero o muchas entidades físicas, y una entidad física puede ser interactuada por cero o muchos usuarios.

La aplicación invoca uno o muchos servicios. Un servicio expone cero o muchos recursos, y un recurso puede ser expuesto por cero o muchos servicios. Un recurso actúa sobre cero o muchas entidades virtuales, y una entidad virtual puede ser asociada por cero o muchos recursos.

Tabla 24. Tipos de asociación del modelo de dominio

Tipo	Gráfica	Concepto
Asociación unidireccional		Relación estructural que describe una conexión entre objetos que se puede recorrer en ambos sentidos.
Asociación bidireccional		Relación estructural que describe una conexión entre objetos que restringe la navegación en un único sentido.
Agregación		Representa la relación como parte-de, que presenta a una entidad como un agregado de partes.
Composición		Implica que los componentes de un objeto sólo pueden pertenecer a un solo objeto agregado.
Generalización		Relación entre una superclase y sus subclases (herencia).

Fuente: El autor

Una entidad virtual representa una entidad en el mundo físico, y una entidad física puede ser representada por una o muchas entidades virtuales. Sobre las entidades físicas intervienen los dispositivos IoT, que pueden ser actuadores y sensores quienes heredan características de una clase padre llamada dispositivo. Una entidad física puede ser monitorizada por cero o muchos sensores; y un sensor puede actuar sobre cero o muchas entidades físicas. De igual forma, una entidad física puede ser actuada por cero o muchos actuadores, y un actuador puede actuar sobre cero o muchas entidades físicas.

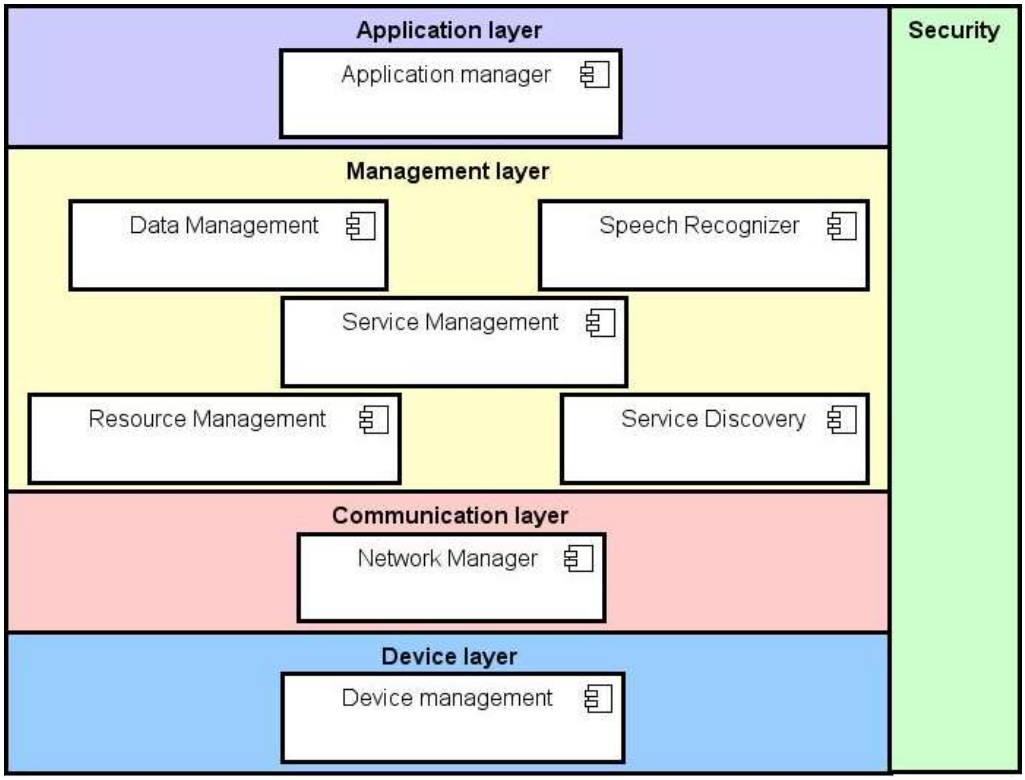
Los recursos sobre dispositivo al igual que los recursos sobre la red son una clasificación de recursos en el dominio IoT. Un dispositivo puede albergar cero o muchos recursos en el dispositivo, y un recurso hace parte del dispositivo. De la misma manera, los cambios que afectan a la entidad virtual podrían manifestarse en la entidad física. Por otra parte, las entidades virtuales son artefactos digitales que se pueden clasificar como activos o pasivos. Los artefactos digitales activos (ADA) están ejecutando aplicaciones de *software*, agentes o servicios que pueden acceder a otros servicios o recursos. Los artefactos digitales pasivos (PDA) son elementos pasivos del *software*, como las bases de datos que pueden ser representaciones digitales de la entidad física.

La extracción de características es un tipo de servicio que invoca al decodificador. Esta operación se realiza tantas veces como se invoque el servicio. El decodificador tiene tres partes fundamentales que pueden ser el modelo del lenguaje, acústico y léxico. Estos permiten realizar el proceso de reconocimiento del habla.

5.2 ARQUITECTURA DE SWITCH

El modelo de la arquitectura es una vista en alto nivel donde se muestra la división de las principales capas que conforman el diseño de SWITCH. Tomando como base lo consultado en la literatura sobre la arquitectura que tiene un *middleware* para IoT (Ver Tabla 7), el análisis de requisitos presentado en el capítulo 4 y el modelado del dominio, en la Figura 26 se ilustra la arquitectura del *middleware* SWITCH propuesto.

Figura 26. Arquitectura de SWITCH



Fuente: El autor

La arquitectura propuesta tiene un estilo en capas el cual se centra en agrupar la funcionalidad relacionada dentro del *middleware* de forma horizontal, una capa encima de la otra. Cada capa proporciona un conjunto de servicios a la capa anterior y utiliza los servicios de la capa siguiente. Entre dos capas adyacentes se proporciona una interfaz claramente definida (Richards, 2015). El estilo arquitectural en capas atiende algunos requisitos no funcionales, a saber: *(i)* Escalabilidad porque se puede incrementar el número de los componentes *(devices)* sin impactar de forma negativa en la capa superior *(Application)*, lo cual les permite tener una

relación independiente; *(ii)* Seguridad, ya que al tener capas aisladas, cuando una de ellas tiene fallas a nivel de seguridad, no implica que las otras deban comprometerse del mismo modo; *(iii)* Mantenibilidad, puesto que el hecho de realizar correcciones de errores de software o fallos *(bug)*, o simples tareas de mantenimiento en una capa, no implica que se deba re-implantar las capas superiores. Apenas se re-implantan las capas que fueron modificadas.

Aunque el estilo arquitectural en capas no especifica el número y los tipos de capas que debe tener una arquitectura *middleware*, en la mayoría de las arquitecturas estudiadas en el capítulo 4 se observó cuatro capas estándar: aplicación, gestión (en algunos casos nombrada como *middleware)*, dispositivos y seguridad (capa transversal). Para especificar la función entre la conexión con los dispositivos y la topología de la red, se agregó una capa más llamada comunicación. Esta capa atiende un requisito no funcional enfocado a la tolerancia a fallos, ya que es responsable por el buen funcionamiento de la red y dispositivos subyacentes. Además, realiza procesos de seguridad en la comunicación de la red.

En la Tabla 25 se describe cada una de las capas que conforman la arquitectura del *middleware* SWITCH.

Tabla 25. Capas de la arquitectura de SWITCH

Capa	Descripción
1. Aplicación (en inglés, *Application*)	Atiende las solicitudes de servicio de alto nivel dentro del mismo proceso de la arquitectura del *middleware* como las que provienen de los usuarios finales o incluso de aplicaciones externas (servicios alojados).
2. Gestión (en inglés, *Management*)	Esta capa se utiliza para componer y orquestar los servicios con diferentes niveles de abstracción. Dentro de las funcionalidades se encuentran la gestión de los servicios, descubrimiento de servicios, entrada, almacenamiento y procesamiento de los datos, reconocimiento del habla y la gestión de los recursos del *middleware.*
6. Comunicación (en inglés, *Communication*)	Esta capa provee una interfaz común para comunicar los dispositivos y la variedad de funcionalidades que tienen estas tecnologías heterogéneas, con los servicios provenientes de la capa de gestión.
4. Dispositivo (en inglés, *Device*)	Esta capa expone los dispositivos para hacerlos accesibles y visibles a otras partes del *middleware*.
7. Seguridad (en inglés, *Security*)	Es el responsable de garantizar la seguridad y la privacidad de los componentes que hacen parte de la arquitectura del *middleware* SWITCH. Tiene en cuenta aspectos como (i) Permiso: ¿qué se puede hacer?; (ii) Prohibición: ¿qué no se debe hacer?; (iii) Obligaciones: ¿qué se debe hacer?

Fuente: El autor

5.3 MODELADO DE LOS COMPONENTES DEL *SOFTWARE*

La arquitectura del *middleware* SWITCH tiene un enfoque de diseño orientado a servicios, es decir, proporciona la funcionalidad del sistema como un conjunto de servicios. A continuación se describe la vista funcional, la vista de servicios, la vista de procesos y la interfaz gráfica de usuario del *middleware* SWITCH, las cuales permiten describir el modelado de los componentes principales del *software*.

5.3.1 Vista funcional de SWITCH. Esta vista describe cómo está conformada la parte estructural de la arquitectura del *middleware* SWITCH, incluida las funciones y responsabilidades de los componentes, interfaces y sus relaciones.

La vista funcional de SWITCH se deriva de la arquitectura (Ver Figura 26) y los requisitos funcionales (Ver Tabla 21). El diseño atiende requisitos no funcionales como la facilidad de implementación, costo reducido, reutilización de componentes y mitigación de la complejidad técnica. La Figura 27 muestra la vista funcional del *middleware* SWITCH representada a través de un diagrama de componentes. La vista funcional está estructurada en cuatro capas, siguiendo como base la arquitectura propuesta para el *middleware SWITCH*.

La capa de aplicación contiene el componente para la gestión de aplicaciones (en inglés, *Application Manager*). Los componentes de la capa de gestión son el gestor de datos (en inglés, *Data Management*), gestor de recursos (en inglés, *Resource Management*), gestor de servicios (en inglés, *Service management*), descubrimiento de servicios (en inglés, *Service Discovery*) y el reconocedor del habla (en inglés, *Speech recognizer*). La capa de comunicación contiene el componente gestor de red (en inglés, *Network manager*). Finalmente, la capa de dispositivos tiene el componente para la gestión de dispositivos (en inglés, *Device Management*).

5.3.1.1 Capa de aplicación. SWITCH es un *middleware* que facilita el desarrollo de aplicaciones IoT con interfaces basadas en voz. Al ser la voz una capacidad humana natural, estas interfaces facilitan la interacción no intrusiva entre humano-máquina y máquina-máquina. El usuario es quien gestiona la aplicación para emitir comandos de ejecución a través del habla.

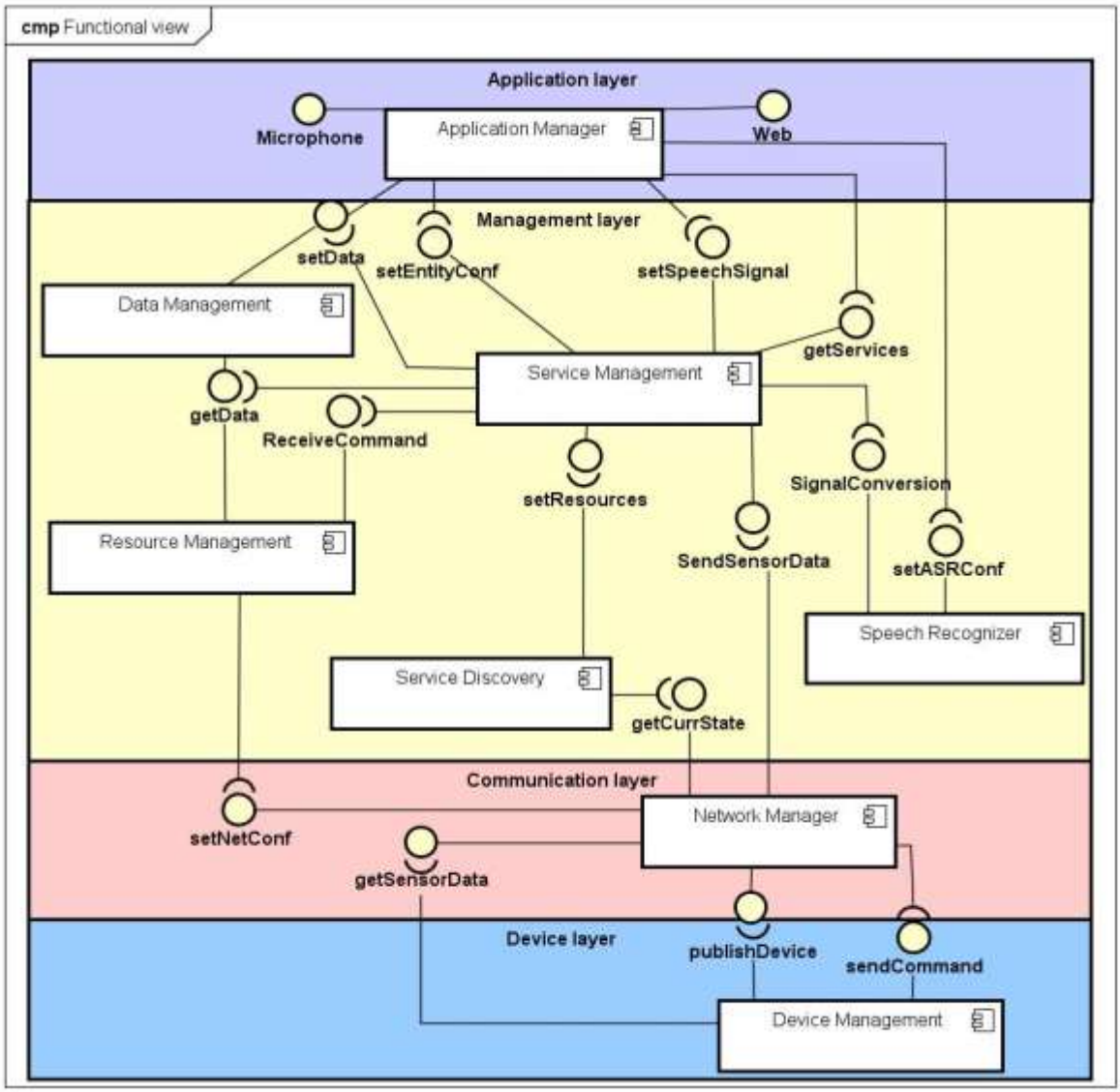
a. *Application Manager*: Este componente es usado por los usuarios finales para la interacción de alto nivel con el *middleware.*

- Los usuarios pueden crear aplicaciones IoT que contenga interfaces de voz. Estas aplicaciones contienen un conjunto de requisitos que son definidos a partir del dominio de aplicación. Sin embargo, como se observa en la sección

4.4, existen requisitos genéricos que toda aplicación IoT tiene y por ende, el middleware debe atender.

- El usuario interactúa con la aplicación a través de la voz para emitir comandos de actuación o monitorización a los dispositivos que hacen parte de esa aplicación. Como resultado el usuario obtiene el comando de voz en formato texto, la intención, la entidad y la respuesta emita por el dispositivo.
- El usuario puede configurar las funcionalidades que tiene el módulo de reconocimiento del habla.

Figura 27. Vista funcional de SWITCH



Fuente: El autor

5.3.1.2 Capa de gestión. Esta capa se utiliza para componer y orquestar los servicios del *middleware* con diferentes niveles de abstracción.

b. *Data Management*: Este componente debe permitir la configuración y administración de la base de conocimiento que utilizará el *middleware* y que será responsable por:

- Almacenar la información de las aplicaciones creadas por los usuarios con sus respectivos requisitos.
- Almacenar la información de los usuarios habilitados para usar el middleware.
- Almacenar información relacionada con los servicios disponibles del middleware.
- Almacenar los datos recolectados por los sensores.
- Almacenar las intenciones y sus respectivas entidades que sirven para la categorización del texto obtenido a través del módulo de reconocimiento del habla.

c. *Service Management*: Este componente administra los servicios que provee el *middleware* entre los cuales se encuentran:

- Servicio para el reconocimiento del habla: Obtiene la señal del habla enviado por el Application Manager y activa el servicio para enviarlo al Speech Recognizer para su posterior conversión a texto. Una vez finaliza el proceso de reconocimiento, este servicio obtiene como resultado el texto derivado del proceso.
- Servicio de categorización del texto: El texto es categorizado de acuerdo a las entidades almacenadas en la base de conocimiento. Una entidad está compuesta por una serie de características (intención, acción, identificador, entre otras) que permiten reconocer cual es la acción ordenada por el usuario a través del habla.
- Servicio de entrenamiento para la categorización del texto: El Service Management recibe a través de la interfaz Web un comando en formato texto y le permite al usuario entrenar de forma manual ese nuevo comando.
- Servicio de ejecución de comando: Envía al Resource Management las entidades reconocidas para su respectiva ejecución.
- Servicio de monitorización: Recibe y administra los datos enviados por los dispositivos.
- - Servicio de datos: Envía y obtiene información de la base de conocimiento de acuerdo al proceso solicitado.

En general, este componente coloca a disposición del usuario los servicios que tiene el SWITCH, es flexible cuando nuevos servicios deber ser añadidos al middleware, es quien orquesta los servicios entre los diferentes componentes con los que tiene relación.

d. *Service Discovery*: Este componente monitoriza la red de dispositivos para descubrir recursos de forma automática, los cuales pueden ser administrados por el *middleware* como nuevos servicios. Esta información la obtiene a través del *Network Manager*. Los tipos de recursos que puede tener la red son: hardware heterogéneos, alimentación y memoria de dispositivos, convertidores de análogo/digital, módulo de comunicación, módulo de software y servicios propios de cada dispositivo. Al ser un proceso descentralizado, cada recurso posee diferentes componentes que el programador percibe como un solo sistema.

e. *Speech Recognizer*: Este componente es el encargado del proceso para el reconocimiento del habla.

- Realiza el proceso de conversión de voz a texto.
- Envía la transcripción del habla al Service Management para su respectiva categorización.
- Cuenta una base de datos que contiene: (i) modelos acústicos; (ii) léxico de reconocimiento y; (iii) corpus del texto.
- Al tener su propia base de datos este componente se puede reutilizar en otro sistema en caso que se requiera.
- Facilita al usuario la administración de sus funcionalidades.

f. *Resource Management*: Este componente monitoriza, resuelve conflictos y asigna de forma justa los recursos debido a que se espera que todas las aplicaciones tengan una calidad del servicio aceptable.

- Obtiene a través del Data Management las aplicaciones que existen con sus respectivos requisitos de usuario.
- Analiza las aplicaciones que existen y sus requisitos, y los servicios y recursos disponibles en el middleware. Basado en estos análisis, toma la mejor decisión de como asignar recursos en la red, enviando la decisión en forma de mensaje al Network Manager.
- Recibe del Service Management las entidades obtenidas a partir de la categorización del texto. Esta información es analizada y enviada al Network Manager.

5.3.1.3 Capa de comunicación. Esta capa provee comunicación entre los dispositivos, la topología de red y los servicios que provee el *middleware.*

g. *Network Manager*: Este componente administra la infraestructura de red en el *middleware*, asegurando que exista escalabilidad sin alterar ningún servicio existente.

- Recibe la orden de ejecución proveniente del Resource Management.
- Verifica si el dispositivo se encuentra disponible para ejecutar la orden.
- Facilita información sobre el estado actual de la red.
- Obtiene información sobre el estado actual de los dispositivos.
- Publica los dispositivos disponibles de la red.
- Garantiza cobertura y conectividad de la red.
- Verifica el estado de energía de la red.

5.3.1.4 Capa de dispositivo. Esta capa expone los dispositivos para hacerlos accesibles y visibles a otras partes del *middleware*.

h. *Device Management*: Este componente administra los dispositivos que hacen parte de la infraestructura de red del *middleware* y los datos que se generen a partir del proceso que ellos realizan.

- Recibe el comando de actuación o monitorización proveniente del Network Manager.
- Envía información del resultado de la ejecución.
- Publica el estado actual de sus dispositivos.

5.3.2 Vista de servicios de SWITCH. La vista de servicios describe cómo interactúan los componentes entre sí a través de servicios, los cuales deben ser admitidos por las aplicaciones que se generen a partir de SWITCH. Esta información es útil para garantizar la interoperabilidad entre las aplicaciones que se generen a partir de la implementación de la arquitectura de SWITCH.

En la Figura 27 se muestran esos servicios a través de las interfaces. Una interfaz es un conjunto de operaciones que describen alguna parte del comportamiento de una clase y le dan acceso (Pressman, 2010). Los servicios se definieron en función de los requisitos funcionales presentados en la sección 4.5.
En la Tabla 26 se describe la funcionalidad de cada servicio, la relación entre los servicios y los requisitos funcionales de SWITCH, y se identifica el componente de la arquitectura que proporciona este servicio.

Tabla 26. Vista de servicios de SWITCH

Servicio	Descripción	Componente	Requisito
Microphone	Se activa cuando el usuario lo requiere, recibe la señal del habla y la guarda en un formato de audio.	*Application Manager*	Req-[2]
Web	Es usada por los usuarios para crear, actualizar y acceder a las aplicaciones.	*Application Manager*	Req-[1], Req-[14], Req-[18], Req-[26]
setASRConfiguration	Recibe los requisitos de la aplicación IoT para configurar la plataforma de ASR.	*Speech Recognizer*	Req-[9], Req-[10], Req-[11], Req-[12], Req-[13]
SignalConversion	Recibe como parámetro de entrada un formato en audio y devuelve un archivo de texto.	*Speech Recognizer*	Req-[3], Req-[4], Req-[5]
getServices	Activa servicios de acuerdo a los requisitos del sistema o del usuario.	*Service Management*	Req-[8], Req-[18], Req-[24]
setEntityConf	Recibe los requisitos de configuración de la categorización del texto en entidades reconocibles.	*Service Management*	Req-[27]
setResources	Envía información sobre los recursos disponibles ya sean hardware o software hacia administrador de servicios.	*Service Management*	Req-[18], Req-[22]
setSpeechSignal	Envía la señal de voz obtenida por la aplicación al *Service Management.* Una vez se obtiene la transcripción del habla a texto, esta interfaz la devuelve a la aplicación.	*Service Management*	Req-[6], Req-[14], Req-[20]
SendSensorData	Recibe los datos de todos los dispositivos y los envía al *Service Management.*	*Service Management*	Req-[18]
setData	Guarda información de los servicios, requisitos de aplicación en la base de conocimiento.	*Data Management*	Req-[17], Req-[18], Req-[27]
getData	Proporciona datos almacenados en la base de conocimiento a los componentes *Service y Resource Management.*	*Data Management*	Req-[7], Req-[18], Req-[23], Req-[27]
ReceiveCommand	Recibe las entidades obtenidas a partir de la categorización del texto y analiza la disponibilidad de los recursos en la red.	*Resource Management*	Req-[18], Req-[19]
getCurrState	Obtiene el estado actual de la infraestructura de red y muestra los servicios que hay en el *middleware*.	*Network Manager*	Req-[18], Req-[22]
setNetConf	Envía información de la configuración de la red de forma específica de acuerdo a lo solicitado por el *Resource Management*.	*Network Manager*	Req-[15], Req-[16], Req-[22]

Servicio	Descripción	Componente	Requisito
publishDevice	Publicar los nuevos dispositivos disponibles para ser gestionados por el administrador de red.	*Network Manager*	Req-[19], Req-[22], Req-[25]
getSensorData	Obtiene el estado y los datos enviados por los dispositivos de la infraestructura de red.	*Network Manager*	Req-[18], Req-[22], Req-[25]
sendCommand	Recibe el mensaje de ejecución proveniente del habla y envía información del resultado de la ejecución.	*Device Management*	Req-[25]

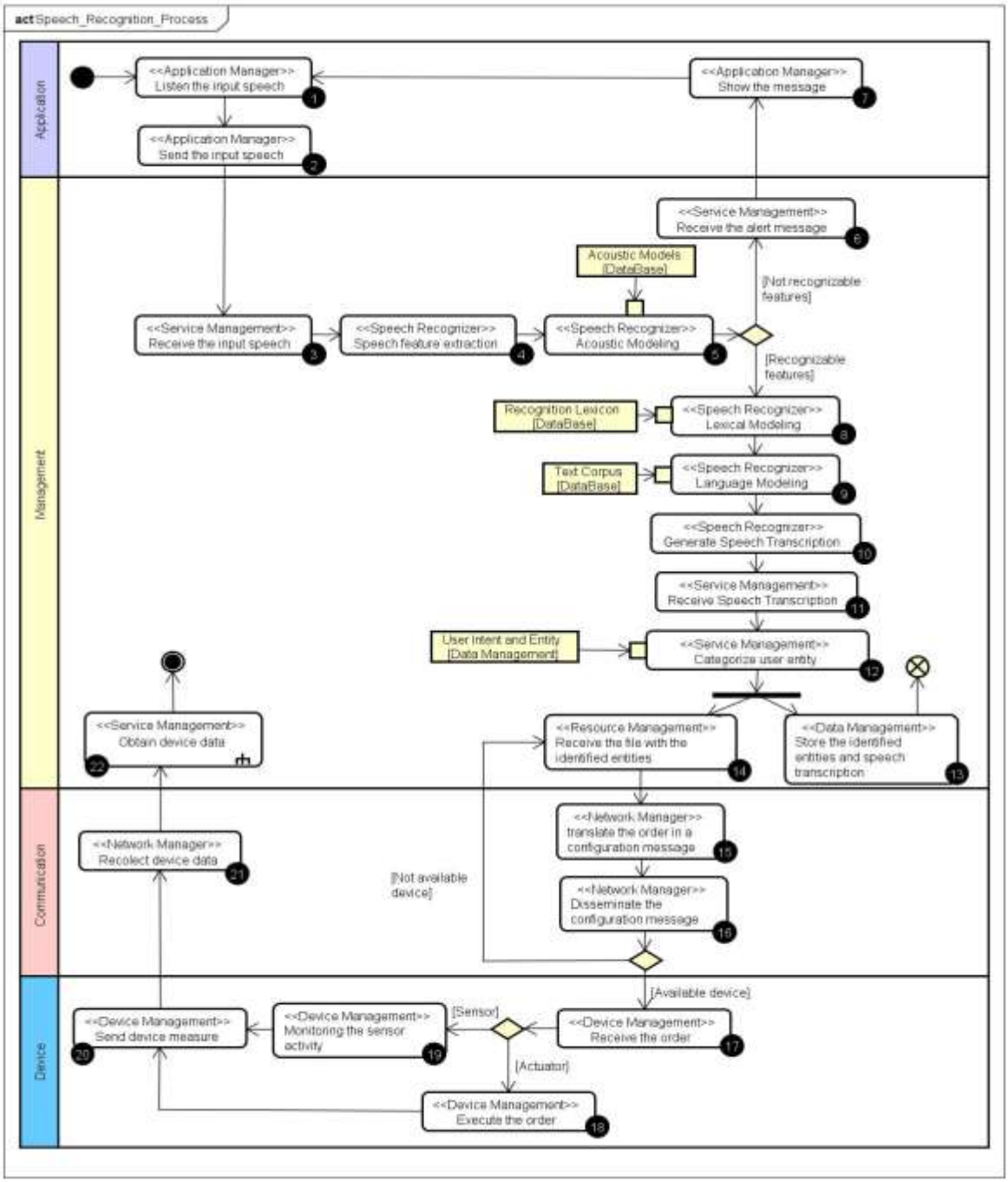
Fuente: El autor

5.3.3 Vista de procesos de SWITCH. Esta vista permite entender cuál es el comportamiento de la arquitectura para ciertos escenarios y las principales funciones que tendrían las aplicaciones que se generen a partir de SWITCH. La vista de proceso de SWITCH para el reconocimiento del habla se representa a través del diagrama de actividades descrito en la Figura 28.

El proceso comienza con la actividad 1, donde el *Application manager* escucha el comando de voz emitido por los usuarios a través de la aplicación y lo envía al *Service Management* (actividad 2). A continuación, este componente activa el servicio para el reconocimiento del habla y en la actividad 3 envía el comando de voz al *Speech Recognizer*. Este componente comienza el proceso de reconocimiento con la extracción de características del habla (actividad 4). La extracción de características implica el análisis de la señal del habla. Esta extracción tiene dos estados: análisis temporal (forma de la onda acústica) y técnica de análisis espectral (representación espectral de la señal del habla). Una de las técnicas de análisis de señal más potentes para estimar los parámetros básicos del habla es el método de predicción lineal (en inglés, *Linear Predictive Coding – LPC*). El algoritmo toma el habla y lo compara con una combinación lineal de muestras de habla pasadas. Al minimizar la suma de las diferencias cuadradas (en un intervalo finito) entre las muestras de habla reales y los valores predichos, se puede determinar un conjunto único de parámetros o coeficientes de predicción (coeficientes cepstrales).

En la actividad 5 se reciben esas características y se realiza el modelamiento acústico. Este modelado procesa las características de la señal del habla discriminando las clases de unidades básicas del habla teniendo en cuenta la variabilidad del habla con respecto al canal y el entorno del hablante. Las características se transforman en fonemas, es decir, unidades segméntales postuladas para un sistema fonológico que da cuenta de los sonidos de una lengua. Para realizar esta actividad, el modelamiento acústico puede usar algoritmos como el modelo oculto de Márkov (en inglés, *Hidden Markov Model - HMM*), modelo mixto gaussiano (en inglés, *Gaussian Mixture Models - GMM*), redes neuronales (en inglés, *Neural networks*), entre otros.

Figura 28. Vista de procesos de SWITCH – Proceso ASR 1



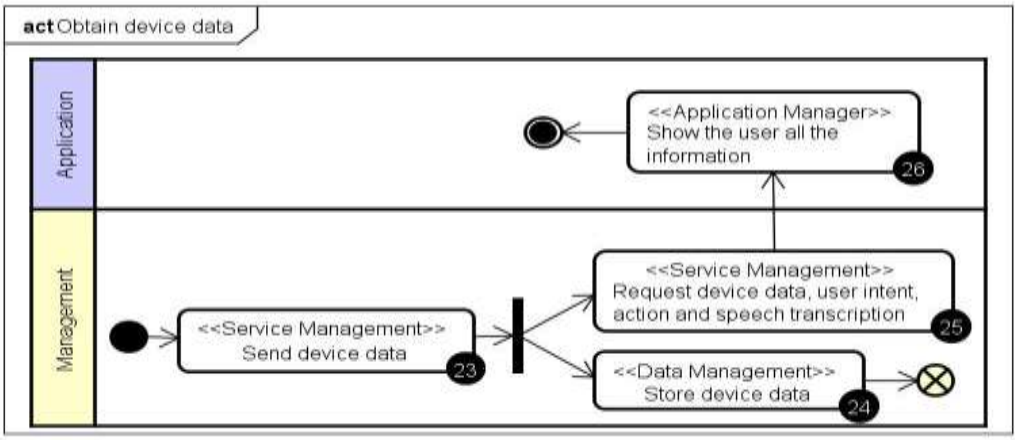
Fuente: El autor

En caso que las características no sean identificadas por exceso de ruido o por ausencia de audio reconocible, en la actividad 6 se envía una alerta al *Service Management* y éste a través de la interfaz *setSpeechSignal* envía el mensaje a la aplicación del usuario (actividad 7). El proceso vuelve a iniciar en la actividad 1 si el usuario así lo requiere. Si las características son reconocidas, se procede al modelamiento del léxico en la actividad 8. El modelado de léxico tiene como objetivo generar el vocabulario de reconocimiento y asignar cada *token* ortográfico usando la base de datos con el léxico de reconocimiento que incluye palabras o sub-palabras con la representación hablada correspondiente (transcripción fonética), de tal forma que los fonemas sean reconocidos como palabras. Una vez se obtienen las palabras, en la actividad 9 se realiza el modelamiento del lenguaje. En este proceso se imponen las restricciones sobre las hipótesis de reconocimiento generadas durante el ASR y se modela la estructura, sintaxis y semántica del lenguaje objetivo. Las palabras se entienden como oraciones con sentido haciendo uso del corpus del texto. Los modelos de lenguaje estadísticos proveen un estimado de probabilidad en la secuencia de palabras. Uno de los métodos más eficientes es el *n-gram,* donde la distribución de probabilidad depende de la disponibilidad de los datos de entrenamiento.

Una vez finalizado el proceso de transcripción del habla al texto, se genera un archivo tipo texto (actividad 10), el cual es enviado al *Service Management* a través de la interfaz *SignalConversion.* El *ServiceManagement* recibe el archivo de texto en la actividad 12 y comienza a realizar el proceso de categorización de las entidades del texto para identificar que acción fue ordenada por el usuario a través del comando de voz. Para esto, la consulta en la base de datos donde se encuentran guardadas las entidades con sus respectivas acciones. Una vez finaliza el proceso de categorización, realiza la actividad 13 para actualizar la base de datos con el archivo de voz, archivo de texto y las entidades identificadas. De forma simultánea, en la actividad 14 se envía al *Resource Management* el archivo con las entidades identificadas. Este componente se encarga de analizar los servicios y recursos disponibles en el *middleware.* En la actividad 15 el componente *Network Manager* recibe el archivo con las entidades identificadas, traduce esos parámetros en un mensaje de configuración y difunde este mensaje al *Device Management.* En la actividad 17, este componente identifica si la actividad la realiza un sensor (monitorización de la actividad) o actuador (ejecución de la orden). En la actividad 20 envía el mensaje de respuesta con los datos de los dispositivos al *Network Manager* y este a su vez lo envía al componente de servicios (actividad 22).

En la Figura 29 se muestra el proceso para la finalización de la actividad relacionada con el reconocimiento del habla.

Figura 29. Vista de servicios de SWITCH – Proceso ASR 2

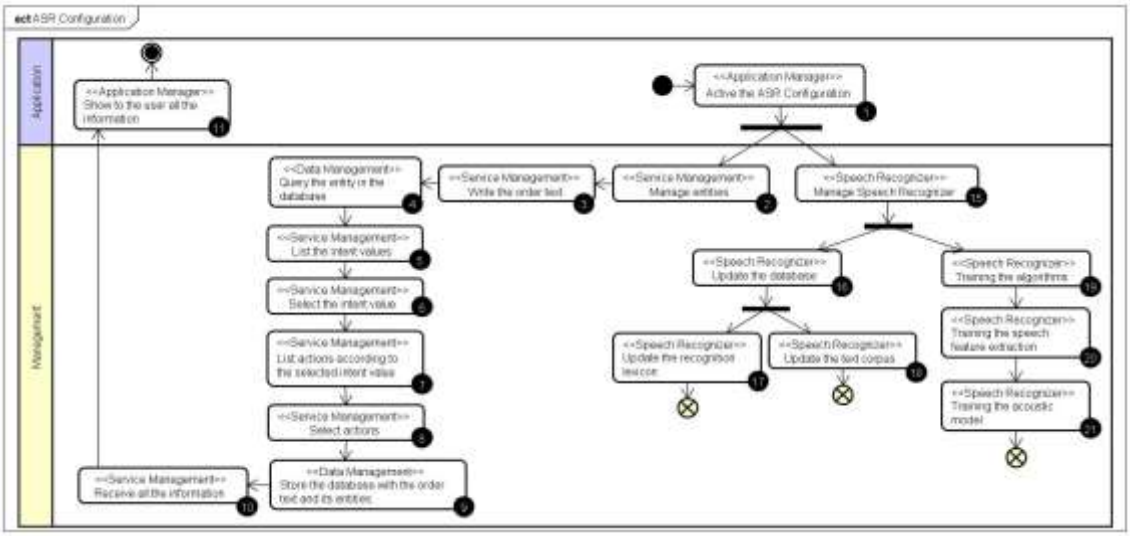


Fuente: El autor

El *Service Management* recibe los datos de los dispositivos y de forma simultánea realiza la actividad 24 para actualizar la base de datos con los datos provenientes de los dispositivos y muestra al usuario a través del *Application Manager* lo que el usuario habló, la transcripción del habla a texto, las entidades que fueron detectadas y la acción realizada por los respectivos dispositivos.

La vista de proceso de SWITCH para la configuración del reconocedor del habla y la gestión de entidades se representa a través del diagrama de actividades descrito en la Figura 30.

Figura 30. Vista de procesos de SWITCH – Configuración del ASR



Fuente: El autor

A través del *Application Manager* el usuario puede administrar la configuración de los módulos para el reconocimiento del habla.

En la actividad 2 se gestiona el entrenamiento de entidades. Una vez el usuario ha seleccionado la aplicación que desea gestionar, se listan los dispositivos que hacen parte de esa aplicación. Para esto, se usa la interfaz *setEntityConf* provista por el *Service Management.* En la actividad 3 se digita la orden de ejecución en formato texto. En la actividad 4 se realiza una consulta a la base de datos para traer todas las entidades que pertenezcan a esa aplicación. Acto seguido se listan los valores de las intenciones para seleccionar cual es la que está más relacionada con la orden de ejecución. De acuerdo a la intención seleccionada, en la actividad 7, se listan las entidades provistas como acciones, las cuales hacen parte de la intención seleccionada. En caso que solo exista una acción por categoría, estas se dejan por defecto. Una vez categorizada la orden de ejecución, se realiza la actividad 9 para actualizar la base de datos la nueva categorización de la orden de ejecución dada. El *Service Management* recibe la respuesta de la base de datos y muestra al usuario la nueva información almacenada.

Las entidades son alimentadas a medida que el componente *Device Management* avisa al *middleware* SWITCH través de la interfaz *publishDevice* que existe un nuevo dispositivo en el sistema. A nivel de código, el dispositivo debe registrar su localización (entidad *where*) dentro del ambiente donde está funcionando la aplicación. Cabe resaltar que usualmente las aplicaciones IoT que tienen interfaces para el reconocimiento del habla, el usuario es quien llena las entidades de forma automática, teniendo un margen de error a nivel sintáctico provisto por el propio usuario. SWITCH controla de forma automática el uso de las entidades en el sistema, es decir, el usuario no las puede crear lo que facilita la gestión de las mismas sin margen de error a nivel sintáctico.

En la actividad 15, el usuario administra el reconocedor del habla a través de la interfaz *setASRConf*. Para esto, no es necesario que se listen las aplicaciones creadas previamente. En este proceso el usuario puede realizar dos actividades, a saber: (i) actualizar la base de datos del reconocedor y (ii) entrenar los algoritmos del reconocedor. La base de datos del reconocer tiene dos partes principales el léxico de reconocimiento y corpus del texto. La actividad 17 y 18 no son dependientes. Los algoritmos para entrenar son aquellos que se encuentran en el módulo de extracción de características del habla y el modelo acústico (actividades 20 y 21 respectivamente).

5.3.4 Interfaz gráfica de usuario. Conocida también GUI (en inglés, *graphical user interface*), permite representar con un entorno visual sencillo la información y acciones disponibles que tendrá acceso el usuario. Dado que la función principal de un *middleware* es facilitar el desarrollo de aplicaciones, como parte del diseño de

este proyecto, a continuación se presentan las principales interfaces del diseño de SWITCH.

Figura 31. Interfaz gráfica SWITCH - Inicio

El diseño de SWITCH tiene una interfaz Web donde el usuario gestiona las aplicaciones, como se observa en la Figura 31. La GUI de inicio tiene tres módulos principales, a saber: (i) Aplicaciones; (ii) Configuración del ASR e (iii) Información general del proyecto.

Fuente: El autor

En el módulo de aplicaciones, el usuario puede crear aplicaciones y administrarlas. En la Figura 32 se muestra el proceso que el usuario debe llevar a cabo para crear las aplicaciones.

Figura 32. Interfaz gráfica SWITCH - Crear aplicaciones

Fuente: El autor

En primera instancia, a través de la información obtenida por el servicio de monitorización activo en el *Network Manager*, se listan al usuario los dispositivos disponibles en la red. El usuario digita el nombre de la aplicación, fecha de inicio, fecha de finalización (no obligatoria) y el dominio de aplicación. Si el nombre de la aplicación ya existe, el sistema emitirá el mensaje solicitando un nuevo nombre. Cada aplicación por defecto tiene un ID único. El usuario selecciona los dispositivos que harán parte de su aplicación y los añade como nuevos requerimientos. Al hacer clic en el botón guardar, esta información es enviada a la base de conocimientos a través de la interfaz *setData.*

El usuario puede administrar todas las aplicaciones que ha creado previamente. En la Figura 33 se detalla las funciones que tiene esta GUI. A través del servicio de datos, se obtiene información de la base de conocimiento sobre las aplicaciones que pertenecen a ese usuario y se listan en pantalla sus características. El usuario selecciona la aplicación que necesita administrar y en pantalla aparecen los dispositivos que hacen parte de ella.

Figura 33. Interfaz gráfica SWITCH - Administrar aplicaciones

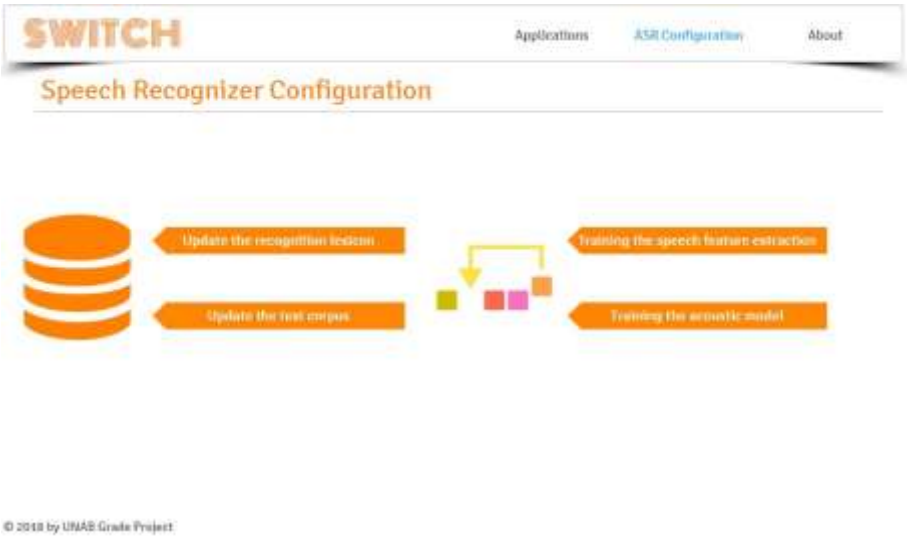
Fuente: El autor

El usuario puede hacer uso del módulo de reconocimiento del habla para enviar un comando de ejecución a cualquiera de los dispositivos que fueron listados. En pantalla aparecerá el comando de voz en formato texto, las entidades identificadas y la respuesta enviada por los dispositivos una vez se ejecuta el comando.

Por otro lado, el usuario puede entrenar las entidades que son usadas para la clasificación del texto que se genera del comando de voz. En primer lugar digita la orden y le da enviar. Internamente esa orden es recibida por el componente *Service Management* y se activa el servicio de entrenamiento para la categorización del texto. El usuario digita el comando y da clic en el botón *Go.* El *Service Management* solicita a la base de datos todas las entidades concernientes con la aplicación que se está administrando. De forma manual, el usuario selecciona la intención a la cual pertenece el comando de texto. Por ejemplo, si el comando digitado por el usuario fue *"Turn off the lights of my favorite place"*, la intención que deberá seleccionar el usuario es *"Light".* El sistema lista las entidades relacionadas con la intención seleccionada. Para este ejemplo sería las acciones *"On and Off",* los lugares *"Living Room and Garage",* identificador *"Yellow and Red".* De acuerdo al comando digitado, el usuario deberá seleccionar la acción *"Off",* si su lugar favorito es la sala, entonces el lugar es *"Living Room"* y el identificador es *"Yellow"*. Cada vez que se mencione en el comando el "lugar favorito", el reconocedor analice que se trata de la sala y en ese lugar se encuentra la luz amarilla. Finalmente, en la base de datos se almacena el comando digitado y las respectivas entidades entrenadas.

Además de las funciones mencionadas anteriormente, el usuario podrá configurar el modulo para el reconocimiento del habla, ver Figura 34.

Figura 34. Interfaz gráfica SWITCH - Configurar ASR



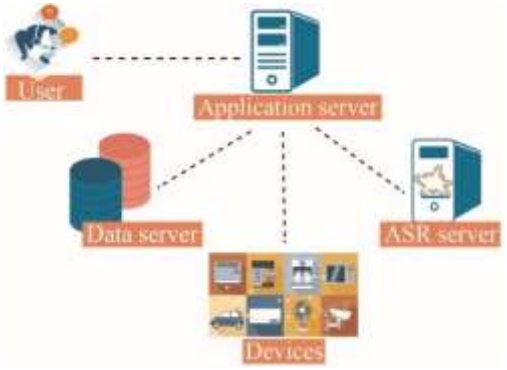
Fuente: El autor

A través de la interfaz *setASRConf* el usuario puede actualizar la base de conocimiento del léxico de reconocimiento y el corpus del texto. Además, podrá entrenar los algoritmos para la extracción de características y el modelamiento acústico.

5.4 MODELADO DE LOS COMPONENTES DEL *HARDWARE*

El modelado de componentes del *hardware*, describe el ambiente en el que el sistema será instalado y ejecutado, incluyendo las dependencia que el sistema tiene en el ambiente de ejecución. En la Figura 35 se observa el *hardware* requerido para el despliegue de SWITCH.

Figura 35. Elementos de *hardware* de SWITCH



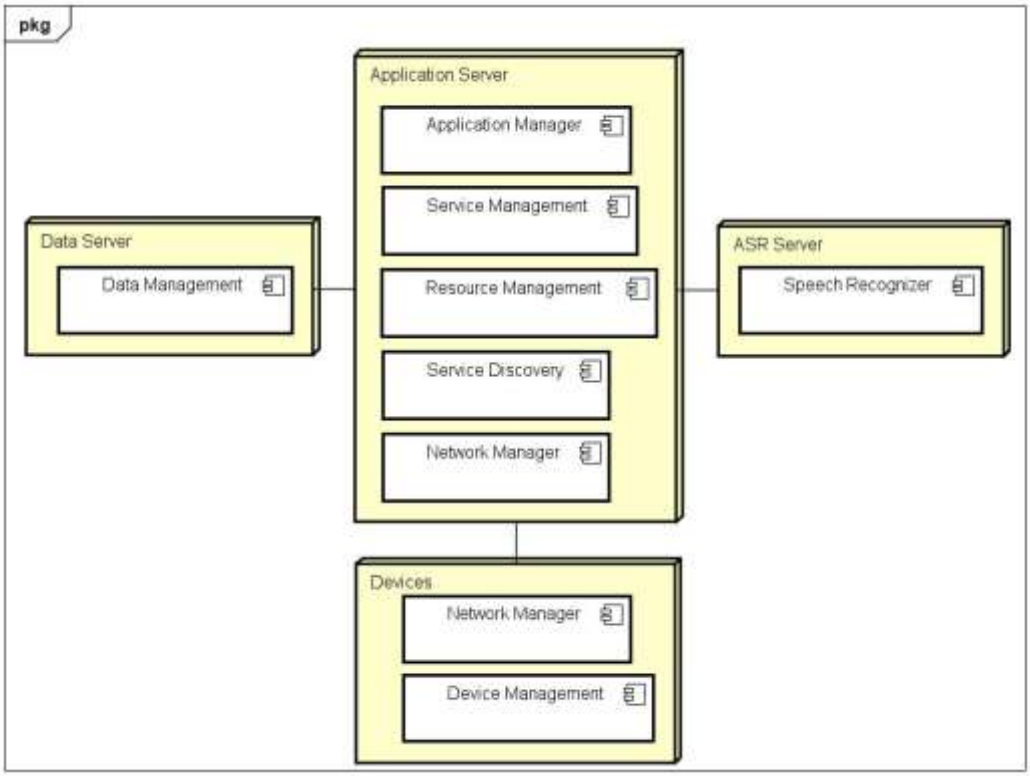
Fuente: El autor

En la Figura 36 se modela la implementación física de los componentes de la arquitectura del *middleware* SWITCH realizada a través de nodos y sus conexiones de red que simbolizan la interacción entre estos nodos. Cada nodo representa una pieza física del equipo donde se implementará una instancia de SWITCH.

En la vista de implementación de SWITCH se consideran los siguientes elementos:

- Servidor de aplicaciones, es un nodo que contiene los servicios proporcionados por los componentes de la capa Application (Application Manager) y los siguientes componentes de la capa Management (Service Management, Resource Management y Service Discovery). En este nodo también se guardan funcionalidad del componente Network Manager el cual permite gestionar la comunicación y transmisión de datos en todo el middleware.
- Servidor de datos, es un nodo que contiene los servicios proporcionados por el componente Data Management y es responsable por el almacenamiento de los datos provenientes en los diferentes procesos que realiza el middleware. La solicitud de información a este nodo puede ser realizada por los componentes Application Manager, Resource Management y Service Management.

Figura 36. Vista de implementación de SWITCH



Fuente: El autor

- Servidor ASR, es un nodo que contiene el componente para el reconocimiento del habla. Administra las bases de datos propias del ASR como el léxico de reconocimiento, el corpus del texto y los diferentes algoritmos utilizados en cada módulo del reconocedor (ver Figura 28). Este nodo puede ser fácilmente reutilizado por un sistema externo al middleware SWITCH, ya que cuenta con los servicios suficientes que debe tener un ASR.
- Dispositivos, este nodo representa los diferentes dispositivos que hacen parte de SWITCH. Estos dispositivos pueden ser asignados a las aplicaciones que se generen a partir del middleware. Se consideran dos tipos de dispositivos: sensores y actuadores. Sin embargo, SWITCH puede ser escalable. Para facilitar la comunicación entre la red y el servidor de aplicaciones, este nodo contiene funcionalidades del componente Network Manager.

6 EVALUACIÓN DEL DISEÑO DE SWITCH

En este capítulo se describe el proceso de evaluación del *middleware* diseñado. De acuerdo a los resultados obtenidos en la búsqueda de literatura con los términos "*middleware* e IoT" realizada en la sección 2.3 *"Estado del arte",* no se obtuvieron de forma implícita instrumentos para evaluar *middleware*. Para obtener una evaluación de su diseño, realizaron pruebas de concepto o prototipos. Cabe resaltar que la mayoría de los diseños son del orden *"Open Source"*. Con base en lo anterior, para la evaluación de SWITCH se desarrollaron tres etapas descritas en las siguientes secciones: En la sección 6.1 se seleccionó la actividad del *middleware* relacionada con el reconocimiento de voz y se realizó una prueba de concepto. En la sección 6.2 se realiza un análisis comparativo entre SWITCH y los *middleware* consultados en la literatura. Finalmente, en la sección 6.3 se adapta un instrumento de evaluación genérico para el diseño de *middleware* para IoT.

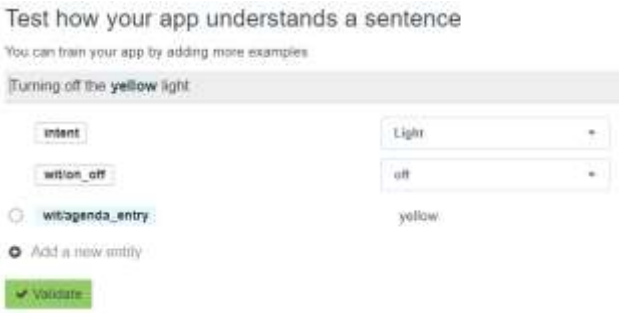
6.1 PRUEBA DE CONCEPTO

En esta prueba de concepto, se seleccionó un escenario de las actividades propuestas en la Figura 28 que involucra el proceso genérico que debe realizar cualquier reconocedor del habla sin la ejecución de los dispositivos. Para efectos de la prueba de concepto de SWITCH se utilizó la plataforma Web *wit.ai*.

A nivel de interfaz

1. El usuario usando la interfaz Web del *Application Manager* crea una aplicación con nombre, descripción y el idioma a utilizar en el reconocedor del habla.
2. El proceso de ASR se realiza en el API de *wit.ai*[27]*.* Para hacer este proceso, previamente se entrenaron de forma manual las entidades (Ver Figura 37).

Figura 37. Entrenamiento de entidades



Fuente: Tomado de *wit.ai*

[27] Más información en la página Web https://wit.ai/

A nivel de código

1. El componente *Application Manager* a través de la interfaz *Microphone* recibe la señal del habla y la guarda en un archivo de audio

```
#COMPONENT: Application manager
#DESCRIPTION: The user voice is taken through the microphone interface
#                              generatin an audio file (.wav)

import pyaudio
import wave

def microphone(RECORD_SECONDS, WAVE_OUTPUT_FILENAME):
    #--------- SETTING PARAMS FOR OUR AUDIO FILE ------------#
    FORMAT = pyaudio.paInt16    # format of wave
    CHANNELS = 2                # no. of audio channels
    RATE = 44100                # frame rate
    CHUNK = 1024                # frames per audio sample
    #--------------------------------------------------------#
    audio = pyaudio.PyAudio()

    # Open a new stream for microphone
    stream = audio.open(format=FORMAT,channels=CHANNELS,
                        rate=RATE, input=True,
                        frames_per_buffer=CHUNK)

    #----------------- start of recording --------------------#
    print("Listening the speech signal")
    frames = []

    for i in range(int(RATE / CHUNK * RECORD_SECONDS)):
        # read audio stream from microphone
        data = stream.read(CHUNK)
        # append audio data to frames list
        frames.append(data)

    #------------------ end of recording --------------------#
    print("Finished recording")
    stream.stop_stream()
    stream.close()
    audio.terminate()
    #------------------ saving audio ------------------------#

    waveFile = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()

def read_audio(WAVE_FILENAME):
    # The Application Manager return the audio
    with open(WAVE_FILENAME, 'rb') as f:
```

```
        audio = f.read()
    return audio
```

2. El *Service Management* recibe el archivo de audio a través de la interfaz *setSpeechSignal.* El componente activa el servicio para el reconocimiento y envía el archivo al reconocedor a través de la interfaz *SignalConversion*

```
import pyaudio
import wave
import requests
import json

# Wit speech API endpoint
API_ENDPOINT = 'https://api.wit.ai/speech'

# Wit.ai api access token
wit_access_token = 'B4LNPCQDV5V3DSL67KPNX4E7CBPMANH2'

def setSpeechSignal(AUDIO):
    # defining headers for HTTP request
    headers = {'authorization': 'Bearer ' + wit_access_token,
               'Content-Type': 'audio/wav'}

    # SPEECH RECONGNIZER
    # SERVICE: setSignalConversion
    # making an HTTP post request
    resp = requests.post(API_ENDPOINT, headers = headers,
                         data = AUDIO)

    # converting response content to JSON format
    data = json.loads(resp.content)

    return data
```

3. El *Service Management* recibe a través de la interfaz *SignalConversion* el archivo en formato texto generado a partir del ASR y el proceso de categorización de entidades para establecer cuál es la función que debe ejecutar el dispositivo. Al usuario se le devuelve como resultado el comando del habla en formato texto y la entidad reconocida.

```
import json

from ApplicationManager import microphone, read_audio
from ServiceManager import setSpeechSignal


def Switch(AUDIO_FILENAME, num_seconds = 5):

    # record audio of specified length in specified audio file
```

```
    microphone(num_seconds, AUDIO_FILENAME)

    # reading audio
    audio = read_audio(AUDIO_FILENAME)

    # SERVICE MANAGER
    # Service: setSpeechSignal
    # This service is used to translate audio file (sound) in a text
    data = setSpeechSignal(audio)
    text = data['_text']

    # DATA MANAGER
    # Service: getData
    # Get IoT command from ASR translation
    iotCommand = data['entities']['on_off'][0]['value']
    print "The device state is:", iotCommand

    # return the User speech
    return text

if __name__ == "__main__":
    text =  Switch('IoTCommand.wav', 4)
    print("\nSwitch - The user said: {}".format(text))
```

A nivel de componentes y de acuerdo a lo diseñado en la arquitectura de SWITCH (ver Figura 26), el usuario se comunica con el sistema a través de una aplicación, y esta a su vez con el componente *Application Manager* a través de la interfaz *Microphone*. La señal del habla se recibe y se guarda en un formato de audio (.wav). El archivo es enviado al *Service Management* a través de la interfaz *setSpeechSignal*. De esta forma se activa el servicio para el reconocimiento del habla. El archivo de audio es enviado por el *Service Management* a través de la interfaz *SignalConversion* para ser traducido a texto por el reconocedor de voz. En la prueba de concepto se emplea un ASR de una plataforma *OpenSource* que brinda el servicio de reconocimiento. Como parte del diseño de SWITCH se establece que el componente *Speech Recognize*r cuenta con sus propias bases de datos y algoritmos de reconocimiento, por ende, puede ser utilizado por otro sistema. A través de la interfaz *SignalConversion* se devuelve la traducción del habla en un archivo generado en formato texto (json) y se realiza el proceso de categorización de entidades. Al usuario se le da como resultado el comando del habla en forma texto y las entidades reconocidas en ese comando. De esta forma se cumple con el escenario planteado para el reconocimiento del habla.

6.2 ANÁLISIS COMPARATIVO

En esta sección se presenta un análisis comparativo entre SWITCH y los *middleware* consultados en la literatura. Para obtener los *middleware* propuestos

para IoT, en la sección 2.3 *"Estado del arte"* se realizó una revisión sistemática de la literatura cual se pudo identificar los requisitos que estos *middleware* contemplan. Como resultado se obtuvieron seis *middleware* descritos en la Tabla 3. A continuación se realiza la comparación de estos *middleware* con SWITCH en relación al cumplimiento de requisitos funcionales, requisitos no funcionales y sus características genéricas. En la Tabla 27 se presenta la comparación a nivel de requisitos funcionales.

Tabla 27. Comparación de *middleware* – Requisitos funcionales

Functional requirements								
Name	**Discovery**	**Resource**	**Data**	**Event**	**Service**	**Code**	**Device**	**Network**
WuKong	x	x	x	x	x	--	x	x
LinkSmart	x	x	x	x	x	--	x	x
OpenIoT	x	x	x	x	--	--	x	x
UIoT	x	x	x	--	--	x	x	x
Node-RED	--	--	x	x	--	x	x	x
MeSchup	x	--	x	x	--	x	x	x
SWITCH	x	x	x	--	x	--	x	x

Fuente: El autor

Wukong, LinkSmart y SWITCH son *middleware* con un enfoque de diseño basado en servicios. Este enfoque tiene ventajas como el descubrimiento de servicios, composición de servicios y reutilización de servicios. Sin embargo, la administración de código no se da fácilmente. Pero cabe resaltar, que la gestión de código no es un requisito base para el diseño de un *middleware* para IoT. En la Tabla 28 se presenta la comparación a nivel de requisitos no funcionales.

Tabla 28. Comparación de *middleware* – Requisitos no funcionales

Non-functional requirements											
Name	**Scalability**	**Security**	**Interoperability**	**Availability**	**Manteniability**	**Real-time**	**Visualization**	**Heterogeneity**	**Context-aware**	**Stream Processing**	**Operability**
WuKong	x	--	--	--	x	x	x	x	x	x	x
LinkSmart	--	x	x	x	x	x	x	x	x	x	x
OpenIoT	x	x	x	--	x	x	x	x	--	x	--
UIoT	x	x	x	x	x	--	x	x	--	x	--
Node-RED	--	--	x	--	--	x	x	x	x	x	x
MeSchup	x	x	x	x	x	x	x	x	x	x	x
SWITCH	x	x	x	x	x	--	x	x	x	x	x

Fuente: El autor

A excepción del *middleware* Node-RED, los demás presentan una arquitectura en capas. Esto facilita que el *middleware* atienda requisitos de calidad como escalabilidad, seguridad y mantenibilidad. En algunos casos, la documentación no los mencionaba.

La versión paga de Node-RED contiene módulos para el reconocimiento del habla a texto y viceversa, por ende, no es de fácil acceso para todos los usuarios. Con MeSchup se pueden construir aplicaciones con reconocimiento del habla pero en el diseño del *middleware* no se contempla este componente. Por su parte, SWITCH en su diseño contempla la utilización de un componente para el reconocimiento del habla a texto. En caso de ser requerido, este componente puede ser utilizado por un sistema externo ya que contiene sus propias bases de datos y no depende de otro componente dentro del *middleware.* El procesamiento de datos en tiempo real no es un requisito básico para el funcionamiento de SWITCH.

En conclusión, el diseño de SWITCH contempló la mayoría de los requisitos funcionales que un *middleware* para IoT requiere. Estos requisitos se evidencian en los componentes que integran las capas de la arquitectura propuesta. Sin embargo, falta evidenciar de forma más explícita la manera como SWITCH atiende los requisitos no funcionales.

6.3 INSTRUMENTO DE EVALUACIÓN

En el capítulo 5 se realizó el modelo de diseño de la arquitectura específica del *middleware* SWITCH que tiene un dominio, componentes que cumplen con requisitos funcionales y no funcionales para IoT y reconocimiento del habla, con sus respectivas interacciones. Dado que en la literatura no se evidenció un instrumento de evaluación de diseño de *middleware* para IoT, se adaptó el enfoque de inspección basado en una lista de verificación llamado FERA (en inglés, *Framework for Evaluation of Reference Architectures*) (Santos, Guessi, Galster, Feitosa, & Nakagawa, 2013). Aunque SWITCH no tiene una arquitectura genérica, es decir, no contempla todos los requisitos funcionales y no funcionales genéricos de un *middleware* que permita desarrollar cualquier aplicación IoT, si se puede evaluar a través de FERA puesto que sigue un modelo de dominio y patrones de diseño.

La construcción de la lista de verificación fue guiada por la literatura disponible sobre arquitecturas de referencia para IoT, requisitos funcionales y no funcionales de un *middleware* para IoT y sistemas de reconocimiento del habla. Se consideran los roles de los *stakeholders* involucrados en el desarrollo de un *middleware* para IoT tomando como base los mencionados en el capítulo 1 y los presentados en FERA, resultando, la lista presentada en la Tabla 29.

Tabla 29. *Stakeholders* para el desarrollo de un *middleware* IoT

Rol	Descripción
Experto del dominio	Posee conocimientos sobre *middleware* para IoT.
Arquitecto de *software*	Diseña la arquitectura del *middleware* y la instancia de acuerdo los requisitos funcionales y no funcionales de las aplicaciones para IoT.
Desarrollador de *software*	Implementa el diseño del *middleware* para IoT.
Administrador de *software*	Toma decisiones estratégicas sobre la administración de recursos y tiempo con base en la documentación de la arquitectura del *middleware* para IoT.
Gerente de redes	Comprende el área específica donde se va a desplegar el *middleware* para IoT, incluyendo la forma de comunicación, dispositivos y la topología de la red.
Administrador de base de datos	Comprende las entradas y salidas que se generan de los procesos realizados por el *middleware* para IoT y las almacena de forma ordenada.
Integrador	Responsable que se cumpla los atributos de calidad del *middleware* para IoT, fallos y aciertos operacionales.

Fuente: El autor

Además, se consideran nueve atributos de calidad que debe tener un *middleware* para IoT: Idoneidad funcional, eficiencia de rendimiento, interoperabilidad, usabilidad, confiabilidad, mantenibilidad, portabilidad, seguridad y escalabilidad. Estos atributos fueron determinados por la revisión sistemática de la literatura presentada en la sección 4.4. La lista de verificación consta de 50 preguntas como se observa en la Tabla 30, la cual puede ser usada por arquitectos de *software* con conocimientos en IoT. El formulario se encuentra en la siguiente dirección: *https://goo.gl/forms/T8vt9qflyzUg1o8g1*

Tabla 30. Instrumento de evaluación para arquitectura *middleware* para IoT

Pregunta	S	N	P	Observaciones
¿Se presenta información general de la arquitectura del *middleware*?				
¿La arquitectura del *middleware* presenta un alcance?				
¿La arquitectura del *middleware* presenta un objetivo?				
¿La arquitectura del *middleware* tiene la sección de requisitos?				
¿Están definidos los stakeholders para el diseño de la arquitectura del middleware?				
¿Están definidos los roles de los stakeholders?				
¿La arquitectura del middleware presenta la terminología del dominio?				

Pregunta	S	N	P	Observaciones
¿La descripción de cada término del dominio está completa?				
¿Las relaciones entre conceptos son claras?				
¿Es claro el inicio-fin de la participación del *stakeholder* en las capas de la arquitectura del *middleware*?				
Desde la ingeniería del *software*, ¿Se consideraron todos los puntos de vista para el diseño de la arquitectura del *middleware*?				
¿Hay una descripción de cada punto de vista utilizado?				
¿Cada punto de vista tiene un nombre específico?				
¿Cada punto de vista identifica de forma clara la participación del *stakeholder*?				
¿Cada punto de vista incluye un estilo arquitectural?				
¿Cada punto de vista usa un lenguaje de descripción arquitectural? (Ej: UML, SysUML)				
Para cada punto de vista, ¿sus componentes son claros y están bien definidos?				
¿Los componentes proporcionan suficiente información para determinar si las funciones enmarcadas por el punto de vista se han cumplido?				
¿La arquitectura del *middleware* satisface los estándares internacionales para IoT?				
¿El diseño de la arquitectura del *middleware* hace uso de buenas prácticas en Ingeniería de sistemas?				
¿Presenta la arquitectura del *middleware* viabilidad para ser implementada?				
¿El nivel de detalle presentado favorece la comprensión de la arquitectura del *middleware*?				
¿Presenta la arquitectura del *middleware* problemas relacionados con la mantenibilidad?				
¿Se consideró el atributo de calidad: Idoneidad funcional?				

Pregunta	S	N	P	Observaciones
¿Se consideró el atributo de calidad: Eficiencia de rendimiento?				
¿Se consideró el atributo de calidad: Interoperabilidad?				
¿Se consideró el atributo de calidad: Usabilidad?				
¿Se consideró el atributo de calidad: Confiabilidad?				
¿Se consideró el atributo de calidad: Seguridad?				
¿Se consideró el atributo de calidad: Portabilidad?				
¿Se consideró el atributo de calidad: Escalabilidad?				
¿La arquitectura del *middleware* especifica cuáles son los mecanismos de variabilidad? (ej: alternativas, elementos obligatorios)				
¿Se puede identificar los componentes funcionales?				
¿Se puede determinar las relaciones entre estos componentes?				
¿Se puede identificar las dependencias de tiempo de ejecución entre estos componentes?				
¿Los diagramas abordan todas las relaciones presentadas en el modelado de componentes?				
¿La arquitectura del *middleware* especifica cómo se tratará la comunicación con el entorno externo?				
¿Se han definido claramente los requisitos, restricciones, estándares y políticas de control de calidad de la arquitectura del *middleware*?				
¿El *middleware* puede evolucionar? ¿Si es así, cómo? ¿Si no, por qué no?				
¿Existen claras dependencias entre los atributos de calidad y el comportamiento de los componentes?				
¿Son los objetivos del dominio que debe cumplir la arquitectural del *middleware* claramente articulados y priorizados?				

Pregunta	S	N	P	Observaciones
¿La descripción del diseño del *middleware* permite una estimación del esfuerzo para implementarlo?				
¿Hay alguna manera de mostrar el beneficio por usar el diseño del *middleware*?				
¿Se puede establecer un cronograma para la implementación e integración del *middleware*?				
¿El diseño de la arquitectura del *middleware* provee suficientes recursos para su desarrollo?				
¿Se puede determinar enfoques para el manejo de errores?				
¿Se puede determinar enfoques para la interacción hombre-máquina?				
¿Se puede determinar enfoques para la gestión de datos?				
¿Se puede determinar los cambios que pueden tener el diseño de la arquitectura del *middleware* y cómo afectaría a sus componentes?				
¿Se puede identificar los elementos de *hardware*?				
Convenciones: *(S): Si (N): No (P): Parcial*				

Fuente: Adaptado de (Santos et al., 2013)

Este instrumento de evaluación facilita a los arquitectos de *software* la verificación de la descripción arquitectónica que tiene el diseño de un *middleware* para IoT con la intención de detectar defectos en esta descripción y continuar con las mejoras del mismo, antes de iniciar el proceso de implementación. Con la evaluación se puede concluir si: *(i)* el diseño del *middleware* proporciona información general (alcance, limitaciones, riesgos); *(ii)* el diseño del *middleware* contiene el modelado del dominio, modelado del *software* y *hardware*; *(iii)* la documentación relacionada con el diseño del *middleware* contiene información importante; tales como decisiones arquitectónicas, mejores prácticas, directrices, políticas, normas, estándares internacionales e interfaces entre componentes; *(iv)* el diseño del *middleware* considera atributos de calidad importantes para IoT; *(v)* el *middleware* se puede implementar fácilmente.

El diseño de la arquitectura de SWTICH fue evaluado por el grupo de investigadores del CEA IoT (Ver Tabla 31), usando la lista de chequeo descrita anteriormente y el documento versión 1.0 que contiene todo el diseño del SWITCH. El documento más

los resultados de la encuesta por pregunta se encuentran en detalle anexos a la carpeta final.

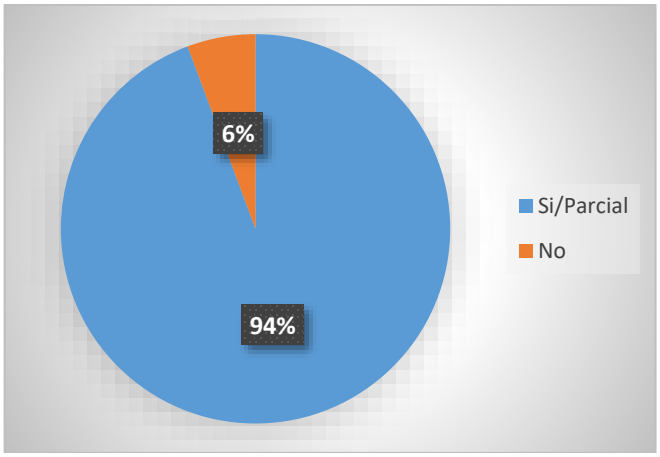
Tabla 31. Grupo de investigadores del CEA IoT

Rol	Nombre	Profesión
Gestión de redes y dispositivos	María Alejandra Culman	Ingeniera en Mecatrónica y Energía. Magister en Telemática.
Arquitecto de *software*	Johan Smith Rueda	Ingeniero de sistemas, especialista en seguridad informática y Magister en Telemática
Desarrollador de *software* y computación en la nube	Javier Pinzón Castellanos	Ingeniero en Electrónica y Magister en desarrollo *software* y Telemática
Administrador de *software*	José Yon García	Ingeniero de sistemas y Magister en Telemática
Gestión e Integrador de dispositivos	Edisson Landazabal	Ingeniero en Electrónica y Magister en Telemática
Experto del dominio	Jesús Martin Talavera	Ingeniero de sistemas, Magister en desarrollo *software* y Doctor en Ciencias computacionales

Fuente: El autor

Se realizó un promedio de los resultados obtenidos en las seis (6) evaluaciones. En la Figura 38 se detalla un gráfico donde se representa el porcentaje de completitud del diseño de SWITCH. El 94% (color azul) es el promedio entre la suma total entre satisfactorios y parcialmente satisfactorios y el 6% (color naranja) el promedio de lo no satisfactorio.

Figura 38. Promedio de las evaluaciones de SWITCH



Fuente: El autor

Se puede concluir que el diseño de SWITCH tiene un alcance claro. En ese diseño se especifica los requisitos funcionales y no funcionales de un *middleware* para IoT que contiene interfaces de voz. Se modela de forma clara el dominio, los componentes funcionales con sus respectivas interfaces, las actividades principales relacionadas con el ASR y el *hardware* para su implantación.

Como recomendación es importante detallar el tiempo de ejecución y protocolos de comunicación de cada componente. El diseño contempla la mayoría de los requisitos no funcionales. Para efectos de la implementación de la arquitectura propuesta, es importante especificar el estilo arquitectural y las buenas prácticas propuestas en cada componente. De esta forma, la participación de cada uno de los requisitos dentro del sistema, será más detallado. El valor agregado del *middleware* es su enfoque para la interacción hombre-máquina, gestión de recursos, almacenamiento de diferentes tipos de datos, descubrimiento de recursos, comunicación con sistemas externos y escalabilidad.

7. CONCLUSIONES

En este trabajo de investigación, se propone el diseño de SWITCH, un *middleware* IoT con interfaces basadas en voz. Para este fin, se realizó una búsqueda detallada en la literatura teniendo en cuenta las arquitecturas de referencia para IoT, las arquitecturas *middleware* para IoT y los sistemas de reconocimiento del habla, de los cuales se obtuvo los requisitos funcionales y no funcionales que son la base del diseño propuesto. Luego, se realizó el diseño del *middleware* usando el lenguaje de descripción arquitectural UML. En este lenguaje fueron modelados el dominio, la arquitectura, los componentes *software* y los componentes de *hardware* del *middleware* propuesto. Finalmente, para la evaluación de SWITCH se escogió un escenario del diagrama de actividades enfocado al reconocimiento del habla. Se realizó una comparación de análisis de requisitos funcionales y no funcionales con los *middleware* consultados en la literatura. Se adaptó un instrumento de evaluación para el diseño de *middleware* para IoT.

Con base a lo realizado en el proyecto, se puede concluir que:

- Un *middleware* es una herramienta *software* que oculta la complejidad en el desarrollo de aplicaciones y ayuda en el proceso de comunicación entre las capas de aplicación y dispositivo.
- La tendencia de desarrollar aplicaciones con interfaces de voz aporta en la tecnología portátil y los dispositivos IoT que contengan pantallas mínimas o inexistentes, ayudando a los usuarios a interactuar efectivamente con los dispositivos de forma no intrusiva. Esa interacción se puede lograr gracias a los sistemas para el reconocimiento del habla, quienes son los que traducen la señal de voz en texto. El texto es enviado a los dispositivos a través de un mensaje para ejecutar la orden dada por el habla.
- SWITCH reúne características de estas dos herramientas y las incluye en componentes funcionales dentro su diseño.
- SWITCH es una propuesta que una vez se implemente, contribuye a la solución de los problemas identificados en la literatura, a saber: *(i)* oculta la complejidad en el desarrollo de aplicaciones IoT; *(ii)* el reconocimiento del habla usado en las aplicaciones ayuda a las personas a interactuar de forma natural con los dispositivos; *(iii)* provee a los dispositivos interfaces digitales.
- SWITCH provee una interfaz Web para monitorizar y gestionar dispositivos desde Internet.
- El diseño del reconocedor del habla provisto por SWITCH se realiza en condiciones ideales, sin embargo, es importante diseñar mejoras en el alcance y eficiencia de este tipo de herramientas para ambientes ruidosos, diferentes idiomas, personas con discapacidad en el habla.

7.1 CONTRIBUCIONES REALIZADAS

Esta sección resume las principales contribuciones de este trabajo de investigación en relación con los objetivos establecidos.

Análisis de los requisitos funcionales y no funcionales de un *middleware* para el desarrollo de aplicaciones IoT que utilicen interfaces basadas en voz

- Un estado del arte como contribución a la literatura sobre *middleware* IoT (Ver sección 2.3). En este estado del arte se listan, analizan y clasifican los *middleware* que han sido creados usando como base requisitos de IoT. Se explica cómo es su arquitectura, requisitos, enfoque de diseño y si incluye como parte de su diseño módulos para el reconocimiento del habla. Se identifica una brecha de investigación relacionada con herramientas que oculten la complejidad en el desarrollo de aplicaciones IoT que contengan características de UbiComp, de tal forma que la comunicación entre humano-máquina sea no intrusiva.
- Una revisión sistemática de la literatura sobre las arquitecturas de referencia (AR) propuestas para aplicaciones IoT (Ver sección 4.1). De esta revisión resultaron 40 estudios primarios, en los cuales se presentan como arquitecturas de referencia para aplicaciones IoT. Estos estudios fueron evaluados para determinar cuáles eran genéricas, descartando aquellas que fueran específicas para un dominio de aplicación. Se puede concluir cuales eran las más mencionadas en la literatura. Se usó el RAModel (Nakagawa et al., 2012), un modelo para identificar cual era la más completa basado en la información provista por la literatura. Finalmente, se analizó la arquitectura y los requisitos de la AR más completa.
- Un estado del arte sobre sistemas para el reconocimiento del habla (Ver sección 4.3). En este estado del arte se analizan las diferentes implementaciones encontradas en la literatura que han sido usadas como parte de IoT, de las cuales se concluyen los tipos, el diseño arquitectural, requisitos y proceso más relevantes de los ASR.
- Un listado de requisitos funcionales y no funcionales genéricos que un *middleware* para IoT podría tener
- Un listado de requisitos funcionales y no funcionales usados para el *middleware* SWITCH propuesto, el cual incluye reconocimiento del habla.

Modelado de los módulos relacionados con el reconocimiento del habla en la arquitectura del *middleware* para IoT propuesto, usando un lenguaje de descripción arquitectural.

- Modelado del dominio de SWTICH que incluye todos los elementos que participan en el diseño.

- Modelado de la arquitectura de SWITCH. La arquitectura tiene un estilo en capas lo cual facilita el cumplimiento de requisitos no funcionales como la escalabilidad, seguridad y mantenibilidad.
- Modelado de los componentes *software* de SWITCH. Estos componentes se componente de 4 partes, a saber: *(i)* vista funcional; *(ii)* la vista de servicios; *(iii)* la vista de procesos y; *(iv)* la interfaz gráfica de usuario.
- Modelado de la implementación física de los componentes *hardware* de SWITCH. Este modelo explica cada nodo *hardware* y sus conexiones de red.

Evaluación de los módulos diseñados mediante la adaptación de un instrumento de valoración para que sea utilizado por expertos en diseño de *software.*

- Una prueba de concepto del escenario del diagrama de actividades enfocado al reconocimiento del habla.
- Una comparación de los *middleware* consultados en la literatura con SWITCH.
- Un instrumento para la evaluación de diseño de *middleware* para IoT adaptado.
- Una autoevaluación de SWITCH realizada usando como base el instrumento de evaluación diseñado.

7.2 TRABAJO FUTURO

Como trabajo futuro de SWITCH se propone:

- Implementar con la ayuda de un lenguaje de descripción arquitectural formal el diseño propuesto.
- Validar a través de prototipos funcionales el diseño de SWITCH.
- Mejorar la base de conocimiento del gestor de entidades alimentándola con otros tipos de entidades y acciones que sean fácilmente categorizadas con el texto emitido por el reconocedor del habla.
- Mover la interfaz del micrófono propuesta en el *middleware* a los dispositivos. De tal forma que el habla se pueda reconocer a partir de estos sensores.
- Realizar de forma semiautomática el proceso de entrenamiento para la categorización de entidades.
- Enviar el instrumento de evaluación a arquitectos *software* en universidades, centros de investigación y empresas para evaluar el diseño de SWITCH. Basado en la información recolectada, establecer mejoras al diseño y al instrumento de evaluación.

REFERENCIAS

Abdmeziem, M. R., Tandjaoui, D., & Romdhani, I. (2016). Architecting the internet of things: state of the art. In *Robots and Sensor Clouds* (pp. 55–75). Springer.

Abreu, D. P., Velasquez, K., Curado, M., & Monteiro, E. (2017). A resilient Internet of Things architecture for smart cities. *Annals of Telecommunications*, *72*(1–2), 19–30.

Adams, K. (2015). *Non-functional Requirements in Systems Analysis and Design*. Springer.

Addo, I. D., Ahamed, S. I., Yau, S. S., & Buduru, A. (2014). A reference architecture for improving security and privacy in Internet of Things applications. In *Mobile Services (MS), 2014 IEEE International Conference on* (pp. 108–115).

Afonso, S., Laranjo, I., Braga, J., Alves, V., & Neves, J. (2015). Multilingual Voice Control for Endoscopic Procedures. In *Internet of Things. User-Centric IoT* (pp. 229–235). Springer.

Akash, S. A., Menon, A., Gupta, A., Wakeel, M. W., Praveen, M. N., & Meena, P. (2014). A novel strategy for controlling the movement of a smart wheelchair using internet of things. In *Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS), 2014 IEEE* (pp. 154–158).

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, *17*(4), 2347–2376.

Al-Jaroodi, J., Aziz, J., & Mohamed, N. (2009). Middleware for RFID systems: An overview. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International* (Vol. 2, pp. 154–159).

Aldosari, H. M. (2015). A Proposed Security Layer for the Internet of Things Communication Reference Model. *Procedia Computer Science*, *65*, 95–98.

Alhamedi, A. H., Snasel, V., Aldosari, H. M., & Abraham, A. (2014). Internet of things communication reference model. In *Computational Aspects of Social Networks (CASoN), 2014 6th International Conference on* (pp. 61–66).

Association for computing machinery ACM. (2012). *CCS 2012*.

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

Baccaglini, E., Gavelli, M., Morello, M., & Vergori, P. (2015). A multimodal user interface using the webinos platform to connect a smart input device to the Web of Things. In *Pervasive and Embedded Computing and Communication Systems (PECCS), 2015 International Conference on* (pp. 1–5).

Bai, J. G., Wei, J. G., Chen, L., He, Y. Q., Wang, J. R., & Dang, J. W. (2013). Design and Implementation of a Housekeeper System. In *Applied Mechanics and Materials* (Vol. 437, pp. 394–398).

Banda, G., Chaitanya, K., & Mohan, H. (2015). An IoT protocol and framework for OEMs to make IoT-enabled devices forward compatible. In *Signal-Image Technology & Internet-Based Systems (SITIS), 2015 11th International Conference on* (pp. 824–832).

Bandyopadhyay, S., Sengupta, M., Maiti, S., & Dutta, S. (2011). A Survey of Middleware for Internet of Things. In A. Özcan, J. Zizka, & D. Nagamalai (Eds.), *Recent Trends in Wireless and Mobile Networks: Third International Conferences, WiMo 2011 and CoNeCo 2011, Ankara, Turkey, June 26-28, 2011. Proceedings* (pp. 288–296). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-21937-5_27

Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., & Meissner, S. (Eds.). (2013). *Enabling Things to Talk*. Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-40403-0

Bauer, M., Bui, N., De Loof, J., Magerkurth, C., Nettsträter, A., Stefa, J., & Walewski, J. W. (2013). IoT Reference Model. In *Enabling Things to Talk* (pp. 113–162).

Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-40403-0_7

Bell, A. G. (1881). The production of sound by radiant energy. *Science*, *2*(48), 242–253.

Bernabe, J. B., Hernández, J. L., Moreno, M. V., & Gomez, A. F. S. (2014). Privacy-preserving security framework for a social-aware internet of things. In *International conference on ubiquitous computing and ambient intelligence* (pp. 408–415).

Berners-Lee, T., Cailliau, R., Groff, J.-R., & Pollermann, B. (1992). World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, *2*(1), 52–58.

Besacier, L., Barnard, E., Karpov, A., & Schultz, T. (2014). Automatic speech recognition for under-resourced languages: A survey. *Speech Communication*, *56*, 85–100.

Blackstock, M., & Lea, R. (2016). FRED: A Hosted Data Flow Platform for the IoT. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs* (p. 2:1--2:5). New York, NY, USA: ACM. http://doi.org/10.1145/3007203.3007214

Bochmann, G. V. (1990). Protocol specification for OSI. *Computer Networks and ISDN Systems*, *18*(3), 167–184.

Borgia, E. (2014). The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, *54*, 1–31.

Bouraoui, H., Jerad, C., Chattopadhyay, A., & Hadj-Alouane, N. Ben. (2017). Hardware Architectures for Embedded Speaker Recognition Applications: A Survey. *ACM Transactions on Embedded Computing Systems (TECS)*, *16*(3), 78.

Boussard, M., Meissner, S., Nettsträter, A., Olivereau, A., Segura, A. S., Thoma, M.,

& Walewski, J. W. (2013). A Process for Generating Concrete Architectures. In *Enabling Things to Talk* (pp. 45–111). Springer.

Brown, A. (2016). *The role of voice in IoT applications*. Retrieved from https://www.strategyanalytics.com/strategy-analytics/blogs/iot/2016/02/19/the-role-of-voice-in-the-internet-of-things#.WD3wMPkrLcc

Buyya, R., & Dastjerdi, A. V. (2016). *Internet of Things: Principles and paradigms*. Elsevier.

Cavalcante, E., Alves, M. P., Batista, T., Delicato, F. C., & Pires, P. F. (2015). An analysis of reference architectures for the internet of things. In *Proceedings of the 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures* (pp. 13–16).

Ccori, P. C., De Biase, L. C. C., Zuffo, M. K., & da Silva, F. S. C. (2016). Device discovery strategies for the IoT. In *Consumer Electronics (ISCE), 2016 IEEE International Symposium on* (pp. 97–98).

Chaqfeh, M. A., & Mohamed, N. (2012). Challenges in middleware solutions for the internet of things. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on* (pp. 21–26).

Chelloug, S. A., & El-Zawawy, M. A. (2017). Middleware for Internet of Things: Survey and Challenges. *Intelligent Automation & Soft Computing*, *0*(0), 1–9. http://doi.org/10.1080/10798587.2017.1290328

CISCO. (2014). *The Internet of Things Reference Model*. San José, California. Retrieved from http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

CISCO. (2016). *Internet of Things at a Glance*. Retrieved from https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf

Colciencias. (2016). *Tipología de proyectos calificados como de carácter científico, tecnológico e innovación* (Vol. 4).

Costa, N., Pereira, A., & Serodio, C. (2007). Virtual Machines Applied to WSN's: The state-of-the-art and classification. In *Systems and Networks Communications, 2007. ICSNC 2007. Second International Conference on* (p. 50).

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed systems: concepts and design* (Fifth edit). Pearson education.

Davis, K. H., Biddulph, R., & Balashek, S. (1952). Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, *24*(6), 637–642.

De, S., Carrez, F., Reetz, E., Tönjes, R., & Wang, W. (2013). Test-enabled architecture for IoT service creation and provisioning. In *The Future Internet Assembly* (pp. 233–245).

Delicato, F. C., Pires, P. F., & Batista, T. (2017). The Resource Management Challenge in IoT. In *Resource Management for Internet of Things* (pp. 7–18). Springer.

Dino, J. (2008). Ames Technology Capabilities and Facilities. Retrieved January 5, 2017, from https://www.nasa.gov/centers/ames/research/technology-onepagers/hc-computing.html

Eisenhauer, M., Rosengren, P., & Antolin, P. (2010). HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems. In D. Giusto, A. Iera, G. Morabito, & L. Atzori (Eds.), *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications* (pp. 367–373). New York, NY: Springer New York. http://doi.org/10.1007/978-1-4419-1674-7_36

European Lighthouse Integrated Project. (2016). Internet of things Architecture IoT-A. Retrieved November 1, 2016, from http://www.iot-a.eu/public/requirements/copy_of_requirements

Evans, D. (2011). *The Internet of Things: How the next evolution of the internet is changing everything*. Retrieved from http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

EY. (2016). *Internet of Things: Human machine interactions that unlock possibilities*. United Kingdom. Retrieved from http://www.ey.com/Publication/vwLUAssets/ey-m-e-internet-of-things/$FILE/ey-m-e-internet-of-things.pdf

Fernandes, J., Nati, M., Loumis, N. S., Nikoletseas, S., Raptis, T. P., Krco, S., … Ziegler, S. (2015). IoT Lab: Towards co-design and IoT solution testing using the crowd. In *Recent Advances in Internet of Things (RIoT), 2015 International Conference on* (pp. 1–6).

Ferreira, H. G. C., Canedo, E. D., & de Sousa, R. T. (2013). IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and arduino. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (pp. 53–60). http://doi.org/10.1109/WiMOB.2013.6673340

Ferreira, H. G., & Sousa Junior, R. T. (2017). Security Analysis of a Proposed Internet of Things Middleware. *Cluster Computing*, *20*(1), 651–660. http://doi.org/10.1007/s10586-017-0729-3

Formisano, C., Pavia, D., Gurgen, L., Yonezawa, T., Galache, J. A., Doguchi, K., & Matranga, I. (2015). The advantages of IoT and cloud applied to smart cities. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on* (pp. 325–332).

Fremantle, P. (2015). *A reference architecture for Internet of Things*. Sri Lanka. Retrieved from https://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/

Gartner Inc. (2014). IT Glossary. Retrieved January 4, 2017, from http://www.gartner.com/it-glossary/telematics/

Gartner Inc. (2016). *Hype Cycle for Emerging Technologies, 2016*.

Gartnet Inc. (2017). *Hype Cycle for Emerging Technologies, 2017*. USA.

Gilchrist, A. (2016). IIoT Reference Architecture. In *Industry 4.0* (pp. 65–86). Springer.

Gluhak, A., Hauswirth, M., Krco, S., Stojanovic, N., Bauer, M., Nielsen, R. H., … Corcho, O. (2011). An Architectural Blueprint for a Real-World Internet. In *Future Internet Assembly* (pp. 67–80).

Gluhak, A., Munoz, L., Sotres, P., Sanchez, L., Roux, P., Sanchez, B., … Hernandez, A. L. (2013). *Third Cycle Architecture Specification*.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. http://doi.org/10.1016/j.future.2013.01.010

Guo, B., Zhang, D., Wang, Z., Yu, Z., & Zhou, X. (2013). Opportunistic IoT: exploring the harmonious interaction between human and the internet of things. *Journal of Network and Computer Applications*, *36*(6), 1531–1539.

Hadim, S., & Mohamed, N. (2006). Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, *7*(3), 1.

Han, X., & Rashid, M. A. (2016). Gesture and voice control of Internet of Things. In *Industrial Electronics and Applications (ICIEA), 2016 IEEE 11th Conference on* (pp. 1791–1795).

Haridas, A. V., Marimuthu, R., & Sivakumar, V. G. (2018). A critical review and analysis on techniques of speech recognition: The road ahead. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, *22*(1), 39–57.

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2010). *Metodología de la investigación*. *McGraw-Hill* (Quinta Edi). México DF.

Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., & Boyle, D. (2014a). Architecture Reference Model. In *From Machine-To-Machine to the Internet of Things* (pp. 167–197). Elsevier. http://doi.org/10.1016/B978-0-12-407684-6.00007-3

Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., & Boyle, D. (2014b). IoT Architecture – State of the Art. In *From Machine-To-Machine to the Internet of Things* (pp. 145–165). Elsevier. http://doi.org/10.1016/B978-0-12-407684-6.00006-1

Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., & Boyle, D. (2014c). IoT Reference Architecture. In *From Machine-To-Machine to the Internet of Things* (pp. 199–223). Elsevier. http://doi.org/10.1016/B978-0-12-407684-6.00008-5

Hollosi, D., Nagy, G., Rodigast, R., Goetze, S., & Cousin, P. (2013). Enhancing wireless sensor networks with acoustic sensing technology: use cases, applications & experiments. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing* (pp. 335–342).

Huang, Z., Lin, K. J., & Shih, C. S. (2016). Supporting Edge Intelligence in Service-Oriented Smart IoT Applications. In *2016 IEEE International Conference on Computer and Information Technology (CIT)* (pp. 492–499). Nadi, Fiji: IEEE. http://doi.org/10.1109/CIT.2016.40

Huang, Z., Tsai, B. L., Chou, J. J., Chen, C. Y., Chen, C. H., Chuang, C. C., … Shih, C. S. (2015). Context and user behavior aware intelligent home control using WuKong middleware. In *2015 IEEE International Conference on Consumer Electronics - Taiwan* (pp. 302–303). Taipei, Taiwan: IEEE. http://doi.org/10.1109/ICCE-TW.2015.7216911

Hui, G. (2014). How the Internet of Things changes Business Models. Retrieved from https://hbr.org/2014/07/how-the-internet-of-things-changes-business-models

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology.

IEEE Computer Society. (2014). *Guide to the Software Engineering - Body of Knowledge.* (P. Bourque & R. E. Fairley, Eds.)*IEEE Computer Society* (V3 ed.). http://doi.org/10.1234/12345678

Igure, V. M., Laughter, S. A., & Williams, R. D. (2006). Security issues in SCADA networks. *Computers & Security*, *25*(7), 498–506.

International Organization for Standardization - ISO. Software product quality, 1 ISO/IEC 25010 34 (2011).

International Telecommunication Union - ITU. (2012). *Recommendation ITU-T Y.2060: Overview of the Internet of things*. *Series Y: Global information infrastructure, internet protocol aspects and next-generation networks - Frameworks and functional architecture models*. Retrieved from https://www.itu.int/rec/T-REC-Y.2060-201206-I

International Telecomunication Union - ITU. (2005). *The Internet of Things*. *ITU Internet Reports*.

Internet Society. (2015). *The Internet of Things (IoT): An Overview*. Geneva, Switzerland. Retrieved from https://www.internetsociety.org/doc/iot-overview

IoT-A Project. (2016). Requirements — IOT-A: Internet of Things Architecture.

IoT Analytics. (2016). *IoT Platforms: Market Report 2015-2021*. Hamburg, Germany. Retrieved from https://iot-analytics.com/product/iot-platforms-market-report-2015-2021-3/

ISO/IEC/IEEE. (2010). *ISO/IEC/IEEE 24765:2010 Systems and software engineering - Vocabulary*.

ISO/IEC JTC 1. (2009). *Study on Sensor Networks (Version 3)*.

ISO, & IEEE. Systems and software engineering - Vocabulary, ISO/IEC/IEEE 24765:2010(E) 1–418 (2010). http://doi.org/10.1109/IEEESTD.2010.5733835

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., … Hamida, A. Ben. (2011). Service-oriented middleware for the Future Internet: state of the art and research directions. *Journal of Internet Services and Applications*, *2*(1), 23–45. http://doi.org/10.1007/s13174-011-0021-3

Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *23*(1), 67–72.

Jelinek, F., Bahl, L., & Mercer, R. (1975). Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, *21*(3), 250–256.

Juang, B.-H., Hou, W., & Lee, C.-H. (1997). Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio Processing*, *5*(3), 257–265.

Juang, B.-H., & Rabiner, L. R. (2005). Automatic speech recognition-a brief history of the technology development. *Elsevier Encyclopedia of Language and Linguistics*, *1*, 24.

Kaneko, M., Arima, K., Usami, M., Sugimura, H., Isshiki, M., & Koh, K. (2015). Development of information living integrated by home appliances and web services. In *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on* (pp. 311–312).

Keh, H.-C., Shih, C.-C., Chou, K.-Y., Cheng, Y.-C., Ho, H.-K., Yu, P.-Y., & Huang, N.-C. (2014). Integrating unified communications and internet of m-health things with micro wireless physiological sensors, *17*(3), 319–328.

Khurana, T. (2017). IPv6 Enables Global Mobile IoT Innovation and Proliferation. Retrieved February 26, 2017, from https://goo.gl/B1E1eF

Kim, J., Lee, J., Kim, J., & Yun, J. (2014). M2M service platforms: survey, issues, and enabling technologies. *IEEE Communications Surveys & Tutorials*, *16*(1), 61–76.

Kostelnik, P., Sarnovsk, M., & Furdik, K. (2011). The semantic middleware for networked embedded systems applied in the internet of things and services domain. *Scalable Computing: Practice and Experience*, *12*(3), 307–316.

Krco, S., Pokric, B., & Carrez, F. (2014). Designing IoT architecture (s): A European perspective. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on* (pp. 79–84).

Kubitza, T. (2016). *Using Speech for End User Programming of Smart Environments in the Internet of Thing*. Germany.

Kubitza, T., & Schmidt, A. (2016). Rapid Interweaving of Smart Things with the meSchup IoT Platform. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct* (pp. 313–316). New York, NY, USA: ACM. http://doi.org/10.1145/2968219.2971379

Kubitza, T., & Schmidt, A. (2017). meSchup: A Platform for Programming Interconnected Smart Things. *Computer*, *50*(11), 38–49.

Kumar, A., Mishra, A., Makula, P., Karan, K., & Mittal, V. K. (2015). Smart Robotic Assistant. In *Region 10 Symposium (TENSYMP), 2015 IEEE* (pp. 25–28).

Lee, G. M., Crespi, N., Choi, J. K., & Boussard, M. (2013). Internet of things. In *Evolution of Telecommunication Services* (pp. 257–282). Springer.

Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, *58*(4), 431–440.

Lin, K. J., Reijers, N., Wang, Y. C., Shih, C. S., & Hsu, J. Y. (2013). Building Smart M2M Applications Using the WuKong Profile Framework. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* (pp. 1175–

1180). Beijing, China: IEEE. http://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.204

Loucopoulus, P., & Karakostas, V. (1995). *System Requirements Engineering*. McGraw-Hill, Inc.

Ma, M., Wang, P., & Chu, C.-H. (2013). Data management for internet of things: challenges, approaches and opportunities. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing* (pp. 1144–1151).

MacGillivray, C. (2016). *Worldwide Internet of Things Forecast Update, 2015-2019*.

Mamei, M., & Zambonelli, F. (2006). *Field-based coordination for pervasive multiagent systems*. Springer Science & Business Media.

Manrique, J. ., Rueda-Rueda, J., & Portocarrero, J. . (2016). Contrasting Internet of Things and Wireless Sensor Network from a conceptual overview. In *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (p. 6). IEEE Computer Society. http://doi.org/978-1-5090-5880-8/16

Marulli, F., Pareschi, R., & Baldacci, D. (2016). The internet of speaking things and its applications to Cultural Heritage. In *Proceedings of IoTBD2016 Conference, SCITEPRESS*.

McCulloch, W. S., & Pitts, W. (1990). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, *52*(1), 99–115.

Meier, R., & Cahill, V. (2002). Steam: Event-based middleware for wireless ad hoc networks. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on* (pp. 639–644).

Mineraud, J., Mazhelis, O., Su, X., & Tarkoma, S. (2016). A gap analysis of Internet-

of-Things platforms. *Computer Communications*, *89*, 5–16.

Miranda, J., Mäkitalo, N., Garcia-Alonso, J., Berrocal, J., Mikkonen, T., Canal, C., & Murillo, J. M. (2015). From the Internet of Things to the Internet of People. *IEEE Internet Computing*, *19*(2), 40–47.

Mittal, Y., Toshniwal, P., Sharma, S., Singhal, D., Gupta, R., & Mittal, V. K. (2015). A voice-controlled multi-functional Smart Home Automation System. In *India Conference (INDICON), 2015 Annual IEEE* (pp. 1–6).

Monteiro, C., Oliveira, M., Bastos, J., Ramrekha, T., & Rodriguez, J. (2014). Social Networks and Internet of Things, an Overview of the SITAC Project. In *International Wireless Internet Conference* (pp. 191–196).

Mottola, L., Murphy, A. L., & Picco, G. Pietro. (2006). Pervasive games in a mote-enabled virtual world using tuple space middleware. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (p. 29).

Nagata, K., Kato, Y., & Chiba, S. (1964). Spoken digit recognizer for Japanese language. In *Audio Engineering Society Convention 16*.

Nakagawa, E. Y., Oquendo, F., & Becker, M. (2012). Ramodel: A reference model for reference architectures. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on* (pp. 297–301).

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. Z. (2017). IoT middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, *4*(1), 1–20.

Nia, A. M., & Jha, N. K. (2016). A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*.

Nitti, M., Pilloni, V., Colistra, G., & Atzori, L. (2016). The virtual object as a major element of the internet of things: a survey. *IEEE Communications Surveys & Tutorials*, *18*(2), 1228–1240.

Nuance Communications. (2016). Majority of Consumers Want Intelligent, Personalized Dialogue with Customer Service. Retrieved February 27, 2017, from https://www.nuance.com/about-us/newsroom/press-releases/opus-intelligent-assistants-and-authentication-conference-2016.html

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, *40*(11), 38–45. http://doi.org/10.1109/MC.2007.400

Park, K.-J., Zheng, R., & Liu, X. (2012). Cyber-physical systems: Milestones and research challenges. *Computer Communications*, *36*(1), 1–7.

Patel, P., & Cassou, D. (2015). Enabling high-level application development for the Internet of Things. *Journal of Systems and Software*, *103*, 62–84.

Payne, G. (2014). The Internet of Things brings a new era of connectivity… and a talking fridge. Retrieved February 27, 2017, from http://whatsnext.nuance.com/connected-living/the-internet-of-things-connectivity/

Petrolo, R., Mitton, N., Soldatos, J., Hauswirth, M., & Schiele, G. (2014). Integrating wireless sensor networks within a city cloud. In *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking Workshops (SECON Workshops)* (pp. 24–27). http://doi.org/10.1109/SECONW.2014.6979700

Pressman, R. (2010). *Ingeniería del software: un enfoque práctico* (Séptima Ed). México DF: McGraw-Hill Interamericana.

Rabiner, L., Levinson, S., Rosenberg, A., & Wilpon, J. (1979). Speaker-independent recognition of isolated words using clustering techniques. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *27*(4), 336–349.

Rabiner, L. R., & Juang, B. H. (2004). *Statistical methods for the recognition and understanding of speech*. *Encyclopedia of language and linguistics*.

Ratkowski, A. (2016). Architecture for Internet of Things Analytical Ecosystem. In *Dependability Engineering and Complex Systems* (pp. 385–393). Springer.

Raveendran, V., Sanjeev, M. R., Paul, N., & Jijina, K. P. (2016). Speech only interface approach for personal computing environment. In *Engineering and Technology (ICETECH), 2016 IEEE International Conference on* (pp. 372–377).

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for internet of things: a survey. *IEEE Internet of Things Journal*, *3*(1), 70–95.

Richards, M. (2015). *Software architecture patterns*. O'Reilly Media, Incorporated.

Robles, T., Alcarria, R., de Andrés, D. M., Navarro, M., Calero, R., Iglesias, S., & López, M. (2015). An IoT based reference architecture for smart water management processes. *JoWUA*, *6*(1), 4–23.

Sakai, T., & Doshita, S. (1962). The Phonetic Typewriter. In *IFIP Congress* (Vol. 445, p. 449).

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., … others. (2014). SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks*, *61*, 217–238.

Sanchez, S., Angel Sicilia, M., & Rodriguez, D. (2012). *Ingeniería del Sofware. Un enfoque desde la guía SWEBOK*. Alfaomega.

Santos, J. F. M., Guessi, M., Galster, M., Feitosa, D., & Nakagawa, E. Y. (2013). A Checklist for Evaluation of Reference Architectures of Embedded Systems. In *SEKE* (Vol. 13, pp. 1–4).

Sarma, S., Brock, D., & Engels, D. (2001). Radio Frequency Identification and the Electronic Product Code. *IEEE Micro*, *21*(6), 50–54. http://doi.org/10.1109/40.977758

Schauer, P., & Debita, G. (2015). Internet of Things Service Systems Architecture.

Seo, S., Kim, J., Yun, S., Huh, J., & Maeng, S. (2015). HePA: Hexagonal Platform Architecture for Smart Home Things. In *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on* (pp. 181–189).

Shen, S., & Carugi, M. (2014). Standardizing the Internet of Things in an evolutionary way. In *ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?, Proceedings of the 2014* (pp. 249–254).

Shih, C. S., Lin, K. J., Chou, J. J., & Chuang, C. C. (2014). Autonomous Service Management for Location and Context Aware Service. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications* (pp. 246–251). Matsue, Japan: IEEE. http://doi.org/10.1109/SOCA.2014.10

Shin, D.-G., & Jun, M.-S. (2015). Home IoT device certification through speaker recognition. In *Advanced Communication Technology (ICACT), 2015 17th International Conference on* (pp. 600–603).

Shrouf, F., Ordieres, J., & Miragliotta, G. (2014). Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm. In *Industrial Engineering and Engineering Management (IEEM), 2014 IEEE International Conference on* (pp. 697–701).

Singh, S., & Singh, N. (2015). Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce. In *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on* (pp. 1577–1581).

Sinha, S., Agrawal, S. S., & Jain, A. (2013). Continuous density Hidden Markov Model for context dependent Hindi speech recognition. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on* (pp. 1953–1958).

Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.-P., Riahi, M., … Herzog, R. (2015). OpenIoT: Open Source Internet-of-Things in the Cloud. In I. Podnar Žarko, K. Pripužić, & M. Serrano (Eds.), *Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia,*

September 18, 2014, Invited Papers (pp. 13–25). Cham: Springer International Publishing. http://doi.org/10.1007/978-3-319-16546-2_3

Sommerville, I. (2011). *Ingeniería del Software*. PEARSON.

Souza, R., & Cardozo, E. (2016). A Resource-Oriented Architecture for the Internet of Things (IoT). In *Connectivity Frameworks for Smart Devices* (pp. 99–116). Springer.

Stravoskoufos, K., Sotiriadis, S., & Petrakis, E. (2016). IoT-A and FIWARE: bridging the barriers between the cloud and IoT systems design and implementation. In *Proc. 6th Int'l Conf. Cloud Computing and Services Science* (pp. 146–153).

Sundmaeker, H., Guillemin, P., Friess, P., & Woelfflé, S. (2010). *Vision and challenges for realising the Internet of Things*. (Cluster of European research projects on the Internet of Things, Ed.)*European Commision*.

Suzuki, J., & Nakata, K. (1961). Recognition of Japanese vowels - Preliminary to the recognition of speech. *Journal of the Radio Research Laboratory*, *8*(37), 193–212.

Talavera Portocarrero, J. M. (2016). *RAMSES: Reference Architectue of Self-Adaptative Middleware for Wireless Sensor Networks*. Universidade Federal fo Rio de Janeiro.

Techopedia. (2017). What is Modeling Language?

The Institute of Electrical and Electronics Engineers. (2014). *2014 IEEE Thesaurus*. Retrieved from http://www.ieee.org/documents/ieee_thesaurus_2013.pdf

Turck, M. (2018). Growing Pains: The 2018 Internet of Things Landscape. Retrieved April 2, 2018, from http://mattturck.com/iot2018/

United Nations Educational Scientific and Cultural Organization. (2016). UNESCO Thesaurus. Retrieved August 29, 2016, from http://vocabularies.unesco.org/

United Nations Educational Scientific and Cultural Organization (UNESCO). (2016). UNESCO Thesaurus. Retrieved April 11, 2016, from http://vocabularies.unesco.org/browser/thesaurus/en/

Unnibhavi, A. H., & Jangamshetti, D. S. (2016). A survey of speech recognition on south Indian Languages. In *Signal Processing, Communication, Power and Embedded System (SCOPES), 2016 International Conference on* (pp. 1122–1126).

Usländer, T., & Epple, U. (2015). Reference model of industrie 4.0 service architectures. *At-Automatisierungstechnik*, *63*(10), 858–866.

Verdouw, C. N., Robbemond, R. M., Verwaart, T., Wolfert, J., & Beulens, A. J. M. (2015). A reference architecture for IoT-based logistic information systems in agri-food supply chains. *Enterprise Information Systems*, 1–25.

Wang, M.-M., Cao, J.-N., Li, J., & Dasi, S. K. (2008). Middleware for wireless sensor networks: A survey. *Journal of Computer Science and Technology*, *23*(3), 305–326.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 94–104.

Weyrich, M., & Ebert, C. (2016). Reference architectures for the internet of things. *IEEE Software*, *33*(1), 112–116.

Whittaker, E. W. D. (2000). *Statistical language modelling for automatic speech recognition of Russian and English*. University of Cambridge.

Wiener, N. (1961). *Cybernetics or Control and Communication in the Animal and the Machine* (Vol. 25). MIT press.

Wortmann, F., Flüchter, K., & others. (2015). Internet of things. *Business & Information Systems Engineering*, *57*(3), 221–224. http://doi.org/10.1007/s12599-015-0383-3

Xu, B., Zhang, D., & Yang, W. (2012). Research on architecture of the Internet of Things for grain monitoring in storage. In *Internet of Things* (pp. 431–438). Springer.

Zhong, N., Ma, J., Huang, R., Liu, J., Yao, Y., Zhang, Y., & Chen, J. (2016). Research challenges and perspectives on Wisdom Web of Things (W2T). In *Wisdom Web of Things* (pp. 3–26). Springer.

Zhou, S., Liu, G., & Lin, C. (2012). An Embedded Voice Inquiry Experimental Platform for Temperature and Humidity Measurement on the Internet of Things. In *Emerging Computation and Information teChnologies for Education* (pp. 533–539). Springer.